NUMERICAL METHODS FOR PROBABILISTIC CONSTRAINED OPTIMIZATION PROBLEM WHERE RANDOM VARIABLES HAVE DEGENERATE CONTINUOUS DISTRIBUTION.

BY OLGA MYNDYUK

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Operations Research

Written under the direction of

Dr. András Prékopa

and approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

May, 2016

ABSTRACT OF THE DISSERTATION

Numerical Methods for Probabilistic Constrained Optimization Problem where Random Variables have Degenerate Continuous Distribution.

by Olga Myndyuk

Dissertation Director: Dr. András Prékopa

Several probabilistic constrained problems (single commodity stochastic network design problem and water reservoir problem) are formulated and solved by use of different numerical methods. The distribution considered are degenerate normal and uniform distributions. The network design problem is to find optimal node and arc capacities under some deterministic and probabilistic constraints that ensure the satisfiability of all demands on a given probability level. The large number of feasibility inequalities is reduced to a much smaller number of them and an equivalent reformulation takes us to a specially structured semi-infinite LP. This, in turn, is solved by a combination of inner and outer algorithms providing us with both lower and upper bounds for the optimum at each iteration. The flood control and serially linked reservoir network design with consecutive k-out-of-n type reliability problems are formulated, simplified and solved. Alternative, derivative-free methods, are proposed and implemented. Various numerical examples are presented and solution methods software library is developed.

Acknowledgements

It is the supreme art of the teacher to awaken joy in creative expression and knowledge. Albert Einstein

First of all, I would like to express my gratitude to my research adviser, Professor András Prékopa, who has been much more than an academic mentor to me. His overall wit, knowledge and erudition in natural and social sciences, history, geography, engineering, culture can hardly be found in one person, and I was truly blessed to work and converse with him. This experience had a very deepening effect on my personality, and I am very grateful for that.

I also want to sincerely thank Dr. Endre Boros, Dr. Vladimir Gurvich, Dr. Melike Baykal-Gürsoy, Dr. Adi Ben-Israel and Dr. Merve Unuvar who serve as my thesis committee, for their assistance and valuable comments.

In general, I am very glad to acknowledge that I was lucky to have known and have learned from the professors at RUTCOR. Not only has it been an exceptional gathering of minds, but also a very cozy and homey environment for all the students which is very hard to find these days.

Special thanks is to my dear friends who supported me all the way, from whom I learned a great deal: Anton Molyboha, Mariya Naumova, Svetlana Solovieva, Anh Ninh, Marina Goldovsky.

And finally, I am very happy to have my closest support circle: my loving and beloved husband and best friend Valery, my dear parents, my sister Oksana and her wonderful family.

Table of Contents

Chapter 1

Introduction

In this work probabilistic constrained optimization problems with degenerate distributed random variables are in the center of interest. Examples of such problems include single-commodity, some other multi-commodity networks, as well as flood control and irrigation.

The optimal operation of water reservoir networks is often modeled in connection with dynamic stochastic programming problems. Gal [14] (1979), Yakowitz [72] (1982), Archibadl et al. [1] (1997) are just a few of the many publications where the solution to optimal water reservoir system is given by the techniques from the dynamic programming. Reliability is always an important issue in engineering design. The irrigation problem we solved in this work was briefly presented in Prékopa, Rapcsák, Szuffa [39] (2010), where the probabilistic constraint prescribes lower bound for a consecutive k -out of-n probability rather than a single joint probability. We found optimal reservoir capacities such that at least k consecutive periods, the demands are met with a probability which is at least as prescribed. The objective function we minimized is the sum of total building costs of each reservoir per its capacity, therefore is also finding the optimal water reservoir capacity. We assumed that the inflow and demand values are normally distributed.

Most of the network problems have random elements and are supposed to satisfy some reliability requirement. In our networks the demands at the nodes and the capacity deficiencies at the nodes and arcs are random and reliability means that all demands can be satisfied at a prescribed probability level that is near 1, in practice. Stochastic programming models can be subdivided into static and dynamic models. The static models can also be subdivided into three further categories: probabilistic constrained, penalty or recourse and hybrid models. The dynamic type models are the two-stage and multi-stage models. In all of the models constructions we have underlying deterministic models that would be

the models in the absence of random variables. Also, in all the models there is decision-observation scheme or sequence, alternating decisions on some of the decision variables and observations of some of the random variables. In the static case this scheme is simply the following: decision on x, observation of the random variable ξ.

Even though our stochastic network design model construction is relatively simple, because it is single-commodity and static, still it is widely applicable and very involved from the point of view of the methodology handling the probabilistic constraint.

Programming under probabilistic constraint was first formulated by [5] Charnes, Cooper, Symonds (1958), where, however, probability levels were prescribed individually on each stochastic constraint, neglecting the stochastic dependence among the random variables. Miller and Wagner formulated a problem with joint probabilistic constraint with independent random variables and Prékopa [34] (1970, 1973) formulated the theoretically correct model, and explored its mathematical properties. For further results in this respect see [46] Prékopa (1995). In the stochastic network design problem we studied here, it is very important to use joint probabilistic constraint in connection with the stochastic inequalities involved, because the feasibility inequalities jointly provide us with necessary and sufficient condition for the existence of a feasible flow, individually they do not have reasonable practical interpretation. Even though many papers have been published on stochastic network design problem formulation and solution, only a relatively small number of them are in some sense antecedents of this one. In Prékopa [38] (1980) a two stage programming under uncertainty was formulated, and, for the first time in the literature, the feasibility region in cooperating power systems was characterized by a system of linear inequalities. A probabilistic constraint was formulated for the solvability of the second stage problem but methodology available at that time made it possible to solve very small problems. In order to efficiently solve probabilistic constrained stochastic programming problems Prékopa [44] (1990) introduced the concept of p-level efficient point, for the case of a discrete random variable. Algorithm, developed later on, based on this concept, made it possible to efficiently solve stochastic networks design problems with discrete random variables, [52] by Pékopa and Unuvar (2015). Efficient pre-processing methods, to identify essential feasibility inequalities have been presented in [45] Prékopa, Boros (1991) and [70] Wallace, Wets (1993).

We also mention the recent papers of [64] Thapalia, Crainic, Kant, Wallace (2010), where the reader can find useful ideas in connection with single commodity stochastic network design, even though their main interest is other from ours.

The organization of the thesis is the following. The next chapter defines the stochastic optimization problem with joint probabilistic constraints, and discusses the conditions under which such problem is convex, as well as discusses the mathematical properties of the probabilistic constraint. Chapter 3 introduces a constraint elimination procedure which can be applied in this case, Chapter 4 considers the particular case of such problem, namely a supply-demand network design and describes the numerical algorithms for stochastic network design problems. Chapter 5 presents numerical examples and compares the numerical algorithms against each other. Chapter 6 considers several derivative-free algorithms, with respective numerical examples. Chapter 7 concludes the thesis.

Chapter 2

The Stochastic Optimization Problem with Probabilistic Constraints

The stochastic optimization problem with (joint) probabilistic constraints has the form

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \leq b \\ &P\{Tx \geq \eta\} \geq p \end{aligned} \tag{2.1}$$

where c, b are given vectors, A, T are given matrices, $0 < p < 1$ is the probability level, typically close to 1, and η is a random vector with a given distribution. This thesis is particularly concerned with the case when the distribution of η is degenerate, that is its support is an affine subspace of R^m. The importance of this case is best illustrated with an example: the network design problem discussed in the following section.

2.1 Network Design Problem

We assume that at the nodes there are demands which are random variables and capacities which are decision variables. There are also capacities on the arcs which are decision variables. At each node, the random demand can be less than or equal to the local node (generating) capacity or can surpass it, making the node of demand or supply (negative demand) type. This implies that we cannot apriori classify the nodes (demand or supply). Fortunately, we can avoid it if we use the definitions in [15] Gale (1957) and [12] Ford, Fulkerson (1962). These are presented below.

Definition 2.1 *A network $[N, y]$ is a pair of a finite set of nodes N and a capacity function defined on arcs $y(i,j), (i,j) \in N \times N$, assumed to have non-negative values or ∞. The arcs are directed and $y(i,j)$ is not necessary equal to $y(j,i)$.*

In our networks every node i has a capacity x_i and also a random demand ξ_i. Having in mind application to interconnected power systems, x_i will be called generating capacity and ξ_i local demand. Out of those two we form the network demand $d(i)$, defined as:

$$d(i) = \xi_i - x_i, i \in N. \tag{2.2}$$

If for realization of ξ_i we have $d(i) > 0$, then at node i the local generating capacity is not enough to satisfy the demand. If, however, $d(i) < 0$ then there is surplus capacity at a node i that may be called supply node in that case.

Definition 2.2 *A flow or feasible flow in a network is a real valued function* $f(i,j), (i,j) \in N \times N$ *such that*

$$\begin{aligned} &f(i,j) + f(j,i) = 0 \\ &f(i,j) \leq y(i,j), (i,j) \in N \times N. \end{aligned} \tag{2.3}$$

This definition is compatible with the definition of power flow in electrical engineering, where the flow from j to i is taken as negative of the flow from i to j. In that case $y(i,j) = y(j,i)), \forall (i,j) \in N \times N$.

In what follows we will use the notations:

$$\begin{aligned} f(A,B) &= \sum_{i \in A, j \in B} f(i,j) \\ y(A,B) &= \sum_{i \in A, j \in B} y(i,j), \text{ for } A, B \subset N, A \cap B = \emptyset \\ d(A) &= \sum_{i \in A} d(i), A \subset N. \end{aligned} \tag{2.4}$$

Definition 2.3 *A demand* $d(i), i \in N$ *is said to be feasible if there exists a flow* f *such that*

$$f(N,i) \geq d(i), i \in N. \tag{2.5}$$

Relationships (2.1) and (2.3) define a convex polyhedral cone in the variables $f(i,j), y(i,j), d(i)$.

The question arises: what is the necessary and sufficient condition on the capacity and demand functions that a flow exists satisfying (2.1) and (2.3)?

In other words, how can we characterize the projection of the convex polyhedral cone determined by the linear equalities/inequalities (2.1) and (2.3), onto the space of the $d(i), i \in N, y(i,j), (i,j) \in N \times N$ variables?

The problem was solved by [15] Gale (1957) and [17] Hoffman (1960) and the answer is:

Theorem 2.1 *The demand is feasible if and only if we have the following inequalities:*

$$\begin{aligned} &d(A) \leq y(\bar{A}, A), A \subset N, \\ &\text{where } \bar{A} = N \setminus A. \end{aligned} \tag{2.6}$$

In what follows we will refer to the inequalities (2.6) as Gale-Hoffman or feasibility inequalities.

The network design problem is to minimize the total cost of generation and providing link capacities while satisfying the network demand with the required probability.

$$\begin{aligned} &\min\{\sum_{i \in N} c_i(x_i) + \sum_{(i,j) \in N \times N} c_{ij}(y_{ij})\} \\ &\text{subject to} \\ &P\left(\begin{array}{l} \xi_i \leq x_i + \sum\limits_{j \in N \setminus \{i\}} y_{ji}, \quad i \in N \\ \sum\limits_{i \in I} \xi_i \leq \sum\limits_{i \in I} x_i + \sum\limits_{i \in I, j \in N \setminus I} y_{ji}, \quad \forall I \subseteq N, I \neq \emptyset \end{array} \right) \geq p \\ &Ax + Dy \geq b \end{aligned} \tag{2.7}$$

where $Ax + Dy \geq b$ may simply be a collection of lower and upper bounds for the decision variables x_i, y_{ij}. Problem (2.7) is clearly a particular case of (2.1), where the random vector η is a $(2^n - 1)$-component linear combination of random demands ξ_i.

2.2 Properties of the Network Design Problem

The network design problem is a particular case of the probabilistically constrained stochastic optimization problem. It has several important properties.

Property 2.1 *The random vector η has highly degenerate distribution. The number of coordinates of η may be much larger than the dimensionality of the affine span of the support of η.*

Property 2.2 (Monotonicity) *In problem* (2.1)*:*

1. $c > 0$
2. $T \geq 0$
3. $A \leq 0$

These properties are characteristic to several other examples of probabilistically constrained optimization problems, in particular the flood control system design problem discussed in Section 5.5 and the irrigation system design problem in Section 6.5.

2.3 The Probabilistic Constraint and its Mathematical Properties

The most important part of our stochastic network design problem is probabilistic constraint

$$P(\eta \leq Tx) \geq p \tag{2.8}$$

where p, $(0 \leq p \leq 1)$ is a fixed probability, chosen by ourselves and in practice near 1. The chosen value of p depends on the type of the problem. It is reasonable to choose p as a large probability, e.g. 0.99, in case of a power system, whereas it may be a smaller number in problems where the damage caused by inequalities is smaller. In practice, however, usually we solve the problems with more than one p values in order to learn more about the network and collect more information for our decision making.

The questions that come up in connection with the probabilistic constraint (2.8) are the following:

1. How can we characterize the set of feasible solutions, i.e. the set of vectors x satisfying (2.8)?
2. Under what conditions is it a convex set?
3. How can we easily obtain the function and gradient values of the constraining function in the inequality (2.8)?

To answer these questions, we first define the multivariate p-quantile of a random vector ξ as the set:

$$MVaR_p(\xi) = \{z | P(\xi \leq z) = p\} \tag{2.9}$$

where the notation of $MVaR_p(\xi)$ (short for Multivariate Value at Risk ξ) is taken from Prékopa (2010). The following theorem holds true:

Theorem 2.2 *If ξ is continuously distributed and has positive and continuously differentiable p.d.f. in R^n, then $MVaR_p(\xi)$ is a continuously differentiable surface in R^n such that given an i, $1 \leq i \leq n$, and any values of the variables $z_1, ..., z_{i-1}, z_i, z_{i+1}, ..., z_n \in MVaR_p(\xi)$ uniquely determines the value of z_i. Moreover, z_i is a decreasing function of each variable.*

The proof needs standard mathematical reasoning, therefore it is omitted. Our most important example for the probability distribution, satisfying the conditions of Theorem 2.2, is the multivariate non-degenerate normal distribution. When we apply Theorem 2.2 in probabilistically constrained programming problems, it is enough to assume the positivity of the p.d.f. of ξ in an open set that contains the set of feasible solutions. Before starting the next theorem, we recall the notations of logconcave function and probability measure.

A function $f(z) \geq 0, z \in R^n$ is said to be logconcave if for any $x, y \in R^n$ and $0 < \lambda < 1$ we have the inequality

$$f(\lambda x + (1-\lambda)y) \geq [f(x)^{\lambda}[f(y)]^{1-\lambda}. \tag{2.10}$$

Note that if (2.10) holds for $(x, y) \in D$, where $D \subset R^n$ is a convex set, then f can be extended to the entire space R^n, if we define $f(x) = 0$ for $x \notin D$.

A probability measure P, defined on the Borel sets of R^n is said to be logconcave, if for any convex sets $A, B \subset R^n$ and $0 < \lambda < 1$, we have the inequality

$$P(\lambda A + (1-\lambda)B) \geq [P(A)]^{\lambda}[P(B)]^{1-\lambda}, \tag{2.11}$$

where $\lambda A + (1-\lambda)B = \{\lambda x + (1-\lambda)y \ | x \in A, y \in B\}$ is the Minkowski combination of the sets A and B.

The next theorem is the fundamental theorem of logconcave measures. For the original proof see Prékopa (1971) and Prékopa (1995).

Theorem 2.3 *If $\xi \in R^n$ has a continuous distribution with logconcave p.d.f., then the probability measure generated by ξ, i.e., $P(C) = P(\xi \in C), C \subset R$ is a Borel set, is logconcave.*

Not all logconcave measures have probability density functions. For example, if probability distribution in R^n is concentrated on a hyperplane, where it has density and is logconcave, then the inequality (2.11) holds true for any pair of convex sets in R^n, but the probability measure obviously does not have density in R^n. The following theorem tells us more about log-concavity.

Theorem 2.4 (Prékopa, 1971) *Let $g_i(x, y), i = 1, \ldots, r$ be concave functions in $\mathbb{R}^{m+n}$ where x is an n-component and y is an m-component vector. Let ξ be an m-component random vector having a logarithmic concave probability distribution. Then the function of the variable x:*

$$P(g_i(x, \xi) \geq 0, \;\; i = 1, \ldots, r) \tag{2.12}$$

is logarithmic concave in the space $\mathbb{R}^n$.

Corollary 2.5 *Let A be an $l \times n$ matrix and suppose that the random vector $\xi \in R^n$ has a continuous distribution with logconcave density. Then the random vector $\eta = A\xi$ has a logconcave distribution.*

If η has degenerate distribution such that its support is an affine subset of R^l, then it can be represented in the form

$$\eta = B\xi + d, \tag{2.13}$$

where d is an arbitrary point of the affine support of η and ξ is a lower-dimensional random vector. It follows from Corollary 2.5 that if ξ has a logconcave distribution, then η also has a logconcave distribution. We further assume that $B \geq 0$ element-wise. There is a practical way to approximate the $MVaR_p(\eta)$ surface, using the functional representation of the surface $MVaR_p(\xi)$. Let us choose $z_1, ..., z_{n-1}$ as the independent variables and z_n the dependent variable in that function and write $z_n = z_n(z_1, ..., z_{n-1})$. The next theorem expresses the mentioned representation of $MVaR_p(\eta)$.

Theorem 2.6 *Suppose that ξ has continuous distribution and logconcave p.d.f. that is positive in the entire space R^n, η is the random variable defined by* (2.13) *and $0 < p < 1$. Then we have the relation*

$$MVaR_p(\eta) \supseteq \{Bz + d, z \in MVaR_p(\xi)\} \tag{2.14}$$

Example 2.1 *Let*

$$\eta = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \xi_1 + \xi_2 \\ \xi_1 + \xi_3 \end{pmatrix} \tag{2.15}$$

and $z_3(z_1, z_2)$ is the function representing the boundary of the surface of the set $\{z|P(\xi \leq z) \geq p\}$, then the boundary surface of the set $\{u|P(\eta \leq u) \geq p\}$ contains the set of vectors:

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3(z_1, z_2) \\ z_1 + z_2 \\ z_1 + z_3(z_1, z_2) \end{pmatrix} \tag{2.16}$$

Proof of Theorem 2.6

Since we have the equation $P(\xi \leq z) = p$, it follows that

$$P(B\xi + d \leq Bz + d) = p \tag{2.17}$$

We have to prove that for any $u \leq Bz + d, u \neq Bz + d$, the probability $P(B\xi + d \leq u)$ decreases below p. For such u, the set $\{\xi : B\xi + d \leq u\}$ does not include the point $Bz + d$ together with an open neighborhood of the point; therefore $P\{B\xi + d \leq u\} < P\{B\xi + d \leq Bz + d\} = p$. ■

2.4 Upper bound and the restricted version of the problem

In the setting of the Theorem 2.6,

$$Tx \geq Bz + d,\ z \in MVaR_p(\xi) \tag{2.18}$$

is a sufficient condition for the original probabilistic constraint.

This leads us to a restricted version of the problem:

$$\begin{aligned} &\min c^T x \\ &\text{s.t.} \\ &Tx \geq Bz - y \\ &F_\xi(z) \geq p \end{aligned} \tag{2.19}$$

If ξ has a log-concave distribution (such as, for example, normal), then this problem is a convex optimization problem, and can be solved with standard techniques or the optimization algorithms discussed in Sections 4.2–4.4. Notice, that the dimension of random vector ξ is much less than the dimension of η, which may significantly improve the computation time.

2.5 Discussion

When the distribution of the random vector η is logconcave, problem (2.1) is convex, which means that algorithms for its solution are readily available; however, with the distribution of η being degenerate, the problem presents additional opportunities and challenges. Both opportunities and challenges stem from the fact that the number of coordinates of η may be much larger than the dimensionality of the affine span of the support of η. This difference is especially important, because computation of multi-dimensional probability distributions is computationally challenging.

On the "opportunities" side, we have developed upper and lower bounds, which reduce to the lower-dimensional problem. On the "challenges" side, we have compared, numerically, several algorithms, and identified the major bottleneck: the derivative computation for the probabilistic constraint. The work-arounds include:

1. Use numerical approximation of the derivative (inexact)

2. Use algorithms which do not require derivatives (not many exist)

In particular, we suggested our own derivative-free algorithm, though without proof.

Chapter 3

Topological Elimination of Gale-Hoffman Constraints for the Network Design Problem

If $|N| = n$, then the number of feasibility inequalities is $2^n - 1$, the case of $S = \emptyset$ being trivial, since we have zeros on both sides. This number is very large even if n is relatively small. However, the Gale-Hoffman theorem inexactly enumerates the feasibility inequalities because there may be many of them which [45] (1991) and later by [70] Wallace and Wets (1993) tells us exactly which are the redundant ones. Let (S) designate the Gale-Hoffman inequality corresponding to $S \subset N$ and $G(S)$ the graph with node set S. The mentioned theorem is:

Theorem 3.1 *The inequality $d(S) \leq y(\bar{S}, S)$ is redundant among the Gale-Hoffman inequalities* (2.6) *if and only if at least one of the graphs $G(S), G(\bar{S})$ are not connected. In that case the inequality $d(S) \leq y(\bar{S}, S)$ is the sum of other Gale-Hoffman inequalities.*

More precisely, if $S = S_1 \bigcup S_2$ and there are no arcs between S_1 and S_2, then $(S) = (S_1) + (S_2)$. [45] Prékopa and Boros (1991) stated the "if" part of the theorem, and in a slightly different form, but their proof contains also the proof of the "only if part". Based on Theorem 2.2 we can eliminate the redundant ones from the Gale-Hoffman inequalities. We subsequently enumerate the sets $S \subset N$, according to their cardinalities, and look for $S, \bar{S}$ such that the condition in Theorem 2.2 is satisfied. This is called elimination by graph structure or graph topology, In [45] Prékopa, Boros (1991) other eliminations are also mentioned. These are based on lower and upper bounds on the local demands $\xi_i, i \in N$. In those papers, however, the supports of the ξ_i were supposed to be finite sets and therefore lower and upper bounds for the local demands came up in a natural way. This may happen in case of continuously distributed local demands, too, for example the ξ_i random variables have uniform distribution in known finite intervals.

In one of our examples the demands have normal distributions for which there are no lower and upper bounds, thus only elimination by graph structure comes into account. In another example, however, we have uniform distributions, thus all eliminations should be applied to remove the redundant inequalities.

In our networks every node i has a capacity x_i and also a random demand ξ_i. Having in mind application to interconnected power systems, x_i will be called generating capacity and ξ_i local demand. Out of those two we form the network demand $d(i)$, defined as:

$$d(i) = \xi_i - x_i, i \in N. \tag{3.1}$$

If for realization of ξ_i we have $d(i) > 0$, then at node i the local generating capacity is not enough to satisfy the demand. If, however, $d(i) < 0$ then there is surplus capacity at a node i that may be called supply node in that case. Simple examples for the Gale-Hoffman inequalities are presented in the next section. Once the elimination by graph structure has been carried out, we write up the remaining inequalities in such a way that we separate those in which S has only one element of N, from the others, where partial sums of them appear. This is the following:

$$\begin{aligned} &\xi_i \leq x_i + \sum_{j \in N \setminus \{i\}} y_{ji}, i \in N \\ &\sum_{i \in I_k} \xi_i \leq \sum_{i \in I_k} x_i + \sum_{i \in I_k, j \in N \setminus I_k} y_{ji}, k = 1, ..., l, \end{aligned} \tag{3.2}$$

where $I_1, ..., I_l$ are subsets of N such that $|I_k| \geq 2, k = 1, ..., l$. The random variables in (2.6) form a $n + l$-component random vector that we designate by η, while ξ designates the random vector the components of which are $\xi_1, ..., \xi_n$.

We start the elimination procedure by listing all the non-empty subsets of the vertices of the graph. We then iterate over this list of subsets, and for each subset S check if the subgraph generated by S is connected. If it is not, then the corresponding inequality is redundant and can be eliminated. The algorithm to check if a subgraph is connected is well-known and is as follows:

Algorithm 1 *1. Choose an arbitrary vertex in S and mark it as "connected".*

2. *For each edge (i, j) of the graph, if $i \in S$ and $j \in S$ and either i or j are marked as "connected" then mark the other as "connected".*

3. *If any marking was changed at the previous step, then go to Step 2.*

4. *If at least one element of S is not marked as "connected" then return* ***false*** *otherwise return* ***true****.*

For example, for the network suggested by [11] Fanelli, Prékopa (2013), see Figure 3.1, the initial list of all possible subsets would be:

{1}	{2}	{1,2}	{3}
{1,3}	{2,3}	{1,2,3}	{4}
{1,4}	{2,4}	{1,2,4}	{3,4}
{1,3,4}	{2,3,4}	{1,2,3,4}	{5}
{1,5}	{2,5}	{1,2,5}	{3,5}
{1,3,5}	{2,3,5}	{1,2,3,5}	{4,5}
{1,4,5}	{2,4,5}	{1,2,4,5}	{3,4,5}
{1,3,4,5}	{2,3,4,5}	{1,2,3,4,5}	

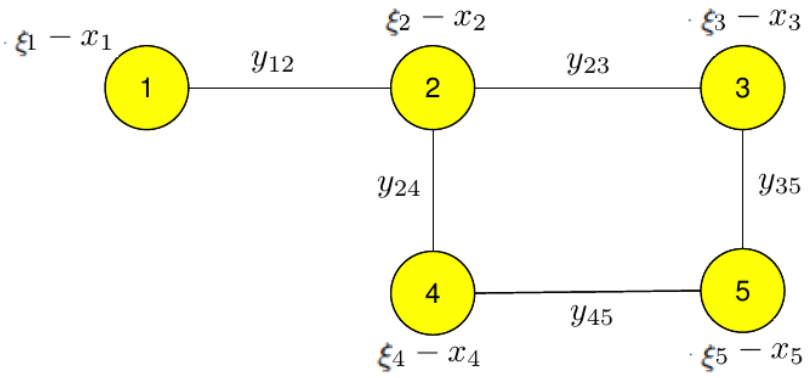


Figure 3.1: An example of a resource distribution network

We then iterate over this list to determine which subsets generate a connected subgraph:

{1}	connected	{2}	connected
{1,2}	connected	{3}	connected
{1,3}	not connected	{2,3}	connected
{1,2,3}	connected	{4}	connected
{1,4}	not connected	{2,4}	connected
{1,2,4}	connected	{3,4}	not connected
{1,3,4}	not connected	{2,3,4}	connected
{1,2,3,4}	connected	{5}	connected
{1,5}	not connected	{2,5}	not connected
{1,2,5}	not connected	{3,5}	connected
{1,3,5}	not connected	{2,3,5}	connected
{1,2,3,5}	connected	{4,5}	connected
{1,4,5}	not connected	{2,4,5}	connected
{1,2,4,5}	connected	{3,4,5}	connected
{1,3,4,5}	not connected	{2,3,4,5}	connected
{1,2,3,4,5}	connected		

To illustrate how a subset is determined to generate a connected subgraph, consider the subset $S = \{1, 2, 5\}$. We start by marking vertex 1 as "connected", see Algorithm 1. We then iterate over all edges of the graph, and when encounter the edge $(1, 2)$, mark the vertex 2 as "connected". Since marking of the vertex 2 has changed, we iterate over all edges again, but this time we do not encounter any edge which would connect vertices of S such that one of them is marked as "connected" and the other is not. At this point we check if all vertices of S are marked "connected": we see that vertices 1 and 2 are, but the vertex 5 is not. Therefore the subset $S = \{1, 2, 5\}$ generates a non-connected subgraph.

There are 10 inequalities which are eliminated. The remaining 21 inequalities are:

$$
\begin{aligned}
&x_1 - \xi_1 + y_{12} \geq 0 \\
&x_2 - \xi_2 + y_{12} + y_{23} + y_{24} \geq 0 \\
&x_1 - \xi_1 + x_2 - \xi_2 + y_{23} + y_{24} \geq 0 \\
&x_3 - \xi_3 + y_{23} + y_{35} \geq 0 \\
&x_2 - \xi_2 + x_3 - \xi_3 + y_{12} + y_{24} + y_{35} \geq 0 \\
&x_1 - \xi_1 + x_2 - \xi_2 + x_3 - \xi_3 + y_{24} + y_{35} \geq 0 \\
&x_4 - \xi_4 + y_{24} + y_{45} \geq 0 \\
&x_2 - \xi_2 + x_4 - \xi_4 + y_{12} + y_{23} + y_{45} \geq 0 \\
&x_1 - \xi_1 + x_2 - \xi_2 + x_4 - \xi_4 + y_{23} + y_{45} \geq 0 \\
&x_2 - \xi_2 + x_3 - \xi_3 + x_4 - \xi_4 + y_{12} + y_{35} + y_{45} \geq 0 \\
&x_1 - \xi_1 + x_2 - \xi_2 + x_3 - \xi_3 + x_4 - \xi_4 + y_{35} + y_{45} \geq 0 \qquad (3.3) \\
&x_5 - \xi_5 + y_{35} + y_{45} \geq 0 \\
&x_3 - \xi_3 + x_5 - \xi_5 + y_{23} + y_{45} \geq 0 \\
&x_2 - \xi_2 + x_3 - \xi_3 + x_5 - \xi_5 + y_{12} + y_{24} + y_{45} \geq 0 \\
&x_1 - \xi_1 + x_2 - \xi_2 + x_3 - \xi_3 + x_5 - \xi_5 + y_{24} + y_{45} \geq 0 \\
&x_4 - \xi_4 + x_5 - \xi_5 + y_{24} + y_{35} \geq 0 \\
&x_2 - \xi_2 + x_4 - \xi_4 + x_5 - \xi_5 + y_{12} + y_{23} + y_{35} \geq 0 \\
&x_1 - \xi_1 + x_2 - \xi_2 + x_4 - \xi_4 + x_5 - \xi_5 + y_{23} + y_{35} \geq 0 \\
&x_3 - \xi_3 + x_4 - \xi_4 + x_5 - \xi_5 + y_{23} + y_{24} \geq 0 \\
&x_2 - \xi_2 + x_3 - \xi_3 + x_4 - \xi_4 + x_5 - \xi_5 + y_{12} \geq 0 \\
&x_1 - \xi_1 + x_2 - \xi_2 + x_3 - \xi_3 + x_4 - \xi_4 + x_5 - \xi_5 \geq 0
\end{aligned}
$$

Chapter 4

Stochastic Network Design Problem using Probabilistic Constraint.

We are looking for optimal capacities x_i, y_{ij} that we call generating and transmission capacities. Our network design problem, however, is applicable not only for interconnected power systems but a variety of practical problems, with suitable interpretation.

Our problem differs from that of [52] Prékopa, Unuvar (2015) only in that here we are working with continuously distributed, rather than discrete random variables, hence the problems mentioned in the Prékopa, Unuvar paper are good examples also here, if the practical situation can better be described by continuously distributed random variables. Our problem is the following:

$$
\begin{aligned}
& min\{\sum_{i\in N} c_i(x_i) + \sum_{(i,j)\in N\times N} c_{ij}(y_{ij})\} \\
& \text{subject to} \\
& P\left(\begin{array}{l} \xi_i \le x_i + \sum\limits_{j\in N\setminus\{i\}} y_{ji}, i \in N \\ \\ \sum\limits_{i\in I_k} \xi_i \le \sum\limits_{i\in I_k} x_i + \sum\limits_{i\in I_k, j\in N\setminus I_k} y_{ji}, k = 1, ..., l, \end{array}\right) \ge p \\
& Ax + Dy \ge b
\end{aligned}
\tag{4.1}
$$

where $Ax + Dy \geq b$ may simply be a collection of lower and upper bounds for the decision variables x_i, y_{ij}. Problem (4.1) can equivalently be stated in the following way:

$$
\begin{aligned}
& min\{\sum_{i\in N} c_i(x_i) + \sum_{(i,j)\in N\times N} c_{ij}(y_{ij})\} \\
& \text{subject to} \\
& x_i + \sum_{j\in N\setminus\{n\}} y_{ij} \geq z_i, i = 1, ..., n \\
& \sum_{i\in I_k} x_i + \sum_{i\in I_k, j\in N\setminus I_k} y_{ij} \geq \sum_{i\in I_k} z_i, k = 1, ..., l \\
& Ax + Dy \geq b, \\
& \text{where } z_n = z_n(z_1, ..., z_{n-1})
\end{aligned}
\tag{4.2}
$$

. Problem (4.2) is a disjunctive, semi-infinite problem, but it is convex because the equivalent problem (4.1) is convex, by the theorem in Chapter 3, provided that the condition in connection with the probability distribution of ξ are satisfied. That condition prescribed the positivity of the p.d.f. of ξ in the entire space. The condition can be relaxed by requiring the positivity of the p.d.f. to hold in an open set that contains the set of those x, y vectors which are feasible with respect to the deterministic constraints.

4.1 Algorithmic Solution of the Network Design Problem

For brevity, we introduce the notation

$$h(x) = P\{Tx \geq \xi\} - p \tag{4.3}$$

and suppose that Slater's condition is satisfied:

$$\exists x^0 \in K^0 \text{ such that } h(x^0) > 0, \tag{4.4}$$

4.2 Supporting hyperplane

The supporting hyperplane method, originally developed by [69] Veinott (1967) was adapted by [40] Prékopa and Szantai (1978) and [63] Szantai (1988) to solve probabilistic constrained optimization problems with continuously distributed random vector and logconcave p.d.f.

The supporting hyperplane algorithm can be summerized as follows:

Algorithm 2 (Supporting hyperplane)

Step 0. Find x^0 *satisfying* $Ax^0 \geq b, x^0 \geq 0, h(x^0) \geq 0$. *Go to Step 1.*

Step 1. Solve the LP:

$$\begin{aligned} &\min c^T x \\ &\textit{subject to} \\ &Ax \geq b \\ &\nabla h(x^i)(x - x^0) \geq 0, i = 1, ..., k \\ &x \geq 0. \end{aligned} \tag{4.5}$$

Let x^{*k} *be an optimal solution. Go to Step 2.*

Step 2. Check for the sign of $h(x^{*k})$. *If* $h(x^{*k}) \geq 0$, *Stop, optimal solution to the problen has been found. Otherwise go to Step 3.*

Step 3: Find λ^k *such that* $0 \leq \lambda^k \leq 1$ *and* $h(x^0 + \lambda^k(x^{*k} - x^0)) = 0$.

Define $x^{k+1} = x^0 + \lambda^k(x^{*k} - x^0)$ *and go to Step 4.*

Step 4. Introduce the cut: $\nabla h(x^{k+1})(x - x^0) \geq 0$. *Set* $k \leftarrow k + 1$ *and go to Step 1.*

4.3 Prékopa-Vizvári-Badics (PVB) Algorithm

The Prékopa-Vizvári-Badics algorithm was introduced to solve probabilistically constrained problems, where the random right-hand side vector ξ has a discrete distribution. It is based on the concept of a p-efficient point by use of which the problem can be reformulated as a disjunctive problem. First we outline the original use of it.

Let $\xi = (\xi_1, ..., \xi_n)^T$ be a random vector, where the support of ξ_i in $z_i = \{z_{i0}, z_{i1}, ..., z_{ik_i}\}, i = 1, ..., n$ and let $z = z_i \times ... \times z_n$.

Definition 4.1 *A value* $z \in Z$ *of* ξ *is a p-efficient point of the probability distribution or its distribution function if* $F(z) \geq p$ *and there is no possible value* $w \in Z$ *such that* $w \leq z, w \neq z, F(w) \geq p$.

Let $z_1, ..., z_N$ be the set of all p-efficient points of the distribution of ξ.

Algorithm 3 (Prékopa-Vizvári-Badics) *Step 0. Enumerate all p-efficient points* $z_1, ...z_N$. *Let* $\bar{z} = \sum_{i=1}^{N} z_i/N$. *Inicialize* $k \leftarrow 0$ *and go to Step 1.*

Step 1. Determine the vectors $w_1, ..., w_h$ such that they form a basis in the space orthogonal to the affine space spanned by $z_1 - \bar{z}, \ldots, z_N - \bar{z}$.

Step 2. Solve the LP:

$$\begin{aligned} &\min c^T x \\ &s.t. \\ &Ax \geq b \\ &w_l^T(Tx - u - \bar{x}) = 0, l = 1, ..., h \\ &v_i^T(Tx - u - \bar{z}) \geq 0, i = 1, ..., k \\ &x \geq 0, u \geq 0. \end{aligned} \tag{4.6}$$

If $k = 0$, then ignore the constraint for $i = 1, ..., k$. Let (x^k, u^k) be an optimal solution. Go to step 3.

Step 3. For $t = Tx^k - u^k - \bar{z}$ solve the auxiliary problem

$$\begin{aligned} &\min e^T \mu = \alpha \\ &\sum_{i=1}^{N} (z_i - \bar{z})\mu_i = Tx^k - u^k - \bar{z} \\ &\mu \geq 0 \end{aligned} \tag{4.7}$$

where $e = (1, 1, ..., 1)^T, (x^k, u^k)$ is the current optimal solution, and $\mu = (\mu_1, ..., \mu_N)$ is the decision vector.

4.4 Hybrid algorithm: combination of the supporting hyperplane and PVB algorithms

It was first introduced in [49] Prékopa (2007). An improved version is due to [53] Prékopa, Myndyuk (2016). Here, however, we assume that ξ has continuous distribution. The role of p-efficient points is taken by points on the boundary of the set of feasible solutions, obtained in the subsequent iterations. The combined application of PVB and supporting hyperplane algorithms provides solution method for problem (2.1), where r.v. has continuous distribution. At each iteration both lower and upper bounds for the optimum are available, and if they are sufficiently close we may stop.

Algorithm 4 (Hybrid algorithm)

- *Step 0. Find* x^0 *satisfying* $Ax^0 \geq b, x^0 \geq 0, h(x^0) > 0$. *Let* $k = 0$, $K_0 = \emptyset$. *Go to Step 1.*

- *Step 1. Solve the LP:*

$$\min c^T x$$

$$\textit{subject to}$$

$$Ax \geq b$$

$$\bigtriangledown h(x^i)(x - x^i) \geq 0, i = 1, \ldots, k$$

$$x \geq 0.$$

  Let x^{*k} *be an optimal solution. Go to Step 2.*

- *Step 2. Check for the sign of* $h(x^{*k})$. *If* $h(x^{*k}) \geq 0$, *Stop, optimal solution to problem (3) has been found. Otherwise go to Step 3.*

- *Step 3. Find* λ^k *such that* $0 < \lambda^k < 1$ *and* $h(x^0 + \lambda^k(x^{*k} - x^0)) = 0$. *Define* $x^{k+1} = x^0 + \lambda^k(x^{*k} - x^0)$ *and go to step 4.*

- *Step 4. Let* $K_{k+1} = K_k \cup \{Tx^{k+1}\}$. *Run PVB algorithm using* K_{k+1} *as the set of p-efficient points. Let* $\tilde{x}^{*k+1}$ *be the result of the PVB algorithm. Then* $c^T x^{*k+1}$ *is a lower bound of the optimal objective value and* $c^T \tilde{x}^{*k+1}$ *is an upper bound of the optimal objective value. Go to Step 5.*

- *Step 5. Introduce the cut:* $\bigtriangledown h(x^{k+1})(x - x^{k+1}) \geq 0$, *set* $k \leftarrow k + 1$ *and go to Step 1.*

4.5 Probability Distributions Commonly Arising in the Considered Problems

The effectiveness of the algorithms described in this section rests on our ability to compute the function $h(x)$, that is the CDF of the respective distribution, and its gradient efficiently. The distributions most commonly used to describe a wide range of physical phenomena are the normal distribution and the uniform distribution [**?**].

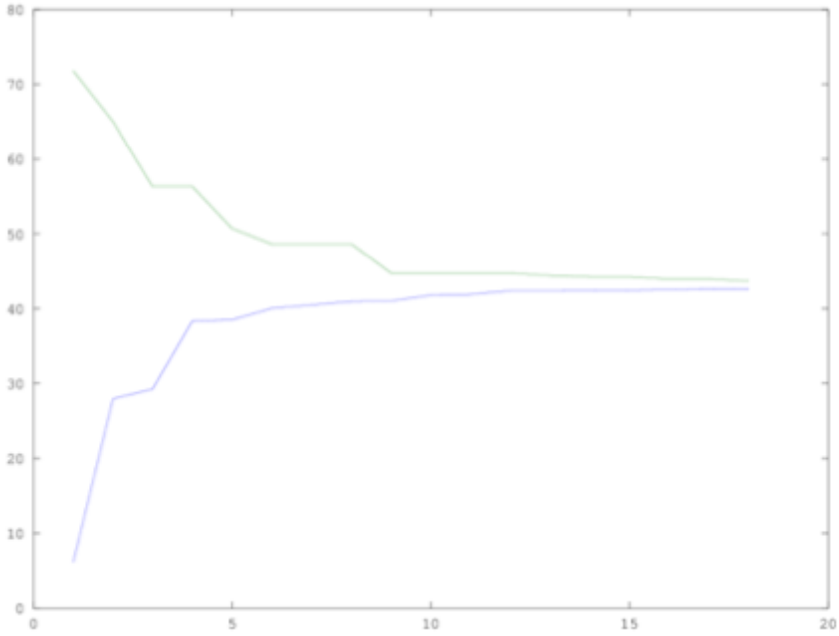


Figure 4.1: Lower and upper bounds of the objective function graphical representation up to convergence to the optimal result

4.5.1 Normal Distribution

The normal distribution is famously logconcave [46] and leads to the stochastic optimization problem being convex. Numerical methods for evaluating the CDF of a multivariate normal distribution have been extensively studied by Dunnett and Sobel (1955), Steck and Owen (1962), Schervish (1984,1985), Nelson (1991), Dunnett (1989,1993), Drezner (1992), Genz (1992), Hajivassiliou, McFadden and Rudd (1996). In particular, a recent algorithm by Genz [16] is applicable to singular multivariate normal distributions, which is crucial for our purposes. The source code of his algorithms has also been made available at `http://www.math.wsu.edu/faculty/genz/software/software.html`

The gradient-based algorithms described in previous sections require the computation of derivatives of the CDF. The derivatives of the mulitvariate normal distribution can be expressed through CDF's of lower dimenstional normal distribution.

Lemma 4.1 *For the CDF* $F(z;\mu,\Sigma)$ *of a multivariate normal distribution with mean* μ

and variance-covariance matrix Σ*, the partial derivative*

$$\frac{\partial}{\partial z_i} F(z; \mu, \Sigma) = F(z_{-i}; \tilde{\mu}^{(i)}(z_i), \tilde{\Sigma}^{(i)}) f(z_i; \mu_i, \Sigma_{i,i}) \tag{4.8}$$

where $z_{-i} = (z_1, \ldots, z_{i-1}, z_{i+1}, \ldots, z_n)^T$*,*

$$\tilde{\mu}^{(i)}(z_i) = \mu_{-i} + \frac{1}{\Sigma_{i,i}} \Sigma_{-i,i} \cdot (z_i - \mu_i) \qquad \tilde{\Sigma}^{(i)} = \Sigma_{-i,-i} - \frac{1}{\Sigma_{i,i}} \Sigma_{i,-i}^T \Sigma_{i,-i} \tag{4.9}$$

4.5.2 Uniform Distribution

- Convexity of the constraint set: the distribution is logconcave
- Complexity of the computation of the CDF:
  - The problem is computationally hard

    For the case of uniform distribution, computation of the probability $P\{Tx \geq \xi\}$ in the left-hand side of the probabilistic constraint reduces to the computation of the volume of a multi-dimensional polytope given as a set of inequality constraints. This problem is known to be #P-hard [10] (For a treatment of "#P-hardness" see [68])
  - The exact algorithm:

    The most straightforward approach to volume computation is:

    1. choose a vertex of the polytope
    2. enumerate facets of the polytope which do not containt the chosen vertex
    3. for each facet, compute the volume of the pyramid with this facet as the base and the chose vertex as the apex
    4. sum the volumes of the pyramids

    Under the hood this reduces to vertex enumeration. See [2] for an efficient algorithm.

    A more efficient algorithm for volume computation has been suggested by Lawrence [27]
  - An even more efficient, but probabilistic method for volume computation is Markov chain Monte Carlo [9].

The uniform distribution typically arises when the random variables to be modelled are known to satisfy certain (often linear) constraints, but nothing else is known about their distribution.

The multivariate uniform distribution over a convex polytope is logconcave. Computation of the probability $P\{Tx \geq \eta\}$ in the left-hand side of the probabilistic constraint reduces to the computation of the volume of a multi-dimensional polytope given as a set of inequality constraints. This problem is known to be #P-hard [10] (For a treatment of "#P-hardness" see [68]) A number of algorithms has been proposed in [27] and also "An Analytical Expression and an Algorithm for the Volume of a Convex Polyhedron in R^n" by Lasserre) We have used the result by Lawrence [27]

Theorem 4.1 (Lawrence) *Suppose a polyhedron* $P = \{x \in R^n : r_i(x) = b_i - a_i^T x \geq 0$ *for* $i = 1, \ldots, m\}$. *Suppose further that* P *is bounded and that for each vertex* v *of* P *the number of indices* i *such that* $r_i(v) = 0$ *is* n. *In particular,* P *is a simple polytope. Suppose* $c \in R^n$ *and* $d \in R$ *are such that the function* $f(x) = c^T x + d$ *is nonconstant on each edge of* P. *Given a vertex* v *of* P, *let*

$$N_v = \frac{f(v)^n}{n!\delta_v \gamma_1 \cdots \gamma_n} \tag{4.10}$$

where, if the indices of the constraints which are binding at v *are* $i_1, \ldots, i_n$, *then* $\gamma_1, \ldots, \gamma_n$ *are such that*

$$c = \gamma_1 a_{i_1} + \cdots + \gamma_n a_{i_n} \tag{4.11}$$

and δ_v *is the absolute value of the determinant of the* $n \times n$ *matrix whose columns are* $a_{i_1}, \ldots, a_{i_n}$. *Then the volume of* P *is*

$$\mathrm{vol}(P) = \sum_v N_v. \tag{4.12}$$

The enumeration of vertices present in the sum (4.12) has been implemented based on Avis and Fukuda [2]

An alternative to the exact methods is the Monte-Carlo approach which could be implemented as follows

1. For a given polytope $Ax \leq b$ find a box $l \leq x \leq u$ which contains the polytope.

2. Generate a random sample $x^{(1)}, \ldots, x^{(N)}$ uniformly distributed in the box.

3. Count the number M of those $x^{(i)}$ for which $Ax^{(i)} \leq b$

4. The volume of the polytope is approximately $\frac{M}{N} \prod_j (u_j - l_j)$

Notice, that the CDF of the uniform distribution is non-smooth: its gradient does not exist at the points z where the boundary of the quadrant $\zeta \leq z$ contains a vertex of the support polytope of the distribution. However, the algorithms described in this Chapter are still applicable, we should just use a subgradient of the CDF whenever the gradient is undefined.

Chapter 5

Numerical Examples

We considered three numerical examples. Two for interconnected power systems: an 8-area and 15-area system, and a water reservoir system, to illustrate our design methodology. The first two networks are taken from [43] Prékopa, Boros (1989). The 8-node network is also used in [52] Prékopa, Unuvar (2015), for illustration, but here we use continuously distributed ξ. The third example is new.

5.1 8-node Network Design Problem

In this example we consider an 8-node network, where all the nodes have random demands following normal distribution. Figure 5.1 shows the power distribution network with 8 areas, and table 5.1 shows the parameters of the distribution of the random demand. We have in mind large scale interconnected power systems, where the power flow can realistically be described by flows presented in Chapter 3. Small scale power systems need more physical parameters in their description, see, e.g., Prékopa (2014). The problem to be solved is:

$$\begin{aligned} &\min_x \sum_i x_i \\ &\text{subject to} \\ &P\left\{\sum_{i\in S} x_i + \sum_{\substack{i\in S\\ j\notin S}} y_{ij} \geq \sum_{i\in S} \eta_i \quad \forall S \subseteq N,\ S \text{ - non-eliminated}\right\} \geq p \\ &x_i \geq 0, \end{aligned} \tag{5.1}$$

where N is the collection of the nodes with random demands. The x_i's are decision variables representing production capabilities at the i, η_i's are random demands at the nodes, and the y_{ij} are arc capacities between nodes i and j. The paramaters of the distribution of η_i

are shown in Table 5.1 and the arc capacities are represented as arc labels in Figure 5.1.

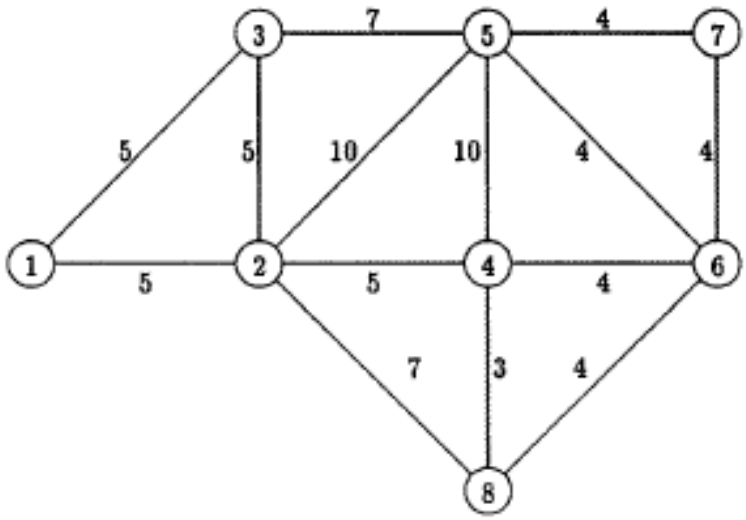


Figure 5.1: Power distribution network with 8 areas

area	expectation	standard deviation
1	8850	1209.4
2	8200	1500
3	9650	1246
4	7900	1469.7
5	9700	1122.50
6	10150	1415.1
7	9550	1171.5
8	9000	1500

Table 5.1: Parameters of the probability distribution of the random demands

We will use normal joint distribution for the components of ξ which implies that there are no finite lower and upper bounds for the demands. Thus, only the topological elimination can be applied for the Gale-Hoffman inequalities. If we execute the elimination, then only 161 out of the initial 255 inequalities remain. Those are presented, together with the upper triangle values of 161×161 covariance matrix in the appendices.

The solution steps are as follows

- Step 0. Formulate the Gale-Hoffman inequalities for our problem and eliminate the

redundant ones using the topological constraint elimination technique described in Chapter 3. The remaining ones determine the probabilistic constraint of the problem:

$$P(Tx \geq \eta) \geq p \tag{5.2}$$

where the matrix T is presented in Appendix A. The random vector η is given by

$$\eta = T\xi - Y \tag{5.3}$$

where

$$Y_j = \sum_{\substack{i \in S \\ j \notin S}} y_{ij} \tag{5.4}$$

and S is the subset of nodes of the graph which determine the constraint j. In the numerical example we chose $p = 0.9$.

Let

$$h(x) = P\left\{ \sum_{i \in S} x_i + \sum_{\substack{i \in S \\ j \notin S}} y_{ij} \geq \sum_{i \in S} \eta_i \quad \forall S \subseteq N,\ S \text{ - non-eliminated} \right\} - p. \tag{5.5}$$

- Step 1. Find x^0 satisfying the probabilistic constraint strictly, and also satisfying the deterministic constraints. Let $k = 0$, $K_0 = \emptyset$.

- Step 2. Solve the LP:

$$\min \textstyle\sum_i x_i$$

subject to

$$x \geq 0.$$

Let $x^{*1} = (0, 0, 0, 0, 0, 0, 0, 0)$ be the optimal solution.

- Step 3. Check that the probabilistic constraint is not satisfied and continue the iteration.

- Step 4. Find λ^0 such that $0 < \lambda^0 < 1$ and $h(x^0 + \lambda^0(x^{*0} - x^0)) = 0$. Define $x^1 = x^0 + \lambda^0(x^{*0} - x^0)$.

- Step 5. Let $K_1 = K_0 \cup \{Tx^1\}$. Run PVB algorithm using K_1 as the set of p-efficient points. Let $\tilde{x}^{*1}$ be the result of the PVB algorithm. Then $\sum_i x_i^{*1}$ is a lower bound of the optimal objective value and $\sum_i \tilde{x}_i^{*1}$ is an upper bound of the optimal objective value.

- Step 6. Find $\bigtriangledown h(x^1)$ and introduce the cut: $\bigtriangledown h(x^1)(x - x^1) \geq 0$. Set $k \leftarrow k + 1 = 1$ and start the next iteration.

- Step 7. Solve the LP:

$$\min \sum_i x_i$$

$$\text{subject to}$$

$$\bigtriangledown h(x^1)(x - x^1) \geq 0$$

$$x \geq 0.$$

  Let x^{*1} be an optimal solution.

- Step 8. Check that x^{*1} does not satisfy the probabilistic constraint and continue with the iteration.

- Step 9. Find λ^1 such that $0 < \lambda^1 < 1$ and $h(x^0 + \lambda^1(x^{*1} - x^0)) = 0$. Define $x^2 = x^0 + \lambda^1(x^{*1} - x^0)$.

- Step 10. Let $K_2 = K_1 \cup \{Tx^2\}$. Run PVB algorithm using K_2 as the set of p-efficient points. Let $\tilde{x}^{*2}$ be the result of the PVB algorithm. Then $\sum_i x_i^{*2}$ is a lower bound of the optimal objective value and $\sum_i \tilde{x}_i^{*2}$ is an upper bound of the optimal objective value.

- Step 11. Find $\bigtriangledown h(x^2)$ and introduce the cut: $\bigtriangledown h(x^2)(x - x^2) \geq 0$. Set $k \leftarrow k + 1 = 2$ and proceed to the next iteration.

Here we present the corresponding numerical values found by the algorithm in the first two iterations.

Iteration 1

$$x_0 = (18022, 18022, 18022, 18022, 18022, 18022, 18022, 18022)^T \tag{5.6}$$

$$x^{*1} = (10104, 10769, 9546.8, 7583.5, 9707.5, 10364, 10251, 9522.3)^T \tag{5.7}$$

$$\lambda^1 = 0.126143 \tag{5.8}$$

$$x^1 = (11103, 11684, 10616, 8900.3, 10756, 11330, 11232, 10595)^T \tag{5.9}$$

$$\tilde{x}^{*1} = (11103, 11684, 10616, 8900.3, 10756, 11330, 11232, 10595)^T \tag{5.10}$$

$$\text{lower bound (from supporting hyperplane)} = 77847.9 \tag{5.11}$$

$$\text{upper bound (from PVB algorithm)} = 86215.1 \tag{5.12}$$

$$\begin{aligned} \bigtriangledown h(x^1) = (&9.0007 \cdot 10^{-6}, 3.3065 \cdot 10^{-7}, 3.0922 \cdot 10^{-5}, 2.5632 \cdot 10^{-5}, \\ &1.8862 \cdot 10^{-6}, 4.1845 \cdot 10^{-5}, 3.5063 \cdot 10^{-5}, 3.5313 \cdot 10^{-5})^T \end{aligned} \tag{5.13}$$

Iteration 2

$$x^{*2} = (9399.8, 6922.3, 11673, 9350.6, 8678.7, 11963, 10251, 10155)^T \tag{5.14}$$

$$\lambda^2 = 0.0999722 \tag{5.15}$$

$$x^2 = (10262, 8032, 12308, 10218, 9612.8, 12569, 11028, 10942)^T \tag{5.16}$$

$$\text{lower bound (from supporting hyperplane)} = 78393.4 \tag{5.17}$$

$$\text{upper bound (from PVB algorithm)} = 84970.2 \tag{5.18}$$

$$\begin{aligned}\bigtriangledown h(x^2) = (&4.5986 \cdot 10^{-5}, 4.5103 \cdot 10^{-5}, 3.634 \cdot 10^{-6}, 8.171 \cdot 10^{-6}, \\ &1.6741 \cdot 10^{-5}, 6.1076 \cdot 10^{-6}, 4.8979 \cdot 10^{-5}, 2.5266 \cdot 10^{-5})^T\end{aligned} \tag{5.19}$$

5.2 Comparison to a discrete distribution

Prékopa, Unuvar (2015) consider a very similar problem where the demands have a discrete (binomial) distribution. In this section we consider a corresponding problem with normally distributed demands.

We consider a corresponding problem where the demands have normal distribution with the same means and variances as in [52] Prékopa, Unuvar (2015) and the rest of the parameters are the same. The comparison of the obtained solutions is shown in Table 5.2.

	Binomial distribution (Prékopa, Unuvar (2015))	Normal distribution	Normal distribution, restricted version
x_1	69	63.444	79.771
x_2	60	44.430	61.966
x_3	40	39.198	38.301
x_4	67	58.928	71.748
x_5	81.0192	61.520	91.699
x_6	61.9808	35.218	53.083
x_7	42	39.023	39.486
x_8	46	44.513	43.549
objective	1233	1083	1238

Table 5.2: Comparison of the optimal values for the 8-node problem, with binomial versus normal demand distributions.

The solution obtained for the case of the binomially distributed demands provides us with an upper bound for the optimal value based on Theorem 2.6. The respective bound for problem with normal distribution is also shown in Table 5.2.

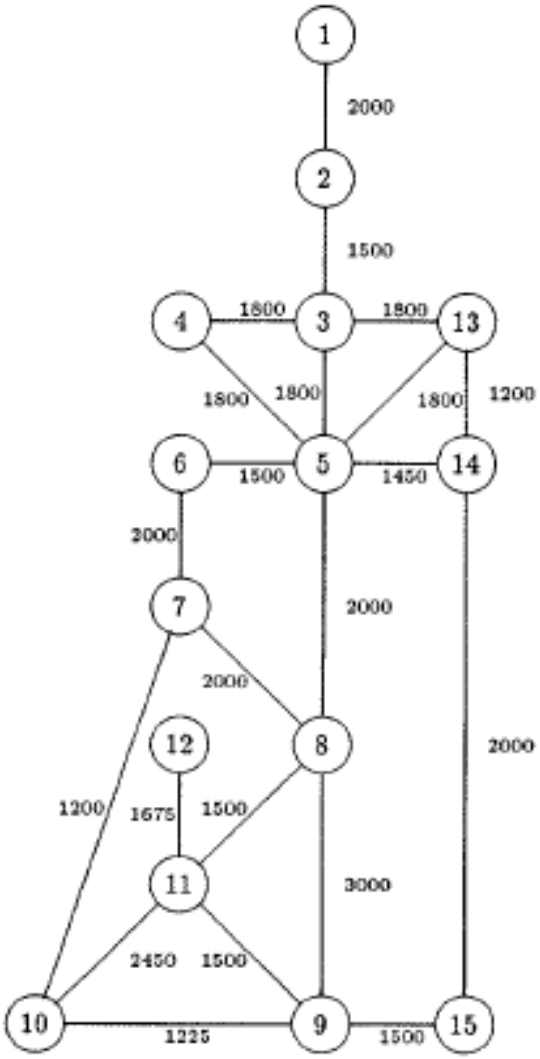


Figure 5.2: Power distribution network with 15 cities

area	expectation	standard deviation
1	1.0000e+03	1.3416e+03
2	1.0000e+03	1.3416e+03
3	1.0600e+03	1.5863e+03
4	8.0000e+02	9.7980e+02
5	1.3000e+03	1.8193e+03
6	0.0000e+00	0.0000e+00
7	1.0000e+03	1.3416e+03
8	6.0000e+02	9.1652e+02
9	8.0000e+02	9.7980e+02
10	1.2000e+03	1.3266e+03
11	9.0000e+02	1.3379e+03
12	1.1000e+03	1.3379e+03
13	9.4000e+02	1.3403e+03
14	1.2200e+03	1.5721e+03
15	1.5000e+03	1.7748e+03

Table 5.3: Parameters of the probability distribution of the random demands

5.3 15-node Network Design Problem

5.4 Summary of computational complexity of the power distribution problems

In this section we summarize the computational effort required to solve problem (5.1) for the 8-node (Chapter 5.1) and 15-node (Chapter 5.3) power network.

5.5 Flood Control Problem

The problem is to design a reservoir system for flood control. Assume now that we can build reservoirs in some parts of the river system, represented by some edges in a directed tree, see Figure 5.3, and the only purpose of the reservoirs will be to retain the flood. This we assume to be random, exist, periodically and independently, once a year. The water comes from terminal points. We assume that the total water quantities can be separated in such a way that we can quantitatively give that amounts which can be lead between river banks and also that amounts which have to be retained by the reservoir system. When retaining the flood, we accept the following policy: first we start to fill up those upstream

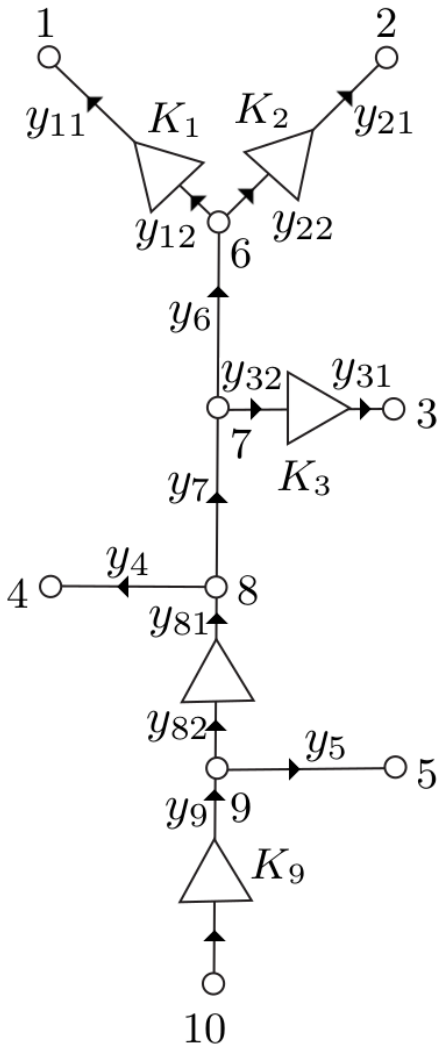


Figure 5.3: Reservoir system for flood control

problem	algorithm	iter.	prob. computations	computation time (sec)	obj. val.	prob. val
8 areas	supporting hyperplane	57	374 + 17 * 161	3998	82966	0.89837
8 areas	hybrid	57	374 + 17 * 161	3998	82966	0.89837
8 areas	restricted version with supporting hyperplane	132	2882	270	96617	0.99979
15 areas	supporting hyperplane	121	2640 + 120 * 2094	4708210	24142.74	0.89852
15 areas	hybrid	121	2640 + 120 * 2094	4708210	24142.74	0.89852
15 areas	supporting hyperplane with approx. derivative	43	924 + 42 * 30	73515	25930.29	0.91968
15 areas	hybrid with approx. derivative	43	924 + 42 * 30	73515	25930.29	0.91968
15 areas	restricted version with supporting hyperplane	519	11396	3078	60481	0.989

Table 5.4: 8-node and 15-node problems solutions with different algorithms: iterations, probability computations, objective values and probability values.

reservoirs which are located on terminal edges. Then, if the flood cannot be retained by these reservoirs, start to fill up the reservoirs on the next edges by the overflown quantities and those input quantities which arrive on terminal edges without reservoir, etc.

Here we are dealing with a special interpretation of the flow and the demand. As we see in Figure 5, the arrows indicating the flow directions are upstream, not downstream. In fact, we are dealing with an upstream flow of freeboard which is initially equal to x_9 at the last reservoir (node 10 can be eliminated, it just designates the area to be protected) and equal to 0 in all the sources. As regards the demand function, first we remark that, together with the reservoir building sites (the triangles) there are all together 14 of them. At each node, however, there may be a source, $\xi = 0$ at reservoir site and $x = 0, \xi = 0$ at a node having one of them, then $\xi - x$ is the demand function as in our general model formulation.

To every terminal vertex there corresponds a random input water quantity. These will be denoted by $\xi_1, ..., \xi_5$. The reservoir capacities to be determined will be denoted by the symbols x_i and the capacities of the river segments will be denoted
$y_{11}, y_{12}, y_{21}, y_{22}, y_{31}, y_{32}, y_4, y_5, y_6, y_7, y_{81}, y_{82}$. It is convenient to choose the subscript i in such a way that it coincide with the subscript of that vertex from which the edge, having the reservoir with capacity x_i, starts.

The random inflows ξ_i are assumed to have jointly normal distribution described by the correlation matrix

$$R_1 = \begin{pmatrix} 1.0 & 0.0 & 0.6 & 0.4 & 0.0 \\ 0.0 & 1.0 & 0.5 & 0.3 & 0.3 \\ 0.6 & 0.5 & 1.0 & 0.7 & 0.6 \\ 0.4 & 0.3 & 0.7 & 1.0 & 0.4 \\ 0.0 & 0.3 & 0.6 & 0.4 & 1.0 \end{pmatrix} \tag{5.20}$$

and expectations, standard deviations given in Table 5.5 The objective function is assumed to be linear (for simplicity) and to be minimized. It is given by:

$$\begin{aligned} &0.4x_1 + 0.5x_2 + 0.6x_3 + 1.2x_8 + 1.8x_9 + y_{11} + 0.25y_{12} + y_{21} + \\ &\quad 0.25y_{22} + y_{31} + 0.25y_{32} + y_4 + y_5 + 0.25y_6 + 0.25y_7 + 0.25y_{81} + 0.25y_{82} \end{aligned} \tag{5.21}$$

	Expectations	Standard deviations
x_1	0.8	0.2
x_2	1.5	0.3
x_3	1.2	0.6
x_4	0.5	0.4
x_5	0.7	0.3

Table 5.5: Parameters of the distribution of the random inflows

The application of the hybrid method gave the optimal solutions shown in Table 5.6

	p=0.9		p=0.95	
	full	restricted	full	restricted
x_1	1.2912	1.26705	1.27494	1.24726
x_2	2.13652	2.12588	2.12031	2.18267
x_3	2.19962	3.5202	2.18969	2.17277
x_8	0.86924	1.1618	0.85266	1.18706
x_9	1.31607	1.23201	1.3353	1.30713
y_{11}	1.2845	1.26705	1.27605	1.24726
y_{12}	0	0	0	0
y_{21}	2.14575	2.12588	2.12488	2.18267
y_{22}	0.00747	0	0.01125	0
y_{31}	2.23178	3.5202	2.21704	2.17277
y_{32}	0.01334	0	0.01312	0
y_4	1.22224	1.1618	1.24018	1.18706
y_5	1.31536	1.23201	1.33126	1.30713
y_6	0	0	0.00583	0
y_7	0.01615	0	0.01225	0
y_{81}	1.22731	1.1618	1.24698	1.18706
y_{82}	1.05317	1.1618	1.14589	1.18706
objective	15.1	17.18	15.11	15.36

Table 5.6: Optimal reservoir and river capacities found by the hybrid algorithm for the Flood Control Problem

Chapter 6

Derivative-free algorithm approach

The obvious reasons to try derivative-free optimization algorithms are the following:

1. Derivative computation takes the majority of the computation time,
2. Especially so for problems with degenerate distribution of the random variable (this is because analytic computation of the derivative involves the chain rule):
   - with $y = Tx$, differentiate $P\{y \geq \xi\}$ in y
   - the derivative in x is T^T(derivative in y)
   - the dimensionality of y may be much greater than the dimensionality of x
3. Especially so for the problems where the probabilistic constraints are generated algorithmically based on a certain rule, and an explicit formula for T is not even available. As an example, see the irrigation problem in Section 6.5 below.

6.1 Direct search methods

Various direct search methods, most notably the Nelder-Mead "simplex" method, were proposed in the early 1960s, and have been enormously popular with practitioners ever since [71].

6.1.1 The Nedler-Mead method

In Nedler-Mead algorithm [33], at each iteration we consider a current simplex defined by $n+1$ vertices, at each point in $\mathbb{R}^n$ and their corresponding values of f. At iteration k, we order and label the current set of vertices as $x_1^{(k)}, ..., x_{(n+1)}^{(k)})$ such that

$$f_1^{(k)} \leq f_2^{(k)} \leq ... \leq f_{n+1}^{(k)} \tag{6.1}$$

where $f_i^{(k)}$ is $f_(x_i^{(k)})$.

Let us say that $x_1^{(k)}$ is the *best* point, and $x_{n+1}^{(k)}$ is the *worst* point, since we are looking to minimize f.

There are two possible scenarios of Nedler-Mead iteration: the *accepted* point (a single new vertex) replaces x_{n+1} in the set of vertices for the new iteration; or a set of n new points including x_1 forms the simplex at the next iteration if a shrink is performed. The search direction is defined by x_{n+1} and $\bar{x}$, the geometric center of all vertices except x_{n+1}.

Iteration k of the Nedler-Mead algorithm.

1. **Order.**

   Order the $n+1$ vertices so that $f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$, using consistent tie-breaking rule.

2. **Reflect**

   Find the *reflection point* x_r from $x_r = \bar{x} + \rho(\bar{x} - x_{n+1})$,

   where $\bar{x}$ is the centroid of the n best vertices (except for x_{n+1}), meaning that $\bar{x} =$

6.1.2 Multidirectional search

The interest in direct-search methods reemerged in 1989 with Ph.D. thesis of Torczon and her papers written together with Dennis [66, 8]. They came up with the multidirectional search method. The reason for developing the multidirectional search method was mostly to make parallel computing environment effective. It was also one of the first direct search algorithms that have been proved to converge to a solution when applied to a convex function.

The multidirectional search method is simplex-based. At every iteration we consider a simplex whose best vertex (corresponding to the lowest function value) is so labeled.

Iteration k of the multidirectional search algorithm with the expansion and contraction coefficients being equal to correspondingly standard values $\chi = 2$ and $\gamma = 1/2$.

Algorithm 5 (Torczon) *For each iteration repeat the following steps*

1. ***Order***.

   The best vertex is labeled as x_1*, so that* $f(x) = argmin_i f(x_i)$.

2. ***Reflect***.

   Define the reflected vertices, $x_r^{(i)} = 2x_1 - x_i$ *for* $i = 2, ..., n+1$*, and evaluate* $f_r^{(i)} = f(x_r^i)$*. If* $min_i\{f_r^{(i)}\} < f_1$*, go to step 3, otherwise go to step 4.*

3. ***Expand***.

   Compute the expanded vertices, $x_e^{(i)} = x_1 + \chi(x_1 - x_i)$*, for* $i = 2, ..., n+1$*, and evaluate* $f_e^{(i)} = f(x_e^{(i)})$*. If* $min_i\{f_e^{(i)}\} < min_i\{f_r^{(i)}\}$*, then accept the expanded simplex, otherwise accept the reflected simplex.*

4. ***Contract***

   Calculate the contracted vertices, $x_c^{(i)} = x_1 + \gamma(x_i - x_1)$ *for* $i = 2, ..., n+1$*, and evaluate* $f_c^{(i)} = f(x_c^i)$*. For* $i = 2, ..., n+1$*, replace* x_i *by* $x_c^{(i)}$*. Terminate the iteration if* $min_i\{f_c^{(i)}\} < f_i$*, otherwise return to step 2.*

The number of function evaluations needed for the multidirectional search is an integer multiple of $2n$. If one of the initial reflection, expansion or contraction steps produces a strict improvement over f_1, the iteration terminates after $2n$ function evaluations. If not, the sequence of reflection, expansion and/or contraction is repeated with a contracted simplex until an improved best function value is found, where each cycle costs a further $2n$ function evaluations. Therefore, a multidirectional search method may require significantly more function evaluations than the Nedler-Mead method. Nonetheless, if the multidirectional search is implemented on a parallel computer with the sufficiently large number of processors, the n function evaluations needed for reflection, expansion or contraction can be performed in parallel. Also, we must underline the great advantage of the multidirectional search method: it's strong convergence properties, which were refined and extended by Torczon. The key of the convergence proofs are uniform linear independence of the simplex edges at every iteration, the scaled lattice structure of generated points and the step control strategy.

6.2 Inner polygonal approximation algorithm

This is a modification of Prékopa-Vizvári-Badics algorithm, suited for problems with continuously-distributed random variables. Denote

$$X = \{\mathbf{x} | P\{T\mathbf{x} \geq \eta\} \geq p\} \tag{6.2}$$

Algorithm 6 (Inner polygonal approximation) *The algorithm is as follows:*

1. *(Initialization)*

   (a) *Find* $\bar{\mathbf{x}}_1 \in \text{int}(X)$

   (b) *In the direction opposite to each of the orths* e_i *find a boundary point of the set* X*:*

   $$\begin{aligned} t_i &= \max\{t : \bar{\mathbf{x}}_1 - e_i t \in X\} \\ \mathbf{x}^{(i)} &= \bar{\mathbf{x}}_1 - e_i t_i \end{aligned} \tag{6.3}$$

   (c) *Let* $E_0 = \{\mathbf{x}^{(i)}\}$ *be the initial value for the set of efficient points. Also, let* $F_0 = \{\mathbf{x}^{(i)}\} = E$ *be the set of "frontier" efficient points.*

   (d) *Set the iteration number* $k = 1$*.*

2. *(Inner problem) At the k-th iteration run Prékopa-Vizvári-Badics algorithm using* E_{k-1} *as the set of efficient points. The Prékopa-Vizvári-Badics algorithm will solve the problem:*

   $$\begin{aligned} &\min \mathbf{c}^T \mathbf{x} \\ &s.t. \\ &A\mathbf{x} \leq \mathbf{b} \\ &\mathbf{x} \geq \mathbf{y} \\ &\mathbf{y} \in \text{conv}(E_{k-1}) \end{aligned} \tag{6.4}$$

   Note: *The set* $\{\mathbf{x} : \mathbf{x} \geq \mathbf{y}, \mathbf{y} \in \text{conv}(E_{k-1})\} \subseteq X$ *and thus problem* (6.4) *yields an upper bound of the true solution of* (2.1)*.*

3. *Check the stopping criterion:*

(a) *Let* $\mathbf{x}_{0k}$ *denote the solution to* (6.4)*. For a chosen tolerance* ϵ*, let* $C_k = \{\mathbf{x} \in E_{k-1} : \|\mathbf{x} - \mathbf{x}_0\| \leq \epsilon, \mathbf{x} \neq \mathbf{x}_{0k}\}$

(b) *Let* L_k *be the hyperplane orthogonal to the direction* $\mathbf{x}_{0k} - \bar{\mathbf{x}}_1$*. Construct the projection of* C_k *onto* L_k *and denote it* $\hat{C}_k$*; find the projection of* $\mathbf{x}_{0k}$ *onto* L_k *and denote it* $\hat{\mathbf{x}}_{0k}$*.*

(c) *If* $\hat{\mathbf{x}}_{0k} \in \text{conv}(\hat{C}_k)$ *then STOP,* $\mathbf{x}_{0k}$ *is the solution. Otherwise, continue.*

4. *(Updating the set of efficient points)*

(a) *From the solution of* (6.4) *note which of the constraints defining* $\text{conv}(E_{k-1})$ *are active. The active constraints have the form:* $\mathbf{s}_{jk}\mathbf{x} \leq r_{jk}$*.*

(b) *For each active constraint determine the vertices of the constraining facet:* $F_{jk} = \{\mathbf{x}^{(i,j)}\} = \{\mathbf{x} \in E_{k-1} : \mathbf{s}_{jk}\mathbf{x} = r_{jk}\}$

(c) *Find the center of the facet* $\mathbf{x}_{0k}^{(j)} = \frac{1}{K}\sum_{i=1}^{K} \mathbf{x}^{(i,j)}$ *where* K *is the cardinality of* F_{jk}*.*

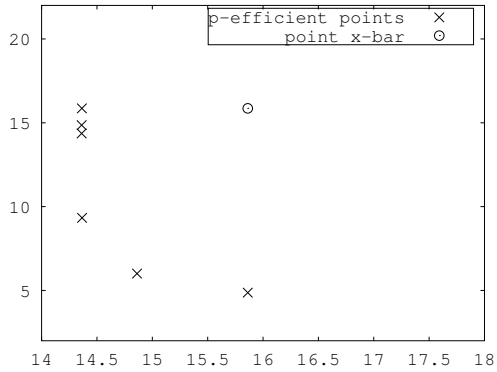
(d) *Find a new efficient point:*

$$t = \max\{t : \mathbf{x}_0^{(j)} + t\mathbf{s}_j \in X\}\mathbf{x}^{(j)} = \mathbf{x}_0^{(j)} + t\mathbf{s}_j$$

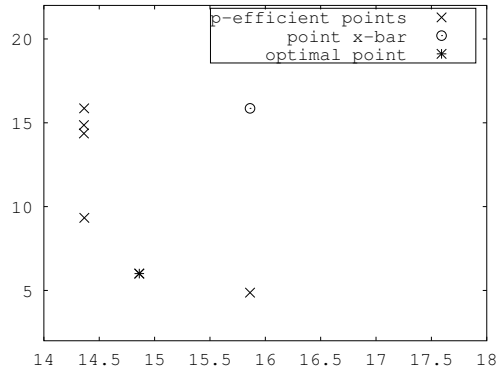
Let $E_k^0 = E_{k-1} \cup \{\mathbf{x}^{(j)}\}$

5. *(Extending frontier) If for each* F_{jk}*:* $F_{jk} \cap F_{k-1} = \emptyset$ *then set* $E_k = E_k^0$*,* $F_k = F_{k-1}$*,* $\bar{x}_{k+1} = \bar{x}_k$ *and go to step 2. Otherwise, let* $\bar{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_k + \mathbf{1}$*; construct new efficient points* $\mathbf{x}^{(i)}$ *as in* (6.3)*; let* $F_k = \{\mathbf{x}^{(i)}\}_i$ *and let* $E_k = E_k^0 \cup F_k$*. Go to step 2.*

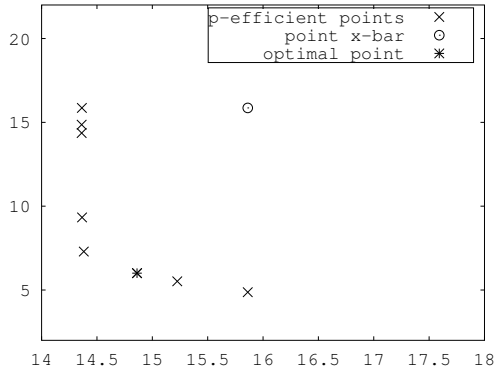
The Figure 6.1 illustrates the operation of one iteration of the algorithm above, as applied to the problem discussed in Section 6.5 below. The Figure 6.2 shows 5 first iterations, namely the efficient points E_{k-1}, the optimal point $\mathbf{x}_{0k}$ and the central point $\bar{\mathbf{x}}_k$ for each iteration. We can observe that the efficient points generated by the algorithm accumulate in the vicinity of the optimal solution.



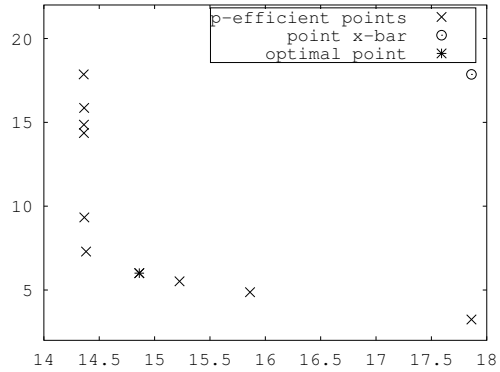
Every iteration starts with some already found efficient points and a central point $\bar{x}$.



As the first step, the efficient point which minimizes the objective function is found.

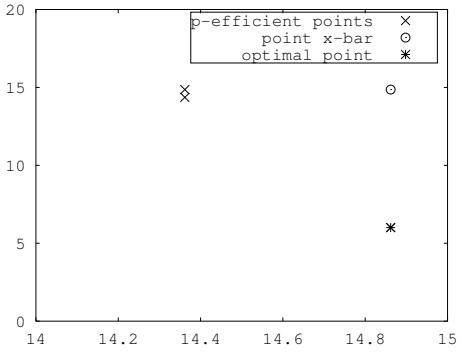


The set of efficient points is refined by adding one extra point for each facet adjacent to the current-optimal point. We start at the center of the facet and move away from the central point until we cross the efficient frontier.

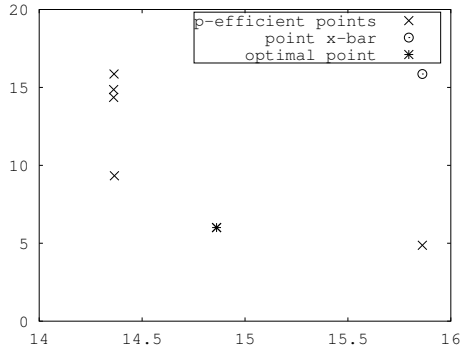


If necessary, the central point is moved further away, so that we can approximate a larger part of the efficient frontier.

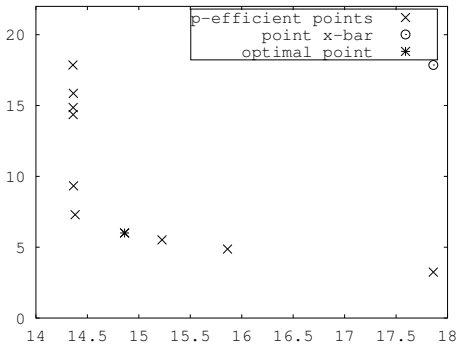
Figure 6.1: Operation of a typical iteration of the optimization algorithm



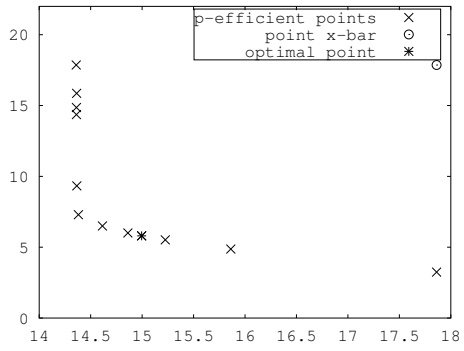
Iteration 1



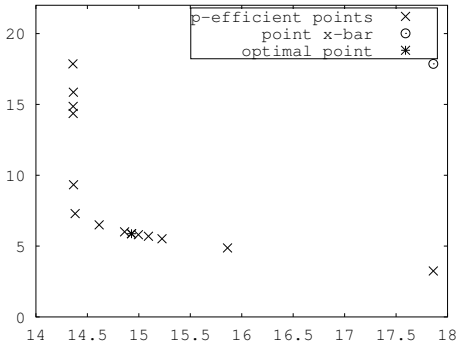
Iteration 2



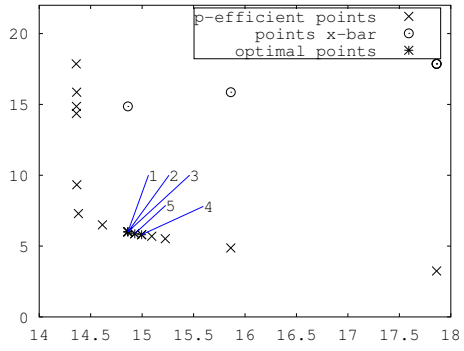
Iteration 3



Iteration 4



Iteration 5



Combined plot of 5 iterations

Figure 6.2: Illustration of the first 5 iterations for a two-reservoir optimization problem

6.3 Construction of unconstrained optimization problem

While the considered derivative-free algorithms solve unconstrained minimization problem, problem (2.1) has nontrivial constraints. It can be reduced to an unconstrained minimization problem under the monotonicity Condition 2.2.

Let B be a matrix whose columns form an orthonormal basis in the linear subspace orthogonal to vector c. Define

$$g(z) = \min\{y : P\{T(Bz + cy) \geq \eta\} \geq p \text{ and } A(Bz + cy) \leq b\} \tag{6.5}$$

Then (2.1) is equivalent to

$$\min_z g(z) \tag{6.6}$$

Notice that $g(z)$ is (almost) an implicitly-defined function whose values could be computed with the bisection method. However, this might be unnecessary: algorithm 5 is based on the comparisons of the form $g(z_1) > g(z_2)$ but does not require the exact values of the function. This lets us adapt the algorithm to our needs even further, by associating with each symplex $\{x_1, \ldots, x_{n+1}\}$ values $\{y_l^{(1)}, \ldots, y_l^{(n+1)}\}$ and $\{y_u^{(1)}, \ldots, y_u^{(n+1)}\}$ such that $g(x_i) \in [y_l^{(i)}, y_u^{(i)}]$. The bounds $y_l^{(i)}$, $y_u^{(i)}$, $y_l^{(j)}$, $y_u^{(j)}$ can be refined as needed whenever a comparison $g(x_i) > g(x_j)$ is of interest.

6.4 Numerical experiments: Power distribution problem

We have solved the problems described in 5.1 and 5.3 using the Torzcon algorithm. The results are shown below:

problem	distribution	iterations	obj. val.	prob. val
8 cities	normal	43	45.55	0.8997
15 cities	normal	127	24218	0.8992
8 cities	uniform	20	20.37	0.895
15 cities	uniform	17	26146	0.897

6.5 Serially Linked Reservoir Network Design Problem with consecutive k-out-of-n type Reliability

The topology of the main river, the side rivers and the possible reservoir sites that are serially linked is illustrated in Figure 6.3.

Time is subdivided into periods and we consider a finite number of them. Periods can be weeks, decades, etc., in practice.

We assume that at the beginning of each period, certain water inputs occur in accordance with the topology of the rivers and reservoirs. If a reservoir becomes full, then the water spills and fills downstream reservoirs. When this is the case, no more water is released from upstream reservoirs to downstream reservoirs. If a reservoir becomes full, then additional water overflows to downstream reservoirs. No more water is released from upstream reservoirs to downstream reservoirs at these times.

At the end of every period, demands occur which can be assigned to separate reservoirs. If possible, every demand is primarily satisfied from the reservoir that they are assigned. If not, then our assumed operating policy is as follows. First, demands are satisfied to the extent that water amounts are in the corresponding reservoirs. Then starting from the reservoir furthest downstream, stop at the first reservoir where there is unsatisfied demand. From here we aggregate the unsatisfied demands of all consecutive reservoirs up to the first non-empty reservoir, and try to meet this aggregated demand. If this is not possible, then we proceed similarly in the upstream direction. If the whole system can meet the demand, then this procedure stops at a certain point and we can then satisfy the downstream demands. This procedure is repeated for the remaining upstream subsystem and so forth. In our model, all demands will be met by a prescribed high probability. Note that the above operating policy is also uniquely determined in the case in which part of the demand remains unsatisfied.

We further assume that if the system is unable to satisfy the demands in a certain period, then a penalty occurs which belongs to the whole system and its function of the unsatisfied part of the total demand. For simplicity, however, we disregard penalties in our model construction.

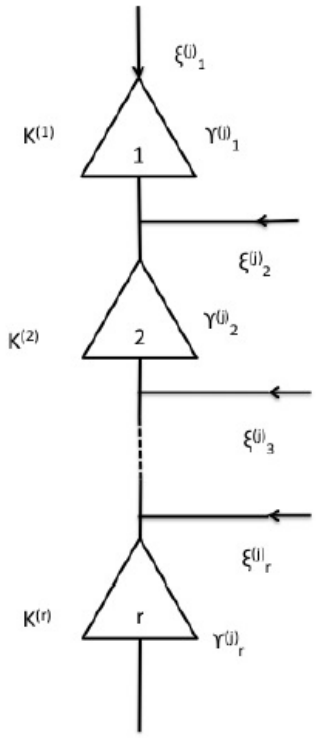


Figure 6.3: Topology of main river, the side rivers, and the possible reservoir sites

Let us introduce the following notation:

r	number of sites;
k	number of consecutive periods that demand will be satisfied
n	number of periods;
$K^{(j)}$	unknown capacity of reservoir j;
$\zeta_i^{(j)}$	water content in reservoir j at end of i^{th} period;
$\xi_i^{(j)}$	direct random inflow into reservoir j in the i^{th} period;
$\eta_i^{(j)}$	direct random demand against reservoir j in the i^{th} period;
$x_i^{(j)}$	1 if j^{th} reservoir satisfies the demand in the i^{th} period, 0 otherwise

We also introduce auxiliary random variables as follows:

$$
\begin{aligned}
& g_j^{(0)}(\mathbf{x}) = 0 \\
& g_j^{(i)}(\mathbf{x}) = \zeta_{j-1}^{(i)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)} - \min\{\zeta_{j-1}^{(i)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)}, K^{(i)}\} \\
& h_j^{(i)}(\mathbf{x}) = \min\{\zeta_j^{(i-1)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)}, K^{(i)}\} \\
& d_j^{(i)}(\mathbf{x}) = h_j^{(i)}(\mathbf{x}) - \eta_j^{(i)} \\
& i = 1, \ldots, r, \qquad j = 1, \ldots, n
\end{aligned}
\tag{6.7}
$$

Quantity $g_j^{(i)}$ is the amount of water that overflowed at reservoir i, and $h_j^{(i)}$ is the amount of water remaining in reservoir i at the beginning of period j when the input water fills up the reservoirs and demand does not yet occur. Using these we write

$$
\begin{aligned}
& \zeta_j^{(1)}(\mathbf{x}) = \min\{d_j^{(1)}(\mathbf{x}), d_j^{(1)}(\mathbf{x}) + d_j^{(2)}(\mathbf{x}), \ldots, d_j^{(1)}(\mathbf{x}) + d_j^{(2)}(\mathbf{x}) + \ldots + d_j^{(r)}(\mathbf{x})\} \\
& \zeta_j^{(i)}(\mathbf{x}) = \max\{0, \min\{d_j^{(i)}(\mathbf{x}), d_j^{(i)}(\mathbf{x}) + d_j^{(i+1)}(\mathbf{x}), \ldots, d_j^{(i)}(\mathbf{x}) + d_j^{(i+1)}(\mathbf{x}) + \ldots + d_j^{(r)}(\mathbf{x})\}\}
\end{aligned}
\tag{6.8}
$$

Our problem, to find optimal reservoir capacities, is the following:

$$
\begin{aligned}
& \min c_1(K^{(1)}) + c_2(K^{(2)}) + \ldots + c_r(K^{(r)}) \\
& \text{subject to } P\{\forall j \forall \mathbf{x} \in X : \zeta_j^{(1)}(\mathbf{x}) \geq 0\} \geq p \\
& X = \left\{ \begin{array}{l} (x_1, \ldots, x_n) \text{ s.t.} \\ x_1 + \cdots + x_k \leq k - 1 \\ x_2 + \cdots + x_{k+1} \leq k - 1 \\ \vdots \\ x_{n-k+1} + \cdots + x_n \leq k - 1 \end{array} \right\}
\end{aligned}
\tag{6.9}
$$

where p is a prescribed probability, near 1 in practice.

Theorem 6.1 *For any fixed $x_i^{(j)}$, $i = 1, \ldots, n$, $j = 1, \ldots, r$, the probability, standing on the left hand side in the probabilistic constraint (1), is a logconcave function of $(K^{(1)}, \ldots, K^{(r)})$.*

Proof Inside the paranthesis of the probability in (1) we have inequalities that we imagine reformulated in such a way that as 0 remains on the right hand sides of the inequalities. Then, we have n inequalities which contain functions of the variables $(K^{(1)}, \ldots, K^{(r)})$ and $(\xi_1^{(j)}, \ldots, \xi_n^{(j)}, \gamma_1^{(j)}, \ldots, \gamma_n^{(j)})$, and these functions are concave. The application of Theorem 1 for this case proves Theorem 2. ■

6.5.1 Simplification of the problem

The set X in (6.9) represents a list of possible scenarios for the sequence of dry and rainy days. This set can be very large, which means that even verifying if the probabilistic constraint is satisfied for a particular realization of the random variables could be computationally intensive. In this section we reduce the set X to a smaller set of worst-case scenarios.

Lemma 6.1 *Let $\tilde{X} \subseteq X$ be such that*

$$\forall \mathbf{x} \in X : \exists \tilde{\mathbf{x}} \in \tilde{X} : \tilde{\mathbf{x}} \geq \mathbf{x} \tag{6.10}$$

where the inequality is component-wise. Then

$$\left(\forall \mathbf{x} \in X : \forall j : \zeta_j^{(1)}(\mathbf{x}) \geq 0\right) \Leftrightarrow \left(\forall \mathbf{x} \in \tilde{X} : \forall j : \zeta_j^{(1)}(\mathbf{x}) \geq 0\right) \tag{6.11}$$

Proof Use induction to show that $\zeta_j^{(1)}(\mathbf{x})$ is monotonously increasing with $\mathbf{x}$. ■

Lemma 6.1 shows that the set X in (6.9) can be substituted by the set of its *maximal* points, that is the points such that every point in X is less-than-or-equal-to a maximal point, but there is no other point in X which is greater-than-or-equal-to a maximal point.

The set of maximal points (it is characterized in Lemma 6.2 below) is best characterized in terms of the lengths of periods of consecutive dry days which we call droughts. For a given $\mathbf{x} \in X$ we denote $l_1, l_2, \ldots, l_s$ the lengths of droughts in scenario $\mathbf{x}$. More formally,

$$\begin{aligned} & x_1 = x_2 = \ldots = x_{l_1} = 1 \\ & x_{l_1+1} = 0 \\ & x_{l_1+2} = x_{l_1+3} = \ldots = x_{l_1+l_2+1} = 1 \\ & x_{l_1+l_2+2} = 0 \\ & \ldots \end{aligned} \tag{6.12}$$

In the case when two consecutive days are rainy ($x_i = 0$, $x_{i+1} = 0$) the respective $l_j = 0$.

Denote $l(\mathbf{x})$ the representation of scenario $\mathbf{x}$ through the drought lengths $(l_1, \ldots, l_s)$. Then the set X can be represented as $X = \{\mathbf{x} : l(\mathbf{x}) \in L\}$ where $L = \{(l_1, \ldots, l_s) : l_i \leq k-1, \sum_i l_i + s = n\}$

Lemma 6.2 *Let $X_{\max}$ be the set of maximal points of X. Then*

$$X_{\max} = \{\mathbf{x} : l(\mathbf{x}) \in L_{\max}\} \tag{6.13}$$

where

$$L_{\max} = \{(l_1, \ldots, l_s) : l_i \leq k-1, l_i + l_{i+1} \geq k-1, \sum_i l_i + s = n\} \tag{6.14}$$

Proof A vector $\mathbf{x}$ is *not* maximal if for one of its coordinates we have $x_r^{(j)} = 0$ and the vector $\mathbf{x}' = (x_1^{(j)}, \ldots, x_{r-1}^{(j)}, 1, x_{r+1}^{(j)}, \ldots, x_n^{(j)})$ also satisfies the constraints defining the set X in (6.9). This only happens if $\forall r_0$ such that $r-k+1 \leq r_0 \leq r$ we have $x_{r_0}^{(j)} + \ldots + x_{r_0+k-1}^{(j)} < k-1$. In other words, for each sequence of coordinates $x_{r_0}^{(j)}, \ldots, x_{r_0+k-1}^{(j)}$ there is at least one zero coordinate besides $x_r^{(j)}$. A vector $\mathbf{x}^{(j)}$ is maximal if there is no such coordinate $x_r^{(j)}$, that is

$$\forall r \in \overline{1,n} : x_r^{(j)} = 0 \Rightarrow \exists r_0, r-k+1 \leq r_0 \leq r : x_{r_0}^{(j)} + \ldots + x_{r_0+k-1}^{(j)} = k-1 \tag{6.15}$$

We can restate the condition (6.15): vector $\mathbf{x}$ is maximal if $\forall i : l_i + l_{i+1} \geq k-1$. Notice, that $\mathbf{x} \in X$ if and only if $\forall i : l_i \leq k-1$. Thus, all maximal vectors $\mathbf{x}$ can be enumerated by listing all sequences of nonnegative integers $l_1, \ldots, l_s$ satisfying:

$$\begin{aligned} & l_i \leq k-1, \quad i = 1, \ldots, s \\ & l_i + l_{i+1} \geq k-1, \quad i = 1, \ldots, s-1 \\ & \textstyle\sum_{i=1}^{s} l_i + s = n \end{aligned} \tag{6.16}$$

■

The sequences $\{l_i\}$ in (6.16) can be enumerated directly with, for example, depth-first search algorithm.

6.5.2 Numerical Results

Here we present 2-reservoir network problem which are serially linked to each other. The total number of periods that is taken into account is 24 days and thus we would like to satisfy the demand for 8 consecutive days out of 24 days for both reservoirs. The cost function for the reservoirs is convex with coefficients of 1.5 for $C^{(1)}$ and 1 for $C^{(2)}$.

We assumed the inflow values and the demand are random coming from normal distribution. We further assumed that demand for different days are dependent to each other

Table 6.1: Results

	DFIPA algorithm	Supporting hyperplane
K_1 Capacity (1000 m^3)	15.0474	15.06498
K_2 Capacity (1000 m^3)	5.7323	5.7687
Optimal Cost	28.3034	28.3662
Number of Iterations	56	8
Computation time	12572	1373

with the covariance matrices for both reservoirs that are available in Appendix A. Inflow variables are also dependent within days and covariance matrices for both reservoirs are also available in Appe The data for the distribution of inflow and demand for each reservoirs that is used for this problem can be reached in Appendix B.

Since there are 24 days, we have 17 0-1 constraints to be satisfied for the first reservoir and 17 0-1 constraints to be satisfied for the second reservoir in addition to demand- inflow feasibility constraints. If we consider all the possible $x_i^{(j)}$ pairs that satisfies the 34×8 system of inequalities, there is going to be 2^{34} total feasible vectors that need to be used to be able to achieve the optimal solution. With the simlpe elimination technique described in Section 6.5.1, we were able to reduce the set of feasible vectors to 254. After that, we used derivative-free inner polygone approximation method that is coded in Matlab to obtain the optimum solution while satisfying the probabilistic constraints.

The results and comparison of the two methods can be found in the table 6.1 above. The results are slightly different because of the difference in the precision of the methods that we use to generate the multi-variate distributions.

Chapter 7

Conclusion

We proposed different solution methodologies for several important classes of probabilistic constrained problems. In particular, we concentrated on the probabilistic constrained problems with degenerate distribution.

As a result of conducted research, not only were we able to perform a qualitative and quantitative analysis and comparison of the classical methods, but also came up with several new efficient algorithms. These include the hybrid algorithm inspired by supporting hyperplane and PVB algorithms, as well as totally different in nature class of derivative-free algorithms inspired by direct search methods.

As another important result of this analysis, a large functional library of Matlab codes was created and tested (all source code is provided in the Appendices and will be uploaded online as an open source for the scholars interested in further developing this topic).

Our solution techniques are novel in the sense that they respond to different objectives such as precision, complexity or simulation. One can refer to supporting hyperplane method with approximate derivative to obtain the best precision and fast results. Not only we offer a better precision and a faster way to solve the considered problems, we also take into account the real life restrictions and applications for each. Our approaches can also be applied to any type of reliability theory applications such as finance, energy, communication, traffic system reliability problems.

References

[1] Archibald T. W., K.I.M. McKinnon, L.C. Thomas 1997. An aggregate stochastic dynamic programming model of multireservoir systems. *Water Resources Research* **33**, 333–340.

[2] Avis, D., and Fukuda, K. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete and Computational Geometry 8*, 1 (1992), 295–313.

[3] Balas E. Disjunctive programming: cutting planes from logical conditions. *Nonlinear Programming 2*, (O.L. Mangasarian, R. R. Meyer, S.M.Robinoson, eds.), 279-312, 1975.

[4] Billington, R., Bhavaraju, M.P. Loss of load probabiity approach to the evaluation of generating capacity capacity probability of two interconnected systems. *Transactions of Canadian Electrical Associations* , 1967

[5] Charnes, A., Cooper, W.W., Symonds, G.H. Cost Horizons and Certainty Equivalents: An Approach to Stochastic Programming of Heating Oil. April 1, 1958

[6] Cheney, E. W., and A. A. Goldstein 1959. Newton's Method for Convex Programming and Tchebyche Approximation. *Numerische Mathematik* **1**, 253–268.

[7] Colorni A., and G. Fronza, 1976. Reservoir management via reliability programming *Water Resources Research*, **12 (1)**, 85–88.

[8] Dennis, J. E. Jr., and Torczon, V. Direct search methods on parallel machines. *SIAM Journal on Optimization 1*, 4 (November 1991), 448–474.

[9] Diaconis, P. The markov chain monte carlo revolution. *Bulletin of the American Mathematical Society* (2009), 179–205.

[10] Dyer, M. E., and Frieze, A. M. On the complexity of computing the volume of a polyhedron. *SIAM J. Comput. 17*, 5 (Oct. 1988), 967–974.

[11] Fanelli, S., Prékopa, A. The probability of a feasible flow in a stochastic transportation network *ISBN-13-9781249891383*, 2009.

[12] Ford, Jr, .R., Fulkerson, D.R. Flows in Networks. *Princeton University Press*, Princeton, NJ, 1962.

[13] Foufoula, E. and P.K.Georgiou Kitanidis 1988. Gradient dynamic programming for stochas- tic optimal control of multidimensional water resources systems, *Water Resources Research* **24**, 1345-1359.

[14] Gal, S. 1979. Optimal management of a multireservoir water supply system, *Water Resources Research* **15** 737-749.

[15] Gale, D., A theorem on flows in network. *Pacific J. Math*, 7, 1073-1083, 1957.

[16] Genz, A. , Kwong, K. Numerical Evaluation of Singular Multivariate Normal Distributions. *Journal of Statistical Computation and Simulation*, 1999

[17] Hoffman, A.J., Some recent applications of the theory of linear inequalities to external combinatorial analysis. *Procedings of Symposia in Applied Mathematics*, Combinatorial Analysis American Mathematical Society, 113-127, 1960

[18] Houck, M. H., 1979. A chance-constrained optimization model for reservoir design and operation, *Water Resources Research*, **15 (5)**, 1011-1016.

[19] Houck,M. H., and B . Datta, 1981. Performance evaluation of a stochastic optimization model of for reservoir design and management with explicit reliability criteria *Water Resources Research*, **17 (4)** 827–832.

[20] Huang, G. H., Daniel Peter Loucks. 2000. An inexact two-stage stochastic programming model for water resources management under uncertainty. *Civil Engineering and Environmental Systems* **17 (2)** 95–118.

[21] Jerzy B. Strycharczyk, Jery R. Stedinger 1987. Evaluation of a "reliability programming" reservoir model *Water Resources Research* **23 (2)** 225–229.

[22] Joeres, E. F., G. J. Seus, and H. M. Engelmann, 1981. The linear decision rule (LDR) reservoir problem with correlated inflows, Model development *Water Resource Research*, **7 (1)**, 18–24.

[23] Joeres E. F., ,J .C . Liebmaann, C. S. ReVelle, 1971. Operating rules for joint operation of raw water resources *Water Resources Research*, **7 (2)** 225– 235.

[24] Kelley, J. E. 1960. The Cutting Plane Method for Solving Convex Programs. *SIAM Journal of Applied Mathematics* **11**, 703–712.

[25] Kirby, W., C. S. ReVelle, and E. Joeres, 1970. Reply, *Water Resources Research*,**6 (4)**, 1242–1245.

[26] Lamond, B.F. and M.J. Sobel 1995. Exact and approximate solutions of affine reservoirs models, *Operations Research* **43** 771-780.

[27] Lawrence, J. Polytope volume computation. *Mathematics of computation 57* (1991), 259–271.

[28] Loucks, D. P. 1969. Stochastic methods for analyzing river basin systems, research project technical completion report, OWRR project C-1034, Dep. of Water Resource Eng. Cornell Univ., Ithaca, N.Y.

[29] Loucks, D. P., 1970. Some comments on linear decision rules and chance constraints, *Water Resources Research*, **6 (2)** 668–671.

[30] Loucks, D. P., and P. J. Dorfman, 1975. An evaluation of some linear decision rules in chance-constrained models for reservoir planning and operation, *Water Resources Research*, **11 (6)** ,777-782.

[31] Marino, M. A., and B. Mohammadi,1983. Reservoir management: A reliability programming approach, *Water Resources Research*, **19 (3)**, 613–620.

[32] Moran, P. A. P. 1959. The theory of storage, Methuen, London.

[33] Nelder, J. A., and Mead, R. A simplex method for function minimization. *The computer journal 7*, 4 (1965), 308–313.

[34] Prékopa, A. On Probabilistic Constrained Programming. Proceedings of the Princeton Symposium on Mathematical Programming. *Princeton University Press*, Princeton, NJ, 113-138, 1970

[35] Prékopa, A. 1971 Logarithmic Concave Measures with Application to Stochastic Programming. *Acta Sci. Math., Szeged* **32**, 301-316.

[36] Prékopa, A., T.Rapcsák, I.Zsuffa, 1978 Serially Linked Reservoir System Design Using Stochastic Programming. *National Water Authority of Hungary* **14, 4**, 672–678.

[37] Prékopa ,A. 1978 Logarithmic Concave Measures and Related Topics. *Proceedings of the International Conference on Stochastic Programming, Oxford, 1974* , New York, 1978.

[38] Prékopa, A., Network planning using two-stage programming under uncertainty. *MRC Technical Summary Report* , 1842, 1978

[39] Prékopa, A., Rapcsák, T., Zsuffa, I. Serially linked reservoir system design using stochastic programming. *Water Resources Research*, 14, 672-678, 1978

[40] Prékopa, A., Szántai, T., A new multivariate gamma distribution and its fitting to empirical steamflow data. *Water Resources Research*, 14, 19-24, 1978

[41] Prékopa, A., Szántai, T., Flood control reservoir system design using stochastic programming. *Mathematical Programming Study* , 9, 138-151, 1978

[42] Prékopa, A., Network Planning using two-stage programming under uncertainty. *Recent Results in Stochastic Programming*, Lecture Notes in Economics and Mathematical Systems 179, (Springer, Berlin), 216-237, 1980.

[43] Prékopa, A., Boros, E. On the existance of a feasible flow in a stochastic transportation network. *Operations Research* 29, 119-129, 1989

[44] Prékopa, A. Dual Method for a One-Stage Stochastic Programming Problem with Random RHS, Obeying a Discrete Probability Distribution. *Zeitschrift fr Operations Research.* 34, 441-461 1990

[45] Prékopa, A., Boros, E., On existance of a feasible flow in a stochastic transportation network. *Operations Research*, 39, 119-129, 1991

[46] Prékopa, A., Stochastic Programming, *Kluwer Scientific Publishers*, Boston, 1995

[47] Prékopa, A., Vizvári, B., Badics, T., Programming under probabilistic constraint with discrete random variable. *New Trends in Mathematical Programming (F. Giannessi, T. Rapsack, eds.)* , Kluwer Scientific Publishers, Boston, 235-255, 1998

[48] Prékopa, A., Probabilistic Programming, *Stochastic Programming Handbooks in OR and MS*, Vol.10, Elsevier, 269-351, 2003

[49] Prékopa, A., On the relationship between probabilistic constrained, disjunctive and multiobjective programming. *RUTCOR Research Report*, 07-2007, 2007

[50] Prékopa, A., Multivariate value at risk and related topics. *Annals of Operations Research*, 193, 49-69, 2012

[51] Prékopa, A., Mayer, J., Strazicky, B., Deak, I., Hoffer, J., Németh, Á., Scheduling of power generation. Springer, New York, 2014

[52] Prékopa, A., Unuvar, M. Single commodity stochastic network design under probabilistic constraint with discrete random variables. *Operations Research*, 2015

[53] Prékopa, A., Myndyuk, O., Szántai, T., On a hybrid method for the solution os the probabilistic constrained stochastic programming problem with continuously distributed random variables. Manuscript, 2016

[54] ReVelle, C. S., E. Joeres, and W. Kirby, 1969. The linear decision rule in reservoir management and design, Development of stochastic model, *Water Resources Research*,**5 (4)**, 767–777.

[55] ReVelle, C. S., and J. Gundelach, 1975. Linear decision rule in reservoir management and design, A rule that minimizes output variance, *Water Resources Research*, **11 (2)**, 197–203.

[56] Simonovic, S., and M. A. Marino, 1980. Reliability programming in reservoir management, Single multipurpose reservoir, *Water Resources Research*,**16 (5)**, 844–848.

[57] Simonovic, S. P., and M. A. Marino, 1981. The role of unconditioned and conditioned cumulative distribution functions in reservoir reliability programming, *J. Hydrol.*, **52 (3/4)**, 219–228.

[58] Simonovic, S. P., and G. T. Orlob, 1984. Risk-reliability programming for optimal water quality control, *Water Resources Research*, **20 (6)**, 639–646.

[59] Simonovic, S., 1979. Two-step algorithm for design-stage long-term control of a multipurpose reservoir, *Advanced Water Resources*, **2**, 47–49.

[60] Spears, H.T., Hicks, K.L., Lee, S.T, Probability of loss of load for three areas. *IEEE Transaction* , PAS-89, 521-527, 1970

[61] Stedinger, Jery Russell, B. F. Sule, Daniel Peter Loucks. 1984. Stochastic Dynamic-Programming Models for Reservoir Operation Optimization. *Water Resources Research* **20 (11)** 1499–1505.

[62] Stedinger, J. R., B. Sule, and D. Pei, 1983. Multiple-reservoir system screening models,*Water Resources Research*, **19 (6)**,1383–1393.

[63] Szántai, T., A computer code of probabilistic constrained stochastic programming problems. *Numerical Technologies for Stochastic Optimization (Ermoliev, Y., Wets, R.J.-B., eds.)* , Springer, New York, 229-235, 1988

[64] Thapalia, B.K., Crainic, T.G., Kaut, M., Wallace, S.W., Single commodity stochastic network design with multiple sources and sinks. *Interuniversity Research Center on Enterprise Networks* , Logistics and Transportation, CIRRELT-2010-59, 2010

[65] Thapalia, B.K., Kaut, M., Wallace, S.W., Crainic, T.G., Single source single-commodity stochastic network design. *Computational Management Science* , Springer, Berlin, 1619, 2010

[66] Torczon, V. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Department of Mathematical Sciences, Rice University, Houston, Texas, May 1989.

[67] Turgeon, A. 1981. A decomposition method for the long-term scheduling of reservoirs in series, *Water Resources Research* **17** 1565-1570.

[68] Valiant, L. G. The complexity of enumeration and reliability problems. *SIAM J. Comput. 8*, 3 (1979), 410–421.

[69] Veinott, A. F. 1967. The Supporting Hyperplane Method for Uni- modal Programming.*Operations Research* **15**, 147–152.

[70] Wllace, S.W., Wets, R.G.-B. The facets of the polyhedral set determined by Gale-Hoffman inequalities. *Mathematical Programming* ,62, 215-222, 1993

[71] Wright, M. H. Direct search methods: once scorned now respectable. In *Numerical Analysis 1995 (Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis)* (Harlow, United Kingdom, 1995), D. F. Griffiths and G. A. Watson, Eds., Addisson Wesley Longman, pp. 191 – 208.

[72] Yakowitz, S. 1982. Dynamic programming applications in water resources, *Water Resources Research* **18** 673-696.

Appendix A

Inequalities left-hand sides after elimination by topology (161 out of 255)

The Gale-Hoffman inequalities for the 8-node power distribution problem have the form

$$\begin{aligned} \xi_i &\leq x_i + \sum_{j \in N \setminus \{i\}} y_{ji}, i \in N \\ \sum_{i \in I_k} \xi_i &\leq \sum_{i \in I_k} x_i + \sum_{i \in I_k, j \in N \setminus I_k} y_{ji}, k = 1, ..., l, \end{aligned} \tag{A.1}$$

where the sets I_k, after elimination by topology, are as follows:

$\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, $\{7\}$, $\{8\}$, $\{1,2\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$, $\{2,4\}$, $\{1,2,4\}$, $\{2,3,4\}$, $\{1,2,3,4\}$, $\{2,5\}$, $\{1,2,5\}$, $\{3,5\}$, $\{1,3,5\}$, $\{2,3,5\}$, $\{1,2,3,5\}$, $\{4,5\}$, $\{2,4,5\}$, $\{1,2,4,5\}$, $\{3,4,5\}$, $\{1,3,4,5\}$, $\{2,3,4,5\}$, $\{1,2,3,4,5\}$, $\{4,6\}$, $\{2,4,6\}$, $\{1,2,4,6\}$, $\{2,3,4,6\}$, $\{1,2,3,4,6\}$, $\{5,6\}$, $\{2,5,6\}$, $\{1,2,5,6\}$, $\{3,5,6\}$, $\{1,3,5,6\}$, $\{2,3,5,6\}$, $\{1,2,3,5,6\}$, $\{4,5,6\}$, $\{2,4,5,6\}$, $\{1,2,4,5,6\}$, $\{3,4,5,6\}$, $\{1,3,4,5,6\}$, $\{2,3,4,5,6\}$, $\{1,2,3,4,5,6\}$, $\{5,7\}$, $\{2,5,7\}$, $\{1,2,5,7\}$, $\{3,5,7\}$, $\{1,3,5,7\}$, $\{2,3,5,7\}$, $\{1,2,3,5,7\}$, $\{4,5,7\}$, $\{2,4,5,7\}$, $\{1,2,4,5,7\}$, $\{3,4,5,7\}$, $\{1,3,4,5,7\}$, $\{2,3,4,5,7\}$, $\{1,2,3,4,5,7\}$, $\{6,7\}$, $\{4,6,7\}$, $\{2,4,6,7\}$, $\{1,2,4,6,7\}$, $\{2,3,4,6,7\}$, $\{1,2,3,4,6,7\}$, $\{5,6,7\}$, $\{2,5,6,7\}$, $\{1,2,5,6,7\}$, $\{3,5,6,7\}$, $\{1,3,5,6,7\}$, $\{2,3,5,6,7\}$, $\{1,2,3,5,6,7\}$, $\{4,5,6,7\}$, $\{2,4,5,6,7\}$, $\{1,2,4,5,6,7\}$, $\{3,4,5,6,7\}$, $\{1,3,4,5,6,7\}$, $\{2,3,4,5,6,7\}$, $\{1,2,3,4,5,6,7\}$, $\{2,8\}$, $\{1,2,8\}$, $\{2,3,8\}$, $\{1,2,3,8\}$, $\{4,8\}$, $\{2,4,8\}$, $\{1,2,4,8\}$, $\{2,3,4,8\}$, $\{1,2,3,4,8\}$, $\{2,5,8\}$, $\{1,2,5,8\}$, $\{2,3,5,8\}$, $\{1,2,3,5,8\}$, $\{4,5,8\}$, $\{2,4,5,8\}$, $\{1,2,4,5,8\}$, $\{3,4,5,8\}$, $\{1,3,4,5,8\}$, $\{2,3,4,5,8\}$, $\{1,2,3,4,5,8\}$, $\{6,8\}$, $\{2,6,8\}$, $\{1,2,6,8\}$, $\{2,3,6,8\}$, $\{1,2,3,6,8\}$, $\{4,6,8\}$, $\{2,4,6,8\}$, $\{1,2,4,6,8\}$, $\{2,3,4,6,8\}$, $\{1,2,3,4,6,8\}$, $\{5,6,8\}$, $\{2,5,6,8\}$, $\{1,2,5,6,8\}$, $\{3,5,6,8\}$, $\{1,3,5,6,8\}$, $\{2,3,5,6,8\}$, $\{1,2,3,5,6,8\}$, $\{4,5,6,8\}$, $\{2,4,5,6,8\}$, $\{1,2,4,5,6,8\}$, $\{3,4,5,6,8\}$, $\{1,3,4,5,6,8\}$, $\{2,3,4,5,6,8\}$, $\{1,2,3,4,5,6,8\}$, $\{2,5,7,8\}$, $\{1,2,5,7,8\}$, $\{2,3,5,7,8\}$, $\{1,2,3,5,7,8\}$, $\{4,5,7,8\}$, $\{2,4,5,7,8\}$, $\{1,2,4,5,7,8\}$, $\{3,4,5,7,8\}$, $\{1,3,4,5,7,8\}$, $\{2,3,4,5,7,8\}$,

$\{1,2,3,4,5,7,8\}$, $\{6,7,8\}$, $\{2,6,7,8\}$, $\{1,2,6,7,8\}$, $\{2,3,6,7,8\}$, $\{1,2,3,6,7,8\}$, $\{4,6,7,8\}$, $\{2,4,6,7,8\}$, $\{1,2,4,6,7,8\}$, $\{2,3,4,6,7,8\}$, $\{1,2,3,4,6,7,8\}$, $\{5,6,7,8\}$, $\{2,5,6,7,8\}$, $\{1,2,5,6,7,8\}$, $\{3,5,6,7,8\}$, $\{1,3,5,6,7,8\}$, $\{2,3,5,6,7,8\}$, $\{1,2,3,5,6,7,8\}$, $\{4,5,6,7,8\}$, $\{2,4,5,6,7,8\}$, $\{1,2,4,5,6,7,8\}$, $\{3,4,5,6,7,8\}$, $\{1,3,4,5,6,7,8\}$, $\{2,3,4,5,6,7,8\}$, $\{1,2,3,4,5,6,7,8\}$

Appendix B

Covariance Matrices Between Days for Demand and Inflow variables for Reservoir 1 and Reservoir 2

$R^{(j)}_{Demand}$	Covariance matrix for demand of reservoir j
$R^{(j)}_{Inflow}$	Covariance matrix for inflow of reservoir j
	where $j = 1, 2$

$$\mathcal{R}^{(\infty)}_{\mathcal{D}\rceil\Updownarrow\dashv\backslash\lceil} = \begin{bmatrix}
1.00 & 0.28 & 0.00 \\
0.28 & 1.00 & 0.20 & 0.00 \\
0.00 & 0.20 & 1.00 & 0.01 & 0.00 \\
0.00 & 0.00 & 0.01 & 1.00 & 0.14 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.14 & 1.00 & 0.21 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.21 & 1.00 & 0.12 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.12 & 1.00 & 0.27 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.27 & 1.00 & 0.31 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.31 & 1.00 & 0.36 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.36 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.23 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.23 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 1.00 & 0.22 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.22 & 1.00 & 0.04 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.03 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 1.00 & 0.02 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 1.00 & 0.10 & 0.00 & 0.00 \\
0.00 & 0.10 & 1.00 & 0.11 & 0.00 \\
0.00 & 0.11 & 1.00 & 0.69 \\
0.00 & 0.69 & 1.00
\end{bmatrix}$$

$$\mathcal{R}^{(\in)}_{\mathcal{D}\rceil\Updownarrow\dashv\backslash\lceil} = \begin{bmatrix}
1.00 & 0.29 & 0.00 \\
0.29 & 1.00 & 0.25 & 0.00 \\
0.00 & 0.25 & 1.00 & 0.14 & 0.00 \\
0.00 & 0.00 & 0.14 & 1.00 & 0.14 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.14 & 1.00 & 0.20 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.20 & 1.00 & 0.15 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.15 & 1.00 & 0.36 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.36 & 1.00 & 0.37 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.37 & 1.00 & 0.12 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.12 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.47 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.47 & 1.00 & 0.09 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.09 & 1.00 & 0.09 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.09 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.09 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.09 & 1.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 1.00 & 0.08 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.08 & 1.00 & 0.44 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.44 & 1.00 & 0.28 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.28 & 1.00 & 0.03 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 1.00 & 0.07 & 0.00 & 0.00 \\
0.00 & 0.07 & 1.00 & 0.18 & 0.00 \\
0.00 & 0.18 & 1.00 & 0.12 \\
0.00 & 0.12 & 1.00
\end{bmatrix}$$

$$\mathcal{R}^{(\infty)}_{\mathcal{I}\setminus\{\updownarrow,\sqsupseteq\}} = \begin{bmatrix}
1.00 & 0.48 & 0.00 \\
0.48 & 1.00 & 0.38 & 0.00 \\
0.00 & 0.38 & 1.00 & 0.32 & 0.00 \\
0.00 & 0.00 & 0.32 & 1.00 & 0.16 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.16 & 1.00 & 0.26 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.26 & 1.00 & 0.23 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.23 & 1.00 & 0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 1.00 & 0.26 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.26 & 1.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 1.00 & 0.42 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.42 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.07 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.07 & 1.00 & 0.26 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.26 & 1.00 & 0.32 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.32 & 1.00 & 0.12 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.12 & 1.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.45 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.45 & 1.00 & 0.12 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.12 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 1.00 & 0.07 & 0.00 & 0.00 \\
0.00 & 0.07 & 1.00 & 0.17 & 0.00 \\
0.00 & 0.17 & 1.00 & 0.00 \\
0.00 & 1.00
\end{bmatrix}$$

$$\mathcal{R}^{(\in)}_{\mathcal{I}\setminus\{\updownarrow\},\sqsupseteq} = \begin{bmatrix}
1.00 & 0.21 & 0.00 \\
0.21 & 1.00 & 0.09 & 0.00 \\
0.00 & 0.09 & 1.00 & 0.07 & 0.00 \\
0.00 & 0.00 & 0.07 & 1.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.01 & 1.00 & 0.06 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.06 & 1.00 & 0.11 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.11 & 1.00 & 0.08 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.08 & 1.00 & 0.16 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.16 & 1.00 & 0.25 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 1.00 & 0.23 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.23 & 1.00 & 0.23 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.23 & 1.00 & 0.09 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.09 & 1.00 & 0.16 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.16 & 1.00 & 0.19 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.19 & 1.00 & 0.03 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 1.00 & 0.03 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & 1.00 & 0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.02 & 1.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.18 & 1.00 & 0.11 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.11 & 1.00 & 0.04 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.04 & 1.00 & 0.15 & 0.00 & 0.00 \\
0.00 & 0.15 & 1.00 & 0.18 & 0.00 \\
0.00 & 0.18 & 1.00 & 0.03 \\
0.00 & 0.03 & 1.00
\end{bmatrix}$$

Appendix C

Distribution of Demand and Inflow Values for Reservoir 1 and Reservoir 2

Table C.1: Demand η Distributions

	RESERVOIR 1		RESERVOIR 2	
	Demand ($\eta_i^{(1)}$) Distribution		Demand ($\eta_i^{(2)}$) Distribution	
Day (i)	Mean 1000 m^3	Standard Deviation 1000 m^3	Mean 1000 m^3	Standard Deviation 1000 m^3
1	2.544	0.2544	1.564	0.1564
2	4.778	0.4778	3.895	0.3895
3	6.4	0.64	5.89	0.589
4	6.481	0.6481	4.763	0.4763
5	7.607	0.7607	5.92	0.592
6	5.933	0.5933	4.327	0.4327
7	4.993	0.4993	3.123	0.3123
8	7.117	0.7117	5.983	0.5983
9	4.67	0.467	3.012	0.3012
10	7.554	0.7554	2.074	0.2074
11	11.167	1.1167	6.93	0.693
12	5.87	0.587	3.21	0.321
13	4.012	0.4012	3.923	0.3923
14	4.014	0.4014	3.217	0.3217
15	4.205	0.4205	3.214	0.3214
16	5.985	0.5985	3.97	0.397
17	6.345	0.6345	5.936	0.5936
18	12.54	1.254	5.32	0.532
19	3.987	0.3987	1.784	0.1784
20	4.231	0.4231	3.984	0.3984
21	4.094	0.4094	3.758	0.3758
22	4.875	0.4875	4.345	0.4345
23	5.73	0.573	2.937	0.2937
24	2.98	0.298	1.984	0.1984

Table C.2: Inflow η Distributions

	RESERVOIR 1 Inflow ($\eta_i^{(1)}$) Distribution		RESERVOIR 2 Inflow ($\eta_i^{(2)}$) Distribution	
Day (i)	Mean 1000 m^3	Standard Deviation 1000 m^3	Mean 1000 m^3	Standard Deviation 1000 m^3
1	6.023	0.6023	3.785	0.3785
2	9.855	0.9855	5.463	0.5463
3	6.778	0.6778	4.739	0.4739
4	9.342	0.9342	5.263	0.5263
5	10.583	1.0583	5.307	0.5307
6	8.521	0.8521	4.851	0.4851
7	6.721	0.6721	3.987	0.3987
8	6.844	0.6844	4.632	0.4632
9	8.9	0.89	5.933	0.5933
10	6.869	0.6869	3.914	0.3914
11	10.229	1.0229	5.743	0.5743
12	5.498	0.5498	3.894	0.3894
13	6.03	0.603	4.62	0.462
14	8.091	0.8091	5.938	0.5938
15	5.53	0.553	4.422	0.4422
16	6.482	0.6482	4.695	0.4695
17	6.437	0.6437	4.909	0.4909
18	13.22	1.322	5.095	0.5095
19	4.4	0.44	3.695	0.3695
20	5.34	0.534	4.56	0.456
21	5.78	0.578	4.234	0.4234
22	6.321	0.6321	5.442	0.5442
23	4.98	0.498	3.986	0.3986
24	5.01	0.501	3.724	0.3724

Appendix D

Derivation of problem (6.9) in the form of (2.1)

To reduce problem (6.9) to the form (2.1) we show that

Lemma D.1 *The set*

$$\{(\mathbf{K},\xi,\eta)|\forall j\forall \mathbf{x}\in X:\zeta_j^{(1)}(\mathbf{x})\geq 0\} \tag{D.1}$$

where

$$\mathbf{K}=(K^{(1)},\ldots,K^{(r)}),\quad \xi=(\xi_1^{(1)},\ldots,\xi_1^{(r)},\ldots,\xi_n^{(1)},\ldots,\xi_n^{(r)}), \tag{D.2}$$

$$\eta=(\eta_1^{(1)},\ldots,\eta_1^{(r)},\ldots,\eta_n^{(1)},\ldots,\eta_n^{(r)}) \tag{D.3}$$

can be represented as

$$\begin{aligned}
&\forall \mathbf{x}\,\exists \tilde{g}_j^{(i)},\tilde{f}_j^{(i)}:\\
&\tilde{g}_j^{(0)}=0\\
&\tilde{g}_j^{(i)}\geq 0\\
&\tilde{h}_j^{(i)}=\tilde{\zeta}_{j-1}^{(i)}+\tilde{g}_j^{(i-1)}+\xi_j^{(i)}-\tilde{g}_j^{(i)}\\
&\tilde{h}_j^{(i)}\leq K^{(i)}\\
&\tilde{h}_j^{(i)}\geq 0\\
&\tilde{f}_j^{(0)}=0\\
&\tilde{f}_j^{(i)}\geq 0\\
&\tilde{\zeta}_j^{(i)}=\tilde{h}_j^{(i)}+\tilde{f}_j^{(i-1)}-\tilde{f}_j^{(i)}-x_j\eta_j^{(i)}\\
&\tilde{\zeta}_0^{(i)}=0\\
&\tilde{\zeta}_j^{(i)}\leq K^{(i)}\\
&\tilde{\zeta}_j^{(i)}\geq 0\\
&i=1,\ldots,r,\qquad j=1,\ldots,n
\end{aligned} \tag{D.4}$$

Proof The following equality will help in future derivations:

$$\sum_{k=1}^{i} \zeta_j^{(k)}(\mathbf{x}) = \min\{d_j^{(1)} + \ldots + d_j^{(i)}, d_j^{(1)} + \ldots + d_j^{(i+1)}, \ldots, d_j^{(1)} + \ldots + d_j^{(r)}\} \tag{D.5}$$

This can be shown by induction. The base of induction ($i = 1$) follows from the definition. Now assume that the induction hypothesis holds for $i - 1$ and notice that

$$\begin{aligned}
\sum_{k=1}^{i} \zeta_j^{(k)}(\mathbf{x}) &= \min\{d_j^{(1)} + \ldots + d_j^{(i-1)}, d_j^{(1)} + \ldots + d_j^{(i)}, \ldots, d_j^{(1)} + \ldots + d_j^{(r)}\} \\
&\quad + \max\{0, \min\{d_j^{(i)}, d_j^{(i)} + d_j^{(i+1)}, \ldots, d_j^{(i)} + \ldots + d_j^{(r)}\}\} \\
&= d_j^{(1)} + \ldots + d_j^{(i-1)} \\
&\quad + \min\{0, d_j^{(i)}, d_j^{(i)} + d_j^{(i+1)}, \ldots, d_j^{(i)} + \ldots + d_j^{(r)}\} \\
&\quad + \max\{0, \min\{d_j^{(i)}, d_j^{(i)} + d_j^{(i+1)}, \ldots, d_j^{(i)} + \ldots + d_j^{(r)}\}\} \\
&= d_j^{(1)} + \ldots + d_j^{(i-1)} \\
&\quad + \min\{0, \min\{d_j^{(i)}, d_j^{(i)} + d_j^{(i+1)}, \ldots, d_j^{(i)} + \ldots + d_j^{(r)}\}\} \\
&\quad + \max\{0, \min\{d_j^{(i)}, d_j^{(i)} + d_j^{(i+1)}, \ldots, d_j^{(i)} + \ldots + d_j^{(r)}\}\} \\
&= d_j^{(1)} + \ldots + d_j^{(i-1)} \\
&\quad + \min\{d_j^{(i)}, d_j^{(i)} + d_j^{(i+1)}, \ldots, d_j^{(i)} + \ldots + d_j^{(r)}\} \\
&= \min\{d_j^{(1)} + \ldots + d_j^{(i)}, d_j^{(1)} + \ldots + d_j^{(i+1)}, \ldots, d_j^{(1)} + \ldots + d_j^{(r)}\}
\end{aligned} \tag{D.6}$$

1. Let $(\mathbf{K}, \xi, \eta)$ as in (D.1). Let $\tilde{\zeta}_j^{(i)} = \zeta_j^{(i)}(\mathbf{x})$ and $\tilde{h}_j^{(i)} = h_j^{(i)}(\mathbf{x})$. Then

$$\begin{aligned}
\tilde{g}_j^{(i)} &= \sum_{k=1}^{i} (\zeta_{j-1}^{(k)}(\mathbf{x}) + \xi_j^{(k)} - h_j^{(i)}(\mathbf{x})) \\
\tilde{f}_j^{(i)} &= \sum_{k=1}^{i} (h_j^{(k)}(\mathbf{x}) - \zeta_j^{(k)}(\mathbf{x}) - x_j \eta_j^{(k)})
\end{aligned} \tag{D.7}$$

Conditions

$$\tilde{\zeta}_j^{(i)} \geq 0, \quad \tilde{\zeta}_j^{(i)} \leq K^{(i)}, \quad \tilde{h}_j^{(i)} \geq 0, \quad \tilde{h}_j^{(i)} \leq K^{(i)}, \tag{D.8}$$

are easy to verify from the definition. Also,

$$\begin{aligned}
\tilde{g}_j^{(i)} &= \sum_{k=1}^{i} (\zeta_{j-1}^{(k)}(\mathbf{x}) + \xi_j^{(k)} - h_j^{(i)}(\mathbf{x})) \\
&= \sum_{k=1}^{i} (g_j^{(i)}(\mathbf{x}) - g_j^{(i-1)}(\mathbf{x})) \\
&= g_j^{(i)}(\mathbf{x}) \geq 0
\end{aligned} \tag{D.9}$$

and

$$\tilde{f}_j^{(i)} = \sum_{k=1}^{i} (h_j^{(k)}(\mathbf{x}) - \zeta_j^{(k)}(\mathbf{x}) - x_j \eta_j^{(k)}) = \sum_{k=1}^{i} (d_j^{(k)}(\mathbf{x}) - \zeta_j^{(k)}(\mathbf{x})) \geq 0 \tag{D.10}$$

where the last inequality follows from (D.5)

2. Let $(\mathbf{K}, \xi, \eta)$ as in (D.4). We will show that

$$\sum_{k=1}^{i} \zeta_j^{(k)}(\mathbf{x}) \geq \sum_{k=1}^{i} \tilde{\zeta}_j^{(k)} \tag{D.11}$$

by induction in j. The base of induction $j = 0$ follows from the definitions of $\zeta_j^{(i)}(\mathbf{x})$ and $\tilde{\zeta}_j^{(i)}$. To prove the step of the induction we assume that (D.11) holds for $j - 1$ and show that

$$\sum_{k=1}^{i} h_j^{(k)}(\mathbf{x}) \geq \sum_{k=1}^{i} \tilde{h}_j^{(k)} \tag{D.12}$$

Will show (D.12) by induction in i. Induction base at $i = 0$ is obvious. If we notice that

$$g_j^{(i)}(\mathbf{x}) + h_j^{(i)}(\mathbf{x}) = \zeta_{j-1}^{(i)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)} \tag{D.13}$$

we can write

$$\begin{aligned}
\sum_{k=1}^{i} h_j^{(k)}(\mathbf{x}) &= \sum_{k=1}^{i-1} h_j^{(k)}(\mathbf{x}) + \min\{\zeta_{j-1}^{(i)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)}, K_j^{(i)}\} \\
&= \min\{ \sum_{k=1}^{i-1} h_j^{(k)}(\mathbf{x}) + \zeta_{j-1}^{(i)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)}, \\
&\qquad \sum_{k=1}^{i-1} h_j^{(k)}(\mathbf{x}) + K_j^{(i)}\} \\
&= \min\{ \sum_{k=1}^{i-1} (\zeta_{j-1}^{(k)}(\mathbf{x}) + \xi_j^{(k)} + g_j^{(k-1)}(\mathbf{x}) - g_j^{(k)}(\mathbf{x})) \\
&\qquad + \zeta_{j-1}^{(i)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)}, \\
&\qquad \sum_{k=1}^{i-1} h_j^{(k)}(\mathbf{x}) + K_j^{(i)}\} \\
&= \min\{ \sum_{k=1}^{i-1} \zeta_{j-1}^{(k)}(\mathbf{x}) + \sum_{k=1}^{i-1} \xi_j^{(k)} - g_j^{(i-1)}(\mathbf{x}) \\
&\qquad + \zeta_{j-1}^{(i)}(\mathbf{x}) + g_j^{(i-1)}(\mathbf{x}) + \xi_j^{(i)}, \\
&\qquad \sum_{k=1}^{i-1} h_j^{(k)}(\mathbf{x}) + K_j^{(i)}\} \\
&= \min\{ \sum_{k=1}^{i} \zeta_{j-1}^{(k)}(\mathbf{x}) + \sum_{k=1}^{i} \xi_j^{(k)}, \sum_{k=1}^{i-1} h_j^{(k)}(\mathbf{x}) + K_j^{(i)}\}
\end{aligned} \tag{D.14}$$

On the other hand

$$\begin{aligned}
\sum_{k=1}^{i} \tilde{h}_j^{(k)} &= \sum_{k=1}^{i} (\tilde{\zeta}_{j-1}^{(k)} + \xi_j^{(k)} + \tilde{g}_j^{(k-1)} - \tilde{g}_j^{(k)}) \\
&= \sum_{k=1}^{i} \tilde{\zeta}_{j-1}^{(k)} + \sum_{k=1}^{i} \xi_j^{(k)} - \tilde{g}_j^{(i)} \\
&\leq \sum_{k=1}^{i} \tilde{\zeta}_{j-1}^{(k)} + \sum_{k=1}^{i} \xi_j^{(k)}
\end{aligned} \tag{D.15}$$

and

$$\sum_{k=1}^{i} \tilde{h}_j^{(k)} = \sum_{k=1}^{i-1} \tilde{h}_j^{(k)} + \tilde{h}_j^{(i)} \leq \sum_{k=1}^{i-1} \tilde{h}_j^{(k)} + K_j^{(i)} \tag{D.16}$$

which together with (D.14) and the induction assumptions (D.11), (D.12) proves (D.12).

Now, from (D.5),

$$\begin{aligned}
\sum_{k=1}^{i} \zeta_j^{(k)}(\mathbf{x}) &= \sum_{k=1}^{i} d_j^{(k)}(\mathbf{x}) + \min\{0, \sum_{k=i+1}^{r} d_j^{(k)}(\mathbf{x})\} \\
&= \min_{q=i,\ldots,r} \{\sum_{k=1}^{q} d_j^{(k)}(\mathbf{x})\} \\
&= \min_{q=i,\ldots,r} \{\sum_{k=1}^{q} (h_j^{(k)}(\mathbf{x}) - x_j \eta_j^{(k)})\} \\
&\geq \min_{q=i,\ldots,r} \{\sum_{k=1}^{q} (\tilde{h}_j^{(k)} - x_j \eta_j^{(k)})\} \\
&= \min_{q=i,\ldots,r} \{\sum_{k=1}^{q} (\tilde{\zeta}_j^{(k)} + \tilde{f}_j^{(k)} - \tilde{f}_j^{(k-1)})\} \\
&= \min_{q=i,\ldots,r} \{\sum_{k=1}^{q} \tilde{\zeta}_j^{(k)} + \tilde{f}_j^{(q)}\} \geq \sum_{k=1}^{i} \tilde{\zeta}_j^{(k)}
\end{aligned} \tag{D.17}$$

where the last inequality holds because all terms are non-negative. This completes the proof of (D.11). Now, (D.11) implies

$$\zeta_j^{(1)}(\mathbf{x}) \geq \tilde{\zeta}_j^{(1)} \geq 0 \quad \forall j \tag{D.18}$$

which means that $(\mathbf{K}, \xi, \eta)$ as in (D.1).

■

The constraints in (D.4) define a convex polytope. Since set (D.1) is a projection of the set defined by the constraints in (D.4), it is also a convex polytope and can be represented in the form

$$Tx \geq B\xi + C\eta + d \tag{D.19}$$

for some matrices T, B, C and a vector d.

Appendix E

Source Code

This appendix provides the MatLab/Octave source code used for the numerical experiments. The most important program files are

1. `prekopa.m` in Section E.1 implements Prekopa-Vizvari algorithm.
2. `hybrid_orig.m` in Section E.2 implements both the supporting hyperplane and the hybrid alorithms.
3. `torczon_imlicit.m` in Section E.3 implements Torczon algorithm.

E.1 prekopa.m

```
% Prekopa-Vizvari-Badics algorithm implementation
%  c - the costs vector
%  A - the matrix of the linear constraints
%  b - the right-hand side of the linear constraints
%  T - the transformation matrix in the probabilistic constraints
%  Z - the set of all p-efficient points, stored as columns in a matrix
% Rows of W0 form a basis in the subspace orthogonal to the affine hull
% of Z
function [x cutinfo]=prekopa(c,A,b,T,Z,W0)
global debugActiveList;
debugActiveList=zeros(0,size(Z,2));
feasibilityTol = 1e-6;
% Check validity of input arguments
% c must be a vector-column
if size(c,2) ~= 1
    c  = c';
```

```
end
% the dimension of the solution vector x
nx=length(c);
% the dimension of the random vector
nu=size(T,1);
if size(Z,1)~=nu
    error 'Expected size(Z,1)=size(T,1) = dimension of the random vector';
end
if isempty(A)
    A=zeros(0,nx);
end
%W=zeros(0,nu);
zbar=mean(Z,2);
if nargin<6
    W0=orthosubspace(Z-zbar*ones(1,size(Z,2)))';
end
% A basis in the space parallel to the affine hull of Z
W0orth=orthosubspace(W0')';
% Additional linear constraints to ensure that the first iterations of the
% algorithm don't lead to unbounded linear optimization problems. These
% constraints will not influence the optimal solution
Alb = [ -T eye(nu) ];
blb = -min(Z,[],2)+ones(nu,1);
% ones(nu,1) are here to make the lower bound strictly less than any final
% solution: omitting them sometimes leads to linearly dependent constraints
% and numerical problems.
% matrix W and vector Wrhs will define the supporting hyperplanes
% as: W*(T*x - u) >= Wrhs
W=zeros(0,nu);
Wrhs=zeros(size(W,1),1);
Wpts=zeros(nu,0);
effdim=nu-size(W0,1);
iterationNumber = 0;
goon=1;
```

```
while goon
    iterationNumber = iterationNumber + 1;
    fprintf(1, 'Prekopa-Vizvari, iteration %d\n', iterationNumber);
    if iterationNumber > 1000
        error('Maximal number of iterations exceeded');
    end
    % Solving the linear optimization problem in step 2 of the algorithm
    % The equality constraints force the solution to be in the same affine
    % space as the p-efficient points
    myAeq = [ W0*T -W0 ];
    mybeq = zeros(size(W0,1),1);
    % Inequality constraints are of three types:
    %  (i) the original inequality constraints A*x <= b
    %  (ii) the auxiliary constraints needed to prevent unbounded optimization
    %       problems
    %  (iii) the constraints from the supporting hyperplanes
    myA = [A zeros(size(A,1),nu) ; Alb ; -W*T W];
    myb = [b ; blb+zbar ; -Wrhs];
    [xu fooval fooexitflag foooutput cutlambda]=...
        linprog([c;zeros(nu,1)],myA,myb,myAeq,mybeq,[-inf(nx,1) ; zbar],[]);
    if fooexitflag < 0
        %error('Could not solve linear programming problem reliably');
        fprintf(1,'Could not solve linear programming problem reliably, ');
        fprintf(1,'exitflag=%g', exitflag);
        x=[];
        cutinfo=[];
        return;
    end
%    if any( [-W * T  W] * xu > -W * zbar - Wrhs + feasibilityTol )
    if any( [-W * T  W] * xu > -Wrhs + feasibilityTol )
        % This may happen because of round-off errors
        save('/tmp/octaveError.mat');
        error('"Internal error": found solution not feasible');
    end
```

```
% Solving the linear optimization problem in the step 3 of the algorithm
x=xu(1:nx);
u=xu((nx+1):(nx+nu)) - zbar;
[mu foo fooexitflag foooutput lambda]=linprog(ones(size(Z,2),1),[],[],...
    W0orth*(Z-zbar*ones(1,size(Z,2))),W0orth*(T*x-u),zeros(size(Z,2),1));
if fooexitflag < 0
    %error('Could not solve linear programming problem reliably');
    x=[];
    cutinfo=[];
    return;
end
% Check stopping criterion
if foo<=1+10^-5
    goon=0;
else
    % Creating the cut
    % Determine active constraints
    if isfield(lambda, 'lower')
        active=( mu>10^(-6) & lambda.lower<1e-5 );    % Matlab case
    else
        active=( mu > 1e-6 );    % Octave case
    end
    debugActiveList=cat(1,debugActiveList,active')   ;
    firstactive=find(active,1,'first');
    restactive=active;
    restactive(firstactive)=0;
    restactiveinds=(1:size(Z,2));
    restactiveinds=restactiveinds(restactive);
    if length(restactiveinds)<effdim-1
        disp('corner case');
        lambda
    end
    % The vector orthogonal to the affine space of all active constraints
    foow=orthosubspace(cat(2,Z(:,restactiveinds)-Z(:,firstactive)*...
```

```
                                    ones(1,length(restactiveinds)),W0'));
        if size(foow,2) > 1
            fprintf(1,'Warning: size(foow,2) = %d > 1 ', size(foow,2));
            fprintf(1,'in prekopa\n');
        end
        foow=foow(:,1)';
        % Make sure foow points inside the feasible set
        if foow*([T -eye(nu)]*xu) > 0
            foow=-foow;
        end
        % Add the new supporting hyperplane
        W=[W ; foow];
        Wrhs=[Wrhs ; foow*([T -eye(nu)]*xu)/foo];
        % If the constraint is satisfied within tolerance, assume we have
        % found a solution
        if [-W(size(W,1),:) * T  W(size(W,1),:)] * xu <= ...
                - Wrhs(length(Wrhs)) + feasibilityTol
            goon = 0;
        end
        Wpts=[Wpts , [T -eye(nu)]*xu/foo + zbar ];
    end
end
x=xu(1:nx);
if nargout >= 2
    cutinfo.A = -W;
    cutinfo.b = diag( -W*Wpts );
    if isempty( cutinfo.b )
        cutinfo.b = zeros(0,1);
    end
    cutinfo.lambda = cutlambda.ineqlin( (size(A,1)+1):(size(A,1)+size(W,1)) );
    cutinfo.pts = Wpts';
end
end
```

E.2 hybrid_orig.m

```
% Solves the probabilistic optimization problem
%  min c'*x
%  s.t.
%  A*x<=b
%  P(T*x>=Z) >= p
% where
%  Z is has c.d.f. F
%  dF is the gradient of F
% If the distribution of Z is singular, rows of W0 define an orthonormal
% basis in the normal subspace to the domain of Z.
%
% Note: The probabilistic constraint is actually: F(T*x) >= p
% The routine should work for any non-decreasing F with convex level
% sets.
% This function is using supporting hyperplane algorithm.
%  c - the costs vector
%  A - the matrix of the linear constraints
%  b - the right-hand side of the linear constraints
%  T - the transformation matrix in the probabilistic constraints
%  F - the CDF of the random vector (a function handle)
%  dF - the gradient of F (a function handle)
%  p - the probability level in the probabilistic constraint
%  xbar - a point in the interior of the feasible set
%  W0 - a matrix whose columns form a basis in the space orthogonal to the
%       affine space containing the p-efficient points
%  supphyp_only - if true, use supporting hyperplane algorithm, otherwise
%       use hybrid algorithm.
function x = hybrid_orig(c,A,b,T,F,dF,p,xbar,W0,supphyp_only)
% number of function invocations
global optstats_num_function_invocations
% number of gradient invocations
global optstats_num_gradient_invocations
```

```
eps=1e-6;
funtol=1e-5;
contol=funtol;
maxstep=1e5;

if nargin < 10
    supphyp_only = false;
end

n = length(c);
m = size(T,1);

optstats_num_function_invocations = 0;
% wrappedF is same as F but also counts the number of invocations
wrappedF = @(x)functionWrapper(F,x);
optstats_num_gradient_invocations = 0;
% Similarly, wrappedDF is same as dF but also counts the number of its
% invocations
wrappedDF = @(x)gradientWrapper(dF,x);

% curA and curb describe a set of linear constraints which will grow as
% supporting hyperplanes are added to it
curA = A;
curb = b;
% The set of all encountered p-efficient points
peffpts = zeros(m,0);

if supphyp_only
    % If we are not interested in the hybrid algorithm, set all its statistics
    % to zero
    hybrid_goon = 0;
    hybrid_num_function_invocations = 0;
    hybrid_num_gradient_invocations = 0;
```

```
    hybrid_num_iterations = 0;
else
    % Set the flag saying that the stopping criterion has not been met
    hybrid_goon = 1;
end
% Set the flag saying that the stopping criterion has not been met
supphyp_goon = 1;
numiter = 0;
while hybrid_goon || supphyp_goon
    numiter = numiter + 1;
    fprintf(1,'Iteration %d\n',numiter);
    % Solve the linear programming problem with the current set of constraints
    % to find the lower bound and a candidate solution x
    [x LB] = linprog(c,curA,curb);
    fprintf(1,'Done linprog\n');
    if F(T*x) >= p
        % If x is feasible, we have found the solution, terminate the
        % algorithms
        hybrid_goon = 0;
        hybrid_num_function_invocations = optstats_num_function_invocations;
        hybrid_num_gradient_invocations = optstats_num_gradient_invocations;
        hybrid_num_iterations = numiter;
        supphyp_goon = 0;
        supphyp_num_function_invocations = optstats_num_function_invocations;
        supphyp_num_gradient_invocations = optstats_num_gradient_invocations;
        supphyp_num_iterations = numiter;
    else
        % Create a supporting hyperplane
        lambda = bisection(...
            @(lambda)(wrappedF(T*(lambda*xbar+(1-lambda)*x))-p), 0, 1, eps );
        fprintf(1,'lambda = %g\n',lambda);
        xi = lambda*xbar+(1-lambda)*x;
        dh = -T'*wrappedDF(T*xi);
        curA = [ curA ; dh' ];
```

```
        curb = [ curb ; dh'*xi ];
        % Add the new point to the set of known p-efficient points
        peffpts = [ peffpts T*xi ];
        peffptsdiffs = peffpts( : , 2:size(peffpts,2) ) - peffpts(:,1) * ...
                       ones(1,size(peffpts,2)-1);
        W0pv = orthosubspace( peffptsdiffs )' ;
        % Now use Prekopa-Vizvari-Badics
        if ~ supphyp_only
            x = prekopa(c,A,b,T,peffpts,W0pv);
            hybridUB = c*x;
            % Check the stopping condition for the hybrid algorithm
            if hybridUB - LB <= funtol
                hybrid_goon = 0;
                hybrid_num_function_invocations = ...
                    optstats_num_function_invocations;
                hybrid_num_gradient_invocations = ...
                    optstats_num_gradient_invocations;
                hybrid_num_iterations = numiter;
            end
        end
        % Check the stopping condition for the supporting hyperplane algorithm
        supphypUB = c*xi;
        if supphypUB - LB <= funtol
            supphyp_goon = 0;
            supphyp_num_function_invocations = ...
                optstats_num_function_invocations;
            supphyp_num_gradient_invocations = ...
                optstats_num_gradient_invocations;
            supphyp_num_iterations = numiter;
        end
        save('sequential_pure_state.mat');
    end
end
fprintf(1,'Number of iterations: %d\n',numiter);
```

```
fprintf(1,'Hybrid: number of iterations: %d\n', hybrid_num_iterations);
fprintf(1,'Hybrid: number of function invocations: %d\n',...
        hybrid_num_function_invocations)'
fprintf(1,'Hybrid: number of gradient invocations: %d\n',...
        hybrid_num_gradient_invocations)'
fprintf(1,'Supporting hyperplane: number of iterations: %d\n',...
        supphyp_num_iterations);
fprintf(1,'Supporting hyperplane: number of function invocations: %d\n',...
        supphyp_num_function_invocations)'
fprintf(1,'Supporting hyperplane: number of gradient invocations: %d\n',...
        supphyp_num_gradient_invocations)'
% x is the solution
end

function res = functionWrapper(F,x)
global optstats_num_function_invocations
optstats_num_function_invocations = optstats_num_function_invocations + 1;
res = F(x);
end

function res = gradientWrapper(F,x)
global optstats_num_gradient_invocations
optstats_num_gradient_invocations = optstats_num_gradient_invocations + 1;
res = F(x);
end
```

E.3 Torczon's derivative-free algorithm: torczon_implicit.m

```
% torczon's derivative-free optimization algorithm
% Minimize the function y(x) given implicitly as f(x,y) = 0
% Here x is a vector, y is scalar.
%  f - the implicit function definition, must be increasing in y
%  x0 - initial value of x
%  y0 - initial value of y (f(x0,y0) = 0 is not required)
```

```
function [x y] = torczon_implicit(f,x0,y0)
% Parameters
% Expansion coefficient
expcoef = 2;
% Contraction coefficient
contrcoef = 1 / expcoef;
% Accuracy in x and y
xeps = 1e-4;
yeps = 1e-4;

if size(x0,2) ~= 1
    error('x0 must be a column-vector');
end
n = size(x0,1);
% The initial value of the simplex
xsimplex = x0 * ones(1,n+1);
xsimplex(1:n,1:n) = xsimplex(1:n,1:n) + eye(n);

printf("torczon_implicit: initializing\n");
% The bounds for the function value at the points of the simplex
% We will always have:
%  f(xsimplex(:,i), fvals(1,i)) <= 0
%  f(xsimplex(:,i), fvals(2,i)) >= 0
fvalsx = [zeros(1,n+1) ; ones(1,n+1)];
[fvalsx bestxind] = find_smallest(f, xsimplex, fvalsx, false(1,n+1));
% bestxind is the index of the point in the simplex where the function value
% is the smallest

% The main loop
goon = true;
iteration_count = 1;
while goon
    printf("torczon_implicit: iteration %d\n", iteration_count);
    % Perform a step of the algorithm
```

```
    [fvalsx xsimplex bestxind finishx] = ...
        torczon_step(fvalsx,xsimplex,bestxind,f);

    bestxind
    xsimplex
    fvalsx

    % Check stopping condition
    if finishx
        goon = false;
    else
        iteration_count = iteration_count + 1;
    end
end

x = xsimplex(:,bestxind);
y = (fvals(1,bestxind) + fvals(2,bestxind)) / 2;
end

% Given the implicitly-defined function y(x) given as f(x,y) = 0 and a set
% of points, find the point where the function has the smallest value.
%  f - defines the implicit function, f(x,y) must be increasing in y
%  x - a set of points, given as columns of a matrix
%  fvals - the known bounds for the implicit function value at points x, when
%          available.
%  initflag - a boolean vector, indicating which entries of favls are
%             initialized.
% That is, if initflag(j) == true then f(x(:,j), fvals(1,j)) <= 0 and
% f(x(:,j), fvals(2,j)) >= 0
% If initflag(j) == false, no such assumption is made, and fvals(1,j) and
% fvals(2,j) are used as an initial guess to find proper bounds.
function [fvals bestx] = find_smallest(f, x, fvals, initflag)
% Dimension of the space
n = size(x,1);
```

```
% Number of points
m = size(x,2);
% Initialize all fvals
printf("torczon_implicit.find_smallest: initializing fvals\n");
for j=1:m
    if ~initflag(j)
        if f(x(:,j), fvals(1,j)) > 0
            step = fvals(2,j) - fvals(1,j);
            goon = true;
            while goon
                fvals(2,j) = fvals(1,j);
                fvals(1,j) = fvals(2,j) - step;
                step = 2 * step;
                if f(x(:,j), fvals(1,j)) <= 0
                    goon = false;
                end
            end
        elseif f(x(:,j), fvals(2,j)) < 0
            step = fvals(2,j) - fvals(1,j);
            goon = true;
            while goon
                fvals(1,j) = fvals(2,j);
                fvals(2,j) = fvals(1,j) + step;
                step = 2 * step;
                if f(x(:,j), fvals(2,j)) >= 0
                    goon = false;
                end
            end
        end
    end
end
% Break ties
printf("torczon_implicit.find_smallest: breaking ties\n");
goon = true;
```

```
while goon
    [lb bestx] = min(fvals(1,:));
    goon = false;
    for j = 1:m
        if (j ~= bestx) && (fvals(1,j) < fvals(2,bestx))
            testy = (fvals(1,j) + fvals(2,j)) / 2;
            if f(x(:,j), testy) <= 0
                fvals(1,j) = testy;
            else
                fvals(2,j) = testy;
            end
            goon = true;
        end
    end
    if goon
        testy = (fvals(1,bestx) + fvals(2,bestx)) / 2;
        if f(x(:,bestx), testy) >= 0
            fvals(2,bestx) = testy;
        else
            fvals(1,bestx) = testy;
        end
    end
end
% The output variables already have the correct values
end

% One iteration of the Torczon algorithm
%  fvals - the bounds for the function values at the simplex coordinates
%  simplex - the coordinates of the current simplex points, stored as columns
%            of a matrix
%  bestind - the index of the simplex point with the smallest function value
%  f - defines the implicit function
% Outputs:
%  fvals - the bounds for the function values at the new simplex coordinates
```

```
%  simplex - the new simplex
%  bestind - the index of the point in the new simplex with the smallest
%            function value
%  finish - true if the stopping criterion has been met
function [fvals simplex bestind finish] = ...
        torczon_step(fvals, simplex, bestind, f)
% Parameters
% Expansion coefficient
expand_coef = 2;
% Contraction coefficient
contract_coef = 1 / expand_coef;
% Accuracy
eps = 1e-4;

n = size(simplex,1);
% Reflected simplex and corresponding bounds for the function values
simplex_refl = zeros(n,n+1);
fvals_refl = zeros(2,n);
% The current simplex and corresponding bounds for the function values
simplex_cur = zeros(n,n+1);
fvals_cur = zeros(2,n);

goon = true;
finish = false;
while goon
    % "Reflect" step
    % Find the reflected simplex
    for i = 1:n+1
        if i == bestind
            simplex_refl(:,i) = simplex(:,i);
            fvals_refl(:,i) = fvals(:,i);
        else
            simplex_refl(:,i) = 2 * simplex(:,bestind) - simplex(:,i);
            fvals_refl(:,i) = fvals(:,bestind);
```

```
    end
end
% Determine which point of the reflected simplex has the smallest
% function value
initflag = false(1,n+1);
initflag(bestind) = true;
[fvals_refl bestind_refl] = ...
    find_smallest(f, simplex_refl, fvals_refl, initflag);
if bestind_refl ~= bestind
    % Reflection yielded an improvement, now do the "expand" step
    % Compute the expanded simplex
    for i = 1:n+1
        if i == bestind
            simplex_cur(:,i) = simplex_refl(:,i);
            fvals_cur(:,i) = fvals_refl(:,i);
        else
            simplex_cur(:,i) = expand_coef * simplex_refl(:,i) +...
                               (1 - expand_coef) * simplex_refl(:,bestind);
            fvals_cur(:,i) = fvals_refl(:,i);
        end
    end
    % Find the point with the smallest function value among the reflected
    % and the expanded simplices
    subindex = true(1,n+1);
    subindex(bestind_refl) = false;
    simplex_ex = [simplex_refl simplex_cur(:,subindex)];
    fvals_ex = [fvals_refl fvals_cur(:,subindex)];
    initflag = false(1,2*n+1);
    initflag(1:(n+1)) = true(1,n+1);
    [fvals_ex bestind_ex] = ...
        find_smallest(f, simplex_ex, fvals_ex, initflag);
    if bestind_ex > n+1
        % The expanded simplex leads to a smaller function value; that
        % will be the outcome of the iteration
```

```
        simplex = simplex_cur;
        fvals(:,subindex) = fvals_ex(:,(n+2):(2*n+1));
        fvals(:,bestind_refl) = fvals_ex(:,bestind_refl);
    else
        % The expanded simplex does not lead to a smaller function value;
        % the reflected simplex will be the outcome of the iteration
        simplex = simplex_refl;
        fvals = fvals_ex(:,1:(n+1));
    end
    goon = false;
else
    % "Contract" step
    % Compute the contracted simplex
    for i = 1:n+1
        if i == bestind
            simplex_cur(:,i) = simplex(:,i);
            fvals_cur(:,i) = fvals(:,i);
        else
            simplex_cur(:,i) = contract_coef * simplex(:,i) +...
                               (1 - contract_coef) * simplex(:,bestind);
            fvals_cur(:,i) = fvals(:,i);
        end
    end
    % Find the point with the smallest function value
    initflag = false(1,n+1);
    initflag(bestind) = true;
    [fvals_cur bestind_cur] = ...
        find_smallest(f, simplex_cur, fvals_cur, initflag);
    if bestind_cur ~= bestind
        % Contraction yielded an improvement; the contracted simplex
        % is the outcome of the iteration
        goon = false;
        bestind = bestind_cur;
    end
```

```
        % If there was no improvement, we will try the same steps starting
        % from the contracted simplex
        simplex = simplex_cur;
        fvals = fvals_cur;
        % check stopping condition
        finish = true;
        for i = 1:n+1
            if norm(simplex(:,i) - simplex(:,bestind)) > eps
                finish = false;
            end
        end
        if finish
            goon = false;
        end
    end
end
end
```