Robust solutions of bi-blend recipe optimization with quadratic constraints

Author: Freek Niewerth
MSc Research report
Universidad de Málaga in co-operation with Wageningen University
and Universidad de Almería
Dpt. Computer Architecture, Operations Research and Logistics
and Computer Architecture and Electronics

Supervisors: E.M.T. Hendrix and L.G. Casado

February 26, 2009

Abstract

Production companies use raw materials to compose end-products. They often make different products with the same raw materials. In this research, the focus lies on the production of two end-products consisting of (partly) the same raw materials as cheap as possible. Each of the products has its own demand and quality requirements consisting of quadratic constraints.

The minimization of the costs, given the quadratic constraints is a global optimization problem, which can be difficult because of possible local optima. Therefore, the multi modal character of the (bi-) blend problem is investigated. Standard optimization packages (solvers) in Matlab and GAMS were tested on their ability to solve the problem. In total 20 test cases were generated and taken from literature to test solvers on their effectiveness and efficiency to solve the problem.

The research also gives insight in adjusting the quadratic constraints of the problem in order to make a robust problem formulation of the bi-blend problem.

Contents

Chapter 1

Introduction

1.1 General goal

This master thesis is written for Wageningen University and the University of Almeria. In order to write a good report the University of Wageningen has set some predefined standards. These standards are described in the Msc Thesis protocol of the WUR. The general goal of a thesis written by a WUR-student is the following: "The overall goal of the thesis is the development of research skills and the ability to analyze and present research results in a systematic and clear way. The thesis is the culmination of the MSc study program in which the student will have to show that he/she is able to design and conduct social science research at an academic level and is able to theoretically reflect on a particular field of research relevant to the MSc program at hand." [1]

1.2 Global optimization

This thesis is about a specific global optimization problem. The objective of a global optimization method is to find the best possible solution (global optimum) of a problem by modifying decision variables in order to minimize (or maximize) an objective function, while there may also exist some local optima. Sometimes the decision variables have to fulfil some requirements.

This can be compared with the search of the highest point above sea level in the world. When we want to maximize the height (objective) of mountains in the world we can vary the location by adjusting the longitude and the latitude. These are the decision variables for this problem. A local optimum in the region of the Alps will be Mont Blanc, in the region of Africa this will be the Kilimanjaro and in the region of Asia the Mount Everest. The global optimum however is the Mount Everest since this is the highest mountain in the world. In Table 1.1 the main characteristics of optima are described. We can see that the local optimum for Asia is also the global optimum for the world. This gives us the following property:

Global optimum $\subset$ Local optima.

Global optimization problems can be difficult because of the existence of local optima. When one would only search in the Alps and conclude Mont Blanc is highest in that region, one could easily think this is the highest mountain of the world since in its direct neighbourhood there is no mountain which is higher than Mont Blanc.

Table 1.1: Optima

Type	Corresponding mountain
Global optimum	Mount Everest
Local optimum	Mont Blanc, Kilimanjaro, Mount Everest

This thesis is about a global optimization problem in which minimizing a function given some decision variables is the objective. We will use the global optimization problem formulation from [5] to describe optimization problem in a mathematical way as is shown in (1.1).

$$\min\{f(x)\}, x \in X \subset \mathbb{R}^n, \tag{1.1}$$

in which $f(x)$ is a real valued continuous function and x varies in a continuous way in a set X with a dimension n. Formula (1.1) describes the minimization of a function f. We are trying to find the global minimum, where there also may exist local minima.

In Figure 1.1, function $f(x)$ is plotted of which we want to know the minimum value on the interval $0 \leq x \leq 20$. In the figure we can see four local optima that are minimal in their environment. The lowest local optimum is the global optimum and the value we want to know.

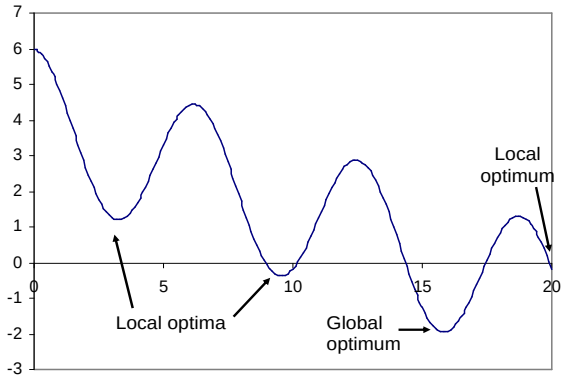


Figure 1.1: Four local optima for $f(x) = 2\cos(x) - \frac{x}{4} + 4$, $0 \leq x \leq 20$.

In this thesis, the global optimization problem of mixing raw ingredients in order to produce two different products will be discussed; the so-called bi-blending problem.

1.3 Problem statement

Companies use raw materials to make products. In fact, they often make different products with the same raw materials. In this research we focus on a problem with two products consisting of (partly) the same raw materials. Each of the products has its own demand and quality requirements consisting of quadratic constraints. In order to optimize this so called bi-blending problem we need to find robust solutions. To decide whether a solution is robust the following definition of a robust decision is used: "A robust decision is the best possible choice, one found by eliminating all the uncertainty possible within available resources, and then choosing, with known and acceptable levels of satisfaction and risk" [11]. This means for example that a found optimal solution must stay feasible despite some known variation in dosage of a raw material.

1.4 Vision

The main goal of the project is to investigate methods to optimize and find robust solutions of bi-blending problems with quadratic constraints. We need to develop a way to generate (robust) solutions numerically. This can either be done with existing software and/or by designing new algorithms. The vision of this thesis is to find a standard procedure to optimize bi- blending problems in a robust way.

1.5 Research questions

Corresponding to the problem statement and vision the main question of this thesis is:

- Which methods can be used to optimize and find robust solutions of bi-blending problems with quadratic constraints?

This question is subdivided in four sub questions:

1. How can the optimization problem be described?
2. What is the multimodal character of the problem?
3. What is the potential of existing standard software?
4. How can the robustness of solutions be determined?

1.6 Structure of this report

In Chapter 2, the blending problem is discussed, in Chapter 3 the bi-blending problem. Chapter 4 gives results of standard optimization software on (bi-) blending problems and in Chapter 5 the robustness problem is further investigated.

Chapter 2

Blending with quadratic constraints

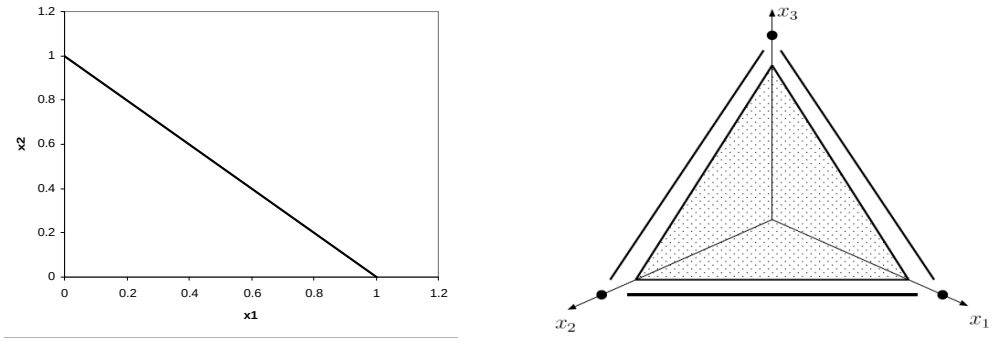
In the food industry, raw materials are put together and processed to produce products. The easiest way of processing is done by putting different raw materials together to mix them. These mixing processes do not only occur in the food sector but also in other types of industries.

2.1 Unit simplex and Cost function

The blending of raw materials can be described in a mathematical way. A vector x with a number of elements equal to the amount of raw materials is used to indicate the fraction (x_i) of every raw material (i) used in the mix. All fractions in the mix have to add up to 1. A recipe (of a mix) is mathematically defined by the unit simplex:

$$S = \{x \in \mathbb{R}^n \mid \sum_{i=1}^{n} x_i = 1;\ x_i \geq 0\}. \tag{2.1}$$

where n denotes the number of raw materials. In Figure 2.1b, the unit simplex is given in $\mathbb{R}^3$ (three raw materials). In this simplex we can find 7 possible sets depending on the composure and the number of raw materials involved. Namely, a recipe consisting of only: x_1, x_2 or x_3, denoted by the points; a recipe consisting of: x_1 and x_2, x_1 and x_3 or x_2 and x_3, denoted by the lines; and a recipe consisting of x_1, x_2 and x_3, denoted by the triangular area.



(a) $\mathbb{R}^2$ (b) $\mathbb{R}^3$

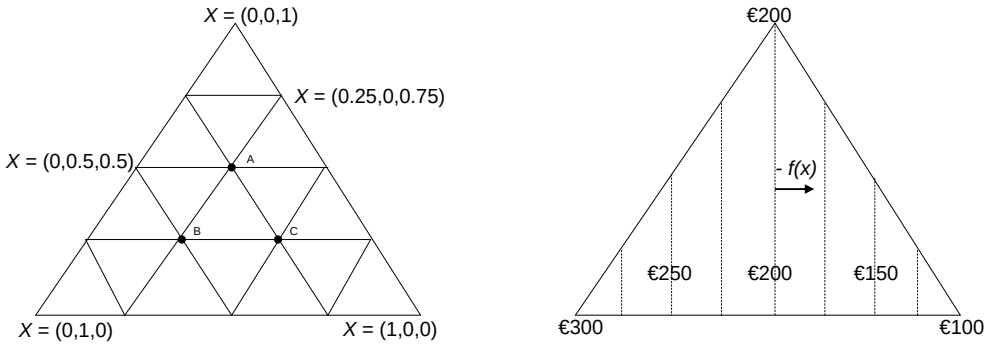
Figure 2.1: UnitSimplex in $\mathbb{R}^2$ and $\mathbb{R}^3$

In blending problems, the objective is to find a recipe x that minimises the cost of the material. The cost function is given by (2.2).

$$f(x) = c^T x, \tag{2.2}$$

where vector c gives the cost of the raw materials.

In Figure 2.2 we sketch the triangular area of Figure 2.1b in two dimensions. The value of x as well as the costs can be read from this triangle. Figure 2.2a shows a grid over the unit simplex so we can read the value of x. Figure 2.2b shows a cost vector c $= (100, 300, 200)^T$ using iso-cost lines.



(a) Grid: A: $x = (0.25, 0.25, 0.5)^T$, B: $x = (0.25, 0.5, 0.25)^T$, C: $x = (0.5, 0.25, 0.25)^T$.

(b) Contours of the cost function

Figure 2.2: 3-Dimensional unit simplex sketched in two dimensions.

2.2 Linear constraints

Recipes have to satisfy some requirements to become the intended end product. For relatively simple blending problems these requirements are bounds on an in-

gredient or linear constraints. Together with the unit simplex these requirements determine the feasible area X, as in (1.1); If $x \in X$, x is a feasible recipe.

Example 2.1
A sports drink manufacturer wants to minimize the costs of his drinks. The product consists of two raw materials; syrup and water. Syrup costs 2 euro per liter whereas water costs only 0.01 euro per liter. If the recipe contains more than 80% water, the recipe is too watery. If the recipe contains more than 40% syrup, the recipe becomes too sweet. Mathematically the problem can be solved by Linear Programming:

$$\begin{array}{lllr} \min & f(x) = c^T x & \text{(Cost} & (2.2)) \\ \text{s.t.} & x \in S & \text{(Unit Simplex} & (2.1)) \\ & x_{water} \leq 0.8 & \text{(Water constraint)} & \\ & x_{syrup} \leq 0.4 & \text{(Syrup constraint)} & \end{array}$$

where $x = \begin{pmatrix} x_{water} \\ x_{syrup} \end{pmatrix}$ and $c = \begin{pmatrix} c_{water} \\ c_{syrup} \end{pmatrix} = \begin{pmatrix} 0.01 \\ 2 \end{pmatrix}$.

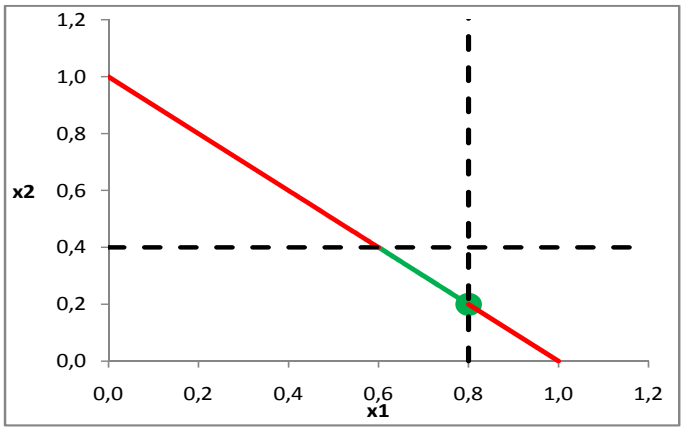


Figure 2.3: The green area denotes the feasible set X. The optimal solution to the 'water-syrup' problem lies in $x^T = (x_{water}, x_{syrup}) = (0.8; 0.2)$ $f(x) = 0.408$.

2.3 Quadratic constraints

In practice there are more difficult blending problems to solve. These problems can have higher dimensions, which depend on the amount of ingredients, and besides linear constraints also have quadratic constraints. An example of a 3-dimensional problem with quadratic constraints is given in [6] with the 'Rum-coke' problem (see Example 2.2). Blending problems with quadratic constraints are also referred to as Quadratic Mixture Design Problems (QMDP). Quadratic constraints are written as:

$$g_p(x) = x^T A_p x + b_p^T x + d_p \leq 0; \quad p = 1, \ldots, P, \tag{2.3}$$

in which A_p is a symmetric n by n matrix, b_p is an n-vector and d_p is a scalar. The quadratic constraints can be summarized in $Q = \{x \in \mathbb{R}^n \mid g_p(x) \leq 0;\ p = 1, \ldots, P\}$[2].

2.4 Formulation of blending problem

In a blending problem we want to minimize the costs of a product. We do this by minimizing the cost function (2.2) over a set $x \in X$ (1.1). In a blending problem X is defined as:

$$X = S \cap Q. \tag{2.4}$$

The blending problem is summarized by (2.5). In Example 2.2 a blending problem is formulated.

$$\begin{array}{lllll} \min & \{f(x) = c^T x\} & \text{(Cost} & & \\ \text{s.t.} & x \in S & \text{(Unit simplex} & (2.1)) & \\ & x \in Q & \text{(Q.constraint(s)} & (2.3)) & \end{array} \tag{2.5}$$

Example 2.2 (The 'Rum-coke' problem)
A bar owner sells rum-coke which consists of three ingredients, namely rum (x_3), coke (x_2) and ice (x_1). The rum-coke has to be strong enough (but not too strong) and cold enough in order to sell it. These requirements are given by two quadratic constraints. The costs for the bar owner have to be minimized, given cost-vector $c = (0.1, 0.7, 4.0)^T$.

The other parameters are as follows:

$$A_1 = \begin{pmatrix} 0 & -16 & 0 \\ -16 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, b_1 = \begin{pmatrix} 8 \\ 8 \\ 0 \end{pmatrix}, d_1 = -1$$

$$A_2 = \begin{pmatrix} 10 & 0 & 2 \\ 0 & 0 & 0 \\ 2 & 0 & 2 \end{pmatrix}, b_2 = \begin{pmatrix} -12 \\ 0 \\ 4 \end{pmatrix}, d2 = 3.7$$

Using all parameter values for (2.1), (2.2) and (2.3) results into instance:

$$\begin{array}{llllll} \min & f(x) = (0.1, 0.7, 4.0)^T x & & & \text{(Cost} & (2.2)) \\ \text{s.t.} & \sum_{i=1}^{3} x_1 & = 1; & \text{(Unit simplex} & (2.1)) \\ & x_1, x_2, x_3 & \geq 0. & \text{(Unit simplex} & (2.1)) \\ & g_1(x) = -32x_1x_2 + 8x_1 + 8x_2 - 1 & \leq 0; & \text{(Q.constraint} & (2.3)) \\ & g_2(x) = 10x_1^2 + 2x_3^2 + 4x_1x_3 - 12x_1 + 4x_3 + 3.7 & \leq 0; & \text{(Q.constraint} & (2.3)) \end{array} \tag{2.6}$$

This problem can be plotted in a 2-dimensional way. In [6] this is done by Figure 2.4.

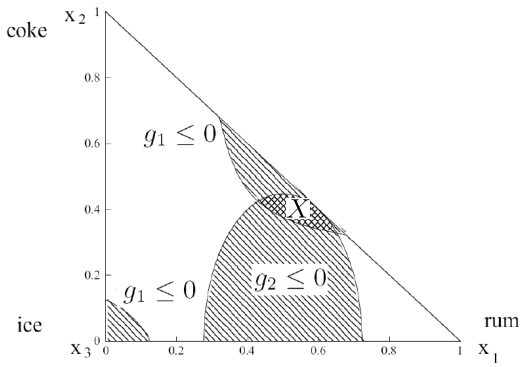
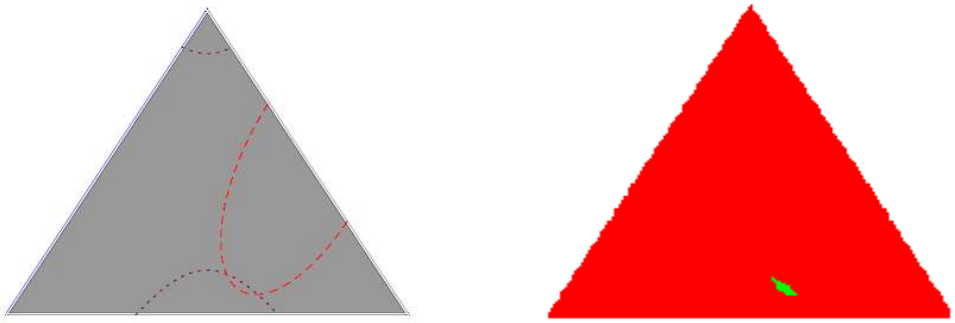


Figure 2.4: The 'Rum coke'-problem sketched in two dimensions. The axis for ice is perpendicular to the 2-dimensional x_1, x_2 - plain in (0, 0); this means for $x_1, x_2 = (0.3, 0.5)$; $x_3 = 0.2$, given the unit simplex. X denotes the feasible area of the problem.

We can also plot this problem according to the notation used in Figure 2.2. Using this notation the 'Rum-coke' problem is depicted in Figure 2.5.



(a) Constraints.

(b) Feasible area.

Figure 2.5: The 'Rum coke'-problem sketched in two dimensions. The green area denotes the feasible area.

2.5 Global and local optima

Like many global optimization problems, blending problems can have several local optima. This can be caused by a feasible area consisting of several compartments or by a feasible compartment which is non-convex.

2.5.1 Feasible area with two compartments

In Example 2.3 a blending problem with two local optima is given. Of these optima only one is the global optimum.

Example 2.3 (A feasible area consisting of two compartments.)
We have a concave quadratic constraint which lies on the unit simplex, the parameters of this problem are:

$$c = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, A_1 = \begin{pmatrix} -5 & 1 \\ 1 & -1 \end{pmatrix}, b_1 = \begin{pmatrix} 3 \\ 0.5 \end{pmatrix}, d_1 = -0.6$$

This problem is plotted in Figure 2.6 where we can see that the feasible area consists of two compartments. Corresponding to the compartments two local optima exist; the global optimum A in $x = (0, 1)^T$ and a local optimum B. B is local since it is more expensive than A (because x_1 is more expensive than x_2).

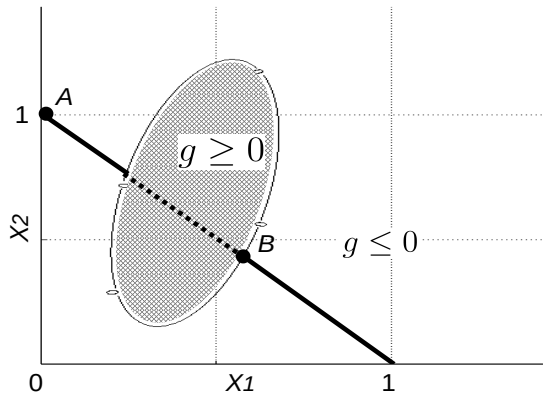


Figure 2.6: The feasible area has two separate compartments denoted by the bold lines on the unit simplex.

2.5.2 Non-convex feasible compartment

When a feasible area of a blending problem is non-convex, a problem can have local optima. This is illustrated in Figure 2.7. Towards x_2 the costs decrease, which means that when we move the contour line to the left, the recipe gets cheaper. While doing this, we obtain different extreme points of the feasible area. In Figure 2.7 this will cause local optima A and B, which in each of their neighbourhoods are the best optima. A will be the global optimum since this is the cheapest local optimum.

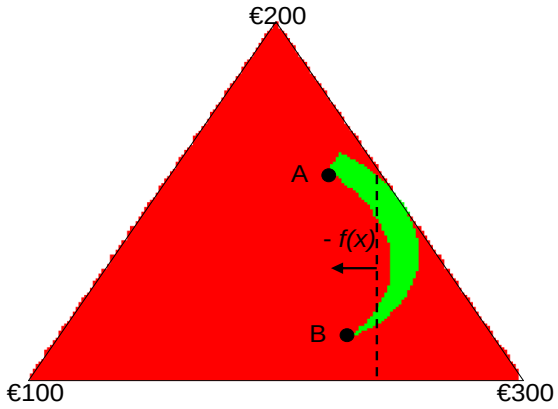


Figure 2.7: Two local optima for a non-convex function.

2.6 Test cases

In Chapter 4 we test different solvers on several blending problems. Therefore we use the 2-dimensional example from Figure 2.6 (test case 11), 11 3-dimensional test cases from [7] and [9] and two 7-dimensional test cases from [4]. An overview of these test cases is made in Table 2.1. The first two headers of the table describe the test case and its dimension. Header 'Characteristics' says something about the multi modality of the corresponding test case. When a test case is indicated with a '-' this means that the test case has only one (global) optimum.

Table 2.1: Test cases

Test case	Dimension	Characteristics
1	3	Non-feasible
2	3	Non-convex
3	3	Non-feasible
4	3	Non-feasible
5	3	-
6	3	-
7	3	Non-convex
8	3	Two compartments & non-convex
9	3	Two compartments
10	3	Two compartments
11	2	Two compartments
Rumcoke	3	-
7dimensionA	7	-
7dimensionB	7	-

The parameters of all the test cases are described in Appendix A, as well as plots for all feasible 3-dimensional cases. The 3 dimensional test cases are plotted

using the 3 dimensional unit simplex from Figure 2.2.

For the 3-dimensional cases we have the 'Rum-coke' problem and 10 other test cases. Test case 1 to 6 are 'regular' blending problems with a feasible area which consists of one compartment or where no feasible solution exists. Test cases 7 to 10 are *extreme* cases. Test case 7 is the non-convex problem from Figure 2.7; it may have multiple local optima when an appropriate cost vector c is chosen. Test case 8 to 10 each have two compartments for their feasible area and thus certainly contain local optima, like Example 2.3.

Although the plot in Figure 2.5 seems straightforward in finding a feasible area, solving a blending problem is not easy. Making a plot can be time consuming and for dimensions higher than 3 a plot cannot be made. Therefore, algorithms are used for solving quadratic blending problems. In some software, like Excel, Matlab and GAMS, *solvers* exist which can help us solving the problems. In Chapter 4 several of these solvers will be tested and compared with each other.

In the next chapter the concept of bi-blending is explained. In many industries, several products are made simultaneously with use of the same raw materials, which creates a new problem when certain raw materials become scarce.

Chapter 3

Bi-Blending

3.1 Introduction

The focus of this research is on bi-blending. The concept of bi-blending is taken from multi-blending where multiple products are made from the same (scarce) raw materials. For this research, the focus is on two products; this is called the bi-blend problem. In industry manufacturers sometimes face the problem of scarcity of raw materials for the products they want to make. As with the blending problem, this can be solved by adding a bigger dosage of another ingredient. For the description of the two recipes, we use the variables x and y for product 1 and product 2 respectively. The scarcity of raw materials is described by a capacity constraint.

3.2 Capacity constraint

The availability of raw materials is described by the capacity constraint. This constraint is given by (3.1).

$$D_1 x_i + D_2 y_i \leq B_i \quad i = 1, \ldots, n, \tag{3.1}$$

where D_1 and D_2 represent the amount of respectively product 1 and product 2 to be made. The amount of available raw material i is given by B_i.

Example 3.1
A farmer wants to grow 10 pigs and 3 cows. A pig needs fodder consisting of 80% corn ($i = 1$) and 20% water ($i = 2$) and a cow respectively 70% and 30%. However, one cow uses twice the amount of raw materials. The farmer has a silo with a capacity of 15 units of corn. Water is unlimited. The farmer wants to know if his silo is big enough to grow the pigs and the cows.

When we enter the values for the variables and data (x, y, D and B) we can easily decide if the silo is big enough:

$$x = \begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix}, y = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}, D_1 = 10, D_2 = 3 * 2 = 6, B = \begin{pmatrix} 15 \\ \infty \end{pmatrix}$$

.

Substitution in (3.1) gives:

$$10 * \begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix} + 6 * \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix} \leq \begin{pmatrix} 15 \\ \infty \end{pmatrix}$$

.

The sum of the used corn $(i = 1)$ *is:* $10 * 0.8 + 6 * 0.7 = 12.2 \leq 15$. *The silo is big enough so there are no problems for the farmer.*

3.3 Cost function

With the capacity constraint added to the problem, the cost function changes since we are now taking two products into account with their individual demands. The cost function of the bi-blending problem can be written as:

$$f(x, y) = \sum_{i=1}^{n} c_i(D_1 x_i + D_2 y_i). \tag{3.2}$$

We can add the capacity constraint and the new cost function to quadratic constraints for each individual product to formulate the bi-blend problem.

3.4 Formulation of the Bi-blend problem

The Bi-Blending problem is described by (3.3).

$$\begin{array}{lllll} \min & \{f(x,y) = \sum_{i=1}^{n} c_i(D_1 x_i + D_2 y_i)\} & \text{(Cost} & (3.2)) & \\ \text{s.t.} & x, y \in S & \text{(Blending} & (2.1)) & \\ & x \in Q_1, \quad y \in Q_2 & \text{(Feasibility} & (2.3)) & \\ & D_1 x_i + D_2 y_i \leq B_i \quad i = 1, \ldots, m & \text{(Capacity constraint} & (3.1)). & \end{array} \tag{3.3}$$

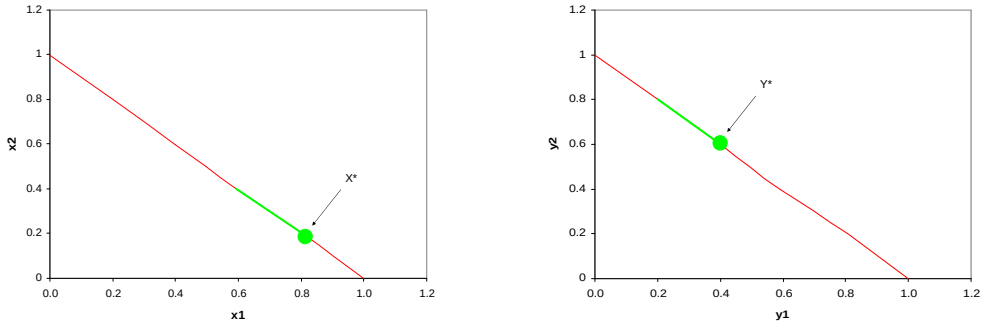
Note that the ingredients are denoted by i. This means that x_i and y_i (for the same i) are fractions of the same ingredient; only the amount of the ingredient varies. We give an easy example of a bi-blend problem with only bounds on the use of raw materials for product 1 and product 2 and a capacity constraint.

3.5 A 2-dimensional example

We have an instance with bounds on the mixtures for product 1 and 2. The parameters for this problem are:

$$c = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, D_1 = 2, \; D_2 = 4, \; x \leq \begin{pmatrix} 0.8 \\ 0.4 \end{pmatrix}, \; y \leq \begin{pmatrix} 0.4 \\ 0.8 \end{pmatrix}, \; B = \begin{pmatrix} 5 \\ 5 \end{pmatrix}.$$

The problem is sketched in Figure 3.1.



(a) x^* (b) y^*

Figure 3.1: Bi-blend problem 1

Table 3.1: Data and solution of Bi-blend instance 1

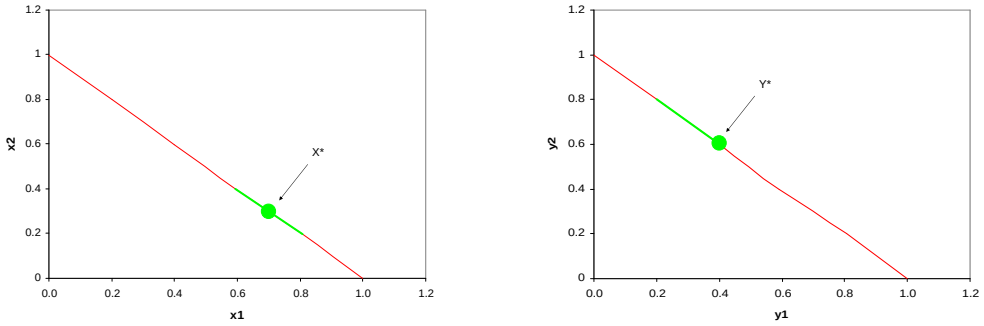
i	x_i	D_1	y_i	D_2	$Totali$	B_i
1	0.8	2	0.4	4	3.2	5
2	0.2	2	0.6	4	2.8	5

$$f^* = 3.2 * 1 + 2.8 * 2 = 8.8$$

We can see that the capacity constraint is not binding in Figure 3.1 and from Table 3.1. Of raw material i=1, 2*0.8 + 4*0.4 = 3.2 is used and of raw material i=2, 2*0.2 + 4*0.6 = 2.8 is used while the capacity is $B^T = (5, 5)$. In both product 1 and 2 the maximum amount of $i = 1$ is used since this is the cheapest ingredient. This means that for product 1 and 2 the cheapest mixtures could be made. This means that when the capacity constraint is not binding we can write the bi-blend problem as two separate blending problems as in (3.4).

$$\min_{x,y \in Q \cap S} \{f(x,y) = \sum_{i=1}^{n} c_i(D_1 x_i + D_2 y_i)\} = \min_{x \in Q_1 \cap S} \{D_1 c^T x\} + \min_{y \in Q_2 \cap S} \{D_2 c^T y\} \quad (3.4)$$

When the capacity constraint is binding, the problem is more interesting. Lets set $B^T = (3, 5)$ (so that $B_1 \leq 3.2$ since $i = 1$ is the cheapest ingredient) and observe what happens. Figures (and corresponding Tables) 3.2 and 3.3 give local optima that are extreme points for this example.



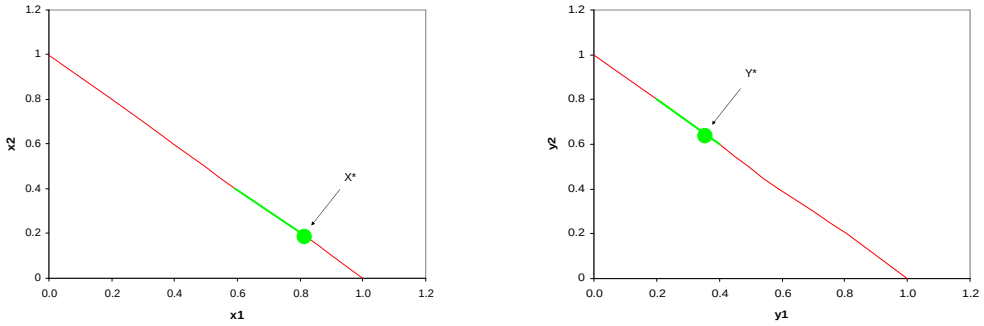
(a) x^* (b) y^*

Figure 3.2: Extreme point 1 for Bi-blend problem 1

Table 3.2: Extreme point 1 for Bi-blend instance 1

i	x_i	D_1	y_i	D_2	$Totali$	B_i
1	0.7	2	0.4	4	3	3
2	0.3	2	0.6	4	3	5

$$f^* = 3 * 1 + 3 * 2 = 9$$



(a) x^* (b) y^*

Figure 3.3: Extreme point 2 for Bi-blend problem 1

Table 3.3: Extreme point 2 for Bi-blend instance 1

i	x_i	D_1	y_i	D_2	$Totali$	B_i
1	0.8	2	0.35	4	3	3
2	0.2	2	0.65	4	3	5

$$f^* = 3 * 1 + 3 * 2 = 9$$

From Figures 3.2 and 3.3 we can conclude that there are infinitely many local optima. This is caused by the binding capacity constraint on the cheapest ingredient ($i = 1$). The sum of x_1 and y_1 will always be 3 (and accordingly the sum of x_2 and y_2 will always be 3), but these ingredients can be distributed among product 1 and 2 according to their individual linear constraints in any way we want. The above mentioned local optima are boundaries of possible recipes. The set of global optimal solutions is a line piece. More alternative solutions are given in Table 3.4. When we have a multidimensional problem, we can get a set of global solutions which covers a multidimensional area when all most cheapest ingredients are binding on the capacity constraint and there exists a feasible solution.

Table 3.4: Alternative solutions for Bi-blend instance 1

i	x_i	D_1	y_i	D_2	$Totali$	B_i
1	0.8	2	0.35	4	3	3
2	0.2	2	0.65	4	3	5
1	0.775	2	0.3625	4	3	3
2	0.225	2	0.6375	4	3	5
1	0.75	2	0.375	4	3	3
2	0.25	2	0.625	4	3	5
1	0.725	2	0.3875	4	3	3
2	0.275	2	0.6125	4	3	5
1	0.7	2	0.4	4	3	3
2	0.3	2	0.6	4	3	5

$$f^* = 3 * 1 + 3 * 2 = 9$$

3.6 Test cases

We construct bi-blend cases using quadratic constraints from Appendix A.1 to make restrictions on product 1 and product 2. We add a cost function and a capacity constraint. The demand D in all these cases will be $D^T = (1, 1)$. We made five 3-dimensional bi-blend test cases which are described in Table 3.5. In every row of the table a test case is given with their corresponding quadratic constraints on x and y and the used cost function and capacity constraint.

Table 3.5: Bi-blend test cases

test case	Q.constraints on x	Q.constraints on y	Cost function	Capacity vector B^T
1	1, 2	22, 24	(1.1, 1.7, 2.0)	(2, 2, 2)
2	1, 2	22, 24	(1.1, 1.7, 2.0)	(0.5, 2, 2)
3	1, 2	22, 24	(1.1, 1.7, 2.0)	(0.5, 0.75, 2)
4	1, 10	11, 15	(3.0, 2.0, 1.0)	(2, 2, 2)
5	1, 10	11, 15	(3.0, 2.0, 1.0)	(2, 2, 0.7)
6	1, 10	11, 15	(3.0, 2.0, 1.0)	(2, 0.7, 0.7)
7	22, 24	2, 6	(3.0, 2.0, 1.0)	(2, 2, 2)
8	22, 24	2, 6	(3.0, 2.0, 1.0)	(2, 2, 0.5)
9	22, 24	2, 6	(3.0, 2.0, 1.0)	(2, 0.9, 0.5)
10	22, 24	2, 6	(3.0, 2.0, 1.0)	(0.42, 2, 0.73)

In the next chapter, optimization algorithms are tested on the test cases we made on (bi-) blending.

Chapter 4

Solvers

There is no use of programming a new algorithm when there already exists a good optimization algorithm (solver) that can get the same results. In this chapter the potential of some existing solvers is examined. This chapter distinguishes two types of solvers; solvers that find local optima and solvers that find the global optimum. Instances of the (bi-) blending problem will be examined with different non-linear programming (NLP) solvers in GAMS and Matlab. In Table 4.1 the used solvers are given with their corresponding access language. Also the type of solver is indicated, which will be further explained in this chapter. There are more standard software packages which contain NLP solvers we did not consider.

Table 4.1: Solvers

Solver	Access language	Type of solver
BARON	GAMS	Global
LGO	GAMS	Global
OQNLP	GAMS	Multi start Local
MINOS	GAMS	Local
SNOPT	GAMS	Local
CONOPT	GAMS	Local
Fmincon	Matlab	Local
Fmincon-multi	Matlab	Multi start Local

4.1 Local solvers

Many standard solvers search for local optima. This means that given a starting point the solver tries to converge to a local optimum using a local search method. A local search may return a local optimum where there may also exist better optima. These solvers are tested to observe how difficult (bi-) blending problems are, and whether they might be solved by a local solver.

4.1.1 Multi start

In the case we want to find the global optimum of an instance while using a local solver, several local searches can be applied with new starting points for each search. In [9] a *multi start* method is used by placing a grid over the unit simplex to perform a local search from each point in the grid. Another way of doing many local searches is to use randomly chosen starting points. In this research this stochastic multi start method is applied in GAMS and Matlab. We can try to find all local optima using the multi start method and pick the best; however this does not guarantee to find the global optimum. "If after some calculation time no solution of the inequality problem has been found, it is not certain whether there exists one [5]." If on the other hand many local searches are done, the chance of not finding an existing global optimum becomes very small.

With use of a loop a multi start can be done while generating the starting points randomly for each new iteration (a local search) of the loop. In Matlab this is easy using a for-loop with randomly chosen starting points on the unit simplex (2.1). We can generate uniformly distributed starting points on the unit simplex using Algorithm (1) from [10]. For Fmincon-multi, 200 random starting points were chosen per instance. Figure 4.1 shows the 3-dimensional unit simplex with the notation used in Figure 2.4 with 200 randomly generated starting points.

Algorithm 1 : Generating uniform random points on the unit simplex

Funct S(n)

1. **for** ($i = 1 : n$)
   $a_i \sim Ne(1)$
2. **for** ($i = 1 : n$)
   $S_i = \frac{a_i}{\sum_{j=1}^{n} a_j}$
3. **return** S

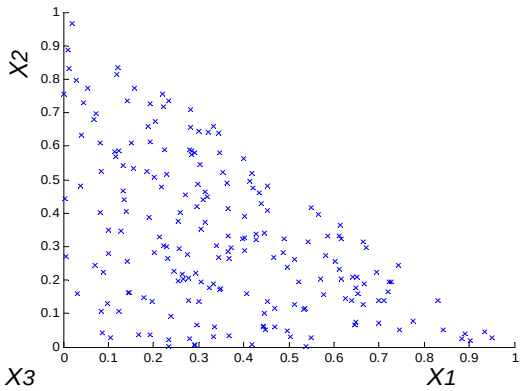


Figure 4.1: 200 uniformly distributed starting points on the unit simplex.

In GAMS it is not straightforward to implement a for-loop, That is why no manual loop is implemented in GAMS. However, in GAMS exists a stochastic multi start solver named OQNLP in which the local solvers of GAMS chosen for this research can be implemented. A disadvantage in using this multi start algorithm is that it generates random starting points over a box, where we would like to have random starting points which are on the unit simplex. We can set bounds on the starting points so it will be near the unit simplex with $0 \leq x \leq 1$ to give a solution to this problem.

4.2 Global solvers

Global solvers are solvers which use information from the whole search region to find the global optimum. With this global information the global character of a found optimum can be guaranteed. In this research GAMS/BARON and GAMS/LGO are solvers which claim to be global optimizers using different methods. This means that they should return the global optimum if a problem has one.

4.3 Performance indicators

4.3.1 Effectiveness

The performance of a solver can be measured with different indicators. The most important indicator for solvers is whether the solver is able to solve a given blending problem; this is called the effectiveness. The way solving ability is hard to measure. For instance a (bi-) blending problem can be feasible or infeasible. Before we can say a problem has a feasible solution or not, we have to find one or we have to prove there is none. If we do find a feasible optimum with a given solver we do not know whether it is global unless the solver can prove it. Only global solvers can make this guaranty, for example by exploring lower bounds over the whole search region. Multi-start local solvers can be very good as well in finding a global optimum, but where *globallity* of a solution might be assumed it is hard to prove it.

4.3.2 Efficiency

An important indicator on efficiency is the computing time a solver needs to solve an instance. This can be measured for example by the number of function evaluations, the number of iterations or simply in real time (given system specifications). In this research the number of iterations is used as a way to measure the efficiency of the solvers. Note that the number of iterations may not be a strong indicator, since for one solver a iteration may take longer than for the other. However, with this measure, we can make a distinction between easier or more difficult test cases.

4.4 Blending cases

To measure the performance of the chosen solvers we implemented the test cases from Section 2.6.

4.4.1 Experimental specifications

To be able to repeat all test cases in a fair way for all solvers, some rules are followed.

- Input parameters for the solver are untouched (default settings).
- For non-multi start solvers the starting point for x is the middle of the unit simplex:

$$x_i = \frac{1}{n}; \quad i = 1, \ldots, n \tag{4.1}$$

In Table 4.2 the best found optimum (f^*) for each test case is given with their corresponding recipe (x^*). These results come from the solvers which had found these optima. For the two and three dimensional test cases we can conclude that these optima are actually global using the plots (Appendix A) we had for all the 3-dimensional cases. For the 7 dimensional cases, the found optima were compared to the (global) optima from [4]; the results in our research were the same. Some of the cases contain local optima. These could be found by the multi start solvers.

Table 4.2: Best found optima for blending cases

Testcase	f^*	x^*
1	Infeasible	-
2	1.5578	(0.442, 0.558, 0)
3	Infeasible	-
4	Infeasible	-
5	1.5289	(0.580, 0.311, 0.109)
6	1.3599	(0.630, 0.245, 0.126)
7	2.214	(0.313, 0.099, 0.588)
8	2.350	(0.027, 0.623, 0.350)
9	1.633	(399, 0.569, 0.032)
10	1.230	(0, 0.230, 0.770)
11	1	(0, 1)
RumCoke	0.5668	(0.591, 0.342, 0.067)
Unispec1	110.8458	(0.413, 0, 0.468, 0, 0.119, 0, 0)
Unispec5b	115.2454	(0.189, 0, 0.048, 0.068, 0.694, 0, 0)

4.4.2 Local solutions

The multi start algorithm OQNLP is applied to find the local optima of the test cases. If specified by the user, OQNLP can return all local optima in a table. Since

in OQNLP the user can specify which local solver to use, all three GAMS local solvers from this research are implemented (MINOS, CONOPT and SNOPT). The results for the three different solvers were the same for all tested cases. In Table 4.3 the local optima for the multi modal cases are given indicated by a 'Rank'; this clarifies whether it is the best, second-best or third-best optimum. Note that a multi start algorithm may miss a local solution (by missing a starting point in a region of attraction), so there is no guarantee that there do not exist more optima. In this research however, we can judge whether there may exist more local optima for a test case since we have plots of all cases. OQNLP did not miss any of the local optima in the two and three dimensional test cases. All these local optima were also found using Fmincon-multi.

Table 4.3: Found optima with OQNLP for multi modal test cases

Testcase	Rank	f^*	x^*
7	1	2.214	(0.313, 0.099, 0.588)
	2	2.270	(0.580, 0.310, 0.110)
8	1	2.350	(0.027, 0.623, 0.350)
	2	2.599	(0, 0.401, 0.599)
	3	2.644	(0.159, 0.197, 0.644)
9	1	1.633	(0.399, 0.569, 0.032)
	2	2.279	(0.360, 0, 0.640)
10	1	1.230	(0, 0.230, 0.770)
	2	1.494	(0, 0.494, 0.506)
11	1	1	(0, 1)
	2	1.570	(0.572, 0.428)

4.4.3 Performance

All problems implemented in Fmincon and GAMS can be solved within a second. Fmincon-multi uses more real time (approximately 30 seconds per instance) but we can reduce this time by reducing the amount of local searches. In Table 4.4 the number of iterations needed to solve the instances are given for each GAMS solver per test case. For OQNLP the performance is always the same. The number of iterations is fixed at 1000 and the global optimum is always found. For this reason, OQNLP is not in the performance table. MINOS and SNOPT distinguish major and minor iterations and also give the number of function evaluations. That is why in their columns three values are given; the major iterations ('major'), the minor iterations ('minor') and the function evaluations ('fun').

In Table 4.5 the results for the Fmincon and Fmincon-multi are given. Fmincon (multi) distinguish iterations ('iterations') and function evaluations ('fun'). In the multi start column the number of times the global optimum is found is indicated by 'G!' since this could be determined with Matlab and it may be an interesting statistic. When an instance does not have local optima but neither found the global optimum 200 times, the solver converged to infeasible points on the rest of

the searches. That is why for some instances with only one optimum 'G!' is not 200.

When a performance is indicated with '2' the second-best optimum is found, not the global. When a performance is indicated with '3' the third-best optimum is found, not the global. When a performance is indicated with 'inf' the solver converges to an infeasible point while there exists a feasible optimum. For all performance measures (iterations) without an indication mark, the best solution from Table 4.2 is found.

Table 4.4: Iterations and convergence of GAMS solvers

Testcase	CONOPT	MINOS	SNOPT	LGO	BARON
		major, minor, fun	major, minor, fun		
1	10	6, 44, 57	6, 8, 11	4421	1
2	22	6, 10, 19	10, 10, 18	4	1
3	8	6, 61, 106	23, 12, 46	5537	1
4	8	11, 64, 85	17, 12, 34	5364	1
5	8	9, 19, 58	14, 5, 31	4	1
6	7	10, 24, 79	3, 2, 5	4	1
7	17^2	$7, 54, 96^{inf}$	$11, 13, 29^2$	4	1
8	30^2	$7, 64, 109^{inf}$	$8, 7, 25^3$	4^2	1
9	5	7, 16, 29	8, 7, 22	4	1
10	8^2	$9, 23, 71^2$	$3, 3, 5^2$	4	1
11	5^2	$14, 19, 61^2$	$3, 1, 5^2$	4	1
RumCoke	12	8, 10, 29	23, 9, 60	4	1
Unispec1	12	8, 34, 74	4, 10, 6	4	1
Unispec5b	11	9, 18, 40	7, 13, 16	4	1

Table 4.5: Iterations and convergence of Matlab solvers

Testcase	Fmincon iterations, fun	Fmincon-multi iterations, fun, G!
1	70, 301	-, -, -
2	5, 24	969, 4726, 174
3	70, 301	-, -, -
4	70, 301	-, -, -
5	6, 30	1137, 5564, 200
6	4, 20	1310, 5962, 180
7	$6, 29^2$	1144, 5589, 63
8	$8, 36^3$	1151, 5420, 85
9	5, 24	1072, 5356, 103
10	$4, 20^2$	1466, 6548, 81
11	$5, 18^2$	773, 2919, 85
RumCoke	10, 47	1444, 7891, 173
Unispec1	6, 56	985, 9480, 200
Unispec5b	7, 64	1259, 11689, 150

4.4.4 Conclusion

From these results we can conclude that three solvers were able to find the global optimum for all test cases. Namely, OQNLP, BARON and Fmincon-multi. A big surprise was the local solution LGO returned on test case 8. This makes this 'global' solver unreliable for finding global solutions.

The results for the local solvers were as expected; When a global solution lies in the region of attraction of the starting point (4.1), the global optimum is found. It shows us that many of the tested cases are easy to solve since it has only one optimum or the optimum was 'near' the middle of the unit simplex. Four out of 14 test cases have two local optima and one testcase has 3 local optima. All these cases were designed to be 'extreme'. The multi modal character of 3-dimensional blending problems is not high. For only one of the five extreme cases the global optimum was found using a local solver.

We can see from the results that the local solvers use different strategies since the found local optimum was not always the same for one test case. For test case 8, MINOS converged to an infeasible point, where CONOPT found the second best optimum and SNOPT and Fmincon found the third best optimum. When one would want to find the global optimum of a blending problem with several local optima, a local solver is not the appropriate optimization tool to use.

Except for LGO, the other global optimizers were successful. BARON uses only one iteration for each test case, but in this iteration a multi start is performed, which means that despite only using one iteration we cannot conclude that it is more efficient than OQNLP. Neither we can conclude that Fmincon-multi is less efficient for the higher 'real' time it uses, because the number of local searches is much higher than in OQNLP. Furthermore, it is easier to get information (like

'G!') from a more customized solver like Fmincon-multi than from the other two. Another big advantage of Fmincon-multi over OQNLP is that the starting points could be set on the unit simplex. Nonetheless, OQNLP also found all local optima. The ability to find local optima by OQNLP and Fmincon-multi was not seen as an advantage, since the goal of solving a blending case is to find the global optimum. For other researches this ability could give the multi start solvers an advantage over BARON. We conclude that BARON, OQNLP and Fmincon-multi are the best solvers for solving the tested instances.

4.5 Bi-blend cases

The bi-blend testcases from Table 3.5 were solved using the described GAMS solvers from Table 4.1.

Table 4.6: Best found solutions for bi-blend cases

Testcase	f^*	x^*	y^*
1	2.748	(0.591, 0.342, 0.067)	(0.594, 0.275, 0.131)
2	3.249	(0.481, 0.385, 0.134)	(0.019, 0.619, 0.362)
3	3.325	(0.481, 0.385, 0.134)	(0.019, 0.365, 0.616)
4	3.753	(0.251, 0, 0.749)	(0.375, 0.500, 0.125)
5	3.854	(0.228, 0.087, 0.685)	(0.327, 0.658, 0.015)
6	3.900	(0.245, 0.080, 0.675)	(0.355, 0.620, 0.025)
7	3.576	(0.008, 0.384, 0.609)	(0.415, 0.348, 0.238)
8	3.994	(0.027, 0.624, 0.348)	(0.466, 0.382, 0.152)
9	4.100	(0.178, 0.545, 0.277)	(0.422, 0.355, 0.223)
10	Infeasible	-	-

4.5.1 Local solutions

In Table 4.7 the local optima of the bi-blending cases are given. The table can be read in the same way as Table 4.3 but the 'Rank' can have a value 'infinite'. This means that there exist infinitely many alternative solutions, which is also depicted in column 'x^*'.

Table 4.7: Found optima with OQNLP for multi modal test cases

Testcase	Rank	f^*	x^*	y^*
3	infinite	3.325	Alternative optima	
4	1	3.753	(0.251, 0, 0.749)	(0.375, 0.500, 0.125)
	2	4.530	(0.500, 0.125, 0.375)	(0.375, 0.500, 0.125)
5	1	3.854	(0.228, 0.087, 0.685)	(0.327, 0.658, 0.015)
	2	4.530	(0.500, 0.125, 0.375)	(0.375, 0.500, 0.125)
6	infinite	3.900	Alternative optima	
9	infinite	4.100	Alternative optima	

4.5.2 Performance

All problems implemented in GAMS can be solved within a second. In table 4.4 the number of iterations needed to solve the problem are given for each solver. MINOS and SNOPT makes the distinction between major and minor iterations and also give the number of function evaluations. In Table 4.8 the number of iterations needed to solve the instances are given for each GAMS solver per test case. The results can be read in the same way as in Table 4.4.

Table 4.8: Iterations and convergence of GAMS solvers

Testcase	CONOPT	MINOS	SNOPT	LGO	BARON
		major, minor, fun	major, minor, fun		
1	16	10, 35, 88	19, 29, 42	4	1
2	18	11, 20, 47	6, 8, 10	4	1
3	16	7, 14, 24	12, 27, 24	4	1
4	12^2	$7, 67, 120^{inf}$	$24, 33, 76^{inf}$	4^2	15
5	8	$11, 33, 68^2$	$36, 28, 162^{inf}$	4^2	1
6	23	10, 30, 53	10, 24, 21	4	1
7	24	91, 264, 757	18, 10, 42	4	1
8	22	101, 283, 843	11, 6, 22	4	1
9	19	52, 88, 315	4, 2, 6	4	1
10	14	12, 23, 39	3, 6, 5	9527	1

4.5.3 Conclusion

From these results we can conclude that two solvers were able to find the global optimum for all test cases. Namely, OQNLP and BARON. LGO again returned local optima on test case 4 and 5. This makes this 'global' solver unreliable for finding global solutions.

Local solvers behaved in the same way as on the blending test cases. This means that they found the 'nearest' optimum. Again we can see from the results

that the local solvers use different strategies since the found local optimum was not always the same for one test case. For test case 4, MINOS and SNOPT converged to an infeasible point, where CONOPT found the second best optimum. When one would want to find the global optimum of a bi-blending problem with several local optima, a local solver is not the appropriate optimization tool to use.

Except for LGO, the other global optimizers were successful. In test case 4, BARON made some branching iterations; this was the only test case from this research which was 'difficult' for BARON in such a way that the global solution was not found (and verified) in the first iteration. OQNLP found all local optima and thus also found the global optimum to each test case. We conclude that BARON and OQNLP are the best solvers for solving the tested instances.

Chapter 5

Robustness

5.1 Introduction

We would like to develop an algorithm that is able to identify solutions that have an ϵ-robustness with respect to the quadratic requirements. From practical considerations, one can define robustness $R(x)$ of a design $x \in X$ (2.4) with respect to X as

$$R(x) = \max\{R \in \mathbb{R}^+ \mid (x + h) \in X,\ \forall h \in \mathbb{R}^n,\ \|h\| \leq R\}. \tag{5.1}$$

This ϵ-robustness concept is depicted in Figure 5.1. We want to find a solution to (bi-) blending while all solutions within a distance ϵ are feasible points as well. In the figure we can see that all points on the circle with a radius ϵ lie in the area $g(x) \leq 0$, and thus are feasible. The centre point of the circle is in this case an ϵ-robust point.

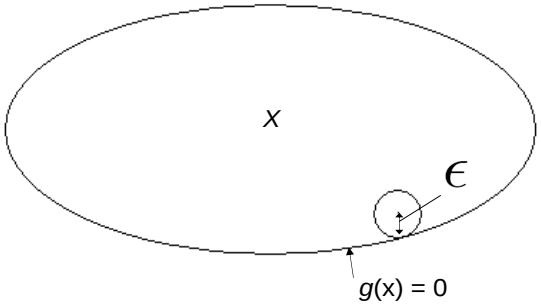


Figure 5.1: ϵ-robustness with respect to X.

Solving the robustness of a point is a global optimization problem which can have local optima. We can measure the minimum distance of a point z to one quadratic constraint g_p using (5.2).

$$\begin{aligned} & r_p(z) = \min\{\sqrt{\textstyle\sum_{i=1}^{n}(z_i - x_i)^2}\} \\ \text{s.t.} \quad & g_p(x) = 0, \quad p = 1, \ldots, P \end{aligned} \tag{5.2}$$

When we want to know the distance to the closest quadratic constraint in a global optimization problem we take the minimum over all distances r_p, $p = 1, \ldots, P$:

$$R(x) = \min_p\{r_p(z)\}, \quad p = 1, \ldots, P. \tag{5.3}$$

Notice that (5.2) is in general hard to solve; it may have several local optima and several KKT points that are not optimal [5].

5.2 ϵ -Robustness with respect to (bi-)blending

In this research all (bi-) blending problems were solved by standard solvers. It is difficult to combine the (bi-) blending problem while being robust simultaneously. In [4] a branch-and-bounds algorithm was designed to solve blending problems with a given ϵ-robustness. In our thesis however, the focus is on standard solvers which were really successful on non-robust (bi-) blending problems. That is why we try to reformulate the blending problem to be robust. In that case we would not have to change the optimization algorithm, but just the instances of the test case. In the next section this approach is explained in more detail.

5.3 Some (heuristic) approaches

The idea of reformulating the blending problem is quite straight forward. The idea is to lay circles with a radius $r = \epsilon$ on the feasible side of a quadratic constraint and draw a new quadratic constraint through the centres of the circles. When we could formulate the problem like this, we could use our standard solvers to find an ϵ-robust optimum. In Figure 5.2 this approach is shown. The dotted lines are the 'ϵ-robust constraints'.

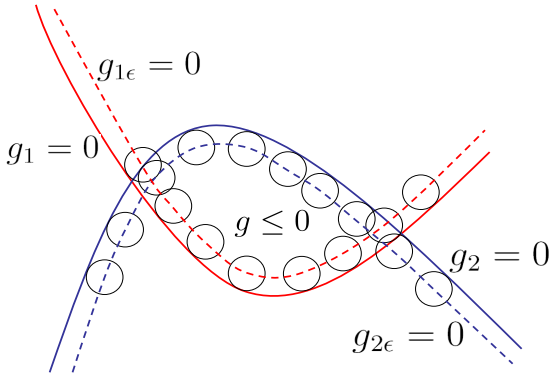


Figure 5.2: ϵ-robustness may be achieved by reformulation of quadratic constraints.

The suggested approach would be a perfect method for solving the robustness problem. The approach leads us to the next question. How can we compute an ϵ-robust constraint?

5.3.1 Reformulation of quadratic constraints

In Section 2.3 there was already a short introduction on quadratic functions. In this section we will give more insight on quadratic functions since we want to reformulate them. With this approach we have to distinguish definite and indefinite quadratic functions (see Figure 5.3). A quadratic function is definite when matrix A of the quadratic form (5.4) has only positive (positive definite) or negative eigenvalues (negative definite).

$$g_p(x) = x^T A_p x + b_p^T x + d_p \tag{5.4}$$

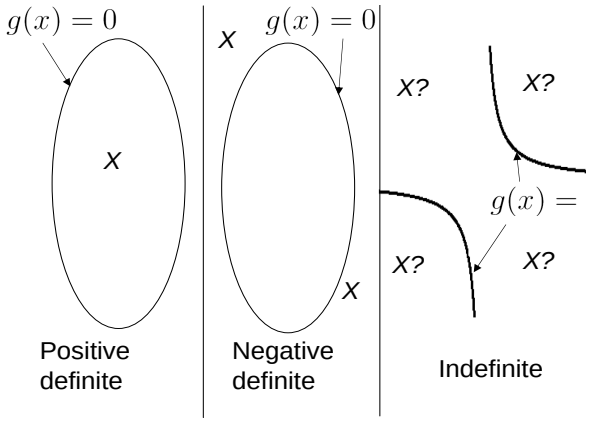


Figure 5.3: Definiteness of quadratic functions.

We tried to reformulate the problem by adjusting d (see (5.4)) or by adjusting its eigenvalues. With the suggested method it appears impossible to lay equally sized circles on an indefinite constraint. This is because the limit of an indefinite function is always its eigenvector. In Figure 5.4 we sketch that laying equally sized circles on a constraint may be difficult for an indefinite function using the methods under investigation.

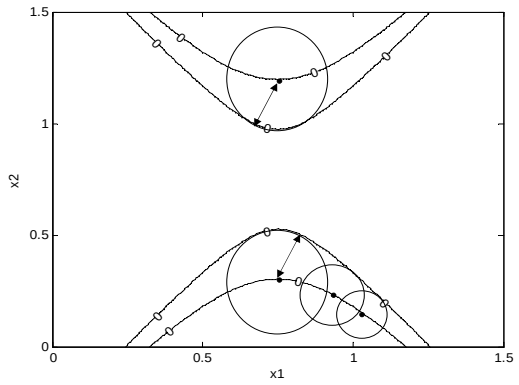


Figure 5.4: Circles with a different size on an indefinite function.

That is why the focus in this chapter was on (negative) definite quadratic functions. At first, we tried to change a quadratic constraint by adjusting the constant term d from (5.4). We did this for a 2-dimensional instance g_1 with parameters:

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}, b_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, d_1 = -1, d_1\epsilon = -0.5. \tag{5.5}$$

This gave us the result as shown in Figure 5.5 from which we can see that the found ellipse does not have equal distance from $g_{1\epsilon}$ to g_1 for all points on the ellipse. Since distance 'A' is much bigger than distance 'B' this is not the appropriate reformulation. In Figure 5.6 the cross section of Figure 5.5 on g_1 is given in direction v_2, with ϵ-robustness on v_2.

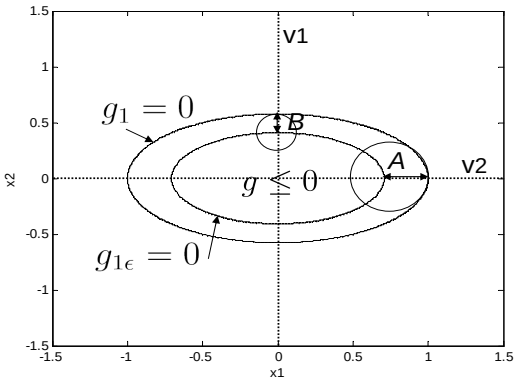


Figure 5.5: Reformulation by adjusting d.

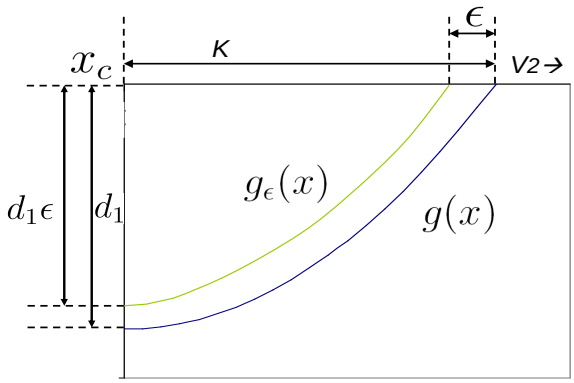


Figure 5.6: Reformulation by adjusting d.

[3] reformulates a quadratic function as (5.6).

$$g(x) = (x - x_c)^T A(x - x_c) + constant \tag{5.6}$$

in which x_c is the stationary point of $f(x)$ and *constant* is the function value of $g(x)$ in the origin ($g(x_c)$). One can derive that x_c is given by (5.7) and *constant* is given by (5.8):

$$x_c = -\frac{1}{2}A^{-1}b, \tag{5.7}$$

$$constant = d - \frac{1}{4}b^T A^{-1}b. \tag{5.8}$$

When we look at this notation we can see that only A determines the shape of the quadratic function, b determines (together with A) the origin and d only

affects the function value (together with A) with the addition of a constant. The *eigenvalues* and *eigenvectors* determine the 'shape'-characteristics of a quadratic function. The 'eigenvalue decomposition' of Matrix A is written in (5.9):

$$A = VDV^T, \tag{5.9}$$

with V representing a matrix with the eigenvector v_i on $V_{i.}$, and D a diagonal matrix with the eigenvalues λ_i on the diagonal.

When we decompose matrix A_1 from (5.5) we get:

$$\begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

From this decomposition we can conclude that on $v_1 = (1, 0)$, $\lambda_1 = 1$. For $v_2 = (0, 1)$, $\lambda_2 = 3$. With this data we can adjust λ_1 and λ_2, in such a way that the new λ's can get us ϵ-robustness where the eigenvectors are intersecting $g(x) = 0$.

In Figure 5.7 the cross section of Figure 5.5 on g_1 is given in direction v_2, with ϵ-robustness on v_2. k is the distance from x_c to $g(0)$ in the direction v_2.

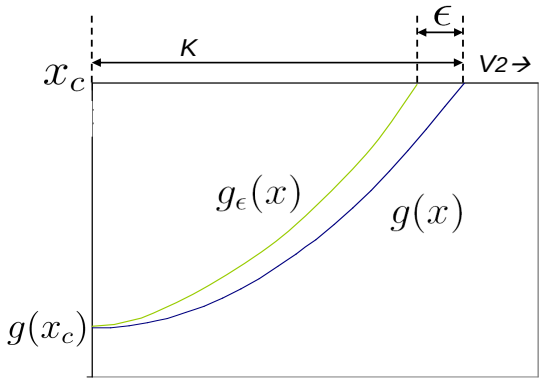


Figure 5.7: Cross section of g_1 on r_2

By adjusting λ we can get the ϵ-robustness on r. This is derived in (5.10).

$$\begin{aligned} &k = \sqrt{\tfrac{-g(x_c)}{\lambda}} \\ &\textbf{if } \text{pos.def: } k_\epsilon = k - \epsilon, \quad \textbf{if } \text{neg.def: } k_\epsilon = k + \epsilon \\ &\lambda_\epsilon = \tfrac{-g(x_c)}{k_\epsilon^2} \end{aligned} \tag{5.10}$$

When we do this for λ_1 and λ_2 from (5.5) we get the following robust matrix D_ϵ of adapted eigenvalues using $\epsilon = 0.05$:

$$D_\epsilon = \begin{pmatrix} 1.1080 & 0 \\ 0 & 3.5959 \end{pmatrix}.$$

With (5.9) we obtain a matrix $A_{1\epsilon}$ which is equal to D_ϵ since V is still the identity matrix. The 'regular' quadratic function g_1 is given by (5.5), while $g_{1\epsilon}$ is given by:

$$A_{1\epsilon} = \begin{pmatrix} 1.1080 & 0 \\ 0 & 3.5959 \end{pmatrix}, b_{1\epsilon} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, d_{1\epsilon} = -1. \tag{5.11}$$

Figure 5.8 shows us that $g_{1\epsilon}$ gives robust solutions on its eigenvectors.

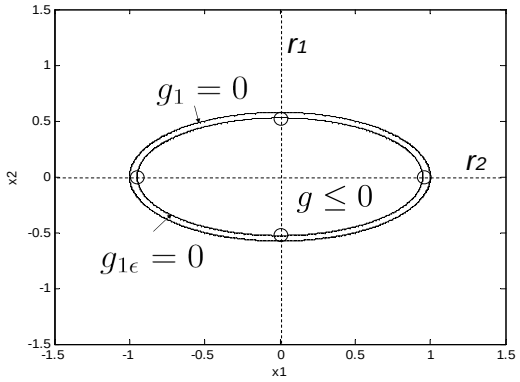


Figure 5.8: Robustness on the eigenvectors.

The robust constraint seems to have an equal distance to the non-robust constraint as we desired. We can test this by solving the global optimization problem (5.2) with points z on $g_{1\epsilon}$. We can generate z by (5.12). We have to choose a vector v which is **not** an eigenvector of $g(x)$.

$$\begin{aligned} \alpha &= \sqrt{\frac{-g(x_c)}{V^T A_\epsilon V}} \\ z &= \alpha V \end{aligned} \tag{5.12}$$

When we take $V^T = (1,\ 1)$ we obtain point $z^T = (0.4611,\ 0.4611)$ which lies on $g_{1\epsilon} = 0$. Substituting z in (5.2) together with g_1 gives a minimum robustness of 0.0490 when we solve this problem with Fmincon. This is smaller than the robustness we have obtained on the eigenvectors of 0.05. We can conclude that with this method we cannot generate an ellipse that is exactly ϵ-robust, but it comes close.

5.3.2 ϵ-robustness on X

In the previous section we tried to reformulate quadratic constraints but we did not take the unit simplex into account. However, the feasible area of a blending instance is defined by X (2.4), which also includes the unit simplex. We can see what happens with the robustness on X when we apply the eigenvalue approach from the previous section on test case 11 from Figure 2.6. In Figure 5.9 the

robustness on the unit simplex in test case 11 is shown and it seems to be a good approximation.

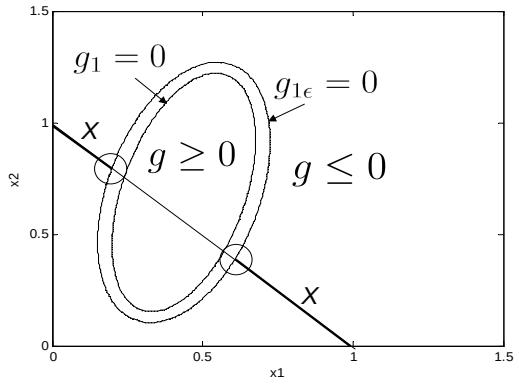


Figure 5.9: Robustness on test case 11.

Another example shows us that we should include the unit simplex in the reformulation. This is shown in Figure 5.10. The difference between ϵ_1 and ϵ_2 is really big and not desirable. If we would want to solve this problem, maybe we should project the quadratic functions on the unit simplex beforehand using the 'projection matrix' from [2] . This is however not further researched in this thesis. Although the heuristic approach from this research does not guarantee solving the robustness problem with respect to (bi-) blending it can be used to generate starting points for a procedure.

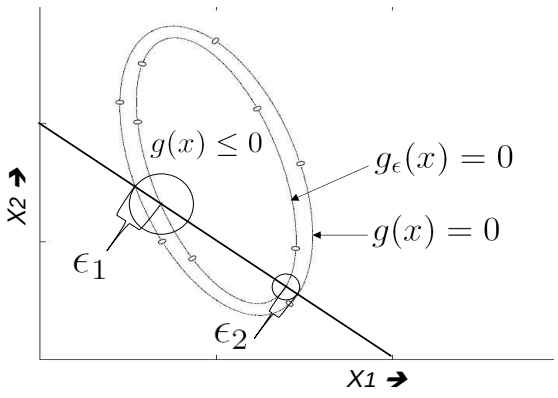


Figure 5.10: A big difference between ϵ_1 and ϵ_2.

5.4 Conclusion

Solving robustness on blending problems is a hard problem. Determining the robustness of a point is already a global optimization problem with several optima.

When we add this problem to the (bi-) blending problem it gets really hard. In standard software it seems impossible to solve the problem directly. This is why we tried to reformulate the quadratic constraints so we could use the old problem formulation as used in Chapter 2 and 3.

Changing the quadratic constraints by adjusting the constant term or the eigenvalue approach is not appropriate for reformulating indefinite quadratic functions. However, changing eigenvalues for (negative) definite functions seemed to work. However, we found a polynomial that was ϵ-robust on its eigenvectors, but not on the rest of its points.

We neither took the unit simplex into account. In order to reformulate a blending problem you should consider the unit simplex at the same time since the feasible area is defined as $X = Q \cap S$. This may be done using a projection matrix.

Chapter 6

Conclusion

The bi-blending problem with quadratic constraints is a global optimization problem, which as all global optimization problems has a multi modal character. This is however not the case for all bi-blending problems. Before we could determine the multi modality of bi-blend problems we investigated the multi modality of single blending problems. We concluded that a blending problem can have local optima due to two characteristics.

1. Existance of several feasible compartments
2. Non-convex compartment in combination with an appropriate cost function.

These characteristics also apply to bi-blending problems. In the bi-blending problem the capacity constraint influences the multi modality.

- When an instance of the bi-blending problem has a capacity constraint which is binding on all of the cheapest ingredients, the problem has alternative solutions for the global optimum.

The potential of existing optimization algorithms is tested on (bi-) blending problems. Solvers in Matlab and GAMS are tested and evaluated. We tested global solvers as well as (multi start) local solvers. The solvers which claimed to be global solvers were LGO and BARON with GAMS as access language. The local solvers were MINOS, SNOPT and CONOPT in GAMS and Fmincon in Matlab. These local solvers were used to examine the multi modal character of (bi-) blending problems; but also for examining the potential of local solvers on (bi-) blending problems. The local solvers could be used as a global optimizer with a stochastic multi start routine. For GAMS the existing multi start routine OQNLP could be used, which generated random starting points on a box, which can be determined by lower and upper bounds supplied by the user. For Matlab a for-loop was used to uniformly generate starting points on the unit simplex.

- Local solvers solved the supplied test cases to the 'nearest' local optimum, which in some cases was not the global optimum. This makes local solvers inappropriate for solving multi modal (bi-) blending instances.

- BARON, OQNLP and Fmincon with a multi start routine returned all global optima for the blending test cases. BARON and OQNLP also returned all global optima for the bi-blending test cases. The self proclaimed global solver LGO found local optima on blending cases and bi-blending test cases which makes it a unreliable solver.

Solving the robustness on blending problems is difficult. Determining the robustness of a point is already a global optimization problem with several optima. We could not solve this problem through standard software, but tried to reformulate the problem by changing the quadratic term in the quadratic constraints of the problem in order to solve it with standard optimization algorithms. The reformulation was only tried for definite quadratic functions. We did this by changing the eigenvalues of the quadratic form but we could only accomplish robustness on the eigenvalues of its function and not on the other points of the polynomial. Neither was the effect on the unit simplex taken into account which gave problems in the end. Reformulation of (bi-) blending problems is a hard procedure; in this research we were not able to solve the robust (bi-) blending problem with reformulation.

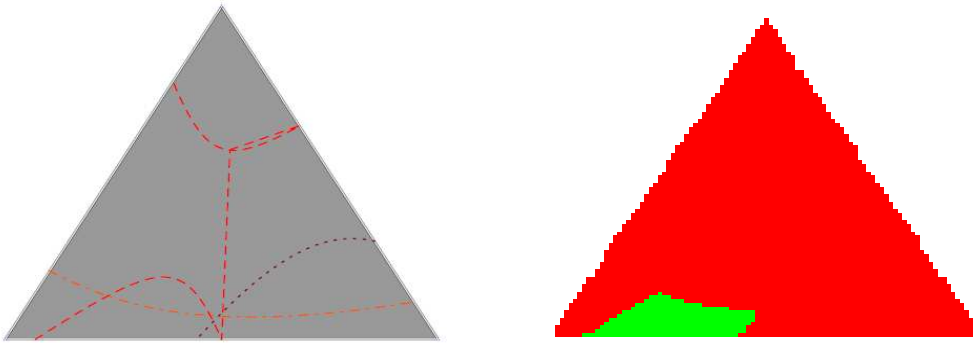
(Bi-) blending problems can be solved with standard software; the robustness problem however not. The reformulation of the quadratic constraints can be used for future research. With this procedure one can produce useful starting points for solving (bi-) blending problems.

Appendix A

Testcases

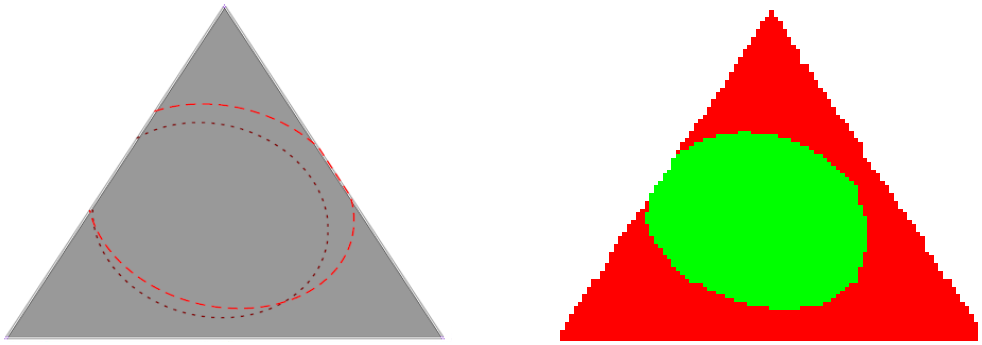
Table A.1: 3-Dimensional testcases

Testcase	Costs	Quadratic constraint
1	1.0, 2.0, 3.0	16, 17, 18
2	1.0, 2.0, 3.0	15, 19, 21
3	1.0, 2.0, 3.0	12, 13, 21
4	1.0, 2.0, 3.0	12, 21, 22
5	1.0, 2.0, 3.0	22, 21
6	1.1, 1.7, 2.0	3, 4, 5, 6, 7
7	3.0, 1.0, 2.0	2, 12, 13
8	2.0, 2.0, 3.0	4, 8, 12, 13
9	1.1, 1.7, 2.0	11, 14, 15
10	3.0, 2.0, 1.0	7, 8, 9, 10
11	2.0, 1.0	23
Rumcoke	0.1, 0.7, 4.0	1, 2



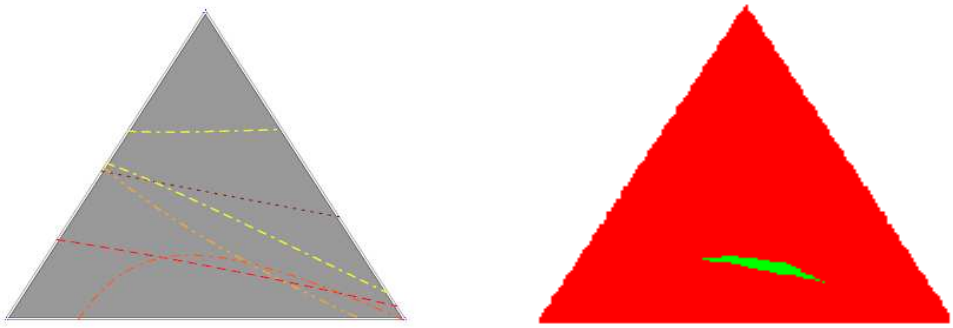
(a) Constraints. (b) Feasible area.

Figure A.1: Testcase 2.



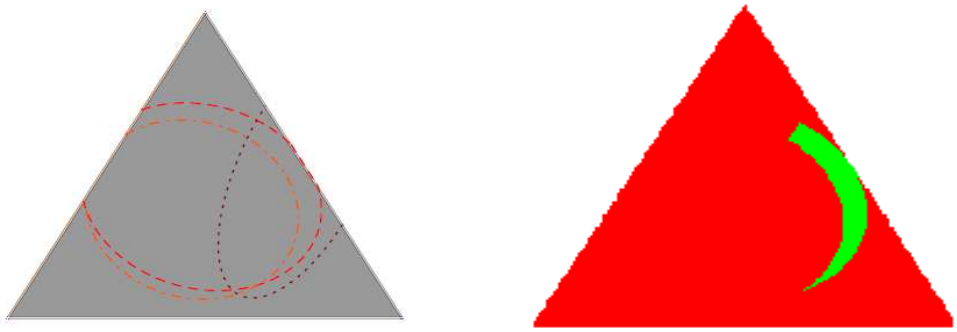
(a) Constraints. (b) Feasible area.

Figure A.2: Testcase 5.



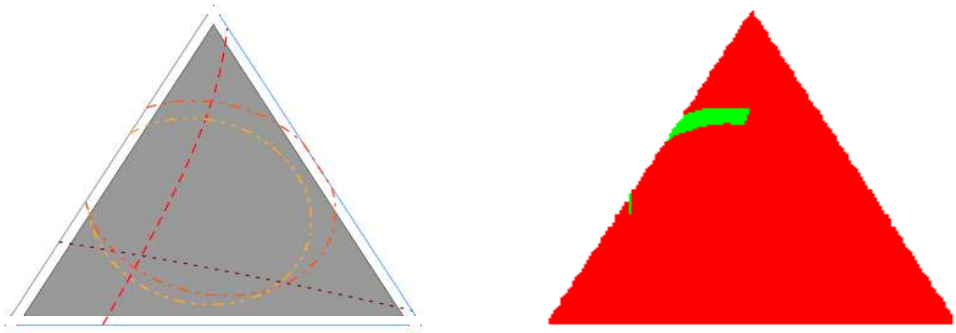
(a) Constraints. (b) Feasible area.

Figure A.3: Testcase 6.



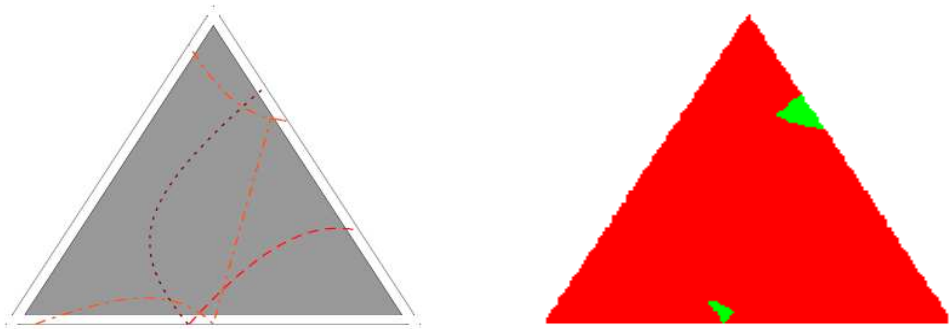
(a) Constraints. (b) Feasible area.

Figure A.4: Testcase 7.



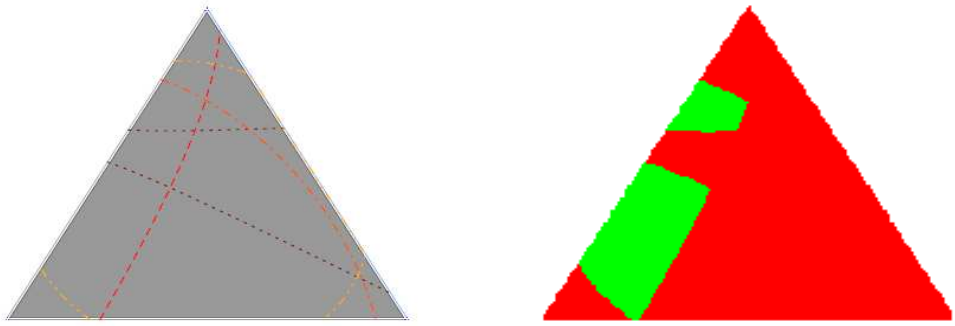
(a) Constraints. (b) Feasible area.

Figure A.5: Testcase 8.



(a) Constraints. (b) Feasible area.

Figure A.6: Testcase 9.



(a) Constraints. (b) Feasible area.

Figure A.7: Testcase 10.

A.1 Quadratic constraints

Quadratic constraint($g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0; \quad i = 1, \ldots, 27$):

$$A_1[3 \times 3] = \begin{pmatrix} 0 & -16 & 0 \\ -16 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$b_1[3 \times 1] = \begin{pmatrix} 8 \\ 8 \\ 0 \end{pmatrix}$$

$$d_1 = -1$$

$$A_2[3 \times 3] = \begin{pmatrix} 10 & 0 & 2 \\ 0 & 0 & 0 \\ 2 & 0 & 2 \end{pmatrix}$$

$$b_2[3 \times 1] = \begin{pmatrix} -12 \\ 0 \\ -4 \end{pmatrix}$$

$$d_2 = 3.7$$

$$A_3[3 \times 3] = \begin{pmatrix} 0.001 & -0.001 & 0.0085 \\ -0.001 & 0.008 & -0.0105 \\ 0.0085 & -0.0105 & -0.021 \end{pmatrix}$$

$$b_3[3 \times 1] = \begin{pmatrix} -0.0145 \\ -0.0205 \\ 0.073 \end{pmatrix}$$

$$d_3 = -0.0165$$

$$A_4[3 \times 3] = \begin{pmatrix} -0.004 & 0.0005 & 0.002 \\ 0.0005 & -0.001 & -0.003 \\ 0.002 & -0.003 & 0.014 \end{pmatrix}$$

$$b_4[3 \times 1] = \begin{pmatrix} 0.0155 \\ 0.0515 \\ -0.121 \end{pmatrix}$$

$$d_4 = -0.006$$

$$A_5[3 \times 3] = \begin{pmatrix} 20.605 & -5.087 & -10.9885 \\ -5.087 & 32.003 & -43.476 \\ -10.9885 & -43.476 & -81.278 \end{pmatrix}$$

$$b_5[3 \times 1] = \begin{pmatrix} 0.1995 \\ -0.097 \\ 126.7685 \end{pmatrix}$$

$$d_5 = -20.5063$$

$$A_6[3 \times 3] = \begin{pmatrix} 0.766 & -0.1205 & 2.4735 \\ -0.1205 & 0.528 & 1.9835 \\ 2.4735 & 1.9835 & -7.822 \end{pmatrix}$$

$$b_6[3 \times 1] = \begin{pmatrix} -2.432 \\ -15.191 \\ 10.712 \end{pmatrix}$$

$$d_6 = 3.21125$$

$$A_7[3 \times 3] = \begin{pmatrix} 116.75 & -3.09 & 168.553 \\ -3.09 & -67.424 & 515.114 \\ 168.553 & 515.114 & -845.215 \end{pmatrix}$$

$$b_7[3 \times 1] = \begin{pmatrix} -287.43 \\ -645.926 \\ 354.537 \end{pmatrix}$$

$$d_7 = 115.0953$$

$$A_8[3 \times 3] = \begin{pmatrix} 1.0 & 3.0 & -0.5 \\ 3.0 & -5.0 & -3.5 \\ -0.5 & -3.5 & -2.0 \end{pmatrix}$$

$$b_8[3 \times 1] = \begin{pmatrix} 0.832 \\ 0.832 \\ 0.832 \end{pmatrix}$$

$$d_8 = 0.968$$

$$A_9[3 \times 3] = \begin{pmatrix} 2.0 & -1.5 & 1.0 \\ -1.5 & 1.0 & -1.0 \\ 1.0 & -1.0 & 3.0 \end{pmatrix}$$

$$b_9[3 \times 1] = \begin{pmatrix} 0.12 \\ 0.12 \\ 0.12 \end{pmatrix}$$

$$d_9 = -1.60$$

$$A_{10}[3 \times 3] = \begin{pmatrix} 4.0 & -1.5 & -1.5 \\ -1.5 & 4.0 & -2.5 \\ -1.5 & -2.5 & 4.0 \end{pmatrix}$$

$$b_{10}[3 \times 1] = \begin{pmatrix} -0.026 \\ -0.026 \\ -0.026 \end{pmatrix}$$

$$d_{10} = -2.141$$

Appendix A: Testcases

$$A_{11}[3 \times 3] = \begin{pmatrix} -4.3 & 1.0 & -3.0 \\ 1.0 & 1.0 & -0.5 \\ -3.0 & -0.5 & 2.5 \end{pmatrix}$$

$$b_{11}[3 \times 1] = \begin{pmatrix} -0.193 \\ -0.193 \\ -0.193 \end{pmatrix}$$

$$d_{11} = 0.193$$

$$A_{12}[3 \times 3] = \begin{pmatrix} 4.0 & -1.0 & -2.0 \\ -1.0 & 5.0 & -3.0 \\ -2.0 & -3.0 & 4.0 \end{pmatrix}$$

$$b_{12}[3 \times 1] = \begin{pmatrix} 1.05 \\ 1.05 \\ 1.05 \end{pmatrix}$$

$$d_{12} = -2.1052$$

$$A_{13}[3 \times 3] = \begin{pmatrix} -4.0 & 1.5 & 1.5 \\ 1.5 & -4.0 & 2.5 \\ 1.5 & 2.5 & -4.0 \end{pmatrix}$$

$$b_{13}[3 \times 1] = \begin{pmatrix} -1.48 \\ -1.48 \\ -1.48 \end{pmatrix}$$

$$d_{13} = 2.36$$

$$A_{14}[3 \times 3] = \begin{pmatrix} 1.0 & 3.0 & -0.5 \\ 3.0 & -5.0 & -3.5 \\ -0.5 & -3.5 & -2.0 \end{pmatrix}$$

$$b_{14}[3 \times 1] = \begin{pmatrix} 0.83 \\ 0.83 \\ 0.83 \end{pmatrix}$$

$$d_{14} = -0.59$$

$$A_{15}[3 \times 3] = \begin{pmatrix} 4.0 & -1.0 & 0.5 \\ -1.0 & 1.0 & 3.5 \\ 0.5 & 3.5 & 0 \end{pmatrix}$$

$$b_{15}[3 \times 1] = \begin{pmatrix} 18.75 \\ 18.75 \\ 18.75 \end{pmatrix}$$

$$d_{15} = -19.5$$

$$A_{16}[3 \times 3] = \begin{pmatrix} -2.0 & -0.5 & 0.5 \\ -0.5 & 2.0 & -2.5 \\ 0.5 & -2.5 & 0 \end{pmatrix}$$

$$b_{16}[3 \times 1] = \begin{pmatrix} 10.89 \\ 10.89 \\ 10.89 \end{pmatrix}$$

$$d_{16} = -9.69$$

$$A_{17}[3 \times 3] = \begin{pmatrix} -5.0 & 1.0 & -3 \\ 1 & 1 & -0.5 \\ -3 & -0.5 & 3 \end{pmatrix}$$

$$b_{17}[3 \times 1] = \begin{pmatrix} 6.45 \\ 6.45 \\ 6.45 \end{pmatrix}$$

$$d_{17} = -2.65$$

$$A_{18}[3 \times 3] = \begin{pmatrix} 5.0 & -1.0 & 1.0 \\ -1.0 & 0 & -2.5 \\ 1.0 & -2.5 & 1 \end{pmatrix}$$

$$b_{18}[3 \times 1] = \begin{pmatrix} 14.71 \\ 14.71 \\ 14.71 \end{pmatrix}$$

$$d_{18} = -14.88$$

$$A_{19}[3 \times 3] = \begin{pmatrix} 1.0 & 3.0 & -0.5 \\ 3.0 & -5.0 & -3.5 \\ -0.5 & -3.5 & -2.0 \end{pmatrix}$$

$$b_{19}[3 \times 1] = \begin{pmatrix} 0.83 \\ 0.83 \\ 0.83 \end{pmatrix}$$

$$d_{19} = -0.95$$

$$A_{20}[3 \times 3] = \begin{pmatrix} -4.0 & -3.0 & -0.5 \\ -3.0 & -4.0 & -2.5 \\ -0.5 & -2.5 & 3.0 \end{pmatrix}$$

$$b_{20}[3 \times 1] = \begin{pmatrix} 3.90 \\ 3.90 \\ 3.90 \end{pmatrix}$$

$$d_{20} = -0.69$$

$$A_{21}[3 \times 3] = \begin{pmatrix} -2.0 & -0.5 & 0.5 \\ -0.5 & 2.0 & -2.5 \\ 0.5 & -2.5 & 0 \end{pmatrix}$$

$$b_{21}[3 \times 1] = \begin{pmatrix} 2.08 \\ 2.08 \\ 2.08 \end{pmatrix}$$

$$d_{21} = 1.53$$

$$A_{22}[3 \times 3] = \begin{pmatrix} -2.0 & -0.5 & 0.5 \\ -0.5 & 2.0 & -2.5 \\ 0.5 & -2.5 & 0 \end{pmatrix}$$

$$b_{22}[3 \times 1] = \begin{pmatrix} 2.08 \\ 2.08 \\ 2.08 \end{pmatrix}$$

$$d_{22} = -0.87$$

$$A_{23}[2 \times 2] = \begin{pmatrix} -5.0 & 1.0 \\ 1.0 & -1.0 \end{pmatrix}$$

$$b_{23}[2 \times 1] = \begin{pmatrix} 3.0 \\ 0.5 \end{pmatrix}$$

$$d_{23} = -0.6$$

UniSpec1

Dimension = 7; Raw material cost = $(114, 115, 107, 127, 115, 106, 108)$
Linear constraint:
$h_1(x) = 0.1493x_1 + 0.6927x_2 + 0.4643x_3 + 0.7975x_4 + 0.5967x_5 + 0.6235x_6 + 0.5284x_7 \geq 0.35$

Appendix A: Testcases

Quadratic constraints ($g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0; \quad i = 1, 2, 3$):

$$A_1[7\times7] = \begin{pmatrix} -1.473 & 8.215 & -27.204 & 46.119 & 2.059 & -11.929 & -12.768 \\ 8.215 & 37.733346 & 5.127 & 95.691 & 34.954 & 20.165 & 19.445 \\ -27.204 & 5.127 & -21.743 & 36.843 & -7.126 & 4.029 & -4.152 \\ 46.119 & 95.691 & 36.843 & 189.643 & 93.359 & 52.904 & 54.802 \\ 2.059 & 34.954 & -7.126 & 93.356 & 31.885 & 7.528 & 10.248 \\ -11.929 & 20.165 & 4.029 & 52.904 & 7.528 & 11.951 & 10.964 \\ -12.768 & 19.445 & -4.152 & 54.802 & 10.248 & 10.964 & 7.197 \end{pmatrix}$$

$$b_1[7 \times 1] = \begin{pmatrix} 4.5675 \\ 34.7289 \\ 70.5707 \\ -82.2761 \\ 29.3169 \\ 71.0818 \\ 63.7614 \end{pmatrix}$$

$$d_1 = -35$$

$$A_2[7\times7] = \begin{pmatrix} 1.35 & -4.41 & 17.60 & -92.45 & 2.74 & -29.94 & -14.05 \\ -4.41 & -39.13 & -6.11 & -126.38 & -29.81 & -63.42 & -43.97 \\ 17.60 & -6.11 & 15.45 & -76.60 & 5.93 & -44.05 & -20.54 \\ -92.45 & -126.38 & -76.60 & -240.64 & -117.46 & -125.18 & -114.98 \\ 2.74 & -29.81 & 5.93 & -117.46 & -22.90 & -47.37 & -30.68 \\ -29.94 & -63.42 & -44.05 & -125.18 & -47.37 & -73.39 & -73.99 \\ -14.05 & -43.97 & -20.54 & -114.98 & -30.68 & -73.99 & -55.33 \end{pmatrix}$$

$$b_2[7 \times 1] = \begin{pmatrix} -2.1232 \\ -9.0403 \\ -42.2072 \\ 190.5292 \\ -9.9529 \\ 1.8162 \\ 5.1622 \end{pmatrix}$$

$$d_2 = 10$$

$$A_3[7\times 7] = \begin{pmatrix} -0.670 & 4.284 & -12.837 & 23.708 & 1.677 & -8.964 & -4.859 \\ 4.284 & 21.380 & -1.188 & 28.990 & 13.216 & 17.177 & 16.620 \\ -12.837 & -1.189 & -21.376 & 9.841 & -7.298 & -10.043 & -8.981 \\ 23.708 & 28.990 & 9.841 & 49.385 & 25.574 & 15.561 & 21.666 \\ 1.677 & 13.216 & -7.298 & 25.574 & 8.419 & 4.149 & 6.595 \\ -8.965 & 17.177 & -10.043 & 15.561 & 4.149 & 1.090 & 6.292 \\ -4.859 & 16.620 & -8.981 & 21.666 & 6.594 & 6.292 & 5.906 \end{pmatrix}$$

$$b_3[7 \times 1] = \begin{pmatrix} 0.7097 \\ -13.0982 \\ 27.5078 \\ -49.1608 \\ -7.3725 \\ 33.6731 \\ 11.3136 \end{pmatrix}$$

$$d_3 = -2$$

UniSpec5b

Dimension = 7; Same raw material cost and similar quadratic requirements as UniSpec1
Quadratic constraints ($g_i(x) = x^T A_i x + b_i^T x + d_i \leq 0; \quad i = 4, 5, 6, 7$):

$$A_4 = -A_1; b_4 = -B_1; d_4 = 45$$

$$A_5 = -A_2; b_5 = -B_2; d_5 = -21$$

$$A_6[7 \times 7] = \begin{pmatrix} 0.0 & -11.556 & -1.114 & 14.690 & -11.411 & 0.121 & -0.150 \\ -11.556 & -3.316 & -2.116 & 7.313 & -8.800 & 19.897 & 9.051 \\ -1.114 & -2.116 & 4.728 & 16.250 & -4.535 & 18.319 & 11.537 \\ 14.690 & 7.313 & 16.250 & 40.428 & 9.766 & 21.512 & 15.266 \\ -11.412 & -8.800 & -4.535 & 9.766 & -10.165 & 10.088 & 1.889 \\ 0.121 & 19.897 & 18.319 & 21.511 & 10.088 & 28.569 & 27.239 \\ -0.150 & 9.051 & 11.537 & 15.266 & 1.889 & 27.239 & 19.965 \end{pmatrix}$$

$$b_6[7 \times 1] = \begin{pmatrix} 1.7278 \\ 23.5166 \\ 5.6724 \\ -32.0798 \\ 19.0154 \\ 16.5074 \\ 7.31003 \end{pmatrix}$$

$$d_6 = -5$$

$$A_7 = A_3; b_7 = B_3; d_7 = -1$$

Appendix B

GAMS code

B.1 Rumcoke

```
SETS
       I      / i1, i2, i3 /
       P   constraint / A, B /;

alias (I,J);

parameter
       L(I) lower bounds
         /   i1 0, i2 0, i3 0 /

       U(I) upper bounds
         /   i1 1, i2 1, i3 1 /

       D(P) constance D for function P
         /    A -1, B 3.7/


       C(I) cost per ingredient I
         /  i1     0.1
            i2     0.7
            i3     4   /;

table
         MATRIX(P,I,J)  A Matrix for function P

                          i1       i2      i3
            A.i1          0       -16      0
            A.i2         -16        0      0
            A.i3          0         0      0
            B.i1          10        0      2
            B.i2          0         0      0
```

```
           B.i3        2          0      2  ;


table
       B(I,P)  B coefficient I for function P
                           A         B
              i1           8       -12
              i2           8         0
              i3           0        -4;



Variables X
          Z ;


Positive Variable X;
X.lo(I)=L(I);
X.up(I)=U(I);









EQUATIONS
       COST           define objective functioN
       CONSTRAINT(P) quadratic constraint p
       UNITSIMPLEX   unit simplex ;

COST ..         Z  =E=  SUM((I), C(I)*X(I)) ;
CONSTRAINT(P) ..   SUM((I,J), X(I)*MATRIX(P,I,J)*X(J)) + SUM((I), B(I,P)*X(I)) +
UNITSIMPLEX ..   SUM(I, X(I))  =E=  1;

MODEL rumcoke /ALL/ ;


option nlp=baron;
rumcoke.optfile = 1 ;
SOLVE rumcoke USING NLP MINIMIZING Z ;
```

B.2 Bi-blend case 1

```
SETS
       I      / i1, i2, i3 /
       P   constraint / A, B /;

alias (I,J);

parameter
       L(I) lower bounds
         /   i1 0, i2 0, i3 0 /

       U(I) upper bounds
         /   i1 1, i2 1, i3 1 /

       D(P) constance D for function P
         /    A -1, B 3.7/


       C(I) cost per ingredient I
         /  i1     0.1
            i2     0.7
            i3     4   /;

table
         MATRIX(P,I,J)  A Matrix for function P

                          i1       i2      i3
            A.i1          0       -16      0
            A.i2         -16       0       0
            A.i3          0        0       0
            B.i1          10       0       2
            B.i2          0        0       0
            B.i3          2        0       2  ;


table
       B(I,P)  B coefficient I for function P
                           A        B
              i1           8       -12
              i2           8        0
              i3           0        -4;



Variables X
```

```
          Z ;


Positive Variable X;
X.lo(I)=L(I);
X.up(I)=U(I);









EQUATIONS
       COST          define objective functioN
       CONSTRAINT(P) quadratic constraint p
       UNITSIMPLEX   unit simplex ;

COST ..        Z  =E=  SUM((I), C(I)*X(I)) ;
CONSTRAINT(P) ..   SUM((I,J), X(I)*MATRIX(P,I,J)*X(J)) + SUM((I), B(I,P)*X(I)) +
UNITSIMPLEX ..   SUM(I, X(I))  =E=  1;

MODEL rumcoke /ALL/ ;


option nlp=baron;
rumcoke.optfile = 1 ;
SOLVE rumcoke USING NLP MINIMIZING Z ;
```

Appendix C

Matlab code

C.1 Rumcoke

C.1.1 $f(x)$

```
function [f] = f(x)

C = [0.1 0.7 4.0];

[f] = C*x
```

C.1.2 $g(x)$

```
function [c, ceq] = gcon(x)

C1 = [0 -16 0 ; -16 0 0; 0 0 0];
C2 = [10 0 2; 0 0 0; 2 0 2];

b1 = [8 8 0];
b2 = [-12 0 -4];

c1 = -1;
c2 = 3.7;

f(1) = x'*C1*x + b1*x + c1;
f(2) = x'*C2*x + b2*x + c2;

c = f;
ceq = [];
```

C.1.3 Fmincon single start

```
m = 3;
xstart = 1/3 * ones(m,1);
LB = zeros(1,m);
```

```
UB = ones(1,m);
Aeq = UB;
Beq = 1;
[X,FVAL,EXITFLAG,OUTPUT]=fmincon(@(x) f(x),xstart,[],[],Aeq,Beq,
LB,UB,@(x)gcon(x))
```

C.1.4 Fmincon-multi

```
t=cputime;
m = 3;
LB = zeros(1,m);
UB = ones(1,m);
Aeq = UB;
Beq = 1;
for i = 1:200
mu = 1;
a = exprnd(mu,m,1);
som = sum(a);
    for j = 1:m
        mstart(j,i) = a(j)/som;
    end
end
mstart;
A = [mstart(1,:);mstart(2,:)];
scatter(A(1,:),A(2,:),'x');
Funeval = 0;
Iter = 0;
j = 1;
for i = 1:200
[X,FVAL,EXITFLAG,OUTPUT]=fmincon(@(x) f(x),mstart(:,i),[],[],
Aeq,Beq,LB,UB,@(x)gcon(x));
if EXITFLAG == 1
    Xt(j,:) = X';
    FVALt(j) = FVAL;
    Funeval = Funeval + OUTPUT.funcCount;
    Iter = Iter + OUTPUT.iterations;
    j = j+1;
end
end
A = [FVALt' Xt]
time = cputime-t
Funeval
Iter
goodanswer = length(A(:,1))
```

Bibliography

[1] A. Boon, M. Kirchmann, G. Nieuwenhoven, M. Smetsers. *Msc thesis Protocol Rules and Regulations*, 2005.

[2] L. G. Casado, E. M. T. Hendrix, and I. García. Infeasibility spheres for finding robust solutions of blending problems with quadratic constraints. *Journal of Global Optimization*, 39:557–593, 2007.

[3] G. D. H. Claassen, Th. H. B. Hendriks, , E. M. T. Hendrix. Nonlinear programming. *Decision science, theory and applications*, 285–339, 2007.

[4] E. M. T. Hendrix, L. G. Casado, and I. García. The semi-continous quadratic mixture design problem. Description and branch-and-bound approach. *European Journal of Operational Research*, 191:803–815, 2008.

[5] E. M. T. Hendrix. *Global optimization at work.* Wageningen University, Wageningen, 1998.

[6] E. M. T. Hendrix and J. D. Pinter. An Application of Lipschitzian Global Optimization to product design. *Journal of Global Optimization*, 1:389-401, 1991.

[7] F. J. Lopez. *Division multiple en algoritmos divide y venceras aplicados a problemas de mezcla.* University of Almeria, 2008.

[8] B. A. Murtagh, W. Murray, R. Raman, M. A. Saunders, P. E. Gill, and E. Kalvelagen. *GAMS/MINOS: A solver for large-scale nonlinear optimization problems*, 2002.

[9] F. Niewerth. *Instanties voor het kwadratische mixture design probleem*, Wageningen University, 2007.

[10] N. Olieman. *Methods for Robustness Programming*, Wageningen University, 2008. ISBN 978-90-8504-876-3.

[11] D. G. Ullman. *Making Robust Decisions.* Trafford Publishing, 2006.