VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MATEMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

SPATIAL DECOMPOSITION FOR DIFFERENTIAL EQUATION CONSTRAINED STOCHASTIC PROGRAMS

PROSTOROVÁ DEKOMPOZICE ÚLOH STOCHASTICKÉHO PROGRAMOVÁNÍ S OMEZENÍMI VE TVARU DIFERENCIÁLNÍCH ROVNIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ZUZANA ŠABARTOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

RNDr. PAVEL POPELA, Ph.D.

BRNO 2012

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav matematiky
Akademický rok: 2011/2012

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Zuzana Šabartová

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:

Prostorová dekompozice úloh stochastického programování s omezeními ve tvaru diferenciálních rovnic

v anglickém jazyce:

Spatial Decomposition for Differential Equation Constrained Stochastic Programs

Stručná charakteristika problematiky úkolu:

Student si prohloubí znalosti problematiky modelů stochastického programování se zaměřením na problematiku úloh s omezeními ve tvaru diferenciálních rovnic. Zaměří se na možnosti docílení původních výsledků v oblasti dekompozice úloh. Téma práce navazuje na dlouhodobě rozvíjenou problematiku optimalizačních modelů zahrnujících omezení ve tvaru diferenciálních rovnic a náhodné koeficienty řešených ve spolupráci s dalšími ústavy (UPEI, EU) a fakultami (FAST). Pro efektivní modifikaci a implementaci modelu diplomant využije vhodné aproximační schéma a dekompoziční principy.

Cíle diplomové práce:

Vybraná třída modelů bude implementována a testována s cílem přispět k řešení náročných inženýrských úloh. Důraz bude kladen na diskretizační a dekompoziční přístupy.

Seznam odborné literatury:

P. Kall, S. W. Wallace. Stochastic Programming, Wiley and Sons, 1993.
J.R. Birge, F. Louveaux. Introduction to Stochastic Programming, Wiley and Sons, 1996.
I. M. Smith, D.V. Griffiths. Programming the Finite Element Method, Wiley and Sons, 1988.
E. Žampachová. Approximations in Stochastic Optimization and Their Applications, VUT FSI Brno, 2010.

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

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2011/2012.

V Brně, dne 26.10.2010

L.S.

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

Summary

Wide variety of optimum design problems in engineering leads to optimization models constrained by ordinary or partial differential equations (ODE or PDE). Numerical methods based on discretising domain are required to obtain a non-differential numerical description of the differential parts of constraints because the analytical solutions can be found only for simple problems. We chose the finite element method.

The real problems are often large-scale and exceed computational capacity. Hence, we employ the progressive hedging algorithm (PHA) - an efficient scenario decomposition method for solving scenario-based stochastic programs, which can be implemented in parallel to reduce the computing time. A modified PHA was used for an original concept of spatial decomposition based on the mesh created for approximation of differential equation constraints. The algorithm consists of a few main steps: solve our problem with a raw discretization, decompose it into overlapping parts of the domain, and solve it again iteratively by the PHA with a finer discretization - using values from the raw discretization as boundary conditions until a given accuracy is reached.

The spatial decomposition is applied to a basic test problem from the civil engineering area: design of beam cross section dimensions. The algorithms are implemented in GAMS software and finally results are evaluated with respect to a computational complexity and a length of overlap.

Key words

optimization, stochastic program, differential equation, finite element method, beam, progressive hedging algorithm, scenario decomposition, spatial decomposition, overlapping constraints

ŠABARTOVÁ, Z. *Spatial decomposition for differential equation constrained stochastic programs*, Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2012. 92 s. Vedoucí diplomové práce RNDr. Pavel Popela, Ph.D.

Abstrakt

Rozsáhlá třída inženýrských optimalizačních úloh vede na modely s omezeními ve tvaru obyčejných nebo parciálních diferenciálních rovnic (ODR nebo PDR). Protože diferenciálních rovnice je možné řešit analyticky jen v nejjednodušších případech, bylo k řešení použito numerických metod založených na diskretizaci oblasti. Zvolili jsme metodu konečných prvků, která umožňuje převod omezení ve tvaru diferenciálních rovnic na omezení ve tvaru soustavy lineárních rovnic.

Reálné problémy jsou často velmi rozsáhlé a přesahují dostupnou výpočetní kapacitu. Výpočetní čas lze snížit pomocí progressive hedging algoritmu (PHA), který umožňuje paralelní implementaci. PHA je efektivní scénářová dekompoziční metoda pro řešení scénářových stochastických úloh. Modifikovaný PHA byl využit pro původní přístup prostorové dekompozice. Aproximace diferenciálních rovnic v modelu problému je dosaženo pomocí diskretizace oblasti. Diskretizace je dále využita pro prostorovou dekompozici modelu. Algoritmus prostorové dekompozice se skládá z několika hlavních kroků: vyřešení problému s hrubou diskretizací, rozdělení oblasti problému do překrývajících se částí a iterační řešení pomocí PHA s jemnější diskretizací s využitím hodnot z hrubé diskretizace jako okrajových podmínek.

Prostorová dekompozice byla aplikována na základní testovací problém z oboru stavebního inženýrství, který se zabývá návrhem rozměrů průřezu nosníku. Algoritmus byl implementován v softwaru GAMS. Získané výsledky jsou zhodnoceny vzhledem k výpočetní náročnosti a délce překrytí.

Klíčová slova

optimalizace, úloha stochastického programování, diferenciální rovnice, metoda konečných prvků, nosník, progressive hedging algoritmus, scénářová dekompozice, prostorová dekompozice, omezení na překrytí

I declare that my master's thesis *Spatial decomposition for differential equation constrained stochastic programs* is the result of my own work according to the instructions of my supervisor RNDr. Pavel Popela, Ph.D., and using the sources duly listed in the bibliography.

Bc. Zuzana Šabartová

I would like to express my deep thanks to my supervisor RNDr. Pavel Popela, Ph.D. for supervising my master's thesis, his invaluable advice, comments, ideas and also motivation and support.

I would also like to thank to Ing. Mgr. Eva Žampachová, Ph.D. for early discussions of the results of her Ph.D. thesis and to Ing. Lubomír Klimeš for his advice which helped to improve my thesis.

Last but not least, my special thanks belong to my parents for their support during my studies and to Pavel for his patience, love and support.

Bc. Zuzana Šabartová

Contents

Chapter 1

Introduction

Wide variety of engineering problems is described by ordinary or partial differential equation (ODE or PDE) constrained models. We can find a closed-form solution only for simple ODEs or PDEs, hence we have to approximate their solution using numerical methods based on discretization of domain in most cases. The most common approximation techniques for solving these equations include the finite difference method, the finite volume method and also the finite element method that was chosen for our purpose. Then we can approximate the initial ODE/PDE constrained optimization problem by a mathematical program. There are a lot of well developed and tested methods for solving these deterministic programs.

In practice, some parameters and data from the given problem description are not given as fixed quantities but very often are random. Therefore, we obtain stochastic programs differing from the deterministic programs mentioned above.

The combination of these two areas leads in many cases to very large-scale ODE/PDE constrained stochastic optimization problems with hundreds of variables or/and equations. Solving these problems is difficult and can exceed the computational capacity. Hence, it is desirable to employ some decomposition techniques.

There are several ways how to decompose optimization problems and we have developed one completely new approach of spatial decomposition suitable for ODE/PDE constrained problems. Our original approach is based on the progressive hedging algorithm (PHA) allowing the scenario decomposition for scenario-based stochastic programs. The algorithm can be implemented in parallel to reduce the computing time and the scale of problems and can be used for deterministic and stochastic programs too. There are some other decomposition algorithms but we used the PHA because there are some experience with this algorithm on our department.

The spatial decomposition is presented on a particular problem from the civil engineering area in the Chapter 6. Our goal is to find an optimum design of beam cross section dimensions.

To be able to create the appropriate model we need to have a basic information about modeling. We obtained the stochastic programming problem so we have to get rid of uncertainties. Hence, we listed some deterministic reformulations. Then, the implementation in GAMS software can be made. All mentioned concepts together with one short solved example for better understanding could be find in the Chapter 3. One more difficulty comes with ODE constraints in our model, we have to approximate them by the finite element method described in the Chapter 4.

Finally, we can involve the spatial decomposition based on the penalty functions approach explained in the Appendix C. The main part of the algorithm comes from the progressive hedging algorithm and other decomposition techniques listed in the Chapter 5. There the progressive hedging algorithm is presented on a quite simple example. The spatial decomposition was worked out for two deterministic reformulations in GAMS (an example of a source code can be found in the Appendix D) and the results were evaluated with respect to a computational complexity. We must remember that the PHA converges to the optimal solution only for programs with a convex objective function and a convex feasible set what is not our case. But we are able to determine a pretty good starting point for the algorithm to provide a convergence in computations.

The research was supported by FME BUT projects no. FSI-J-11-7 "Optimalizace a numerické modelování úloh s fázovými a strukturálními přeměnami" and no. FSI-J-12-22 "Aplikace metod numerického modelování a optimalizace v inženýrských úlohách se změnou skupenství struktury". The thesis has been inspired by the problems solved in a research plan from MŠMT of the Czech Republic no. MSM0021630519.

Chapter 2

Aims of master's thesis

The main aims of this master's thesis can be divided as follows:

1. We will introduce main ideas of optimization with focus on stochastic programming in which decisions are taken under an uncertainty modeled by random variables. The uncertainties have to be removed by a deterministic reformulation. Therefore, the list of these reformulations will be given. The ideas of stochastic programming will be extended to two-stage stochastic programs. The most of the mentioned notion and concepts will be illustrated on a simple solved problem.

2. Since we are focused on the differential equation constrained stochastic programs, we need a numerical method to treat up the derivatives. The finite element method has been chosen. Hence, we will give a basic information about this method and its main steps.

3. Mathematical programs modeling real problems are usually large-scale. Therefore, a decomposition technique allowing a parallel implementation is required. We will give the basic insight to scenario decomposition methods focusing mainly on the progressive hedging algorithm (PHA). The progressive hedging algorithm forms the basis of the original spatial decomposition approach. A simple example solved by the PHA is included for better understanding of this algorithm. Further, we will focus on our concept of the spatial decomposition and we will evaluate the advantages of the parallel implementation.

4. Last but not least, the foregoing knowledge will be applied to a particular test problem from the area of civil engineering. The finite element method will be used for the approximation of the derivatives in the problem formulation. The spatial decomposition will be implemented for two deterministic reformulations of this problem. Finally, the results will be evaluated and discussed.

Chapter 3

Optimization

3.1 Motivation

Optimization problems arise in many different disciplines. Optimization plainly dominates the design, operation and planning of engineering systems. A bridge is designed by minimizing its building costs but maintaining appropriate security standards. Railway systems are expanded to minimize building and operation costs while operation and security standards must hold. Analogously, if you decide to optimize an electric energy system power demands has to be supplied at minimum costs. Note that this section is based on [1].

Optimization is "the science of the best" in the sense that it helps us to make a decision which is not only respectable, but the best decision subject to certain constraints describing the domain where the decision has to be taken. *Mathematical programming models* provide the appropriate framework for these optimization decisions in a precise and formal manner.

The objective to be minimized (or maximized) is expressed as a real-valued mathematical function named as the *objective function*. This function depends on one or several *decision variables* whose optimal values are sought.

The restrictions that have to be satisfied define what is denominated the *feasibility set* of the problem. This set should include many possible decisions which make sense for the optimization problem. The feasibility region is formally defined through equality and/or inequality conditions and we called them as *constraints* of the problem.

Mathematical programming problems are classified depending on the type of variables and the objective function and the functions used for the constraints. If the variables are continuous and both the objective function and the constraints are linear, the problem is called as *linear programing problem (LP)*. If any of the variables is integer or binary, while the constraints and the objective function are both linear, the problem is denominated *mixed-integer linear programming problem (MILP)*. Analogously, if the objective function or any of the constraints is nonlinear and all variables are continuous, the problem is the *nonlinear programming problem (NLP)* and so on.

Linear programming problems are routinely solved even if they involve hundreds or thousands of variables and constraints. Nonlinear problems are solved if they meet certain regularity conditions related to the mathematical concept of convexity. Mixed-integer problems are generally hard to solve and can be numerically intractable.

To be able to solve an optimization problem, we have to create the model of reality first. We identify activities which we can control and influence. Each such activity is associated with a decision variable whose value is to be decided. The remaining quantities are constants in the problem. We create a real-valued objective function of the variable values. The quantity is minimized or maximized depending on our goal. The activity levels cannot be arbitrarily large, it is usually associated with some resources or demands. So we create constraints. We can also meet with some uncertainties in our model, then we are speaking about the *stochastic programming problem (SP)*.

The modelling process comes with some difficulties. The communication can be difficult because two groups speak different languages in terms of describing the problem. The optimization problem quite often has uncertainties in the data, which moreover are not always easy to collect or to quantify. There is often a conflict between problem solvability and problem realism. We can get thanks some optimization algorithms an optimal value and an optimal solution, if they exist. This result is then interpreted and evaluated, which may lead to alterations of the model, and to questions regarding the applicability of the optimal solution. The optimization model can also be altered in order to answer sensitivity analysis type questions concerning the effect of small variations in data. The final problems are connected to the interpretation of the result. The result has to make sense to those who want to use the solution. It must be possible to transfer the solution back into the world where the problem came from.

The forming a good optimization model is basically a difficult process. It is often possible to construct more than one form of an mathematical model that represents the same problem equally accurately, and the computational complexity can differ between them. A well-designed model is crucial for success of the application.

3.2 History of optimization

Several branches of mathematics are associated with the optimization: analysis, topology, algebra, discrete mathematics, etc. Optimization is also sometimes called as mathematical programming (G. B. Dantzig [10], 1947-1949). The term *program* has nothing to do with a computer program, it should be understood as a decision program, that is a strategy or decision rule. This section is based mainly on the literature referenced in [1].

The history of optimization is quite long. Many geometrical or mechanical problems, that Archimedes, Euclides and others formulated and also solved, are optimization problems. We can mention, for instance, the problem of maximizing the volume of a three-dimensional object built from a two-dimensional sheet of metal with a given area.

Many years later some other famous mathematicians like D. Bernoulli, J. L. Lagrange, L. Euler or K. Weierstrass developed variational calculus by studying problems in applied physics such as how to find the best trajectory for a flying object. The notion of optimality and how to characterize an optimal solution was developed at the same time.

The fastest development of optimization occurred in the Second World War, when the US and British military commands hired scientists from several disciplines to try to solve complex problems regarding the best way to construct convoys in order to avoid or protect the cargo ships from German submarines and how to best cover the British isles with radars and so on.

Among the scientists that took part in the Second World War we find several researchers in mathematics, physics, and economics, who contributed greatly to the foun-

dations of the optimization as we now know it. We mention only few of them here. G. B. Dantzig invented the simplex method for solving the linear optimization problems, as well as the whole machinery of modelling such problems. The knowledge of duality came from J. von Neumann. A large part of the duality theory was developed in collaboration with the mathematician A. W. Tucker.

The stochastic programming, where uncertain parameters occurred, has been studied since 1955, when G. B. Dantzig introduced a concept of the linear programming under an uncertainty. The theory for stochastic programs is much more complicated and it is harder to find some generalized laws for a wide class of problems. The first important theoretical results were published in the sixties by pioneers of the stochastic programming A. Madansky, R. Wets, A. Prékopa, etc. Lately, the first monographs appeared (e.g. by P. Kall). The seventies brought deep theoretical results (e.g. from R. T. Rockafellar). A remarkable progress was made in the eighties with development of algorithms and the multistage stochastic programming (J. R. Birge [5]). New areas of interest were integer stochastic programming problems and stochastic programming networks (S. W. Wallace [19]). Usually we obtain large scale problems with hundreds of variables so we need some decomposition techniques to be able to solve our problems in a finite time. In the nineties, the question related to the availability of modelling and algorithmic tools and parallel implementation of algorithms has become more an more important.

3.3 Key concepts of mathematical analysis

The analysis of optimization problems and related algorithms requires the basic understanding of the multidimensional analysis and other branches of mathematics. Here we only give the essential definitions, and basic facts that we will use in subsequent chapters.

Definition 3.3.1. A set $S \subseteq \mathbb{R}^n$ is called *convex* if for each $\mathbf{x}^1, \mathbf{x}^2 \in S$, the point

$$\lambda \mathbf{x}^1 + (1 - \lambda)\mathbf{x}^2$$

for $\forall \lambda \in (0, 1)$ belongs to S. This says that all points on a line connecting two points in the set are in the set. A set is convex if, from everywhere in S, all other points of S are visible.

Theorem 3.3.1. *Suppose that S_k, $k \in K$, is any collection of convex sets. Then, their intersection*

$$S = \bigcap_{k \in K} S_k$$

is convex set too.

Proof. Assume that $\mathbf{x}^1$ and $\mathbf{x}^2$ belong to S. Then $\mathbf{x}^1 \in S_k$ and $\mathbf{x}^2 \in S_k$ for all $k \in K$. Take $\lambda \in (0, 1)$. By the convexity of the sets S_k, $\lambda \mathbf{x}^1 + (1 - \lambda)\mathbf{x}^2 \in S_k$ for all $k \in K$. So,

$$\lambda \mathbf{x}^1 + (1 - \lambda)\mathbf{x}^2 \in \bigcap_{k \in K} S_k = S.$$

□

Definition 3.3.2. A function $f : S \to \mathbb{R}$, where S is a convex subset of $\mathbb{R}^n$, is *convex* if for any $\mathbf{x}_1, \mathbf{x}_2 \in S$ and any $\lambda \in (0, 1)$

$$f(\lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda) f(\mathbf{x}_2)$$

holds.

In other words, a function is convex if the function between two arbitrary points is lower[1] or equal as the straight line between these two points.

Definition 3.3.3. Consider $S \subseteq \mathbb{R}^n$ is a nonempty set and $f : \mathbb{R}^n \to \mathbb{R}$. We say that $\mathbf{x}_{\min} \in S$ is a *global minimum* of f over S if f attains its lowest value over S at $\mathbf{x}_{\min}$, i.e.,

$$f(\mathbf{x}_{\min}) \leq f(\mathbf{x}), \ \forall \mathbf{x} \in S.$$

In the following text we will use the *norm*, or the *length* of a vector $\mathbf{v} \in \mathbb{R}^n$ with the following meaning

$$\|\mathbf{v}\| = \sqrt{(\mathbf{v}, \mathbf{v})} = \sqrt{\sum_{i=1}^{n} v_i^2}.$$

We can use some other norms of course, but this is the most common one. Thanks to norm we can introduce the open Euclidean ball with radius ε centered at $\mathbf{x}$ as

$$B_\varepsilon(\mathbf{x}) = \{\mathbf{y} \in \ \mathbb{R}^n \mid \ \|\mathbf{y} - \mathbf{x}\| \leq \varepsilon\}.$$

We can use the open ball for the following definition.

Definition 3.3.4. Consider $S \subseteq \mathbb{R}^n$ is a nonempty set and $f : \mathbb{R}^n \to \mathbb{R}$. We say that $\mathbf{x}_{\min} \in S$ is a *local minimum* of f over S if

$$\exists \varepsilon > 0 \text{ such that } f(\mathbf{x}_{\min}) \leq f(\mathbf{x}), \ \forall \mathbf{x} \in S \cap B_\varepsilon(\mathbf{x}_{\min}).$$

Note that a global minimum in particular is a local minimum. When is a local minimum the global one? This question is resolved in the case of convex problems by the following fundamental theorem.

Theorem 3.3.1 (Fundamental theorem of global optimality)**.** *Consider $S \subseteq \mathbb{R}^n$ is a nonempty set and $f : \mathbb{R}^n \to \mathbb{R}$, where S is a convex set and f is convex on S. Then, every local minimum of f over S is also the global minimum.*

Proof. Suppose that $\mathbf{x}_{\min}$ is a local minimum but not the global one, while $\mathbf{x}$ is the global minimum. Then $f(\mathbf{x}) < f(\mathbf{x}_{\min})$. By the convexity of S and f,

$$\lambda \mathbf{x} + \ (1 - \ \lambda)\mathbf{x}_{\min} \in \ S,$$

$$f(\lambda \mathbf{x} + (1 - \lambda)\mathbf{x}_{\min}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{x}_{\min}) < f(\mathbf{x}_{\min})$$

for each $\lambda \in (0, 1)$. Choosing $\lambda > 0$ small enough leads to a contradiction to the local optimality of $\mathbf{x}_{\min}$. □

We can image from the proof design how it works. If $\mathbf{x}_{\min}$ is a local minimum, then f cannot go down-hill from $\mathbf{x}_{\min}$ in any direction, but if f has in $\mathbf{x}$ a lower value, then f has to go down-hill sooner or later. No convex function can have this shape.

This amount of theory should be enough for this moment, we will add some other definitions, theorems and some notions later.

[1] Word lower should be understood in the sense of the comparison between the y-coordinates of the respective function at the same coordinates.

3.4 Deterministic programming

We have already mentioned what the mathematical programming is about. In this section we would like to describe deterministic programs. *Deterministic programs* are mathematical programs for which all data[2] is deterministic, i.e., fully known.

A mathematical program was only mentioned but we have to formulate formally the general mathematical programming problem. A large class of situations involving optimization can be expressed in the following form.

Definition 3.4.1. A general *mathematical programming problem* is defined as

$$\begin{aligned} &\min_{\mathbf{x}} \ f(\mathbf{x}), \\ &\text{s.t. } \ \mathbf{x} \in C, \end{aligned} \tag{3.1}$$

where $\mathbf{x}$ is a vector of decision variables, $C \subseteq \mathbb{R}^n$ is a feasible set, $n \in \mathbb{N}$ and $f : C \to \mathbb{R}$ is an objective function to be minimized (eventually maximized). The feasible set C is determined by equality or inequality constraints

$$C = \{\mathbf{x} \mid \mathbf{g}(\mathbf{x}) \leq \mathbf{0}, \mathbf{h}(\mathbf{x}) = \mathbf{0}, \mathbf{x} \in X\},$$

where $X \subseteq \mathbb{R}^n$.

Sometimes we can obtain more than one objective function to be minimized or maximized. For example, we want to maximize a rigidity of a beam that we are designing and at same time minimize its weight. These types of problems are difficult to handle because the objective functions are often contradictory.

One possibility how to labor with more functions is to assign weights to each objective function depending on their relative importance and then define a composite objective function as a weighted sum of all these functions, as follows:

$$f(\mathbf{x}) = w_1 f_1(\mathbf{x}) + \cdots + w_N f_N(\mathbf{x}), \tag{3.2}$$

where $w_1, \ldots, w_N$ are suitable weighting factors and $N \in \mathbb{N}$, N is finite. The success of the method clearly depends on clever choice of these weighting factors.

Another possibility is to select the most important goal as the single objective function and treat the others as constraints with reasonable limiting values. Detailed information about a multiple criteria optimization can be found in [33].

The methods for solving the general form of the optimization problem require a considerable numerical effort. More efficient methods are available for certain special forms of the general problem. For this purpose, the optimization problems are usually classified into the following types.

Definition 3.4.2. An *unconstrained problem* is defined as

$$\min_{\mathbf{x}} f(\mathbf{x}),$$

where $\mathbf{x}$ is a vector of decision variables and $f : \mathbb{R} \to \mathbb{R}$ is an objective function.

[2]By the word data we mentioned parameters and coefficients of the program.

These problems have an objective function but no constraints. Obviously the objective function is nonlinear, because the minimum of an unconstrained linear objective function is $-\infty$.

Many real world situations can be modeled as linear programs, it means that we have a linear objective function and constraints are linear too.

Definition 3.4.3. A *linear programming problem* in the *standard form* is a problem of the form

$$\begin{aligned} \min_{\mathbf{x}} \ & \mathbf{c}^{\mathrm{T}}\mathbf{x}, \\ \text{s.t.} \ & \mathbb{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \leq \mathbf{0}^n, \end{aligned} \tag{3.3}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}^n$ and $\mathbf{c}^{\mathrm{T}}$ is transposed $\mathbf{c}$, $\mathbb{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$.

Each linear program can be transformed into the standard form by following few steps:

- Express the objective function in the minimization form - the minimization is equivalent with the maximization of the objective function multiplied by (-1).
- Transform all the constraints into equality constraints by adding additional variables. We also require non-negative right-hand sides of constraints.
- Transform any unrestricted and non-positive variables into non-negative ones - by splitting it into two parts, the first is positive and the second is negative.

The standard form is the form that the simplex method requires. The simplex method is efficient and robust algorithm for solving these problems. We have to transform a solution gained by the simplex method back to the origin variables.

If the objective function is quadratic and all constraint functions are linear functions of decision variables, the problem is called a *quadratic programming problem*. The problem from the progressive hedging algorithm formulation presented in the fourth chapter is an example of a quadratic optimization problem.

General constrained optimization problems, in which one or more functions are nonlinear, are called *nonlinear programming problems*.

Because of the straight relation of mathematical programs to the underlying programs in the stochastic programming we define a parametric mathematical program.

Definition 3.4.4. A *parametric mathematical program* is defined as

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}, \mathbf{a}), \\ \text{s.t.} \ & \mathbf{x} \in C(\mathbf{a}), \end{aligned} \tag{3.4}$$

where $\mathbf{a} \in \mathbb{R}^k$ is a constant parameter and $C(\mathbf{a})$ is the feasible set.

Some other types of mathematical programs can be found, for example if all variables are integer we are talking about integer programs, if only some variables are integer then it is the mixed-integer programming, et cetera.

There are many theoretical results in mathematical programming theory and many methods for solving different problems were described. Their summary can be find for instance in [1].

3.5 Stochastic programming

In the previous section we discussed deterministic programs. This section covers a case when the decisions must be made under an *uncertainty* so the model parameters are not completely known. These optimization applications can be modeled by the stochastic programming. Model parameters of a problem can be considered uncertain and are thus represented as *random variables*. So we need to introduce the basic concepts of probability theory.

We can model the uncertainty by an experiment, the result of an experiment is called its outcome. In general, we cannot predict with a certainty the outcome of an experiment in advance of its completion, we can only list the collection of possible outcomes.

Definition 3.5.1. The set of all possible outcomes of an experiment is called the *sample space* and is denoted by Ω.

We think of events as subsets of the sample space Ω. Many common situations require that the collection of events has to be closed under the operation of taking countable unions. Any collection of subsets of Ω with these properties is called a σ*-field*.

Definition 3.5.2. A collection $\mathscr{F}$ of subsets of Ω is called a σ*-field* if it satisfies the following conditions[3]:

(a) $\emptyset \in \mathscr{F}$,

(b) if $A_1, A_2, \ldots \in \mathscr{F}$ then $\bigcup_{i=1}^{\infty} A_i \in \mathscr{F}$,

(c) if $A \in \mathscr{F}$ then $A^C \in \mathscr{F}$.

We wish to be able to discuss the likelihoods of the occurrence of events. So we define a probability function $\mathbb{P}$ applied to the set of events. Likelihoods of the members of $\mathscr{F}$ is called a *probability measure*.

Definition 3.5.3. A *probability measure* $\mathbb{P}$ on $(\Omega, \mathscr{F})$ is a function $\mathbb{P} : \mathscr{F} \to [0, 1]$ satisfying

(a) $\mathbb{P}(\emptyset) = 0$, $\mathbb{P}(\Omega) = 1$,

(b) if $A_1, A_2, \ldots$ is a collection of disjoint members of $\mathscr{F}$, in that $A_i \cap A_j = \emptyset$ for all pairs i, j satisfying $i \neq j$, then

$$\mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mathbb{P}(A_i).$$

The triple $(\Omega, \mathscr{F}, \mathbb{P})$, comprising a set Ω, a σ-field $\mathscr{F}$ of subsets of Ω and a probability measure $\mathbb{P}$ on $(\Omega, \mathscr{F})$, is called a probability space. Furthermore, if $\mathbb{P}(A) = 1$, we say that A occurs *almost surely*, the abbreviation a.s. is often used.

We shall not always be interested in an experiment itself, but rather in some consequence of its random outcome. These consequences may be thought as a function which map Ω into the real line $\mathbb{R}$, and these functions are called *random variables*.

Definition 3.5.4. A *random variable* is a function $\xi : \Omega \to \mathbb{R}$ with property that $\{\omega \in \Omega : \xi(\omega) \leq x\} \in \mathscr{F}$ for each $x \in \mathbb{R}$. Such a function is said to be $\mathscr{F}$*-measurable*.

[3] A^C is complement of A, event that no outcome in A occurs.

Every random variable has a *distribution function* which is very important and useful.

Definition 3.5.5. The *distribution function* of a random variable ξ is the function $F : \mathbb{R} \to [0, 1]$ given by $F(x) = \mathbb{P}(\xi(\omega) \leq x)$.

Definition 3.5.6. A *mean value*, or an *expectation*, or an *expected value* of the random variable $\xi : \Omega \to \mathbb{R}$ is defined to be

$$\mathbb{E}(\xi) = \int_{\Omega} \xi(\omega) \mathrm{d}\mathbb{P}(\omega), \tag{3.5}$$

where the integral converges absolutely[4]. The expectation for discrete random variable[5] ξ is given by

$$\mathbb{E}(\xi) = \sum_i \xi_i p_i,$$

where $p_i = \mathbb{P}\{\xi = \xi_i\}$ and the series converges absolutely[6].

Definition 3.5.7. A *variance* (also is called as the *2nd central moment*) of the random variable $\xi : \Omega \to \mathbb{R}$ is defined to be

$$\mathrm{var}(\xi) = \mathbb{E}(\xi - \mathbb{E}(\xi))^2.$$

More detailed information, further definitions and concepts from the probability theory can be found in [16].

Now we have defined the uncertainty what is the main ingredient in many decision problems. So we can define problems where the model parameters are not completely known.

Stochastic programming (SP) uses approach based on probabilistic models of the uncertainty. The objective functions and the constraints of the corresponding mathematical programming model can be defined by averaging possible outcomes or considering probabilities of events.

The first step to obtain stochastic program is the formulation of an underlying program. This can be done easily from the parametric mathematical program by replacing some constant parameters by random variables in (3.4).

Definition 3.5.8. An *underlying program (UP)* is defined as

$$\begin{aligned} &\min_{\mathbf{x}} \ f(\mathbf{x}, \omega), \\ &\text{s.t.} \ \ \mathbf{x} \in C(\omega), \end{aligned} \tag{3.6}$$

where $\omega \in \Omega$ is a random vector element.

The random data is usually realized by a finite number of parameters. Therefore, the objective function is given as $f(\mathbf{x}, \omega) = f(\mathbf{x}, \boldsymbol{\xi}(\omega))$, where $\boldsymbol{\xi}(\omega) : \Omega \to \mathbb{R}^K$ is a finite dimensional random vector defined on probability space $(\Omega, \mathscr{F}, \mathbb{P})$ and $f(\mathbf{x}, \boldsymbol{\xi})$ is a function

[4]The integral $\int_A f(x)\mathrm{d}x$ of a real or complex-valued function converges absolutely if $\int_A |f(x)|\,\mathrm{d}x < \infty$.

[5]The random variable ξ is called *discrete* if it takes values in some countable subset $\{\xi_1, \xi_2, \dots\}$ of $\mathbb{R}$.

[6]A real or complex series $\sum_{n=0}^{\infty} a_n$ converges abolutely if $\sum_{n=0}^{\infty} |a_n| < \infty$.

of two vector variables $\mathbf{x}$ and $\boldsymbol{\xi}$. Realization of $\boldsymbol{\xi}$ is $\boldsymbol{\xi}(\omega_s)$ for each $\omega_s \in \Omega$ and we will use the notation $\boldsymbol{\xi}_s$ for this realization.

An important question is how to work with the uncertainty or in other words the randomness of parameters. The program (3.6) is not well defined, because we do not know what is the meaning of the minima until we observe the realizations of $\boldsymbol{\xi}$. We need to assign *the deterministic reformulation* to (3.6) to be able to solve this program correctly. We deal with the probability distribution instead of constant parameters in case of deterministic programming. Let us assume that the probability distribution of $\boldsymbol{\xi}$ is completely known.

Deterministic reformulations

We will now present different kinds of deterministic reformulation of the underlying program described in (3.6), that correctly interpret random parameters.

The main question is when the decision $\mathbf{x}$ has to be taken. Whether before the random parameters $\boldsymbol{\xi}$ are observed or after the observations $\boldsymbol{\xi}_s$ are known. When the decision $\mathbf{x}$ is made after observing the randomness $\boldsymbol{\xi}$, this case is called the *wait-and-see (WS)* approach. This approach is applicable when we have the perfect information about the future. In this case, we can modify our decision by the observation, that's why the decision $\mathbf{x}(\omega)$ and also the objective function $f(\mathbf{x}(\omega), \boldsymbol{\xi}(\omega))$ are random variables. This approach has its importance specifically for long-term planning.

But the decision makers must often take a decision before the observations of $\boldsymbol{\xi}$ are known. Therefore, the decision $\mathbf{x}$ must be the same for any future realization of $\boldsymbol{\xi} \in \Xi$, where Ξ is the space of all possible realizations of $\boldsymbol{\xi}$. We usually call this approach as *here-and-now (HN)* approach in stochastic programming.

Several approaches of deterministic equivalents of the objective function and of the feasible set can be done. We can divide the equivalents into two classes, the equivalents of the objective function and equivalents of the feasible set. Combining these two classes will result in the deterministic equivalents of (3.6), some typical deterministic equivalents are listed further. All discussed equivalents define one-stage stochastic program, the structure of equivalents is even simpler because the randomness can enter only the objective or the feasible set. The used notion was taken from [26].

We denote the optimal objective function values for any deterministic reformulation $\bullet$ as $z^{\bullet}_{\min}$ and optimal decision as $\mathbf{x}^{\bullet}_{\min}$. We assume that the expected value is taken with respect to a known probability distribution and that $\mathbb{E}(\boldsymbol{\xi})$ and $\mathbb{E}(f(\mathbf{x}, \boldsymbol{\xi}))$ exist and are well defined.

Definition 3.5.9. *Wait-and-see (WS) deterministic reformulation* of the underlying program (3.6) is defined by

$$\mathbb{E}_{\boldsymbol{\xi}}(f(\mathbf{x}^{\mathrm{WS}}(\boldsymbol{\xi}), \boldsymbol{\xi})), \tag{3.7}$$

where

$$\begin{aligned} f(\mathbf{x}^{\mathrm{WS}}(\boldsymbol{\xi}), \boldsymbol{\xi}) = \min_{\mathbf{x}(\boldsymbol{\xi})} \; & f(\mathbf{x}(\boldsymbol{\xi}), \boldsymbol{\xi}), \\ \text{s.t. } & \mathbf{x}(\boldsymbol{\xi}) \in C(\boldsymbol{\xi}), \forall \boldsymbol{\xi} \in \Xi. \end{aligned} \tag{3.8}$$

Its optimal value is $z^{\mathrm{WS}}_{\min}$ and the optimal solution is denoted by $\mathbf{x}^{\mathrm{WS}}_{\min}$. Unfortunately, finding the WS solution may be impossible if the information about the future is not available. Therefore, the HN approach is usually used and several HN deterministic reformulations are commonly used instead of the WS equivalent.

Definition 3.5.10. *Individual scenario (IS) deterministic reformulation* of the underlying program (3.6) is defined as

$$\begin{aligned} &\min_{\mathbf{x}} \ f(\mathbf{x}, \boldsymbol{\xi}_s), \\ &\text{s.t. } \ \mathbf{x} \in C(\boldsymbol{\xi}_s), \end{aligned} \tag{3.9}$$

where $\boldsymbol{\xi}_s \in \Xi$ is a specified individual scenario. Denote its optimal value and solution by $z^{\text{IS}}_{\min}$ and $\mathbf{x}^{\text{IS}}_{\min}$.

This reformulation is based on the idea that the random parameters in the (3.6) are replaced by a typical realization $\boldsymbol{\xi}_s$ called a *scenario*. It is useful when we have a recommendation from the experts that some scenario is a typical realization of $\boldsymbol{\xi}$.

Another frequently used reformulation is obtained when we remove the uncertainty by taking its expected value.

Definition 3.5.11. *Expected value (EV) deterministic reformulation* of the underlying program (3.6) is defined as

$$\begin{aligned} &\min_{\mathbf{x}} \ f(\mathbf{x}, \mathbb{E}(\boldsymbol{\xi})), \\ &\text{s.t. } \ \mathbf{x} \in C(\mathbb{E}(\boldsymbol{\xi})), \end{aligned} \tag{3.10}$$

where $\mathbb{E}(\boldsymbol{\xi})$ is the expected value of $\boldsymbol{\xi}$, see (3.5) for the definition of expected value. The optimal value is denoted $z^{\text{EV}}_{\min}$ and the optimal solution as $\mathbf{x}^{\text{EV}}_{\min}$.

This program is useful for initial studies of applications of stochastic programming, but it often leads to solution with low reliability so it is always unacceptable for the users.

Further we could want to measure how good is the solution $\mathbf{x}^{\text{EV}}_{\min}$ of EV deterministic reformulation for the underlying objective function, so we define the following characteristic.

Definition 3.5.12. Consider the EV deterministic reformulation with optimal solution $\mathbf{x}^{\text{EV}}_{\min}$. We define the *expected objective function value for the optimal solution of the expected value deterministic reformulation (EEV)* as

$$\text{EEV} = \mathbb{E}_{\boldsymbol{\xi}}(f(\mathbf{x}^{\text{EV}}_{\min}, \boldsymbol{\xi})). \tag{3.11}$$

The EEV characteristic can be used to measure whether $z^{\text{EV}}_{\min}$ looks realistic by computing the difference

$$\text{EEV} - z^{\text{EV}}_{\min}$$

between the optimistic forecasted objective function value $z^{\text{EV}}_{\min}$ and the true average cost computed by the EEV.

We can define one more deterministic equivalent using the expected value incorporated in the objective function.

Definition 3.5.13. *Expected objective (EO) deterministic reformulation* of the underlying program (3.6) is defined as

$$\begin{aligned} &\min_{\mathbf{x}} \ \mathbb{E}_{\boldsymbol{\xi}}(f(\mathbf{x}, \boldsymbol{\xi})), \\ &\text{s.t. } \ \mathbf{x} \in C(\boldsymbol{\xi}) \text{ a.s.} \end{aligned} \tag{3.12}$$

and we denote the minimal objective function value as $z^{\text{EO}}_{\min}$ and the optimal solution as $\mathbf{x}^{\text{EO}}_{\min}$.

Between the EO and the EV solution a relation can be found, this relation is called *Jensen's inequality* (1906).

Theorem 3.5.1 (Jensen's inequality)**.** *For function $f(\mathbf{x}, \boldsymbol{\xi})$, which is convex in $\boldsymbol{\xi}$, the inequality $\mathbb{E}(f(\mathbf{x}, \boldsymbol{\xi})) \geq f(\mathbf{x}, \mathbb{E}(\boldsymbol{\xi}))$ is satisfied.*

It seems reasonable to compare the optimal values for different equivalents. Therefore, we define the following value that measures a relation between the EEV and the optimal value of the EO objective function.

Definition 3.5.14. The *value of stochastic solution (VSS)* is defined as

$$\mathrm{VSS} = \mathrm{EEV} - z_{\min}^{\mathrm{EO}}. \tag{3.13}$$

The VSS characteristic measures how much can be saved when the true HN approach is used instead of the EV approach. It expresses how suitable is to use the EV approach instead of the EO approach and also how many could be gained by solving the EO program instead of the simpler EV program. Unfortunately we have to compute the EO solution and the EVV characteristic. A small value of the VSS means that the approximation of the stochastic program by the EV program is a good one.

In the similar way, we try to find how to compare optimal solutions of the WS and the HN programs. We consider the EO program as a suitable representative of the class of HN deterministic reformulations.

Definition 3.5.15. The *expected value of perfect information (EVPI)* is defined as

$$\mathrm{EVPI} = z_{\min}^{\mathrm{EO}} - z_{\min}^{\mathrm{WS}}. \tag{3.14}$$

It measures the maximum amount a decision maker would be ready to pay in return for complete (and accurate) information about the future. The large EVPI says that the information about the future is valuable, a small value of the EVPI informs about little savings when we reach the perfect information.

The following relations between the defined values have been established by A. Madansky and J. Jensen.

Theorem 3.5.1. *The defined values satisfied the following relations.*

$$z_{\min}^{WS} \leq z_{\min}^{EO} \leq \mathrm{EEV},$$

and moreover for stochastic programs with fixed objective coefficients and any convex objective function $f(\boldsymbol{\xi})$ of $\boldsymbol{\xi}$:

$$z_{\min}^{EV} \leq z_{\min}^{WS}.$$

Proof. The proofs are obtainable from [5]. □

Because in the engineering problems we usually can not buy any additional information about the future, therefore the VSS becomes more practically relevant characteristic of using stochastic programming than the EVPI.

We can also use some other reformulations such as a *variance objective (VO)* if we want to avoid the large fluctuations of $f(\mathbf{x}, \boldsymbol{\xi})$. It can be also useful to find compromise

between two deterministic reformulations, e.g., between EO and VO. But this is not goal of this thesis, more detailed information can be find in [27].

In some models, constraints need not hold almost surely as we assumed to this point. They can instead hold with some probability. These *probabilistic*, or *chance* constraints take the form:

$$P(A^i(\omega)x \geq h^i(\omega)) \geq p^i,$$

where $0 < p^i < 1$ and $i = 1, \ldots, I$ is an index of the constraints that must hold jointly. We can, of course, model these constraints in a general expectational form. Or we can provide the deterministic equivalents of these constraints. More detailed information in [5].

Two-stage stochastic programming

In the previous part, we discussed stochastic programming problems in which the decision maker took only one decision. In this section we mention programs in which the decision maker will take two decisions in two different moments in time.

The first decision $\mathbf{x}$ is taken when there are no available information about the future realization $\boldsymbol{\xi}_s$ of random parameters $\boldsymbol{\xi}$. This decision is called a *first-stage decision* and the period when this decision is taken is called the *first stage (master program).*

The second decision $\mathbf{y}(\boldsymbol{\xi})$ is taken after particular realization of random parameters $\boldsymbol{\xi}_s$ becomes known. The decision is called a *second-stage decision* and the corresponding period is called the *second stage (subprogram).* Such a decision process can be described as follows:

$$\text{decision } \mathbf{x} \longrightarrow \text{observation } \boldsymbol{\xi}_s \longrightarrow \text{decision } \mathbf{y}(\boldsymbol{\xi}_s),$$

where $\mathbf{y}(\boldsymbol{\xi})$ means dependence of $\mathbf{y}$ on $\boldsymbol{\xi}$.

We may put together the first-stage and the second-stage program to have a complete mathematical description of the discussed decision situation. We have chosen the linear program for its simplicity.

Definition 3.5.16. *Two-stage stochastic linear program* is the problem of finding

$$\begin{aligned} \min_{\mathbf{x}} \quad & \left(\mathbf{c}^{\mathrm{T}}\mathbf{x} + \mathbb{E}_{\boldsymbol{\xi}}\left(Q(\mathbf{x}, \boldsymbol{\xi})\right)\right), \\ \text{s.t.} \quad & \mathbb{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where $Q(\mathbf{x}, \boldsymbol{\xi})$ is the optimal value of the second stage problem

$$\begin{aligned} \min_{\mathbf{y}} \quad & \mathbf{q}(\boldsymbol{\xi})^{\mathrm{T}}\mathbf{y}(\boldsymbol{\xi}), \\ \text{s.t.} \quad & \mathbb{T}(\boldsymbol{\xi})\mathbf{x} + \mathbb{W}(\boldsymbol{\xi})\mathbf{y}(\boldsymbol{\xi}) = \mathbf{h}(\boldsymbol{\xi}), \\ & \mathbf{y}(\boldsymbol{\xi}) \geq \mathbf{0}. \end{aligned}$$

The vector $\mathbf{x}$ represents the first-stage decision. The second-stage decision $\mathbf{y}(\boldsymbol{\xi})$ depends on the vector $\boldsymbol{\xi} = (\mathbf{q}, \mathbf{h}, \mathbb{T}, \mathbb{W})$ where some elements can be random. The matrix $\mathbb{T}$ is called technology matrix and $\mathbb{W}$ is recourse matrix. If the matrix $\mathbb{W}$ is fixed, the program is called *two-stage program with fixed recourse.* The two-stage programs are sometimes called *programs with recourse.*

For a given realization $\boldsymbol{\xi}_s$, the second-stage data $\boldsymbol{\xi}_s = (\mathbf{q}_s, \mathbf{h}_s, \mathbb{T}_s, \mathbb{W}_s)$ become known. The notation was simplified in the following manner $\mathbf{q}_s = \mathbf{q}(\boldsymbol{\xi}_s)$ and so on. The second-stage decision $\mathbf{y}_s$ or $\mathbf{y}_s(\mathbf{x})$ must be taken. The decisions $\mathbf{y}_s$ are typically not the same under different realizations of $\boldsymbol{\xi}_s$. But the decisions have to be chosen in order that the constraints hold almost surely, i.e., for all $\boldsymbol{\xi} \in \Xi$ with the potential exception of set with zero probability.

The objective function is composed of a deterministic term $\mathbf{c}^{\mathrm{T}}\mathbf{x}$ and the expectation of the second-stage objective $\mathbf{q}_s^{\mathrm{T}}\mathbf{y}_s$ taken over all realization of the random parameters $\boldsymbol{\xi}$. The second-stage term is more complicated because the value $\mathbf{y}_s$ is the solution of a linear program for each realization of uncertainty $\boldsymbol{\xi}_s$. To stress this fact we can use the notation of a deterministic equivalent program. For each realization of $\boldsymbol{\xi}$, let

$$\begin{aligned} Q(\mathbf{x}, \boldsymbol{\xi}) = \min_{\mathbf{y}} \ & \mathbf{q}(\boldsymbol{\xi})^{\mathrm{T}}\mathbf{y}(\boldsymbol{\xi}), \\ \text{s.t.} \ & \mathbb{W}(\boldsymbol{\xi})\mathbf{y}(\boldsymbol{\xi}) = \mathbf{h}(\boldsymbol{\xi}) - \mathbb{T}(\boldsymbol{\xi})\mathbf{x}, \\ & \mathbf{y}(\boldsymbol{\xi}) \geq \mathbf{0} \end{aligned}$$

be the second-stage value function. Then, define the expected second-stage value function

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

and the deterministic equivalent program is

$$\begin{aligned} \min_{\mathbf{x}} \ & \mathbf{c}^{\mathrm{T}}\mathbf{x} + \mathcal{Q}(\mathbf{x}), \\ \text{s.t.} \ & \mathbb{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

This representation of a two-stage stochastic program illustrates that the major difference from a deterministic formulation is in the second-stage value function. If that function is given, then a stochastic program is just an ordinary nonlinear program.

The generalization of the presented two-stage stochastic programs are the *multi-stage stochastic programs* that combine time and uncertainty in a more complex way. There are several stages (more than two) and the decisions are taken sequentially in different moments and also the realizations of random parameters sequentially become known. Some difficulties with possible dependencies of random parameters across the stages can occur. The two-stage program is the special case of multi-stage program. The multi-stage stochastic programs were not solved in this thesis, we mentioned them only with a reference to the further research. More detailed information about the stochastic programming can be found in [5], [19], [27], [30].

3.6 GAMS

All following models have been implemented in the optimization software GAMS. Therefore, we give here a basic information about General Algebraic Modeling System (GAMS), which was started as a research project at the World Bank in 1976. GAMS went commercial in 1987.

GAMS was developed to provide a high-level language for the compact representation of large and complex models, to allow changes to be made in model specifications simply

and safely, to allow unambiguous statements of algebraic relationships and to permit model descriptions that are independent of solution algorithms.

The GAMS model representation is in a form that can be easily read by people and by computers. This means that the GAMS program itself is the documentation of the model, and that the separate description required in the past is no longer needed.

The modelling language follows modelling steps discussed in the Section 3.1. A GAMS model is a collection of statements in the GAMS Language. The terminology adopted is as follows: indices are called `Sets`, a given data are called `Parameters`, decision variables are called `Variables` and constraints and the objective function are called `Equations`. Moreover we need the keyword `Model` followed by the name of the model followed by a list of equation names, which determines the collection of `Equations` included in a model. The statement `Solve` calls the solver. After that we type name of the model to be solved and the keyword `using` with some available solution procedure (e.g., `lp` for linear programming problems or `nlp` for nonlinear one). There are many others keywords, but we have mentioned the most important of them. The statements are consistent to mathematical programming problem parts. GAMS has a wide Model Library which is full of useful examples and models.

The solvers differ in the methods they use, in whether they find a globally optimal solution with proven optimality, in the size of models they can handle, and in the format of models they accept. CPLEX is a solver for linear programs and is based on the simplex method described for example in [1]. CONOPT is a solver especially for nonlinear programs and its algorithm is based on the generalized reduced gradient (GRG) method. Since we solved nonlinear programs and used the CONOPT solver, you can find some additional information about this solver in the Appendix B. BARON is a solver for nonlinear and mixed-integer nonlinear programs. It implements algorithm of the branch-and-bound.

We have encountered some difficulties during the GAMS implementation of our models. For example, the matrix operations such as multiplication or summing of two matrices are not implemented in this software and you have to create them yourself. The absolute value is a non-smooth function and may cause numerical problems, especially when the arguments of the function are variables. Therefore, we have utilized a known transformation for absolute values illustrated further. The term $|x| \leq 1$ can be replaced by the following two terms

$$\begin{aligned} x &\leq 1, \\ -x &\leq 1. \end{aligned}$$

We cannot plot our results in GAMS directly, so all graphical results have been obtained by MATLAB R2009b which was developed by The MathWorks, Inc., USA. Figures not directly related to outcomes from GAMS were made by mfpic, a package of macros for METAPOST - part of LaTeX, described in [21].

3.7 News vendor problem

This section presents a classical simple stochastic programming problem taken over with some additional comments from [5]. We extended the problem solution by some notes and mainly by implementation in GAMS. This problem is included to help in understanding of the previous theory.

Let us formulate our problem. A news vendor goes to the publisher every morning and buys x newspapers at a price of c per paper. This number is usually bounded above by some limit u, representing either the news vendor's purchase power or a limit set by the publisher to each vendor. The vendor then walks along the streets to sell as many newspapers as possible at the selling price q. Any unsold newspaper can be returned to the publisher at a return price r, with $r < c$.

We are asked to help the news vendor decide how many newspapers x to buy every morning. Demand for newspapers varies over days and is described by a random variable ξ.

It is assumed here that the news vendor cannot return to the publisher during the day to buy more newspapers. Other news vendors would have taken the remaining newspapers. Readers also only want the last edition.

To describe the news vendor's profit, we define y as the effective sales and w as the number of newspapers returned to the publisher at the end of the day. We may then formulate the problem as

$$\begin{aligned} \min \;& cx + \mathcal{Q}(x), \\ \text{s.t.} \;& 0 \leq x \leq u, \end{aligned}$$

where

$$\mathcal{Q}(x) = \mathbb{E}_\xi Q(x, \xi),$$

and

$$\begin{aligned} Q(x,\xi) = \min \;& \left(-qy(\xi) - rw(\xi)\right), \\ \text{s.t.} \;& y(\xi) \leq \xi, \\ & y(\xi) + w(\xi) \leq x, \\ & y(\xi), w(\xi) \geq 0, \end{aligned}$$

where $\mathbb{E}_\xi$ denotes the expectation with respect to ξ.

In this notation, $-\mathcal{Q}(x)$ is the expected profit on sales and returns, while $-Q(x,\xi)$ is the profit on sales and returns if the demand is at a level ξ. So we used the EO deterministic reformulation, where term cx can be given out of the expectation because it is not function of ξ. The model illustrates the two-stage aspect of the news vendor problem. The buying decision has to be taken before any information is given on the demand (the HN approach). When the demand is known in the so-called second stage, which represents the end of the sales period of a given edition, the profit can be computed (the WS approach). The model is a typical example on the two-stage stochastic problem with fixed recourse which was mentioned earlier.

The profit can be computed analytically taking a few steps and applying some simple rules. You can find the exact procedure how to compute the analytical solution in [5]. The optimal solution has the following form:

$$\begin{cases} x_{\min} = 0 & \text{if} \quad \frac{q-c}{q-r} < F(0), \\ x_{\min} = u & \text{if} \quad \frac{q-c}{q-r} > F(u), \\ x_{\min} = F^{-1}\left(\frac{q-c}{q-r}\right) & \text{otherwise}, \end{cases} \tag{3.15}$$

where $F(\xi)$ represents the distribution function of ξ (see (3.5.5) for a definition of the distribution function). So the vendor may still need to consult a statistician, who would

provide an accurate the distribution function $F(\xi)$. Only then a precise solution $x_{\min}$ will be available.

Moreover we implemented our model into GAMS. We do not further require integer values for variables as it is defined by selling pieces of newspaper because we emphasize explanatory role of our example.

We used specified values of parameters and also the specified distribution function. We supposed that $c = 0.5$ \$, $q = 1.5$ \$, $r = 0.2$ \$ and demand is uniform on the interval $[50, 150]$. The simple GAMS model that utilizes approximation of the expected value of the objective function by sampling from the uniform distribution to get the scenario-based two-stage program follows.

```
$title The news vendor problem

Set                     s       scenarios                   /1*1021/

Parameter               d(s)    newspaper demand;
                        d(s) = uniform(50,150);

Parameter               p(s)    scenario probability;
                        p(s) = 1/card(s);

Positive variables      x       bought quantity
                        y(s)    effective sales
                        w(s)    returned quantity;

Scalars                 c       price per paper             /0.5/
                        q       selling price per paper     /1.5/
                        r       return price                /0.2/;

Variable                obj     variable for objective function;

Equation                objective       objective function
                        demandbound(s)  sold papers bound by demand
                        inventory(s)    inventory bound;

objective.. obj =e= c*x + sum(s, p(s)*(- q*y(s) - r*w(s)));
demandbound(s).. y(s) =l= d(s);
inventory(s).. y(s) + w(s) =l= x;

model newsboy /all/;
solve newsboy using lp minimizing obj;
display x.l, obj.l;
```

We ran the model and got the optimal solution and the optimal value of the objective function in this solution. The news vendor should buy $x_{\min}^{\mathrm{EO}} = 125$ newspapers from the publisher and then he can expect gain around $-z_{\min}^{\mathrm{EO}} = 86$ \$ according to the obtained solution.

```
----     31 VARIABLE x.L                  =      125.582  bought quantity
            VARIABLE obj.L                =      -86.898  objective function
```

We can check our result with the analytical solution from (3.15). We assumed that the demand has the uniform distribution on $[50, 150]$ with the distribution function

$$F(\xi) = \begin{cases} 0 & \xi < 0, \\ \frac{\xi - 50}{150 - 50} & 50 \leq \xi < 150, \\ 1 & \xi \geq 150. \end{cases}$$

The inverse function (also called the *α-quantile of F*) inside of $[50, 150]$ has the form

$$F^{-1}(\alpha) = 50 + \alpha \cdot (150 - 50).$$

Now we have all necessary information and we can compute the exact optimal solution.

$$x_{\min} = F^{-1}\left(\frac{q-c}{q-r}\right) = 50 + \frac{q-c}{q-r} \cdot (150 - 50) = 50 + \frac{1.5 - 0.5}{1.5 - 0.2} \cdot 100 = 126.923 \text{ newspapers.}$$

We can see, that the solution from the GAMS implementation is little bit more pessimistic than the exact one, because it is computed with a not large enough number of realizations of the uncertainty (scenarios) obtained by sampling of the uniform dsitibution.

We also solved this model as the deterministic one by using the expected value of the demand instead of a random demand, i.e., the EV reformulation was employed. We obtained the optimal solution $x_{\min}^{\mathrm{EV}} = 100$ newspapers and the value of the objective function in this solution is $-z_{\min}^{\mathrm{EV}} = 100$ \$.

We can check if the relation mentioned in the Theorem 3.5.1 holds

$$\begin{aligned} z_{\min}^{\mathrm{EV}} &\leq z_{\min}^{\mathrm{EO}} \\ -100 &\leq -86.898. \end{aligned}$$

Some of the defined concepts and relations were implemented and demonstrated on this simple example. We hope that it could be profitable for understanding of the basic ideas of the stochastic programming.

Chapter 4

Finite element method

Mathematical modeling is a simplifying step. But models of physical systems are not necessarily simple to solve. They often involve ordinary or partial differential equations in space and time subject to boundary and/or initial conditions.

We can treat up derivatives by analytical or numerical solutions. Analytical solutions cannot be applied to a wide class of problems or the problems have to be restricted to regular geometries and simple boundary conditions. So often a numerical evaluation is useful. Here is where the finite element method and the digital computer enter the scene.

Unlike the traditional finite difference method, the finite element method is not obtained as an approximation of the differential equation directly. Instead integrated forms of the differential equations are used. In the following section inspired by [17] we shall look at a simple one-dimensional example and deduce the formulations that we need in order to formulate the finite element method discretization for our further mathematical programing model.

The theoretical side of the finite element method for finding the solutions of partial differential equations based on the functional analysis such as a convergence to an optimal solution was deeply studied at the Institute of Mathematics at the Faculty of Mechanical Engineering at Brno University of Technology, e.g. by M. Zlámal, A. Ženíšek, J. Franců or L. Čermák, their results can be studied in [15], [9] written in Czech or in [35], [13] written in English. We are focused on a practical application of this method not on a theoretical view. The eligibility of the solution can be checked from the physical meaning of the solved problem.

4.1 Virtual work

We consider the balance of virtual work for a simple problem, which covers many different physical problems: find v such that

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(cv - k\frac{\mathrm{d}v}{\mathrm{d}x}\right) = f,\ 0 < x < 1, \tag{4.1}$$

where c is a velocity parameter, assumed known. This equation describes a heat conduction in a fluid flow with a flow velocity c, with a temperature v still being a function

of a position x. Following boundary conditions must be supplied

$$v(0) = \frac{\mathrm{d}v}{\mathrm{d}x}(1) = 0,$$
$$k\frac{\mathrm{d}v}{\mathrm{d}x}(1) - cv(1) = 0.$$

We start by multiplying (4.1) by a function u, such that $u(0) = 0$, followed by integrating the differential equation over the domain

$$\int_0^1 \frac{\mathrm{d}}{\mathrm{d}x}\left(cv - k\frac{\mathrm{d}v}{\mathrm{d}x}\right) u\mathrm{d}x = \int_0^1 fu\mathrm{d}x.$$

We use the integration by parts to obtain

$$-\int_0^1 \left(cv - k\frac{\mathrm{d}v}{\mathrm{d}x}\right)\frac{\mathrm{d}u}{\mathrm{d}x}\mathrm{d}x + \left[cv - k\frac{\mathrm{d}v}{\mathrm{d}x}\right]_0^1 = \int_0^1 fu\mathrm{d}x,$$

and apply the boundary conditions to arrive at the following formulation of (4.1): find v such that

$$\int_0^1 \left(-cv + k\frac{\mathrm{d}v}{\mathrm{d}x}\right)\frac{\mathrm{d}u}{\mathrm{d}x}\mathrm{d}x = \int_0^1 fu\mathrm{d}x, \tag{4.2}$$

for all admissible u called as a *test function*, we say that the differential equation has been tested with u. Clearly some restrictions have to be put on our choice of u and on the solution v. At the very least, they have to be nice enough for the integrals in (4.2) to exist.

The basic idea is that if we just test with a sufficiently large number of test functions (indeed, infinitely many), it seems probable that the differential equation is forced to hold point-wise. By use of the partial integration, we can go back one step and write (4.2) as

$$\int_0^1 \left(f + \frac{\mathrm{d}}{\mathrm{d}x}\left(-cv + k\frac{\mathrm{d}v}{\mathrm{d}x}\right)\right) u\mathrm{d}x = 0,$$

so that if we are allowed to choose v and u from a sufficiently large class of functions, then we may say that the virtual work principle is equivalent to the differential equation. An important point is that the equations of virtual work allow us to work with functions that are less regular than required for solutions to the differential equation. Clearly, this is beneficial if we wish to use (4.2) to generate approximate solutions, which normally are less regular than the exact solution.

It is common in engineering to interpret (4.2) as a statement about the balance between the external and the internal *virtual work*, the test functions are then regarded as *virtual displacements*.

4.2 Galerkin's method

Galerkin's method is an approximation method which was based on the virtual work equation (4.2). If the approximate solution is written as

$$V(x) = \sum_{i=1}^{n} a_i N_i(x), \tag{4.3}$$

where $i \in \{1, 2, \ldots, n\}$, the a_i are unknown real numbers and the $N_i(x)$ are known, simple functions, such as polynomials or trigonometric functions, then one should make sure that (4.2) is satisfied for all admissible u of the same form as V, i.e., $u = \sum_{i=1}^{n} b_i N_i(x)$. The only way to ensure this is to enforce (4.2) for each N_i separately, since then the equation must also hold for an arbitrary combination of the N_i[1].

Thus, one should simply choose as test functions N_j, $j \in \{1, 2, \ldots, n\}$. With this choice we find that we have to solve the algebraic problem

$$\int_0^1 k \frac{\mathrm{d}N_j}{\mathrm{d}x} \left(\sum_{i=1}^{n} a_i \frac{\mathrm{d}N_i}{\mathrm{d}x} \right) \mathrm{d}x = \int_0^1 f N_j \mathrm{d}x \tag{4.4}$$

for $j = 1, 2, \ldots, n$. So, we obtained the Galerkin's procedure. It is always possible to take a general differential equation and multiply with a test function, integrate over the domain to obtain the Galerkin's method. The real strength of the Galerkin's method lies in its minimization properties.

It is usable to know the formula for the integration by parts for functions of several variables to obtain the virtual work equation for much more complicated problems. We can avoid some differentiating by using:

$$\int_\Omega \frac{\partial u}{\partial x_i} v \mathrm{d}\Omega = \int_{\partial\Omega} n_i u v ds - \int_\Omega u \frac{\partial v}{\partial x_i} \mathrm{d}\Omega, \tag{4.5}$$

where Ω is a domain in two or three dimensions, $\partial\Omega$ is the boundary of the domain, u and v are functions and n_i are components of the outward pointing normal to $\partial\Omega$.

The previous two sections were based on the literature referenced in [17], where some additional information about the FEM can be found.

4.3 Basic concepts of finite element method

The Galerkin's method has been discussed only for approximations that are defined and also continuous on the whole interval. However, there is nothing that requires this. The approximation can very well be only piecewise continuous and integrated piece-wisely. The Galerkin's method together with the use of piecewise polynomials is what constitutes the finite element method.

Let us summarize major steps of the finite element method and make some explanatory comments further.

1. Discretization of the domain into a finite number of subdomains (elements).
2. Selection of interpolation functions.
3. Development of the element matrix for the subdomain.
4. Assembly of the element matrices for each subdomain to obtain the global matrix for the entire domain.
5. Imposition of the boundary conditions.

[1] This claim is based on the knowledge of linear algebra, namely the theory of Hilbert space and its base.

6. Solution of equations.

7. Additional computations (if desired).

The described procedure will be applied for particular problem and explained in details.

The finite element method (FEM) is a technique for numerical solving partial or ordinary differential equations by discretising these equations in their space dimensions, same as the finite difference method (FDM). With the FDM, the differential equation is written for each node, and the derivatives are replaced by finite differences. This method is easy to understand and employable in simple problems, it becomes difficult to apply to problems with complex geometries or complex boundary conditions, or for nonisotropic material properties. In contrast, the FEM uses integral formulations (e.g., (4.2)) to create a system of algebraic equations (e.g., (4.4)). The solution is generated by connecting the individual solutions for each element, allowing for the continuity at the inter-elemental boundaries.

The FEM is a numerical procedure which can be used to obtain a solution to a large class of engineering problems involving stress analysis, heat transfer, electromagnetism, fluid flow, etc.. This method could be viewed as a procedure for obtaining numerical approximations to the solution of boundary value problems posed over a domain. This domain is replaced by the finite union of disjoint subdomains called finite elements. The replacement is done by discretization. So the FEM reduces the problem to that of a finite number of unknowns by dividing the domain into elements and by expressing the unknown field variable in terms of the assumed approximating functions within each element. These unknown functions are defined in terms of the values (function and its derivatives) of the field variables at specific points, reffered *nodes*. Nodes are usually located along the element boundaries, and they connect adjacent elements. Functions determined by unit node values are called *shape functions*. These functions are constructed in several ways, Lagrange polynomials are usually used, and they approximate an unknown function on each subdomain.

The Galerkin's method is used on each finite element to obtain the local element matrix. After that we can create the global matrix for whole problem composing the local matrices together by an assembly process. We also have to consider given boundary conditions.

It was stated that the most popular and also the most widely used discretization technique in structural mechanics is the finite element method. There are some other methods, for example the energy-based finite difference method or the finite volume method, these methods are particularly well entrenched in computational gas dynamics. More detailed information especially about the finite element method can be find in [14], [17], [24], [34] or further for a particular case.

Chapter 5

Decomposition

The size of a mathematical programming problem can be very large. One can encounter in practice problems with several hundred thousands of equations and/or unknowns. To solve these problems the use of some special techniques is either convenient or required. Alternatively, a distributed solution of large problems may be desirable for technical or practical reasons. Decomposition techniques allow certain type of problems to be solved in a decentralized or distributed fashion. Alternatively, they lead to a drastic simplification of the solution procedure of the problem under study.

Some decomposition algorithms for deterministic programming were developed, e.g., the Dantzig-Wolfe decomposition algorithm for linear programming problems with complicating constraints same as the Benders decomposition, see [3]. Nonlinear programming problems can be decomposed only if they have decomposable structure, there are three basic procedures: the Lagrangian relaxation, the augmented Lagrangian decomposition and the optimality condition decomposition [8]. The last procedure presents the most efficient computational behaviour in the most of cases. There are also some methods of decomposition for mixed-integer linear programming problems.

It is also very useful to do a sensitivity analysis of our decomposition method of the parameters settings influences. Different methods of decomposition with detailed algorithms were described by E. Castillo in [8].

Decomposition procedures are computational techniques that split the problem into at least two smaller and/or simpler subproblems. The price that has to be paid for such a simplification is repetition. That is, instead of solving the original compacted problem, at least two problems are solved iteratively, i.e., repetitively. Obviously we have to consider our profit from doing a decomposition and analyze the numerical behaviour of the used decomposition algorithm and show that the result obtained by the decomposition technique is identical to the solution of the original problem.

If we deal with an uncertainty, our problems are usually larger than in the deterministic case. So some sophisticated decomposition methods for stochastic programming problems were developed too.

5.1 Scenario decomposition methods

Consideration of uncertainties dramatically increases the size of a resulting mathematical program. When these problems are formulated appropriately the resulting decomposable block structure could be advantageously exploited for parallelization. The uncertainty

is incorporated into the problem by the use of scenarios. Each realization of random quantities is referred to a scenario.

Those decomposable optimization problems are ubiquitous in engineering and science applications. There are some decomposition techniques which can be used, for example the Dantzig-Wolfe decomposition approach, the L-shaped decomposition, the Benders decomposition and mainly the progressive hedging algorithm to that we will focus on because it is appropriate for our purposes and there is an experience with this algorithm at our university. Heuristic techniques can also be considered. The most of mentioned methods can be studied from [5].

The most of decompositions are based on the master programme and subprograms. A separate calculation of subprograms may be realized in parallel instead of the usual serial way. We will explain this concept deeper in the next section.

5.2 Progressive hedging algorithm

In this section, we will describe a parallel computational technique for solving scenario-based stochastic programming problems known as a *progressive hedging algorithm (PHA)* developed by R. T. Rockafellar and R. J.-B. Wets in 1991. More detailed information can be found, e.g., in their article [28].

PHA achieves a full separation of the scenario subproblems for each iteration to deal with the parallelization of solving those subproblems on a hardware with several processors simultaneously. Therefore, we have less work at each iteration but the number of iterations may be greater. The PHA solves a version of the scenario subproblem and progressively enforces the nonanticipativity constraints. The benefits of the parallel implementation are described in the Section 5.7.

J. M. Mulvey and H. Vladimirou implemented the progressive hedging algorithm on several shared memory machines and also on the network workstations for their stochastic network programs used in finance. The comparison showed an efficiency about 90% for the parallel implementation and worse results for the distributed implementation caused by slow communication in the computer network. You can read more in [25].

We will consider scenario-based models for the stochastic programming problem. Therefore, the uncertainty is realized by scenarios, the uncertain parameters can reach only specified values and each setting of uncertain parameters is modeled by one scenario. We denote all scenarios by set S,

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

where L is the number of all scenarios, assumed a small number. If the number of scenarios is large, we choose several scenarios, for instance, by an expert opinion about their importance, or by a representative discretization or by sampling. For each scenario $s \in S$, we solve a subproblem

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}, s), \\ &\text{s.t. } \mathbf{x} \in C_s, \end{aligned} \tag{5.1}$$

where $f(\mathbf{x}, s)$ is the objective function and $C_s \subset \mathbb{R}^n$ is the feasible set for the scenario s. We assume that each subproblem has the optimal solution $\mathbf{x}_s$ for all $s \in S$.

Further we continue with a scenario analysis. We analyze all scenario-solutions $\mathbf{x}_s$, discover trends or some clusters of solution. Then a weighted sum of scenario solutions $\mathbf{x}_s$ is computed and again analyzed by the scenario analysis, etc. Our goal is to find one universal solution that is "optimal" for an arbitrary scenario occurs.

Denote the weight corresponding to the scenario s and its solution $\mathbf{x}_s$ by p_s for all $s \in S$. The weights p_s fulfill conditions:

$$\begin{aligned} 0 \leq p_s \leq 1, \\ \sum_{s \in S} p_s = 1. \end{aligned}$$

In other words, p_s is the probability that a particular scenario s occurs. These weights may be obtained, e.g., from experts recommendations corresponding to the relative importance of each scenario. Further, define an average solution $\hat{\mathbf{x}}$ as

$$\hat{\mathbf{x}} = \sum_{s \in S} p_s \mathbf{x}^s. \tag{5.2}$$

The average solution can be considered as a defense against the uncertainty of the model in the PHA. In this algorithm, the average solution is used in the penalty terms for the scenario-related objective functions as it is further shown.

If we are looking for the solution that will be resistant and robust with respect to all possible scenarios, we will solve the following problem:

$$\begin{aligned} &\min_{\mathbf{x}} \sum_{s \in S} p_s f(\mathbf{x}, s), \\ &\text{s.t. } \mathbf{x} \in \bigcap_{s \in S} C_s. \end{aligned} \tag{5.3}$$

Its solution $\mathbf{x}_{\min}$ hedges all possible realizations of uncertain parameters that can occur.

But the scenario analysis is still reasonable in comparison to solve the problem (5.3) directly. One reason is that the stochastic programming problem is often very large, difficult to solve and exceeds computational capacity. If we use the scenario analysis and the weights are changed during the computing process, we can easily check how these changes change the solution. Other significant reason is that the parallel computing can speed up the calculations by working with several scenarios at the same time.

5.3 PHA for one-stage optimization problems

In this section, we will introduce the PHA for one-stage stochastic programs. So we go from scenario-solutions $\mathbf{x}_s$ of subproblems (5.1) to the solutions that converge to the solution $\mathbf{x}_{\min}$ of the problem (5.3).

We make an assumption that all scenario-solutions $\mathbf{x}_s$ of subproblems (5.1) for $s \in S$ are known. The average solution $\hat{\mathbf{x}}$ defined in (5.2) is called *implementable*, i.e., scenario-independent. We called a general solution $\mathbf{x}$ as *admissible* if it is feasible for all scenario subproblems, i.e., for each $s \in S$. Thus, admissibility is equivalent with a requirement

$$\mathbf{x} \in \bigcap_{s \in S} C_s,$$

where C_s is a feasible set for a given scenario s.

We are looking for the *feasible* solution $\mathbf{x}_{\min}$ to the problem (5.3) which means that the solution is implementable and also admissible. Admissibility has not to be unconditionally satisfied. The decision maker can accept a slightly inadmissible solution, for example, if the violation of the feasible set C_s was realized by a particular scenario s with a low probability. Therefore, we can accept a solution that is nearly admissible.

However, we will look for a feasible solution, thus implementable and admissible in the algorithm described below. The procedure will generate a sequence of solutions $\hat{\mathbf{x}}^j$, $j = 1, 2, \ldots$ from scenario-solutions $\mathbf{x}_s$ of subproblems (5.1). This sequence converges to the optimal solution $\mathbf{x}_{\min}$ of (5.3) for the convex case and its authors report algorithm convergence also for certain non convex cases. Its terms $\hat{\mathbf{x}}^j$ are obtained by increasing the requirement that the scenario-solutions $\mathbf{x}_s$ to the subproblems have to be implementable. The exact structure of the algorithm for one-stage models taken from [20] and [36] follows.

One-stage progressive hedging algorithm

0. Choose a penalty parameter $\varrho > 0$ and the termination parameter $\varepsilon > 0$. Set a vector $\mathbf{w}_s^0 = \mathbf{0}$ for each $s \in S$, set the initial estimate $\hat{\mathbf{x}}^0 = \mathbf{0}$ and $j = 1$.

1. For each $s \in S$ solve the approximation problem obtained by a modification of (5.3)

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}, s) + (\mathbf{w}_s^{j-1})^{\mathrm{T}}\mathbf{x} + \frac{\varrho}{2}\left\|\mathbf{x} - \hat{\mathbf{x}}^{j-1}\right\|^2, \\ &\text{s.t. } \mathbf{x} \in C_s \end{aligned} \tag{5.4}$$

   and denote its optimal solution as $\mathbf{x}_s^j$.

2. Calculate the average solution

$$\hat{\mathbf{x}}^j = \sum_{s \in S} p_s \mathbf{x}_s^j.$$

3. Evaluate the termination condition

$$\delta = \left(\left\|\hat{\mathbf{x}}^{j-1} - \hat{\mathbf{x}}^j\right\|^2 + \sum_{s \in S} p_s \left\|\mathbf{x}_s^j - \hat{\mathbf{x}}^j\right\|^2 \right)^{\frac{1}{2}} \leq \varepsilon. \tag{5.5}$$

   If the condition holds, stop the algorithm and $\hat{\mathbf{x}}^j$ is the solution to the problem (5.3) with a given tolerance ε. Otherwise, update the perturbation term

$$\mathbf{w}_s^j = \mathbf{w}_s^{j-1} + \varrho(\mathbf{x}_s^j - \hat{\mathbf{x}}^j)$$

   for each $s \in S$ and return to step 1 of the algorithm with $j = j + 1$.

Let us describe the algorithm deeper. The algorithm generates a sequence of solutions converging to the solution $\mathbf{x}_{\min}$ of the problem (5.3). For this purpose we solve only linear-quadratic perturbed versions of scenario-based subproblems (5.1).

The objective function (5.4) contains two *penalty* terms on the comparison with the objective function of subproblems (5.1). This arrangement is based on the *augmented Lagrangian function*. The penalty functions and the Lagrangian function concept are mentioned in the Appendix C.

We are looking for a solution that will stay optimal for an arbitrary scenario, i.e., we want to find $\hat{\mathbf{x}}^j$ close to $\mathbf{x}_s^j$ for all $s \in S$. The quadratic penalty term

$$\frac{\varrho}{2}\|\mathbf{x} - \hat{\mathbf{x}}^{j-1}\|^2$$

forces $\mathbf{x}_s^j$ to $\hat{\mathbf{x}}^{j-1}$. The linear penalty term $(\mathbf{w}_s^{j-1})^{\mathrm{T}}\mathbf{x}$ penalizes the difference between $\mathbf{x}_s^j$ and $\hat{\mathbf{x}}^j$ from the foregoing iteration of the algorithm.

Remark that norms used above are Euclidean norms on $\mathbb{R}^n$ that are defined in the Section 3.3. We can use some other norms than Euclidean, of course. But this is the most common.

Note that the termination condition (5.5) of the algorithm measures how close $\mathbf{x}_s^j$ is to $\hat{\mathbf{x}}^j$ for all $s \in S$ and how $\hat{\mathbf{x}}^j$ varies with j, therefore we call δ as the *distance parameter*. If the termination condition holds, we found the solution with a given tolerance ε, where $\varepsilon > 0$, and the loop of the algorithm is terminated. Otherwise the algorithm continues to the new iteration.

The behaviour of the progressive hedging algorithm is extremely sensitive to the choice of the penalty parameter ϱ. Most problems could be solved faster by a properly searched value of ϱ. But there is unfortunately no universal approach how to determine the value of ϱ to obtain a good behaviour of the algorithm. The penalty parameter has to be determined by experiments. This fact is the biggest weakness of the progressive hedging algorithm. Some numerical manipulations of the penalty parameters were done for example in the literature referenced as [18] with substantial savings.

Because the choice of the penalty parameter is so difficult, we can also change the penalty parameter with iterations and utilize heuristics. For instance, if the difference between two subsequent distance parameters δ is "large", we can enlarge the value of ϱ and conversely.

The main advantage of the PHA is that it uses locally convergent nonlinear programming algorithms having many available well-tested implementations. In addition, the solution averages guarantee a robustness of computational processes, but the convergence is usually slow.

The progressive hedging algorithm can be formulated for two-stage and also for multistage programming problems but it is not goal of this thesis. The exact algorithms with a complementary terminology can be found in [20]. Even though, one concept from the multi-stage PHA is important for further discussions, so let us tell a few words about *nonanticipativity constraints*.

Nonanticipativity

We require satisfying a *nonanticipativity* of the first-stage decision in the two-stage or multistage stochastic programming. The first-stage decision has to be taken before any observation of random parameters $\boldsymbol{\xi}$ is known. The principle of nonanticipativity consists in independence of the first-stage decision on the future realization of $\boldsymbol{\xi}$, so the first-stage decision is constant for whatever happens in the future.

This requirement can be ensured directly by adding the *nonanticipativity constraints* explicitly to the formulation of our problem, i.e., we require a constant first-stage decision for all scenarios. For instance, we require fulfillment of the following constraint for two scenarios

$$\forall s^1, s^2 \in S : \mathbf{x}_{s^1} = \mathbf{x}_{s^2}.$$

The nonanticipativity requirement can be alternatively ensured by adding the penalty term to the objective function, see the Appendix C for more information about penalty functions. In the one-stage PHA, the nonanticipativity is ensured by the quadratic penalty term in the program (5.4).

5.4 One-stage PHA example

Let us present the one-stage progressive hedging algorithm on a simple model. This example with a solution and a figure was taken from [20]. The example illustrates the steps of the PHA very well and can help to understand the point of the algorithm.

The model with two scenarios and two variables has the form:

$$\begin{aligned} \min\ & (x_1 - \xi_1^s)^2 + (x_2 - \xi_2^s)^2, \\ \text{s.t.}\ & \xi_3^s \leq x_1 \leq \xi_4^s, \\ & \xi_5^s \leq x_2 \leq \xi_6^s, \end{aligned}$$

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

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

and for the scenario s^2:

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

The objective functions are paraboloids with vertices in points $(3, 4)$ and $(4, 3)$. The feasible sets are two shifted squares. The scenario s^1 corresponding to $\boldsymbol{\xi}^1$ is represented by blue color and the scenario s^2 corresponding to $\boldsymbol{\xi}^2$ is represented by red color. The dashed circles represent cuts of paraboloids by planes parallel to the x_1x_2 plane, so-called *contours*.

The optimal solution for the scenario s^1 is obviously the right upper vertex of the square feasible set $\mathbf{x}^1_{\min} = (3, 4)^{\mathrm{T}}$ and the optimal solution for the scenario s^2 is the point $\mathbf{x}^2_{\min} = (4, 3)^{\mathrm{T}}$. These points could be viewed in the Figure 5.1 as small colored squares.

The results produced by the progressive hedging algorithm for individual scenarios $\mathbf{x}_s^j$ in individual iterations are pictured by small red and blue circles and average solutions $\hat{\mathbf{x}}^j$ are pictured by black circles. The probability of both scenarios equals to $\frac{1}{2}$, the penalty parameter was set as $\varrho = 3$ and the termination tolerance as $\varepsilon = 10^{-9}$.

The algorithm produces the sequence of points $\hat{\mathbf{x}}_j$ that converges to the optimal solution $\mathbf{x}_{\min} = (3, 3)^{\mathrm{T}}$ plotted by green circle. This point is implementable and admissible, hence feasible. The optimal solution with given tolerance was reached in 17 iterations of the PHA.

Let us note here again that the choice of penalty parameter is crucial. A value of ϱ has to be determine by experimentations. There is the significant relationship between the penalty parameter and the number of iterations needed to find the solution with a given tolerance even in this simple case. Detailed information is stated in [20].

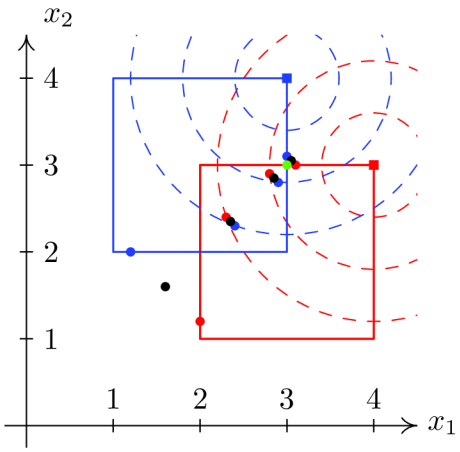

Figure 5.1: Scenario solutions and generated average solutions

5.5 Idea of spatial decomposition

In this section our original approach of the spatial decomposition method for large-scale partial or ordinary differential equation (PDE/ODE) constrained programs (some applications and algorithms for these programs can be found in [6]) is described. This method can be used for both deterministic and stochastic programming problems. But the main ideas will be discussed on deterministic programs.

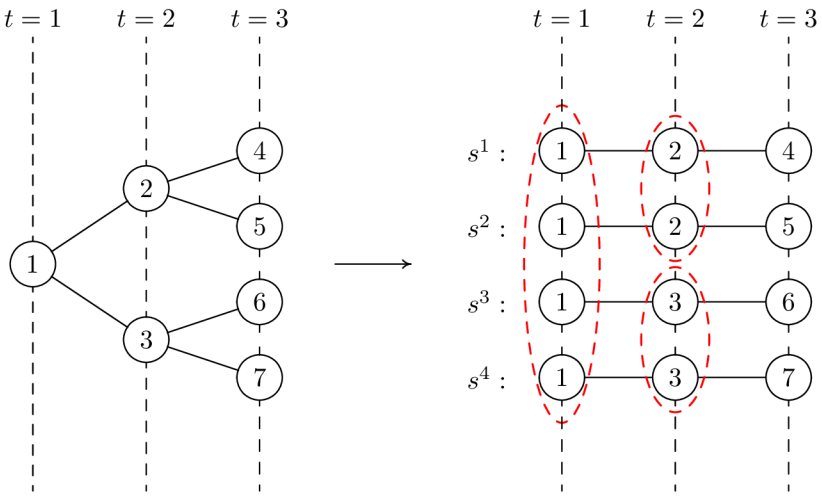

Figure 5.2: PHA - nonanticipativity

The idea of the spatial decomposition is based on the scenario decomposition method - the PHA. It is algorithm designed for the decomposition into individual scenarios with involvement the nonanticipativity requirement into the objective function by the penalty terms. It allows a parallel implementation. The decomposition can be viewed schematically in the Figure 5.2, where the nonanticipativity requirements are depicted by red dashed ellipses.

This thought was extended by M. Steinbach in his presentation at the Stochastic Programming Conference in Berlin 2001, see [32]. He applied ideas from the dynamic

programming for his multi-stage stochastic problem. He identified state variables linking subsequent stages, relaxed them and included them in the form of penalty terms in the extended objective functions related to both scenarios and stages. Schematically, a state of system in one stage is $\mathbf{y}_1$ and the state in another stage is $\mathbf{y}_2(\mathbf{y}_1^*)$. The asterisk represents the relaxation of the same state variable $\mathbf{y}_1$ for subsequent stages. Therefore, we replace one state variable for two stages by two different variables, i.e., one for each stage. However, we build the distance term

$$\|\mathbf{y}_1 - \mathbf{y}_1^*\|,$$

that leads to addition a penalty term into an objective function similar to the PHA. This approach can be called as a decomposition in time and it also allows the parallel implementation.

Steinbach's approach had inspired P. Popela towards the idea of a spatial decomposition for more complex optimized design structures that was born several years later. He also discussed this idea with specialists in civil and mechanical engineering and they have shown their interests to this approach. Therefore, after the recent discussion among D. Morton, P. Popela and M. Steinbach at the International Conference Prague Stochastics, 2010, the goal for this thesis to test the idea seriously has been defined. The stochastic program involving differential equation-based constraints for the prototype application has been chosen.

5.6 Basic steps of spatial decomposition

The spatial decomposition can be employed for solving PDE/ODE constrained programs which are very common in engineering applications. The algorithm uses a mesh which was created for the approximative description of PDE/ODE constraints and it could be used for both stochastic and also deterministic programs. We present the steps of algorithm for deterministic programs because of its simplicity. Stochastic programs can be reformulated easily as the deterministic programs. An uncertainty will be added to the algorithm in particular example presented further.

Let us describe our original approach step by step for a deterministic model with PDE or ODE constraints. The algorithm is followed by several remarks.

1. Choose a penalty parameter and a tolerance, set all necessary initial values.

2. Use an approximation scheme based on the discretization for PDE/ODE constraints to reformulate derivatives as a system of linear equations.

3. Solve the optimization model with a raw discretization to obtain boundary conditions for subproblems.

4. Introduce a decomposition of the problem's domain into parts with an overlap.

5. Add the values of approximative solution at the end points of subdomains computed in step 2 as boundary conditions to models of subproblems. Solve subproblems with a finer discretization on individual subdomains.

6. Compute the average solution, i.e., average two or more values on overlaps and take particular values of solution on parts of domain, where no overlap is available.

7. Evaluate the termination condition. If the condition holds, you have the solution with a given tolerance and you can stop the algorithm. Otherwise, increase the counter of iterations by one and solve modified subproblems again. Modification lies in adding the penalty term to the objective function same as in the PHA and in not considering boundary conditions from step 2 longer. Recompute the perturbation term same as in the PHA and go back to step 6.

We have to use a numerical method to obtain a non-differential numerical description of the constraints. We chose the finite element method described in the Chapter 4. This method is based on a discretization of the domain into a finite number of subdomains. The created discretization is moreover used for the spatial decomposition.

The raw discretization determines possible spatial decompositions. We can decompose the original domain only in end points of finite elements (nodes). The finer discretization has to be selected in such a way that we received the values of a solution in the matching spatial points. Therefore, we must carefully consider how many elements should be used in the raw and the finer mesh respectively.

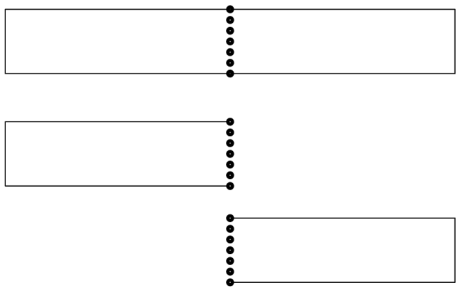

Figure 5.3: Line overlap

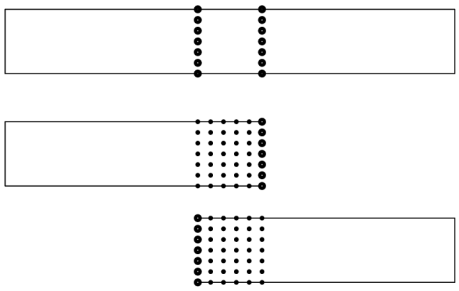

Figure 5.4: Spatial overlap

The length of overlap and the setting of penalty parameters are very important for the behaviour of the algorithm. But we do not have any universal approach how to determine them. The setting of these parameters for a test problem will be discussed later. We tried to implement the degenerate line overlap (Figure 5.3) with unsatisfactory results. Therefore, we recommend to use a spatial overlap with a greater length, schematically shown in the Figure 5.4.

The boundary conditions gained from the solution on the raw discretized domain can be used only in the first iteration of the algorithm. Otherwise, we will get the inaccurate solution with jumps.

By splitting the problem we obtained smaller subproblems to solve but we have to repeat our computations iteratively. Hence, we have to consider responsibly if this decomposition technique is suitable for our problem or not.

We still have to remember that the progressive hedging algorithm converges to the optimal solution only for convex feasible sets and objective functions. We will meet very often real problems for which the convergence of the PHA is not guaranteed. On the other hand, we can find a pretty good starting point to provide the convergence of the algorithm. The convergence theorem of the PHA with its proof can be find in [5] or [28].

5.7 Parallel implementation of PHA

Real optimization problems are modeled using large-scale programs. Usually only one processor is used to perform all tasks required by the algorithm. But the implemented algorithm may be unsuccessful in searching the optimal solution because of existing computer speed limits. Some algorithms mentioned earlier decompose the problem into several steps called tasks, which are related in some sense but can be proceeded independently. Therefore, a multiprocessor parallel technique can be used. The description of the parallel implementation can be found in [20].

The parallelism is provided by the property of the progressive hedging algorithm that decomposes an original problem into independent subproblems for each particular scenario in the scenario decomposition or for each particular part in the spatial decomposition. The subproblems can be solved separately and in parallel on hardware with several processors instead of the classical serial technique. This fact saves the computing time since n subproblems can be solved simultaneously on n parallel processors in the same time as one subproblem on one-processor machine.

The classical approach is the serial implementation. The subproblems, the total number of subproblems is L, are solved one by one as is schematically depicted in the Figure 5.5. Denote the computing time for solving one subproblem for a particular scenario or a part by τ and the total number of iterations of the progressive hedging algorithm by N. The total computing time consumed by solving all subproblems serially is $T^s_{total} = \tau NL$, the superscript s indicates the serial approach.

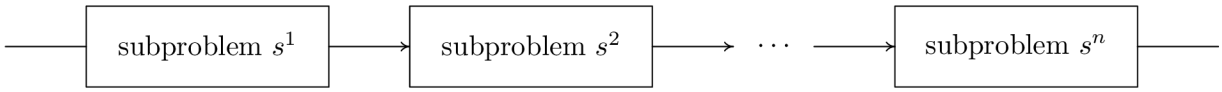

Figure 5.5: Serial implementation scheme

The alternative approach is based on availability of n processors, n is usually less than the number of subproblems L. These processors can solve n subproblems simultaneously, i.e., in parallel. The parallel part of computations, its scheme is in the Figure 5.6, is repeated in a loop until all subproblems are solved, i.e., $\lceil \frac{L}{n} \rceil$-times[1]. Therefore, this approach is in fact a combination of parallel computing of n subproblems and a serial loop repeated $\lceil \frac{L}{n} \rceil$-times. Denote again the computing time for solving n subproblems in one loop as τ. The total computing time consumed by solving all subproblems in parallel is then $T^p_{total} = \tau N \lceil \frac{L}{n} \rceil$, the superscript p indicates the parallel approach.

We can compare stated theoretical total computing times for the serial and the parallel implementation of the progressive hedging algorithm. Make an assumption that the optimal solution is reached in N iterations. Then, we obtain the following inequality.

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

So the total computing time consumed by parallel implementation is less or equal than the time consumed by serial implementation. The equality holds for $n = 1$ of parallel

[1] $\lceil x \rceil$ is the smallest integer not less than x, this map is called ceiling function.

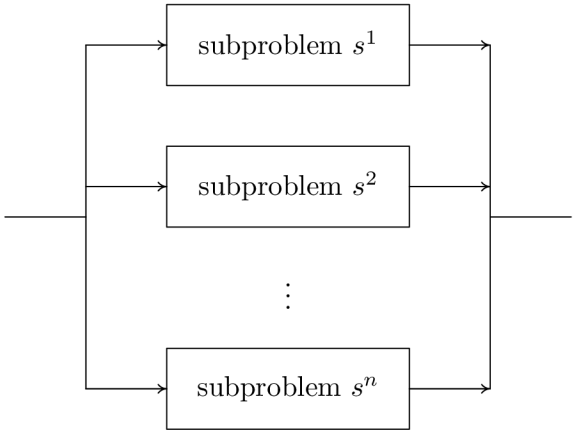

Figure 5.6: Parallel implementation scheme

processors, that is the serial case. If we increase the number of processors n, the difference $T^s_{total} - T^p_{total}$ increases. For $n \geq L$ we get the least total computing time consumed by a parallel implementation $T^p_{total} = \tau N$.

Chapter 6

Design of beam cross section dimensions

6.1 Problem formulation

Consider an ordinary differential equation constrained two-stage stochastic nonlinear program modeling an optimization problem from the area of civil engineering describing a deflection of a beam[1].

This problem was taken from [36], we did only some supplementary changes in a load of the beam. The author solved it by using the finite difference method and with focus on scenario-based models. Our goal is the solution with the finite element method and especially the implementation of the concept of the spatial decomposition. The main advantage of using the already solved example is the possibility to compare some early results.

The objective of the optimization is to find an optimal design of beam cross section dimensions while its weight is minimized (6.1) and rigidity is maximized (6.2), see the model and the Figure (6.1) further.

$$\min \ \rho abl, \tag{6.1}$$

$$\max \ \frac{E(\xi)ab^3}{12}, \tag{6.2}$$

$$\text{s.t. } E(\xi)\frac{ab^3}{12}\frac{\mathrm{d}^4 v}{\mathrm{d}x^4}(\xi,x) = h(x),\ x \in \langle 0,l\rangle, \xi \in \Xi, \tag{6.3}$$

$$v(\xi,0) = 0, \frac{\mathrm{d}v}{\mathrm{d}x}(\xi,0) = 0, \xi \in \Xi, \tag{6.4}$$

$$v(\xi,l) = 0, \frac{\mathrm{d}v}{\mathrm{d}x}(\xi,l) = 0, \xi \in \Xi, \tag{6.5}$$

$$\left| E(\xi)\frac{\mathrm{d}^2 v}{\mathrm{d}x^2}(\xi,x)\frac{b}{2} \right| \leq \sigma_{limit},\ x \in \langle 0,l\rangle, \xi \in \Xi, \tag{6.6}$$

$$a_{min} \leq a \leq a_{max}, \tag{6.7}$$

$$b_{min} \leq b \leq b_{max}, \tag{6.8}$$

[1] Assume that cross-section dimensions remain constant throughout its length and are substantially smaller than the length of the beam. Therefore, the beam can be modeled by the prismatic bar and the ordinary differential equation describes the deflection of the centerline.

where ρ is the beam density, a and b are decision variables (dimensions of the beam cross section), l is the beam length, $\xi : \Xi \to \mathbb{R}$ is a random variable, E is random Young's modulus[2] (because of varying uncertain material characteristics), x is the space coordinate, $v(\xi, x)$ is a deflection with the opposite direction than the axis y and $h(x)$ is a deterministic static load.

The ODE (6.3) describes transverse deflection of the beam, boundary conditions for clamped end points are given by (6.4) and (6.5), i.e., there are zero transverse deflections and their slopes. Furthermore, the maximum stress

$$\sigma_{max}(x) = \frac{M(x)}{J} y_{max} = \pm E \frac{\mathrm{d}^2 v}{\mathrm{d}x^2}(x) \frac{b}{2},$$

where

$$M(x) = -EJ \frac{\mathrm{d}^2 v}{\mathrm{d}x^2}(x)$$

is the bending moment, $J = \frac{ab^3}{12}$ is the second moment of the cross section with respect to the axis z and $y_{max} = \pm\frac{b}{2}$, must be bounded because of safety reasons, see the constraint (6.6). Limiting value σ_{limit} is defined as stress at which a material begins deform plastically. It is the end of the area of elastic behaviour described by Hooke's law where the stress is proportional to the relative deformation. Finally, the dimensions of the beam cross section must be bounded, see (6.7) and (6.8).

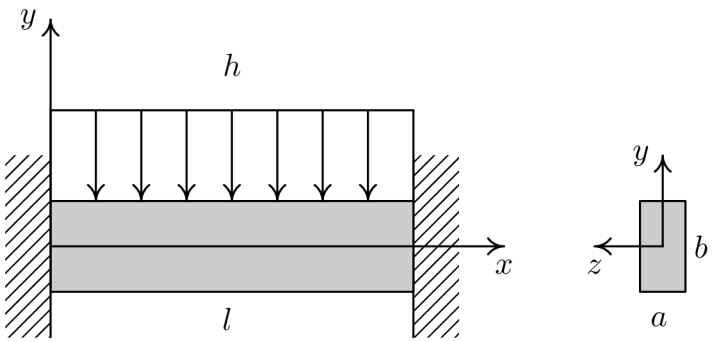

Figure 6.1: Scheme of loaded beam and its cross section

Hence, we obtain a continuous two-stage stochastic nonlinear program. The first stage here-and-now decision is realized by variables a and b. The second stage wait-and-see decision $v(\xi_s, x)$ is taken after an observation of random parameter ξ.

6.2 IS deterministic reformulation

As we mentioned in the Section 3.5, a deterministic reformulation of the underlying program (UP) (6.1)-(6.8) has to be made. We are not able to solve the UP directly. For instance, we do not know how to minimize the objective function in variable x if this function also contains some uncertain parameters.

We will consider the IS reformulation, it means that the random parameter in the program is replaced by a typical realization - individual scenario value, it is one specified

[2]The Young's modulus is the constant describing the elastic properties of a material.

value of Young's modulus. This reformulation can be understood as the EV reformulation too, because the expected value of random parameter can be used as the individual scenario. We took the value of Young's modulus from material tables for a specific material. Hence, we obtained a deterministic nonlinear program, where we denoted Young's modulus in the individual scenario ξ_s as E_s and similarly the deflection in the individual scenario as $v_s(x)$.

$$\min \ \rho abl, \tag{6.9}$$

$$\max \ \frac{E_s a b^3}{12}, \tag{6.10}$$

$$\text{s.t. } E_s \frac{ab^3}{12} \frac{\mathrm{d}^4 v_s}{\mathrm{d}x^4}(x) = h(x), \ x \in \langle 0, l \rangle, \tag{6.11}$$

$$v_s(0) = 0, \frac{\mathrm{d}v_s}{\mathrm{d}x}(0) = 0, \tag{6.12}$$

$$v_s(l) = 0, \frac{\mathrm{d}v_s}{\mathrm{d}x}(l) = 0, \tag{6.13}$$

$$\left| E_s \frac{\mathrm{d}^2 v_s}{\mathrm{d}x^2}(x) \frac{b}{2} \right| \leq \sigma_{limit}, \ x \in \langle 0, l \rangle, \tag{6.14}$$

$$a_{min} \leq a \leq a_{max}, \tag{6.15}$$

$$b_{min} \leq b \leq b_{max}. \tag{6.16}$$

The reformulated program (6.9)-(6.16) does not contain any uncertainties, the random parameter was removed. But we still need to do a few steps to receive a solvable and implementable model. We have stated above that the approximations of derivatives must be made, i.e., the finite element method with uniform grid for discretization in the space coordinate x is used to get rid of the derivatives in ODE constraints.

6.3 FEM for beam element

The described problem was solved using the finite difference method in [36]. We will use the finite element method to obtain a numerical and non-differential description of differential constraints. This method can be advantageously used further. The accuracy of both methods is same.

We need an approximation of the fourth derivative (the highest derivative in the model) of the unknown function v_s included in the constraint (6.11), the second derivative of v_s contained in (6.14) can be easily obtained subsequently.

The one-dimensional slender beam with the space dimension x is subdivided into N finite elements according to the Figure 6.2. Each element is bounded by two *nodes*.
In the following text we will denote the approximation of function $v_s(x)$ in the node x_e as

$$V_{s,e} \approx v_s(x_e)$$

and its derivative in the same node as

$$\theta_{s,e} \approx \frac{\partial v_s}{\partial x}(x_e).$$

Now consider the e-th element loaded by uniformly distributed transverse load $h(x)$ per unit length, schematically shown in the Figure 6.3. There are two degrees of freedom

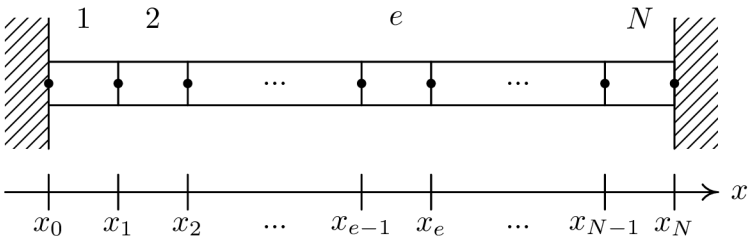

Figure 6.2: Meshed beam

in each node. The end nodes x_{e-1} and x_e are loaded by forces F_{e-1}, F_e and moments M_{e-1}, M_e, gained by the discretization of load $h(x)$, that result in translations $V_{s,e-1}$, $V_{s,e}$ and rotations $\theta_{s,e-1}$, $\theta_{s,e}$. The length of element is d, where $d = \frac{l}{N}$.

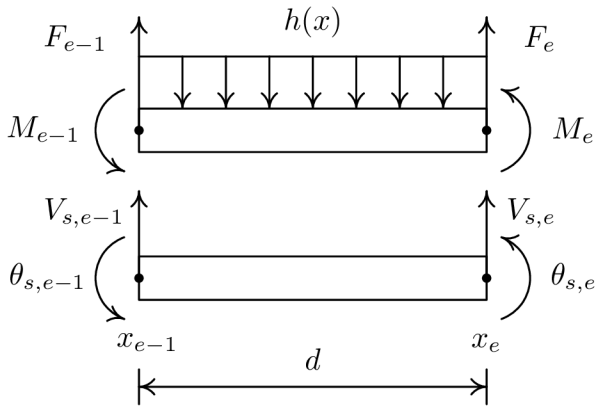

Figure 6.3: Slender beam element

The function v_s on the e-th element - $v_{s,e}$ is approximated by well chosen *shape functions* and discrete nodal values $V_{s,e}$ and $V_{s,e-1}$, but we do not use only nodal values but also the nodal values of derivatives $\theta_{s,e-1}$, $\theta_{s,e}$. We write

$$v_{s,e} \approx \begin{pmatrix} N_1, & N_2, & N_3, & N_4 \end{pmatrix} \begin{pmatrix} V_{s,e-1} \\ \theta_{s,e-1} \\ V_e \\ \theta_{s,e} \end{pmatrix}. \tag{6.17}$$

Shape functions N_i, $i \in \{1, 2, 3, 4\}$ are exactly chosen cubic polynomials

$$\begin{aligned} N_1 &= \frac{1}{d^3}(d^3 - 3dx^2 + 2x^3), \\ N_2 &= \frac{1}{d^2}(d^2x - 2dx^2 + x^3), \\ N_3 &= \frac{1}{d^3}(3dx^2 - 2x^3), \\ N_4 &= \frac{1}{d^2}(x^3 - dx^2). \end{aligned}$$

These shape functions have the property that they or their derivatives equal one at a specific node and zero at all others, the properties are illustrated in the Figure 6.4. Note that

the scale of the vertical axis for N_1 and N_3 is different than the scale for N_2 and N_4 for a better clarity.

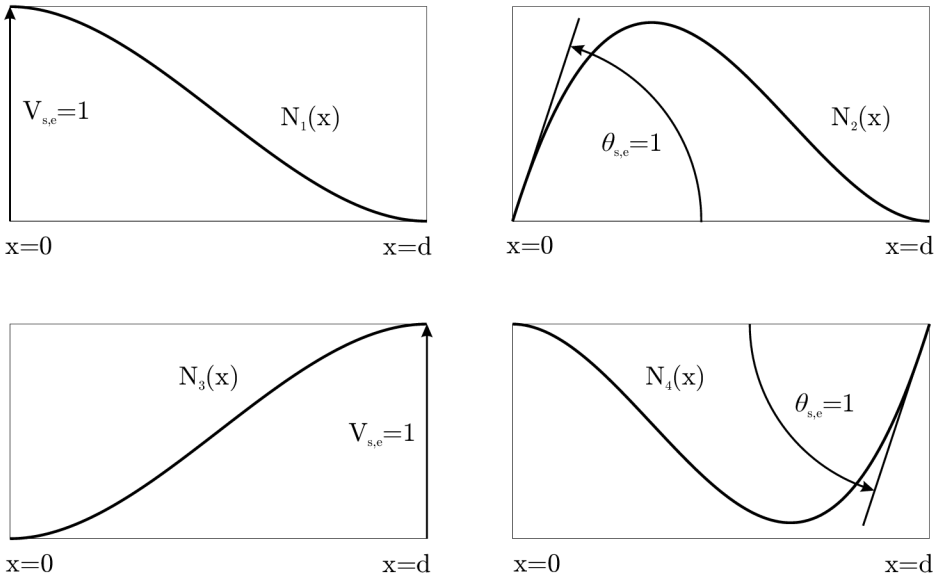

Figure 6.4: Shape functions

To develop the element matrix for the subdomain we need to derive the integral formulation of (6.11) as was mentioned in the Chapter 4. Therefore, we multiply (6.11) by a test function u followed by integrating over the domain to obtain

$$\int_0^d uE_s\frac{ab^3}{12}\frac{\mathrm{d}^4v_s}{\mathrm{d}x^4}(x)\mathrm{d}x = \int_0^d h(x)u\mathrm{d}x. \tag{6.18}$$

Because this integral identity has to be satisfied for all admissible u, we can simply choose the shape function itself as the test function. Substitution (6.17) into (6.18) and using the mentioned test function u on the e-th element lead to the four element equations:

$$\int_0^d \begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix} E_s\frac{ab^3}{12}\frac{\partial^4}{\partial x^4}\begin{pmatrix} N_1, & N_2, & N_3, & N_4 \end{pmatrix}\begin{pmatrix} V_{s,e-1} \\ \theta_{s,e-1} \\ V_{s,e} \\ \theta_{s,e} \end{pmatrix}\mathrm{d}x = \int_0^d h\begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix}\mathrm{d}x. \tag{6.19}$$

Integration by parts stated in (4.5) is used to avoid differentiating four times, the boundary terms were neglected because they are not significant in the global approximation matrix structure

$$\int N_i\frac{\partial^4 N_j}{\partial x^4}\mathrm{d}x \approx -\int \frac{\partial N_i}{\partial x}\frac{\partial^3 N_j}{\partial x^3}\mathrm{d}x \approx \int \frac{\partial^2 N_i}{\partial x^2}\frac{\partial^2 N_j}{\partial x^2}\mathrm{d}x,$$

where $i, j \in \{1, 2, 3, 4\}$. Hence, assuming that E_s, a, b and h are not functions of x and also $V_{s,e-1}$, $\theta_{s,e-1}$, $V_{s,e}$, $\theta_{s,e}$ are specific values of constants, the equations (6.19) become

$$E_s\frac{ab^3}{12}\int_0^d \frac{\partial^2 N_i}{\partial x^2}\frac{\partial^2 N_j}{\partial x^2}\mathrm{d}x\begin{pmatrix} V_{s,e-1} \\ \theta_{s,e-1} \\ V_{s,e} \\ \theta_{s,e} \end{pmatrix} = h\int_0^d \begin{pmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{pmatrix}\mathrm{d}x,$$

where $i, j \in \{1, 2, 3, 4\}$. Evaluation of the integrals gives the symmetric element matrix:

$$E_s \frac{ab^3}{12} \begin{pmatrix} \frac{12}{d^3} & \frac{6}{d^2} & -\frac{12}{d^3} & \frac{6}{d^2} \\ & \frac{4}{d} & -\frac{6}{d^2} & \frac{2}{d} \\ \text{sym.} & & \frac{12}{d^3} & -\frac{6}{d^2} \\ & & & \frac{4}{d} \end{pmatrix} \begin{pmatrix} V_{s,e-1} \\ \theta_{s,e-1} \\ V_{s,e} \\ \theta_{s,e} \end{pmatrix} = h \begin{pmatrix} \frac{d}{2} \\ \frac{d^2}{12} \\ \frac{d}{2} \\ -\frac{d^2}{12} \end{pmatrix}. \tag{6.20}$$

The equation (6.20) recovers the standard slope-deflection equation for beam elements. In more compact matrix notation (we have to multiply each element of matrix by $\frac{1}{12}$ first)

$$E_s ab^3 \mathbb{K}_e \begin{pmatrix} V_{s,e-1} \\ \theta_{s,e-1} \\ V_{s,e} \\ \theta_{s,e} \end{pmatrix} = \mathbf{h}_e.$$

So we developed the element matrix for the subdomain, this matrix is the same for all elements e, where $e \in \{2, \ldots, N-1\}$. The first and the last element matrices are affected by zero boundary conditions (6.12) and (6.13). The conditions determine the zero values of $V_{s,0}$, $\theta_{s,0}$, $V_{s,N}$ and $\theta_{s,N}$. Therefore, we have to delete the corresponding rows and columns from the element matrices to avoid the singularity in the global matrix.

Now we know how all the element matrices look like and we can easily put together the global matrix for the entire domain. The global matrix can be obtained by an assembly operation realized by the localization operator, it is a $4 \times 2(N+1)$ matrix of the form

$$\mathbb{L}_e = \begin{pmatrix} 0 & \ldots & 0 & 1 & 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 0 & 0 & 1 & 0 & \ldots & 0 \end{pmatrix},$$

where 1 is placed in the $(2e-1)$-th, $(2e)$-th, $(2e+1)$-th, $(2e+2)$-th columns. Then we received the approximation for the entire domain

$$E_s ab^3 \mathbb{K} \begin{pmatrix} V_{s,0} \\ \theta_{s,0} \\ \vdots \\ V_{s,N} \\ \theta_{s,N} \end{pmatrix} = E_s ab^3 \mathbb{K} \mathbf{V}_s = \mathbf{h}, \text{ where } \mathbb{K} = \sum_{e=1}^{N} \mathbb{L}_e^{\mathrm{T}} \mathbb{K}_e \mathbb{L}_e \text{ and } \mathbf{h} = \sum_{e=1}^{N} \mathbb{L}_e^{\mathrm{T}} \mathbf{h}_e, \tag{6.21}$$

where the first two and the last two equations are redundant because they contain the known values of deformations and their known slopes, so we have to delete them. This approach can be used only for zero boundary conditions. If there are some nonzero boundary conditions, we have to treat them up by a different approach described later. Now we have the approximation of the fourth derivative of unknown function $v_s(x)$ for whole beam with impositioned boundary conditions.

Further, we have to deal with the second derivative of the unknown function $v_s(x)$ contained in (6.14). Now we assume that we already know the vector discretely approxi-

mating the function $v_s(x)$:

$$\mathbf{V}_s = \begin{pmatrix} V_{s,0} \\ \theta_{s,0} \\ \vdots \\ V_{s,N} \\ \theta_{s,N} \end{pmatrix}$$

from the previously described system of linear equations (6.21). We have to limit the stress on each element, so we substitute (6.17) into (6.14) on e-th element to obtain

$$\left| E_s \frac{b}{2} \begin{pmatrix} N_1'', & N_2'', & N_3'', & N_4'' \end{pmatrix} \begin{pmatrix} V_{s,e-1} \\ \theta_{s,e-1} \\ V_{s,e} \\ \theta_{s,e} \end{pmatrix} \right| \leq \sigma_{limit}.$$

This equation describing the stress in one specific node holds only for the end nodes belonging precisely to one element. The constraint for the first node with the spatial coordinate x_0

$$\left| E_s \frac{b}{2} \begin{pmatrix} N_1''(0), & N_2''(0), & N_3''(0), & N_4''(0) \end{pmatrix} \begin{pmatrix} V_{s,0} \\ \theta_{s,0} \\ V_{s,1} \\ \theta_{s,1} \end{pmatrix} \right| \leq \sigma_{limit}$$

must be satisfied and also for the the last node with the spatial coordinate x_N

$$\left| E_s \frac{b}{2} \begin{pmatrix} N_1''(d), & N_2''(d), & N_3''(d), & N_4''(d) \end{pmatrix} \begin{pmatrix} V_{s,N-1} \\ \theta_{s,N-1} \\ V_{s,N} \\ \theta_{s,N} \end{pmatrix} \right| \leq \sigma_{limit}$$

must hold. The rest of nodes belongs to two adjacent elements. Thus, we have the value of stress from the left element and also from the right element. The stresses are not equal, so we take the average stress in this nodal stress discontinuity

$$E_s \frac{b}{4} \left| \begin{pmatrix} N_1''(0), & N_2''(0), & N_3''(0), & N_4''(0) \end{pmatrix} \begin{pmatrix} V_{s,e-1} \\ \theta_{s,e-1} \\ V_{s,e} \\ \theta_{s,e} \end{pmatrix} + \right.$$

$$\left. + \begin{pmatrix} N_1''(d), & N_2''(d), & N_3''(d), & N_4''(d) \end{pmatrix} \begin{pmatrix} V_{s,e} \\ \theta_{s,e} \\ V_{s,e+1} \\ \theta_{s,e+1} \end{pmatrix} \right| \leq \sigma_{limit}.$$

This constraint must hold for nodes with the spatial coordinates x_e, where the index $e \in \{2, \ldots, N-1\}$. We can rewrite this constraints together with the constraints for end

nodes in a matrix notation as

$$E_s\frac{b}{2}\left|\begin{pmatrix} -\frac{6}{d^2} & -\frac{4}{d} & \frac{6}{d^2} & -\frac{2}{d} & 0 & & & \dots & & & 0 \\ -\frac{3}{d^2} & -\frac{2}{d^2} & \frac{6}{d^2} & 0 & -\frac{3}{d^2} & \frac{2}{d} & 0 & & \dots & & 0 \\ 0 & 0 & -\frac{3}{d^2} & -\frac{2}{d^2} & \frac{6}{d^2} & 0 & -\frac{3}{d^2} & \frac{2}{d} & 0 & \dots & 0 \\ \vdots & & & & \vdots & & & & & & \vdots \\ 0 & \dots & 0 & -\frac{3}{d^2} & -\frac{2}{d^2} & \frac{6}{d^2} & 0 & -\frac{3}{d^2} & \frac{2}{d} & 0 & 0 \\ 0 & & \dots & & 0 & -\frac{3}{d^2} & -\frac{2}{d^2} & \frac{6}{d^2} & 0 & -\frac{3}{d^2} & \frac{2}{d} \\ 0 & & & \dots & & & 0 & \frac{6}{d^2} & \frac{2}{d} & -\frac{6}{d^2} & \frac{4}{d} \end{pmatrix}\begin{pmatrix} V_{s,0} \\ \theta_{s,0} \\ V_{s,1} \\ \theta_{s,1} \\ \vdots \\ V_{s,N-1} \\ \theta_{s,N-1} \\ V_{s,N} \\ \theta_{s,N} \end{pmatrix}\right| \leq \sigma_{limit},$$

where $|\cdot|$ indicates the absolute value. The equivalent approximative equation to constraint (6.14) on the whole beam has the simple matrix form (we multiplied each element of matrix by $\frac{1}{2}$ first to make the notation more compact):

$$|E_s b\,\mathbb{C}\mathbf{V}_s| \leq \sigma_{limit}. \tag{6.22}$$

The order of accuracy of the used finite element approximation is $\mathcal{O}(d^2)$, i.e., the difference between exact and approximative solution is proportional to h^2.

6.4 IS reformulation with FEM approximations

In mathematical programming we can deal only with one objective function, so we have to create a multi-criterial, single-objective function instead of two objective functions stated in (6.9) and (6.10) by the weighted sum approach described in (3.2):

$$\min\left(-\alpha\frac{E_s a b^3}{12 c_{rigid}} + \beta\frac{\rho a b l}{c_{weight}}\right), \tag{6.23}$$

where α, β are weighting coefficients, $\alpha, \beta \in [0,1]$ and $\alpha + \beta = 1$, c_{rigid}, c_{weight} are typical values of rigidity and weight of the beam (normalizing constants). These values were obtained by author of [36] as the optimal values of objective function of two single-objective optimization problems. The maximization of function (6.10) is equivalent with minimization of the same function multiplied by (-1).

Now we can rewrite our IS reformulated two-objective beam model into a deterministic nonlinear program with derivatives approximated by the FEM and with only one objective function.

$$\min \quad \left(-\alpha\frac{E_s a b^3}{12 c_{rigid}} + \beta\frac{\rho a b l}{c_{weight}}\right), \tag{6.24}$$

$$\text{s.t.} \quad E_s a b^3 \mathbb{K}\mathbf{V}_s = \mathbf{h}, \tag{6.25}$$

$$V_{s,0} = 0, \theta_{s,0} = 0, \tag{6.26}$$

$$V_{s,N} = 0, \theta_{s,N} = 0, \tag{6.27}$$

$$|E_s b\,\mathbb{C}\mathbf{V}_s| \leq \sigma_{limit}, \tag{6.28}$$

$$a_{min} \leq a \leq a_{max}, \tag{6.29}$$

$$b_{min} \leq b \leq b_{max}, \tag{6.30}$$

where (6.25) is taken from (6.21), (6.28) is derived in (6.22) and how to obtain the objective function is described at the beginning of this section. The constraints (6.26), (6.27) are rewritten constraints for clamped end points into approximate notation and (6.29), (6.30) are the constraints from the original model limiting values of the cross section dimensions.

The results are presented for the following input data. For better scaling we did not compute with SI units but with units common in engineering computations, i.e. length is considered in mm (millimeters), weight in t (tons) and stress is given in MPa (megapascals). The load is uniform and is given per unit length $h_s(x) = 10$ Nmm^{-1}, the length of steel beam is $l = 1000$ mm with density $\rho = 7.85 \cdot 10^{-9}$ tmm^{-3}. The stress limitation is $\sigma_{limit} = 100$ MPa. Number of elements was set to $N = 100$ and bounding values of beam dimensions are $a_{min} = b_{min} = 10$ mm, $a_{max} = b_{max} = 100$ mm. The weighting coefficients are chosen as $\alpha = 0.5$, $\beta = 0.5$. The normalizing constants have the following values $c_{rigidity} = 1.80 \cdot 10^{12}$ Nmm^2, $c_{weight} = 0.007$ t, Young's modulus was found in material tables for steel, $E = 2,1 \cdot 10^5$ MPa.

Model consists of the objective function (6.24) and constraints (6.25)-(6.30) was implemented in GAMS software and it was solved with the solver CONOPT (more about this solver in the Appendix B). We obtained the optimal objective function value $z^{\mathrm{IS}}_{\min} = 0.379$. The optimal dimensions are $a^{\mathrm{IS}}_{\min} = 10$ mm and $b^{\mathrm{IS}}_{\min} = 70.707$ mm. The largest optimal deflection of beam is 0.421 mm and is placed in the middle of beam. We also computed the stress in each node by substituting the deflection vector into (6.22). The deflection and the stress on the whole beam are presented in the Figure 6.5. The largest stress 100 MPa is placed in the clamped end points.

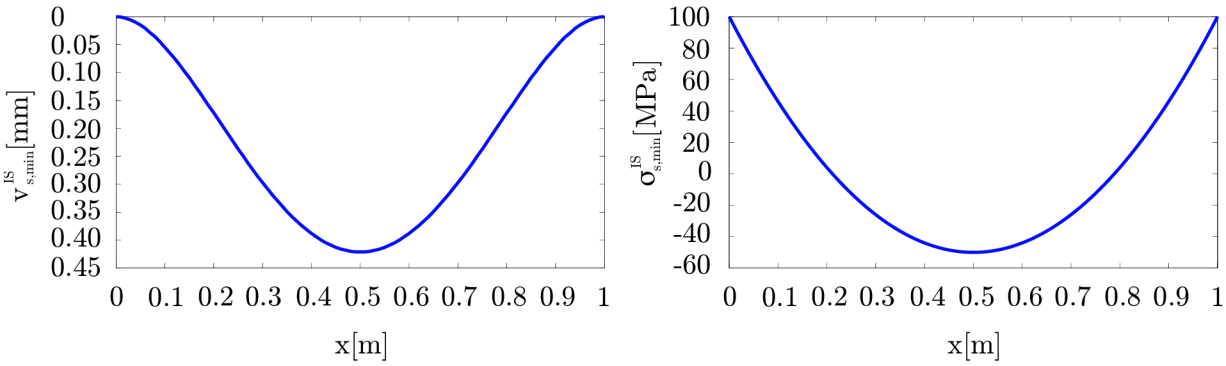

Figure 6.5: The optimal beam deflection and stress

We were interested in the influence of the mesh size h on the accuracy of the solution. Thus, we solved the model with different numbers of elements N. We used only even N to have one node in the middle of beam, where the maximum of deflection is placed. Then, we are able to compare the maximal deflection in various mesh sizes. The solutions for our model solved with different mesh sizes are listed in the Table 6.1 and plotted in the Figure 6.6.

As can be seen from the table and mainly from the figure, where $z^{\mathrm{IS}}_{\min}$, $\max(\mathbf{V}^{\mathrm{IS}}_{s,\min})$ and $b^{\mathrm{IS}}_{\min}$ are plotted as functions of N, the accuracy of obtained solution is influenced by mesh size primarily when N is small, for our case $N \leq 18$. There is no need to use too many elements N, it does not result in the significant improvement of the accuracy. It is always necessary to weigh between the accuracy and computational costs.

N[-]	$a^{\mathrm{IS}}_{\mathrm{min}}$[mm]	$b^{\mathrm{IS}}_{\mathrm{min}}$[mm]	$z^{\mathrm{IS}}_{\mathrm{min}}$[-]	$\max(\mathbf{V}^{\mathrm{IS}}_{s,\mathrm{min}})$[mm]	$\max \sigma^{\mathrm{IS}}_{s,\mathrm{min}}$[MPa]
2	10	61.237	0.332	0.648	100
4	10	68.465	0.368	0.464	100
6	10	69.722	0.374	0.439	100
8	10	70.156	0.376	0.431	100
10	10	70.356	0.377	0.427	100
12	10	70.465	0.378	0.425	100
14	10	70.530	0.378	0.424	100
18	10	70.601	0.378	0.423	100
22	10	70.638	0.378	0.422	100
28	10	70.666	0.379	0.422	100
36	10	70.683	0.379	0.421	100
50	10	70.697	0.379	0.421	100
76	10	70.705	0.379	0.421	100
100	10	70.707	0.379	0.421	100
200	10	70.710	0.379	0.421	100
300	10	70.710	0.379	0.421	100

Table 6.1: Solutions of the FEM approximated IS reformulation for different N

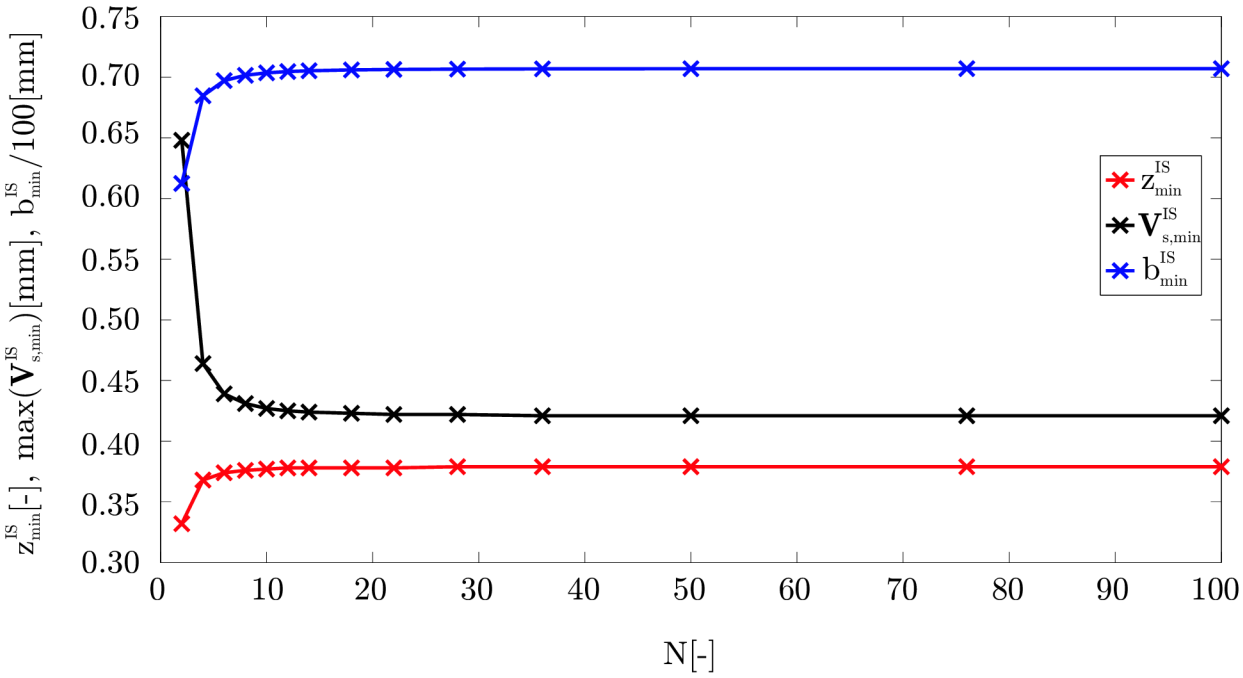

Figure 6.6: Solutions of the FEM approximated IS reformulation for different N

Comparison of FDM and FEM

We can compare our results with FDM approximation scheme results, which was implemented in [36]. We made only some additional corrections of load in GAMS code presented in the Appendix A.2 of [36] and solved the problem with the same setting of parameters.

We received completely same results as for the FEM approximation scheme, it means we obtained the optimal objective function value $z^{\mathrm{IS}}_{\min} = 0.379$ and the optimal beam cross section dimensions $a^{\mathrm{IS}}_{\min} = 10$ mm, $b^{\mathrm{IS}}_{\min} = 70.707$ mm. But if we solve the problem by the FEM, we get one extra information - approximation of the first derivative of v_s, i.e. $\theta_{s,e}$ in each node and the approximation scheme is more compact and more general. In the Figure 6.7 you can see that the result is completely same in the rate of beam deflection too, the blue line - indicates the FEM solution and the red line - indicates the FDM solution are overlapping along the whole beam. The accuracies of these methods are the same in this one-dimensional case, i.e. $\mathcal{O}(h^2)$.

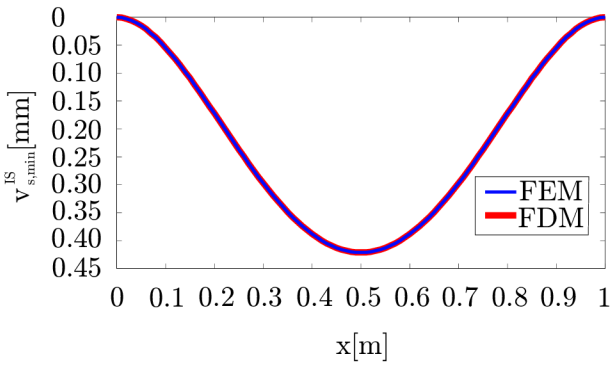

Figure 6.7: FEM, FDM beam deflection

6.5 Spatial decomposition for IS reformulation

This section provides the detailed description of the original concept of the spatial decomposition based on the progressive hedging algorithm applied on the discussed beam problem. Note that this problem is a basic test problem for the spatial decomposition concept. In fact this problem can be solved quickly and no decomposition is needed. The spatial decomposition was implemented in GAMS software as well.

Consider the steel beam with length l. We have introduced the FEM approximation technique. Thus, we can solve the model (6.24)-(6.30) on the raw mesh, let us call this mesh as *primary* mesh. We chose to solve the beam problem by using four element mesh, $N = 4$ and we obtained the deflection $V_{s,e}$ and the rotation $\theta_{s,e}$ for $e \in \{0, 1, 2, 3, 4\}$

$$\overline{\mathbf{V}}_s = \big(\ 0,\quad 0,\quad 0.261,\quad 0.00,\quad 0.464,\quad 0,\quad 0.261,\quad -0.001,\quad 0,\quad 0\ \big)^{\mathrm{T}}\,\mathrm{mm}.$$

We can use these values as boundary conditions for subproblems created by the spatial decomposition. It is important to realize that we solved only a small-scale problem consuming the short computing time thanks to using only a few of elements. Hence, we got an inaccurate solution that was computed earlier and can be found in the Table 6.1. The described situation is presented in the Figure 6.8. The values that will be used as boundary conditions are marked by black circles. As you can see only the solution in the middle node will not be used in following computations.

The next step of our concept lies in the decomposition of problem's domain into two overlapping subdomains. We have several possibilities how to carry out the decomposition but only one possibility for our choice of the primary mesh gives the same only rotated structure of problem for both parts, this symmetric possibility was used for its clarity

and simplicity and is illustrated in the Figure 6.8, where $x_{p,e}$ is the coordinate of the e-th node on the p-th part and the length of one part is $l_p = 750$ mm.

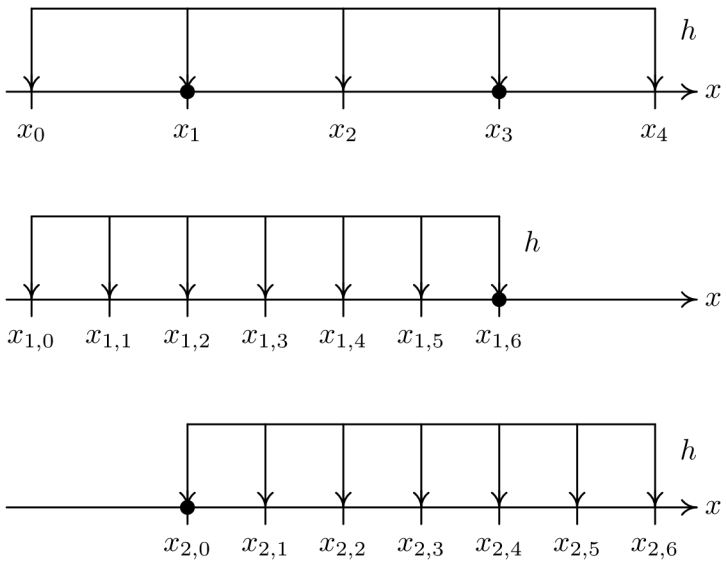

Figure 6.8: Decomposed structure of the steel beam

We have to employ a *secondary* finer mesh on the both subdomains. A secondary mesh has to be chosen to get values of deflection and their slopes in the same spatial points on both parts. We chose the number of elements $N = 6$ and solved both subproblems with mentioned boundary conditions. But these conditions are not longer equal to zero as in the original model. So we have to modify the model a little bit. Let us define a vectors $\mathbf{V}_{s,p}^{\mathrm{bc},0}$ with lengths $2(N+1)$ for $p \in \{1,2\}$ filled by nonzero boundary conditions on appropriate places, the rest of elements is put equal to zero. The upper index 0 indicates pre-iteration of the PHA, in that the objective function does not contain any penalty terms.

$$\mathbf{V}_{s,1}^{\mathrm{bc},0} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \overline{V}_{s,3} \\ \overline{\theta}_{s,3} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0.261 \\ -0.001 \end{pmatrix}, \mathbf{V}_{s,2}^{\mathrm{bc},0} = \begin{pmatrix} \overline{V}_{s,1} \\ \overline{\theta}_{s,1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0.261 \\ 0.001 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

We have to modify the equation (6.25) to the equation containing non-zero boundary conditions and also the constraints (6.26) and (6.27) to obtain the models for $p \in \{1,2\}$

$$\begin{aligned} \min \quad & \left(-\alpha \frac{E_s a b^3}{12 c_{rigid}} + \beta \frac{\rho a b l_p}{c_{weight}} \right), \\ \text{s.t.} \quad & E_s a b^3 \mathbb{K} \mathbf{V}_{s,p}^0 = \mathbf{h} - E_s a b^3 \mathbb{K} \mathbf{V}_{s,p}^{\mathrm{bc},0}, \\ & V_{s,p,0}^0 = V_{s,p,0}^{\mathrm{bc},0}, \theta_{s,p,0}^0 = \theta_{s,p,0}^{\mathrm{bc},0}, \\ & V_{s,p,N}^0 = V_{s,p,N}^{\mathrm{bc},0}, \theta_{s,p,N}^0 = \theta_{s,p,N}^{\mathrm{bc},0}, \\ & \left| E_s b \, \mathbb{C} \mathbf{V}_s^0 \right| \leq \sigma_{limit}, \\ & a_{min} \leq a \leq a_{max}, \\ & b_{min} \leq b \leq b_{max}. \end{aligned}$$

We solved these models on different subdomains independently (i.e., possibly in parallel) and then we can compute the initial average solutions $\hat{\mathbf{V}}^0_{s,p}$ for $p \in \{1,2\}$ for the "warm" start of our algorithm

$$\begin{aligned}\hat{\mathbf{V}}^0_{s,1} &= \left(\; V^0_{s,1,0}, \;\; \ldots, \;\; \theta^0_{s,1,1}, \;\; \tfrac{V^0_{s,1,2}+V^0_{s,2,0}}{2}, \;\; \ldots, \;\; \tfrac{\theta^0_{s,1,6}+\theta^0_{s,2,4}}{2} \; \right)^{\mathrm{T}}, \\ \hat{\mathbf{V}}^0_{s,2} &= \left(\; \tfrac{V^0_{s,1,2}+V^0_{s,2,0}}{2}, \;\; \ldots, \;\; \tfrac{\theta^0_{s,1,6}+\theta^0_{s,2,4}}{2}, \;\; V^0_{s,2,5}, \;\; \ldots, \;\; \theta^0_{s,2,6} \; \right)^{\mathrm{T}}.\end{aligned} \tag{6.31}$$

We can average only on the overlap, we simply take the proper value of the deformation out of the overlap. Let us make the notation clearer. We used four indices for the deflection $V^j_{s,p,e}$, where s is previously used index for realization of uncertain parameters by the scenario s, p indicates different parts and $p \in \{1,2\}$, e is the nodal index and j is the counter of iterations, similarly for slopes.

Finally we can employ a modified PHA algorithm for the spatial decomposition. For this purpose we have to define some aiding operators. First one is a localization operator $\mathbf{1}_0$ to be able to work with translations and rotations separately. They have different dimensions so we need to use different penalty parameters. We also need the vector $\mathbf{1}$ filled by ones.

$$\mathbf{1}_0 = \left(\; 1, \;\; 0, \;\; 1, \;\; 0, \;\; \ldots, \;\; 1, \;\; 0, \; \right)^{\mathrm{T}}, \mathbf{1} = \left(\; 1, \;\; 1, \;\; 1, \;\; 1, \;\; \ldots, \;\; 1, \;\; 1, \; \right)^{\mathrm{T}},$$

where the length of $\mathbf{1}_0$ and $\mathbf{1}$ is $2(N+1)$. For the same reason we define localization square matrices of the size $2(N+1) \times 2(N+1)$:

$$\mathbb{I}_0 = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & & \vdots & & & \vdots \\ 0 & \ldots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \mathbb{I} = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & \ldots & 0 & 1 & 0 \\ 0 & \ldots & 0 & 0 & 1 \end{pmatrix}.$$

Then proceed to own modified progressive hedging algorithm step by step, some comments and notes follow.

0. Choose the penalty parameters $\varrho_v = 40 > 0$, $\varrho_\theta = 40 \cdot 10^3 > 0$ and the termination parameter $\varepsilon = 10^{-3} > 0$. Set $\mathbf{w}^0_p = \mathbf{0}$, use initial average solutions $\hat{\mathbf{V}}^0_{s,p}$ from (6.32) for $p = 1,2$ and set the iteration counter as $j = 1$.

1. For $p = 1,2$ solve the problem

$$\begin{aligned} \min \;\; & \left(-\alpha \frac{E_s a b^3}{12 c_{rigid}} + \beta \frac{\rho a b l_p}{c_{weight}} + (\mathbf{w}^{j-1}_p)^{\mathrm{T}} \mathbf{V}^j_{s,p} + \frac{1}{2} \varrho_v \left\| \mathbf{1}_0^{\mathrm{T}} (\mathbf{V}^j_{s,p} - \hat{\mathbf{V}}^{j-1}_{s,p}) \right\|^2 + \right. \\ & \left. + \frac{1}{2} \varrho_\theta \left\| (\mathbf{1} - \mathbf{1}_0)^{\mathrm{T}} (\mathbf{V}^j_{s,p} - \hat{\mathbf{V}}^{j-1}_{s,p}) \right\|^2 \right), \\ \text{s.t.} \;\; & E_s a b^3 \mathbb{K} \mathbf{V}^j_{s,p} = \mathbf{h} - E_s a b^3 \mathbb{K} \mathbf{V}^{\mathrm{bc},j}_{s,p}, \\ & \left| E_s b \, \mathbb{C} \mathbf{V}^j_{s,p} \right| \le \sigma_{limit}, \\ & a_{min} \le a \le a_{max}, \\ & b_{min} \le b \le b_{max}. \end{aligned}$$

and denote its solutions as $\mathbf{V}^j_{s,p}$.

2. Calculate average solutions on both parts

$$\begin{aligned}\hat{\mathbf{V}}_{s,1}^{j} &= \left(\; V_{s,1,0}^{j}, \;\; \ldots, \;\; \theta_{s,1,1}^{j}, \;\; \tfrac{V_{s,1,2}^{j}+V_{s,2,0}^{j}}{2}, \;\; \ldots, \;\; \tfrac{\theta_{s,1,6}^{j}+\theta_{s,2,4}^{j}}{2} \; \right)^{\mathrm{T}}, \\ \hat{\mathbf{V}}_{s,2}^{j} &= \left(\; \tfrac{V_{s,1,2}^{j}+V_{s,2,0}^{j}}{2}, \;\; \ldots, \;\; \tfrac{\theta_{s,1,6}^{j}+\theta_{s,2,4}^{j}}{2}, \;\; V_{s,2,5}^{j}, \;\; \ldots, \;\; \theta_{s,2,6}^{j} \; \right)^{\mathrm{T}}.\end{aligned} \tag{6.32}$$

3. Evaluate the termination condition

$$\delta = \left(\sum_{p=1}^{2} \left\| \hat{\mathbf{V}}_{s,p}^{j-1} - \hat{\mathbf{V}}_{s,p}^{j} \right\|^2 + \frac{1}{2} \left\| \mathbf{V}_{s,p}^{j} - \hat{\mathbf{V}}_{s,p}^{j} \right\|^2 \right)^{\frac{1}{2}} \leq \varepsilon.$$

If the condition holds, then stop, the solution to the problem with given tolerance ε has the form

$$\begin{aligned}\hat{\mathbf{V}}_{s}^{j} = \Big(\; & V_{s,1,0}^{j}, \;\; \theta_{s,1,0}^{j}, \;\; V_{s,1,1}^{j}, \;\; \theta_{s,1,1}^{1}, \;\; \tfrac{V_{s,1,2}^{j}+V_{s,2,0}^{j}}{2}, \;\; \tfrac{\theta_{s,1,2}^{j}+\theta_{s,2,0}^{j}}{2}, \;\; \ldots \\ & \ldots, \;\; \tfrac{V_{s,1,6}^{j}+V_{s,2,4}^{j}}{2}, \;\; \tfrac{\theta_{s,1,6}^{j}+\theta_{s,2,4}^{j}}{2}, \;\; V_{s,2,5}^{j}, \;\; \theta_{s,2,5}^{j}, \;\; V_{s,2,6}^{j}, \;\; \theta_{s,2,6}^{j} \; \Big)\end{aligned}$$

Otherwise, calculate for $p \in \{1, 2\}$

$$\mathbf{w}_{p}^{j} = \mathbf{w}_{p}^{j-1} + \varrho_{v} \mathbb{I}_{0} (\mathbf{V}_{s,p}^{j} - \hat{\mathbf{V}}_{s,p}^{j}) + \varrho_{\theta} (\mathbb{I} - \mathbb{I}_{0}) (\mathbf{V}_{s,p}^{j} - \hat{\mathbf{V}}_{s,p}^{j})$$

and

$$\mathbf{V}_{s,1}^{\mathrm{bc},j} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ V_{s,1,N}^{j} \\ \theta_{s,1,N}^{j-1} \end{pmatrix}, \mathbf{V}_{s,2}^{\mathrm{bc}} = \begin{pmatrix} V_{s,2,1}^{j} \\ \theta_{s,2,1}^{j} \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

set $j = j + 1$, and return to step 1 of algorithm.

Let us add some notes. One important note is connected with boundary conditions. The conditions can be used as explicit constraints only in the pre-iteration $j = 0$. If we used them in all iterations, we would get optimal deflection with jumps and we would not expect a high accuracy. We can think about them as some starting values.

Penalty terms don't modify the value of objective function at the end of algorithm, their absolute values are negligible. The optimal value of objective function is not comparable to the optimal value of objective function on the whole beam because we are working only with $l_p = 750$ mm long subdomains and the length is contained in the objective function explicitly. But the value can be easily recomputed from the subdomain to the whole beam:

$$z_{s}^{j} = z_{s,p}^{j} + \beta \frac{\rho a b}{c_{weight}} (l - l_p).$$

So all values of the objective function are recomputed to the whole beam in the following text to have a comparison with the solution obtained earlier.

We used the earlier defined Euclidean norm in steps 1 and 3 of the modified PHA. We can also try to use some other norms, if the obtained solution is unsatisfactory for us.

The modified PHA can be written in more compact form of course, but we choose to write it in this extensive form for better understanding and illustrating the computing procedure in details.

The whole procedure was implemented in GAMS with mentioned primary and secondary meshes, penalty parameters from the step 0 of the PHA were used. The values of penalty parameters were determined by comparison the objective function values and the difference between a deflection and an average deflection and similarly for slopes. The same data as in the non-decomposed IS model implementation was used. The example of the source code from the GAMS implementation is listed in the Appendix D. The optimal solution with the required accuracy was reached in 26 iterations. Obtained optimal beam cross section dimensions are $a^{26} = 10$ mm and $b^{26} = 70.145$ mm, the recomputed optimal value of the objective function is $z_s^{26} = 0.376$ related to the whole beam. In the Figure 6.9 the deflection rates in the first three iterations are presented. The position of maximal deflection is corrected to the right point - the middle of the beam.

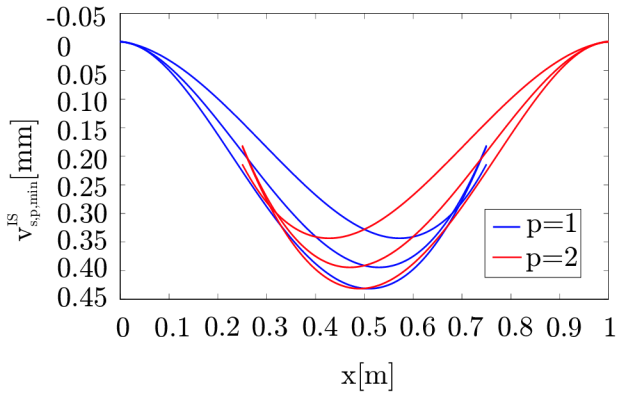

Figure 6.9: First three iterations of PHA

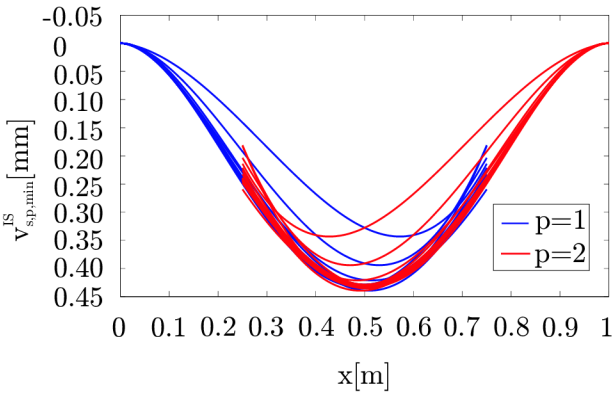

Figure 6.10: All iterations of PHA

The rest of iterations is illustrated in the Figure 6.10. Note that we obtained only 7 values of deflection in each iteration. Hence, we made an interpolation by a polynomial to plot the graph of the deflection in each iteration. The optimal deflection rate with the maximal deflection value 0.431 mm is placed in the largest concentration of rates.

Penalty parameter ϱ_v	Penalty parameter ϱ_θ	Number of iterations j
40	$40 \cdot 10^3$	26
10	$10 \cdot 10^3$	25
5	$5 \cdot 10^3$	25
1	$1 \cdot 10^3$	24
0.1	$0.1 \cdot 10^3$	27
0.01	$0.01 \cdot 10^3$	27

Table 6.2: The number of iterations needed for different penalty parameters

The algorithm is very sensitive to the choice of the penalty parameter. Furthermore, we have two penalty parameters. One of them ϱ_v caused overshoot up and the second one ϱ_θ caused overshoot down from the optimal solution (see Figure 6.10). The effect

of penalty parameters is illustrated in the Table 6.2. This table for $\varepsilon = 10^{-3}$ shows that we could save two iterations of the PHA by better choice of penalty parameters, i.e. $\varrho_v = 1$ and $\varrho_\theta = 1 \cdot 10^3$.

Length of overlap

One more interesting point is how the length of the overlap influences the accuracy of the solution obtained by the spatial decomposition. Therefore, we solved our problem with different lengths of the overlap and listed results in following tables. In the Figure 6.11 you can see the designs of different symmetric overlaps. We used the primary mesh with 8 finite elements, hence we had three possibilities for the symmetric overlap. It is the 6 elements long overlap, the 4 and the 2 elements long overlap. Then, we introduced secondary finer mesh by dividing each element from primary mesh into two parts and solved decomposed problem iteratively by the modified PHA. We set $\varrho_v = 20$ and $\varrho_\theta = 20 \cdot 10^3$. The rest of the used data was the same as earlier.

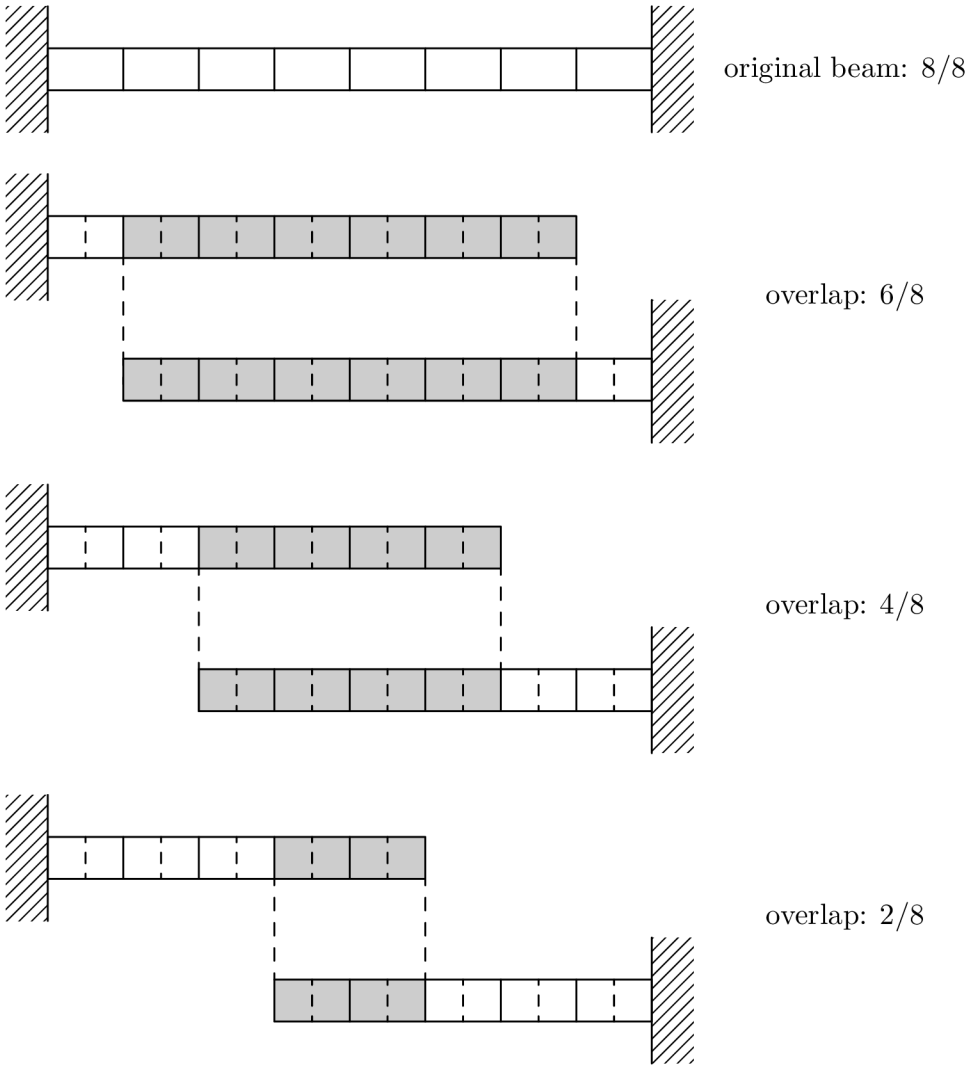

Figure 6.11: Different overlap's lengths

We define one more distance parameter δ_0 because of evaluation reasons. This parameter takes into account the distance between a^j and $a^{\mathrm{IS}}_{\mathrm{min}}$ obtained on the whole beam

with $N = 100$ finite elements - we consider this solution as accurate enough, b^j and $b^{\mathrm{IS}}_{\min}$ and also the distance between $\max(\mathbf{V}^j_s)$ and $\max(\mathbf{V}^{\mathrm{IS}}_{s,\min})$. The optimal value of the objective function was not considered because its value depends on already considered values of cross section dimensions. The distance parameter is defined as follows

$$\delta_0 = \sqrt{(a^{\mathrm{IS}}_{\min} - a^j)^2} + \sqrt{(b^{\mathrm{IS}}_{\min} - b^j)^2} + 10\sqrt{\left(\max\{\mathbf{V}^{\mathrm{IS}}_{s,\min} - \mathbf{V}^j_s\}\right)^2}.$$

We solved the decomposed problem for different overlap lengths and different given tolerances ε and wrote down how many iterations we needed to fulfill the tolerance and also the recomputed values of the objective function, values of cross-section dimensions and maximal deflections of beam. We also computed the distance from the solution obtained on the whole beam with $N = 100$ and listed all results in the Table 6.3.

ε	overlap length	j	z^j_s[-]	$\max \mathbf{V}^j_s$[mm]	a^j[mm]	b^j[mm]	δ_0
0.005	6/8	25	0.379	0.424	10.000	70.647	0.089
	4/8	51	0.378	0.435	10.000	70.822	0.153
	2/8	39	0.470	0.460	15.178	56.746	20.148
0.0025	6/8	26	0.379	0.424	10.000	70.657	0.079
	4/8	55	0.378	0.423	10.000	70.438	0.286
	2/8	64	0.497	0.549	16.787	54.225	24.546
0.001	6/8	29	0.378	0.423	10.000	70.545	0.184
	4/8	59	0.378	0.424	10.000	70.629	0.104
	2/8	96	0.513	0.564	17.753	52.874	27.017
0.0001	6/8	35	0.379	0.423	10.000	70.570	0.161
	4/8	69	0.378	0.423	10.000	70.571	0.160
	2/8	175	0.523	0.573	18.333	52.113	28.449
0.00001	6/8	42	0.379	0.423	10.000	70.572	0.159
	4/8	77	0.378	0.423	10.000	70.572	0.159
	2/8	253	0.523	0.574	18.390	52.039	28.590

Table 6.3: Results for different overlap lengths with the δ termination condition

The tolerance ε is connected with the original distance parameter δ, this parameter evaluates distance between solutions in two foregoing iterations. This tolerance gives us an information about the speed of convergence of the algorithm but not about the distance from the exact solution. Therefore, we decided to modify the stop condition according to the distance from the solution obtained on the whole beam with $N = 100$ elements characterized by the distance parameter δ_0 and the tolerance ε_0. We solved the model again for different accuracies and listed results in the Table 6.4.

Now, we can evaluate the results listed in the Table 6.3 and in the Table 6.4. The optimal solution obtained by the spatial decomposition into two parts with the shortest overlap is not accurate enough. It is even getting worse with additional iterations. Thus, the length of 2 elements, i.e. one quarter of beam, is not sufficient. The algorithm with the longest overlap needed the smallest number of iterations but the number of equations is not reduced enough by the decomposition. The overlap with the length of the half

ε_0	overlap length	j	z_s^j[-]	$\max \mathbf{V}_s^j$[mm]	a^j[mm]	b^j[mm]
6.5	6/8	5	0.339	0.390	10.000	72.114
	4/8	4	0.585	0.465	13.421	71.202
	2/8	2	0.475	0.426	10.541	76.437
2	6/8	5	0.339	0.390	10.000	72.114
	4/8	10	0.332	0.378	10.000	70.416
	2/8	> 500	–	–	–	–
1	6/8	15	0.371	0.421	10.000	70.513
	4/8	10	0.332	0.378	10.000	70.416
	2/8	> 500	–	–	–	–
0.1	6/8	21	0.380	0.423	10.000	70.787
	4/8	43	0.377	0.428	10.000	70.691
	2/8	> 500	–	–	–	–
0.051	6/8	> 500	–	–	–	–
	4/8	53	0.379	0.423	10.000	70.735
	2/8	> 500	–	–	–	–

Table 6.4: Results for different overlap lengths with the δ_0 termination condition

of beam provided the best results. We also managed to find the best approximation of the exact solution by this choice of the overlap.

It could be interesting to know how close to the solution obtained on the mesh with $N = 100$ elements we are on other meshes. Hence, we evaluated the δ_0 distance parameter for different numbers of elements N.

Number of elements N	2	4	10	...	26	28	...	100
Distance parameter δ_2	64.917	2.668	0.414	...	0.057	0.048	...	0

Table 6.5: The distance parameter δ_0 for different N

From the Table 6.5 is clear that by the spatial decomposition procedure we are able to obtain the solution with the roughly same accuracy as the accuracy of the solution computed directly on the mesh with $N = 26$ elements is. Let us repeat that this accuracy was gained on the primary mesh with 8 elements and the secondary mesh with 12 elements, the used length of overlap was one half of the whole beam length and we needed 53 iterations of the PHA.

6.6 EO reformulation with FEM approximations

We assume random Young's modulus, the randomness of Young's modulus can be caused by different heat-treating processes of steel such as forming, rolling, annealing or by different quality of the material. We were dealing with the IS reformulation in the previous text. We used only one chosen scenario to represent the random Young's modulus but we have

more possibilities how to work with the uncertainty, for example the EO deterministic reformulation.

The random variable must be approximated by the scenario-based approach. Therefore, the random Young's modulus $E(\xi)$ is represented by a realizations $E(\xi_s) = E_s$, $s = 1, \ldots, R$. The continuous two-stage stochastic nonlinear program (6.1)-(6.8) is approximated by a large multi-objective deterministic nonlinear program. The FEM method was used again to approximate the derivatives in the model.

$$\min \ \rho abl, \tag{6.33}$$

$$\max \ \sum_{s=1}^{R} p_s \frac{E_s a b^3}{12}, \tag{6.34}$$

$$\text{s.t.} \ E_s a b^3 \mathbb{K}\mathbf{V}_s = \mathbf{h}, s = 1, \ldots, R, \tag{6.35}$$

$$V_{s,0} = 0, \theta_{s,0} = 0, s = 1, \ldots, R, \tag{6.36}$$

$$V_{s,N} = 0, \theta_{s,N} = 0, s = 1, \ldots, R, \tag{6.37}$$

$$|E_s b \, \mathbb{C}\mathbf{V}_s| \leq \sigma_{limit}, s = 1, \ldots, R, \tag{6.38}$$

$$a_{min} \leq a \leq a_{max}, \tag{6.39}$$

$$b_{min} \leq b \leq b_{max}. \tag{6.40}$$

This multi-objective program can be modified to single-objective one by the same weighted sum approach as earlier. The objective functions (6.33) and (6.34) are replaced by the objective function:

$$\min_{a,b} \left(-\alpha \sum_{s=1}^{R} p_s \frac{E_s a b^3}{12 c_{rigid}} + \beta \frac{\rho abl}{c_{weight}} \right). \tag{6.41}$$

The model with the objective function (6.41) and constraints (6.35)-(6.40) was implemented in GAMS with the same data as was used in the IS reformulation. The Young's modulus was assumed random:

$$E_s = 2 \cdot 10^5 \text{ MPa} + E_{random,s}, \text{ where } E_{random,s} \sim U(-1 \cdot 10^4, 5 \cdot 10^4) \text{ MPa},$$

where $U(a, b)$ is the continuous uniform distribution on the support $[a, b]$. We restricted the number of scenarios to $R = 3$ because we will deal with the spatial decomposition further and we want to maintain the clarity of results and the implementation. The proper values were generated by pseudorandom values generator from uniform distribution in MATLAB:

$$E_1 = 1.9714 \cdot 10^5 \text{ MPa}, E_2 = 2.1990 \cdot 10^5 \text{ MPa}, E_3 = 2.4758 \cdot 10^5 \text{ MPa}.$$

The representation of random variable is quite simple, we should use hundreds of scenarios to acceptable representation. But we only want to illustrate how our algorithm works for different types of reformulation. This problem was solved using the Monte Carlo technique and large number of scenarios in [36].

We solved the reformulation directly with $N = 100$ elements and we obtained the optimal objective function value $z_{\min}^{\text{EO}} = 0.379$. Thus, the inequality from the Theorem 3.5.1 is fulfilled with the equality because of independence the value of the objective function on here-and-know variables, i.e., the deflection. The optimal dimensions are $a_{\min}^{\text{EO}} = 10$ mm and $b_{\min}^{\text{EO}} = 70.707$ mm. The largest optimal deflection of beam is placed in the middle of beam for all three scenarios. The deflections and the stresses along the whole beam are presented in the Figure 6.12. The stress rate is same for all scenarios, because it is independent of the realization of the random variable, the reasons are described in [36].

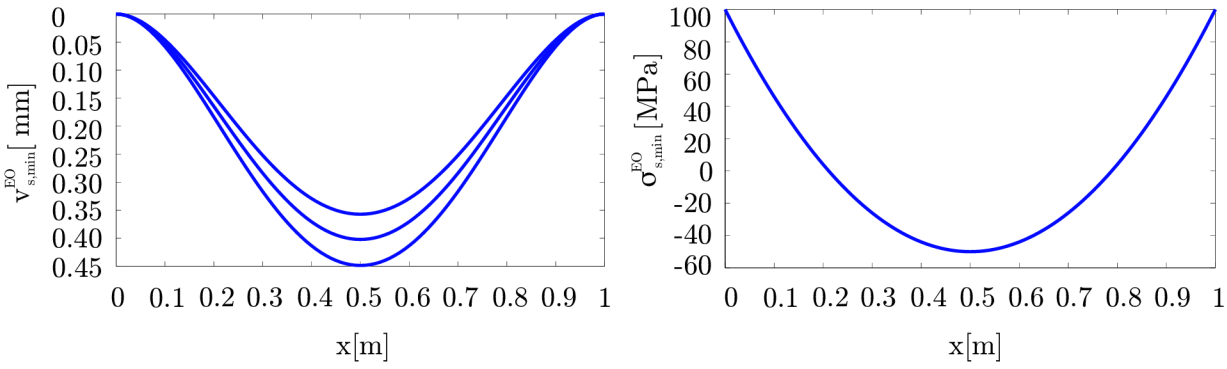

Figure 6.12: The optimal beam deflection and stress

6.7 Spatial decomposition for EO reformulation

We will not specify the spatial decomposition algorithm step by step for the EO reformulation. The main idea is the same as for the IS reformulated problem. We only point out some interesting details and comment the results.

We can employ the spatial decomposition approach or the spatial decomposition approach combined with the scenario decomposition. The first approach is nearly the same as the worked out decomposition for IS reformulation. Only one difference is in penalty parameters contained in the modified objective function, we have to add the penalty terms for each scenario. The objective function for the PHA part of decomposition is then in the following form

$$\min \left(-\alpha \sum_{s=1}^{R} p_s \frac{E_s a b^3}{12 c_{rigid}} + \beta \frac{\rho a b l}{c_{weight}} + (\mathbf{w}_p^{j-1})^{\mathrm{T}} \mathbf{V}_{s,p}^{j} + \frac{1}{2} \varrho_v \left\| \mathbf{1}_0^{\mathrm{T}} (\mathbf{V}_{s,p}^{j} - \hat{\mathbf{V}}_{s,p}^{j-1} \right\|^2 + \right.$$
$$\left. + \frac{1}{2} \varrho_\theta \left\| (\mathbf{1} - \mathbf{1}_0)^{\mathrm{T}} (\mathbf{V}_{s,p}^{j} - \hat{\mathbf{V}}_{s,p}^{j-1}) \right\|^2 \right),$$

where $\hat{\mathbf{V}}_{s,p}^{j}$ is computed for each scenario separately. The corresponding constraints have to be fulfill for all scenarios $s \in \{1, \ldots, R\}$. Note that we are not penalize the cross section dimensions a, b between parts nor scenarios. This approach enables decompose our model into two spatial subproblems. This idea is explained deeper at the end of this section.

If we employ the spatial decomposition together with the scenario decomposition we can obtain decomposition into $2R$ subproblems and reduce the computational time significantly. We used only three scenarios in the EO reformulation so we obtained 6 subproblems, each of them has the objective function

$$\min \left(-\alpha \frac{E_s a b^3}{12 c_{rigid}} + \beta \frac{\rho a b l}{c_{weight}} + (\mathbf{w}_p^{j-1})^{\mathrm{T}} \mathbf{V}_{s,p}^{j} + \frac{1}{2} \varrho_v \left\| \mathbf{1}_0^{\mathrm{T}} (\mathbf{V}_{s,p}^{j} - \hat{\mathbf{V}}_{s,p}^{j-1} \right\|^2 + \right.$$
$$\left. + \frac{1}{2} \varrho_\theta \left\| (\mathbf{1} - \mathbf{1}_0)^{\mathrm{T}} (\mathbf{V}_{s,p}^{j} - \hat{\mathbf{V}}_{s,p}^{j-1}) \right\|^2 \right),$$

where $s = 1, 2, 3$ and $p = 1, 2$. The appropriate constraints must be fulfilled. Both approaches were implemented with almost same results presented further.

We solved the decomposed model iteratively in GAMS software with penalty parameters $\varrho_v = 10$, $\varrho_\theta = 10 \cdot 10^3$. The optimal solution with the accuracy $\varepsilon = 10^{-4}$ was reached in $j = 72$ iterations. The found optimal beam cross section dimensions values are $a^{72} = 10$ mm and $b^{72} = 70.158$ mm. The optimal value of the objective function recomputed to the whole beam is $z^{72} = 0.375$. The obtained deflections and stresses for the scenarios $s = 1, 2, 3$ are plotted in the Figure 6.13 by dashed red lines, we can compare them with solutions from the previous section plotted by blue lines.

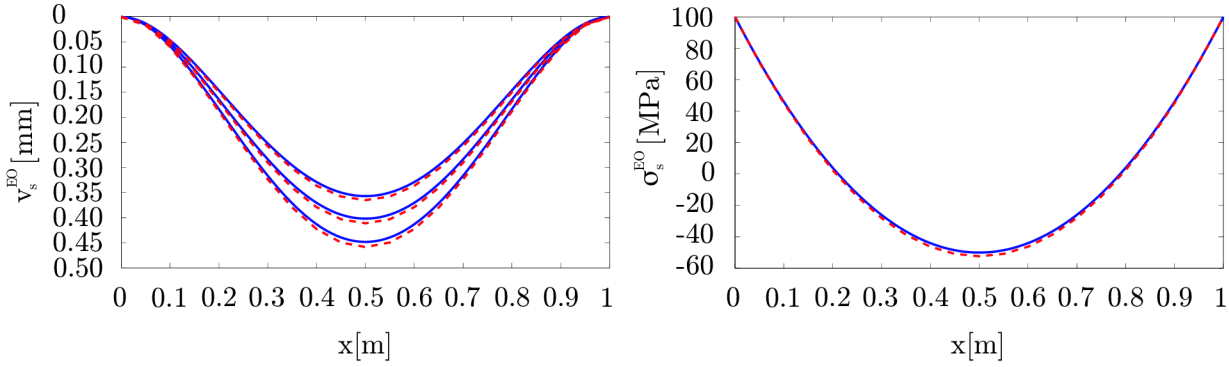

Figure 6.13: Comparison of the spatial decomposition with the direct solution

The accuracy $\varepsilon = 10^{-3}$ used for IS reformulation was reached in 33 iterations but the absolute value of penalty terms was not small enough. The spatial decomposition of the EO reformulation requires generally more iterations then the spatial decomposition of the IS reformulation because the objective function contains more penalty terms to minimize or more subproblems have to be solved.

Note that we have not included penalty terms on all possible here-and-now variables related to the first stage. The question is why we found it useful.

When we formulated the problem, we noted that cross-section dimensions are here-and-now variables. Hence, they are not dependent on the realization of the uncertainty by a scenario. Nevertheless, these variables can depend on the appropriate spatial part. The use of our approach is supported by the following idea.

Consider a simple optimization problem

$$\begin{aligned} &\min \ f(x, y), \\ &\text{s.t. } \ y = g(x), \end{aligned}$$

that can be decomposed to two fully independent subproblems

$$\begin{aligned} &\min \ f(x_1, y_1), \\ &\text{s.t. } \ y_1 = g(x_1), \end{aligned}$$

and

$$\begin{aligned} &\min \ f(x_2, y_2), \\ &\text{s.t. } \ y_2 = g(x_2). \end{aligned}$$

Further we have to require the fulfillment of the nonanticipativity constraints

$$y_1 = y_2,$$
$$x_1 = x_2.$$

The fulfillment of these constraints can be guaranteed by adding a penalty term for the difference between y_1, y_2 and x_1, x_2 respectively to the objective function.

However, we may use only penalty term related to x_1 and x_2 and leave the term related to dependent variables y_1 and y_2 as we indirectly utilize constraints $y_1 = g(x_1)$, $y_2 = g(x_2)$. This penalty term guarantees that $\|x_1 - x_2\| \to 0$ and in case of the locally uniqueness of the optimal solution also that $\|y_1 - y_2\| \to 0$. If we used both penalty terms for the difference of x_1 and x_2 and also for the difference of y_1, y_2, we would add in fact the redundant constraint that creates computational problems in the runs of algorithms.

This idea explains why we were averaging only the deflection $\mathbf{V}_{s,p}$ and not cross section dimensions a, b on different parts in the implementation of both reformulations.

Chapter 7

Conclusions

The applicability of the spatial decomposition approach to two deterministic reformulation of one civil engineering problem has been discussed. Problem has been concerned in the optimal design of beam cross section dimensions with a random Young's modulus. The model led to the ODE constrained multi-objective stochastic nonlinear program.

In general, the proposed computational scheme consisting of the modified progressive hedging algorithm and some additional steps applicable for approximated ODE/PDE constrained deterministic and also stochastic programs seems to be robust enough for future applications to advanced large-scale optimization problems in which a decomposition is required.

The spatial decomposition has been implemented and tested with respect to future possibilities of parallel computing of large engineering problems. The implementation has shown that the approach can be used even if the mathematical conditions for the convergence are not fulfilled but still a suitable starting point can be found.

Main disadvantage of the progressive hedging algorithm that forms the basis of the spatial decomposition is that the performance of the algorithm is very sensitive to the choice of the penalty parameter ϱ. Unfortunately, there is no general rule how to determine the best value of this parameter. Furthermore, in the spatial decomposition several penalty parameters are contained. The convergence performance of the PHA can be improved by some heuristic techniques allowing the updating ϱ in each step. Many aspects of the procedures need further investigations. In particular, we need a method for adjusting the penalties to ensure fast convergence. For any realistic problem the number of scenarios will be formidable.

Future research could concern in the practical parallel implementation of the spatial decomposition on multiple processors computers and in testing the algorithm for other two-dimensional and also three-dimensional problems. It is not recommended to treat up whole model in GAMS because the finite element method can not be comfortably used in this software. Hence, it could be good to connect some other software with GAMS as a solver.

Bibliography

[1] N. Andréasson, A. Evgrafov, M. Patriksson. *An Introduction to Continuous Optimization.* Studentlitteratur, Lund, 2009.

[2] M. S. Bazaraa, H. D. Sherali, C. M. Shetty. *Nonlinear Programming: Theory and Algoritms.* A John Wiley & Sons, New Jersey, 2006.

[3] J. F. Benders. *Partioning procedures for solving mixed-variables programming problems.* Numerische Mathematik 4, pages 238-252, 1962.

[4] M. A. Bhatti. *Practical Optimization Methods With Mathematica Applications.* Springer-Verlag, New York, 2000.

[5] J. R. Birge, F. Louveaux. *Introduction to Stochastic Programming.* Springer-Verlag, New York, 1997.

[6] L. T. Biegler, O. Ghattas, M. Heinkenschloss, B. van Bloemen Waanders. *Large-Scale PDE-Constrained Optimization.* Springer-Verlag, Berlin Heidelberg, 2003.

[7] M. R. Bussieck, L. S. Lasdon, J. D. Pintér, N. V. Sahinidis. *Global Optimization with GAMS Applications and Performance.* GAMS Development Corporation, 2003. Available at `http://gams.com/presentations/present_BLOglobal.pdf`. [Online; cited December 25, 2011].

[8] E. Castillo, R. Minguez, A. J. Conejo, R. Garcia-Bertrand. *Decomposition Techniques in Mathematical Programming.* Springer Science+Business Media, Berlin, 2006.

[9] L. Čermák. *Řešení eliptických a parabolických problémů druhého řádu s obecnými okrajovými podmínkami metodou konečných prvků.* PhD thesis, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 1981.

[10] G. B. Dantzig, M. N. Thapa. *Linear Programming, 1: Introduction.* Springer series in operations research, New York, 1997.

[11] A. de Silva, D. Abramson. *Computational Experience with the Parallel Progressive Hedging Algorithm for Stochastic Linear Programs.* Griffith University, Nathan, 1994.

[12] A. S. Drud. *CONOPT.* ARKI Consulting and Development A/S, Bagsvaerd, Denmark. Available at `http://www.gams.com/dd/docs/solvers/conopt.pdf`. [Online; cited December 25, 2011].

[13] M. Feistauer, A. Ženíšek. *Finite Element Solution of Nonlinear Elliptic Problems.* Numerische Mathematik, Volume 50, Number 4, pages 451-475, 1986.

[14] C. A. Felippa. *Introduction to Finite Elements Methods.* Lecture Notes, University of Colorado at Boulder, 2001.

[15] J. Franců. *Moderní metody řešení diferenciálních rovnic.* Akademické nakladatelství CERM, 2. rozšířené vydání, Brno, 2006.

[16] G. Grimmett, D. Stirzaker. *Probability and Random Processes.* Oxford University Press Inc., New York, 2001.

[17] P. Hansbo. *Computations Engineering.* Lecture Notes, Chalmers University of Technology at Göteborg, 2009.

[18] T. Helgason, S. W. Wallace. *Approximate scenario solutions in the progressive hedging algorithm: A numerical study with an application to fisheries management.* Annals of Operations Research 31, pages 425-444, 1991.

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

[20] L. Klimeš. *Stochastic Programming Algorithms.* Master's thesis, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010.

[21] M. Krátká. *Tvorba obrázků pro matematické texty pomocí metapostu.* Masarykova univerzita v Brně, Přírodovědecká fakulta, Brno, 2001.

[22] P. Lidström. *Mechanical Vibrations.* Lecture Notes, Lunds Tekniska Högskola at Lund, Div. of Mechanics, 2010.

[23] E. Madenci, I. Guven. *The finite element method and applications in engineering using ANSYS.* Springer Science+Business Media, New York, 2006.

[24] S. Moaveni. *Finite element analysis, Theory and Application with ANSYS.* Prentice-Hall, New Jersey, 1999.

[25] J. M. Mulvey, H. Vladimirou. *Applying the progressive hedging algorithm to stochastic generalized networks.* Annals of Operations Research 31, pp. 399-424, 1991.

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

[27] P. Popela. *Stochastic Programming.* Lecture Notes, University of Malta, Department of Statistics and Operations Research, 2004.

[28] R. T. Rockafellar, R. J.-B. Wets. *Scenarios and policy aggregation in optimization under uncertainity.* Mathematics of Operations Research 16, pp. 119-147, 1991.

[29] R. E. Rosenthal, *GAMS - A User's Guide.* Gams Development Corporation, Washington DC, 2011. Available at `http://gams.com/dd/docs/bigdocs/GAMSUsersGuide.pdf`. [Online; cited December 25, 2011].

[30] A. Shapiro, D. Dentcheva, A. Ruszczyński. *Lectures on Stochastic Programming: Modelling and Theory.* Society for Industrial and Applied Mathematics and the Mathematical Programming Society, Philadelphia, 2009.

[31] I. M. Smith, D.V. Griffiths. *Programming the finite element method.* John Wiley & Sons, New York, second edition, 1988.

[32] M. C. Steinbach. *Tree-Sparse Convex Programs.* Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 2001.

[33] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application.* John Wiley & Sons, New York, 1986.

[34] O. C. Zienkiewicz, R. L. Taylor. *The Finite Element Method: Volume 1: The Basic.* Butterworth-Heinemann, Oxford, 2000.

[35] M. Zlamal. *On the Finite Element Method.* Numerische Mathematik, Volume 12, Number 5 , pages 394-409, 1968.

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

[37] E. Žampachová, P. Popela, M. Mrázek. *Optimum beam design via stochastic programming.* Paper in Kybernetika, volume 46, pages 575-586, 2010.

List of abbreviations

I would like to introduce abbreviations used in the thesis. Symbols and a notation are not listed here. Everything is explained when it is used.

LP	linear programming problem
MILP	mixed-integer linear programming problem
NLP	nonlinear programming problem
UP	underlying program
SP	stochastic programming problem
WS	wait-and-see
HN	here-and-now
IS	individual scenario
EV	expected value
EEV	expected result of using the expected value solution
EO	expected objective
VSS	value of stochastic solution
EVPI	expected value of perfect information
GAMS	general algebraic modeling system
GRG	generalized reduced gradient method
FDM	finite difference method
FEM	finite element method
PHA	progressive hedging algorithm
PDE	partial differential equation
ODE	ordinary differential equation
MATLAB	matrix laboratory

SI	international system of units
s.t.	such that, subject to
a.s.	almost surely
RG	reduced gradient method
KKT	Karush-Kean-Tucker
TPP	three-point pattern
PDF	portable document format

Appendix A

Optimality conditions

In this appendix, we will briefly describe the *Karush-Kuhn-Tucker (KKT) optimality conditions* for the problem with inequality constraints

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}), \\ \text{s.t.} \ & g_i(\mathbf{x}) \leq 0, \ i = 1, \ldots, m, \\ & \mathbf{x} \in X. \end{aligned} \tag{A.1}$$

Theorem A.1 (Karush-Kuhn-Tucker necessary optimality conditions)**.** *Consider the program (A.1). Assume that X is a nonempty open set in $\mathbb{R}^N$, f and $g_i : \mathbb{R}^N \to \mathbb{R}$ for all $i = 1, \ldots, m$ are functions, the set I is defined as $I = \{i \in \{1, \ldots, m\} : g_i(\bar{\mathbf{x}}) = 0\}$ and $\bar{\mathbf{x}}$ is a feasible point. Assume that the functions f and g_i for all $i \in I$ are differentiable at point $\bar{\mathbf{x}}$, the functions g_i for all $i \notin I$ are continuous at the point $\bar{\mathbf{x}}$. Furthermore, the gradients $\nabla g_i(\bar{\mathbf{x}})$ for all $i \in I$ are linearly independent. If the point $\bar{\mathbf{x}}$ is a local minimum to the problem (A.1), then there exist numbers μ_i for all $i \in I$ such that*

$$\begin{aligned} \nabla f(\bar{\mathbf{x}}) + \sum_{i \in I} \mu_i \nabla g_i(\bar{\mathbf{x}}) = \mathbf{0}, \\ \mu_i \geq 0, \ \forall i \in I. \end{aligned} \tag{A.2}$$

If g_i for all $i \notin I$ are also differentiable at point $\bar{\mathbf{x}}$, then (A.2) can be rewritten to the equivalent form

$$\begin{aligned} \nabla f(\bar{\mathbf{x}}) + \sum_{i \in I} \mu_i \nabla g_i(\bar{\mathbf{x}}) = \mathbf{0}, \\ \mu_i g_i(\bar{\mathbf{x}}) = 0, \ \forall i = 1, \ldots, m, \\ \mu_i \geq 0, \ \forall i = 1, \ldots, m. \end{aligned} \tag{A.3}$$

The scalars μ_i are called the Lagrange multipliers. A point $\bar{\mathbf{x}}$ is said to be a *Karush-Kuhn-Tucker (KKT) point* if there exist Lagrange multipliers $\mu_1, \ldots, \mu_m$ such that the point $\bar{\mathbf{x}}$ with them satisfies the KKT optimality conditions.

The proof of the previous theorem can be found in [1], these conditions can be also easily extended to programs with equality constraints.

Appendix B

Solver CONOPT

Nonlinear models created with GAMS must be solved with a nonlinear programing (NLP) algorithm. Currently, three solvers for NLP are available - CONOPT, MINOS and SNOPT. These solvers are based on different mathematical algorithms, and they behave differently for different models. GAMS cannot select the best algorithm automatically, we must select one as the default. CONOPT was chosen for our models. It is well suited for models with very nonlinear constraints and recursive equations, variables are solved and removed from the model. We have models where many equations have to be solved and CONOPT has been designed for large and sparse models what is our case.

CONOPT is a generalized reduced gradient algorithm (GRG) based solver specifically designed for large nonlinear programming problems. This solver was developed by A. Drud. The actual implementation has many modifications to make it efficient for large models and for models written in the GAMS language. Details of the algorithm can be found in [2]. Here we will give a basic description of the reduced gradient and generalized reduced gradient algorithms and also Newton-Raphson line search method used in the GRG method.

B.1 Reduced gradient method

In this section we want to introduce procedure for generating improving feasible directions. The method depends on reducing the dimensionality of the problem by representing all the variables in terms of an independent subset of the variables. The reduced gradient method (RG) was developed by P. Wolfe in 1963 to solve a nonlinear programming problem having linear constraints. The method was generalized by J. Abadie and J. Carpentier in 1969 to handle nonlinear constraints. Consider the following problem.

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}), \\ \text{s.t.} \ & \mathbb{A}\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where $\mathbb{A}$ is an $m \times n$ matrix of rank m, $\mathbf{b}$ is a vector with length m, $\mathbf{x}$ is a vector of unknown variables with length n and f is a continuously differentiable function on $\mathbb{R}^n$. We have to make some non-degeneracy assumptions. Any m columns of $\mathbb{A}$ are linearly independent, every extreme point of the feasible region has m strictly positive variables.

Now let $\mathbf{x}$ be a feasible solution. By the non-degeneracy assumptions, note that $\mathbb{A}$ can be decomposed into $(\mathbb{B}, \mathbb{N})$ and $\mathbf{x}^{\mathrm{T}}$ into $\left(\mathbf{x}_{\mathbb{B}}^{\mathrm{T}}, \mathbf{x}_{\mathbb{N}}^{\mathrm{T}}\right)$, where $\mathbb{B}$ is an $m \times m$ invertible

matrix and $\mathbf{x}_{\mathbb{B}} > 0$. Here $\mathbf{x}_{\mathbb{B}}$ is called the *basic vector*, and each of its components is strictly positive. The components of the *nonbasic vector* $\mathbf{x}_{\mathbb{N}}$ may be positive or zero. Let $\nabla f(\mathbf{x})^{\mathrm{T}} = \left(\nabla_{\mathbb{B}} f(\mathbf{x})^{\mathrm{T}}, \nabla_{\mathbb{N}} f(\mathbf{x})^{\mathrm{T}}\right)$, where $\nabla_{\mathbb{B}} f(\mathbf{x})$ is the gradient of f with respect to the basic vector $\mathbf{x}_{\mathbb{B}}$ and analogously $\nabla_{\mathbb{N}} f(\mathbf{x})$. A direction $\mathbf{d}$ is an improving feasible direction of f at $\mathbf{x}$ if $\nabla f(\mathbf{x})^{\mathrm{T}}\mathbf{d} < 0$, and if $\mathbb{A}\mathbf{d} = \mathbf{0}$ with $d_j \geq 0$ if $x_j = 0$. We now specify a direction vector $\mathbf{d}$ satisfying these properties, $\mathbf{d}^{\mathrm{T}}$ is decomposed into $\left(\mathbf{d}_{\mathbb{B}}^{\mathrm{T}}, \mathbf{d}_{\mathbb{N}}^{\mathrm{T}}\right)$. Note that $\mathbf{0} = \mathbb{A}\mathbf{d} = \mathbb{B}\mathbf{d}_{\mathbb{B}} + \mathbb{N}\mathbf{d}_{\mathbb{N}}$, then $\mathbf{d}_{\mathbb{B}} = -\mathbb{B}^{-1}\mathbb{N}\mathbf{d}_{\mathbb{N}}$. Let

$$\mathbf{r}^{\mathrm{T}} = (\mathbf{r}_{\mathbb{B}}^{\mathrm{T}}, \mathbf{r}_{\mathbb{N}}^{\mathrm{T}}) = \nabla f(\mathbf{x})^{\mathrm{T}} - \nabla_{\mathbb{B}} f(\mathbf{x})^{\mathrm{T}}\mathbb{B}^{-1}\mathbb{A} = \left(\mathbf{0}, \nabla_{\mathbb{N}} f(\mathbf{x})^{\mathrm{T}} - \nabla_{\mathbb{B}} f(\mathbf{x})^{\mathrm{T}}\mathbb{B}^{-1}\mathbb{N}\right)$$

be the *reduced gradient*, and let us examine the term $\nabla f(\mathbf{x})^{\mathrm{T}}\mathbf{d}$:

$$\nabla f(\mathbf{x})^{\mathrm{T}}\mathbf{d} = \nabla_{\mathbb{B}} f(\mathbf{x})^{\mathrm{T}}\mathbf{d}_{\mathbb{B}} + \nabla_{\mathbb{N}} f(\mathbf{x})^{\mathrm{T}}\mathbf{d}_{\mathbb{N}} = \left(\nabla_{\mathbb{N}} f(\mathbf{x})^{\mathrm{T}} - \nabla_{\mathbb{B}} f(\mathbf{x})^{\mathrm{T}}\mathbb{B}^{-1}\mathbb{N}\right)\mathbf{d}_{\mathbb{N}} = \mathbf{r}_{\mathbb{N}}^{\mathrm{T}}\mathbf{d}_{\mathbb{N}}.$$

We must choose $\mathbf{d}_{\mathbb{N}}$ that $\mathbf{r}_{\mathbb{N}}^{\mathrm{T}}\mathbf{d}_{\mathbb{N}} < 0$ and that $d_j \geq 0$ if $x_j = 0$.

The following rule is adopted. For each nonbasic component j, let $d_j = -r_j$ if $r_j \leq 0$, and let $d_j = -x_j r_j$ if $r_j > 0$. This ensures that $d_j \geq 0$ if $x_j = 0$, and prevents unduly small steps sizes when $x_j > 0$, but small, while $r_j > 0$. This also helps make the direction-finding map closed, thereby enabling convergence. Furthermore, $\nabla f(\mathbf{x})^{\mathrm{T}}\mathbf{d} \leq 0$, where the strict inequality holds if $\mathbf{d}_{\mathbb{N}} \neq 0$. We have described a procedure for constructing an improving feasible direction. This fact, as well as the fact that $\mathbf{d} = \mathbf{0}$ holds if and only if $\mathbf{x}$ is a KKT point defined in the Appendix A. This fact was proved in [2].

The main steps of algorithm of reduced gradient method are listed below.

0. Choose a starting point $\mathbf{x}_1$ satisfying $\mathbb{A}\mathbf{x}_1 = \mathbf{b}, \mathbf{x}_1 \geq 0$ and let $k = 1$ and go to Step 1.

1. Let $\mathbf{d}_k^{\mathrm{T}} = (\mathbf{d}_{\mathbb{B}}^{\mathrm{T}}, \mathbf{d}_{\mathbb{N}}^{\mathrm{T}})$ where $\mathbf{d}_{\mathbb{N}}$ and $\mathbf{d}_{\mathbb{B}}$ are obtained as below from (B.4) and (B.5), respectively. If $\mathbf{d}_k = 0$, stop; $\mathbf{x}_k$ is a KKT point. Otherwise, go to Step 2.

$$I_k = \text{ index set of the } m \text{ largest components of } \mathbf{x}_k, \tag{B.1}$$

$$\mathbb{B} = \{\mathbf{a}_j \mid j \in I_k\},\ \mathbb{N} = \{\mathbf{a}_j \mid j \notin I_k\}, \tag{B.2}$$

$$\mathbf{r}^{\mathrm{T}} = \nabla f(\mathbf{x}_k)^{\mathrm{T}} - \nabla \mathbb{B} f(\mathbf{x}_k)^{\mathrm{T}}\mathbb{B}^{-1}\mathbb{A}, \tag{B.3}$$

$$d_j = \begin{cases} -r_j & \text{if } j \notin I_k \text{ and } r_j \leq 0, \\ -x_j r_j & \text{if } j \notin I_k \text{ and } r_j > 0, \end{cases} \tag{B.4}$$

$$\mathbf{d}_{\mathbb{B}} = -\mathbb{B}^{-1}\mathbb{N}\mathbf{d}_{\mathbb{N}}. \tag{B.5}$$

2. Solve the following line search problem:

$$\begin{aligned} \min\ & f(\mathbf{x}_k + \lambda \mathbf{d}_k), \\ \text{s.t. } & 0 \leq \lambda \leq \lambda_{max}, \end{aligned}$$

where

$$\lambda_{max} = \begin{cases} \min_{l \leq j \leq n}\{\frac{-x_{jk}}{d_{jk}} \mid d_{jk} < 0\} & \text{if } \mathbf{d}_k \ngeq 0, \\ \infty & \text{if } \mathbf{d}_k \geq 0, \end{cases} \tag{B.6}$$

and x_{jk}, d_{jk} are the j-th components of $\mathbf{x}_k$ and $\mathbf{d}_k$, respectively. Let λ_k be an optimal solution, and let $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$. Replace k by $k + 1$ and go to Step 1.

B.2 Generalized reduced gradient algoritm

We can extend the reduced gradient method to handle nonlinear constraints. This extension is referred to as the *generalized reduced gradient method* (GRG), and is sketched below briefly.

Consider a nonlinear programming problem of the form

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}), \\ \text{s.t.} \ & \mathbf{h}(\mathbf{x}) = \mathbf{0}, \\ & \mathbf{x} \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ represents m equality constraints, $\mathbf{x} \in \mathbb{R}^n$, and suitable variable transformations have been used to represent all variables as being nonnegative. Here, any inequality constraint can be assumed to have been written as an equality by introducing a nonnegative slack variable.

Now, given a feasible solution $\mathbf{x}_k$, consider a linearization of $\mathbf{h}(\mathbf{x}) = \mathbf{0}$ given by

$$\mathbf{h}(\mathbf{x}_k) + \nabla\mathbf{h}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = \mathbf{0},$$

where $\nabla\mathbf{h}(\mathbf{x}_k)$ is the $m \times n$ Jacobian[1] of $\mathbf{h}$ evaluated at $\mathbf{x}_k$. Noting that $\mathbf{h}(\mathbf{x}_k) = \mathbf{0}$, the set of linear constraints given by $\nabla\mathbf{h}(\mathbf{x}_k)\mathbf{x} = \nabla\mathbf{h}(\mathbf{x}_k)\mathbf{x}_k$ is of the form $\mathbb{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{x}_k \geq 0$ is a feasible solution. Assuming that the Jacobian $\mathbb{A} = \nabla\mathbf{h}(\mathbf{x}_k)$ has full row rank, and partitioning it suitably into $[\mathbb{B}, \mathbb{N}]$ and, accordingly partitioning $\mathbf{x}^{\mathrm{T}} = (\mathbf{x}_{\mathbb{B}}^{\mathrm{T}}, \mathbf{x}_{\mathbb{N}}^{\mathrm{T}})$ (where hopefully, $\mathbf{x}_{\mathbb{B}} > \mathbf{0}$ in $\mathbf{x}_k$), we can compute the reduced gradient $\mathbf{r}$ via (B.3) and, hence, obtain the direction of motion $\mathbf{d}_k$ via (B.5) and (B.4). As before, we obtain $\mathbf{d}_k = \mathbf{0}$ if and only if $\mathbf{x}_k$ is a KKT point, hence the procedure terminates. Otherwise, a line search is performed along $\mathbf{d}_k$.

Earlier versions of this method adopted the following strategy. First, a line search is performed by determining λ_{max} via (B.6) and then finding λ_k as the solution to the line search problem to

$$\begin{aligned} \min \ & f(\mathbf{x}_k + \lambda\mathbf{d}_k), \\ \text{s.t.} \ & 0 \leq \lambda \leq \lambda_{max}. \end{aligned}$$

This gives $\mathbf{x}^\star = \mathbf{x}_k + \lambda_k\mathbf{d}_k$. Since $\mathbf{h}(\mathbf{x}^\star) = \mathbf{0}$ is not necessarily satisfied, we need a correction step. Toward this end, the Newton-Raphson method is then used to obtain $\mathbf{x}_{k+1}$ satisfying $\mathbf{h}(\mathbf{x}_{k+1}) = \mathbf{0}$, starting with the solution $\mathbf{x}^\star$ and keeping the components of $\mathbf{x}_{\mathbb{N}}$ fixed at the values $\mathbf{x}_{\mathbb{N}}^\star$. Hence, $\mathbf{x}_{\mathbb{N}}$ remains at $\mathbf{x}_{\mathbb{N}}^\star \geq \mathbf{0}$ during this iterative process, but some components of $\mathbf{x}_{\mathbb{B}}$ may tend to become negative. At such a point, a switch is made by replacing a negative basic variable x_r with a nonbasic variable x_q that is preferably positive and that has a significantly nonzero element in the corresponding row r of the column $\mathbb{B}^{-1}\mathbf{a}_q$. The Newton-Raphson process then continues as above with the revised basis (having now fixed x_r at zero) and the revised linearized system, until a nonnegative solution $\mathbf{x}_{k+1}$ satisfying $\mathbf{h}(\mathbf{x}_{k+1}) = \mathbf{0}$ is finally obtained.

More recent versions of the GRG method adopt a discrete sequence of positive step sizes and attempt to find a corresponding $\mathbf{x}_{k+1}$ for each such step size sequentially using

[1] The Jacobian matrix is the matrix of all first-order partial derivatives of vector function $\mathbf{h}$ with respect to the vector $\mathbf{x}_k$.

the foregoing Newton-Raphson scheme. Using the value $f(\mathbf{x}_{k+1})$ at each such point, when a three-point pattern (TPP) of the quadratic interpolation method is obtained, a quadratic fit is used to determine a new step size, for which the corresponding point $\mathbf{x}_{k+1}$ is again computed as above using the Newton-Raphson scheme. A feasible point having the smallest objective value thus found is used as the next iterate. This technique appears to yield a more reliable algorithm.

The iterative Newton-Raphson scheme complicates convergence arguments. The existing convergence proofs use restrictive and difficult to verify assumptions. Nonetheless, this type of algorithm provides quite a robust and efficient scheme for solving nonlinear programming problems.

The individual steps are of course much more detailed in a practical implementation like CONOPT. The optimizing steps are specialized in several versions according to the whether the model appears to be almost linear or not. For "almost" linear models some of the linear algebra work involving the Jacobian and $\mathbb{B}$ matrices can be avoided or done using cheap LP-type updating techniques and the steepest edge procedure can be useful. Similarly, when the model appears to be fairly nonlinear other aspects can be optimized, the set of basic variables will often remain constant over several iterations, and other parts of the sparse matrix algebra will take advantage of this.

B.3 Newton-Raphson line search method

Newton's method is method for minimizing a function of a single variable. The method of Newton is a procedure that deflects the steepest descent direction by premultiplying it by the inverse of the Hessian matrix (square matrix of second order partial derivatives of a function f). This operation is motivated by finding a suitable direction for the quadratic approximation to the function. To motivate the procedure, consider the following approximation q at a given point $\mathbf{x}_k$:

$$q(\mathbf{x}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^{\mathrm{T}}(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^{\mathrm{T}}\mathbb{H}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k),$$

where $\mathbb{H}(\mathbf{x}_k)$ is the Hessian matrix of f at $\mathbf{x}_k$. A necessary condition for a minimum of the quadratic approximation q is that $\nabla q(\mathbf{x}) = \mathbf{0}$, or $\nabla f(\mathbf{x}_k) + \mathbb{H}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) = \mathbf{0}$. Assuming that the inverse of $\mathbb{H}(\mathbf{x}_k)$ exists, the successor point $\mathbf{x}_{k+1}$ is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbb{H}^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k). \tag{B.7}$$

Equation (B.7) can be viewed as an application of the *Newton-Raphson method* to the solution of the system of equations $\nabla f(\mathbf{x}) = 0$. Given a well-determined system of nonlinear equations, each iteration of the Newton-Raphson method adopts a first-order Taylor series approximation to this equation system at the current iterate and solves the resulting linear system to determine the next iterate. Applying this to the system $\nabla f(\mathbf{x}) = 0$ at an iterate $\mathbf{x}_k$, the first-order approximation to $\nabla f(\mathbf{x})$ is given by $\nabla f(\mathbf{x}_k) + \mathbb{H}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$. Setting this equal to zero and solving produces the solution $\mathbf{x} = \mathbf{x}_{k+1}$ as given by (B.7).

Appendix C

Penalty functions

In this appendix we deal with the approaches to convert nonlinear programming problems with equality and/or inequality constrains into an equivalent unconstrained problem or problems with simple constraints.

There are two alternative approaches achieving this described detailed in [1]. The first is called the penalty, or the exterior penalty function method, in which we add a penalty term to the objective function for points not lying in the feasible set and thus violating some of the constraints. This method generated a sequence of infeasible points whose limit is an optimal solution to the original problem. The second method is the barrier or interior penalty function method, in which a barrier penalty term that prevents the points generated from leaving the feasible region is added to the objective function. This method generates a sequence of feasible interior points whose limit is an optimal solution to the original constrained problem.

Clearly we would like to transfer some properties of original constrained problems, such as convexity, smoothness, etc. to penalized problems as well. We can achieve this by carefully choosing penalty functions.

Further only the basic concept of penalty functions is introduced. The basic idea behind all penalty algorithms is to replace constrained problem with the equivalent unconstrained one or with a sequence of unconstrained problem. The constraints are placed into the objective function via a penalty parameter in a way that penalizes any violation of the constraints.

C.1 Exterior penalty function method

Consider the following problem with single constraint:

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}), \\ \text{s.t.} \ & h(\mathbf{x}) = 0. \end{aligned}$$

This problem is replaced by the unconstrained problem, where the penalty parameter $\mu > 0$ is an appropriate large number:

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}) + \mu h^2(\mathbf{x}), \\ \text{s.t.} \ & \mathbf{x} \in \mathbb{R}^n. \end{aligned}$$

We can see that an optimal solution to the above problem must have $h^2(\mathbf{x})$ close to zero, otherwise a large penalty term $\mu h^2(\mathbf{x})$ will occur.

Now consider problem with single inequality constraint:

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}), \\ \text{s.t.} \ & g(\mathbf{x}) \leq 0. \end{aligned}$$

The previous approach is not appropriate, because a penalty will occur whether $g(\mathbf{x}) < 0$ or $g(\mathbf{x}) > 0$. But a penalty is desired only it the point $\mathbf{x}$ is not feasible, that is, if $g(\mathbf{x}) > 0$. A suitable unconstrained problem is given by:

$$\begin{aligned} \min_{\mathbf{x}} \ & f(\mathbf{x}) + \mu \max\{0, g(\mathbf{x})\}, \\ \text{s.t.} \ & \mathbf{x} \in \mathbb{R}^n. \end{aligned}$$

If $g(\mathbf{x}) \leq 0$, then $\max\{0, g(\mathbf{x})\} = 0$ and no penalty occurs and if $g(\mathbf{x}) > 0$, then $\max\{0, g(\mathbf{x})\} > 0$ and the penalty term $\mu g(\mathbf{x})$ is realized. If differentiability is desirable, we can consider instead a penalty function term of the type $\mu \left(\max\{0, g(\mathbf{x})\}\right)^2$.

In general, a penalty function must incur a positive penalty for infeasible points and no penalty for feasible points. If we consider inequality constraints of the form $g_i(\mathbf{x}) \leq 0$ for $i = 1, \ldots, m$ and equality constraints of the form $h_i(\mathbf{x})$ for $i = 1, \ldots, l$, a suitable penalty function α is defined by

$$\alpha(\mathbf{x}) = \sum_{i=1}^{m} \phi\left(g_i(\mathbf{x})\right) + \sum_{i=1}^{l} \psi\left(h_i(\mathbf{x})\right),$$

where ϕ and ψ are continuous functions satisfying the following:

$$\phi(y) \begin{cases} = 0 & \text{if } y \leq 0 \\ > 0 & \text{if } y > 0 \end{cases}, \ \psi(y) \begin{cases} = 0 & \text{if } y = 0 \\ > 0 & \text{if } y \neq 0. \end{cases}$$

Typically, ϕ and ψ are of the forms

$$\begin{aligned} \phi(y) &= \left(\max\{0, y\}\right)^p, \\ \psi(y) &= |y|^p, \end{aligned}$$

where p is a positive integer. Then the penalty function α is of the form

$$\alpha(\mathbf{x}) = \sum_{i=1}^{m} \left(\max\{0, g_i(\mathbf{x})\}\right)^p + \sum_{i=1}^{l} |h_i(\mathbf{x})|^p.$$

The function $f(\mathbf{x}) + \mu\alpha(\mathbf{x})$ is the auxiliary function.

Theorem C.1 (global convergence of a penalty method)**.** *Assume that the original constrained problem possesses optimal solutions. Then, every limit point of the sequence* $\{\mathbf{x}_\mu\}$*,* $\mu \to \infty$ *of globally optimal solutions to equivalent unconstrained problem is globally optimal in the original constrained problem.*

Proof of previous theorem can be found in [1] and more detailed information can be found in [2].

C.2 Interior penalty function method

The idea behind exterior penalty functions is to approximate a feasible set on the whole $\mathbb{R}^n$. The interior penalty function methods construct approximations only inside the feasible set and set a barrier against leaving it. The method generates a sequence of interior points that converges to it.

We consider inequality constraints of the form $g_i(\mathbf{x})$ for $i = 1, \ldots, m$. For the method to work, we need to assume that there exists a strictly feasible point $\hat{\mathbf{x}} \in \mathbb{R}^n$, such that $g_i(\hat{\mathbf{x}}) < 0$, $i = 1, \ldots, m$. So in contrast with the exterior method, we cannot include equality constraints into the penalty term. Of course, it is possible to extend the discussion to equality constraints, but we prefer simple notation.

Suitable penalty function α is defined by

$$\alpha(\mathbf{x}) = \sum_{i=1}^{m} \phi\left(g_i(\mathbf{x})\right),$$

where ϕ is continuous function satisfying following:

$$\phi(y) \begin{cases} > 0 & \text{if } y < 0, \\ = \infty & \text{otherwise.} \end{cases}$$

Typical example of the function ϕ is $\phi_1(y) = -y^{-1}$ or $\phi_2(y) = -\log\left(\min\{1, -y\}\right)$.

Similarly to the exterior penalty functions, the theorem about the convergence to globally optimal solutions can be formulated with some additional assumptions. More detailed information could be found again in [1] and especially in [2].

C.3 Computational difficulties

As the penalty parameter increases in the exterior penalty methods or decreases in the interior penalty methods, the approximating problem becomes more and more ill-conditioned. Therefore, a typical computational strategy is to start from "safe" values of the penalty parameter (relatively small for exterior penalties, or large for interior penalties), and then proceed step after step slightly modifying the penalty parameter heuristically (for example by multiplying it with some number close to 1).

We have to note here that there is no general rule how to determine the value of penalty parameter to obtain optimal solution in the shortest time. The same problem was mentioned in the PHA description.

We are interested in penalty functions that can reach the optimal solution to the original problem for finite value of the penalty parameter. The *augmented Lagrangian penalty function* satisfies this property and preserves differentiability of the objective function. We take the shifted quadratic penalty function and expend it. Then we have the penalty function which is composed of a linear and a quadratic term. Again the equality constraints are treated up in other way than the inequality constraints. Here we gave you only basic insight, for more detailed information see [5].

Appendix D

Sample of GAMS source code

```
$title Spatial decomposition for IS reformulated beam

Scalars   N            number of elements on part      /6/
          l            length of part of beam [mm]     /750/
          bb           load constant [Nmm-1]           /10/
          ro           steel density [tmm-3]           /7.85E-9/
          alfa         rigidity weight coefficient     /0.5/
          beta         weight weight coefficient       /0.5/
          sigma        stress limitation [MPa]         /100/
          rigidity     normalization constant          /1.75E12/
          weight       normalization constant          /0.007/
          E            Young's modulus [MPa]           /210E3/;

Sets      i            node index                      /1*14/
          el           element index                   /1*6/
          j            index in element                /1*4/;

Parameter d            spatial step;
          d = l/N;

Parameter hel(el,i)    extern forces and moments on one element;
          loop(el,
               loop(i,
                    hel(el,i)$(ord(i) eq (2*ord(el)-1))=bb*d/2;
                    hel(el,i)$(ord(i) eq (2*ord(el)))=bb*d**2/12;
                    hel(el,i)$(ord(i) eq (2*ord(el)+1))=bb*d/2;
                    hel(el,i)$(ord(i) eq (2*ord(el)+2))=-bb*d**2/12;
               );
           );

Parameter h(i) extern forces and moments for whole beam;
          h(i) = sum(el, hel(el,i));

Alias(i,ii);

Parameter bcvec1(i)    vector with boundary conditions - part 1;
          bcvec1(i)=0;
          bcvec1(i)$(ord(i) eq (card(i)-1))=0.261;
          bcvec1(i)$(ord(i) eq card(i))=-0.001;

Parameter bcvec2(i)    vector with boundary conditions - part 2;
          bcvec2(i)=0;
          bcvec2('1')=0.261;
          bcvec2('2')=0.001;
```

```
Parameter Kel(j,j)      symmetric element stiffness matrix;
          Kel('1','1')=12/(d**3);     Kel('1','3')=-Kel('1','1');
          Kel('3','1')=-Kel('1','1');Kel('3','3')=Kel('1','1');
          Kel('1','2')=6/(d**2);      Kel('1','4')=Kel('1','2');
          Kel('2','1')=Kel('1','2'); Kel('2','3')=-Kel('1','2');
          Kel('3','2')=-Kel('1','2');Kel('3','4')=-Kel('1','2');
          Kel('4','1')=Kel('1','2'); Kel('4','3')=-Kel('1','2');
          Kel('2','2')=4/d;           Kel('4','4')=Kel('2','2');
          Kel('2','4')=2/d;           Kel('4','2')=Kel('2','4');

Parameter Lel(el,j,i)     localization operator;
          loop(j,loop(i,
                   Lel(el,'1',i)$(ord(i) eq (2*ord(el)-1))=1;
                   Lel(el,'2',i)$(ord(i) eq (2*ord(el)))=1;
                   Lel(el,'3',i)$(ord(i) eq (2*ord(el)+1))=1;
                   Lel(el,'4',i)$(ord(i) eq (2*ord(el)+2))=1););

Parameter LelTrans(el,i,j)   transposed localization operator;
          loop(el,loop(i,loop(j,LelTrans(el,i,j)=Lel(el,j,i));));

Alias(j,jj);

Parameter KL(el,j,i)          Kel*Lel for all elements;
          loop(el,loop(j,loop(i,KL(el,j,i)=
          sum(jj,Kel(j,jj)*Lel(el,jj,i));)););

Parameter LKL(el,i,i)         LelTrans*KL for all elements;
          loop(el,loop(ii,loop(i,LKL(el,i,ii)=
          sum(j,LelTrans(el,i,j)*KL(el,j,ii));)););

Parameter K(i,i)              stiffness matrix for the beam;
          K(i,i)=0; loop(el,K(i,ii)=K(i,ii)+LKL(el,i,ii));

Variables z1        variable for objective function on part 1
          z2        variable for objective function on part 2
          v1(i)     deformation of part 1 (displacements and rotations)
          v2(i)     deformation of part 2 (displacements and rotations);

Positive variables
          a            dimension of cross section
          b            dimension of cross section;

Equations obj1                  objective function on part 1
          obj2                  objective function on part 2
          BCL11                 left boundary condition for defl. - part 1
          BCL12                 left boundary condition for rot.  - part 1
          BCR11(i)              right boundary condition for defl.- part 1
          BCR12(i)              right boundary condition for rot. - part 1
          BCL21                 left boundary condition for defl. - part 2
          BCL22                 left boundary condition for rot.  - part 2
          BCR21(i)              right boundary condition for defl.- part 2
          BCR22(i)              right boundary condition for rot. - part 2
          FEM1(i)               constraint with 4th derivative    - part 1
          FEM2(i)               constraint with 4th derivative    - part 2
          MaxStress10           maximal stress in the first node  - part 1
          MaxStress1d(el,i)     maximal stress in the last node   - part 1
          MaxStress1(el,i)      maximal stress in i-th node       - part 1
          MinStress10           minimal stress in the first node  - part 1
          MinStress1d(el,i)     minimal stress in the last node   - part 1
          MinStress1(el,i)      minimal stress in i-th node       - part 1
          MaxStress20           maximal stress in the first node  - part 2
```

```
          MaxStress2d(el,i)     maximal stress in the last node   - part 2
          MaxStress2(el,i)      maximal stress in i-th node       - part 2
          MinStress20           minimal stress in the first node  - part 2
          MinStress2d(el,i)     minimal stress in the last node   - part 2
          MinStress2(el,i)      minimal stress in i-th node       - part 2;

*--------------IS deterministic reformulation part 1 ---------------------*
obj1.. z1 =e= -alfa*E*a*b**3/(12*rigidity)+beta*ro*a*b*l/weight;
BCL11.. v1('1')=e=0;
BCL12.. v1('2')=e=0;
BCR11(i)$(ord(i) eq (card(i)-1)).. v1(i)=e=bcvec1(i);
BCR12(i)$(ord(i) eq card(i)).. v1(i)=e=bcvec1(i);
FEM1(i)$((ord(i) ne 1)and(ord(i) ne 2)and(ord(i) ne (card(i)-1))and
(ord(i) ne card(i))).. ((E*a*(b**3))/(12))*sum(ii$((ord(ii) ne 1)and
(ord(ii) ne 2)and(ord(ii) ne (card(ii)-1))and(ord(ii) ne card(ii))),
K(i,ii)*v1(ii))=e=h(i)-((E*a*(b**3))/(12))*sum(ii,K(i,ii)*bcvec1(ii));
MaxStress10..E*(b/2)*((-6/d**2)*v1('1')+(-4/d)*v1('2')+(6/d**2)*v1('3')+
(-2/d)*v1('4'))=l=sigma;
MaxStress1d(el,i)$((ord(el)eq card(el))and(ord(i)eq(2*ord(el)))..E*(b/2)*
((6/d**2)*v1(i-1)+(2/d)*v1(i)+(-6/d**2)*v1(i+1)+(4/d)*v1(i+2))=l=sigma;
MaxStress1(el,i)$((ord(el)ne card(el))and(ord(i)eq(2*ord(el)))..
E*(b/4)*((6/d**2)*v1(i-1)+(2/d)*v1(i)+(-6/d**2)*v1(i+1)+(4/d)*v1(i+2)+
(-6/d**2)*v1(i+1)+(-4/d)*v1(i+2)+(6/d**2)*v1(i+3)+(-2/d)*v1(i+4))=l=sigma;
MinStress10..-E*(b/2)*((-6/d**2)*v1('1')+(-4/d)*v1('2')+(6/d**2)*v1('3')+
(-2/d)*v1('4'))=l=sigma;
MinStress1d(el,i)$((ord(el)eq card(el))and(ord(i)eq(2*ord(el)))..-E*(b/2)*
((6/d**2)*v1(i-1)+(2/d)*v1(i)+(-6/d**2)*v1(i+1)+(4/d)*v1(i+2))=l=sigma;
MinStress1(el,i)$((ord(el)ne card(el))and(ord(i)eq(2*ord(el)))..
-E*(b/4)*((6/d**2)*v1(i-1)+(2/d)*v1(i)+(-6/d**2)*v1(i+1)+(4/d)*v1(i+2)+
(-6/d**2)*v1(i+1)+(-4/d)*v1(i+2)+(6/d**2)*v1(i+3)+(-2/d)*v1(i+4))=l=sigma;
*--------------IS deterministic reformulation part 2 ---------------------*
obj2.. z2 =e= -alfa*E*a*b**3/(12*rigidity)+beta*ro*a*b*l/weight;
BCL21.. v2('1')=e=bcvec2('1');
BCL22.. v2('2')=e=bcvec2('2');
BCR21(i)$(ord(i) eq (card(i)-1)).. v2(i)=e=0;
BCR22(i)$(ord(i) eq card(i)).. v2(i)=e=0;
FEM2(i)$((ord(i) ne 1)and(ord(i) ne 2)and(ord(i) ne (card(i)-1))and
(ord(i) ne card(i))).. ((E*a*(b**3))/(12))*sum(ii$((ord(ii) ne 1)and
(ord(ii) ne 2)and(ord(ii) ne (card(ii)-1))and(ord(ii) ne card(ii))),
K(i,ii)*v2(ii))=e=h(i)-((E*a*(b**3))/(12))*sum(ii,K(i,ii)*bcvec2(ii));
MaxStress20..E*(b/2)*((-6/d**2)*v2('1')+(-4/d)*v2('2')+(6/d**2)*v2('3')+
(-2/d)*v2('4'))=l=sigma;
MaxStress2d(el,i)$((ord(el)eq card(el))and(ord(i)eq(2*ord(el)))..E*(b/2)*
((6/d**2)*v2(i-1)+(2/d)*v2(i)+(-6/d**2)*v2(i+1)+(4/d)*v2(i+2))=l=sigma;
MaxStress2(el,i)$((ord(el)ne card(el))and(ord(i)eq(2*ord(el)))..
E*(b/4)*((6/d**2)*v2(i-1)+(2/d)*v2(i)+(-6/d**2)*v2(i+1)+(4/d)*v2(i+2)+
(-6/d**2)*v2(i+1)+(-4/d)*v2(i+2)+(6/d**2)*v2(i+3)+(-2/d)*v2(i+4))=l=sigma;
MinStress20..-E*(b/2)*((-6/d**2)*v2('1')+(-4/d)*v2('2')+(6/d**2)*v2('3')+
(-2/d)*v2('4'))=l=sigma;
MinStress2d(el,i)$((ord(el)eq card(el))and(ord(i)eq(2*ord(el)))..-E*(b/2)*
((6/d**2)*v2(i-1)+(2/d)*v2(i)+(-6/d**2)*v2(i+1)+(4/d)*v2(i+2))=l=sigma;
MinStress2(el,i)$((ord(el)ne card(el))and(ord(i)eq(2*ord(el)))..
-E*(b/4)*((6/d**2)*v2(i-1)+(2/d)*v2(i)+(-6/d**2)*v2(i+1)+(4/d)*v2(i+2)+
(-6/d**2)*v2(i+1)+(-4/d)*v2(i+2)+(6/d**2)*v2(i+3)+(-2/d)*v2(i+4))=l=sigma;
*------------------0-iteration without penalty terms part 1---------------*
model Determ1    /obj1, BCL11, BCL12, BCR11, BCR12, FEM1, MaxStress10,
                  MaxStress1d, MaxStress1, MinStress10, MinStress1d,
                  MinStress1/;
a.lo=10;
a.up=100;
b.lo=10;
```

```
b.up=100;
a.l=100;
b.l=100;
v1.l(i)=0;
solve Determ1 using nlp minimizing z1;
Display a.l, b.l, z1.l, v1.l;
*------------------0-iteration without penalty terms part 2---------------*
model Determ2    /obj2, BCL21, BCL22, BCR21, BCR22, FEM2, MaxStress20,
                 MaxStress2d, MaxStress2, MinStress20, MinStress2d,
                 MinStress2/;
a.lo=10;
a.up=100;
b.lo=10;
b.up=100;
a.l=100;
b.l=100;
v2.l(i)=0;
solve Determ2 using nlp minimizing z2;
Display a.l, b.l, z2.l, v2.l;
*-------------------------PHA:1-p iterations-----------------------------*
Scalars     rho          penalization coeffitcient            / 40.00 /,
            epsilon      toleration for stop algorithm        / 0.0001 /,
            q            coefficient for rotations            / 1000 /;

Parameter   delta        stop condition;
            delta = 1;

Sets        p            maximal iteration index              /0*1000/ ;


Parameter eo(i)       even or odd parameter (diferrentitate v and theta);
            loop(i,if(((0.5*ord(i)) eq (round(0.5*ord(i)))),eo(i)=0;
                 else eo(i)=1;));

Parameter   vbar1(i) average solution - part 1;
            vbar1(i) = 0;

Parameter   vbar2(i) average solution - part 2;
            vbar2(i) = 0;

Parameter   vbar1bf(i) average solution in previous iteration - part 1;

Parameter   vbar2bf(i) average solution in previous iteration - part 2;

Parameter w1(i)       linear penalization parameter - part 1;
          w1(i)=0;

Parameter w2(i)       linear penalization parameter - part 2;
          w2(i)=0;

Variables zPHA1     variable for PHA objective function - part 1
          zPHA2     variable for PHA objective function - part 2;

Equations objPHA1   objective PHA function with penalty terms - part 1
          objPHA2   objective PHA function with penalty terms - part 2;

objPHA1.. zPHA1 =e= -alfa*E*a*b**3/(12*rigidity)+beta*ro*a*b*l/weight +
sum(i,(eo(i)*w1(i)*v1(i))+((1-eo(i))*w1(i)*v1(i))) + 0.5*rho*
sum(i,eo(i)*(v1(i)-vbar1(i))*(v1(i)-vbar1(i))+q*(1-eo(i))*(v1(i)-
vbar1(i))*(v1(i)-vbar1(i)));
objPHA2.. zPHA2 =e= -alfa*E*a*b**3/(12*rigidity)+beta*ro*a*b*l/weight +
```

```
sum(i,(eo(i)*w2(i)*v2(i))+((1-eo(i))*w2(i)*v2(i))) + 0.5*rho*
sum(i,eo(i)*(v2(i)-vbar2(i))*(v2(i)-vbar2(i))+q*(1-eo(i))*(v2(i)-
vbar2(i))*(v2(i)-vbar2(i)));
*--------------------------PHA-part1-------------------------------------*
model PHA1      /objPHA1, BCL11, BCL12, FEM1, MaxStress10,
                 MaxStress1d, MaxStress1, MinStress10, MinStress1d,
                 MinStress1/;
a.lo=10;
a.up=100;
b.lo=10;
b.up=100;
a.l=100;
b.l=100;
v1.l(i)=0;
*--------------------------PHA-part2-------------------------------------*
model PHA2      /objPHA2, BCR21, BCR22, FEM2, MaxStress20,
                 MaxStress2d, MaxStress2, MinStress20, MinStress2d,
                 MinStress2/;
a.lo=10;
a.up=100;
b.lo=10;
b.up=100;
a.l=100;
b.l=100;
v2.l(i)=0;
*-----------------------------------------------------------------------*
vbar1('1')=v1.l('1');
vbar1('2')=v1.l('2');
vbar1('3')=v1.l('3');
vbar1('4')=v1.l('4');
vbar1(i)$((ord(i) ne 1)and(ord(i) ne 2)and(ord(i) ne 3)and(ord(i) ne 4))=
0.5*(v1.l(i)+v2.l(i-4));

vbar2(i)$(ord(i) eq card(i))=v2.l(i);
vbar2(i)$(ord(i) eq (card(i)-1))=v2.l(i);
vbar2(i)$(ord(i) eq (card(i)-2))=v2.l(i);
vbar2(i)$(ord(i) eq (card(i)-3))=v2.l(i);
vbar2(i)$((ord(i) ne card(i))and(ord(i) ne (card(i)-1))and
(ord(i) ne (card(i)-2))and(ord(i) ne (card(i)-3)))=0.5*(v1.l(i+4)+v2.l(i));

loop( p, if((delta gt epsilon),
solve PHA1 using nlp minimizing zPHA1;
solve PHA2 using nlp minimizing zPHA2;

vbar1bf(i)=vbar1(i);
vbar1('1')=v1.l('1');
vbar1('2')=v1.l('2');
vbar1('3')=v1.l('3');
vbar1('4')=v1.l('4');
vbar1(i)$((ord(i) ne 1)and(ord(i) ne 2)and(ord(i) ne 3)and(ord(i) ne 4))=
0.5*(v1.l(i)+v2.l(i-4));

vbar2bf(i)=vbar2(i);
vbar2(i)$(ord(i) eq card(i))=v2.l(i);
vbar2(i)$(ord(i) eq (card(i)-1))=v2.l(i);
vbar2(i)$(ord(i) eq (card(i)-2))=v2.l(i);
vbar2(i)$(ord(i) eq (card(i)-3))=v2.l(i);
vbar2(i)$((ord(i) ne card(i))and(ord(i) ne (card(i)-1))and
(ord(i) ne (card(i)-2))and(ord(i) ne (card(i)-3)))=0.5*(v1.l(i+4)+v2.l(i));

delta = (sum(i,(vbar1bf(i)-vbar1(i))*(vbar1bf(i)-vbar1(i)))+sum(i,
```

```
(vbar2bf(i)-vbar2(i))*(vbar2bf(i)-vbar2(i)))+0.5*(sum(i,(v1.l(i)-
vbar1(i))*(v1.l(i)-vbar1(i)))+sum(i,(v2.l(i)-vbar2(i))*(v2.l(i)-
vbar2(i)))))**0.5;

w1(i)=w1(i)+rho*eo(i)*(v1.l(i)-vbar1(i))+rho*q*(1-eo(i))*(v1.l(i)-vbar1(i))
w2(i)=w2(i)+rho*eo(i)*(v2.l(i)-vbar2(i))+rho*q*(1-eo(i))*(v2.l(i)-vbar2(i))

bcvec1(i)$(ord(i) eq (card(i)-1))=v1.l(i);
bcvec1(i)$(ord(i) eq card(i))=v1.l(i);
bcvec2('1')=v2.l('1');
bcvec2('2')=v2.l('2');););
*-------------------------------------------------------------------------*
```

Appendix E

What is on the CD

The CD attached to the thesis contains

- the thesis in PDF format: `thesis.pdf`
- the implementation of the news vendor problem (Section 3.7): `newsvendor.gms`
- the implementation of the IS reformulation with FEM approximations of the beam problem (Section 6.4): `ISFEM.gms`
- the implementation of the spatial decomposition for the IS reformulation of the beam problem (Section 6.5): `ISdecomp.gms`
- the implementation of the EO reformulation with FEM approximations of the beam problem (Section 6.6): `EOFEM.gms`
- the implementation of the spatial decomposition for the EO reformulation of the beam problem (Section 6.7): `EOdecomp.gms`