VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MATEMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

STOCHASTIC PROGRAMMING ALGORITHMS

ALGORITMY STOCHASTICKÉHO PROGRAMOVÁNÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUBOMÍR KLIMEŠ

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. PAVEL POPELA, Ph.D.

BRNO 2010

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav matematiky
Akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Lubomír Klimeš

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Matematické inženýrství (3901T021)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Algoritmy stochastického programování

v anglickém jazyce:

Stochastic Programming Algorithms

Stručná charakteristika problematiky úkolu:

Student si prohloubí znalosti problematiky algoritmů stochastického programování se zaměřením na problematiku efektivní modifikace a implementace dekompozičních algoritmů. Pro vybranou třídu dekompozičních algoritmů nalezne vhodné úpravy a heuristická doplnění. Vytvoří knihovnu programových nástrojů použitelnou pro aplikace v inženýrském návrhu.

Cíle diplomové práce:

Vybraná třída algoritmů bude implementována a testována s cílem přispět k řešení rozsáhlých inženýrských úloh.

Seznam odborné literatury:

Kall, P., Wallace, S.W. Stochastic Programming, Wiley and Sons, 1993.
Birge J.R., Louveaux, F. Introduction to Stochastic Programming, Springer 1996

Vedoucí diplomové práce: RNDr. Pavel Popela, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2009/2010.

V Brně, dne 20.11.2009

L.S.

prof. RNDr. Josef Šlapal, CSc. Ředitel ústavu	prof. RNDr. Miroslav Doupovec, CSc. Děkan fakulty

Summary

Stochastic programming and optimization are powerful tools for solving a wide variety of engineering problems including uncertainty.

The progressive hedging algorithm is an effective decomposition method for solving scenario-based stochastic programmes. Due to the vertical decomposition, this algorithm can be implemented in parallel thereby the computing time and other resources could be considerably spared.

The theoretical part of this master's thesis deals with mathematical and especially with stochastic programming. Further, the progressive hedging algorithm is presented and discussed in detail.

In the practical part, the original parallel implementation of the progressive hedging algorithm is suggested, fruitfully discussed and tested to simple problems. Furthermore, the presented parallel implementation is used for solving the continuous casting process of steel slabs and the results are appraised.

Keywords

optimization, stochastic programming, scenario-based programmes, progressive hedging algorithm, parallelization, continuous casting process

Abstrakt

Stochastické programování a optimalizace jsou mocnými nástroji pro řešení široké škály inženýrských problémů zahrnujících neurčitost.

Algoritmus progressive hedging je efektivní dekompoziční metoda určená pro řešení scénářových stochastických úloh. Z důvodu vertikální dekompozice je možno tento algoritmus implementovat paralelně, čímž lze významně ušetřit výpočetní čas a ostatní prostředky.

Teoretická část této diplomové práce se zabývá matematickým a zejména pak stochastickým programováním a detailně popisuje algoritmus progressive hedging.

V praktické části je navržena a diskutována původní paralelní implementace algoritmu progressive hedging, která je pak otestována na jednoduchých úlohách. Dále je uvedená paralelní implementace použita pro řešení inženýrského problému plynulého odlévání ocelové bramy a na závěr jsou získané výsledky zhodnoceny.

Klíčová slova

optimalizace, stochastické programovaní, scénářové úlohy, progressive hedging algoritmus, paralelizace, plynulé odlévání oceli

KLIMEŠ, L. *Stochastic programming algorithms.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010. 95 s. Vedoucí diplomové práce RNDr. Pavel Popela, Ph.D.

Affirmation

I declare that the master's thesis is the result of my own work and all used sources are duly listed in the bibliography.

Bc. Lubomír Klimeš

Acknowledgement

I would like to thank RNDr. Pavel Popela, Ph.D. for supervising my master's thesis, for all his support, advice, valuable comments and suggestions.

I would also like to express my gratitude to Ing. Tomáš Mauder for his advice and fruitful discussions.

My special thanks belong to my parents for their support and love and to Tereza for her love, forbearance and support.

Bc. Lubomír Klimeš

Contents

CHAPTER 1

Aims of the Master's Thesis

The main aims of this master's thesis are following:

1. To formulate basic ideas and principles of stochastic programming in which the uncertainty is modelled by random variables. Further, to describe the main approaches *wait-and-see* and *here-and-now* of stochastic programming and to list common deterministic equivalents to the stochastic models. Furthermore, to deal more precisely with two-stage stochastic programming and with the scenario-based approach that will play the crucial role in this thesis. To give the basic ideas of multi-stage stochastic programming and to define basic qualitative characteristics to compare the results acquired by various approaches of stochastic programming to each other.

2. To describe in detail the progressive hedging algorithm – a method for solving scenario-based stochastic problems. To present the theory of this algorithm based on the scenario aggregation principle. To state that algorithm for one-stage stochastic problems and further, to generalize it for multi-stage case.

3. Due to the fact that the progressive hedging algorithm in each iteration requires to solve scenario-independent subproblems, to deal with the parallelization of solving those subproblems on hardware with several processors. The using of parallelism will save the time needed for computing since in the same time several subproblems will be solved simultaneously. To deal with the parallelization via Message Passing Interface, MPI, which is the API[1] library allowing the parallel computing. Further, to design and program a software in C++ for solving scenario-based stochastic programmes by using GAMS optimization software and the progressive hedging algorithm. Furthermore, to discuss in detail the parallel implementation of the two-stage progressive hedging algorithm that can be very often used in practical applications. To show the behaviour of the progressive hedging algorithm in simple illustrative problems that could help the reader to understand the fundamental principles of the progressive hedging algorithm and also of stochastic programming. The main outcome will be universal implementations of the one-stage and especially

[1] Application Programming Interface

of the two-stage progressive hedging algorithm that can be easily applied to various optimization problems.

4. To apply all foregoing knowledges and to test the parallel implementation of the progressive hedging algorithm for the technical problem of the continuous casting process of steel slabs. This problem will be scenario-based and will embrace the uncertainty via possible breakdown in the continuous casting machine. To derive in detail the model from the second-order partial differential equation and its two-stage scenario-based modification. Finally, to present the results obtained by using the parallel implementation of the progressive hedging algorithm and to discuss the suitability of the stochastic programming approach.

CHAPTER 2

Introduction

The mathematical programming and optimization is an interdisciplinary branch of science in which mathematics plays a significant role. The aim of optimization is to establish an appropriate model of a particular problem and find its optimal solution (or optimal solutions) that satisfies constraints and conditions of that model and minimizes or maximizes its objective function.

People are dealing with decision problems and in certain sence with optimization from ancient times. The rapid enlargement of computers in 1950s brings the intensive theoretical research in this branch. Recently, due to the rich market competition, there is an intention to design, manufacture and distribute goods with minimal loads and maximal profit, and therefore, the optimization and methods of mathematical programming are powerful tools to realize this idea.

In case all parameters and data of a given problem are fully known, we talk about deterministic programming (see [2, 12]). Unfortunately, in many engineering problems those parameters and data are often not fully known and such problems embrace the randomness which brings the uncertainty to those models. The stochastic programming deals with problems under uncertainty in which the randomness is modelled by random variables for which the probability distribution is known.

In this thesis, we will deal with stochastic programming and optimization. The thesis can be divided into two parts – theoretical and practical.

In the theoretical part, in particular in Chapter 3 and 4, we will describe the basis of stochastic programming – the stochastic programme and its deterministic equivalents that correctly interpret the randomness, here-and-now and wait-and-see decision approaches and we will introduce the concept of scenarios and scenario-based programmes. Further, we will especially deal with two-stage stochastic programming in which the decision maker takes two decisions – the first decision is taken when no outcome of random parameters is known and the second one is taken when a particular realization of random parameters has been observed and is already known. We will also wish to compare results obtained by using various approaches of stochastic programming and therefore, the useful characteristics will be presented for this purpose. Further, we will present the progressive hedging algorithm – a decomposition algorithm based on the aggregation principle for solving

scenario-based stochastic programmes that was introduced by American mathematicians Rockafellar and Wets in 1980s (see [19, 23]).

Chapters 5 and 6 deal with the practical part of this thesis. Chapter 5 is concerned with a parallel implementation of the progressive hedgign algorithm whose theoretical principles are presented in Chapter 4. The parallelism is provided by the feature of the progressive hedging algorithm that decomposes an original problem into independent subproblems for each particular scenario. Hence, in case that the multi-processor hardware is available, these independent subproblems can be solved simultaneously. This arrangement will save the computing time since n subproblems can be solved simultaneously on n parallel processors in roughly same time as one subproblem on one-processor machine. In this chapter, the one-stage and two-stage parallel implementations of the progressive hedging algorithm, programmed in C++ programming language with using GAMS optimization software as a solver for subproblems and Message Passing Interface, MPI, an API for parallel computing, will be greatly discussed and tested for simple problems.

In Chapter 6, we will test the parallel implementation from the foregoing Chapter 5 for the particular engineering problem – to the continuous casting process of steels slabs that will embrace the uncertainty via possible breakdown in a cooling part of continuous casting machine. We will assemble the programme from the second-order partial differential equation describing the process of continuous casting and its two-stage scenario-based modification. Using that model, we will test the parallel implementation of the progressive hedging algorithm described in the foregoing chapters. In conclusion, we will also determine qualitative characteristics of acquired solution and will discuss the suitability of using the stochastic programming approach.

In Appendix A, the reduced gradient (RG) and the generalized reduced gradient (RGR) methods are in detailed discussed. In Appendix B, the optimality conditions for unconstrained and also for constrained problems are presented. In Appendix C, the topics of the penalty functions and the augmented Lagrangian function are discussed.

The research leading to the results of Chapter 5 was supported by project from MSMT of the Czech Republic no. 1M06047 and the obtained results of this chapter will be applied by a grant from the Grant Agency of the Czech Republic reg. no. 103/08/1658 and by a research plan from MSMT of the Czech Republic no. MSM0021630519.

The research leading to the results of Chapter 6 was supported by FME BUT project BTU BD13002 “Matematické modelování a optimalizace v průmyslových aplikacích” and the results contained in Chapter 6 will be applied by a grant GAČR106/08/0606 “Modelování přenosu tepla a hmoty při tuhnutí rozměrných systémů hmotných kovových materiálů” and GAČR106/09/0940 “Numerický a stochastický model plynule odlévaných ocelových předlitků obdélníkového profilu”.

CHAPTER 3

Optimization

In this chapter, we will introduce the concept of mathematical programming, deterministic and especially stochastic optimization. Further, we will present several qualitative characteristics useful for the comparison of results acquired by different approaches. At the end of this chapter, we will deal with two-stage and multi-stage stochastic programming.

3.1 Mathematical Programming

The mathematical programming is an interdisciplinary branch of science dealing with problems to optimize a value of *objective function* with respect to given *constraints*. To *optimize* means that we are looking for a particular *optimal solution* that *minimizes* or *maximizes* an objective function and satisfies given constraints. The using of minimization or maximization of an objective function depends on a character of a particular problem to be solved. Note that the maximization problem can be easily transformed to the equivalent minimization problem and conversely, the minimization problem can be transformed to the equivalent maximization problem (see e.g. [1]).

Nevertheless, the optimal solution usually cannot be chosen arbitrarily. The constraints mentioned above specify the *feasible set* as a subset of finite-dimensional space and the optimal solution has to be searched only in this feasible set. The constraints can be divided into two classes – the first class contains inequality constraints and the second one is formed by equality constraints. Remark that the term mathematical programme or just *programme* is often used instead of mathematical programming problem.

Now, we can formulate the general mathematical programming problem: Find a solution $\boldsymbol{x}_{\min}$ that

$$\begin{array}{lll} \text{minimizes} & f(\boldsymbol{x}) & \\ \text{subject to} & g_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m, \\ & h_j(\boldsymbol{x}) = 0, & j = 1, \ldots, n, \\ & \boldsymbol{x} \in X, & \end{array} \tag{3.1}$$

where X is a subspace of N-dimensional space, $X \subset \mathbb{R}^N$, f, g_i for all $i = 1, \ldots, m$ and h_j for all $j = 1, \ldots, n$ are functions $\mathbb{R}^N \to \mathbb{R}$. Our goal is to find at least one optimal solution $\boldsymbol{x}_{\min}$ to 3.1. The feasible set C is determined as

$$C = \{\boldsymbol{x} \in X \,|\, g_i(\boldsymbol{x}) \leq 0 \text{ for all } i = 1, \ldots, m;\ h_j(\boldsymbol{x}) = 0 \text{ for all } j = 1, \ldots, n\}\,.$$

Note that we usually will omit the sentence "Find a solution $\boldsymbol{x}_{\min}$ that…". We can also use more compact vector notation and rewrite (3.1) to the following form:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}, \\ & && \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \\ & && \boldsymbol{x} \in X, \end{aligned} \tag{3.2}$$

where $\boldsymbol{g}$ and $\boldsymbol{h}$ are vector functions, $\boldsymbol{g}\colon \mathbb{R}^N \to \mathbb{R}^m$, $\boldsymbol{h}\colon \mathbb{R}^N \to \mathbb{R}^n$. Note that also the following form of mathematical programme is sometimes used instead of (3.1) or (3.2):

$$? \in \operatorname*{argmin}_{\boldsymbol{x}} \Big\{ f(\boldsymbol{x}) \,|\, \boldsymbol{x} \in C = \{\boldsymbol{x} \in X \,|\, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0},\ \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}\} \Big\}, \tag{3.3}$$

where "argmin" denotes the set of all optimal solutions (see e.g. [17]).

We have to note here that the so-called *optimality conditions* play the significant role in mathematical programming. This topic is discussed in Appendix B.

3.2 Deterministic Programming

We already know what the mathematical programming is and what is its main goal. In this section, we will describe the class of deterministic programmes which is a subset of all mathematical programmes. *Deterministic programme* is a mathematical programme for which all data are deterministic and fully known. By the term "data" we mean all parameters and coefficients of the problem to be solved. In other words, there is no uncertainty in a deterministic model. This is the most significant difference in the comparison with stochastic programmes.

The deterministic programme has the form:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}, \boldsymbol{a}) \\ &\text{subject to} && \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{a}) \leq \boldsymbol{0}, \\ & && \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{a}) = \boldsymbol{0}, \\ & && \boldsymbol{x} \in X, \end{aligned} \tag{3.4}$$

where $\boldsymbol{a} \in \mathbb{R}^K$ is a K-dimensional constant vector. We have used the vector $\boldsymbol{a}$ in (3.4) to emphasize that all parameters are constant and fully known (cf. (3.6)). The special case of (3.4) is a programme having all components (objective function and constraints) linear and X being the convex polyhedral set. Such a programme is called a *linear* deterministic programme. Its form is:

$$\begin{aligned} &\text{minimize} && \boldsymbol{c}^T\boldsymbol{x} \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ & && \boldsymbol{x} \in X, \end{aligned} \tag{3.5}$$

where $\boldsymbol{c}$ is N-dimensional vector, $\boldsymbol{A}$ is $(m+n) \times N$ matrix and $\boldsymbol{b}$ is $(m+n)$-dimensional vector. In the so-called standard form of a linear programme, the requirement of non-negativity $\boldsymbol{x} \geq \boldsymbol{0}$ of variable $\boldsymbol{x}$ is added to (3.5). Note that there exist many important linear programmes having special structures that lead to special algorithms. An example of special structure can be the staircase structure: coefficients a_{ij} of matrix $\boldsymbol{A} = (a_{ij})_{i=1,\dots,m+n;\, j=1,\dots,N}$ satisfy the condition $a_{ij} = 0$ for all $j > i$.

3.3 Stochastic Programming

In the foregoing section, we discussed the concept of deterministic programming and deterministic models. In practical problems and real applications, the using of deterministic models is bounded since the real models include parameters that are not fully known. In other words, the real problems include a certain level of *uncertainty* and the using of deterministic models can lead to distorted or even completely incorrect results (see [2]).

In general, different approaches for modelling problems including uncertainty may be used. One approach of them is *stochastic programming* in which the uncertain parameters are modelled by *random variables* (see [2, 17]).

Definition 3.1. Let the triplet $(\Omega, \mathscr{A}, P)$ be a probability space. The mapping $\xi\colon \Omega \to \mathbb{R}$ is called a *random variable* if for all $x \in \mathbb{R}$ holds

$$\{\omega\colon \xi(\omega) \leq x\} \in \mathscr{A}.$$

The general form of stochastic programme is

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}, \boldsymbol{\xi}) \\ \text{subject to} & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}) \leq \boldsymbol{0}, \\ & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{0}, \\ & \boldsymbol{x} \in X, \end{array} \tag{3.6}$$

where $\boldsymbol{\xi} = (\xi_1, \dots, \xi_K)^T$, $\boldsymbol{\xi}(\omega)\colon \Omega \to \mathbb{R}^K$, is a finite-dimensional random vector formed by random variables on the probability space $(\Omega, \mathscr{A}, P)$. The mapping $\boldsymbol{\xi}\colon \Omega \to \mathbb{R}^K$ induces a probability measure $\mathcal{P}$ on the space $\mathbb{R}^K$ with the Borel algebra $\mathscr{B}$ as underlying σ-algebra (see [30, 8, 9]). Let us denote the corresponing probability space by $(\Xi, \mathscr{B}, \mathcal{P})$. Thus, f, $\boldsymbol{g}$ and $\boldsymbol{h}$ in (3.6) are functions $f\colon \mathbb{R}^N \times \Xi \to \mathbb{R}$, $\boldsymbol{g}\colon \mathbb{R}^N \times \Xi \to \mathbb{R}^m$ and $\boldsymbol{h}\colon \mathbb{R}^N \times \Xi \to \mathbb{R}^n$.

The feasible set $C(\boldsymbol{\xi})$ of (3.6) can be written in the form

$$C(\boldsymbol{\xi}) = \{\boldsymbol{x} \in X \,|\, \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}) \leq \boldsymbol{0};\, \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{0}\}.$$

Observe that the model (3.6) is in fact the deterministic model (3.2) in which some constant parameters have been replaced by random parameters. The programme (3.6) is called an *underlying mathematical programme*.

The concept of random variable on the probability space included in the model allows to deal with the probability distribution instead of constant parameters in case of deterministic programming. We will assume that the probability distribution of $\boldsymbol{\xi}$ is completely known.

Figures 3.1 and 3.2 illustrate original two-dimensional feasible set including random parameters. The feasible set $C(\boldsymbol{\xi})$ is determined by the quadratic inequality constraint

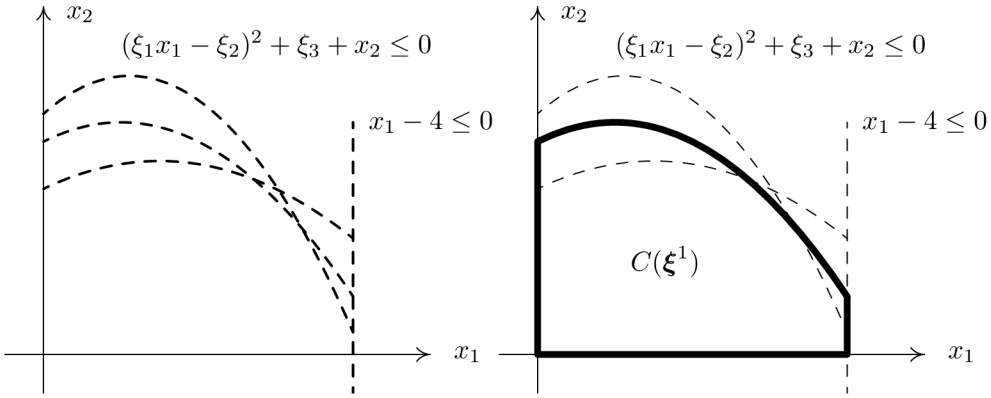

Figure 3.1: An example: the feasible set $C(\boldsymbol{\xi})$ including random influences

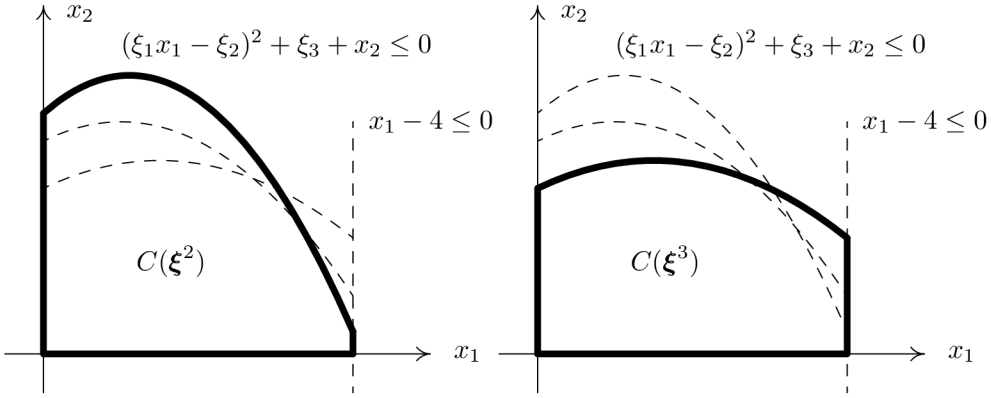

Figure 3.2: An example: the feasible set $C(\boldsymbol{\xi})$ including random influences

$(\xi_1 x_1 - \xi_2)^2 + \xi_3 + x_2 \leq 0$, by the linear inequality constraint $x_1 - 4 \leq 0$ and by the requirement of non-negativity of variable $\boldsymbol{x}$, $x_1 \geq 0$, $x_2 \geq 0$. The figures show the feasible set for three different realization of random parameters: $\boldsymbol{\xi}^1 = (\xi_1^1, \xi_2^1, \xi_3^1)^T = \left(\frac{1}{2}, \frac{1}{2}, -3\right)^T$, $\boldsymbol{\xi}^2 = (\xi_1^2, \xi_2^2, \xi_3^2)^T = \left(\frac{2}{3}, \frac{2}{3}, -\frac{7}{2}\right)^T$ and $\boldsymbol{\xi}^3 = (\xi_1^3, \xi_2^3, \xi_3^3)^T = \left(\frac{1}{5}, \frac{2}{3}, -\frac{5}{2}\right)^T$.

But what is the meaning of programme (3.6)? When a particular realization of random parameters $\boldsymbol{\xi}^p$ is observed and becomes known, one can replace $\boldsymbol{\xi}$ in (3.6) by particular $\boldsymbol{\xi}^p$ and the meaning of the model is clear. But what is the meaning of given programme before the realization of $\boldsymbol{\xi}$ is observed?

Our aim is to find a solution to the programme (3.6). There are two basic approaches. One of them is to wait for the particular realization $\boldsymbol{\xi}^p$ of $\boldsymbol{\xi}$ and then solve the formed deterministic programme. This technique is called the *wait-and-see* approach. On the other hand, the decision maker usually cannot wait to observe the realization of $\boldsymbol{\xi}$. He has to take the decision before the realization of $\boldsymbol{\xi}$ becomes known and he wants to find a solution that will be in some sence "optimal" for each realization of $\boldsymbol{\xi}$. This approach is called *here-and-now*. In this case, we will need a tool that allows us to properly deal with random parameters in (3.6).

As we already mentioned, the model (3.6) is not well defined and this fact lead to the concept of *deterministic equivalents* that correctly interpret random parameters in (3.6).

Deterministic Equivalents

In this section, we will deal with the programme (3.6). Our goal is to find its deterministic equivalent – the model that correctly interprets random parameters included in (3.6). There are several approaches how the deterministic equivalents of the objective function and of the feasible set can be defined. Hence, we may consider two classes: deterministic equivalents of the objective function and deterministic equivalents of the feasible set. By the combination of elements of those two classes, the deterministic equivalents of (3.6) can be assembled. We list below several typical deterministic equivalents of programme (3.6) (see e.g. [17, 30]).

Wait-and-see Programme

The *wait-and-see* deterministic equivalent of programme (3.6), WS programme, is defined by

$$z^{\mathrm{WS}} = \mathbb{E}_{\boldsymbol{\xi}} \left(\min_{\boldsymbol{x}(\boldsymbol{\xi})} \left\{ f\Big(\boldsymbol{x}(\boldsymbol{\xi}), \boldsymbol{\xi}\Big) \mid \boldsymbol{x}(\boldsymbol{\xi}) \in C(\boldsymbol{\xi}) \text{ for all } \boldsymbol{\xi} \in \Xi \right\} \right) \tag{3.7}$$

and its optimal solution is denoted by $\boldsymbol{x}^{\mathrm{WS}}$. Unfortunately, above *wait-and-see* approach usually cannot be used due to the lack of information about the future. Therefore, *here-and-now* approach and its following *here-and-now* deterministic equivalents are commonly used instead of (3.7).

Individual Scenario Programme

This programme is based on the idea that the random parameters in the underlying programme (3.6) are replaced by a typical realization $\boldsymbol{\xi}^s$. Such a particular realization $\boldsymbol{\xi}^s$ is called a *scenario*. Hence, the individual scenario programme (IS programme) has the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}, \boldsymbol{\xi}^s) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}^s) \leq \boldsymbol{0}, \\ & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{\xi}^s) = \boldsymbol{0}, \\ & \boldsymbol{x} \in X. \end{aligned} \tag{3.8}$$

Its optimal solution is denoted $\boldsymbol{x}^{\mathrm{IS}}$. This IS programme is useful in case the expert is able to determine a typical realization of $\boldsymbol{\xi}$.

"Fat solution" Programme

The idea of "fat solution" programme is similar to the individual scenario approach. But instead of a typical representative $\boldsymbol{\xi}^s$ is considered the worst realization of $\boldsymbol{\xi}$ that can occur. Thus, we expect that the decision is then robust to all possible realizations of $\boldsymbol{\xi}$. The form of this so-called MM programme is

$$\begin{aligned} \text{minimize} \quad & \sup_{\boldsymbol{\xi} \in \Xi} f(\boldsymbol{x}, \boldsymbol{\xi}) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}) \leq \boldsymbol{0} \text{ almost surely}, \\ & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{\xi}) = \boldsymbol{0} \text{ almost surely}, \\ & \boldsymbol{x} \in X. \end{aligned} \tag{3.9}$$

Its solution is denoted $\boldsymbol{x}^{\mathrm{MM}}$. *Almost surely* means that given constraints are satisfied for all $\boldsymbol{\xi} \in \Xi$ except of a set $\{\boldsymbol{\xi} : \boldsymbol{\xi} \in \Xi\}$ whose measure is zero.

Observe that the evaluation of objective function (the calculation of supremum) is from the computational point of view very expensive and leads to big costs. Therefore, this schema is usually not appropriate to use.

Expected Value Programme

The idea of EV programme is to utilize a characteristic of random variable. In this case, the expected value $\mathbb{E}$ of random vector $\boldsymbol{\xi}$ is used to remove the uncertainty of (3.6). The expected value programme (EV programme) has the form

$$\begin{aligned} \text{minimize} \quad & f\Big(\boldsymbol{x}, \mathbb{E}(\boldsymbol{\xi})\Big) \\ \text{subject to} \quad & \boldsymbol{g}\Big(\boldsymbol{x}, \mathbb{E}(\boldsymbol{\xi})\Big) \leq \mathbf{0}, \\ & \boldsymbol{h}\Big(\boldsymbol{x}, \mathbb{E}(\boldsymbol{\xi})\Big) = \mathbf{0}, \\ & \boldsymbol{x} \in X. \end{aligned} \tag{3.10}$$

Its optimal solution is denoted $\boldsymbol{x}^{\mathrm{EV}}$. Note that for the calculation of expected values, the requirement of complete available information about random vector $\boldsymbol{\xi}$ is crucial.

Expected Objective Programme

The EO programme is based on the expected value incorporated in the objective function. Its form reads

$$\begin{aligned} \text{minimize} \quad & \mathbb{E}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x}, \boldsymbol{\xi})\Big) \\ \text{subject to} \quad & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\xi}) \leq \mathbf{0} \text{ almost surely}, \\ & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{\xi}) = \mathbf{0} \text{ almost surely}, \\ & \boldsymbol{x} \in X. \end{aligned} \tag{3.11}$$

The solution of EO programme is denoted $\boldsymbol{x}^{\mathrm{EO}}$. In case that the variance of objective function has to be taken into account, the objective of (3.11) can be replaced, for instance, by

$$\mathbb{E}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x}, \boldsymbol{\xi})\Big) + \lambda \sqrt{\mathrm{var}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x}, \boldsymbol{\xi})\Big)},$$

where $\lambda > 0$ is a parameter. Remark that the reason for the square root is the unit compatibility.

In what follows, we will define some helpful qualitative characteristic that will be useful for the comparison of above approaches to each other (see [2]).

Definition 3.2. Consider the EV programme (3.10) and let $\boldsymbol{x}^{\mathrm{EV}}$ be its optimal solution. Then, the quantity

$$\mathrm{EEV} = \mathbb{E}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x}^{\mathrm{EV}}, \boldsymbol{\xi})\Big) \tag{3.12}$$

is called the *expected result of using EV solution*, EEV.

The above definition of the expected result of using EV solution leads to the useful characteristic VSS.

Definition 3.3. Let EEV be the expected result of using EV solution (3.12) and $\boldsymbol{x}^{\mathrm{EO}}$ be the optimal solution to the EO programme (3.11). Then the quantity

$$\mathrm{VSS} = \mathrm{EEV} - \mathbb{E}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x}^{\mathrm{EO}}, \boldsymbol{\xi})\Big) \tag{3.13}$$

is called the *value of stochastic solution*, VSS.

Remark that the value of $\mathbb{E}_{\boldsymbol{\xi}}(f(\boldsymbol{x}^{\mathrm{EO}}, \boldsymbol{\xi}))$ is actually the value of objective function of programme (3.11) for $\boldsymbol{x} = \boldsymbol{x}^{\mathrm{EO}}$, i.e., for its optimal solution. This value is usually denoted z^{EO}, i.e.,

$$z^{\mathrm{EO}} = \mathbb{E}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x}^{\mathrm{EO}}, \boldsymbol{\xi})\Big). \tag{3.14}$$

Hence, $\mathrm{VSS} = \mathrm{EEV} - z^{\mathrm{EO}}$.

What is the meaning of VSS? The value of stochastic solution is a useful characteristic since it expresses how suitable is to use the EV approach instead of EO approach: the using of EV approach and EV solution is suitable for small values of VSS. On the other hand, the VSS gives us the information "how many" could be gained by solving EO programme instead of simpler EV programme. Unfortunately, the weakness is that the computation of VSS requires both EO solution and EVV.

The second useful characteristic, the expected value of perfect information, is introduced below.

Definition 3.4. Let z^{EO} be the value of objective function of EO programme for its optimal solution $\boldsymbol{x}^{\mathrm{EO}}$ and let z^{WS} be defined by (3.7). Then, the *expected value of perfect information*, EVPI, is defined as follows,

$$\mathrm{EVPI} = z^{\mathrm{EO}} - z^{\mathrm{WS}}. \tag{3.15}$$

The expected value of perfect information represents "how many" should be gained when the perfect information about the future is available. In other words, this value represents how many the decision maker should be ready to pay for the perfect and accurate information about future.

Nonetheless, since there is usually no additional information about the future that could be "bought" in technical application, the value of stochastic solution, VSS, is a more relevant characteristic of using stochastic programming than the expected value of perfect information, EPVI.

The following theorem states the relationship between values of z^{WS}, z^{EO} and EEV.

Theorem 3.1. *Let z^{WS}, z^{EO} and EEV be defined by* (3.7), (3.14) *and* (3.12), *respectively. Then, it holds the following inequality,*

$$z^{WS} \leq z^{EO} \leq \mathrm{EEV}.$$

Proof. See [2, page 140]. □

3.4 Two-Stage Stochastic Programming

In the foregoing section, we discussed problems of stochastic programming in which the decision maker had to take the only one decision. In this section, we will deal with

programmes that are "discretized" in time and in which the decision maker will take two decisions in two different moments in time.

From the decision maker point of view, we can divide decisions to be taken into two types according to the available information (see [2]):

1. the decision that the decision maker takes at the beginning of the experiment. In this moment, there is no available information about the future particular realization of random parameters. In other words, this decision has to be taken before the realization of random parameters is observed. Such a decision is called a *first-stage decision* and the period (the beginning of the experiment) when this decision is taken is called the *first stage*.

2. the decision that the decision maker takes after the realization of random parameters is observed and becomes known. Such a decision is called a *second-stage decision* and the period when this decision is taken is called the *second stage*.

We denote the first-stage decision by $\boldsymbol{x}$ and the second-stage decision by $\boldsymbol{y}(\boldsymbol{\xi})$. Observe that $\boldsymbol{y}(\boldsymbol{\xi})$ is actually vector function of $\boldsymbol{\xi}$. Note that the second-stage decision $\boldsymbol{y}$ may depend also on the first-stage decision, but this dependence is not obvious explicitly in opposite to the dependence of $\boldsymbol{y}$ on the realization of $\boldsymbol{\xi}$.

The moments when the decision maker takes the decision are called *stages*. Note that there is a difference between stages and time periods. In general, stages do not correspond to time periods and a stage can include several time periods. Hence, the stage decision can consist of several time period decisions (see [17]).

3.4.1 Two-Stage Stochastic Programmes with Recourse

The significant programme in stochastic programming is the *two-stage stochastic programe with recourse* (see [2, 12]). The reasons and the main idea are following: in the first-stage, the decision maker takes the decision $\boldsymbol{x}$ without any information about the realization of $\boldsymbol{\xi}$. More specifically, in the first stage there is no realization of $\boldsymbol{\xi}$ available. Then, the programme continues to the second stage. In the second stage, the particular realization of $\boldsymbol{\xi}$ is observed and becomes known for the decision maker. The desicion maker takes the second-stage decision $\boldsymbol{y}$. The purpose of the second-stage decision $\boldsymbol{y}$ is to *correct* prospective *mistakes* or *unfeasibility* that may be caused by the fist-stage decision $\boldsymbol{x}$. This type of the second-stage decision is called the *recourse*. Note that the second stage is actually the deterministic programme since the random parameters already take particular values.

We will introduce the linear case of the two-stage stochastic programme with fixed recourse and further, we will generalize it to the non-linear case.

The two-stage linear programme with fixed recourse reads: Find $\boldsymbol{x}$ and $\boldsymbol{y}$ such that

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T\boldsymbol{x} + \mathbb{E}_{\boldsymbol{\xi}}\Big(\min_{\boldsymbol{y}} \boldsymbol{q}(\boldsymbol{\xi})^T\boldsymbol{y}(\boldsymbol{\xi})\Big) & \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, & \\ & \boldsymbol{T}(\boldsymbol{\xi})\boldsymbol{x} + \boldsymbol{W}\boldsymbol{y}(\boldsymbol{\xi}) = \boldsymbol{h}(\boldsymbol{\xi}) \qquad & \text{almost surely,} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{y}(\boldsymbol{\xi}) \geq \boldsymbol{0} \qquad & \text{almost surely.} \end{aligned} \tag{3.16}$$

The vector $\boldsymbol{q}$ in (3.16) in the second stage represents the *costs* of recourse $\boldsymbol{y}$. The deterministic constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$ are connected to the first-stage decision and the

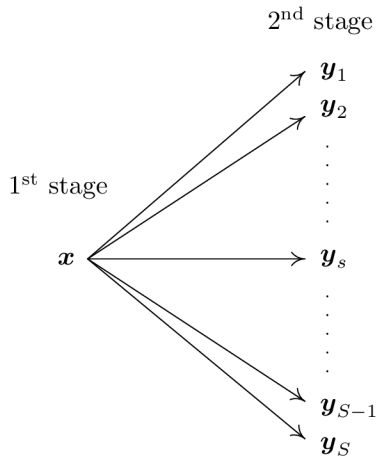

Figure 3.3: Two-stage decisions structure and the requirement of nonanticipativity

constraints $\boldsymbol{T}(\boldsymbol{\xi})\boldsymbol{x} + \boldsymbol{W}\boldsymbol{y}(\boldsymbol{\xi}) = \boldsymbol{h}(\boldsymbol{\xi})$ and $\boldsymbol{y}(\boldsymbol{\xi}) \geq \boldsymbol{0}$ are connected to the both first-stage and second-stage decisions.

The generalization of programme (3.16) for the non-linear case may have the following form: Find $\boldsymbol{x}$ and $\boldsymbol{y}$ such that

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) + \mathcal{Q}(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{g}^1(\boldsymbol{x}) \leq \boldsymbol{0}, \\ &&& \boldsymbol{h}^1(\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{3.17}$$

where $\mathcal{Q}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\xi}}\Big(Q(\boldsymbol{x}, \boldsymbol{\xi})\Big)$ and

$$\begin{aligned} &Q(\boldsymbol{x}, \boldsymbol{\xi}) = \min_{\boldsymbol{y}}\Big\{q\Big(\boldsymbol{y}(\boldsymbol{\xi}), \boldsymbol{\xi}\Big)\Big\} \\ &\text{subject to} \qquad \boldsymbol{t}_1(\boldsymbol{x}, \boldsymbol{\xi}) + \boldsymbol{g}^2\Big(\boldsymbol{y}(\boldsymbol{\xi}), \boldsymbol{\xi}\Big) \leq \boldsymbol{0} \text{ almost surely}, \\ &\qquad\qquad\quad \boldsymbol{t}_2(\boldsymbol{x}, \boldsymbol{\xi}) + \boldsymbol{h}^2\Big(\boldsymbol{y}(\boldsymbol{\xi}), \boldsymbol{\xi}\Big) = \boldsymbol{0} \text{ almost surely}. \end{aligned} \tag{3.18}$$

Observe that programmes (3.16) and (3.17) search in their second stages for the minimum over $\boldsymbol{y}$. This means that they are looking for the *cheapest* recourse.

Similarly to the linear case, functions $\boldsymbol{g}^1$ and $\boldsymbol{h}^1$ are linked to the first-stage decision and functions $\boldsymbol{t}_1$, $\boldsymbol{t}_2$, $\boldsymbol{g}^2$ and $\boldsymbol{h}^2$ are linked to the both first-stage and second-stage decisions.

We assume that all these functions are continuous in $\boldsymbol{x}$ for any fixed $\boldsymbol{\xi}$ and measurable in $\boldsymbol{\xi}$ for any fixed $\boldsymbol{x}$ since under this assumption the function $Q(\boldsymbol{x}, \boldsymbol{\xi})$ is measurable and $\mathcal{Q}$ well-defined. The function $\mathcal{Q}$ is called the *recourse cost function*.

The very important thing in two-stage (and also in multi-stage) stochastic programming is that the first-stage decision has to safisfy the requirement of *nonanticipativity*. The decision maker has to take the first-stage decision *before* any observation of random parameters $\boldsymbol{\xi}$ is available and known. The requirement of nonanticipativity means that the first-stage decision has to be independent on the future realization of $\boldsymbol{\xi}$. In other words, the first-stage decision is constant for whatever happens in the future. This idea is schematically shown in Figure 3.3.

We introduced the two-stage linear and non-linear stochastic programmes with recourse. We have to note that the second-stage decision – the recourse – is required only

when the recourse action is needed and this recourse brings additional costs. Thus, we assume that in case the recourse action is not required, the second-stage decision $\boldsymbol{y} = \boldsymbol{0}$ and the recourse cost is zero. These programmes are called *stochastic programmes with recourse*, (see [17]).

On the other hand, there exist many practical problems where the second-stage decision is not only the recourse action and is also required by the second-stage constraints. This means that this second-stage decision cannot be omited and must be taken also in case the recourse action is not needed. These programmes are called *two-stage stochastic programmes* in general, (see [17]).

The third case is the programme in which the second stage is composed of the combination of two discussed approaches – the first part is the decision required by the second stage itself and the second part is the recourse decision. These programmes are called *two-stage stochastic programmes with recourse*, (see [17]).

3.4.2 Scenario-Based Stochastic Programmes

In this chapter, we discuss several stochastic programmes and approaches. Observe that they include very frequently the expected values in the objective function or/and in the constraint equations. These expectations are in general in the integral form (see [17, 12, 2])

$$\mathbb{E}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x},\boldsymbol{\xi})\Big) = \int_{\Xi} f(\boldsymbol{x},\boldsymbol{\xi})\,\mathrm{d}P$$

and this fact brings computational problems since it is hard and expensive to compute these expectations repeatedly. Moreover, these integrals are often multidimensional.

Because of these reasons, many practitioners often deal with *scenario-based* programmes and *scenario analysis* (see [23]). This approach is also very convenient in case the random parameter $\boldsymbol{\xi}$ has the finite discrete distribution, i.e., $|\Xi| < \infty$. The idea of scenario-based programme follows.

The uncertainty and random events are modelled by *scenarios*. The scenario s is actually a set of particular values $\boldsymbol{\xi}^s$ of random parameters. The set of all scenarios is denoted by

$$S = \Big\{s^i \,|\, i = 1, \ldots, L\Big\}$$

where L is the number of scenarios. We can take all scenarios in case the set Ξ is small. In other hand, if the set Ξ is large, we can, for instance, ask a specialist in desired branch to choose several scenarios according to their relevance.

We denote p_s the probability that the scenario $s \in S$ occurs, $p_s = P(\boldsymbol{\xi} = \boldsymbol{\xi}^s) \geq 0$ and $\sum_{s\in S} p_s = 1$. Hence, the expected value of some function $\varrho(\cdot, \boldsymbol{\xi})$ is then easily

$$\mathbb{E}_{\boldsymbol{\xi}}\Big(\varrho(\cdot,\boldsymbol{\xi})\Big) = \sum_{s\in S} p_s \varrho(\cdot, \boldsymbol{\xi}^s).$$

The advantage is that we can put out all scenarios having $p_s = 0$ that cannot occur. In the following paragraphs, we will use the notation $\boldsymbol{y}_s = \boldsymbol{y}(\boldsymbol{\xi}^s)$, $\boldsymbol{q}_s = \boldsymbol{q}(\boldsymbol{\xi}^s)$, $\boldsymbol{W}_s = \boldsymbol{W}(\boldsymbol{\xi}^s)$,

$\boldsymbol{T}_s = \boldsymbol{T}(\boldsymbol{\xi}^s)$ and $\boldsymbol{h}_s = \boldsymbol{h}(\boldsymbol{\xi}^s)$. The scenario-based two-stage stochastic linear programme has the form

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T\boldsymbol{x} + \sum_{s\in S} p_s Q(\boldsymbol{x}, \boldsymbol{\xi}^s) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{3.19}$$

where

$$\begin{aligned} Q(\boldsymbol{x}, \boldsymbol{\xi}^s) = \min_{\boldsymbol{y}_s} & \left\{ \boldsymbol{q}_s^T \boldsymbol{y}_s \right\} \\ \text{subject to} \quad & \boldsymbol{T}_s\boldsymbol{x} + \boldsymbol{W}_s\boldsymbol{y}_s = \boldsymbol{h}_s, \\ & \boldsymbol{y}_s \geq \boldsymbol{0}. \end{aligned} \tag{3.20}$$

We can rewrite the above programme to the more explicit structure that is often used:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T\boldsymbol{x} + \sum_{s\in S} p_s \boldsymbol{q}_s^T \boldsymbol{y}_s \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} && && = \boldsymbol{b}, \\ & \boldsymbol{T}_1\boldsymbol{x} + \boldsymbol{W}_1\boldsymbol{y}_1 && && = \boldsymbol{h}_1, \\ & \boldsymbol{T}_2\boldsymbol{x} && + \boldsymbol{W}_2\boldsymbol{y}_2 && = \boldsymbol{h}_2, \\ & \vdots && \ddots && \vdots \\ & \boldsymbol{T}_S\boldsymbol{x} && && + \boldsymbol{W}_S\boldsymbol{y}_S = \boldsymbol{h}_S, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \quad \boldsymbol{y}_1 \geq \boldsymbol{0}, \ldots, \boldsymbol{y}_S \geq \boldsymbol{0}. \end{aligned} \tag{3.21}$$

It is obvious that the size of programme (3.21) very quickly grows with the number of scenarios. The non-linear case of the two-stage scenario-based stochastic programme reads:

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) + \sum_{s\in S} p_s Q(\boldsymbol{x}, \boldsymbol{\xi}^s) \\ \text{subject to} \quad & \boldsymbol{g}^1(\boldsymbol{x}) \leq \boldsymbol{0}, \\ & \boldsymbol{h}^1(\boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{3.22}$$

where

$$\begin{aligned} Q(\boldsymbol{x}, \boldsymbol{\xi}^s) = \min_{\boldsymbol{y}_s} & \left\{ q(\boldsymbol{x}, \boldsymbol{y}_s, \boldsymbol{\xi}^s) \right\} \\ \text{subject to} \quad & \boldsymbol{t}_1(\boldsymbol{x}, \boldsymbol{\xi}^s) + \boldsymbol{g}^2\Big(\boldsymbol{y}(\boldsymbol{\xi}^s), \boldsymbol{\xi}^s\Big) \leq \boldsymbol{0} \text{ almost surely}, \\ & \boldsymbol{t}_2(\boldsymbol{x}, \boldsymbol{\xi}^s) + \boldsymbol{h}^2\Big(\boldsymbol{y}(\boldsymbol{\xi}^s), \boldsymbol{\xi}^s\Big) = \boldsymbol{0} \text{ almost surely}. \end{aligned} \tag{3.23}$$

As we mentioned earlier, we require the fulfillment of *nonanticipativity* in stochastic programmes. In the context of scenario-based programmes, this requirement means that the first-stage solution $\boldsymbol{x}$ has to be *scenario-independent*. This can be ensured either directly as in (3.22) or by adding the *nonanticipativity constraints* (see Sections 4.1, 4.2, 5.1) to the programme:

$$\forall\, u, v \in S: \quad \boldsymbol{x}_u = \boldsymbol{x}_v.$$

3.5 Multi-Stage Stochastic Programming

The generalization of the two-stage stochastic programmes presented in the foregoing section leads to the *multi-stage stochastic programmes*. The main idea of those programmes is following: there are several stages T (more than two) and the decision maker takes his decisions sequentially and also the realizations of random parameters are sequentially observed and become known. These actions proceed alternatively. This process for T stages can be schematically illustrated as follows:

$$\begin{aligned}
&\boldsymbol{x}_1 \longrightarrow \boldsymbol{\xi}_1 \longrightarrow \boldsymbol{x}_2(\boldsymbol{x}_1, \boldsymbol{\xi}_1) \longrightarrow \boldsymbol{\xi}_2 \longrightarrow \boldsymbol{x}_3(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{\xi}_1, \boldsymbol{\xi}_2) \longrightarrow \\
&\longrightarrow \boldsymbol{\xi}_3 \longrightarrow \cdots \longrightarrow \boldsymbol{\xi}_{t-1} \longrightarrow \boldsymbol{x}_t(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_{t-1}) \longrightarrow \cdots \\
&\quad \cdots \longrightarrow \boldsymbol{\xi}_{T-1} \longrightarrow \boldsymbol{x}_T(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{T-1}, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_{T-1}) \quad [\longrightarrow \boldsymbol{\xi}_T].
\end{aligned}$$

In general, the decision $\boldsymbol{x}_t$ in stage t depends on all previous decisions $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{t-1}$ and all realizations of random parameters $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_{t-1}$. From the decision maker point of view, the observation of $\boldsymbol{\xi}_T$ in square brackets after the last desicion $\boldsymbol{x}_T$ is only additional information and the decision maker cannot react to it.

Note that in multi-stage programmes there can occur the difficulty with possible dependencies of random parameters across stages or even within stages.

Also remark that the two-stage programme is actually the special case of multi-stage programme.

Since this thesis is not aimed at multi-stage programming, we refer the reader to [17, 2, 12] for detailed information about multi-stage programmes and their formulations.

CHAPTER 4

Progressive Hedging Algorithm

In this chapter, we will describe a *progressive hedging algorithm* – a method for solving scenario-based stochastic optimization problems. At first, we will present this method and its theory for one-stage optimization problems based on the *scenario aggregation principle*. Further, we will generalize this algorithm for multi-stage optimization problems.

The progressive hedging algorithm was developed and introduced by R. J.-B. Wets and R. T. Rockafellar in 1980s, more detailed information the reader can find in their papers [23] and [19].

Let us recall that the scenario-based optimization models and scenario analysis are mathematical tools for modelling and analysing the stochastic optimization problems. The uncertainty in scenario-based models is modelled by scenarios – this means that stochastic parameters can reach only specified values and each setting of stochastic parameters is modelled by one scenario (see Subsection 3.4.2). Denote the set of all scenarios by S,

$$S = \left\{ s^i \,|\, i = 1, \ldots, L \right\},$$

where L is the number of all scenarios. For each scenario $s \in S$, we have a following subproblem:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}, s) \\ \text{subject to} & \boldsymbol{x} \in C_s, \end{array} \tag{4.1}$$

where $f(\boldsymbol{x}, s)$ is the objective function and the set C_s is the feasible set for the given scenario s. Assume that this subproblem has the optimal scenario-solution $\boldsymbol{x}^s$ for all $s \in S$. The scenario analysis deals with the analyse of all scenario-solutions $\boldsymbol{x}^s$, investigates if there is a general trend or if there are clusters of solutions. Then, a weighted "combination" of scenario-solutions $\boldsymbol{x}^s$ is constructed and again analyzed by the scenario analysis, etc. Our aim is to find one solution that will be in some sence optimal when an arbitrary scenario situation occurs.

Let us denote by p_s for all $s \in S$ the *weight* corresponding to the scenario s and its solution $\boldsymbol{x}^s$, where $p_s \geq 0$ and $\sum_{s \in S} p_s = 1$. We can consider that weight p_s as the probability that a particular scenario s occurs. The weights may be obtained, for instance,

from experts according to the relative importance of each scenario. Further, define an "average" solution $\hat{\boldsymbol{x}}$ as

$$\hat{\boldsymbol{x}} = \sum_{s \in S} p_s \boldsymbol{x}^s.$$

We can consider this combination of scenario-solutions as a defence against uncertainty of the model.

On the other hand, if we want to find the solution that will be resistant and robust with respect to all possible scenarios that can occur, we should solve the following problem:

$$\begin{aligned} &\text{minimize} && \sum_{s \in S} p_s f(\boldsymbol{x}, s) \\ &\text{subject to} && \boldsymbol{x} \in \bigcap_{s \in S} C_s. \end{aligned} \tag{4.2}$$

The solution $\boldsymbol{x}^*$ of the problem (4.2) is the solution that hedges all possible realizations of uncertain parameters that can occur.

Nevertheless, there are several reasons to deal with scenario analysis instead of solving the problem (4.2) directly. One reason is that the problem (4.2) is often very large and difficult to solve and exceeds computational capacity. Other reason is that the weights can be changed and the decision maker can easily observe how these changes in weights change the solution. The third significant reason for the scenario analysis is that the parallel computing facility can be used to process several scenarios at the same time and to speed up the calculations.

4.1 Aggregation Principle for One-stage Stochastic Programmes

In this section, we will present a method for one-stage stochastic programming problems based on the scenario aggregation principle that "blends" scenario-solutions $\boldsymbol{x}^s$ of problems (4.1) to the overall solution that will converge to the solution $\boldsymbol{x}^*$ of the problem (4.2). Note that the presented algorithm is not only one using the aggregation principle.

Assume that all scenario-solutions $\boldsymbol{x}^s$ of subproblems (4.1) for $s \in S$ are known. The solution $\hat{\boldsymbol{x}} = \sum_{s \in S} p_s \boldsymbol{x}^s$ is said to be *implementable* which means that it is scenario-independent. A solution $\boldsymbol{x}$ is said to be *admissible* if it is feasible for all scenario subproblems, i.e., for each $s \in S$ (see [23]). In other words, admissibility means that $\boldsymbol{x} \in \bigcap_{s \in S} C_s$, where C_s is a feasible set for a given scenario s.

Our goal is to find the solution $\boldsymbol{x}^*$ to the problem (4.2) that is *feasible* which means both implementable and admissible (see [23]). On the other hand, admissibility is not a property of solution that has to be unconditionally satisfied. We can imagine that the decision maker can accept a solution that is "slightly" unadmissible, for instance, if the violation of feasible set C_s happens for a paricular scenario s that is unlikely to occur. Hence, we can accept a solution that is nearly admissible.

However, in the algorithm described below, we will look for a solution that will be *feasible*, thus, it will be implementable and also admissible. This method will generate from scenario-solutions $\boldsymbol{x}^s$ a sequence of solutions $\hat{\boldsymbol{x}}^j$, $j = 1, 2, \ldots$ that will converge to the solution $\boldsymbol{x}^*$ of (4.2). The terms $\hat{\boldsymbol{x}}^j$ are obtained by increasing the requirement that the scenario-solutions $\boldsymbol{x}^s$ to the subproblem (4.1) have to be implementable. The algorithm follows.

One-stage Progressive Hedging Algorithm

Initialization. Choose the penalty parameter $\varrho > 0$ and the termination parameter $\varepsilon > 0$. Set $\boldsymbol{w}^0(s) = \mathbf{0}$ for each $s \in S$, $\hat{\boldsymbol{x}}^0 = \mathbf{0}$ and $j = 1$.

Main part.

1. For each $s \in S$ solve the problem

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}, s) + \boldsymbol{w}^{j-1}(s)^T \cdot \boldsymbol{x} + \tfrac{1}{2}\varrho \left\| \boldsymbol{x} - \hat{\boldsymbol{x}}^{j-1} \right\|^2 \\ &\text{subject to} && \boldsymbol{x} \in C_s \end{aligned} \tag{4.3}$$

   and denote its solution as $\boldsymbol{x}^{js}$.

2. Calculate

$$\hat{\boldsymbol{x}}^j = \sum_{s \in S} p_s \boldsymbol{x}^{js}.$$

   If the termination condition

$$\delta = \left(\left\| \hat{\boldsymbol{x}}^{j-1} - \hat{\boldsymbol{x}}^j \right\|^2 + \sum_{s \in S} p_s \left\| \boldsymbol{x}^{js} - \hat{\boldsymbol{x}}^j \right\|^2 \right)^{\frac{1}{2}} \leq \varepsilon \tag{4.4}$$

   holds, then stop, $\hat{\boldsymbol{x}}^j$ is the solution to the problem (4.2) with given tolerance ε. Otherwise, calculate

$$\boldsymbol{w}^j(s) = \boldsymbol{w}^{j-1}(s) + \varrho(\boldsymbol{x}^{js} - \hat{\boldsymbol{x}}^j)$$

   for each $s \in S$, set $j = j+1$, and return to step 1 of the main part of algorithm.

Several comments to the algorithm follow. Observe that the algorithm generates a sequence of solutions converging to the solution $\boldsymbol{x}^*$ of the problem (4.2), but it only requires the capability to solve scenario-based subproblems (4.1) and linear-quadratic perturbed versions of them.

Indeed, in the objective function (4.3), there are two so-called *penalty* terms in the comparison with the objective function of subproblems (4.1). This arrangement is slightly based on the *augmented Lagrangian* function. This topic is described in detail in Appendix C.

Recall that our goal is to find a solution that will be optimal when an arbitrary scenario occurs. This means that we need to find $\hat{\boldsymbol{x}}^j$ that is close to $\boldsymbol{x}^{js}$ for all $s \in S$. Hence, we see that due to the quadratic penalty term $\frac{1}{2}\varrho\|\boldsymbol{x} - \hat{\boldsymbol{x}}^{j-1}\|^2$, the algorithm is "forced" to seek $\boldsymbol{x}^{js}$ close to $\hat{\boldsymbol{x}}^{j-1}$. Further, the linear penalty term $\boldsymbol{w}^{j-1}(s)^T \cdot \boldsymbol{x}$, specifically the term $\boldsymbol{w}^{j-1}(s)$ penalizes the difference between $\boldsymbol{x}^{js}$ and $\hat{\boldsymbol{x}}^j$ from the foregoing iteration of the algorithm.

Also notice that the termination criterion (4.4) of the algorithm "measures" how close $\hat{\boldsymbol{x}}^j$ is to $\boldsymbol{x}^{js}$ for all $s \in S$ and how $\hat{\boldsymbol{x}}^j$ varies with j. Hence, δ in (4.4) is called the distance parameter. If the square root of the sum of squares of differences in all coordinates in vectors $\boldsymbol{x}^{js}$ and $\hat{\boldsymbol{x}}^j$ plus the square of the difference between $\hat{\boldsymbol{x}}^j$ and $\hat{\boldsymbol{x}}^{j-1}$ is less than $\varepsilon > 0$, then the loop of algorithm is terminated since the solution with given tolerance ε has been found, otherwise the algorithm proceeds to the next iteration.

Finally, remark that norms used above are Euclidean norms that are defined by

$$\|\boldsymbol{x}-\boldsymbol{y}\| = \Big((x_1-y_1)^2+\cdots+(x_N-y_N)^2\Big)^{\frac{1}{2}},$$

where N is the dimension of both vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. Nevertheless, one can imagine to use another norm than Euclidean, for instance norms $\|\boldsymbol{x}-\boldsymbol{y}\|_1 = \sum_i^N |x_i-y_i|$ or $\|\boldsymbol{x}-\boldsymbol{y}\|_\infty = = \max_i |x_i-y_i|$.

4.2 Progressive Hedging Algorithm for Multi-Stage Stochastic Programmes

In the foregoing section, we presented the algorithm for solving one-stage stochastic programming problems based on the aggregation principle. In this section, we will extend our considerations and will present a modification of that algorithm for multi-stage stochastic programming problems. This algorithm is called the *progressive hedging algorithm.*

At the beginning, recall basic ideas of multi-stage stochastic programming. The main distinction between one-stage and multi-stage stochastic programming is that a multi-stage case has more decision stages and the decision maker takes more decisions than only one. More precisely, the decision maker has to take the first-stage decision before a realization of stochastic parameters $\boldsymbol{\xi}^s$ is known. Then, the programme continues to the second stage, a particular realization $\boldsymbol{\xi}^s$ of random parameters becomes available and the decision maker continues with his second-stage decision. Then, next particular realization of random parameters is again observed and the process continues to the decision of the third stage, etc. Hence, the process of making decisions and spreading information is schematically following:

$$1^{\text{st}} \text{ decision } \boldsymbol{x}_1 \quad\longrightarrow\quad \boldsymbol{\xi}^s \text{ is observed} \quad\longrightarrow\quad 2^{\text{nd}} \text{ stage decision } \boldsymbol{x}_2(\boldsymbol{\xi}^s) \quad\longrightarrow\quad \cdots$$

It is evident that the question *"What* does the decision maker know and *when* does he know it?" is crucial in multi-stage stochastic programming. Note that the decision stages necessarily need not correspond to time periods, but we will keep this notation.

Let $t = 1,\ldots,T$ be the stage index and assume that the decision maker is confronted with scenario s. Then $\boldsymbol{x}_t(s)$ denotes a decision with respect to the scenario s in stage t. The mapping $\boldsymbol{X}\colon S \to \underbrace{\mathbb{R}^{n_1}\times\cdots\times\mathbb{R}^{n_T}}_{T}$

$$\boldsymbol{X}(s) = \Big(\boldsymbol{x}_1(s),\ldots,\boldsymbol{x}_T(s)\Big)$$

that assigns to each $s\in S$ a vector consisting of decisions in particular stages is called a *policy.* The very important property of policies is that if there are two scenarios s^i and s^j that are for the decision maker undistinguishable under available information at time $\hat{t}$, then decisions $\boldsymbol{x}_1(s^i),\ldots,\boldsymbol{x}_{\hat{t}}(s^i)$ and $\boldsymbol{x}_1(s^j),\ldots,\boldsymbol{x}_{\hat{t}}(s^j)$ generated by the policy $\boldsymbol{X}$ have to be identical. Thus, it must hold

$$\boldsymbol{x}_\tau(s^i) = \boldsymbol{x}_\tau(s^j)$$

for all $\tau = 1,\ldots,\hat{t}$. This consideration is formalized below.

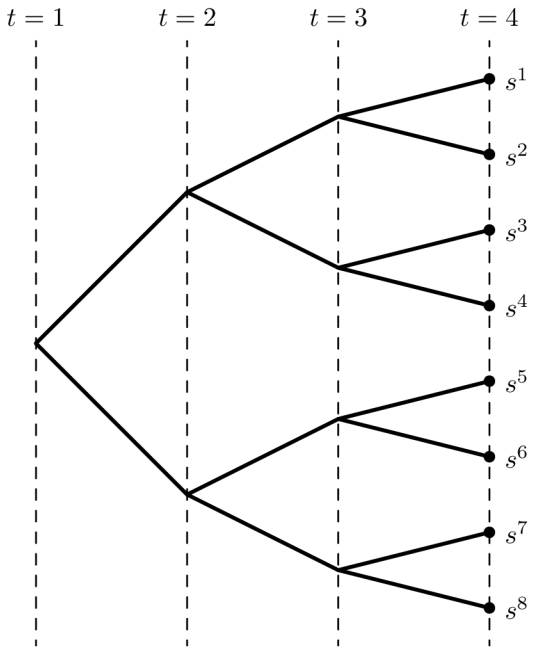

Figure 4.1: An example: a scenario structure

Let $\mathscr{P}_t$ be the coarsest partition of set of all scenarios S in sets A_k with property that if $A_k \in \mathscr{P}_t$ and scenarios $s^i, s^j \in A_k$, then the scenarios s^i and s^j are *undistinguishable* up to time t. This means that the decision $\boldsymbol{x}_t(\cdot)$ may only depend on the information that is available at time t. In other words,

$$\boldsymbol{x}_t(\cdot) \text{ must be constant on } A_k \text{ for each } A_k \in \mathscr{P}_t \text{ for all } t = 1, \ldots, T. \tag{4.5}$$

Example of Scenario Structure and Its Partitions

To make clear the meaning of scenario structure and its partitioning, see Figure 4.1 and 4.2. Figure 4.1 describes the scenario structure in time from the decision maker point of view for some four-stage case. It is obvious that at the beginning, in stage one, there is no available significant information that could help the decision maker to distinguish between all scenarios. This means that the decision in the first stage has to be identical for all scenarios $s \in S$. More presicely, the first component $\boldsymbol{x}_1(s)$ of policy $\boldsymbol{X}(s) = (\boldsymbol{x}_1(s), \ldots, \boldsymbol{x}_4(s))$ is not a function of s, but a constant, say $\boldsymbol{\alpha}$. Hence, the policy $\boldsymbol{X}$ is $\boldsymbol{X}(s) = (\boldsymbol{\alpha}, \cdot, \cdot, \cdot)$ for each $s \in S$.

In the second stage, new information is observed and the decision maker can distinguish two classes of scenarios. The first class A_1 includes scenarios s^1, s^2, s^3 and s^4 and the second class A_2 includes scenarios s^5, s^6, s^7 and s^8. The new available information can help the decision maker to identify the class including the scenario he is confronted with. But the decision maker still cannot distinguish between all scenarios in both classes A_1 and A_2. Hence, if we are confronted with a scenario from class A_1, then the second stage decision is, say, $\boldsymbol{\beta}$. On the other hand, in case a scenario is from class A_2, the second stage decision $\boldsymbol{x}_2 = \boldsymbol{\gamma}$. Thus, the policy is $\boldsymbol{X}(s) = (\boldsymbol{\alpha}, \boldsymbol{\beta}, \cdot, \cdot)$ for $s \in \{s^1, s^2, s^3, s^4\}$ and $\boldsymbol{X}(s) = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \cdot, \cdot)$ for $s \in \{s^5, s^6, s^7, s^8\}$. Similarly, in the third stage, new information again becomes available and the decision maker can refine two classes A_1 and A_2 from the second stage to new four classes A_1, A_2, A_3 and A_4. Each new class A_k, $k = 1, \ldots, 4$

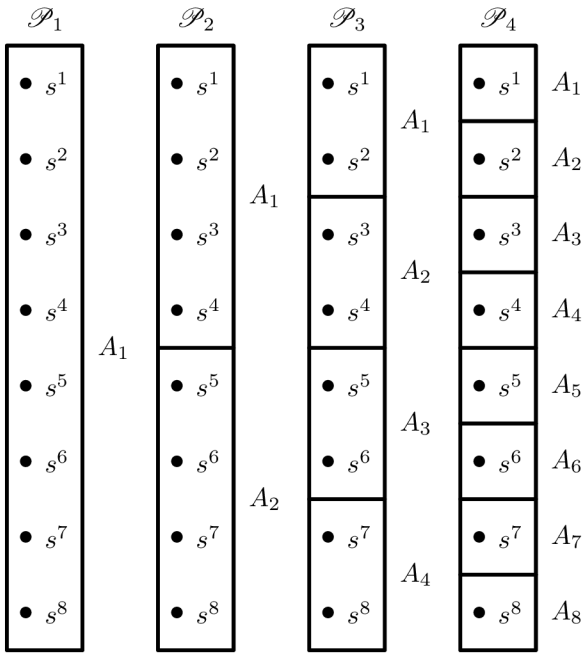

Figure 4.2: An example: partitions $\mathscr{P}_t$

includes two scenarios that are undistinguishable for the decision maker. In the third stage, the policy $\boldsymbol{X}$ is

$$\boldsymbol{X}(s) = \begin{cases} (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\delta}, \cdot) & s \in \{s^1, s^2\}, \\ (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\varepsilon}, \cdot) & s \in \{s^3, s^4\}, \\ (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\zeta}, \cdot) & s \in \{s^5, s^6\}, \\ (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\eta}, \cdot) & s \in \{s^7, s^8\}. \end{cases}$$

In the fourth stage, new information again becomes known and allows to the decision maker to refine four classes of scenarios from the third stage to new eight classes A_k, $k = 1, \ldots, 8$, whereas each class A_k includes only one scenario s^k. Hence, the decision maker can identify each scenario and the policy $\boldsymbol{X}$ is

$$\boldsymbol{X}(s) = \begin{cases} (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\delta}, \boldsymbol{\theta}) & s = s^1, \\ (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\delta}, \boldsymbol{\iota}) & s = s^2, \\ (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\varepsilon}, \boldsymbol{\kappa}) & s = s^3, \\ (\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\varepsilon}, \boldsymbol{\lambda}) & s = s^4, \\ (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\zeta}, \boldsymbol{\mu}) & s = s^5, \\ (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\zeta}, \boldsymbol{\nu}) & s = s^6, \\ (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\eta}, \boldsymbol{\xi}) & s = s^7, \\ (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\eta}, \boldsymbol{\pi}) & s = s^8. \end{cases}$$

Now, we know how does the structure of policy $\boldsymbol{X}$ look like, but the open question is, of course, particular values of vectors $\boldsymbol{\alpha}, \ldots, \boldsymbol{\pi}$. Figure 4.2 describes the partitions $\mathscr{P}_t$ in sets A_k for $t = 1, \ldots, 4$.

Now, we can define the set $\mathscr{N}$ of all *implementable* policies by the condition (4.5),

$$\mathscr{N} = \{\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T) \colon S \to \underbrace{\mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_T}}_{T} \mid \boldsymbol{x}_t \text{ is constant on each } A \in \mathscr{P}_t \text{ for all } t = 1, \ldots, T\}.$$

Note that available information is usually increasing in time. Hence, the partition $\mathscr{P}_{t+1}$ in time $t+1$ is a refinement of partition $\mathscr{P}_t$ in time t. A policy $\boldsymbol{X}$ is said *admissible* if the following condition

$$\boldsymbol{X} \in \mathscr{C} = \{\boldsymbol{X} \mid \boldsymbol{X}(s) \in C_s \text{ for all } s \in S\}$$

holds, which means that the policy $\boldsymbol{X}$ satisfies the explicit constraints for all $s \in S$.

Now, we can formulate the multi-stage scenario-based stochastic programming problem: Find a policy $\boldsymbol{X} \colon S \to \underbrace{\mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_T}}_{T}$ that

$$\begin{aligned} &\text{minimize} && \sum_{s \in S} p_s f\Big(\boldsymbol{X}(s), s\Big) \\ &\text{subject to} && \boldsymbol{X} \in \mathscr{C}. \end{aligned} \tag{4.6}$$

Our aim is to find the solution to (4.6) that is *feasible*. A solution is feasible if it is both *implementable* and *admissible*. Similarly as in one-stage case, the requirement of admissibility can be waived if the violation of constraints is not "hard" or if the violation occurs only for unlikely scenario or scenarios. On the other hand, the requirement of implementability is crucial and cannot be violated or waived.

In what follows, we will introduce the multi-stage version of the algorithm described in the foregoing section based on the aggregation principle. Define for each $A_k \in \mathscr{P}_t$

$$p_{A_k} = \sum_{s \in A_k} p_s$$

and

$$\hat{\boldsymbol{x}}_t(A_k) = \frac{1}{p_{A_k}} \sum_{s \in A_k} p_s \boldsymbol{x}_t(s).$$

This vector $\hat{\boldsymbol{x}}_t(A_k)$ is a weighted "average" of $\boldsymbol{x}_t(s)$ of all scenarios $s \in A_k$ in time t. Then, by setting

$$\hat{\boldsymbol{x}}_t(s) = \hat{\boldsymbol{x}}_t(A_k) \text{ for all } s \in A_k,$$

we will get an implementable "average" policy $\hat{\boldsymbol{X}}(s) = (\hat{\boldsymbol{x}}_1(s), \ldots, \hat{\boldsymbol{x}}_T(s))$. The mapping $J \colon \boldsymbol{X} \mapsto \hat{\boldsymbol{X}}$ is a projection[1], depends on weights p_s and is called the *aggregation operator*.

The progressive hedging algorithm, PHA, presented below is the method based on the scenario aggregation principle and allows by blending solutions of particular scenario problems to generate a sequence of solutions converging to the solution of the problem (4.6) that is both implementable and admissible.

[1] a mapping J is called a *projection* if it holds that J is linear and $J^2 = J$

Progressive Hedging Algorithm

Initialization. Choose the penalty parameter $\varrho > 0$ and the termination parameter $\varepsilon > 0$. Set $\boldsymbol{W}^0(s) = (\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{T})$ for all $s \in S$, $\hat{\boldsymbol{X}}^0(s) = (\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{T})$ for all $s \in S$ and $j = 1$.

Main part.

1. For all $s \in S$ solve the problem

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T, s) + \sum_{t=1}^{T} \left(\boldsymbol{w}_t^{j-1}(s)^T \cdot \boldsymbol{x}_t + \tfrac{1}{2}\varrho \left\| \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t^{j-1}(s) \right\|^2 \right) \\ &\text{subject to} && \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \in C_s \end{aligned} \tag{4.7}$$

   and denote its solution as $\boldsymbol{X}^j(s) = \left(\boldsymbol{x}_1^j(s), \ldots, \boldsymbol{x}_T^j(s) \right)$.

2. Calculate for all $s \in S$ and $t = 1, \ldots, T$ an "average" solution $\hat{\boldsymbol{X}}^j$:

$$\hat{\boldsymbol{x}}_t^j(s) = \frac{1}{p_{A_k}} \sum_{s \in A_k} p_s \boldsymbol{x}_t^j(s).$$

   If the termination condition

$$\delta = \left(\sum_{t=1}^{T} \sum_{s \in S} \left\| \hat{\boldsymbol{x}}_t^{j-1}(s) - \hat{\boldsymbol{x}}_t^j(s) \right\|^2 + \sum_{t=1}^{T} \sum_{s \in S} p_s \left\| \boldsymbol{x}_t^j(s) - \hat{\boldsymbol{x}}_t^j(s) \right\|^2 \right)^{\frac{1}{2}} \leq \varepsilon \tag{4.8}$$

   holds, then stop, $\hat{\boldsymbol{X}}^j(s) = \left(\hat{\boldsymbol{x}}_1^j(s), \ldots, \hat{\boldsymbol{x}}_T^j(s) \right)$ is the solution to (4.6) with given tolerance ε. Otherwise, for all $t = 1, \ldots, T$ and $s \in S$ update the term

$$\boldsymbol{w}_t^j(s) = \boldsymbol{w}_t^{j-1}(s) + \varrho \left(\boldsymbol{x}_t^j(s) - \hat{\boldsymbol{x}}_t^j(s) \right),$$

   set $j = j + 1$ and return to the step 1 of the main part of the algorithm.

Several comments and remarks follow. Observe that similarly as in one-stage case, the progressive hedging algorithm produces a sequence of solutions to the problem (4.6), but it only requires the capability to solve scenario-based subproblems and their linear-quadratic perturbed versions.

If we will consider weights p_s as probabilities, then we can interpet the "average" decision $\hat{\boldsymbol{x}}_t$ as the conditional expectation of $\boldsymbol{x}_t$, $\hat{\boldsymbol{x}}_t(s) = \mathbb{E}(\boldsymbol{x}_t \,|\, A_k)(s)$.

Notice that the purpose of the term $\frac{1}{2}\varrho \|\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t^{j-1}\|^2$ is to penalize the "distance" between $\boldsymbol{x}_t$ and $\hat{\boldsymbol{x}}_t^{j-1}$, and therefore, the algorithm is "forced" to minimize this term to find an optimal solution. The meaning of terms $\boldsymbol{w}_t$ is the "prices" that are linked with the implicit constraints that feasible solutions – policies – have to be implementable.

The initial value of $\boldsymbol{W}^0(s)$ for all $s \in S$ is adjusted to $(\mathbf{0}, \ldots, \mathbf{0})$, but we can use any $\boldsymbol{W}^0(s)$ satisfying $\sum_{s \in A_k} p_s \boldsymbol{w}_t^0(s) = 0$ for all $A_k \in \mathscr{P}_t$ and all $t = 1, \ldots, T$. Also the initial value $\hat{\boldsymbol{X}}^0$ can be adjusted (instead of $(\mathbf{0}, \ldots, \mathbf{0})$) to "average" solution of initial scenario-solutions and this arrangement can very effectively improve the behaviour of the algorithm. Especially, this setting can increase the number of iterations needed to safisfy the given tolerance ε.

We have to note here that the behaviour of the progressive hedging algorithm significantly depends on the choice of penalty parameter ϱ. Unfortunately, the weakness of the progressive hedging algorithm is that there is no general rule how to determine the penalty parameter to obtain a "good" behaviour and the penalty parameter ϱ has to be determined by experiments. For special case of linear mixed-integer[2] programmes, J.-P. Watson, D. L. Woodruff and D. R. Strip suggested in their paper [22] techniques how to determine the appropriate penalty parameter and other improvement of using the progressive hedging algorithm. Also remark that the penalty parameter can change with iterations which brings an opportunity for heuristics.

The progressive hedging algorithm was also successfully tested for optimum design problems in civil engineering (see [30, 31]).

Note that the reader can also find useful advice and discussion about the progressive hedging algorithm, numerical techniques and software in [6].

Suppose that the objective functions $f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, s)$ and the feasible sets C_s are convex for all $s \in S$. Furthermore, denote $\boldsymbol{X}^*$ as a unique optimal solution to the problem (4.6) and assume that the condition

$$\{\boldsymbol{W} \in \mathscr{N}^{\perp} \mid -\boldsymbol{W}(s) \text{ is normal vector to the set } C_s \text{ at the point } \boldsymbol{X}^*(s)\} = \{\boldsymbol{0}\}$$

holds for all $s \in S$, where $\mathscr{N}^{\perp}$ denotes the orthogonal complement of $\mathscr{N}$. Then, the sequence of $\boldsymbol{X}^j$ generated by the progressive hedging algorithm converges to $\boldsymbol{X}^*$. For more information about the convergence in convex and also in non-convex problems see [19].

In our considerations, we have assumed that the weights p_s are obtained from experts regarding to the importance of each scenario. Note that it is usually very useful to analyze how the solution reacts when the weights are changed. R. J.-B. Wets in [23] presented the method how to obtain the solution for the changed weights when the solution $\boldsymbol{x}^*$ for particular weights is known.

[2] mixed-integer programmes are programmes in which a part of variables is restricted to be integers

CHAPTER 5

Parallel Implementation of Progressive Hedging Algorithm

In the foregoing chapter, we presented the progressive hedging algorithm – the method for solving scenario-based problems in stochastic optimization. The properties of this method allow us to use the parallel approach instead of classical serial techniques. This parallel implementation can significantly speed up the computing and save the time.

In this chapter, we will describe the parallel implementations of the progressive hedging algorithm, we will start with one-stage stochastic programmes and will illustrate its behaviour to a simple problem. Further, we will also describe the implementation for two-stage stochastic programmes and will test it for the farmer's problem.

5.1 Implementation Approaches to Progressive Hedging Algorithm

The very important fact is that the progressive hedging algorithm belongs to the class of decomposition methods. The crucial principle of those methods is that the original problem to be solved is decomposed into smaller subproblems and these subproblems are solved separately.

In particular, the progressive hedging algorithm decomposes the original problem into *independent* scenario-based subproblems. This so-called vertical decomposition occurs in the first step of the main part of the algorithm (see page 26). Indeed, the original problem is decomposed to L subproblems

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T, s) + \sum_{t=1}^{T} \left(\boldsymbol{w}_t^{j-1}(s)^T \cdot \boldsymbol{x}_t + \tfrac{1}{2}\varrho \left\| \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t^{j-1}(s) \right\|^2 \right) \\ &\text{subject to} && \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \in C_s \end{aligned}$$

for each $s \in S$, where $|S| = L$.

The requirement of nonanticipativity is ensured by the penalty term involving $\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t^{j-1}(s)$ with $t = 1$ in the objective function of each subproblem.

All these subproblems are independent to each other, and therefore, there is no rule in what order these subproblems have to be solved. In other words, the algorithm continues with step 2 as soon as all subproblems are solved.

The classical approach of step 1 is the serial implementation. This means that all subproblems are solved sequentially, one by one. This is schematically shown in Figure 5.1.

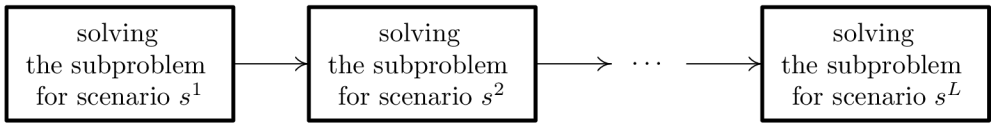

Figure 5.1: Serial approach – one by one

Denote the computing time needed for solving one subproblem for a particular scenario by τ and the total number of iterations of the progressive hedging algorithm by N. Hence, the computing time needed for solving all subproblems in one iteration is $T^s_{\text{iter}} = \tau L$ and the total computing time consumed by solving subproblems is $T^s_{\text{total}} = \tau N L$, where the superscript s stands for serial.

On the other hand, assume that we have a computing hardware with n processors. Thus, these n processors can solve n subproblems simultaneously – in parallel. This idea is shown in Figure 5.2.

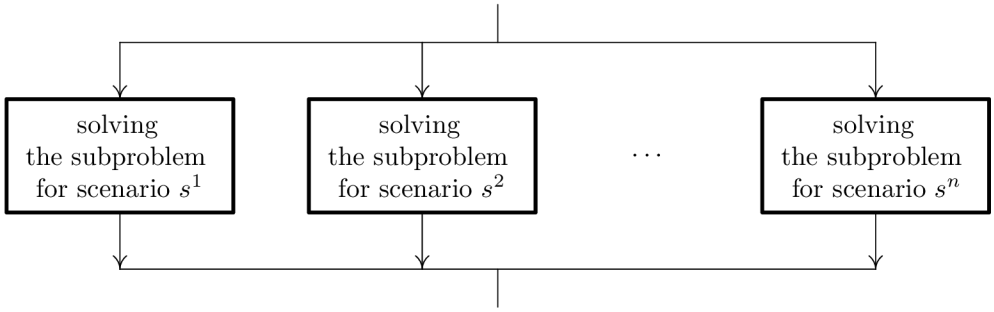

Figure 5.2: Parallel approach – n subproblems are solved simultaneously

Usually, the number of processors n is less than the number of subproblems (scenarios) L. Thus, the parallel part is repeated in a loop until all subproblems are solved, i.e., the loop is repeated $\left\lceil \frac{L}{n} \right\rceil$-times, where $\lceil x \rceil$ denotes the smallest integer greater than x. In other words, the first loop solves subproblems for scenarios $s^1, \ldots, s^n$. The second loop computes subproblems for scenarios $s^{n+1}, \ldots, s^{2n}$, etc. Of course, in the last loop, the number of used processors may be less than n. The schema of foregoing considerations is shown in Figure 5.3. Note that this approach is actually the combination of parallel computing of n subproblems and a serial loop performed $\left\lceil \frac{L}{n} \right\rceil$-times.

Hence, the computing time needed for solving n subproblems in one loop is τ and the computing time needed for solving all subproblems in $\left\lceil \frac{L}{n} \right\rceil$ loops in one iteration of the progressive hedging algorithm is $T^p_{\text{iter}} = \tau \left\lceil \frac{L}{n} \right\rceil$. Finally, the total computing time consumed by solving subproblems is $T^p_{\text{total}} = \tau N \left\lceil \frac{L}{n} \right\rceil$, where the superscript p stands for parallel.

Now, we can compare theoretical computing times for serial and parallel implementation of the progressive hedging algorithm. Assume that the algorithm will reach the

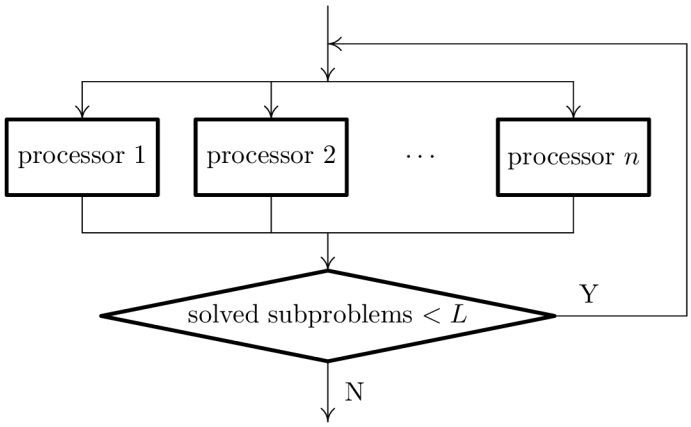

Figure 5.3: Parallel approach with loop

optimal solution in N iterations. It is obvious that the total computing time consumed by parallel implementation T^p_{total} is less or equal than the total computing time consumed by serial implementation T^s_{total},

$$T^s_{\text{total}} = \tau N L \leq \tau N \left\lceil \frac{L}{n} \right\rceil = T^p_{\text{total}}.$$

The equality holds for the number of parallel processors $n = 1$, which is actually the serial case. But with increasing n, $n < L$, the ratio $\frac{T^p_{\text{total}}}{T^s_{\text{total}}}$ decreases and for $n \geq L$ is constant and equals to $\frac{1}{L}$.

5.2 Principle of Parallel Implementation

In this section, we will give and describe the concept of parallel implementation of the progressive hedging algorithm. Note that this concept is, of course, not the only one that can be implemented.

The concept that the author of this thesis decided to implement is following. The main program is written in C++ programming language including the object-oriented programming approach. The purpose of that main program is to drive the flow of the algorithm and to make necessary computations. For the parallel running of optimization subproblems is used the Message Passing Interface, MPI. All scenario subproblems are solved by using GAMS which is a modelling system for mathematical programming. These ideas are schematically shown in Figure 5.4. Also note that all implementations and computing described in this thesis were realized in the operating system Linux Ubuntu 9.04 Jaunty Jackalope, 32 bits version.

5.2.1 Main Program

As we already mentioned, the main program is written in C++ including object-oriented programming approach and consists of three fundamental classes. We suppose that the reader is familiar with the basic ideas of object-oriented programming and with C++ programming language. The reader can find very detailed information about these topics in [18].

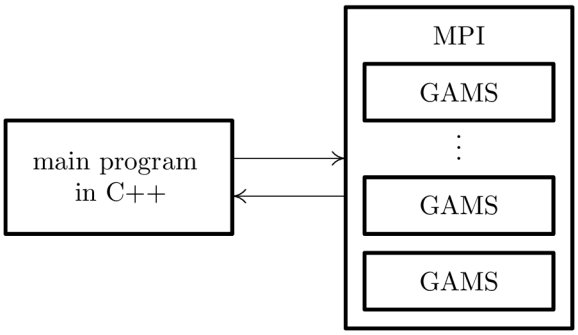

Figure 5.4: Concept of implementation

In the following, we will describe the framework of main program and the purposes of its three fundamental classes: SCENARIO, SCENARIOS and PHA.

SCENARIO This class is the basic element of main program and its instance represents one particular scenario that can occur. Its attributes (data) are particular values of random parameters $\boldsymbol{\xi}^s$, actual weights $\boldsymbol{W}$ and scenario solution $\boldsymbol{X}$, the probability p_s that this scenario occurs and several other secondary data (see Section 4.2). Also several methods are implemented, but they are important only from the programming point of view.

```
class SCENARIO {
  public:
    double * data;
    VECTOR * weights1;
    VECTOR * weights2;
    VECTOR * solution1;
    VECTOR * solution2;
    int numParameters;
    double probability;
    SCENARIO(int, int);
    SCENARIO(SCENARIO &);
    ~SCENARIO();
};
```

SCENARIOS The attributes of this class are the list of all scenarios, "average" solution $\hat{\boldsymbol{X}}$ and other secondary data. Further, several methods are implemented: method for loading the scenario setting (particular values of random parameters and the probability) from a text file, method for making GAMS source files, method for loading solutions of subproblems from GAMS output LST files, methods for calculating average solution $\hat{\boldsymbol{X}}$ and the distance parameter δ, updating weights $\boldsymbol{W}$ and several secondary methods.

```
class SCENARIOS {
  public:
    SCENARIO ** scenarios;
    VECTOR * averageSolution1;
    VECTOR ** averageSolution2;
    int numScenarios;
    int numParameters;
    int numVariables1;
    int numVariables2;
    double rho;
    SCENARIOS(char *, int, int);
    ~SCENARIOS();
    void changeRho(double);
    void checkInfeasibility(char *, int);
    void updateAllWeights(double);
```

```
    void printAllScenarioSolutions(void);
    void loadAllScenarioSolutionsFromLSTFiles(void);
    double calcTerminationCondition(void);
    void calcAverageSolution(void);
    void readScenarioFromLine(SCENARIO *, char *);
    void loadScenariosSettingFromFile(char*);
    void printAllScenarios(void);
    void checkIntegrity(double);
    void makeGAMSSourceFiles(char*);
    static void readMatrixDataFromLSTFile(char * sourceFile,
      char * nameOfVariable, MATRIX & m);
    static void readVectorDataFromLSTFile(char * sourceFile,
      char * nameOfVariable, VECTOR & v);
    static void readObjValueFromLSTFile(char * sourceFile,
      char * nameOfVariable, SCALAR & s);
};
```

PHA The instance of this class is the heart of main program. It consists of four attributes: an instance of SCENARIOS class, a template file and parameters ϱ and ε. The PHA class has two methods: one for running the progressive hedging algorithm and one for running GAMS via MPI in parallel.

```
class PHA {
  public:
    SCENARIOS * S;
    char * templet;
    double eps, rho;
    PHA(SCENARIOS *, char *, double, double);
    void runPHA(double, int);
    void runGAMSinParallel(int);
};
```

5.2.2 Message Passing Interface

Message Passing Interface, MPI, is an API library that allows to implement and run programs in parallel. The MPI is available for wide class of programming languages, but the most common is C, C++ and earlier also Fortran. There are several implementation of Message Passing Interface, for instance OpenMPI, MPICH or LAM/MPI. In the implementation of the progressive hedging algorithm in this thesis, the LAM/MPI implementation of MPI with C programming language has been used. The main aim of MPI is to maintain the running of GAMS in parallel. But in fact, the MPI is very powerful tool for parallel programming and the main feature is that programs running in parallel can exchange information and communicate to each other.

The fundamental principle is following. The programmer writes a program that he wishes to run in parallel and includes the MPI library, thereby MPI commands become available. Thus, the message passing can be implemented. Then, the source file is compiled by MPI compiler instead of ordinary compiler. In particular, the source file written in C has to be compiled by `mpicc` instead of ordinary `gcc` compiler. Finally, the compiled program can be run in parallel via MPI by command `mpirun`. For details, see e.g. [11].

5.2.3 GAMS

The General Algebraic Modeling System, GAMS, is a modeling system for solving problems of mathematical programming and optimization. This system includes solvers for linear, nonlinear and mixed integer problems of mathematical programming. The handling with GAMS is very similar to writing programs in ordinary programming language.

The mathematical problem is written in statements in a text file. This text file is an input for GAMS. The output is a LST file containing results and additional information about the execution. For details about GAMS, see [3].
Let us present a very simple example of nonlinear mathematical programme:

$$\begin{aligned} &\text{minimize} && x_1 + x_2 + 2x_3 + x_2^2 + 2x_3^2 \\ &\text{subject to} && x_1 + x_2 + x_3 \geq 2, \\ & && x_2 + x_3 = 1, \\ & && x_1,\, x_2,\, x_3 \geq 0, \\ & && x_1,\, x_2,\, x_3 \in \mathbb{R}. \end{aligned} \tag{5.1}$$

The GAMS source file for this programme reads:

```
Sets
    i   / 1, 2, 3 / ;
Parameters
    a(i)
      /  1   1
         2   1
         3   2  /
    b(i)
      /  1   0
         2   1
         3   2  /
    c(i)
      /  1   0
         2   1
         3   1  / ;
Variables
    z;
Positive variables
    x(i) ;
Equations
    objective
    constraintA
    constraintB ;
    objective .. z =e= sum(i, a(i) * x(i)) + sum(i, b(i) * x(i) * x(i)) ;
    constraintA .. sum(i, x(i)) =g= 2 ;
    constraintB .. sum(i, c(i) * x(i)) =e= 1 ;
Model example / all / ;
Solve example using nlp minimizing z ;
```

The GAMS output LST file contains information about the execution (if there are some non-optimal, infeasible, unbounded solutions or even errors) and these results: the optimal solution $\boldsymbol{x}_{\text{opt}} = (x_1, x_2, x_3)^T = (1; 0.8333; 0.1667)^T$ and the value of objective function $z = 2.9167$. The part of output LST file with optimal values of variables and the optimal value of objective function reads:

```
                            LOWER       LEVEL       UPPER      MARGINAL

---- VAR z                  -INF         2.9167     +INF          .

---- VAR x

         LOWER          LEVEL          UPPER         MARGINAL

1          .             1.0000        +INF             .
2          .             0.8333        +INF             .
3          .             0.1667        +INF           EPS
```

For the implementation of the progressive hedging algorithm, GAMS with the nonlinear solver CONOPT has been used for solving each scenario subproblem. The reader can find more detailed information about CONOPT solver in Appendix A.

5.3 How Does It Work?

In foregoing sections, we described the concept of parallel implementation of the progressive hedging algorithm and its individual parts. In what follows, we will put all considerations of above paragraphs together and will explain how exactly the parallel implementation of the progressive hedging algorithm works. This is also schematically shown in Figure 5.5.

The input for the program are the following two text files: the file describing the setting of scenarios – each scenario is represented by one line including its probability and particular values of random parameters, and the file containing the template of source files for GAMS. This template includes special labels instead of particular values of random parameters.

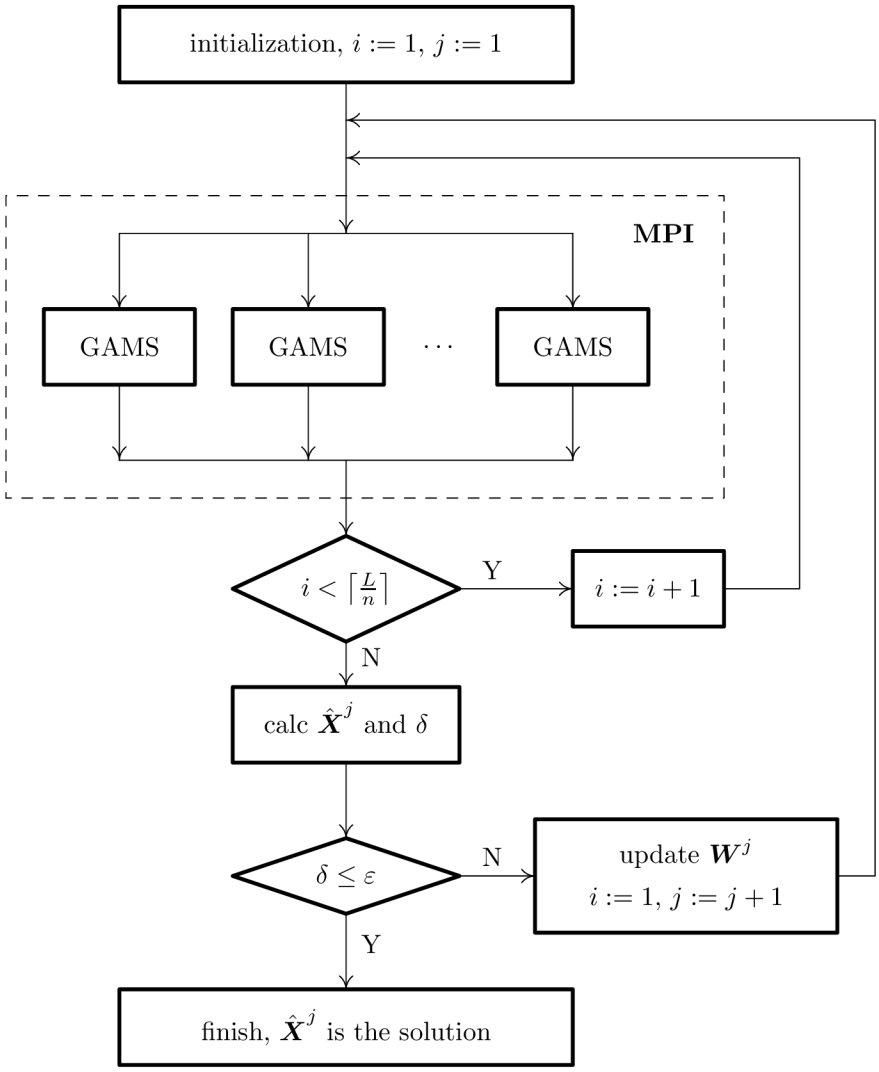

Figure 5.5: The concept of parallel implementation of the progressive hedging algorithm

The main program starts with the initialization part: it loads both input files described above, allocates the memory for the list of all scenarios (an instance of SCENARIOS class) and creates all scenarios (instances of SCENARIO class). In the initialization, it is also checked the consistency of scenarios (if all scenarios have the same number of random

parameters and if the sum of their probabilities equals to one). Also other parameters are set up: the number of available processors n for parallel computing, the termination tolerance ε and the penalty parameter ϱ.

Then, the main program creates the source files for GAMS according to the template file and to the setting of each scenario – special labels in the template are replaced by particular values of each scenario. The number of generated source files for GAMS equals to the number of all scenarios.

The main program continues with the parallel computing for first n scenario-based programmes via MPI. The MPI is started with n threads and each thread executes GAMS and solves own scenario programme. When last thread with GAMS is finished, also MPI is turned off. Now, it is checked if all scenarios were solved via GAMS. If not so, the MPI part is repeated with next n scenarios until all scenario programmes are solved.

Then, the main program loads all scenario solutions $\boldsymbol{X}$ from LST files (GAMS output files) and continues with the evaluation of "average solution" $\hat{\boldsymbol{X}}$ and the distance parameter δ. If that distance δ is less or equals to the required tolerance ε, the main program finishes, $\hat{\boldsymbol{X}}$ is the solution. Otherwise, the main program updates the weights $\boldsymbol{W}$ for all scenarios and returns back to the MPI part.

5.4 One-Stage Progressive Hedging Algorithm

In this section, we will deal with the implementation of the one-stage progressive hedging algorithm, PHA, and will test its behaviour for a simple problem. In next section, we will extend this one-stage implementation to two-stage case and will test it for the farmer's problem.

The progressive hedging algorithm for one-stage problems is described in Section 4.1 on page 20. It is actually special case of the general T-stage progressive hedging algorithm described on page 26 for $T = 1$ and this fact can be easily checked.

Recall that the scenario-based subproblems to be solved in the first step of main part read

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}, s) + \boldsymbol{w}^{j-1}(s)^T \cdot \boldsymbol{x} + \tfrac{1}{2}\varrho \left\| \boldsymbol{x} - \hat{\boldsymbol{x}}^{j-1}(s) \right\|^2 \\ &\text{subject to} && \boldsymbol{x} \in C_s, \end{aligned} \tag{5.2}$$

which are linear-quadratic perturbed versions of original programme.

Now, we will test our one-stage implementation for the simple problem including two scenarios and two variables that will help us to understand the behaviour of the algorithm. This problem is slightly based on example presented in [17]. We will also discuss what happend when we change probabilities of scenarios or the penalty parameter.

5.4.1 One-Stage PHA Example

Let us consider the following problem with two scenarios:

$$\begin{aligned} &\text{minimize} && (x_1 - \xi_1^s)^2 + (x_2 - \xi_2^s)^2 \\ &\text{subject to} && \xi_3^s \le x_1 \le \xi_4^s, \\ & && \xi_5^s \le x_2 \le \xi_6^s \end{aligned} \tag{5.3}$$

with the particular realization of random parameters for scenario s^1:

$$\boldsymbol{\xi}^1 = (\xi_1^1, \xi_2^1, \xi_3^1, \xi_4^1, \xi_5^1, \xi_6^1)^T = (3, 4, 1, 3, 2, 4)^T$$

and for scenario s^2:

$$\boldsymbol{\xi}^2 = (\xi_1^2, \xi_2^2, \xi_3^2, \xi_4^2, \xi_5^2, \xi_6^2)^T = (4, 3, 2, 4, 1, 3)^T.$$

The objective functions are actually paraboloids having vertices in points $(3, 4)$ and $(4, 3)$ and feasible sets are two shifted squares. More presicely: the objectives are paraboloids of type $z(x_1, x_2) = (x_1 - c_1)^2 + (x_2 - c_2)^2$ in $\mathbb{R}^3$ space with vertex in $(c_1, c_1, 0) \in \mathbb{R}^3$. This situation (2D view to x_1x_2 plane) is pictured in Figure 5.6: the scenario s^1 corresponding to $\boldsymbol{\xi}^1$ is represented by blue colour and the scenario s^2 corresponding to $\boldsymbol{\xi}^2$ is represented by red colour. The dashed circles represent cuts of paraboloids by planes parallel to x_1x_2 plane, so-called contours.

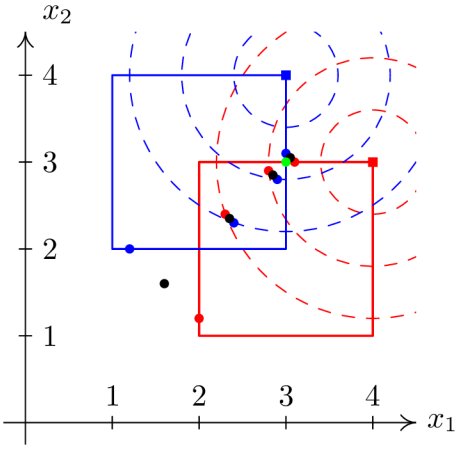

Figure 5.6: Scenario solutions and generated $\hat{\boldsymbol{x}}^j$ solution for setting $p_1 = p_2 = \frac{1}{2}$

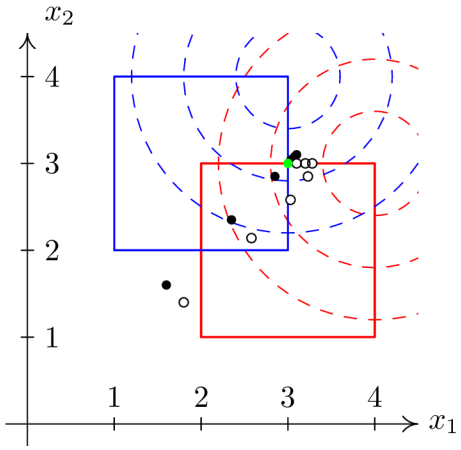

Figure 5.7: Generated $\hat{\boldsymbol{x}}^j$ solutions for setting $p_1 = p_2 = \frac{1}{2}$ and $p_1 = \frac{1}{4}$, $p_2 = \frac{3}{4}$

It is obvious that the optimal solution for the scenario s^1 is the right upper vertex of square feasible set, i.e., $\boldsymbol{x}^1_{\text{opt}} = (3, 4)^T$ and the optimal solution for scenario s^2 is the point $\boldsymbol{x}^2_{\text{opt}} = (4, 3)^T$. These points are pictured by small coloured squares in Figure 5.6.

The results produced by the progressive hedging algorithm are in Table 5.1 for the following setting: the probability of both scenarios equals to $\frac{1}{2}$, the penalty parameter $\varrho = 3$, the termination tolerance $\varepsilon = 10^{-9}$ and the distance parameter defined by (4.4).

The points $\hat{\boldsymbol{x}}^j$ produced by the algorithm (see Table 5.1) converge to the optimal point $\boldsymbol{x}_{\text{opt}} = (3, 3)^T$ (green point in Figure 5.6). That point is obviously the optimum for considered problem with two scenarios. It is also easy to check that this point $\boldsymbol{x}_{\text{opt}}$ is both implementable and admissible, i.e., it is feasible.

The dependence between the number of iterations of algorithm needed to satify the tolerance ε and the penalty parameter is shown in Table 5.2. The smallest number of iterations j needed for given $\varepsilon = 10^{-9}$ is for the choice of penalty parameter about $\varrho = 3$. The increasing of ϱ leads to the increasing of the number of iterations. On the other hand, also the decreasing of ϱ leads again to the increasing of the number of iterations. Hence, there is the significant relationship between the penalty parameter and the number of iterations needed to find the solution with the given tolerance. Therefore, the proper

Table 5.1: Results for problem (5.3): $\varrho = 3$, $p_1 = p_2 = \frac{1}{2}$

Iteration j	$\boldsymbol{x}^{j,1}$	$\boldsymbol{x}^{j,2}$	$\hat{\boldsymbol{x}}^j$	δ
1	$(1.2000, 2.0000)^T$	$(2.0000, 1.2000)^T$	$(1.6000, 1.6000)^T$	2.4000
2	$(2.4000, 2.3200)^T$	$(2.3200, 2.4000)^T$	$(2.3600, 2.3600)^T$	1.0778
3	$(2.8320, 2.8000)^T$	$(2.8000, 2.8320)^T$	$(2.8160, 2.8160)^T$	0.6457
⋮	⋮	⋮	⋮	⋮
10	$(3.0000, 3.0086)^T$	$(3.0086, 3.0000)^T$	$(3.0043, 3.0043)^T$	0.0155
11	$(3.0000, 3.0006)^T$	$(3.0006, 3.0000)^T$	$(3.0003, 3.0003)^T$	0.0057
⋮	⋮	⋮	⋮	⋮
15	$(3.0000, 2.9990)^T$	$(2.9990, 3.0000)^T$	$(2.9995, 2.9995)^T$	0.0010
16	$(3.0000, 3.0000)^T$	$(3.0000, 3.0000)^T$	$(3.0000, 3.0000)^T$	0.0007
17	$(3.0000, 3.0000)^T$	$(3.0000, 3.0000)^T$	$(3.0000, 3.0000)^T$	$< 10^{-9}$

choice of penalty parameter is crucial. Unfortunately, there is no general rule to determine the penalty parameter ϱ and its suitable value differs for different problems to be solved. This implies one of the weakness of the progressive hedging algorithm: the significant sensitivity of behaviour of the algorithm to the value of penalty parameter ϱ. A "good" value of ϱ has to be determine by experimentations.

Table 5.2: The numbers of iterations needed for different values of ϱ

Penalty parameter ϱ	Number of iterations	Penalty parameter ϱ	Number of iterations
0.1	> 100	5	24
0.5	73	10	25
1	39	20	29
2	19	50	41
3	17	300	> 100

We have tested the algorithm for two scenarios with the same probability $p_1 = p_2 = \frac{1}{2}$. But what happend when we change the probabilities of each scenario, for instance, to $p_1 = \frac{1}{4}$ and $p_2 = \frac{3}{4}$? In first moment, one can intuitively think that in this case the optimal solution $\boldsymbol{x}_{\text{opt}}$ should be close to the optimal solution of scenario s^2 (since its probability is greater than the probability of scenario s^1), i.e., to the point $(4, 3)$. But this is not possible since such a point (say a point $(3 + \varepsilon, \cdot)$, where $\varepsilon > 0$) is not admissible. In fact, candidates for the optimal solution have to be from the set $\{(x_1, x_2)\colon (x_1, x_2) \in \langle 2, 3\rangle \times \langle 2, 3\rangle\}$ satisfying the property of admissibility and the right upper corner of that set, i.e., the point $(3, 3)$, is the "closest" admissible point to both optimal scenario solutions $(3, 4)$ and $(4, 3)$.

Hence, the optimal solution for both probability settings is the point $\boldsymbol{x}_{\text{opt}} = (3,3)^T$. But there is another difference. The set of points $\hat{\boldsymbol{x}}^j$ generated by the algorithm for the setting $p_1 = \frac{1}{4}$ and $p_2 = \frac{3}{4}$ do not coincide with the set of points $\hat{\boldsymbol{x}}^j$ generated for the setting $p_1 = p_2 = \frac{1}{2}$. Several those points are pictured in Figure 5.7: black filled points are generated by the algorithm with setting $p_1 = p_2 = \frac{1}{2}$ and black unfilled points represent points generated with setting $p_1 = \frac{1}{4}$, $p_2 = \frac{3}{4}$. Observe that unfilled points are at first "attracted" by the dominant scenario s^2 to the point $(4,3)$, but due to the requirement of admissibility, next unfilled points converge to the point $(3,3)$.

Moreover, also the numbers of iterations needed to safisty the tolerance ε are not equal: 17 iterations for the setting $p_1 = p_2 = \frac{1}{2}$ and 26 iterations for the setting $p_1 = \frac{1}{4}$, $p_2 = \frac{3}{4}$.

5.5 Two-Stage PHA Modification

In the foregoing section, we deal with the one-stage implementation of the progressive hedging algorithm and tested it for a problem with two scenarios. This trivial one-stage problem was proposed to help the reader to understand the fundamental behaviour of the progressive hedging algorithm.

In this section, we will extend our considerations to the two-stage programme (see Section 3.4) and its implementation. We will derive the particular case of the progressive hedging algorithm (see Section 4.2) for two-stage programmes: all scenarios in the first stage will be undistiguishable and thus, the first-stage decision will be scenario independent. After the first-stage decision will be taken, a particular realization of random parameters will be observed and all scenarios in the second stage will become distinguishable to each other. This means that the partition $\mathscr{P}_1$ in the first stage will be the coarsest partition of set of all scenarios S in sets containing the only set A_1. On the other hand, the partition $\mathscr{P}_2$ in the second stage will be the finest partition of S including sets A_1, $\ldots$, A_L, where L is the number of all scenarios, i.e., $L = |S|$ (see Section 4.2 and Example on page 23).

The above considerations will lead to the particular case of the progressive hedging algorithm and also to its certain simplifications.

In what follows, we will start with the general T-stage progressive hedging algorithm (see page 26) and we will successively modify that for two-stage case described above.

Suppose L scenarios, i.e., $S = \{s^1, \ldots, s^L\}$ and $|S| = L$. Since $T = 2$ in two-stage programme, policies $\hat{\boldsymbol{X}}^j$ and weights $\boldsymbol{W}^j(s)$ consist of two components, $\hat{\boldsymbol{X}}^j = \left(\hat{\boldsymbol{x}}_1^j, \hat{\boldsymbol{x}}_2^j\right)$ and $\boldsymbol{W}^j(s) = \left(\boldsymbol{w}_1^j(s), \boldsymbol{w}_2^j(s)\right)$.

The scenario subproblems (step 1 of the main part of algorithm) read in general

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T, s) + \sum_{t=1}^{T} \left(\boldsymbol{w}_t^{j-1}(s)^T \cdot \boldsymbol{x}_t + \tfrac{1}{2}\varrho \left\| \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t^{j-1}(s) \right\|^2 \right) \\ &\text{subject to} && \boldsymbol{x}_1, \ldots, \boldsymbol{x}_T \in C_s. \end{aligned} \tag{5.4}$$

For desired two-stage case, we obtain the following subprogrammes:

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}_1, \boldsymbol{x}_2, s) + \boldsymbol{w}_1^{j-1}(s)^T \cdot \boldsymbol{x}_1 + \tfrac{1}{2}\varrho \left\| \boldsymbol{x}_1 - \hat{\boldsymbol{x}}_1^{j-1}(s) \right\|^2 + \\ & + \boldsymbol{w}_2^{j-1}(s)^T \cdot \boldsymbol{x}_2 + \tfrac{1}{2}\varrho \left\| \boldsymbol{x}_2 - \hat{\boldsymbol{x}}_2^{j-1}(s) \right\|^2 \\ \text{subject to} \quad & \boldsymbol{x}_1, \boldsymbol{x}_2 \in C_s. \end{aligned} \tag{5.5}$$

In the second step of the main part of algorithm, the "average" solutions $\hat{\boldsymbol{x}}_t^j$ for both $t = \{1, 2\}$ and all $s \in S$ are calculated by using the following formula:

$$\hat{\boldsymbol{x}}_t^j(s) = \frac{1}{p_{A_k}} \sum_{s \in A_k} p_s \boldsymbol{x}_t^j(s). \tag{5.6}$$

Since all scenarios are in the first stage undistinguishable, the partition $\mathscr{P}_1$ consists of the only set A_1 (in other words, the partition $\mathscr{P}_1$ consists of the set S itself), and therefore, all scenarios $s^1, \ldots, s^L$ are elements of A_1. Hence, $p_{A_1} = \sum_{s \in S} p_s = 1$ and for an arbitrary scenario $s \in S$, the first-stage "average" decision $\hat{\boldsymbol{x}}_1^j$ reads

$$\hat{\boldsymbol{x}}_1^j(s) = \frac{1}{p_{A_k}} \sum_{s \in A_k} p_s \boldsymbol{x}_1^j(s) = \sum_{s \in S} p_s \boldsymbol{x}_1^j(s) \equiv \hat{\boldsymbol{x}}_1^j.$$

Analogously, in the second stage, all scenarios are distinguishable after a particular realization $\boldsymbol{\xi}^p$ of random parameters is observed and thus, the partition $\mathscr{P}_2$ in the second stage consists of L sets $A_1, \ldots, A_L$. Each set A_k for all $k = 1, \ldots, L$ is a singleton, i.e., it consists of one element, $A_k = \left\{ s^k \right\}$ and $|A_k| = 1$.

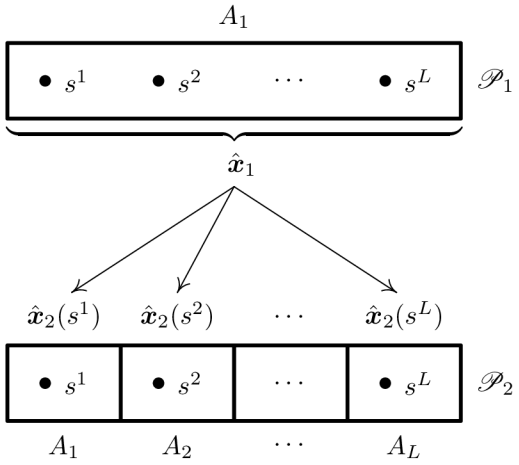

Figure 5.8: Partitions $\mathscr{P}_1$ and $\mathscr{P}_2$

Hence, $p_{A_k} = p_s$ and the second-stage "average" decision $\hat{\boldsymbol{x}}_2^j$ for a particular scenario s reads

$$\hat{\boldsymbol{x}}_2^j(s) = \frac{1}{p_{A_k}} \sum_{s \in A_k} p_s \boldsymbol{x}_2^j(s) = \boldsymbol{x}_2^j(s). \tag{5.7}$$

When both "average" solutions $\hat{\boldsymbol{x}}_1^j$ and $\hat{\boldsymbol{x}}_2^j$ are available, the distance parameter δ is calculated by the formula (4.8) for $T = 2$. Thus,

$$\delta = \left(\sum_{s \in S} \left\| \hat{\boldsymbol{x}}_1^{j-1}(s) - \hat{\boldsymbol{x}}_1^j(s) \right\|^2 + \sum_{s \in S} \left\| \hat{\boldsymbol{x}}_2^{j-1}(s) - \hat{\boldsymbol{x}}_2^j(s) \right\|^2 + \right.$$
$$\left. + \sum_{s \in S} p_s \left\| \boldsymbol{x}_1^j(s) - \hat{\boldsymbol{x}}_1^j(s) \right\|^2 + \sum_{s \in S} p_s \left\| \boldsymbol{x}_2^j(s) - \hat{\boldsymbol{x}}_2^j(s) \right\|^2 \right)^{\frac{1}{2}}. \quad (5.8)$$

But we derived in (5.7) that $\hat{\boldsymbol{x}}_2^j(s) = \boldsymbol{x}_2^j(s)$. Moreover, the first-stage decision $\hat{\boldsymbol{x}}_1^j(s)$ is scenario independent, i.e., $\hat{\boldsymbol{x}}_1^j(s) \equiv \hat{\boldsymbol{x}}_1^j$ is constant for all scenarios $s \in S$. Hence, the termination criteria (5.8) can be simplified to the form:

$$\delta = \left(L \left\| \hat{\boldsymbol{x}}_1^{j-1} - \hat{\boldsymbol{x}}_1^j \right\|^2 + \sum_{s \in S} \left\| \hat{\boldsymbol{x}}_2^{j-1}(s) - \hat{\boldsymbol{x}}_2^j(s) \right\|^2 + \sum_{s \in S} p_s \left\| \boldsymbol{x}_1^j(s) - \hat{\boldsymbol{x}}_1^j \right\|^2 \right)^{\frac{1}{2}}. \quad (5.9)$$

If the distance δ is greater than the tolerance ε, the algorithm continues with the updating of weights $\boldsymbol{W}(s)$ for all $s \in S$:

$$\begin{aligned} \boldsymbol{w}_1^j(s) &= \boldsymbol{w}_1^{j-1} + \varrho \left(\boldsymbol{x}_1^j(s) - \hat{\boldsymbol{x}}_1^j, \right), \\ \boldsymbol{w}_2^j(s) &= \boldsymbol{w}_2^{j-1} + \varrho \left(\boldsymbol{x}_2^j(s) - \hat{\boldsymbol{x}}_2^j(s) \right). \end{aligned} \quad (5.10)$$

But due to the fact that the vectors of weights are initialized to zero, i.e., $\boldsymbol{w}_1^0(s) = \boldsymbol{w}_2^0(s) = \mathbf{0}$ for all $s \in S$ and $\hat{\boldsymbol{x}}_2^j(s) = \boldsymbol{x}_2^j(s)$, we conclude that the vector of weights corresponing to the second stage will be equal to zero for any iteration $j \geq 0$, i.e., $\boldsymbol{w}_2^j(s) = \mathbf{0}$ for all $j \geq 0$ and all $s \in S$. Hence, the update (5.10) of $\boldsymbol{w}_2^j(s)$ can be omitted. Moreover, the vector of weights $\boldsymbol{W}^j(s)$ can be reduced to the form $\boldsymbol{W}^j(s) = \left(\boldsymbol{w}_1^j(s) \right)$, since its second element $\boldsymbol{w}_2^j(s)$ identically equals to zero. This arrangement will save the memory.

Finally, the above discussion leads to the following two-stage implementation of the progressive hedging algorithm.

Two-Stage Progressive Hedging Algorithm

Initialization. Choose the penalty parameter $\varrho > 0$ and the termination parameter $\varepsilon > 0$. Set $\boldsymbol{W}^0(s) = \boldsymbol{w}_1^0(s) = \mathbf{0}$ for all $s \in S$, $\hat{\boldsymbol{X}}^0(s) = (\mathbf{0}, \mathbf{0})$ for all $s \in S$ and $j = 1$.

Main part.

1. For all $s \in S$ solve the problem

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}_1, \boldsymbol{x}_2, s) + \boldsymbol{w}_1^{j-1}(s)^T \cdot \boldsymbol{x}_1 + \frac{1}{2}\varrho \left\| \boldsymbol{x}_1 - \hat{\boldsymbol{x}}_1^{j-1} \right\|^2 \\ \text{subject to} & \boldsymbol{x}_1, \boldsymbol{x}_2 \in C_s \end{array} \quad (5.11)$$

and denote its solution as $\boldsymbol{X}^j(s) = \left(\boldsymbol{x}_1^j(s), \boldsymbol{x}_2^j(s) \right)$.

2. Calculate for all $s \in S$ an "average" solution $\hat{\boldsymbol{X}}^j(s) = \left(\hat{\boldsymbol{x}}_1^j, \hat{\boldsymbol{x}}_2^j(s)\right)$:

$$\hat{\boldsymbol{x}}_1^j(s) = \hat{\boldsymbol{x}}_1^j = \sum_{s \in S} p_s \boldsymbol{x}_1^j(s),$$
$$\hat{\boldsymbol{x}}_2^j(s) = \boldsymbol{x}_2^j(s).$$

If the termination condition

$$\delta = \left(L \left\| \hat{\boldsymbol{x}}_1^{j-1} - \hat{\boldsymbol{x}}_1^j \right\|^2 + \sum_{s \in S} \left\| \hat{\boldsymbol{x}}_2^{j-1}(s) - \hat{\boldsymbol{x}}_2^j(s) \right\|^2 + \sum_{s \in S} p_s \left\| \boldsymbol{x}_1^j(s) - \hat{\boldsymbol{x}}_1^j \right\|^2 \right)^{\frac{1}{2}} \leq \varepsilon$$

holds, then stop, $\hat{\boldsymbol{X}}^j(s) = \left(\hat{\boldsymbol{x}}_1^j, \hat{\boldsymbol{x}}_2^j(s)\right)$ is the solution to the original programme with given tolerance. Otherwise, for all $s \in S$ update the term

$$\boldsymbol{w}_1^j(s) = \boldsymbol{w}_1^{j-1}(s) + \varrho \left(\boldsymbol{x}_1^j(s) - \hat{\boldsymbol{x}}_1^j \right),$$

set $j = j + 1$ and return to the step 1 of the main part of algorithm.

5.5.1 Two-Stage PHA Application: Farmer's Problem

In the foregoing section, we described and discussed the two-stage implementation of the progressive hedging algorithm. In this section, we will test this two-stage implementation in parallel manner that is based on discussions from Sections 5.2, 5.3 and 5.5. For this testing, the farmer's problem from the book [2] has been chosen. Notwithstanding, note that this problem was not solved in [2] by using the progressive hedging algorithm.

The farmer's problem is a typical problem of two-stage stochastic programming. Consider a farmer who has a farm and specializes in raising three crops: grain, corn and sugar beets. Assume that it is a winter and the farmer wants to decide how to divide his 500 acres of land available for the planting to maximize his total profit. In other words, how many acres of land should he devote to grain, corn and sugar beets?

He knows that 200 tons of wheat and 240 tons of corn for cattle feed are needed. This cattle feed can be raised on the farm or bought. Any production of wheat or corn over the feed requirement is sold. The production of the third crop – sugar beets – is completely sold. But there is a quota 6 000 tons for sugar beets production that is imposed by the goverment. If this quota is exceeded, the selling price of the abundance will be significantly reduced.

The uncertainty is included to the model via weather that significantly affects the yields of each crop. Most crops need the rain and the moisture at the beginning of the planting, then the sunshine is optimal with the occasional rain. The sunshine and dry weather is also grateful for the harvesting. Due to the above requirements, the yields depend on the weather during the whole planting period.

We will model the uncertainty of weather by the scenario approach. Assume three possibilities of yields depending on weather: profitable yields when the weather is favourable (scenario s^1), mean yields for the ordinary weather (scenario s^2) and lower yields when the weather is unfavourable (scenario s^3). Assume that the probabilities of all scenarios are equal. Thus, $p_1 = p_2 = p_3 = \frac{1}{3}$. All data and parameters are given in Table 5.3.

As we mentioned above, this model has the two-stage structure. This means that there are two decision moments when the farmer has to take his decisions.

Table 5.3: The parameters for the farmer's problem

Parameter	Wheat	Corn	Sugar
Total available land [acres]	500		
Profitable yield [tons/acre]	3	3.6	24
Mean yield [tons/acre]	2.5	3	20
Lower yield [tons/acre]	2	2.4	16
Planting cost [dollars/acre]	150	230	260
Selling price [dollars/ton]	170	150	36 under 6 000 tons
			10 above 6 000 tons
Purchase price [dollars/ton]	238	210	unavailable
Requirement for feeding [tons]	200	240	0

In winter, the farmer has to decide how to parcel his land to each crop for the next year. Obviously, he does not know what weather will occur during the whole planting period. Hence, this first-stage decision is under uncertainty. In particular, the type of the first-stage decision is *here-and-now* since the decision maker cannot wait and observe the particular weather – the first-stage decision has to be taken when no information about future weather is available.

The time goes on and the particular weather and also the yields become known. In other words, the particular realization of random parameters (weather) is observed. After the harvesting, the yields of each crop are known and the second-stage decision is taken. Now, the farmer has to decide how many tons of each crop should be sold and how many tons should be purchased from sellers. The type of the second-stage decision is *wait-and-see* since the decision is taken after the particular values of random parameters are observed and known. The aim of the farmer is still to maximize his total profit. The notation of variables used below is shown in Table 5.4.

Table 5.4: The variables for the farmer's problem

x_1	the number of acres devoted to wheat
x_2	the number of acres devoted to corn
x_3	the number of acres devoted to sugar beets
s_1	the number of tons of wheat to be sold
s_2	the number of tons of corn to be sold
s_3^*	the number of tons of sugar beets to be sold at the favourable price
s_3	the number of tons of sugar beets to be sold at the lower price
p_1	the number of tons of wheat to be purchased
p_2	the number of tons of corn to be purchased
ξ_1	the yield of wheat, $\xi_1 \in \{2, 2.5, 3\}$
ξ_2	the yield of corn, $\xi_2 \in \{2.4, 3, 3.6\}$
ξ_3	the yield of sugar beets, $\xi_3 \in \{16, 20, 24\}$

All above considerations lead to the following programme:

$$\begin{aligned}
\text{minimize} \quad & 150x_1 + 230x_2 + 260x_3 - 170s_1 - 150s_2 - 36s_3^* - 10s_3 + 238p_1 + 210p_2, \\
\text{subject to} \quad & x_1 + x_2 + x_3 \leq 500, \\
& \xi_1 x_1 + p_1 - s_1 \geq 200, \\
& \xi_2 x_2 + p_2 - s_2 \geq 240, \\
& s_3^* + s_3 \leq \xi_3 x_3, \\
& s_3^* \leq 6\,000, \\
& x_1, x_2, x_3, s_1, s_2, s_3^*, s_3, p_1, p_2 \geq 0.
\end{aligned} \tag{5.12}$$

As we alredy mentioned, the farmer wants to maximize his total profit. In other word, he wishes to minimize his costs and negative costs imply the positive profit for the farmer. Also observe that the above programme 5.12 includes three random parameters ξ_1, ξ_2, ξ_3. In fact, the uncertainty is modelled by three scenarios $\boldsymbol{\xi}^1 = (3, 3.6, 24)^T$, $\boldsymbol{\xi}^2 = (2.5, 3, 20)^T$ and $\boldsymbol{\xi}^3 = (2, 2.4, 16)^T$, where $\boldsymbol{\xi}^i = (\xi_1^i, \xi_2^i, \xi_3^i)^T$.

The fist-stage decision is the parcelling of the farmer's land $\boldsymbol{x}_1 = (x_1, x_2, x_3)^T$ and the second-stage decision is the answer to the question "How many tons of each crop should be sold and purchased to maximize the farmer's total profit?", $\boldsymbol{x}_2 = (s_1, s_2, s_3^*, s_3, p_1, p_2)^T$.

Hence, combining above considerations with the two-stage progressive hedging algorithm (see page 41), the subproblems to be solved in the first step of the main part of the algorithm read: for all $s \in S = \{s^1, s^2, s^3\}$ solve

$$\begin{aligned}
\text{minimize} \quad & 150x_1 + 230x_2 + 260x_3 - 170s_1 - 150s_2 - 36s_3^* - 10s_3 + 238p_1 + 210p_2 + \\
& + w_{1,1}^{j-1} x_1 + w_{1,2}^{j-1} x_2 + w_{1,3}^{j-1} x_3 + \\
& + \tfrac{1}{2}\varrho\left(\left(x_1 - \hat{x}_{1,1}^{j-1}\right)^2 + \left(x_2 - \hat{x}_{1,2}^{j-1}\right)^2 + \left(x_3 - \hat{x}_{1,3}^{j-1}\right)^2\right), \\
\text{subject to} \quad & x_1 + x_2 + x_3 \leq 500, \\
& \xi_1^s x_1 + p_1 - s_1 \geq 200, \\
& \xi_2^s x_2 + p_2 - s_2 \geq 240, \\
& s_3^* + s_3 \leq \xi_3^s x_3, \\
& s_3^* \leq 6\,000, \\
& x_1, x_2, x_3, s_1, s_2, s_3^*, s_3, p_1, p_2 \geq 0,
\end{aligned} \tag{5.13}$$

where $w_{1,i}^{j-1}$ with $i \in \{1, 2, 3\}$ are components of the weight vector $\boldsymbol{w}_1^{j-1} = \left(w_{1,1}^{j-1}, w_{1,2}^{j-1}, w_{1,3}^{j-1}\right)^T$ corresponding to the fist-stage decision $\boldsymbol{x}_1$ and $\hat{x}_{1,i}^{j-1}$ with $i \in \{1, 2, 3\}$ are components of the "average" solution $\hat{\boldsymbol{x}}_1^{j-1} = \left(\hat{x}_{1,1}^{j-1}, \hat{x}_{1,2}^{j-1}, \hat{x}_{1,3}^{j-1}\right)^T$. Both vectors $\boldsymbol{w}_1^{j-1}$ and $\hat{\boldsymbol{x}}_1^{j-1}$ have the same dimension as the first-stage decision, i.e., their dimension is three.

The results for the farmer's problem obtained by using the two-stage progressive hedging algorithm are shown in Table 5.5. Notice that the second-stage decision consists of six variables $\hat{\boldsymbol{x}}_2 = (\hat{x}_{2,1}, \hat{x}_{2,2}, \hat{x}_{2,3}, \hat{x}_{2,4}, \hat{x}_{2,5}, \hat{x}_{2,6})^T = (s_1, s_2, s_3^*, s_3, p_1, p_2)^T$ and only non-zero elements are shown in Table 5.5.

For this problem, the penalty parameter $\varrho = 0.25$ was chosen. The number of iterations needed to satisfy the termination condition $\delta \leq \varepsilon = 10^{-9}$ with $\varrho = 0.25$ was $j = 130$. The progresses of the elements $\hat{x}_{1,1}$, $\hat{x}_{1,2}$ and $\hat{x}_{1,3}$ of the first-stage decision $\hat{\boldsymbol{x}}_1$ are pictured in Figure 5.9, 5.10 and 5.11, respectively.

Table 5.5: The results for the farmer's problem obtained by using the progressive hedging algorithm with the penalty parameter $\varrho = 0.25$

Variable		The number of iterations							
		0	20	40	60	80	100	120	130
$\hat{\boldsymbol{x}}_1$	$\hat{x}_{1,1}$	134	129.73	165.66	169.25	170.24	169.98	189.99	170.00
	$\hat{x}_{1,2}$	57.00	96.11	82.29	80.36	79.88	80.01	80.00	80.00
	$\hat{x}_{1,3}$	308.00	274.16	252.05	250.36	249.88	250.01	250.00	250.00
$\hat{\boldsymbol{x}}_2^1$	$\hat{x}_{2,1}^1$	350.00	224.88	297.91	308.05	310.64	309.94	309.99	310.00
	$\hat{x}_{2,2}^1$	0.00	150.14	62.51	50.34	47.24	48.07	48.01	48.00
	$\hat{x}_{2,3}^1$	6000.0	6000.0	6000.0	6000.0	6000.0	6000.0	6000.0	6000.0
$\hat{\boldsymbol{x}}_2^2$	$\hat{x}_{2,1}^2$	100.00	100.00	215.52	223.37	225.53	224.95	224.99	225.00
	$\hat{x}_{2,3}^2$	6000.0	6000.0	5075.8	5013.0	4995.7	5000.0	5000.0	5000.0
$\hat{\boldsymbol{x}}_2^3$	$\hat{x}_{2,1}^3$	0.00	55.20	129.58	138.27	140.57	139.95	139.99	140.00
	$\hat{x}_{2,3}^3$	6000.0	4358.0	4037.9	4006.9	3997.7	4000.2	4000.0	4000.0
	$\hat{x}_{2,6}^3$	180.00	0.00	41.17	46.96	48.34	47.97	47.99	48.00
δ		13 348	1 920	17.07	0.47	0.05	$4 \cdot 10^{-4}$	$2 \cdot 10^{-5}$	$< 10^{-9}$

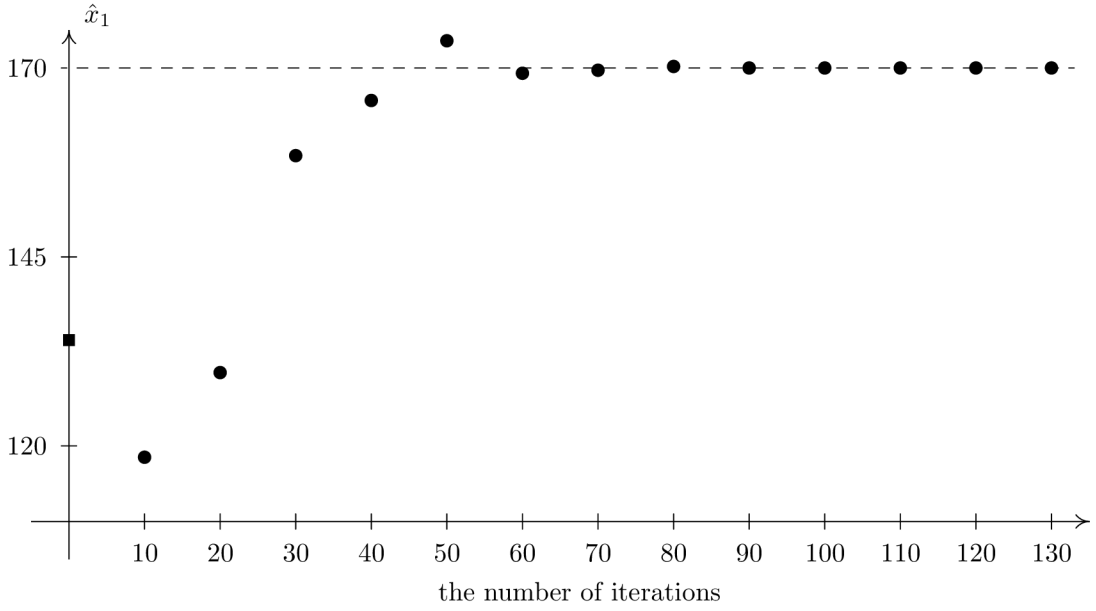

Figure 5.9: Progress of $\hat{x}_{1,1}$ of the fist-stage decision $\hat{\boldsymbol{x}}_1$

The results in Table 5.5 show that at the beginning of the planting period the farmer should devote 170 arces of his land to wheat, 80 acres to corn and 250 acres to sugar beets to minimize his costs,

$$\hat{\boldsymbol{x}}_1 = (170, 80, 250)^T.$$

Then, the farmer keeps with the planting and the particular weather becomes known for him.

If the weather during the planting period is favourable (in other words, the farmer is confronted with scenario s^1), he should take the second-stage decision

$$\hat{\boldsymbol{x}}_2 = \hat{\boldsymbol{x}}_2^1 = \left(s_1^1, s_2^1, s_3^{*;1}, s_3^1, p_1^1, p_2^1\right)^T = (310, 48, 6\,000, 0, 0, 0)^T.$$

This means that the farmer should sell 310 tons of wheat, 48 tons of corn and 6 000 tons of sugar beets. No wheat or corn must be purchased and the farmer's profit is 167 000 dollars.

Note that the value of objective function for $\hat{\boldsymbol{x}}_1$ and $\hat{\boldsymbol{x}}_2$ is negative since it represents the farmer's costs and the negative costs imply the positive farmer's profit.

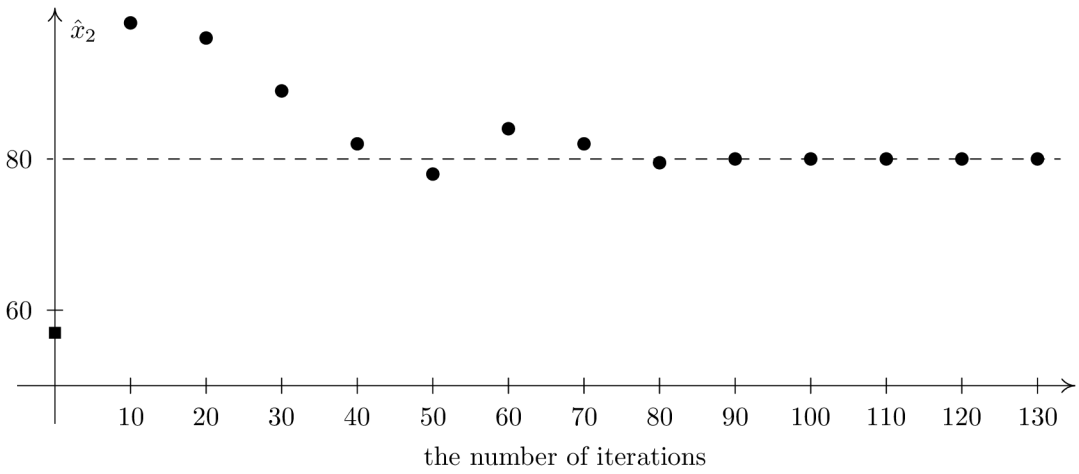

Figure 5.10: Progress of $\hat{x}_{1,2}$ of the fist-stage decision $\hat{\boldsymbol{x}}_1$

On the other hand, if the weather is ordinary (not favourable and also not unfavourable) which corresponds to the scenario s^2, the farmer should take the following second-stage decision:

$$\hat{\boldsymbol{x}}_2 = \hat{\boldsymbol{x}}_2^2 = \left(s_1^2, s_2^2, s_3^{*;2}, s_3^2, p_1^2, p_2^2\right)^T = (225, 0, 5\,000, 0, 0, 0)^T.$$

Hence, 225 tons of wheat and 5 000 tons of sugar beets should be sold and no wheat or corn must be purchased. This decision leads to the farmer's profit 109 350 dollars.

Finally, if the weather is unfavourable (scenario s^3), the farmer should take the second-stage decision

$$\hat{\boldsymbol{x}}_2 = \hat{\boldsymbol{x}}_2^3 = \left(s_1^2, s_2^2, s_3^{*;3}, s_3^2, p_1^2, p_2^2\right)^T = (140, 0, 4\,000, 0, 0, 48)^T.$$

Thus, the farmer should sell 140 tons of wheat and 4 000 tons of sugar beets. But in addition, he must purchase 48 tons of corn to feed his cattle. In this case, the farmer's profit is 48 820 dollars.

Let us discuss about the results obtained by using the progressive hedging algorithm. The farmer's profits are 167 000, 109 350 and 48 820 dollars for favourable, ordinary and unfavourable weather, respectively. Hence, the farmer's mean profit is 108 390 dollars provided $p_1 = p_2 = p_3 = \frac{1}{3}$.

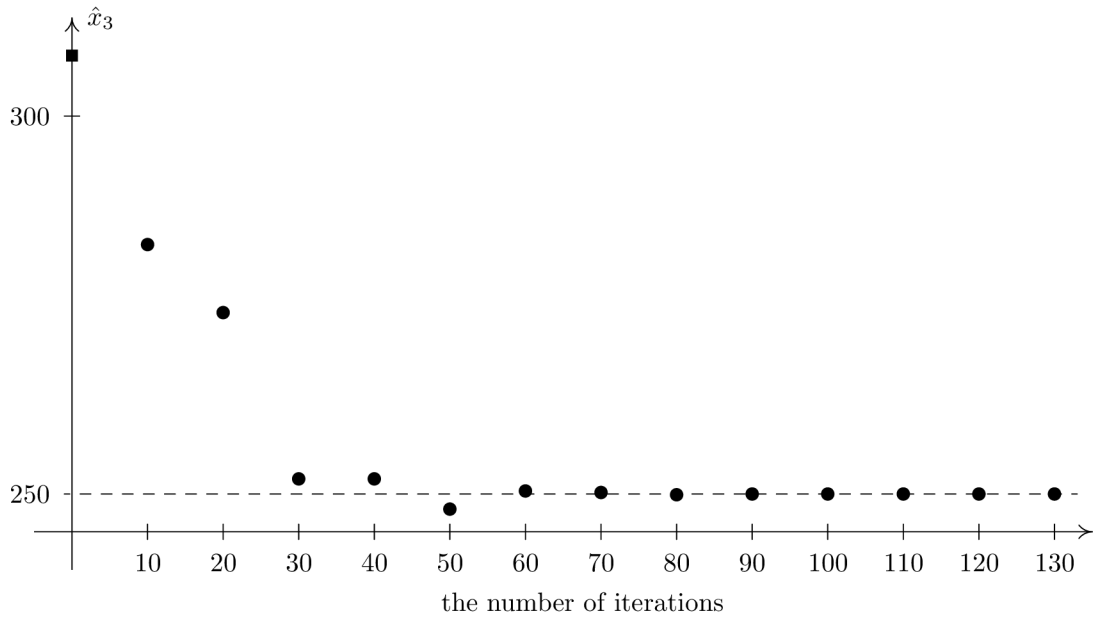

Figure 5.11: Progress of $\hat{x}_{1,3}$ of the fist-stage decision $\hat{\boldsymbol{x}}_1$

Assume the situation that in winter the farmer knows what weather will occur during the planting period. Thus, he wants to solve the programme 5.12 with particular values $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)^T$. Hence, for the favourable weather, the vector $\boldsymbol{\xi}^1 = (3, 3.6, 24)^T$ is chosen and the solution to 5.12 reads:

$$\widetilde{\boldsymbol{x}}_1^1 = (183.33, 66.67, 250)^T \qquad \widetilde{\boldsymbol{x}}_2^1 = (350, 0, 6\,000, 0, 0, 0)^T.$$

In this case, the farmer's profit is 167 667 dollars.

Similarly, for the ordinary weather, the programme 5.12 with $\boldsymbol{\xi}^2 = (2.5, 3, 20)^T$ leads to the solution

$$\widetilde{\boldsymbol{x}}_1^2 = (120, 80, 300)^T \qquad \widetilde{\boldsymbol{x}}_2^2 = (100, 0, 6\,000, 0, 0, 0)^T$$

with the farmer's profit 118 600 dollars.

For the unfavourable weather, the vector $\boldsymbol{\xi}^3 = (2, 2.4, 16)^T$ with programme 5.12 is considered and its solution is

$$\widetilde{\boldsymbol{x}}_1^3 = (100, 25, 375)^T \qquad \widetilde{\boldsymbol{x}}_2^3 = (0, 0, 6\,000, 0, 0, 180)^T.$$

The farmer's profit reachs 59 950 dollars.

Assume that the probabilities of each weather realization are equal to $\frac{1}{3}$. Thus, in the long run of planting seasons, the farmer's mean profit is 115 406 dollars. This corresponds to the situation under perfect information since the farmer knows what weather will occur during the planting period.

Unfortunately, in a real situation the farmer clearly does not know what weather will be realized in a coming planting period. Therefore, the best decision what the farmer can take is the decision given in Table 5.5. The difference between the mean profit under perfect information and the mean profit based on decisions given in Table 5.5, i.e., $115\,406 - 108\,390 = 7\,016$ dollars, is called the *expected value of perfect information*, EVPI (see Definition 3.4). This characteristic represents the loss of the profit corresponding to

the uncertainty included in the model. In other words, the EVPI represents the maximum that the farmer should pay to obtain perfect information about future, of course, in case that it is possible.

The second useful characteristic is the *value of stochastic solution*, VSS (see Definition 3.3). In this case, the *expected value of using EV solution*, EVV (see Definition 3.2), represents the farmer's mean profit by using EV programme (3.10). Hence, the first-stage decision $\boldsymbol{x}_1^{\mathrm{EV}} = (120, 80, 300)^T$ is taken and the second-stage decision depends on a particular scenario:

$$\begin{aligned}
\boldsymbol{x}_2^{\mathrm{EV},1} &= (160, 48, 6\,000, 1\,200, 0, 0)^T, \\
\boldsymbol{x}_2^{\mathrm{EV},2} &= (100, 0, 6\,000, 0, 0, 0)^T, \\
\boldsymbol{x}_2^{\mathrm{EV},3} &= (40, 0, 4\,800, 0, 0, 48)^T.
\end{aligned}$$

The corresponing profits are 148 000, 118 600 and 55 120 dollars, respectively, and thus, EVV = −107 240 dollars.

The value of stochastic solution, VSS, is given by (3.13), where $\mathbb{E}_{\boldsymbol{\xi}}\left(f\left(\boldsymbol{x}_{\min}^{\mathrm{EO}}, \boldsymbol{\xi}\right)\right)$ is the expected value of objective function with EO solution $\boldsymbol{x}_{\min}^{\mathrm{EO}}$. To find this $\boldsymbol{x}_{\min}^{\mathrm{EO}}$ solution, one has to solve a problem

$$\text{minimize} \quad \mathbb{E}_{\boldsymbol{\xi}}\Big(f\left(\boldsymbol{x}, \boldsymbol{\xi}\right)\Big).$$

But in the farmer's problem based on scenario approach, it holds

$$\mathbb{E}_{\boldsymbol{\xi}}\Big(f\left(\boldsymbol{x}, \boldsymbol{\xi}\right)\Big) = \sum_{s=1}^{3} p_s f\left(\boldsymbol{x}, \boldsymbol{\xi}^s\right).$$

Hence, to find $\boldsymbol{x}_{\min}^{\mathrm{EO}}$ requires to solve the problem

$$\text{minimize} \quad \sum_{s=1}^{3} p_s f\left(\boldsymbol{x}, \boldsymbol{\xi}^s\right). \tag{5.14}$$

But the problem (5.14) coincides with the programme (4.2) for which the progressive hedging algorithm gives the solution. Thus, $\boldsymbol{x}_{\min}^{\mathrm{EO},s}$ corresponing to the scenario s read

$$\begin{aligned}
\boldsymbol{x}_{\min}^{\mathrm{EO},1} &= \left(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2^1\right) = \left((170, 80, 250)^T, (310, 48, 6\,000, 0, 0, 0)^T\right), \\
\boldsymbol{x}_{\min}^{\mathrm{EO},2} &= \left(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2^2\right) = \left((170, 80, 250)^T, (225, 0, 5\,000, 0, 0, 0)^T\right), \\
\boldsymbol{x}_{\min}^{\mathrm{EO},3} &= \left(\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2^3\right) = \left((170, 80, 250)^T, (140, 0, 4\,000, 0, 0, 48)^T\right).
\end{aligned}$$

Finally,

$$\mathbb{E}_{\boldsymbol{\xi}}\left(f\left(\boldsymbol{x}_{\min}^{\mathrm{EO}}, \boldsymbol{\xi}\right)\right) = \sum_{s=1}^{3} p_s f\left(\boldsymbol{x}_{\min}^{\mathrm{EO},s}\right) = -108\,390 \text{ dollars}$$

and thus,

$$\mathrm{VSS} = \mathrm{EEV} - \mathbb{E}_{\boldsymbol{\xi}}\Big(f(\boldsymbol{x}_{\min}^{\mathrm{EO}}, \boldsymbol{\xi})\Big) = 1\,150 \text{ dollars}.$$

This value represents the gain obtained by using stochastic programming approaches, the progressive hedging algorithm and solving EO programme instead of simpler EV programme.

CHAPTER 6

Continuous Casting Process of Steel Slabs

In the foregoing Chapters 4 and 5, we discussed the progressive hedging algorithm – the method for solving scenario-based stochastic optimization programmes, and its parallel implementation based on the fact that the progressive hedging algorithm belongs to the class of decomposition algorithms.

We also tested this algorithm for two exemplary and simple problems – the problem including two paraboloids with the one-stage version of the progressive hedging algorithm, and the farmer's problem with the two-stage progressive hedging algorithm.

In this chapter, we will present the application of the two-stage progressive hedging algorithm to the problem of the *continuous casting process of steel slabs*. This problem is solved in the collaboration with Energy Institute, Department of Thermodynamics and Environmental Engineering at Faculty of Mechanical Engineering, Brno University of Technology.

For detailed information about the continuous casting process, see e.g. [13, 14, 15, 28].

6.1 Continuous Casting Method

The continuous casting process of steel slabs is the modern production method of steel in steelworks. In this process, *molten* steel is transformed to *solid* semi-finished products, so-called blanks: billets, blooms or slabs (see Figure 6.1). These semi-products are designated for next processing, for instance, for rolling in finishing mills.

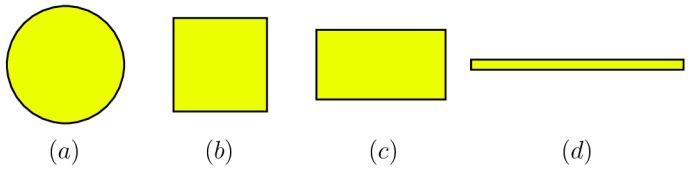

Figure 6.1: Products of continuous casting: (a) and (b) billets, (c) bloom, (d) slab

Before 1950, steel was casted into stationary molds to form ingots. In 1950s, the continuous casting method was evolved. This method has achieved to increase the productivity, quality and effectivity, and therefore, the continuous casting method became the most widespread method of steelmaking.

In these days, over 95 % of the world production of steel is produced by the continuous casting approach. Note that this method is, beside steel, also appropriate for the casting of aluminium, copper and their alloys.

The requirements of increasing in the productivity, decreasing in costs and the competitiveness involve more accurate models and sophisticated approaches. The optimization and mathematical programming approaches are welcome in these problems. Moreover, these models describing real situations also must embrace the uncertainty to depict a particular situation more credibly. These considerations lead to stochastic optimization techniques.

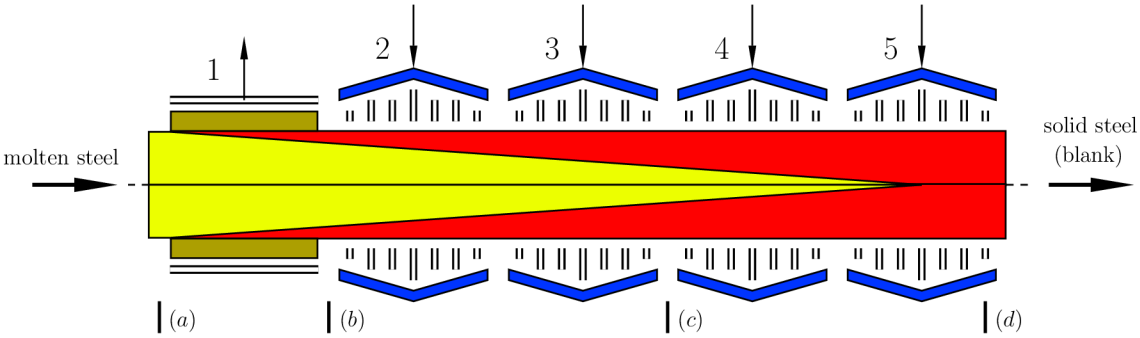

Figure 6.2: Schema of continuous casting process

The simplified schema of the continuous casting method is shown in Figure 6.2 where the yellow colour represents molten steel and the red colour represents solidified steel. The bold arrow on the left indicates molten steel that is transfered from an electric furnace or from a convector and enters to the continuous casting machine through the copper crystallizer (1). The crystallizer is water-cooled and its purpose is to form the profile of casted blank and to cool down its surface to form the solid crust. Thus, behind the crystallizer, the profile is already given by the solid crust, but under the crust – in the core – steel is still liquid. In Figure 6.3, there are shown the cross-sections of billet in four positions (a)–(d) corresponding to Figure 6.2, the position (b) corresponds to the cross-section behind the crystallizer mentioned above.

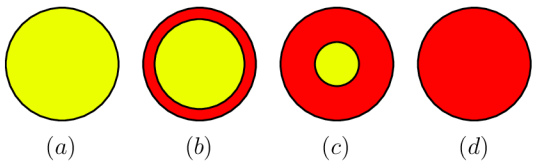

Figure 6.3: Cross-sections of a casted billet

From the crystallizer, steel with the crust on the surface continues through cooling zones (2)–(5). The purpose of these water-cooled zones is the further cooling down of casted billet so that the output of the continuous casting machine reachs the blank that is completely solidified in its cross-section (position (d) in Figures 6.2 and 6.3). The adjustment of the cooling parameters in zones (2)–(5) is crucial since it significantly

affects parameters and properties of casted steel. Note that the cooling parameters are actually the flow volumes of water flowing through each cooling zone.

Since the problem in Figure 6.2 is symmetric with respect to the horizontal axis, we will deal with the simplified model that is shown in Figure 6.4.

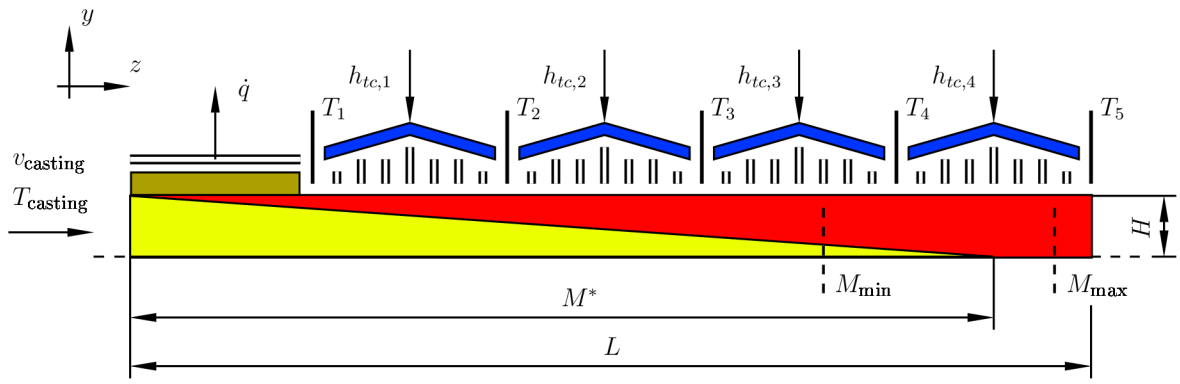

Figure 6.4: Schema of simplified continuous casting process

Our goal will be the maximization of the casting velocity v_{casting} (productivity) and to find the corresponing optimal adjustment of the cooling parameters $h_{tc,1}$, $h_{tc,2}$, $h_{tc,3}$ and $h_{tc,4}$ for required parameters of produced steel. The uncertainty will be also included to the model – we will deal with the question how the service of the continuous casting machine should change the cooling parameters $h_{tc,1}$, …, $h_{tc,4}$ when one cooling zone suddenly breaks down. The mathematical description and its derivation follow. Note that the deterministic model presented below is based on papers [13, 15] and especially on [14].

6.2 Mathematical Model of Continuous Casting Process

The mathematical description of the continuous casting process is based on the Fourier-Kirchhoff equation describing the heat conduction in a solidifying blank. Since the continuous casting process embraces several phases and their transformations, the enthalpy must be included to the model. Hence, the equation describing the process reads:

$$\frac{\partial H}{\partial t} = \lambda \left(\frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) + v_z \frac{\partial H}{\partial z} \qquad \text{in } \Omega \times (0, \tau\rangle, \tag{6.1}$$

where $T(y, z, t)$ is the temperature in a point (y, z) at time t, $H(y, z, t)$ represents the enthalpy in a point (y, z) at time t, λ is the heat conductivity and v_z is the casting velocity. The following initial and boundary conditions

$$T(y, z, 0) = T_0(y, z) \qquad \text{in } \Omega \times \{0\}, \tag{6.2}$$

$$H(y, z, 0) = H_0(y, z) \qquad \text{in } \Omega \times \{0\}, \tag{6.3}$$

$$T = T_{\text{casting}} \qquad \text{in } \Gamma_{\text{in}} \times (0, \tau\rangle, \tag{6.4}$$

$$-\lambda \frac{\partial T}{\partial n} = 0 \qquad \text{in } \Gamma_{\text{out}} \times (0, \tau\rangle \text{ and } \Gamma_S \times (0, \tau\rangle, \tag{6.5}$$

$$-\lambda \frac{\partial T}{\partial n} = \dot{q} \qquad \text{in } \Gamma_M \times (0, \tau\rangle, \tag{6.6}$$

$$-\lambda \frac{\partial T}{\partial n} = (h_{tc,j} + h_{tc_r}) (T_{\text{surface}} - T_\infty) \qquad \text{in } \Gamma_{C,j} \times (0, \tau\rangle, \quad j = 1, 2, 3, 4 \tag{6.7}$$

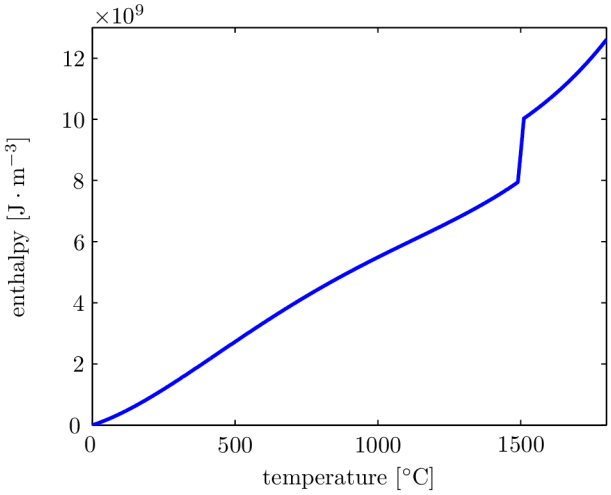

Figure 6.5: The enthalpy-temperature relationship for stell 11 325

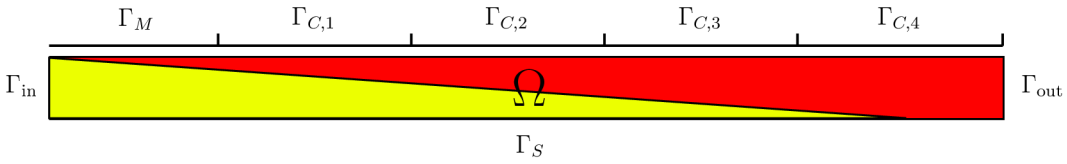

Figure 6.6: Schema of boundary condition surfaces

are added to the equation (6.1) and together form the mathematical model of the continuous casting process.

Further, the temperature-enthalpy relationship is known. But the difficulty is, due to the phase transformations, that it is given for particular steel by empirical values, not in an analytical form. The example of the enthalpy-temperature relationship for steel 11 325 is shown in Figure 6.5. Thus, an approximation or other techniques have to be used to obtain the above relationship in a practicable form.

To solve the continuous casting problem numerically, the above model including the second-order partial differential equation is discretized by the finite difference method.

Let Δy, Δz and Δt be the discretization steps in spatial axes y, z and in time axis t, respectively, and let $M+1$ be the number of nodes in the spatial axis y, $N+1$ be the number of nodes in the spatial axis z and $\tau+1$ be the number of nodes in time axis t. The discretized region Ω is shown in Figure 6.7. Thus, $T^n_{j,k}$ denotes the temperature in a node $[j,k]$ at time n. Similarly, $H^n_{j,k}$ denotes the enthalpy in a node $[j,k]$ at time n.

Hence, the equation (6.1) can be rewritten by the finite difference method as follows,

$$\frac{H^{n+1}_{j,k}-H^n_{j,k}}{\Delta t}=\lambda\left(\frac{T^n_{j+1,k}-2T^n_{j,k}+T^n_{j-1,k}}{(\Delta y)^2}+\frac{T^n_{j,k+1}-2T^n_{j,k}+T^n_{j,k-1}}{(\Delta z)^2}\right)+ \\ +v_z\left(\frac{H^n_{j,k}-H^n_{j,k-1}}{\Delta z}\right) \qquad \text{for} \qquad \begin{cases} j=1,\dots,M-1 \\ k=1,\dots,N-1 \\ n=1,\dots,\tau-1 \end{cases}. \tag{6.8}$$

The discretized initial conditions (6.2) and (6.3) read

$$T^0_{j,k}=T_0(j\Delta y,k\Delta z), \qquad H^0_{j,k}=H_0(j\Delta y,k\Delta z) \qquad \text{for} \qquad \begin{cases} j=1,\dots,M-1 \\ k=1,\dots,N-1 \end{cases}. \tag{6.9}$$

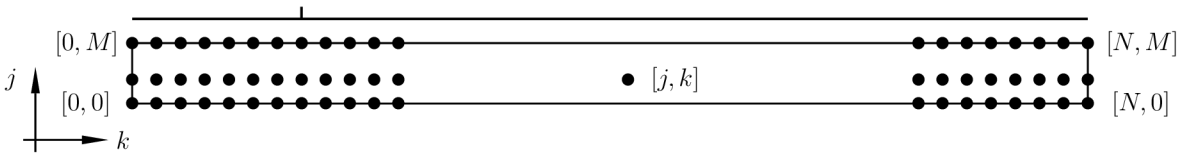

Figure 6.7: Schema of discretization

Similarly, the discretized boundary condition (6.4) reads

$$T^n_{j,0} = T_{\text{casting}} \quad \text{for} \quad \begin{cases} j = 1, \dots, M-1 \\ n = 1, \dots, \tau \end{cases} . \tag{6.10}$$

For the discretization of boundary condition (6.5), we will use the central difference and so-called fictive nodes. The discretization of (6.5) reads

$$-\lambda \left(\frac{T^n_{j,N+1} - T^n_{j,N-1}}{2\Delta z} \right) = 0 \quad \text{for} \quad \begin{cases} j = 1, \dots, M-1 \\ n = 1, \dots, \tau \end{cases} \tag{6.11}$$

and

$$-\lambda \left(\frac{T^n_{-1,k} - T^n_{1,k}}{2\Delta y} \right) = 0 \quad \text{for} \quad \begin{cases} k = 0, \dots, N \\ n = 1, \dots, \tau \end{cases} \tag{6.12}$$

Thus, $T^n_{j,N+1} = T^n_{j,N-1}$ from (6.11), and therefore, the equation for corresponding nodes reads:

$$\begin{aligned} \frac{H^{n+1}_{j,N} - H^n_{j,N}}{\Delta t} = \lambda \left(\frac{T^n_{j+1,N} - 2T^n_{j,N} + T^n_{j-1,N}}{(\Delta y)^2} + \frac{2T^n_{j,N-1} - 2T^n_{j,N}}{(\Delta z)^2} \right) + \\ + v_z \left(\frac{H^n_{j,N} - H^n_{j,N-1}}{\Delta z} \right) \quad \text{for} \quad \begin{cases} j = 1, \dots, M-1 \\ n = 1, \dots, \tau - 1 \end{cases} . \end{aligned} \tag{6.13}$$

Analogously, $T^n_{-1,k} = T^n_{1,k}$ from (6.12) and the equation reads:

$$\begin{aligned} \frac{H^{n+1}_{0,k} - H^n_{0,k}}{\Delta t} = \lambda \left(\frac{2T^n_{1,k} - 2T^n_{0,k}}{(\Delta y)^2} + \frac{T^n_{0,k+1} - 2T^n_{0,k} + T^n_{0,k-1}}{(\Delta z)^2} \right) + \\ + v_z \left(\frac{H^n_{0,k} - H^n_{0,k-1}}{\Delta z} \right) \quad \text{for} \quad \begin{cases} k = 0, \dots, N \\ n = 1, \dots, \tau - 1 \end{cases} . \end{aligned} \tag{6.14}$$

From the discretization $\frac{T^n_{M+1,k} - T^n_{M-1,k}}{2\Delta x} = -\frac{\dot{q}}{\lambda}$ of the boundary condition (6.6), we conclude that $T^n_{M+1,k} = -\frac{2\Delta x \dot{q}}{\lambda} + T^n_{M-1,k}$ and hence, the equation for corresponding nodes reads:

$$\begin{aligned} \frac{H^{n+1}_{M,k} - H^n_{M,k}}{\Delta t} = \lambda \left(\frac{-2\frac{\Delta x \dot{q}}{\lambda} + 2T^n_{M-1,k} - 2T^n_{M,k}}{(\Delta y)^2} + \frac{T^n_{M,k+1} - 2T^n_{M,k} + T^n_{M,k-1}}{(\Delta z)^2} \right) + \\ + v_z \left(\frac{H^n_{M,k} - H^n_{M,k-1}}{\Delta z} \right) \quad \text{for} \quad \begin{cases} k = 0, \dots, N_C \\ n = 1, \dots, \tau - 1 \end{cases} , \end{aligned} \tag{6.15}$$

where N_C denotes the last node belonging to the crystallizer in axis z.

Finally, the discretization $\frac{T^n_{M+1,k}-T^n_{M-1,k}}{2\Delta x} = -\frac{(h_{tc,j}+h_{tc_r})(T_{\text{surface}}-T_\infty)}{\lambda}$ of boundary condition (6.7) leads to the equation

$$\frac{H^{n+1}_{M,k}-H^n_{M,k}}{\Delta t} = \lambda\left(\frac{-2\frac{\Delta x(h_{tc,j}+h_{tc_r})(T_{\text{surface}}-T_\infty)}{\lambda}+2T^n_{M-1,k}-2T^n_{M,k}}{(\Delta y)^2}+\right.$$
$$\left.+\frac{T^n_{M,k+1}-2T^n_{M,k}+T^n_{M,k-1}}{(\Delta z)^2}\right)+$$
$$+v_z\left(\frac{H^n_{M,k}-H^n_{M,k-1}}{\Delta z}\right) \qquad \text{for} \qquad \begin{cases} k=k(j)\in\hat{N}_j \\ n=1,\ldots,\tau-1 \end{cases}, \tag{6.16}$$

where $\hat{N}_j$ for $j=1,2,3,4$ are sets of nodes corresponding to the j-th cooling zone.

The empirical values of the temperature-enthalpy relationship (see Figure 6.5) were approximated by the following fifth-order Fourier series

$$T^n_{j,k} = A_0+\sum_{i=1}^{5}\left[A_i\sin(i\alpha H^n_{j,k})+B_i\cos(i\alpha H^n_{j,k})\right], \tag{6.17}$$

where coefficients A_0, A_1, B_1, …, A_5, B_5 and scale coefficient α were calculated by MATLAB software.

Moreover, the following additional constraints are added to the above model because of extra technological requests.

1. Upper bound for the cooling parameters $h_{tc,1}$, …, $h_{tc,4}$,

$$h_{tc,1}\le h_{tc,1,\max}, \quad h_{tc,2}\le h_{tc,2,\max}, \quad h_{tc,3}\le h_{tc,3,\max}, \quad h_{tc,4}\le h_{tc,4,\max}. \tag{6.18}$$

2. Mechanical and qualitative properties of produced steel are very sensitive to the temperature fluctuations in the casted product. To reduce those fluctuations and improve the temperature course, the following constraints to the temperatures at control positions 1–5 (see Figure 6.4) are added to the model:

$$\begin{aligned} T_{1,\min}&\le T^\tau_{M,k_1}\le T_{1,\max},\\ T_{2,\min}&\le T^\tau_{M,k_2}\le T_{2,\max},\\ T_{3,\min}&\le T^\tau_{M,k_3}\le T_{3,\max},\\ T_{4,\min}&\le T^\tau_{M,k_4}\le T_{4,\max},\\ T_{5,\min}&\le T^\tau_{M,k_5}\le T_{5,\max}, \end{aligned} \tag{6.19}$$

   where k_i corresponds to the closest node to the control position for temperature T_i in z axis (see Figure 6.4).

3. To ensure the output of the continuous casting machine in the cross-section to be solid, the following conditions

$$T^\tau_{0,k_{\min}}\ge T_{\text{liquid}}, \tag{6.20}$$
$$T^\tau_{0,k_{\max}}\le T_{\text{solid}} \tag{6.21}$$

specifying the so-called metalurgical length M^* (see Figure 6.4) are added to the model, where[1] $k_{\min} = \left\lfloor N\frac{M_{\min}}{L}\right\rceil$ and $k_{\max} = \left\lfloor N\frac{M_{\max}}{L}\right\rceil$, L is the total length of the continuous casting machine, $M_{\min}$ and $M_{\max}$ are the so-called minimal and maximal metalurgical length, respectively. The condition (6.21) guarantees that all steel in the cross-section of casted product is solidified.

Now, we can formulate the programme for the problem of the continuous casting of steel based on above discretized equations and constraints, where unknown variables to be optimized are the casting velocity v_z, the cooling parameters $h_{tc,1}, \ldots, h_{tc,4}$, the temperature $T^n_{j,k}$ and the enthalpy $H^n_{j,k}$ in all nodes in spatial axes and time axis:

$$\begin{array}{ll} \text{maximize} & v_z \\ \text{subject to} & \text{(6.8), (6.9), (6.10), (6.13), (6.14), (6.15),} \\ & \text{(6.16), (6.17), (6.18), (6.19), (6.20), (6.21).} \end{array} \tag{6.22}$$

6.3 Results for Continuous Casting Process

The programme (6.22) was solved for steel 11 325 by using the optimization software GAMS and its solver CONOPT for particular parameters given in Table 6.1. For detailed information about CONOPT solver and about the generalized reduced gradient method used in CONOPT, see Appendix A.

The initial values for the temperature $T^0_{j,k}$ and the enthalpy $H^0_{j,k}$ were calculated by MATLAB software by solving the equation (6.1) with particular fixed values of the cooling parameters and the constant initial temperature (1600 °C) and the initial enthalpy ($1.066 \cdot 10^{10}\,\mathrm{J \cdot m^{-3}}$).

The programme (6.22) for the setting in Table 6.1 includes over 72 000 variables to be optimized and the computation requires 34 minutes and 45 seconds with 1 255 iterations of the CONOPT solver on the laptop with AMD Turion X2 Ultra Dual-Core 2.2 GHz and 2.4 GB of RAM memory. Its optimal solution is given in Table 6.2, the temperature field for $n = 100$ is shown in Figure 6.8 and the temperature in the core (blue colour) and on the surface of a casted billet (red colour) are shown in Figure 6.9.

6.4 Two-Stage Stochastic Programme for Continuous Casting Process

In the foregoing sections, we described the method for steelmaking – the continuous casting process – and derived its mathematical model. Further, we calculated its optimal solution by using the GAMS software for parameters given in Table 6.1. In this section, we will include the uncertainty to the model via scenario approach and we will deal with the following problem.

Assume that during the continuous casting process the total breakdown in the second cooling zone can occur with the probability p_2. This error means that the parameter $h_{tc,2}$ becomes to zero. Hence, we can model the situation by two scenarios s^1 and s^2. The scenario s^1 will describe the normal situation without any error and we denote its

[1] $\lfloor x \rceil$ denotes the nearest integer to x

Table 6.1: Parameters for programme (6.22)

Parameter	Value	Parameter	Value
τ	100	$h_{tc,2,\max}$	$400\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
N	36	$h_{tc,3,\max}$	$400\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
M	10	$h_{tc,4,\max}$	$300\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
λ	$33\,\mathrm{W}\cdot\mathrm{m}^{-1}\cdot\mathrm{K}^{-1}$	$T_{1,\min}$	1300 °C
L	20 m	$T_{1,\max}$	1550 °C
H	0.125 m	$T_{2,\min}$	1050 °C
T_∞	20 °C	$T_{2,\max}$	1250 °C
h_{tc_r}	$4.51\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$T_{3,\min}$	900 °C
$\dot{q}$	$-2.97\cdot10^3\,\mathrm{W}\cdot\mathrm{m}^{-2}$	$T_{3,\max}$	1000 °C
N_1	2	$T_{4,\min}$	700 °C
N_2	10	$T_{4,\max}$	900 °C
N_3	18	$T_{5,\min}$	700 °C
N_4	26	$T_{5,\max}$	800 °C
N_5	35	T_{liquid}	1511 °C
$h_{tc,1,\max}$	$500\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	T_{solid}	1490 °C

Table 6.2: Optimal solution for programme (6.22) with parameters given in Table 6.1

Parameter	Optimal value
Casting velocity v_z	$2.2158\,\mathrm{m}\cdot\mathrm{min}^{-1}$
Cooling parameter $h_{tc,1}$ for zone 1	$476.24\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
Cooling parameter $h_{tc,2}$ for zone 2	$249.54\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
Cooling parameter $h_{tc,3}$ for zone 3	$325.29\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
Cooling parameter $h_{tc,4}$ for zone 4	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$

probability p_1. On the other hand, the scenario s^2 will describe the situation with the total breakdown in the second cooling zone and we denote its probability p_2.

Thus, we have the two-stage stochastic programme with two scenarios. The questions to be answered are following: How we should set the cooling parameters $h_{tc,1}, \ldots, h_{tc,4}$ at the beginning of the casting process (the first-stage decision) to maximize the casting velocity, but regarding to both scenarios and to the possible breakdown in the second cooling zone? And in case the breakdown occurs, how we should change the cooling parameters $h_{tc,1}$, $h_{tc,3}$ and $h_{tc,4}$ (the second-stage decision), whereas $h_{tc,2} = 0$? Further, we will be also interested in the possible gain that may be reached by solving the stochastic programme. Thus, what is the value of stochastic solution, VSS?

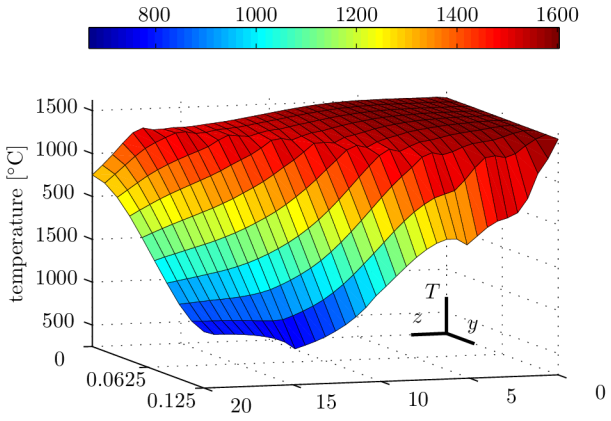

Figure 6.8: Temperature field for optimal values and $n = 100$

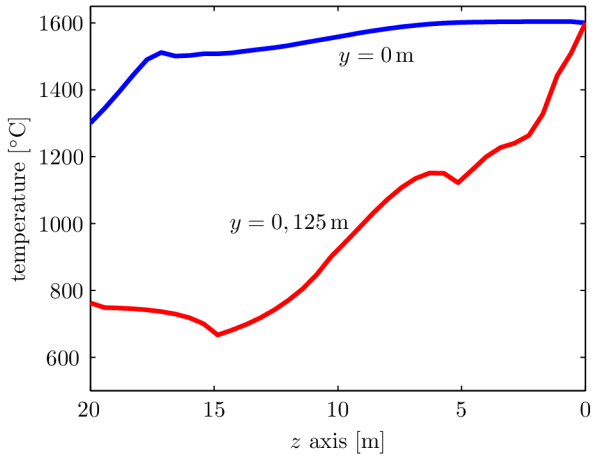

Figure 6.9: Temperature in the core (blue colour) and on the surface (red colour) of casted billet for optimal values and $n = 100$

To include the scenario approach to the model, the programme (6.22) must be modified to the two-stage stochastic programme by replacing the cooling parameters $h_{tc,1}, \ldots, h_{tc,4}$ by the cooling parameters $h_{tc,1,x}, \ldots, h_{tc,4,x}$ and $h_{tc,1,y}, \ldots, h_{tc,4,y}$. The nonnegative variables $h_{tc,\cdot,x}$ correspond to the first-stage decision satisfying the modified equation (6.18),

$$h_{tc,1,x} \leq h_{tc,1,\max}, \quad h_{tc,2,x} \leq h_{tc,2,\max}, \quad h_{tc,3,x} \leq h_{tc,3,\max}, \quad h_{tc,4,x} \leq h_{tc,4,\max}. \tag{6.23}$$

On the other hand, the variables $h_{tc,\cdot,y}$ corresponing to the second-stage decision denote the "correction" of the first-stage decision and may be also negative, i.e., the cooling parameters that should be adjusted in the second-stage are $h_{tc,\cdot,x} + h_{tc,\cdot,y}$ satisfying

$$h_{tc,\cdot,x} + h_{tc,\cdot,y} \leq h_{tc,\cdot,\max}. \tag{6.24}$$

Obviously, for the scenario s^1 we require

$$h_{tc,1,y} = h_{tc,2,y} = h_{tc,3,y} = h_{tc,4,y} = 0$$

since the control of the casting machine wish to change the setting of the cooling parameters only when the breakdown occurs. For the scenario s^2, to guarantee the breakdown in the second cooling zone, it must hold $h_{tc,2,x} + h_{tc,2,y} = 0$.

Denote the time when the breakdown in the second cooling zone occurs by τ_B. Then, the equation (6.16) is modified to the form

$$\begin{aligned} \frac{H^{n+1}_{M,k} - H^n_{M,k}}{\Delta t} = \lambda \Bigg(& \frac{-2\frac{\Delta x (h_{tc,j,x} + h_{tc_r})(T_{\text{surface}} - T_\infty)}{\lambda} + 2T^n_{M-1,k} - 2T^n_{M,k}}{(\Delta y)^2} + \\ & + \frac{T^n_{M,k+1} - 2T^n_{M,k} + T^n_{M,k-1}}{(\Delta z)^2} \Bigg) + \\ & + v_z \left(\frac{H^n_{M,k} - H^n_{M,k-1}}{\Delta z} \right) \qquad \text{for} \qquad \begin{cases} k = k(j) \in \hat{N}_j \\ n = 1, \ldots, \tau_B \end{cases} \end{aligned} \tag{6.25}$$

and

$$\frac{H_{M,k}^{n+1}-H_{M,k}^{n}}{\Delta t}=\lambda\left(\frac{-2\frac{\Delta x(h_{tc,j,x}+h_{tc,j,y}+h_{tcr})(T_{\text{surface}}-T_{\infty})}{\lambda}+2T_{M-1,k}^{n}-2T_{M,k}^{n}}{(\Delta y)^2}+\right.$$
$$\left.+\frac{T_{M,k+1}^{n}-2T_{M,k}^{n}+T_{M,k-1}^{n}}{(\Delta z)^2}\right)+$$
$$+v_z\left(\frac{H_{M,k}^{n}-H_{M,k-1}^{n}}{\Delta z}\right)\qquad\text{for}\qquad\left\{\begin{array}{l}k=k(j)\in\hat{N}_j\\ n=\tau_B+1,\ldots,\tau-1\end{array}\right.,\quad(6.26)$$

where $\hat{N}_j$ for $j=1,2,3,4$ are sets of nodes corresponding to the j-th cooling zone. In particular, for the parameters given in Table 6.1, the sets $\hat{N}_j$ are $\hat{N}_1=\{N_1+1,\ldots,N_2\}=$ $=\{3,4,5,\ldots,10\}$, $\hat{N}_2=\{N_2+1,\ldots,N_3\}=\{11,12,13,\ldots,18\}$, $\hat{N}_3=\{N_3+1,\ldots,N_4\}=$ $=\{19,20,21,\ldots,26\}$ and $\hat{N}_4=\{N_4+1,\ldots,N_5\}=\{27,28,29\ldots,35\}$.

6.5 Results for Two-Stage Continuous Casting Process

For the above two-stage stochastic programme modelling the continuous casting process with two scenarios described in the foregoing sections, the parallel implementation of the two-stage progressive hedging algorithm, in detailed discussed in Chapter 4 and especially in Chapter 5, was used.

The algorithm was tested on the laptop ASUS X71-TL with the operating system Linux Ubuntu, 32 bits with dual-core CPU AMD Turion X2 Ultra Dual-Core ZM-82 with the frequency 2,2 GHz and 2,4 GB of RAM memory. Due to the dual-core CPU hardware, the two-scenario programme could be realized very effectively in parallel – each core computes one scenario problem in each iteration of the progressive hedging algorithm.

The parameters for the two-stage stochastic programme coincide with parameters given in Table 6.1 with $\tau_D=5$, $p_1=0.95$ and $p_2=0.05$.

The illustration of running implementation of the progressive hedging algorithm is shown in Figure 6.10: the upper left window is the system monitor application, the upper right window is the main window of the application where all results and information about each iteration are displayed. The both lower window are running GAMS applications, left for scenario s^1 and right for scenario s^2. The main window is shown in detail in Figure 6.11.

The optimal solutions for both scenarios s^1 and s^2 are presented in Table 6.3, the temperature field and the temperatures on the surface and in the core of casted product for scenario s^1 and s^2 are for $n=100$ pictured in Figures 6.12, 6.13 and 6.14, 6.15, respectively. Note that Figures 6.12 and 6.13 coincide with Figures 6.8 and 6.9, respectively.

To decrease the number of iterations of the progressive hedging algorithm needed to satisfy the given tolerance, the intial value of $\hat{\boldsymbol{x}}_1$, i.e., the value $\hat{\boldsymbol{x}}_1^0$, was set to the weighted average of both first-stage scenario solutions,

$$\hat{\boldsymbol{x}}_1^0=\left(\hat{h}_{tc,1,x}^0,\hat{h}_{tc,1,x}^0,\hat{h}_{tc,1,x}^0,\hat{h}_{tc,1,x}^0\right)^T=$$
$$=\left(\sum_{j=1}^{2}p_j\cdot h_{tc,1,x}^j,\ldots,\sum_{j=1}^{2}p_j\cdot h_{tc,4,x}^j\right)^T=(477.41,257.06,309.08,0)^T.\quad(6.27)$$

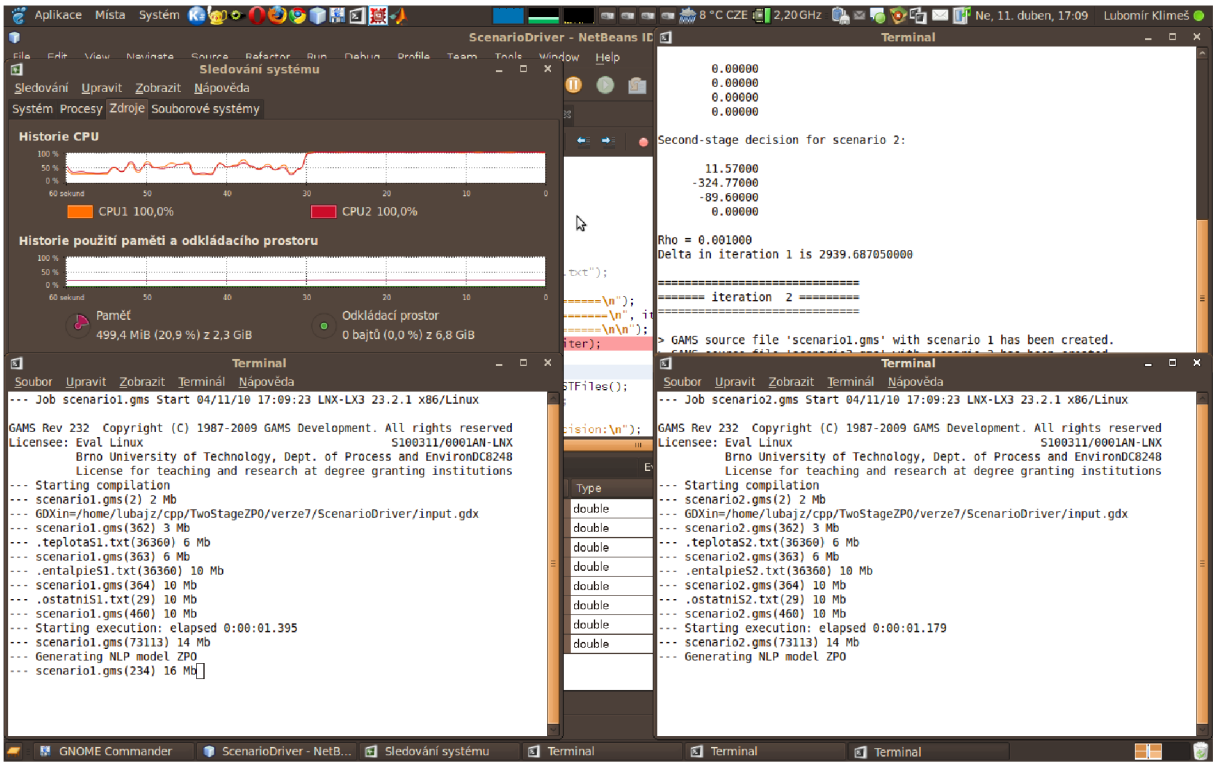

Figure 6.10: The illustration of running programme

Table 6.3: Optimal solution for scenarios s^1 and s^2

Scenario s^1		Scenario s^2	
Parameter	Optimal value	Parameter	Optimal value
v_z	$2.2158\,\mathrm{m}\cdot\mathrm{min}^{-1}$	v_z	$1.8396\,\mathrm{m}\cdot\mathrm{min}^{-1}$
$h^1_{tc,1,x}$	$476.24\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,1,x}$	$500.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
$h^1_{tc,2,x}$	$249.54\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,2,x}$	$400.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
$h^1_{tc,3,x}$	$325.29\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,3,x}$	$1.17\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
$h^1_{tc,4,x}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,4,x}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
$h^1_{tc,1,y}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,1,y}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
$h^1_{tc,2,y}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,2,y}$	$-400.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
$h^1_{tc,3,y}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,3,y}$	$66.89\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$
$h^1_{tc,4,y}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$	$h^2_{tc,4,y}$	$0.00\,\mathrm{W}\cdot\mathrm{m}^{-2}\cdot\mathrm{K}^{-1}$

As we already mentioned, for the continuous casting problem with two scenarios, the parallel implementation of the progressive hedging algorithm, in detail discussed in Chapters 4 and 5, was used with the penalty parameter $\varrho = 0.001$ and the tolerance $\varepsilon = 10^{-9}$. Due to the above choice of the initial setting $\hat{\boldsymbol{x}}^0$ and the dominance of the first scenario with probability $p_1 = 0.95$, the progressive hedging algorithm required only 4 iterations to satisfy the termination criteria. But without the initial setting (6.27) of $\hat{\boldsymbol{x}}^0$,

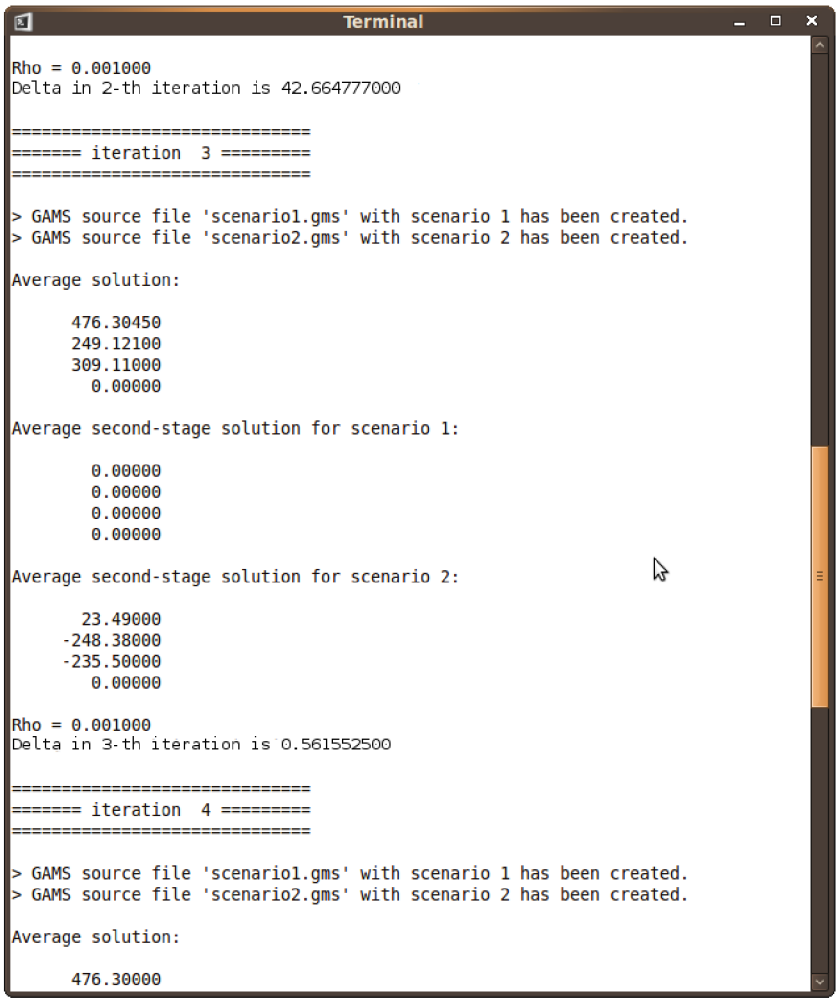

Figure 6.11: The illustration of the main window

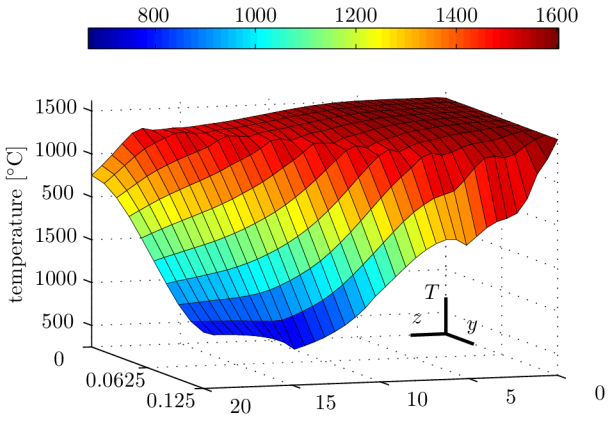

Figure 6.12: Temperature field for scenario s^1 and $n = 100$

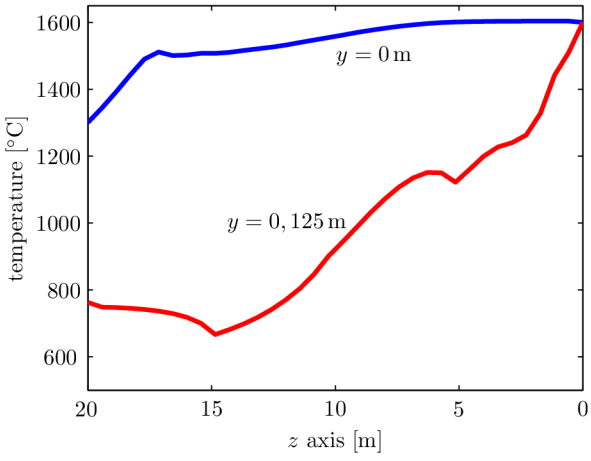

Figure 6.13: Temperature in the core (blue colour) and on the surface (red colour) of casted billet for scenario s^1 and $n = 100$

the number of iterations exceeded the value 50 without the fulfillment of the tolerance criteria and the experiment was stopped due to the huge time-consuming.

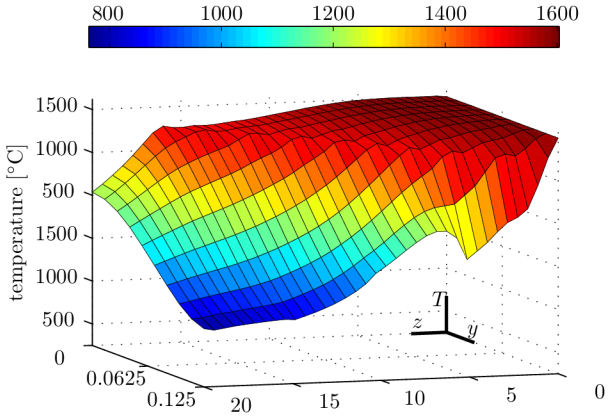

Figure 6.14: Temperature field for scenario s^2 and $n = 100$

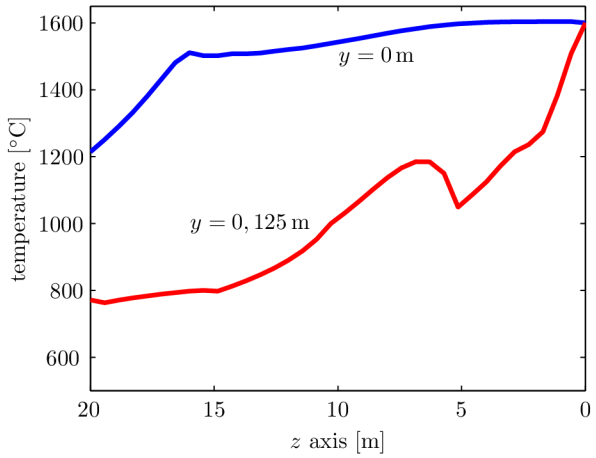

Figure 6.15: Temperature in the core (blue colour) and on the surface (red colour) of casted billet for scenario s^2 and $n = 100$

The results are given in Table 6.4. Hence, the optimal initial setting of the cooling parameters is

$$h_{tc,1} = 476.32, \qquad h_{tc,2} = 249.11, \qquad h_{tc,3} = 309.11, \qquad h_{tc,4} = 0.00.$$

If no breakdown occurs, no change in the setting of the cooling parameters is done. On the other hand, in case the breakdown in the second cooling zone occurs, the control service of the continuous casting machine should change the setting of the cooling parameters as follows,

$$h_{tc,1} = 476.32 + 23.68 = 500.00, \qquad h_{tc,2} = 249.11 - 249.11 = 0,$$
$$h_{tc,3} = 309.11 - 240.97 = 68.14, \qquad h_{tc,4} = 0.00.$$

The temperature fields and temperatures on the surface and in the core of casted product for optimal values obtained by using the progressive hedging algorithm for both scenarios s^1 and s^2 are shown in Figures 6.16, 6.17, 6.18 and 6.19, respectively.

The difference between temperature fields obtained by solving the particular scenario programme (Figure 6.12 and 6.14) and by using the progressive hedging algorithm (Figures 6.16 and 6.18) are shown for both scenarios s^1 and s^2 in Figures 6.20 and 6.21, respectively.

Let us now discuss about the solution obtained by using the progressive hedging algorithm, compare it with the EV approach via VSS characteristic and with case of the perfect future information via EVPI characteristic.

As we already mentioned in Chapter 3, the value of stochastic solution, VSS, represents the possible profit obtained by solving the EO programme instead of simpler EV programme. We also discussed in Chapter 5 that the progressive hedging algorithm gives the desired EO solution to the EO programme. Thus, for the continuous casting process, the first-stage EO decision obtained by using the progressive hedging algorithm reads

$$\boldsymbol{x}_1^{\mathrm{EO}} = (476.32, 249.11, 309.11, 0.00)^T.$$

The second-stage EO decision corresponding to the scenario s^1 reads

$$\boldsymbol{x}_2^{\mathrm{EO},1} = (0.00, 0.00, 0.00, 0.00)^T$$

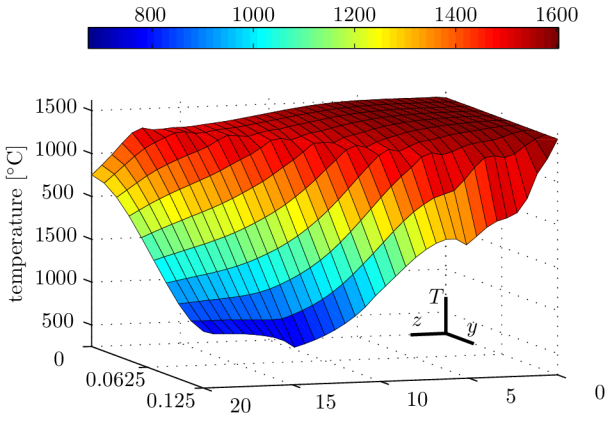

Figure 6.16: Temperature field for scenario s^1 and $n = 100$ for values obtained by using PHA

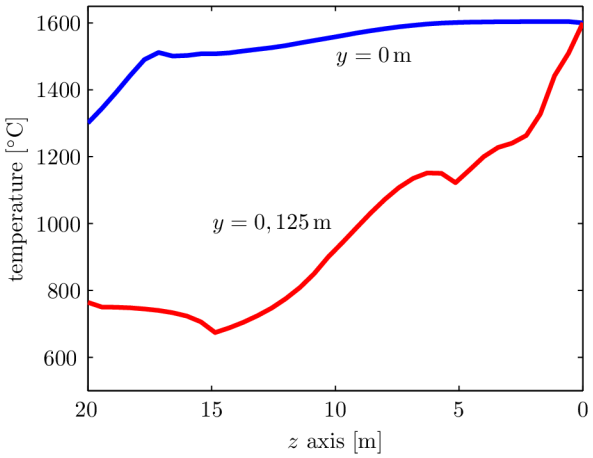

Figure 6.17: Temperature in the core (blue colour) and on the surface (red colour) of casted billet for scenario s^1 and $n = 100$ for values obtained by using PHA

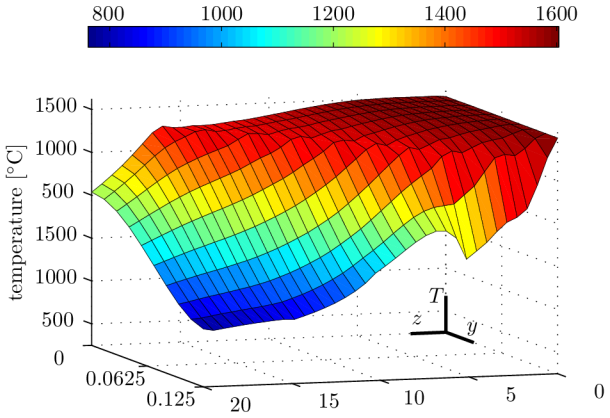

Figure 6.18: Temperature field for scenario s^2 and $n = 100$ for values obtained by using PHA

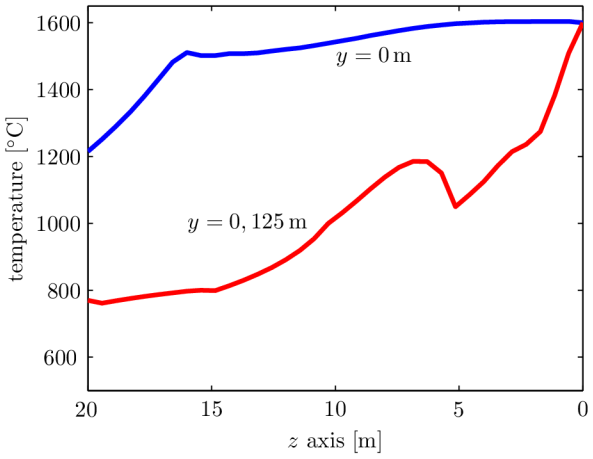

Figure 6.19: Temperature in the core (blue colour) and on the surface (red colour) of casted billet for scenario s^2 and $n = 100$ for values obtained by using PHA

with the casting velocity $v_z^{\mathrm{EO},1} = 2.215633\,\mathrm{m}\cdot\mathrm{min}^{-1}$, the second-stage EO decision for the scenario s^2 is

$$\boldsymbol{x}_2^{\mathrm{EO},2} = (23.68, -249.11, -240.97, 0.00)^T$$

with the casting velocity $v_z^{\mathrm{EO},2} = 1.836583\,\mathrm{m}\cdot\mathrm{min}^{-1}$. Thus,

$$z^{\mathrm{EO}} = \sum_{j=1}^{2} p_j \cdot v_z^{\mathrm{EO},j} = 2.196681\,\mathrm{m}\cdot\mathrm{min}^{-1}.$$

By solving the EV programme (3.10), one obtains the following results: the first-stage EV decision reads

$$\boldsymbol{x}_1^{\mathrm{EV}} = (451.24, 229.54, 305.29, 0.00)^T,$$

the second-stage EV decision for scenario s^1 is

$$\boldsymbol{x}_2^{\mathrm{EV},1} = (0.00, 0.00, 0.00, 0.00)$$

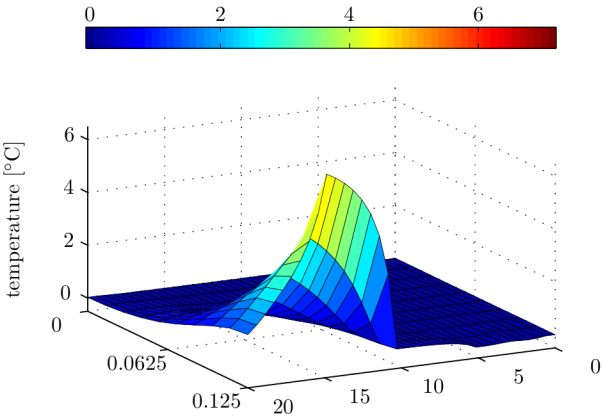

Figure 6.20: Difference of temperature fields for scenario s^1 and $n = 100$

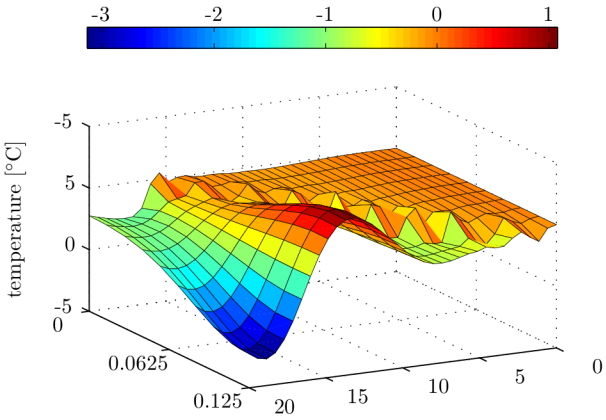

Figure 6.21: Difference of temperature fields for scenario s^2 and $n = 100$

Table 6.4: The results for the continuous casting process with $\varrho = 0.001$, $\varepsilon = 10^{-9}$

Variable			Iteration				
			0	1	2	3	4
$\hat{\boldsymbol{x}}_1$	$h_{tc,1,x}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	477.41	476.42	476.32	476.32	476.32
	$h_{tc,2,x}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	257.06	249.85	249.11	249.11	249.11
	$h_{tc,2,x}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	309.08	309.11	309.11	309.11	309.11
	$h_{tc,3,x}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	0.00	0.00	0.00	0.00	0.00
$\hat{\boldsymbol{x}}_2^1$	$h^1_{tc,1,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	0.00	0.00	0.00	0.00	0.00
	$h^1_{tc,2,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	0.00	0.00	0.00	0.00	0.00
	$h^1_{tc,3,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	0.00	0.00	0.00	0.00	0.00
	$h^1_{tc,4,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	0.00	0.00	0.00	0.00	0.00
$\hat{\boldsymbol{x}}_2^2$	$h^2_{tc,1,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	0.00	22.57	24.58	23.78	23.68
	$h^2_{tc,2,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	−400	−257.06	−242.64	−248.37	−249.11
	$h^2_{tc,3,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	72.44	−241.05	−240.90	−240.96	−240.97
	$h^2_{tc,4,y}$	$[\mathrm{W \cdot m^{-2} \cdot K^{-1}}]$	0.00	0.00	0.00	0.00	0.00
δ			116 070	53.15	42.79	0.56	$< 10^{-9}$

with the casting velocity $v_z^{\mathrm{EV},1} = 2.188541\,\mathrm{m \cdot min^{-1}}$, the second-stage EV decision for the scenario s^2 reads

$$\boldsymbol{x}_2^{\mathrm{EV},2} = (48.76, -229.54, -236.92, 0.00)^T$$

with the corresponding casting velocity $v_z^{\mathrm{EV},1} = 1.835535\,\mathrm{m \cdot min^{-1}}$. Hence,

$$\mathrm{EEV} = \sum_{j=1}^{2} p_j \cdot v_z^{\mathrm{EV},j} = 2.170891\,\mathrm{m \cdot min^{-1}}.$$

Finally, the value of stochastic solution (see Definition 3.3) for the maximization problem is given by

$$\mathrm{VSS} = z^{\mathrm{EO}} - \mathrm{EEV} = 0.025790\,\mathrm{m}\cdot\mathrm{min}^{-1}.$$

This value represents the gain in the productivity by solving the EO programme.

To calculate the expected value of perfect information, EVPI, one has to find z^{WS}. For the continuous casting process, one obtains by solving WS programme for scenario s^1 the first-stage decision

$$\boldsymbol{x}_1^{\mathrm{WS},1} = (476.24, 249.54, 325.29, 0.00)^T$$

and the second-stage decision $\boldsymbol{x}_2^{\mathrm{WS},1} = (0.00, 0.00, 0.00, 0.00)^T$ with the casting velocity $v_z^{\mathrm{WS},1} = 2.215812\,\mathrm{m}\cdot\mathrm{min}^{-1}$. For the scenario s^2, one gets the first-stage decision

$$\boldsymbol{x}_1^{\mathrm{WS},2} = (500.00, 400.00, 1.17, 0.00)^T$$

and the second-stage decision $\boldsymbol{x}_2^{\mathrm{WS},2} = (0.00, -400.00, 66.89, 0.00)^T$ with the corresponding casting velocity $v_z^{\mathrm{WS},2} = 1.839574\,\mathrm{m}\cdot\mathrm{min}^{-1}$. Hence,

$$z^{\mathrm{WS}} = \sum_{j=1}^{2} p_j \cdot z^{\mathrm{WS},j} = 2.197000\,\mathrm{m}\cdot\mathrm{min}^{-1}.$$

Then, the expected value of perfect information (see Definition 3.4), EVPI, for the maximization problem is given by

$$\mathrm{EVPI} = z^{\mathrm{WS}} - z^{\mathrm{EO}} = 0.000319\,\mathrm{m}\cdot\mathrm{min}^{-1}.$$

This value represents the maximum expressed in the casting velocity which the decision maker should be ready to pay to obtain the perfect information about future.

CHAPTER 7

Conclusion

This master's thesis deals with stochastic programming and optimization. The described theory and tools of mathematical programming can be fruitfully utilized in many practical applications in which the uncertainty and the randomness play a significant role. Also the implementations of the progressive hedging algorithm presented in this thesis can be easily used for wide variety of optimization problems.

In the theoretical part, in Chapter 3, the basis of stochastic programming, deterministic equivalents, two-stage, multi-stage and scenario-based programmes are presented. In Chapter 4, the progressive hedging algorithm – the method for solving scenario-based stochastic programmes – is introduced for the one-stage and also for the muti-stage case.

In the practical part, in Chapter 5, the original parallel implementation of the progressive hedging algorithm is discussed in details for the one-stage and two-stage case. The parallel implementations are realized by using C++ programming language, the message passing interface for parallel computing, GAMS software for solving scenario-based subproblems and the operating system Linux Ubuntu. There are also described two illustrative problems that are solved by using the realized parallel implementations. These problems were chosen and designed to help the reader to understand fundamental principles and behaviour of the progressive hedging algorithm. In Chapter 6, the two-stage scenario-based model for the continuous casting process of steel slab is derived and solved by using the two-stage parallel implementation of the progressive hedging algorithm from Chapter 5. All results are visualized and the used approach of stochastic programming is reviewed by qualitative characteristics.

In three appendixes, the reduced gradient and generalized reduced gradient methods, the optimality conditions and the augmented Lagrangian penalty functions are presented.

The author of this thesis will be pleased if this thesis will be helpful for engineers interested in applications of stochastic programming, scenario-based programmes, the progressive hedging algorithm in general and also its parallel implementation or in the continuous casting process of steel slabs.

The further intention of the author of this master's thesis is to deal with the continuous casting process of steel slabs, with the process of heat transfer and with stochastic optimization in Ph.D. study programme in Energy Institute, Department of Thermodynamics

and Environmental Engineering at Faculty of Mechanical Engineering, Brno University of Technology.

Bibliography

[1] M. S. Bazaraa, H. D. Sherali, and C. M. Sheety. *Nonlinear Programming: Theory and Algorithms*. John Wiley & Sons, New York, second edition, 1993. ISBN 0-471-55793-5.

[2] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research. Springer Verlag, New York, 1997. ISBN 0-387-98217-5.

[3] A. Brooke, D. Kendrick, A. Meeraus, and R. Raman. *GAMS — A User's Guide*. GAMS Development Corporation, Washington, DC, USA, 2008.

[4] A. de Silva and D. Abramson. Computational experience with the parallel progressive hedging algorithm for stochastic linear programs, 1994.

[5] Arne Drud. *CONOPT*. ARKI Consulting and Development A/S, Bagsvaerd, Denmark. Available at `http://www.gams.com/dd/docs/solvers/conopt.pdf`.

[6] J. Dupačová, J. Hurt, and J. Štěpán. *Stochastic Modeling in Economics and Finance*, chapter P. Popela: Numerical Techniques and Available Software, pages 206–227. Kluwer Academic Publishers, 2002. ISBN 1-4020-0840-6.

[7] V. Dvořák. *Architektura a programování paralelních systémů*. VUT Brno, nakladatelství VUTIUM, first edition, 2004. ISBN 80-214-2608-X.

[8] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, third edition, 1968. ISBN 978-0471257080.

[9] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley, second edition, 1991. ISBN 978-0471257097.

[10] M. Goossens, F. Mittelbach, S. Rahtz, D. Roegel, and H. Voß. *The LaTeX Graphics Companion*. Addison-Wesley, second edition, 2008. ISBN 0-321-50892-0.

[11] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, second edition, 1999. ISBN 978-0262571326.

[12] P. Kall and S. W. Wallace. *Stochastic Programming*. John Wiley & Sons, Inc., Chichester, second edition, 1994.

[13] T. Mauder. Optimization methods for the secondary cooling zone of a continuous casting process of steel slabs. *Strojárstvo/Strojírenství*, 2009:175–176, 2009. ISSN 1335-2938.

[14] T. Mauder. Teze k dizertační práci, 2010. Vysoké učení technické v Brně, Fakulta strojního inženýrství.

[15] T. Mauder, F. Kavička, J. Štětina, Z. Franěk, and M. Masarik. A mathematical & stochastic modelling of the concasting of steel slabs. In *18th International conference on metallurgy and materials, Hradec nad Moravicí*, pages 41–48. Tanger, s.r.o., 2009. ISBN 978-80-87294-10-9.

[16] F. Mittelbach, M. Goossens, J. Braams, D. Carlisle, and Ch. Rowley. *The LaTeX Companion*. Addison-Wesley, second edition, 2004. ISBN 0-201-36299-6.

[17] P. Popela. *An Objected-Oriented Approach to Multistage Stochastic Programming*. PhD thesis, Charles University in Prague, 1998.

[18] S. Prata. *Mistrovství v C++*. Computer Press, Praha, first edition, 2001. ISBN 80-7226-339-0.

[19] R. T. Rockafellar and R. J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. In *Mathematics of Operation Research*, volume 16, pages 119–147. INFORMS, Linthicium, Maryland, 1991.

[20] R. E. Rosenthal. *A GAMS Tutorial*. Naval Postgraduate School, Monterey, California USA.

[21] R. Stones and N. Matthew. *Linux: Začínáme programovat*. Computer Press, Praha, 2000. ISBN 80-7226-307-2.

[22] J.-P. Watson, D. L. Woodruff, and D. R. Strip. Progressive hedging innovations for a class of stochastic resource allocation problems. Technical Report 2007-3722J 2007-3722J, Sandia National Laboratories, 2008.

[23] R. J.-B. Wets. An aggregation principle in scenario analysis and stochastic optimization. In S. W. Wallace, editor, *Algorithms and Model Formulations in Mathematical Programming*. Springer Verlag, New York, 1989.

[24] Wikipedia. Continuous Casting.
Available at `http://en.wikipedia.org/wiki/Continuous_casting`.
[Online; cited March 28, 2010].

[25] Wikipedia. General Algebraic Modelling System.
Available at `http://en.wikipedia.org/wiki/General_Algebraic_Modeling_System`.
[Online; cited April 4, 2010].

[26] Wikipedia. Message Passing Interface.
Available at `http://en.wikipedia.org/wiki/Message_Passing_Interface`.
[Online; cited April 15, 2010].

[27] Wikipedia. Multivariate Random Variable.
Available at http://en.wikipedia.org/wiki/Multivariate_random_variable.
[Online; cited May 16, 2010].

[28] J. Štětina. *Dynamický model teplotního pole plynule odlévané bramy.* PhD thesis, Vysoká škola báňská – Technická univerzita Ostrava, 2007.

[29] E. Žampachová. Approximate solution of PDE constrained stochastic optimization problems. In *FME Junior conference*, pages 16–23. Brno, 2009.

[30] E. Žampachová. *Approximations in stochastic optimization and their applications.* PhD thesis, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010.

[31] E. Žampachová, P. Popela, and M. Mrázek. Optimum beam design via stochastic programming. In *Kybernetika*, volume 46, pages 575–586, 2010.

Used Symbols and Abbreviations

$\mathbb{R}^N$, $\mathbb{E}^N$	N-dimensional real space, N-dimensional Euclidean space
$\lvert\boldsymbol{u}\rvert$	number of elements (dimension) of vector $\boldsymbol{u}$
$\boldsymbol{x}$, $\boldsymbol{x}_1$	first-stage dicision
$\boldsymbol{y}$, $\boldsymbol{x}_2$	second-stage decision
$\boldsymbol{x}_j$	j-th-stage decision
$\boldsymbol{w}_j$	weights corresponding to a j-th-stage decision
ξ, $\boldsymbol{\xi}$	random variable, random vector
$\mathbb{E}_{\boldsymbol{\xi}}(\cdot)$	expected value with respect to $\boldsymbol{\xi}$
WS	wait-and-see
IS	individual scenario
MM	"fat solution"
EV	expected value
EO	expected objective
EEV	expected result of using EV solution
VSS	value of stochastic solution
EVPI	expected value of perfect information
$\boldsymbol{X}(\cdot)$	policy, $\boldsymbol{X}(\cdot) = \big(\boldsymbol{x}_1(\cdot), \ldots, \boldsymbol{x}_t(\cdot)\big)$
$\boldsymbol{W}(\cdot)$	weights, $\boldsymbol{W}(\cdot) = \big(\boldsymbol{w}_1(\cdot), \ldots, \boldsymbol{w}_t(\cdot)\big)$
$\mathscr{P}_t$	partition at time t

A_k	set in partition $\mathscr{P}_t$
$\mathscr{N}$	set of implementable policies
$\mathscr{C}$	set of admissible policies
API	Application Programming Interface
MPI	Message Passing Interface
PHA	progressive hedging algorithm
GAMS	General Algebraic Modelling System
$\lceil x \rceil$	smallest integer greater than x
$\lfloor x \rceil$	nearest integer to x
ϱ	penalty parameter
ε	tolerance parameter
δ	distance parameter
T	temperature
H	enthaply
v_z	casting velocity
λ	heat conductivity
$\dot{q}$	heat flow
T_{casting}	casting temperature
T_∞	temperature of water flowing to cooling zones
$h_{tc,\cdot}$, h_{tc_r}	heat transfer coefficient, reduced heat transfer coefficient
Δy, Δz, Δt	discretization steps
T_{solid}, T_{liquid}	solidus temperature, liquidus temperature
NLP	non-linear programming/programme
DNLP	non-linear programming/programme with discontinuous derivatives
RG	reduced gradient
GRG	generalized reduced gradient
FJ	Fritz John
KKT	Karush-Kuhn-Tucker
ALAG	augmented Lagrangian

APPENDIX **A**

Solver CONOPT

In this thesis, the general algebraic modelling system GAMS is used to solve each scenario-based subproblem in the implementation of the progressive hedging algorithm. Since these scenario-based subproblems require to solve linar-quadratic perturbed versions of original (in general nonlinear) problem, the solver CONOPT for nonlinear problems is used.

The solver CONOPT is designed for NLP and DNLP models. NLP models are nonlinear models in which all functions (objective function, constraint functions, ...) with their derivatives are smooth[1]. On the other hand, DNLP models are such models in which all functions are smooth, but their derivatives can be discontinuous. For instance, the functions min, max or absolute value can be used in DNLP models, but not in NLP models.

The CONOPT solver and its algorithm is based on the Generalized Reduced Gradient (GRG) method for solving nonlinear problems with nonlinear constraints. The GRG method is the generalization of the Reduced Gradient (RG) method which is an algorithm for solving nonlinear problems with linear constraints. The both methods belong to the class of methods of feasible directions for which the following principle is typical. The reader can find detailed information about CONOPT implementation in GAMS, its settings and algorithmic details in [5].

Let $\min\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in X\}$ be a problem to be solved and assume $\boldsymbol{x}_k$ to be a feasible point. The algorithms based on the method of feasible directions proceed as follows: a direction $\boldsymbol{d}_k$ is determined and afterwards, one-dimensional optimization problem $\min\{\boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k\}$ is solved with respect to $\lambda_k > 0$ so that the new point $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k$ satisfies two conditions: $\boldsymbol{x}_{k+1}$ is feasible and the objective value at $\boldsymbol{x}_{k+1}$ is better than at $\boldsymbol{x}_k$, i.e., satisfying $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$. This leads to a new point $\boldsymbol{x}_{k+1}$ and the above process is repeated. The question is, of course, how to determine the direction $\boldsymbol{d}_k$.

[1] the function f is said to be *smooth* if f is continuous and has continuous derivatives of all orders

A.1 Reduced Gradient Method

The Reduced Gradient method is an algorithm for generating improving feasible directions. This algorithm for solving nonlinear problems with linear constraints was developed by P. Wolfe in 1963. The main idea is to reduce the dimensionality of the original problem by using an independent subset of variables.

Consider the following nonlinear problem with linear constraints:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ & && \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{A}$ is an matrix of type $m \times n$ with rank m, $\boldsymbol{b}$ is an m vector and function f is a continuously differentiable. Further, it is assumed that any m columns of matrix $\boldsymbol{A}$ are linearly independent and every extreme point[2] of the feasible region has m strictly positive components. This so-called nondegeneracy assumption guarentees that each feasible point has at least m positive components and, at most, $n - m$ components equal to zero.

Due to the nondegeneracy assumption, the matrix $\boldsymbol{A}$ can be decomposed into $(\boldsymbol{B}, \boldsymbol{N})$ and a feasible point $\boldsymbol{x}^T$ can be decomposed into basic and nonbasic vectors, $(\boldsymbol{x}_B^T, \boldsymbol{x}_N^T)$. Note that $\boldsymbol{B}$ is invertible matrix of type $m \times m$ and $\boldsymbol{x}_B > 0$.

A direction $\boldsymbol{d}$ to be an improving feasible direction of function f at point $\boldsymbol{x}$ has to satisfy two conditions:

$$\boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{d} < 0, \tag{A.1}$$

$$\boldsymbol{A}\boldsymbol{d} = \boldsymbol{0} \text{ with } d_j \geq 0 \text{ for } x_j = 0. \tag{A.2}$$

Hence, our goal is to determine a direction $\boldsymbol{d}$ satisfying above two conditions. Assume that the direction $\boldsymbol{d}$ is decomposed into two parts, $\boldsymbol{d}^T = (\boldsymbol{d}_B^T, \boldsymbol{d}_N^T)$. Let $\boldsymbol{d}_B = -\boldsymbol{B}^{-1}\boldsymbol{N}\boldsymbol{d}_N$ and observe that the condition $\boldsymbol{A}\boldsymbol{d} = \boldsymbol{B}\boldsymbol{d}_B + \boldsymbol{N}\boldsymbol{d}_N = \boldsymbol{0}$ holds for any vector $\boldsymbol{d}_N$ thereby (A.2) is satisfied. Define the *reduced gradient* $\boldsymbol{r}$ as follows,

$$\begin{aligned} \boldsymbol{r}^T = \left(\boldsymbol{r}_B^T, \boldsymbol{r}_N^T\right) = \boldsymbol{\nabla} f(\boldsymbol{x})^T - \boldsymbol{\nabla}_B f(\boldsymbol{x})^T \boldsymbol{B}^{-1}\boldsymbol{A} = \\ = \left(\boldsymbol{0}, \boldsymbol{\nabla}_N f(\boldsymbol{x})^T - \boldsymbol{\nabla}_B f(\boldsymbol{x})^T \boldsymbol{B}^{-1}\boldsymbol{N}\right), \end{aligned} \tag{A.3}$$

where $\boldsymbol{\nabla}_B f(\boldsymbol{x})$ and $\boldsymbol{\nabla}_N f(\boldsymbol{x})$ are gradients with respect to basic vector $\boldsymbol{x}_B$ and nonbasic vector $\boldsymbol{x}_N$, respectively. To satisfy (A.1), it must hold

$$\begin{aligned} \boldsymbol{\nabla} f(\boldsymbol{x})^T \boldsymbol{d} = \boldsymbol{\nabla}_B f(\boldsymbol{x})^T \boldsymbol{d}_B + \boldsymbol{\nabla}_N f(\boldsymbol{x})^T \boldsymbol{d}_N = \\ = \left(\boldsymbol{\nabla}_N f(\boldsymbol{x})^T - \boldsymbol{\nabla}_B f(\boldsymbol{x})^T \boldsymbol{B}^{-1}\boldsymbol{N}\right) \boldsymbol{d}_N = \boldsymbol{r}_N^T \boldsymbol{d}_N < 0 \end{aligned} \tag{A.4}$$

with $d_j \geq 0$ for $x_j = 0$. To satisfy (A.4), for each component j of nonbasic $\boldsymbol{x}_N$ let $d_j = -r_j$ for $r_j \leq 0$ and $d_j = -x_j r_j$ for $r_j > 0$. This choice of d_j guarantees that $d_j \geq 0$ for $x_j = 0$. In addition, it also improves the step size and enables the convergence.

Finally, the algorithm described below gives $\boldsymbol{d} = \boldsymbol{0}$ iff the point $\boldsymbol{x}$ is KKT point. The reader can find the proof in [1] as well as the proof of convergency of the Reduced Gradient method.

[2] a point $\boldsymbol{x} \in S$ is called an *extreme point* of nonempty convex set S if $\boldsymbol{x} = \lambda \boldsymbol{x}_1 + (1 - \lambda)\boldsymbol{x}_2$ with $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$ and $\lambda \in (0, 1)$ implies $\boldsymbol{x} = \boldsymbol{x}_1 = \boldsymbol{x}_2$

Reduced Gradient Algorithm

Initialization. Choose an initial point $\boldsymbol{x}_1$ such that $\boldsymbol{A}\boldsymbol{x}_1 = \boldsymbol{b}$ and $\boldsymbol{x}_1 \geq \boldsymbol{0}$. Set $k = 1$ and proceed to the main part of algorithm.

Main part.

1. Let I_k be an index set of the m largest components of $\boldsymbol{x}_k$ and $\boldsymbol{a}_j$ be the j-th column of matrix $\boldsymbol{A}$. Define

$$\boldsymbol{B} = \{\boldsymbol{a}_j \,|\, j \in I_k\}, \qquad \boldsymbol{N} = \{\boldsymbol{a}_j \,|\, j \notin I_k\},$$

and calculate the reduced gradient

$$\boldsymbol{r}^T = \boldsymbol{\nabla} f(\boldsymbol{x}_k)^T - \boldsymbol{\nabla}_B f(\boldsymbol{x}_k)^T \boldsymbol{B}^{-1}\boldsymbol{A} \tag{A.5}$$

and the nonbasic part $\boldsymbol{d}_N$ of direction $\boldsymbol{d}_k$ according to the following rule:

$$d_j = \begin{cases} -r_j & \text{for } j \notin I_k \text{ and } r_j \leq 0, \\ -x_j r_j & \text{for } j \notin I_k \text{ and } r_j > 0. \end{cases} \tag{A.6}$$

Then, the improving feasible direction reads

$$\boldsymbol{d}_k^T = \left(-\boldsymbol{B}^{-1}\boldsymbol{N}\boldsymbol{d}_N, \boldsymbol{d}_N\right).$$

If $\boldsymbol{d}_k = \boldsymbol{0}$, then stop, the point $\boldsymbol{x}_k$ is KKT point. Otherwise, go to the second step of the main part.

2. Solve the following line search problem:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}_k + \lambda \boldsymbol{d}_k) \\ \text{subject to} & 0 \leq \lambda \leq \lambda_{\max}, \end{array} \tag{A.7}$$

where

$$\lambda_{\max} = \begin{cases} \min_{1 \leq j \leq n} \left\{ \frac{-x_{jk}}{d_{jk}} \,|\, d_{jk} < 0 \right\} & \text{for } \boldsymbol{d}_k \ngeq \boldsymbol{0}, \\ \infty & \text{for } \boldsymbol{d}_k \geq \boldsymbol{0} \end{cases}$$

and where x_{jk} and d_{jk} denote the j-th component of vectors $\boldsymbol{x}_k$ and $\boldsymbol{d}_k$, respectively. Assume λ_k to be an optimal solution to (A.7) and set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k$, $k = k + 1$ and return to the step 1 of the main part.

A.2 Generalized Reduced Gradient Method

The Reduced Gradient method described above in Section A.1 was generalized by J. Abadie and J. Carpentier in 1969 to the Generalized Reduced Gradient (GRG) method designated for solving nonlinear problems including nonlinear constraints.

Consider the following nonlinear problem with nonlinear constraints:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \\ & \boldsymbol{x} \geq 0, \end{array}$$

where $\boldsymbol{h}(\boldsymbol{x})$ represents a nonlinear vector function $\boldsymbol{h}\colon \mathbb{R}^n \to \mathbb{R}^m$ with m components, $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), \ldots, h_m(\boldsymbol{x}))^T$. Note that the constraints are assumed to be in the form of equality and any inequality constraint can be rewritten to the equality form by adding a slack variable, for instance, an inequality $g(\boldsymbol{x}) \leq 0$ can be rewritten by using a nonnegative slack variable $x^* \geq 0$ to $g(\boldsymbol{x}) + x^* = 0$.

Let $\boldsymbol{x}_k$ be a feasible point and $\boldsymbol{\nabla h}(\boldsymbol{x}_k)$ be the Jacobian matrix of function $\boldsymbol{h}$ at point $\boldsymbol{x}_k$ defined by

$$\boldsymbol{\nabla h}(\boldsymbol{x}_k) = \begin{pmatrix} \frac{\partial h_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial h_1(\boldsymbol{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial h_m(\boldsymbol{x})}{\partial x_n} \end{pmatrix} (\boldsymbol{x}_k).$$

Consider the linearization of the constraint $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}$,

$$\boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{\nabla h}(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) = \boldsymbol{0}. \tag{A.8}$$

Since $\boldsymbol{x}_k$ is a feasible point, $\boldsymbol{h}(\boldsymbol{x}_k) = \boldsymbol{0}$ holds, and therefore above linerizated equation (A.8) can be written in the form

$$\boldsymbol{\nabla h}(\boldsymbol{x}_k)\boldsymbol{x} = \boldsymbol{\nabla h}(\boldsymbol{x}_k)\boldsymbol{x}_k. \tag{A.9}$$

By notation $\boldsymbol{\nabla h}(\boldsymbol{x}_k) = \boldsymbol{A}$ and $\boldsymbol{\nabla h}(\boldsymbol{x}_k)\boldsymbol{x}_k = \boldsymbol{b}$ we conclude that (A.9) is in the form $\boldsymbol{Ax} = \boldsymbol{b}$. Assume that the rank of matrix $\boldsymbol{A} = \boldsymbol{\nabla h}(\boldsymbol{x}_k)$ is m and that $\boldsymbol{A}$ can be decomposed into $(\boldsymbol{B}, \boldsymbol{N})$ and $\boldsymbol{x}_k^T$ into $(\boldsymbol{x}_B^T, \boldsymbol{x}_N^T)$ with $\boldsymbol{x}_B > \boldsymbol{0}$. Thus, with above assumptions, the reduced gradient $\boldsymbol{r}$ can be computed by using formula (A.5) and the improving feasible direction by (A.6). If $\boldsymbol{d}_k = \boldsymbol{0}$, then $\boldsymbol{x}_k$ is KKT point and the algorithm is stopped. Otherwise, the method proceeds to the line search problem (A.7).

Observe that due to the linerization (A.8), the new point $\boldsymbol{x}^* = \boldsymbol{x}_k + \lambda_k \boldsymbol{d}_k$ does not necessarily satisfy the constraint $\boldsymbol{h}(\boldsymbol{x}^*) = \boldsymbol{0}$. To fulfill that, the correction of $\boldsymbol{x}^*$ has to be performed. For this purpose, the Newton–Raphson method can be used to obtain the feasible point $\boldsymbol{x}_{k+1}$ satisfying $\boldsymbol{h}(\boldsymbol{x}_{k+1}) = \boldsymbol{0}$. Thus, the input for the Newton–Raphson procedure is the point $\boldsymbol{x}^*$, not necessarily satisfying $\boldsymbol{h}(\boldsymbol{x}^*) = \boldsymbol{0}$, and the output is the point $\boldsymbol{x}_{k+1}$ satisfying $\boldsymbol{h}(\boldsymbol{x}_{k+1}) = \boldsymbol{0}$.

More detailed information about RG, GRG and Newton–Raphson methods the reader can find in [1].

APPENDIX **B**

Optimality Conditions

In this appendix, we will present and discuss the optimality conditions for mathematical programming problems. We will start with the unconstrained problems and will introduce the *necessary* and *sufficient* optimality conditions. Then, we will generalize our discussion by taking into account equality and inequality constraints. Finally, we will present the *second-order optimality conditions* for discussed problems. Note that the reader can find all proofs of presented theorems in [1].

B.1 Unconstrained Problems

In this section, we will deal with an unconstrained problem, i. e., with a problem without any constraints, in the following form:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{x} \in \mathbb{E}^N. \end{aligned} \tag{B.1}$$

Let us begin with the definitions of the *global* and *local* minimum.

Definition B.1. Consider the programming problem (B.1). The point $\bar{\boldsymbol{x}} \in \mathbb{E}^N$ is said to be a *global minimum* if it holds

$$f(\bar{\boldsymbol{x}}) \leq f(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \mathbb{E}^N.$$

Definition B.2. Let $\varepsilon > 0$ and $\bar{\boldsymbol{x}} \in \mathbb{E}^N$. Then, the set

$$N_\varepsilon(\bar{\boldsymbol{x}}) = \left\{ \boldsymbol{x} \in \mathbb{E}^N \colon |\boldsymbol{x} - \bar{\boldsymbol{x}}| < \varepsilon \right\}$$

is said to be an *ε-neighborhood* of the point $\bar{\boldsymbol{x}}$.

Definition B.3. Consider the programming problem (B.1). The point $\bar{\boldsymbol{x}} \in \mathbb{E}^N$ is said to be a *local minimum* if there exists a positive number ε such that

$$f(\bar{\boldsymbol{x}}) \leq f(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in N_\varepsilon(\bar{\boldsymbol{x}}).$$

If there exists just one local minimum $\bar{\boldsymbol{x}}$, then it is said to be a *strict local minimum*. More precisely, it must satisfy

$$f(\bar{\boldsymbol{x}}) < f(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in N_\varepsilon(\boldsymbol{x}), \quad \bar{\boldsymbol{x}} \neq \boldsymbol{x}.$$

The following theorem presents the necessary condition for the point to be a minimum.

Theorem B.1 (Necessary Optimality Conditions for Unconstrained Problem)**.** *Let the function $f\colon \mathbb{E}^N \to E$ be twice differentiable at the point $\bar{\boldsymbol{x}}$. If the point $\bar{\boldsymbol{x}}$ is a local minimum, then it holds*

$$\boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) = \left(\frac{\partial f}{\partial x_1}(\bar{\boldsymbol{x}}), \ldots, \frac{\partial f}{\partial x_N}(\bar{\boldsymbol{x}}) \right) = \mathbf{0} \tag{B.2}$$

and the Hessian matrix (in short the Hessian) at the point $\bar{\boldsymbol{x}}$

$$\boldsymbol{H}(\bar{\boldsymbol{x}}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{pmatrix} \tag{B.3}$$

is positive semidefinite.

The condition (B.2) is the condition of the first order since the first derivatives play the crucial roles in this condition. Similarly, the condition (B.3) is of the second order. Let us also explain the meaning of the *necessary* condition. Theorem B.1 says that if some point is a minimum of a considered problem, then the *necessary* conditions (B.2) and (B.3) are satisfied. On the other hand, although the both foregoing conditions are satisfied in some point $\widetilde{\boldsymbol{x}}$, we still cannot decide if the minimum occurs in $\widetilde{\boldsymbol{x}}$ or not. For this purpose, we need a stronger tool, the *sufficient* optimality condition presented in the following theorem.

Theorem B.2 (Sufficient Optimality Conditions for Unconstrained Problem)**.** *Let the function $f\colon \mathbb{E}^N \to \mathbb{E}$ be twice differentiable at the point $\bar{\boldsymbol{x}}$. If $\boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) = \mathbf{0}$ and the Hessian matrix $\boldsymbol{H}(\bar{\boldsymbol{x}})$ is positive definite, then the point $\bar{\boldsymbol{x}}$ is a strict local minimum.*

Thus, the both satisfied conditions at $\bar{\boldsymbol{x}}$, $\boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) = \mathbf{0}$ and $\boldsymbol{H}(\bar{\boldsymbol{x}})$ be a positive definite, guarantee that a strict local minimum occurs at $\bar{\boldsymbol{x}}$. Hence, the name *sufficient* optimality conditions.

B.2 Optimality Conditions for Constrained Problems

In this section, we will extend our discussion to problems with constraints. We will start with a problem including inequality constraints and will show an intuitive geometric optimality conditions. Afterwards, we will formalize this geometric schema into rigorous mathematical conditions that lead to *Fritz John* and *Karush–Kuhn–Tucker* conditions. Finally, we will include equality constraints to the foregoing theory and will present important results.

B.2.1 Geometric Interpretation of Optimality Conditions

In following, we are interested in optimality conditions and their geometric meaning. Consider the problem

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{x} \in S, \end{aligned} \tag{B.4}$$

where $S \subset \mathbb{E}^N$ is a general feasible set. Afterwards, we will precise a set S by inequality conditions. Nevertheless, basic definitions are now needed.

Definition B.4. Let S be a nonempty set in $\mathbb{E}^N$ and $\bar{\boldsymbol{x}} \in \operatorname{cl} S$ be a point from the closure of S. Then, for some positive δ, the set D defined by

$$D = \{\boldsymbol{d} : \bar{\boldsymbol{x}} + \lambda \boldsymbol{d} \in S, \quad \boldsymbol{d} \neq \boldsymbol{0}, \quad \forall \lambda \in (0, \delta)\}$$

is said to be the *cone of feasible direction* at point $\bar{\boldsymbol{x}}$.

Hence, a small shift from a feasible point $\bar{\boldsymbol{x}}$ in the direction $\boldsymbol{d} \in D$ is again a feasible point.

Definition B.5. Consider a function $f \colon \mathbb{E}^N \to \mathbb{E}$ and $\bar{\boldsymbol{x}} \in \operatorname{cl} S$. Then, for some positive δ, the set F defined by

$$F = \{\boldsymbol{d} : f(\bar{\boldsymbol{x}} + \lambda \boldsymbol{d}) < f(\bar{\boldsymbol{x}}), \quad \forall \lambda \in (0, \delta)\}$$

is said to be the *cone of improving directions* at point $\bar{\boldsymbol{x}}$.

The interpretation of above definition is following: a small shift from a point $\bar{\boldsymbol{x}}$ in the direction $\boldsymbol{d} \in F$ "improves" the value of the objective function. The following theorem presents the *necessary* and *sufficient* conditions to the problem (B.4).

Theorem B.3. *Assume the problem* (B.4)*, where $f : \mathbb{E}^N \to \mathbb{E}$ is differentiable at a point $\bar{\boldsymbol{x}} \in S$ and $S \subset \mathbb{E}^N$ is a nonempty set. Define the set F_0 as follows,*

$$F_0 = \left\{\boldsymbol{d} \colon \boldsymbol{\nabla} f(\bar{\boldsymbol{x}})^T \boldsymbol{d} < 0\right\}. \tag{B.5}$$

If a point $\bar{\boldsymbol{x}}$ is a local optimal solution, then it holds

$$F_0 \cap D = \emptyset,$$

where D stands for the cone of feasible directions of set S at the point $\bar{\boldsymbol{x}}$. On the contrary, assume that the condition $F_0 \cap D = \emptyset$ holds, the function f is pseudoconvex[1] *at the point $\bar{\boldsymbol{x}}$ and there exists an ε-neighborbood $N_\varepsilon(\bar{\boldsymbol{x}})$ of the point $\bar{\boldsymbol{x}}$ and it holds*

$$\boldsymbol{d} = (\boldsymbol{x} - \bar{\boldsymbol{x}}) \in D \quad \forall \boldsymbol{x} \in S \cap N_\varepsilon(\bar{\boldsymbol{x}}).$$

Then, the point $\bar{\boldsymbol{x}}$ is a local minimum to the problem (B.4).

Now, assume that a general set of feasible solutions S is described by using inequality constraints. In following, we will deal with the programming problem in the form

$$\begin{array}{lll} \text{minimize} & f(\boldsymbol{x}) & \\ \text{subject to} & g_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m, \\ & \boldsymbol{x} \in X, & \end{array} \tag{B.6}$$

where X is a nonempty open set in $\mathbb{E}^N$. Hence, the feasible set S is specified by

$$S = \{\boldsymbol{x} \in X, g_i(\boldsymbol{x}) \leq 0 \text{ for all } i = 1, \ldots, m\}. \tag{B.7}$$

[1] a differentiable function $f : S \subset \mathbb{E}^N \mapsto \mathbb{E}$ is said to be *pseudoconvex* if for all $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ satistying $\boldsymbol{\nabla} f(\boldsymbol{x}_1)^T (\boldsymbol{x}_2 - \boldsymbol{x}_1) \geq 0$ the condition $f(\boldsymbol{x}_2) \geq f(\boldsymbol{x}_1)$ holds

Definition B.6. Consider the problem (B.6) where the feasible is defined by (B.7). Let $\bar{\boldsymbol{x}} \in S$ be a feasible point. Then, the set I defined by

$$I = \{i \in \{1, \ldots, m\} : g_i(\bar{\boldsymbol{x}}) = 0\} \tag{B.8}$$

is said to be the *index set of active constraints*. Assume that the functions g_i are differentiable at the point $\bar{\boldsymbol{x}}$ for all $i \in I$ and g_i are continuous at $\bar{\boldsymbol{x}}$ for $i \notin I$. Then, we define the set G_0 as follows,

$$G_0 = \left\{\boldsymbol{d}\colon \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}})^T \boldsymbol{d} < 0 \text{ for all } i \in I\right\}. \tag{B.9}$$

Theorem B.4. *Consider the problem* (B.6) *where f and g_i for all $i = 1, \ldots, m$ are functions $f, g_i\colon \mathbb{E}^N \to \mathbb{E}$. Let $\bar{\boldsymbol{x}}$ be a feasible point and assume that the functions f and g_i for all $i \in I$ are differentiable at point $\bar{\boldsymbol{x}}$ and g_i are continuous at $\bar{\boldsymbol{x}}$ for all $i \notin I$. If the point $\bar{\boldsymbol{x}}$ is a local minimum, then*

$$F_0 \cap G_0 = \emptyset,$$

where F_0 and G_0 are defined by (B.5) *and* (B.9)*, respectively. Conversely, if it holds $F_0 \cap G_0 = \emptyset$, f is a pseudoconvex function at a point $\bar{\boldsymbol{x}}$ and the functions g_i for all $i \in I$ are strictly pseudoconvex*[2] *over some ε-neigborhood $N_\varepsilon(\bar{\boldsymbol{x}})$ of the point $\bar{\boldsymbol{x}}$, $\varepsilon > 0$, then the point $\bar{\boldsymbol{x}}$ is a local minimum to the problem* (B.6).

B.2.2 Fritz John Conditions for Inequality Constraints

In the following theorems, we will formalize the geometric optimality conditions based on the condition $F_0 \cap G_0 = \emptyset$ to the optimality conditions including the gradient of objective function and the gradients of active constraints. These optimality conditions were derived by Fritz John in 1948.

Theorem B.5 (Fritz John Necessary Optimality Conditions)**.** *Consider the problem* (B.6)*. Let f and g_i for all $i = 1, \ldots, m$ be functions $f, g_i\colon \mathbb{E}^N \to \mathbb{E}$, the set I be defined by* (B.8) *and a point $\bar{\boldsymbol{x}}$ be a feasible solution. Moreover, assume that the functions f and g_i for all $i \in I$ are differentiable at $\bar{\boldsymbol{x}}$ and functions g_i for all $i \notin I$ are continuous at $\bar{\boldsymbol{x}}$. If a point $\bar{\boldsymbol{x}}$ is a local minimum to the problem* (B.6)*, then there exist numbers u_0 and u_i for all $i \in I$ such that*

$$\begin{aligned} u_0 \boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) + \sum_{i \in I} u_i \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}}) &= \boldsymbol{0}, \\ u_0, u_i &\geq 0 \quad \text{for all } i \in I, \\ (u_0, \boldsymbol{u}_I) &\neq (0, \boldsymbol{0}), \end{aligned} \tag{B.10}$$

where $\boldsymbol{u}_I$ is a vector that consists of all u_i for $i \in I$. Moreover, if the functions g_i for all $i \notin I$ are also differentible at the point $\bar{\boldsymbol{x}}$, then the condition (B.10) *can be rewritten to the equivalent form*

$$\begin{aligned} u_0 \boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}}) &= \boldsymbol{0}, \\ u_i g_i(\bar{\boldsymbol{x}}) &= 0 \quad \text{for all } i = 1, \ldots, m, \\ u_0, u_i &\geq 0 \quad \text{for all } i = 1, \ldots, m, \\ (u_0, \boldsymbol{u}) &\neq (0, \boldsymbol{0}), \end{aligned} \tag{B.11}$$

[2] a differentiable function $f : S \subset \mathbb{E}^N \mapsto \mathbb{E}$ is said to be *strictly pseudoconvex* if for each $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$, satistying $\boldsymbol{\nabla} f(\boldsymbol{x}_1)^T(\boldsymbol{x}_2 - \boldsymbol{x}_1) \geq 0$ the condition $f(\boldsymbol{x}_2) \geq f(\boldsymbol{x}_1)$ holds

where $\boldsymbol{u} = (u_1, \ldots, u_m)^T$.

Observe that $u_i > 0$ for all $i \in I$ and $u_i = 0$ for all $i \notin I$. The condition of feasibility of point $\bar{\boldsymbol{x}}$ in Theorem B.5 is called the *primal feasibility condition*, the conditions $u_0 \boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}}) = \boldsymbol{0}$, $u_0, u_i \geq 0$ for all $i = 1, \ldots, m$ and $(u_0, \boldsymbol{u}) \neq (0, \boldsymbol{0})$ are called the *dual feasibility conditions* and finally, the conditions $u_i g_i(\bar{\boldsymbol{x}}) = 0$ for all $i = 1, \ldots, m$ are called the *complementary slackness conditions*. The primal feasibility, dual feasibility and complementary slackness conditions together are called the *Fritz John optimality conditions*. A point $\bar{\boldsymbol{x}}$ is said to be a *Fritz John (FJ) point* if there exist so-called Lagrange multipliers $\bar{u}_0, \bar{u}_1, \ldots, \bar{u}_m$ such that the point $\bar{\boldsymbol{x}}$ with $\bar{u}_0, \bar{u}_1, \ldots, \bar{u}_m$ satisfies the Fritz John conditions (B.11).

Theorem B.6 (Fritz John Sufficient Optimality Conditions)**.** *Consider the problem* (B.6). *Let* $X \subset \mathbb{E}^N$ *be a nonempty open set,* f *and* g_i *for all* $i = 1, \ldots, m$ *be functions* $f, g_i \colon \mathbb{E}^N \to \mathbb{E}$. *Let a point* $\bar{\boldsymbol{x}}$ *be a FJ point and the set* I *be defined by* (B.8). *Define the set* S *by*

$$S = \{\boldsymbol{x} \in X \colon g_i(\boldsymbol{x}) \leq 0, i \in I\}.$$

If there exists an ε*-neighborhood* $N_\varepsilon(\bar{\boldsymbol{x}})$, $\varepsilon > 0$ *of the point* $\bar{\boldsymbol{x}}$ *such that the function* f *is pseudoconvex over* $N_\varepsilon(\bar{\boldsymbol{x}}) \cap S$ *and* g_i *for all* $i \in I$ *are strictly pseudoconvex over* $N_\varepsilon(\bar{\boldsymbol{x}}) \cap S$, *then the point* $\bar{\boldsymbol{x}}$ *is a local minimum to the problem* (B.6).

Note that the set S in above theorem is the relaxation of the feasible region of (B.6) since all non-active constraints g_i, i. e., g_i for all $i \notin I$, are taken out.

Now, we will discuss the Fritz John necessary optimal conditions (B.10). Observe that if the condition $\boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}}) = \boldsymbol{0}$ for some $i \in I$ at a point $\bar{\boldsymbol{x}}$ is safisfied, then we can set the corresponding u_i to any positive numbers and the others u_i to zero, thereby the Fritz John optimality conditions (B.10) will be satistied regardless to the objective function and hence, $\bar{\boldsymbol{x}}$ will be FJ point. Moreover, by adding a redundant constraint to the problem, any feasible point $\bar{\boldsymbol{x}}$ can be a FJ point. In particular, suppose the redundant condition $g_{m+1}(\boldsymbol{x}) = -\|\bar{\boldsymbol{x}} - \boldsymbol{x}\| \leq 0$. It holds for all $\boldsymbol{x} \in \mathbb{E}^N$ and becomes to be active for $\boldsymbol{x} = \bar{\boldsymbol{x}}$, and therefore, $\boldsymbol{\nabla} g_{m+1}(\bar{\boldsymbol{x}}) = \boldsymbol{0}$. Hence, with $u_{m+1} > 0$ and $u_i = 0$ for all $i = 0, \ldots, m$, the conditions (B.10) are satisfied and the point $\bar{\boldsymbol{x}}$ is FJ point. From the geometrical point of view discussed above, in this case $G_0 = \emptyset$, the condition $F_0 \cap G_0 = \emptyset$ is satisfied regardless to the set F_0 and $\bar{\boldsymbol{x}}$ is FJ point. Thus, we wish to incorporate the objective function (more precisely its gradient) to the optimality conditions. These considerations lead to the Karush–Kuhn–Tucker optimality conditions.

B.2.3 Karush–Kuhn–Tucker Conditions for Inequality Constraints

In this subsection, we will describe the *Karush–Kuhn–Tucker (KKT) optimality conditions* for the problem with inequality constraints motivated by the discussion at the end of the foregoing subsection. KKT conditions are the extension of FJ conditions with positive Lagrange multiplier u_0 corresponding to the term $\boldsymbol{\nabla} f$. In following, we can suppose that $u_0 = 1$ without loss of generality, since other Lagrange multipliers u_i are just rescaled. These well-known KKT optimality conditions were independently derived by Karush in 1939 and by Kuhn–Tucker in 1951.

Theorem B.7 (Karush–Kuhn–Tucker Necessary Optimality Conditions)**.** *Consider the problem* (B.6). *Let* X *be a nonempty open set in* $\mathbb{E}^N$, f *and* g_i *for all* $i = 1, \ldots, m$ *be*

functions $f, g_i \colon \mathbb{E}^N \to \mathbb{E}$, *the set* I *be defined by* (B.8) *and* $\bar{\boldsymbol{x}}$ *be a feasible point. Assume that the functions* f *and* g_i *for all* $i \in I$ *are differentiable at point* $\bar{\boldsymbol{x}}$, *the functions* g_i *for all* $i \notin I$ *are continuous at the point* $\bar{\boldsymbol{x}}$ *and the gradients* $\boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}})$ *for all* $i \in I$ *are linearly independent. If the point* $\bar{\boldsymbol{x}}$ *is a local minimum to the problem* (B.6), *then there exist numbers* u_i *for all* $i \in I$ *such that*

$$\begin{aligned} \boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) + \sum_{i \in I} u_i \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}}) &= \mathbf{0}, \\ u_i &\geq 0 \quad \textit{for all } i \in I. \end{aligned} \tag{B.12}$$

If g_i *for all* $i \notin I$ *are also differentiable at point* $\bar{\boldsymbol{x}}$, *then* (B.12) *can be rewritten to the equivalent form*

$$\begin{aligned} \boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}}) &= \mathbf{0}, \\ u_i g_i(\bar{\boldsymbol{x}}) &= 0 \quad \textit{for all } i = 1, \dots, m, \\ u_i &\geq 0 \quad \textit{for all } i = 1, \dots, m. \end{aligned} \tag{B.13}$$

The scalars u_i are called the *Lagrange multipliers*, as in the FJ conditions. Similarly, the condition of feasibility of $\bar{\boldsymbol{x}}$ is called the *primal feasibility condition*, the conditions $\boldsymbol{\nabla} f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}}) = \mathbf{0}$ with $u_i \geq 0$ for all $i = 1, \dots, m$ are called the *dual feasibility conditions* and the requirement $u_i g_i(\bar{\boldsymbol{x}}) = 0$ for all $i = 1, \dots, m$ is called the *complementary slackness condition*. The primal feasibility, dual feasibility and complementary slackness conditions together are called the *Karush–Kuhn–Tucker optimality conditions*.

A point $\bar{\boldsymbol{x}}$ is said to be a *Karush–Kuhn–Tucker (KKT) point* if there exist Lagrange multipliers $\bar{u}_1, \dots, \bar{u}_m$ such that the point $\bar{\boldsymbol{x}}$ with $\bar{u}_1, \dots, \bar{u}_m$ satisfies the Karush–Kuhn–Tucker optimality conditions.

Theorem B.8 (Karush–Kuhn–Tucker Sufficient Optimality Conditions)**.** *Consider the problem* (B.6). *Let* $X \subset \mathbb{E}^N$ *be a nonempty open set,* f *and* g_i *for all* $i = 1, \dots, m$ *be functions* $f, g_i \colon \mathbb{E}^N \to \mathbb{E}$, *the set* I *be defined by* (B.8) *and a point* $\bar{\boldsymbol{x}}$ *be a KKT point. Define the set* S *by*

$$S = \{\boldsymbol{x} \in X \colon g_i(\boldsymbol{x}) \leq 0, i \in I\}.$$

If there exists an ε*-neighborhood* $N_\varepsilon(\bar{\boldsymbol{x}})$, $\varepsilon > 0$ *such that* f *is pseudoconvex over* $N_\varepsilon(\bar{x}) \cap S$ *and* g_i *for* $i \in I$ *are differentiable at* $\bar{\boldsymbol{x}}$ *and are quasiconvex*[3] *over* $N_\varepsilon(\bar{x}) \cap S$, *then the point* $\bar{\boldsymbol{x}}$ *is a local minimum to the problem* (B.6).

Note that the set S is, as in Theorem B.6, the relaxation of the feasible region of (B.6) since all non-active constraints g_i are taken out. Also note that it can be shown that if f and g_i are convex at the point $\bar{\boldsymbol{x}}$, then KKT conditions (B.13) are sufficient.

So far, we considered problems with inequality constraints. In following, we will take equality constraints into account and will generalize the above theory.

[3] a function $f \colon S \subset \mathbb{E}^N \mapsto \mathbb{E}$ is said to be *quasiconvex*, if for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S$ it holds

$$f\big(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2\big) \leq \max\{f(\boldsymbol{x}_1), f(\boldsymbol{x}_2)\}$$

for all $\lambda \in (0, 1)$

Consider the following problem including both inequality and equality constraints:

$$\begin{array}{lll} \text{minimize} & f(\boldsymbol{x}) & \\ \text{subject to} & g_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m, \\ & h_j(\boldsymbol{x}) = 0, & j = 1, \ldots, n, \\ & \boldsymbol{x} \in X. & \end{array} \tag{B.14}$$

The following theorem is an extension of Theorem B.4 taking into account also equality constraints.

Theorem B.9. *Consider the problem* (B.14) *and let* $X \subset \mathbb{E}^N$ *be a nonempty open set,* f, g_i *for all* $i = 1, \ldots, m$ *and* h_j *for all* $j = 1, \ldots, n$ *be functions* $f, g_i, h_j \colon \mathbb{E}^N \to \mathbb{E}$ *and the set* I *be defined by* (B.8). *Assume that a point* $\bar{\boldsymbol{x}}$ *is a local minimum to the problem* (B.14), *the functions* f *and* g_i *for all* $i \in I$ *are differentiable at* $\bar{\boldsymbol{x}}$, *the functions* g_i *for all* $i \notin I$ *are continuous at* $\bar{\boldsymbol{x}}$ *and the functions* h_j *for all* $j = 1, \ldots, n$ *are continuously differentiable*[4] *at* $\bar{\boldsymbol{x}}$. *If the gradients* $\boldsymbol{\nabla} h_j(\bar{\boldsymbol{x}})$ *for all* $j = 1, \ldots, n$ *are linearly independent, then*

$$F_0 \cap G_0 \cap H_0 = \emptyset, \tag{B.15}$$

where

$$\begin{aligned} F_0 &= \left\{ \boldsymbol{d} \colon \boldsymbol{\nabla} f(\bar{\boldsymbol{x}})^T \boldsymbol{d} < 0 \right\}, \\ G_0 &= \left\{ \boldsymbol{d} \colon \boldsymbol{\nabla} g_i(\bar{\boldsymbol{x}})^T \boldsymbol{d} < 0 \text{ for all } i \in I \right\}, \\ H_0 &= \left\{ \boldsymbol{d} \colon \boldsymbol{\nabla} h_j(\bar{\boldsymbol{x}})^T \boldsymbol{d} = 0 \text{ for all } j = 1, \ldots, n \right\}. \end{aligned}$$

Conversely, assume that (B.15) *holds. If the function* f *is pseudoconvex at* $\bar{\boldsymbol{x}}$, *the functions* g_i *for all* $i \in I$ *are strictly pseudoconvex over* $N_\varepsilon(\bar{\boldsymbol{x}})$ *for some* $\varepsilon > 0$ *and the functions* h_j *for all* $j = 1, \ldots, n$ *are affine*[5], *then the point* $\bar{\boldsymbol{x}}$ *is a local minimum to the problem* (B.14).

B.2.4 Fritz John Conditions for Inequality and Equality Constraints

In following, we will proceed similarly as in Subsection B.2.2. We will successively formalize the results from Theorem B.9 based on the geometric interpretation of the optimality condition $F_0 \cap G_0 \cap H_0 = \emptyset$ to the algebraic form that leads to Fritz John necessary and sufficient conditions.

Theorem B.10 (Fritz John Necessary Optimality Conditions)**.** *Consider the problem* (B.14). *Let* $X \subset \mathbb{E}^N$ *be a nonempty set,* f, g_i *for all* $i = 1, \ldots, m$ *and* h_j *for all* $j = = 1, \ldots, n$ *be functions* $f, g_i, h_j \colon \mathbb{E}^N \to \mathbb{E}$, *a point* $\bar{\boldsymbol{x}}$ *be a feasible solution and the set* I *be defined by* (B.8). *Assume that the functions* f *and* g_i *for all* $i \in I$ *are differentiable at the point* $\bar{\boldsymbol{x}}$, *the functions* g_i *for all* $i \notin I$ *are continuous at the point* $\bar{\boldsymbol{x}}$ *and the functions* h_j *for all* $j = 1, \ldots, n$ *are continuously differentiable at the point* $\bar{\boldsymbol{x}}$. *If the point* $\bar{\boldsymbol{x}}$ *is a*

[4] a function f is said to be *continuously differentiable* if it is differentiable and this derivative is a continuous function

[5] a function f is said to be *affine* if it can be expressed in the form $f(\boldsymbol{x}) = f(x_1, \ldots, x_N) = a_1 x_1 + + \cdots + a_N x_N + b = \boldsymbol{a}^T \boldsymbol{x} + b$, where $\boldsymbol{a}$ is a $N \times 1$ vector and b is a scalar

local minimum to the problem (B.14)*, then there exist numbers* u_0*,* u_i *for all* $i \in I$ *and* v_j *for all* $j = 1, \dots, n$ *such that*

$$\begin{aligned} u_0 \nabla f(\bar{\boldsymbol{x}}) + \sum_{i \in I} u_i \nabla g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} v_j \nabla h_j(\bar{\boldsymbol{x}}) = \mathbf{0}, \\ u_0, u_i \geq 0 \quad \text{for all } i \in I, \\ (u_0, \boldsymbol{u}_I, \boldsymbol{v}) \neq (0, \mathbf{0}, \mathbf{0}), \end{aligned} \tag{B.16}$$

where $\boldsymbol{u}_I$ *is a vector that consists of all* u_i *for* $i \in I$ *and* $\boldsymbol{v} = (v_1, \dots, v_n)^T$*. Moreover, if the functions* g_i *for all* $i \notin I$ *are also differentiable at the point* $\bar{\boldsymbol{x}}$*, then the above conditions* (B.16) *can be rewritten to the equivalent form*

$$\begin{aligned} u_0 \nabla f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \nabla g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} v_j \nabla h_j(\bar{\boldsymbol{x}}) = \mathbf{0}, \\ u_i g_i(\bar{\boldsymbol{x}}) = 0 \quad \text{for all } i = 1, \dots, m, \\ u_0, u_i \geq 0 \quad \text{for all } i = 1, \dots, m, \\ (u_0, \boldsymbol{u}, \boldsymbol{v}) \neq (0, \mathbf{0}, \mathbf{0}), \end{aligned} \tag{B.17}$$

where $\boldsymbol{u} = (u_1, \dots, u_m)^T$ *and* $\boldsymbol{v} = (v_1, \dots, v_n)^T$*.*

Note that there is no restriction of scalars v_j for all $j = 1, \dots, n$ in Theorem B.10, $v_j \lesseqgtr 0$, but $\boldsymbol{v} = (v_1, \dots, v_n)^T \neq (0, \dots, 0) = \mathbf{0}$.

Similarly as in Theorem B.5, the condition of the feasibility of the point $\bar{\boldsymbol{x}}$ is called the *primal feasibility condition*, the conditions $u_0 \nabla f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \nabla g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} v_j \nabla h_j(\bar{\boldsymbol{x}}) = \mathbf{0}$, $u_0, u_i \geq 0$ for all $i = 1, \dots, m$ and $(u_0, \boldsymbol{u}, \boldsymbol{v}) \neq (0, \mathbf{0}, \mathbf{0})$ are called the *dual feasibility conditions* and finally, the conditions $u_i g_i(\bar{\boldsymbol{x}}) = 0$ for all $i = 1, \dots, m$ are called the *complementary slackness conditions*. The primal feasibility, dual feasibility and complementary slackness conditions together are called the *Fritz John optimality conditions*.

A point $\bar{\boldsymbol{x}}$ is said to be a *Fritz John (FJ) point* if there exist Lagrange multipliers $\bar{u}_0$, $\bar{u}_1, \dots, \bar{u}_m$ and $\bar{v}_1, \dots, \bar{v}_n$ such that the point $\bar{\boldsymbol{x}}$ with $\bar{u}_0$, $\bar{u}_1, \dots, \bar{u}_m$, $\bar{v}_1, \dots, \bar{v}_n$ satisfies the Fritz John conditions (B.17).

The following theorem states the Fritz John sufficient conditions.

Theorem B.11 (Fritz John Sufficient Optimality Conditions)**.** *Consider the problem* (B.14) *and let* $X \subset \mathbb{E}^N$ *be a nonempty open set,* f*,* g_i *for all* $i, \dots, m$ *and* h_j *for all* $j = 1, \dots, n$ *be functions* $f, g_i, h_j \colon \mathbb{E}^N \to \mathbb{E}$*. Let a point* $\bar{\boldsymbol{x}}$ *be a FJ point and the set* I *be defined by* (B.8)*. Furthermore, define the set* S *by*

$$S = \{\boldsymbol{x} \colon g_i(\boldsymbol{x}) \leq 0 \text{ for all } i \in I, h_j(\boldsymbol{x}) = 0 \text{ for all } j = 1, \dots, n\}.$$

If the functions h_j *for all* $j = 1, \dots, n$ *are affine and the gradients* $\nabla h_j(\bar{\boldsymbol{x}})$ *for all* $j = 1, \dots, n$ *are linearly independent and if there exists an* ε*-neighborhood* $N_\varepsilon(\bar{\boldsymbol{x}})$ *of the point* $\bar{\boldsymbol{x}}$*, such that the function* f *is pseudoconvex over* $S \cap N_\varepsilon(\bar{\boldsymbol{x}})$ *and the functions* g_i *for all* $i \in I$ *are strictly pseudoconvex over* $S \cap N_\varepsilon(\bar{\boldsymbol{x}})$*, then the point* $\bar{\boldsymbol{x}}$ *is a local minimum to the problem* (B.14)*.*

B.2.5 Karush–Kuhn–Tucker Conditions for Inequality and Equality Constraints

The following Karush–Kuhn–Tucker optimality conditions are motivated by the similar discussion as in case of problems with inequality constraints since the Fritz John optimality condition do not prescribe a positive Langangian multiplier u_0 and it may happen that Fritz John conditions are satisfied regardless to the objective function. The Karush–Kuhn–Tucker conditions do not permit this situation and the objective function is incorporated to play a significant role in these optimality conditions.

Theorem B.12 (Karush–Kuhn–Tucker Necessary Optimality Conditions)**.** *Consider the problem* (B.14)*. Let $X \subset \mathbb{E}^N$ be a nonempty open set and f, g_i for all $i = 1, \dots, m$ and h_j for all $j = 1, \dots, n$ be functions $f, g_i, h_j \colon \mathbb{E}^N \to \mathbb{E}$. Let a point $\bar{\boldsymbol{x}}$ be a feasible solution to the problem* (B.14) *and the set I be defined by* (B.8)*. Assume that the functions f and g_i for all $i \in I$ are differentiable at the point $\bar{\boldsymbol{x}}$, the functions g_i for all $i \notin I$ are continuous at the point $\bar{\boldsymbol{x}}$ and the functions h_j for all $j = 1, \dots, n$ are continuously differentiable at the point $\bar{\boldsymbol{x}}$. Moreover, assume that the gradients $\nabla g_i(\bar{\boldsymbol{x}})$ for all $i \in I$ and $\nabla h_j(\bar{\boldsymbol{x}})$ for all $j = 1, \dots, n$ are linearly independent. If the point is a local minimum to the problem* (B.14)*, then there exist unique numbers u_i for all $i \in I$ and v_j for all $j = 1, \dots, n$ such that*

$$\begin{aligned} \nabla f(\bar{\boldsymbol{x}}) + \sum_{i \in I} u_i \nabla g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} v_j \nabla h_j(\bar{\boldsymbol{x}}) = \boldsymbol{0}, \\ u_i \geq 0 \quad \text{for all } i \in I. \end{aligned} \tag{B.18}$$

Furthermore, if the functions g_i for all $i \notin I$ are also differentiable at the point $\bar{\boldsymbol{x}}$, then the above conditions (B.18) *can be rewritten to the form*

$$\begin{aligned} \nabla f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \nabla g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} v_j \nabla h_j(\bar{\boldsymbol{x}}) = \boldsymbol{0}, \\ u_i g_i(\bar{\boldsymbol{x}}) = 0 \quad \text{for all } i = 1, \dots, m, \\ u_i \geq 0 \quad \text{for all } i = 1, \dots, m. \end{aligned} \tag{B.19}$$

As in foregoing cases, the scalars u_i and v_j are called the *Lagrange multipliers*. Similarly, the condition of the feasibility of $\bar{\boldsymbol{x}}$ is called the *primal feasibility condition*, the conditions $\nabla f(\bar{\boldsymbol{x}}) + \sum_{i=1}^{m} u_i \nabla g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} v_j \nabla h_j(\bar{\boldsymbol{x}}) = \boldsymbol{0}$ with $u_i \geq 0$ for all $i = 1, \dots, m$ are called the *dual feasibility conditions* and the requirement $u_i g_i(\bar{\boldsymbol{x}}) = 0$ for all $i = 1, \dots, m$ is called the *complementary slackness condition*. The primal feasibility, dual feasibility and complementary slackness conditions together are called the *Karush–Kuhn–Tucker optimality conditions*.

A point $\bar{\boldsymbol{x}}$ is said to be a *Karush–Kuhn–Tucker (KKT) point* if there exist Lagrange multipliers $\bar{u}_1, \dots, \bar{u}_m, \bar{v}_1, \dots, \bar{v}_n$ such that the point $\bar{\boldsymbol{x}}$ with $\bar{u}_1, \dots, \bar{u}_m, \bar{v}_1, \dots, \bar{v}_n$ satisfies the Karush–Kuhn–Tucker optimality conditions.

If we allow relatively slight additional assumptions to the convexity of the funtions f, g_i and h_j in Theorem B.12, then it can be shown that the Karush–Kuhn–Tucker optimality conditions are also sufficient conditions for local optimal solution. This result states the following theorem.

Theorem B.13 (Karush–Kuhn–Tucker Sufficient Optimality Conditions)**.** *Consider the problem* (B.14)*. Let $X \subset \mathbb{E}^N$ be a nonempty open set and f, g_i for all $i = 1, \dots, m$ and*

h_j for all $j = 1, \ldots, n$ be functions $f, g_i, h_j \colon \mathbb{E}^N \to \mathbb{E}$. Let a point $\bar{\boldsymbol{x}}$ be a feasible solution to the problem (B.14) *and the set I be defined by* (B.8). *Further, let a point $\bar{\boldsymbol{x}}$ be the KKT point. Assume that the function f is pseudoconvex at the point $\bar{\boldsymbol{x}}$, the functions g_i for all $i \in I$ are quasiconvex at the point $\bar{\boldsymbol{x}}$, the functions h_j for all $j \in J = \{j \colon \bar{v}_j > 0\}$ are quasiconvex and the functions h_j for all $j \in K = \{j \colon \bar{v}_j < 0\}$ are quasiconcave*[6] *at the point $\bar{\boldsymbol{x}}$. Then, the point $\bar{\boldsymbol{x}}$ is a global minimum to the problem* (B.14).

Note that all optimality conditions discussed above in this chapter can also be written in more compact vector form. For example, we can write the Karush–Kuhn–Tucker necessary optimality conditions (B.19) in the form

$$\begin{aligned} \nabla f(\bar{\boldsymbol{x}}) + \nabla \boldsymbol{g}(\bar{\boldsymbol{x}})^T \boldsymbol{u} + \nabla \boldsymbol{h}(\bar{\boldsymbol{x}})^T \boldsymbol{v} &= \boldsymbol{0}, \\ \boldsymbol{u}^T \boldsymbol{g} &= 0 \\ \boldsymbol{u} &\geq \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{g} \colon \mathbb{E}^N \to \mathbb{E}^m$ and $\boldsymbol{h} \colon \mathbb{E}^N \to \mathbb{E}^n$ are vector functions, $\boldsymbol{g}(\bar{\boldsymbol{x}}) = (g_1(\bar{\boldsymbol{x}}), \ldots, g_m(\bar{\boldsymbol{x}}))^T$, $\boldsymbol{h}(\bar{\boldsymbol{x}}) = (h_1(\bar{\boldsymbol{x}}), \ldots, h_n(\bar{\boldsymbol{x}}))^T$ and $\boldsymbol{u} = (u_1, \ldots, u_m)^T$, $\boldsymbol{v} = (v_1, \ldots, v_m)^T$, $\bar{\boldsymbol{x}} = (\bar{x}_1, \ldots, \bar{x}_N)^T$ are m-, n- and N-dimensional vectors, respectively.

B.3 Second-Order Optimality Conditions

All optimality conditions so far discussed in this appendix are called the *first-order optimality conditions* since they contain only the first derivatives of functions f, g_i and h_j, i. e. their gradients. In this section, we will briefly introduce the concept of the *second-order optimality conditions* including the second-order derivatives that are based on the following motivation. The reader can find further information in detail in [1].

In theorem B.1, we saw that the necessary condition $\nabla g(\bar{\boldsymbol{x}}) = \boldsymbol{0}$ holds for all optimal solutions to a given problem. Nevertheless, if the condition $\nabla g(\bar{\boldsymbol{x}}) = \boldsymbol{0}$ holds at $\bar{\boldsymbol{x}}$, there are still three posibilities since the point $\bar{\boldsymbol{x}}$ can be a local minimum, a local maximum or a saddle point. However, we can use the second-order (or higher order) optimality conditions to reduce the set of suggested solutions by the first-order necessary conditions.

Consider the problem (B.14). The second-order optimality conditions are based on the concept of the *restricted Lagrangian function* $L(\boldsymbol{x})$ defined by

$$L(\boldsymbol{x}) \equiv \Phi(\boldsymbol{x}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}) \equiv f(\boldsymbol{x}) + \sum_{i \in I} \bar{u}_i g_i(\boldsymbol{x}) + \sum_{j=1}^{n} \bar{v}_i h_j(\boldsymbol{x}), \tag{B.20}$$

where the set I of active constraints is defined by (B.8).

Theorem B.14 (Karush–Kuhn–Tucker Second-Order Necessary Optimality Conditions)**.** *Consider the problem* (B.14). *Let $X \subset \mathbb{E}^N$ be a nonempty open set and functions f, g_i for all $i = 1, \ldots, m$ and h_j for all $j = 1, \ldots, n$ be twice differentiable. Let a point $\bar{\boldsymbol{x}}$ be a local minimum to the problem* (B.14) *and the set I be defined by* (B.8), *the restricted Lagrangian function $L(\boldsymbol{x})$ by* (B.20) *and its Hessian at the point $\bar{\boldsymbol{x}}$ by*

$$\nabla^2 L(\bar{\boldsymbol{x}}) \equiv \nabla^2 f(\bar{\boldsymbol{x}}) + \sum_{i \in I} \bar{u}_i \nabla^2 g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} \bar{v}_j \nabla^2 h_j(\bar{\boldsymbol{x}}), \tag{B.21}$$

[6] a function $f \colon S \subset \mathbb{E}^N \mapsto \mathbb{E}$ is said to be *quasiconcave* if the function $-f$ is quasiconvex

where $\nabla^2 f(\bar{\boldsymbol{x}})$, $\nabla^2 g_i(\bar{\boldsymbol{x}})$ *for all* $i \in I$ *and* $\nabla^2 h_j(\bar{\boldsymbol{x}})$ *for all* $j = 1, \ldots, n$ *are the Hessians of objective and constraint functions, respectively, at the point* $\bar{\boldsymbol{x}}$. *Assume that the Hessians* $\nabla^2 g_i(\bar{\boldsymbol{x}})$ *for all* $i \in I$ *and* $\nabla^2 h_j(\bar{\boldsymbol{x}})$ *for all* $j = 1, \ldots, n$ *are linearly independent. Then, the point* $\bar{\boldsymbol{x}}$ *is a KKT point. Furthermore, it holds*

$$\boldsymbol{d}^T \nabla^2 L(\bar{\boldsymbol{x}}) \boldsymbol{d} \geq 0 \text{ for all } \boldsymbol{d} \in \bar{C}, \tag{B.22}$$

where

$$\bar{C} = \left\{ \boldsymbol{d} \neq \boldsymbol{0} \colon \nabla g_i(\bar{\boldsymbol{x}})^T \boldsymbol{d} \leq 0 \text{ for all } i \in I, \nabla h_j(\bar{\boldsymbol{x}})^T \boldsymbol{d} = 0 \text{ for all } j = 1, \ldots, n \right\}. \tag{B.23}$$

Theorem B.15 (Karuh–Kuhn–Tucker Second-Order Sufficient Optimality Conditions)**.** *Consider the problem* (B.14). *Let* $X \subset \mathbb{E}^N$ *be a nonempty open set and the functions* f, g_i *for all* $i = 1, \ldots, m$ *and* h_j *for all* $j = 1, \ldots, n$ *be twice differentiable. Let a point* $\bar{\boldsymbol{x}}$ *be a KKT point to the problem* (B.14) *with corresponding Langrange multipliers* $\bar{\boldsymbol{u}}$ *and* $\bar{\boldsymbol{v}}$ *and the set* I *be defined by* (B.8). *Define the sets* $I^+ = \{i \in I \colon \bar{u}_i > 0\}$ *and* $I^0 = \{i \in I \colon \bar{u}_i = 0\}$, *the restricted Lagrangian function* $L(\boldsymbol{x})$ *by* (B.20) *and denote its Hessian*[7] *at the point* $\bar{\boldsymbol{x}}$ *by*

$$\nabla^2 L(\bar{\boldsymbol{x}}) \equiv \nabla^2 f(\bar{\boldsymbol{x}}) + \sum_{i \in I} \bar{u}_i \nabla^2 g_i(\bar{\boldsymbol{x}}) + \sum_{j=1}^{n} \bar{v}_j \nabla^2 h_j(\bar{\boldsymbol{x}}), \tag{B.24}$$

where $\nabla^2 f(\bar{\boldsymbol{x}})$, $\nabla^2 g_i(\bar{\boldsymbol{x}})$ *for all* $i \in I$ *and* $\nabla^2 h_j(\bar{\boldsymbol{x}})$ *for all* $j = 1, \ldots, n$ *are the Hessians of objective and constraint functions, respectively, at the point* $\bar{\boldsymbol{x}}$. *Define the cone*[8] C *by*

$$C = \left\{ \boldsymbol{d} \neq \boldsymbol{0} \colon \nabla g_i(\bar{\boldsymbol{x}})^T \boldsymbol{d} = 0 \text{ for all } i \in I^+, \nabla g_i(\bar{\boldsymbol{x}})^T \boldsymbol{d} \leq 0 \text{ for all } i \in I^0, \right.$$
$$\left. \nabla h_j(\bar{\boldsymbol{x}})^T \boldsymbol{d} = 0 \text{ for all } j = 1, \ldots, n \right\}. \tag{B.25}$$

If $\boldsymbol{d}^T \nabla^2 L(\bar{\boldsymbol{x}}) \boldsymbol{d} > 0$ *holds for all* $\boldsymbol{d} \in C$, *then the point* $\bar{\boldsymbol{x}}$ *is a strict local minimum to the problem* (B.14).

[7] see the formula (B.3) on the page 78

[8] a nonempty set $C \subset \mathbb{E}^N$ is said to be a *cone* with vertex zero if for all $\boldsymbol{x} \in C$ holds that $\lambda \boldsymbol{x} \in C$ for all $\lambda \geq 0$

APPENDIX C

Augmented Lagrangian Techniques

In this appendix, we will describe how to convert programming problems with equality and inequality constraints into an equivalent unconstrained problem. The advangate is that we can then use for solving the algorithms developed for unconstrained problems. In fact, there are two approches how to do that. The first approach is to use the *penalty functions* in which a penalty term is appended to the objective function to penalize any violation of the constraints. This method produces a sequence of solutions that are infeasible and the limit of that sequence gives the optimal solution to the original problem. Since this method produces infeasible points converging to the optimal feasible point, we also call this technique as the *exterior penalty function method*. The second approach is to use the *barrier functions* in which the barrier term is appended to the objective function in order to obviate generated feasible solutions to leave the feasible region. As in penalty function approach, in the limit this method gives an optimal solution to the original problem. Since this method generates feasible points converging to the optimal solution of the original problem, we also call this approach as the *interior penalty function method*. In following, we will focus to the penalty function approach, exact penalty functions and the augmented Lagrangian penalty function that is closely linked with the progressive hedging algorithm. The reader can find more information about barrier function methods as well as all proofs of presented theorems in [1].

C.1 Concept of Penalty Function

As we noticed above, the penalty function methods convert the programming problem with constraints into a unconstrained problem. The penalty term is added to the objective function to penalize any violation of constraints. In fact, the constraints of original

problem are incorporated to the penalty term. To make it clear, consider the following problem with equality and inequality constraints:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) \\ &\text{subject to} && g(\boldsymbol{x}) \leq 0, \\ & && h(\boldsymbol{x}) = 0, \\ & && \boldsymbol{x} \in X \subset \mathbb{E}^N. \end{aligned}$$

Now, we want to convert this problem into an equivalent problem in the form:

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) + \mu\alpha(\boldsymbol{x}) \\ &\text{subject to} && \boldsymbol{x} \in \mathbb{E}^N, \end{aligned}$$

where μ is the so-called penalty coefficient, $\mu > 0$ is a large number and $\alpha(\boldsymbol{x})$ is the penalty term. Intuitively, the optimal solution for the above problem must have the value of $\alpha(\boldsymbol{x})$ close to zero, since otherwise the penalty term will cause the large penalty. Thus, we are looking for the function $\alpha(\boldsymbol{x}) = \alpha_g(\boldsymbol{x}) + \alpha_h(\boldsymbol{x})$ such that the term $\alpha_g(\boldsymbol{x})$ will be zero for $g(\boldsymbol{x}) \leq 0$ and positive for $g(\boldsymbol{x}) > 0$ and the term $\alpha_h(\boldsymbol{x})$ will be zero for $h(\boldsymbol{x}) = 0$ and positive for $h(\boldsymbol{x}) \neq 0$. A suitable penalty functions $\alpha_g(\boldsymbol{x})$ and $\alpha_h(\boldsymbol{x})$ are

$$\alpha(\boldsymbol{x}) = \alpha_g(\boldsymbol{x}) + \alpha_h(\boldsymbol{x}) = \max\{0, g(\boldsymbol{x})\} + |h(\boldsymbol{x})|.$$

Indeed, if $g(\boldsymbol{x}) \leq 0$, then the penalty term $\alpha_g(\boldsymbol{x}) = 0$ and if $h(\boldsymbol{x}) = 0$, then $\alpha_h(\boldsymbol{x}) = 0$ and no penalty is realized. However, if $g(\boldsymbol{x}) > 0$ or $h(\boldsymbol{x}) \neq 0$, then $\alpha(\boldsymbol{x}) > 0$ and the penalty is realized.

In general, the penalty function must cause a positive penalty for infeasible solutions and no penalty for feasible solutions. For the problems with equality and inequality constraints in the form $g_i(\boldsymbol{x}) \leq 0$ and $h_j(\boldsymbol{x}) = 0$, where $i = 1, \ldots, m$, $j = 1, \ldots, n$, a suitable penalty function is in the form

$$\alpha(\boldsymbol{x}) = \sum_{i=1}^{m} \Phi\Big(g_i(\boldsymbol{x})\Big) + \sum_{j=1}^{n} \Psi\Big(h_j(\boldsymbol{x})\Big), \tag{C.1}$$

where functions Φ and Ψ are continuous functions with the following properties:

$$\Phi(\xi) = 0 \text{ for } \xi \leq 0, \qquad \Phi(\xi) > 0 \text{ for } \xi > 0 \tag{C.2}$$

$$\Psi(\xi) = 0 \text{ for } \xi = 0, \qquad \Psi(\xi) > 0 \text{ for } \xi \neq 0. \tag{C.3}$$

Thus, the typical forms of Φ and Ψ are

$$\begin{aligned} \Phi(\xi) &= \Big(\max\{0, \xi\}\Big)^p, \\ \Psi(\xi) &= |\xi|^p, \end{aligned}$$

where p is a positive integer. Finally, we can rewrite the penalty function $\alpha(\boldsymbol{x})$ as

$$\alpha(\boldsymbol{x}) = \sum_{i=1}^{m} \Big(\max\{0, g_i(\boldsymbol{x})\}\Big)^p + \sum_{j=1}^{n} |h_j(\boldsymbol{x})|^p. \tag{C.4}$$

C.2 Penalty Function Approach

In this section, we will introduce the crucial result that allows to use the penalty method as a method for solving problems with constraints.

Theorem C.1. *Consider the following constrained problem:*

$$\begin{array}{lll} \textit{minimize} & f(\boldsymbol{x}) & \\ \textit{subject to} & g_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m, \\ & h_j(\boldsymbol{x}) = 0, & j = 1, \ldots, n, \\ & \boldsymbol{x} \in X, & \end{array} \tag{C.5}$$

where f, g_i and h_j are continuous functions on $\mathbb{E}^N$ and X is a nonempty set in $\mathbb{E}^N$. Let $\alpha(\boldsymbol{x})$ be a continuous function given by (C.1) *satisfying* (C.2) *and* (C.3). *Assume that the problem has a feasible solution. Moreover, suppose that for each μ, there exists a solution $\boldsymbol{x}_\mu \in X$ to the problem to minimize $f(\boldsymbol{x}) + \mu\alpha(\boldsymbol{x})$ subject to $\boldsymbol{x} \in X$, and that $\{\boldsymbol{x}_\mu\}$ is contained in a compact subset of X. Then,*

$$\inf\{f(\boldsymbol{x}) : \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}, \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}, \boldsymbol{x} \in X\} = \sup_{\mu \geq 0}\{\theta(\mu)\} = \lim_{\mu \to \infty} \theta(\mu),$$

where $\boldsymbol{g}(\boldsymbol{x}) = \Big(g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x})\Big)^T$, $\boldsymbol{h}(\boldsymbol{x}) = \Big(h_1(\boldsymbol{x}), \ldots, h_n(\boldsymbol{x})\Big)^T$ *and*

$$\theta(\mu) = \inf\{f(\boldsymbol{x}) + \mu\alpha(\boldsymbol{x}) : \boldsymbol{x} \in X\} = f(\boldsymbol{x}_\mu) + \mu\alpha(\boldsymbol{x}_\mu).$$

Moreover, the limit $\bar{\boldsymbol{x}}$ of any convergent subsequence of $\{\boldsymbol{x}_\mu\}$ is an optimal solution to the original problem and $\mu\alpha(\boldsymbol{x}_\mu) \to 0$ as $\mu \to \infty$.

Thus, from the above theorem, by choosing the penalty parameter μ large enough, we can construct the solution $\boldsymbol{x}_\mu$ arbitrary close to the solution to the original problem. The popular method to solve the problem (C.5) is to solve a sequence of the problems

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) + \mu_i\alpha(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in X, \end{array}$$

where μ_i is an increasing sequence of penalty parameters. Then, the set $\{\boldsymbol{x}_{\mu_i}\}$ consists of infeasible solutions to the problem (C.5) and (by Theorem C.1) these solutions for a large enough penalty parameters μ_i converge to the optimal solution of the original problem (C.5). This also explains the name *exterior penalty function method*, since it generates the infeasible points from an exterior of the feasible set X. The above technique is described in the following algorithm for solving the problem (C.5).

Initialization. Set $\varepsilon > 0$ to be a termination parameter. Choose an initial point $\boldsymbol{x}_0$, an initial penalty parameter μ_0 and a parameter $\beta > 1$. Let $k = 0$ and go to the main part of the algorithm.

Main part.

1. Start with $\boldsymbol{x}_k$ and solve the problem

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) + \mu_k\alpha(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{x} \in X, \end{array}$$

Let $\boldsymbol{x}_{k+1}$ be an optimal solution for the above problem and continue with step 2 of the main part.

2. If $\mu_k \alpha(\boldsymbol{x}_{k+1}) < \varepsilon$, then stop, the point $\boldsymbol{x}_{k+1}$ is an approximate solution of the original problem with the penalty less than ε. Otherwise, set $\mu_{k+1} = \beta\mu_k$, $k = k + 1$ and go to the step 1 of the main part of the algorithm.

C.3 Exact Penalty Functions

For the penalty function methods described in Section C.1 and C.2, we have seen that we have to take the penalty parameter μ infinitely large if we want to reach the optimal solution to the original problem in the limit sense. This can also cause numerical difficulties and ill-conditioning effects. Naturally, in practise we can take the penalty parameter just finitely large. Thus, the question is whether we can construct a penalty function that reachs an exact optimal solution to the original problem for reasonable finite value of the penalty parameter μ. Such penalty functions are called *exact* penalty functions. We will present one example of a penalty function with this property as a particular case of (C.4) under additional assumptions and in the next section, we will present the concept of the augmented Lagrangian function that also safisties the mentioned property and plays the role in the progressive hedging algorithm.

Let us consider the following problem

$$\begin{array}{lll} \text{minimize} & f(\boldsymbol{x}) & \\ \text{subject to} & g_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m, \\ & h_j(\boldsymbol{x}) = 0, & j = 1, \ldots, n, \\ & \boldsymbol{x} \in X, & \end{array}$$

and its penalized objective function in the form

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + \mu \left(\sum_{i=1}^{m} \max\{0, g_i(\boldsymbol{x})\} + \sum_{j=1}^{n} |h_j(\boldsymbol{x})| \right). \tag{C.6}$$

In fact, its penalty part is in the form (C.4) with $p = 1$. Assume $\bar{\boldsymbol{x}}$ to be a KKT[1] point with Lagrangian multipliers $\bar{u}_i$, $i \in I = \{i \colon g_i(\bar{\boldsymbol{x}}) = 0\}$ and $\bar{v}_j$ for $j = 1, \ldots, n$. Moreover, suppose that the functions f and g_i for $i \in I$ are convex functions and h_i for $i = 1, \ldots, n$ are affine functions. Then, it can be shown that for finite $\mu \geq$ $\geq \max\{\bar{u}_i, i \in I, |\bar{v}_j|, j = 1, \ldots, n\}$ the point $\bar{\boldsymbol{x}}$ minimizes the penalized objective function $F(\boldsymbol{x})$ defined by (C.6).

C.4 Augmented Lagrangian Penalty Functions

As we discussed in the foregoing section, we are interested in penalty functions that can reach the optimal solution to the original problem for finite value of the penalty parameter. In this section, we will introduce the concept of the *augmented Lagrangian penalty function* (ALAG penalty function) that satisfies this property and in addition possess the property of being differentiable.

Let us start with the problem containing only equality constraints that we will later extend to the problem with equality and inequality constrains.

[1] a point satisfying the Karush–Kuhn–Tucker conditions, see Appendix B

C.4.1 Problems With Equality Constraints

Consider the following problem with equality constraints:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) \\ \text{subject to} & h_j(\boldsymbol{x}) = 0, \qquad j = 1, \ldots, n. \end{array}$$

If we take the quadratic penalty function (the general penalty function (C.4) with $p = 2$), we will need infinite value of the penalty parameter μ to obtain the solution to the original problem. However, try to shift the origin of the penalty term to the point $\boldsymbol{w} = (w_1, \ldots, w_n)$ and consider the penalized objective function for the problem with perturbations $\boldsymbol{w}$ of the right-hand sides of constraints,

$$f(\boldsymbol{x}) + \mu \sum_{j=1}^{n} \Big(h_j(\boldsymbol{x}) - w_j \Big)^2.$$

By the expansion of the quadratic term in the sum,

$$f(\boldsymbol{x}) + \mu \sum_{j=1}^{n} \left(h_j^2(\boldsymbol{x}) - 2h_j(\boldsymbol{x})w_j + w_j^2 \right).$$

Denote $v_j = -2\mu w_j$ and put out the constant term $\sum_{j=1}^{n} w_j^2$. Finally, we can write

$$F_{\mathrm{ALAG}}(\boldsymbol{x}, \boldsymbol{v}) = f(\boldsymbol{x}) + \sum_{j=1}^{n} v_j h_j(\boldsymbol{x}) + \mu \sum_{j=1}^{n} h_j^2(\boldsymbol{x}). \tag{C.7}$$

Note that (C.7) is the standard Lagrangian function augmented by the quadratic penalty term $\mu \sum_{j=1}^{n} h_j^2(\boldsymbol{x})$ which explains the name *augmented Lagrangian penalty function.*

The following theorem shows that the augmented Lagrangian penalty function is the exact penalty function.

Theorem C.2. *Consider the following problem*

$$\begin{array}{ll} \textit{minimize} & f(\boldsymbol{x}) \\ \textit{subject to} & h_j(\boldsymbol{x}) = 0, \qquad j = 1, \ldots, n. \end{array} \tag{C.8}$$

and let the KKT solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{v}})$ satisfy the second-order KKT sufficent condition for a local minimum (see Theorem B.15). Then, there exists a penalty parameter $\bar{\mu}$ such that for $\mu \geq \bar{\mu}$, the augmented Lagrangian penalty function $F_{ALAG}(\cdot, \bar{\boldsymbol{v}})$ also reachs a strict local minimum at the point $\bar{\boldsymbol{x}}$. Especially, in case that the function f is convex and the functions h_j are affine for all $j = 1, \ldots, n$, then any minimizing solution $\bar{\boldsymbol{x}}$ for the problem (C.8) *also minimizes $F_{ALAG}(\cdot, \bar{\boldsymbol{v}})$ for all $\mu \geq 0$.*

C.4.2 Problems With Equality and Inequality Constraints

In following, we will show how to generalize the foregoing theory for the problems including both equality and inequality constraints.

Consider the following problem

$$\begin{array}{lll} \text{minimize} & f(\boldsymbol{x}) & \\ \text{subject to} & g_i(\boldsymbol{x}) \leq 0, & i = 1, \ldots, m, \\ & h_j(\boldsymbol{x}) = 0, & j = 1, \ldots, n. \end{array} \tag{C.9}$$

To include the inequality constraints to the foregoing theory presented above, we need to rewrite equivalently all inequality constraints $g_i(\boldsymbol{x}) \leq 0$ for all $i = 1, \ldots, m$ as equality constraints. This is accomplished by adding the complementary variables s_i to all inequality constraints. Thus, we can transform the inequality constraints to the form

$$g_i(\boldsymbol{x}) + s_i^2 = 0$$

for all $i = 1, \ldots, m$, where obviously $s_i^2 \geq 0$.

Assume $\bar{\boldsymbol{x}}$ to be a KKT point for the problem (C.9) with optimal Lagrange multipliers $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ corresponding with the inequality and equality constraints, respectively. Suppose that *the strict complementary slackness conditions* $\bar{u}_i g_i(\boldsymbol{x}) = 0$ hold for all $i = 1, \ldots, m$ with $\bar{u}_i > 0$ for all $i \in I = \{i : g_i(\bar{\boldsymbol{x}}) = 0\}$. Further, assume that the second-order sufficient conditions (see Theorem B.15) hold for the point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{v}})$. Then, we can apply Theorem C.2 to the problem

$$\begin{array}{lll} \text{minimize} & f(\boldsymbol{x}) & \\ \text{subject to} & g_i(\boldsymbol{x}) + s_i^2 = 0, & i = 1, \ldots, m, \\ & h_j(\boldsymbol{x}) = 0, & j = 1, \ldots, n. \end{array}$$

All conditions of Theorem C.2 are satisfied at the point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{s}})$, where $s_i^2 = -g_i(\bar{\boldsymbol{x}})$ for all $i = 1, \ldots, m$. Thus, the point $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{s}})$ for the penalty parameter μ large enough is a strict local minimum for the following augmented Lagrangian penalty function at the point $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}})$:

$$F_{\text{ALAG}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} u_i \left(g_i(\boldsymbol{x}) + s_i^2 \right) + \\ + \sum_{j=1}^{n} v_j h_j(\boldsymbol{x}) + \mu \left(\sum_{i=1}^{m} \left(g_i(\boldsymbol{x}) + s_i^2 \right) + \sum_{j=1}^{n} h_j^2(\boldsymbol{x}) \right)$$

that can be rewritten to the more suitable form:

$$F_{\text{ALAG}}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}) = f(\boldsymbol{x}) + \mu \sum_{i=1}^{m} \left(g_i(\boldsymbol{x}) + s_i^2 + \frac{u_i}{2\mu} \right)^2 - \sum_{i=1}^{m} \frac{u_i^2}{4\mu} + \sum_{j=1}^{n} v_j h_j(\boldsymbol{x}) + \mu \sum_{j=1}^{n} h_j^2(\boldsymbol{x}).$$

APPENDIX D

What Is on the CD

The CD is attached to the thesis and contains

- the thesis in PDF[1] format

  ♣/`thesis.pdf`,

- the parallel implementation of the one-stage progressive hedging algorithm for illustrative example with two paraboloids (see Subsection 5.4.1, page 36)

  ♣/`OneStagePHAParaboloids/`,

- the parallel implementation of the two-stage progressive hedging algorithm for the farmer's problem (see Subsection 5.5.1, page 42)

  ♣/`TwoStagePHAFarmersProblem/`,

- GAMS source files for both scenarios s^1 and s^2 of the continuous casting process (see Section 6.4, page 55)

  ♣/`ContinuousCasting/`,

- the parallel implementation of the two-stage progressive hedging algorithm for the two-stage continuous casting process (see Sections 6.4 and 6.5, pages 55 and 58, respectively)

  ♣/`TwoStageContinuousCasting/`,

where ♣ denotes your CD-ROM drive.

[1] Portable Document Format