Optimizing a Nonlinear Function Using Rosen's Projection Algorithm with Infeasible Starting Points

By
Erik van der Klooster (209391)

A thesis submitted in partial fulfillment of the requirements for the degree of
Master in Business Analytics and Operations Research

Tilburg School of Economics and Management
Tilburg University

Supervised by:
prof.dr.ir. R. Sotirov
dr. J.C. Vera Lizcano

Date: August 15, 2017

ABSTRACT

This thesis presents an algorithm to solve a nonlinear optimization problem under linear constraints. Our algorithm to solve this problem consists of two phases. In the first phase, a point is calculated that is globally optimal, but that violates a selected subset of the complete set of constraints. In the second phase, Rosen's projection algorithm is modified into an algorithm that iteratively finds directions that make the violated constraints feasible. These directions are also always feasible with regards to the constraints that are not violated in Phase 1. Once all constraints are feasible, we use the original version of Rosen's algorithm to find a locally optimal point.

We investigate several variants of our algorithm and analyze which variant gives the best results. We also compare three line search algorithms that we can use in our algorithm: backtracking line search, golden section search and a line search by a second order approximation. We analyze which line search algorithm works best for some datasets. The results show that backtracking line search and golden section search have a similar performance, but backtracking line search is slightly better.

The performance of our algorithm is compared to the performance of the original version of Rosen's projection algorithm. We initialize Rosen's projection algorithm by first taking several first order approximations of the objective function and then solving several linear programs.

When comparing the algorithms for some datasets, we find that our modified projection algorithm gives results in a shorter amount time, if in Phase 1 a point with few violated constraints and with a small feasibility gap is found. If the initialization of the original version of Rosen's algorithm finds a good initial point for a dataset, then we see that Rosen's algorithm gives faster results.

Keywords: Rosen's Algorithm, Singular Value Decomposition, Line Search, Projection matrix.

CONTENTS

1. INTRODUCTION

This thesis presents an algorithm that optimizes a non-convex nonlinear objective function under linear constraints. The thesis is based on the following business case: A large health-care insurance company in the Netherlands is collaborating with the market research company GfK (Gesellschaft für Konsumforschung) to develop its marketing strategies. Within these strategies, the main focus is to optimize media expenditure. The company spends a budget of several million euro during the year to sell new insurance policies. Before spending the entire budget, the health-care insurance company wants to have an optimal allocation of their budget over the various media instruments and weeks.

The function that we optimize for GfK is an econometric market model developed by GfK whose purpose is to predict the effects of various media spend choices on the weekly and annual sales of new insurances. Our objective function is a generalization of the objective function of GfK.

In Section 1.1, we define a general mathematical model. In Chapter 2, we use this mathematical model to review the literature that has been written about our problem. In particular, we look at Rosen's projection algorithm (Rosen, 1960). We also review three line search algorithms in Section 2.4 that we use in Rosen's algorithm. These algorithms are backtracking line search, golden section search and a line search algorithm that is based on a second order approximation of an objective function.

In Chapter 3, we define an optimization problem that is more specific to the problem of the health-care company. We also define our objective function and the constraints of the problem in that chapter. In Chapter 4, we present an algorithm to solve this optimization problem. This algorithm is based on Rosen's algorithm. The numerical results for this algorithm are analyzed in Chapter 5. We compare the results of this algorithm on several datasets in that chapter. Finally in Chapter 6 we present our conclusions and we consider some areas for further improvements.

1.1 Mathematical model

In this section, we present a general mathematical model that is considered in Chapter 2. Later in Chapter 3, we present a special case of this general mathematical model that is more related to the problem of the health-care company.

We first introduce the decision variable $y \in \mathbb{R}^N_+$. Now we consider the following problem with a nonlinear objective function and linear constraints:

$$\max_y \{S(y) \quad \text{s.t. } Ay = p, \quad By \geq q, \quad y \geq 0\}. \tag{1.1}$$

The equation $Ay = p$ forms a set of K constraints, with $\sum_{n=1}^{N} a_{kn}y_n = p_k,\ k = 1, \ldots, K$. The matrix $A = (a_{kn}) \in \mathbb{R}^{K \times N}$ and vector $p \in \mathbb{R}^K$. Similarly, the inequality $By \geq q$ forms a set of L constraints, with $B = (b_{ln}) \in \mathbb{R}^{L \times N}$ and $q \in \mathbb{R}^L$.
The nonlinear function $S : \mathbb{R}_+^N \to \mathbb{R}_+$ is assumed to be twice continuously differentiable. There are many functions that occur in nonlinear optimization problems that satisfy this criterion. One example is the Rosenbrock function $f(x) = \sum_{i=1}^{N-1} \left[100\left(y_{i+1} - y_i^2\right)^2 + (y_i - 1)^2\right]$. This is a function that is often used to as a test function to analyze the performance of optimization algorithms. Another example is $S(y) = d \prod_{n=1}^{N} y_n^{a_n}$, with $d \in \mathbb{R}_+$ and $a \in \mathbb{R}^N$. This is an objective function that occurs in geometric programming (Boyd, Kim, Vandenberghe, & Hassibi, 2007). This function also occurs when applying the exponential function to an econometric log-log model $\log(S(y)) = \log(d) + \sum_{n=1}^{N} a_n \log(y_n)$.

It is also possible to write problem (1.1) with only equality constraints and non-negativity constraints on the variables by using L additional variables $z \in \mathbb{R}^L$. Therefore we write, $\sum_{n=1}^{N} b_{ln}y_n - z_l = p_l,\ z_l \geq 0,\ l = 1, \ldots, L$. Therefore, we can write problem (1.1) in the following way:

$$\max_y \{S(y) \quad \text{s.t. } Ay = p, \quad y \geq 0, z_l \geq 0\}. \tag{1.2}$$

We use this formulation with only equality constraints in Subsection 2.5.1.

2. LITERATURE REVIEW

In this chapter, we review literature related to our problem. We first review KKT conditions, pseudoinverses and the singular value decomposition. Then we review several optimization algorithms for general (non-convex) nonlinear programming (NLP) problems. We consider Rosen's projection algorithm, sequential quadratic programming, and barrier functions. We also review the three line search algorithms: backtracking line search, golden section search and a line search algorithm that is based on a second order approximation of the objective function $S(y)$.

2.1 KKT conditions

First we review the Karush-Kuhn-Tucker (KKT) conditions by Karush (1939), and Kuhn and Tucker (1951). The KKT conditions state that if y^* is a local optimum for problem (1.1) and if a regularity condition is satisfied, then the point $(y^{*\top}, \lambda^{*\top}, \mu^{*\top})$, where $\lambda \in \mathbb{R}^K$ and $\mu \in \mathbb{R}^L$ are the KKT multipliers, satisfies the following first-order equations:

$$\begin{aligned}
\nabla S(y^*) &= \sum_{k=1}^{K} \lambda_k a_k - \sum_{l=1}^{L} \mu_l b_l, \\
\sum_{n=1}^{N} a_{kn} y_n &= p_k, && k = 1, \ldots, K, \\
\sum_{n=1}^{N} b_{ln} y_n &\geq q_l, && l = 1, \ldots, L, \\
\mu_l &\geq 0, && l = 1, \ldots, L, \\
\mu_l \sum_{n=1}^{N} (b_{ln} y_n - q_l) &= 0, && l = 1, \ldots, L,
\end{aligned} \tag{2.1}$$

where $a_k = (a_{k1}, \ldots, a_{kN})^\top$, $b_l = (b_{l1}, \ldots, b_{lN})^\top$.

There are several types of regularity conditions. For our problem only one of these conditions is relevant.

For nonlinear problems with nonlinear equality constraints $f(y) \in \mathbb{R}^K$ and nonlinear inequality constraints $g(y) \in \mathbb{R}^L$ constraints, the first KKT condition changes. The condition would be: $\nabla S(y^*) = \sum_{k=1}^{K} \lambda_k \nabla f(y^*) - \sum_{l=1}^{L} \mu_l \nabla g(y^*)$. Notice that this condition includes the linear approximations at the point y^* to the constraints $f(y)$ and $g(y)$. From these linear approximations follows the sufficient condition for the KKT conditions that the linearized functions $f(y^*) + \nabla f(y^*)$ and $g(y^*) + \nabla g(y^*)$ are good approximations to $f(y)$ and $g(y)$ in the neighborhood of y^*.

We can see that problem (1.1) is linearly constrained, so the linearized approximation to these constraints are exactly equal to the constraints at any point y. This means that the regularity condition, that is needed for the validity of the KKT, is satisfied. This regularity condition is also known under the name "linearity constraint qualification".

Next to the above first-order necessary conditions, there is also a second-order necessary condition:

$$\begin{aligned} x^\top \nabla^2 S(y^*) x \leq 0, \\ x \neq 0 \end{aligned} \tag{2.2}$$

where $x \in \{x \,|a_k x = 0,\ \forall k,\ b_l x = 0,\ \forall l \in \mathcal{A}(y^*)\}$ and $\mathcal{A}(y^*) = \{l \,|b_l y^* - q_l = 0\}$, the set of active constraints. In other words, all feasible directions x should be concave.
If all points y for which (2.1) and (2.2) hold are enumerated, then the point with the highest function value is the global maximum.

The second order sufficiency condition tells us that if for a point y^* the equations in system (2.1) hold and if equation (2.2) holds with a strict inequality, then this point is a local maximum.

2.2 Pseudoinverse and singular value decomposition

In this section, we review some properties of the Moore-Penrose pseudoinverse that are of importance when formulating our algorithm. We also briefly review singular value decomposition in this section, as singular value decomposition has some similarities with the pseudoinverse and because singular value decomposition can be used in the calculation of the pseudoinverse.

A pseudoinverse is, as the name 'pseudoinverse' implies, a matrix that mimics properties of the inverse, without being an actual inverse. One of the nice properties of a pseudoinverse compared to a regular inverse, is that it is also defined for matrices that do not have full row rank or that are not square matrices. The Moore-Penrose pseudoinverse was initially independently described in several papers. It was first described by Moore (1920) and later by Bjerhammar (1951) and Penrose (1955).

The Moore-Penrose pseudoinverse of any matrix A is the matrix A^+ that satisfies the following equations:

$$AA^+A = A \tag{2.3a}$$
$$A^+AA^+ = A^+ \tag{2.3b}$$
$$(A^+A)^\top = A^+A \tag{2.3c}$$
$$(AA^+)^\top = AA^+ \tag{2.3d}$$

Especially equation (2.3a) and equation (2.3b) are very similar to some well known properties that any square and invertible matrices has. In particular we have for any square and invertible matrix M, the equation $MM^{-1} = I$, where I is the identity matrix of the appropriate size. This equation is equivalent to $MM^{-1}M = M$. An important note that needs to be made here is that equations (2.3a) to (2.3d) do not automatically imply that $AA^+ = I$. The following lemma is a known result of the uniqueness of A^+.

Lemma 1. *Let $A^+ \in \mathbb{R}^{N\times K}$ be a matrix, then the matrix A^+ that satisfies equations* (2.3a) *to* (2.3d) *is the unique Moore-Penrose pseudoinverse.*

Proof. This proof is by contradiction.
Let $A^+ \in \mathbb{R}^{N\times K}$ and $C \in \mathbb{R}^{N\times K}$ be matrices that satisfy equations (2.3a) to (2.3d) and $A^+ \neq C$. Then we have the following:

$$AA^+ = ACAA^+ = (AC)^\top (AA^+)^\top = (C^\top A^\top)(A^{+\top} A^\top) = C^\top (AA^+ A)^\top = C^\top A^\top = (AC)^\top = AC \tag{2.4}$$

A similar relation holds for $A^+A = CA$. We now use that $AA^+ = AC$ and $A^+A = CA$ to obtain the following equation:

$$A^+ = A^+AA^+ = A^+AC = CAC = C \tag{2.5}$$

This is a contradiction with the assumption that $A^+ \neq C$, so there can be only one matrix that satisfies (2.3a) to (2.3d), so A^+ is unique. □

It has also been proven that A^+ always exists (Penrose, 1955).

The Moore-Penrose pseudoinverse has many other useful properties. We mention a few of them.

1. If A is invertible, then $A^{-1} = A^+$.
2. If A^+ satisfies all properties from (2.3), then $A^\top \left(AA^\top\right)^+$ does so too. So $A^+ = A^\top \left(AA^\top\right)^+$, because A^+ is unique. Similarly, $(A^+)^\top = \left(AA^\top\right)^+ A$.
3. If A has orthonormal rows, so if $a_i^\top a_i = 1$ and $a_i^\top a_j = 0$, for $\forall i, j, i \neq j$, then $A^+ = A^\top$.
4. If $A \in \mathbb{R}^{m\times(m+n)}$ and $A = [D, E]$, where D is an $m \times m$ diagonal matrix and E is an $m \times n$ empty matrix, then $A^+ = \left[D^+, E^\top\right]^\top$ is a matrix of size $(m+n) \times m$, where D^+ only differs from D in the non-zero elements of D, which are inverted.

Properties 1 to 4 show their usefulness when discussing singular value decomposition and projection matrices. We also need these properties in Section 4.2 when we present some improvements to our algorithm.

We continue this section by briefly reviewing Singular Value Decomposition (SVD). The SVD of a matrix A can be used to compute the Moore-Penrose pseudoinverse of that matrix. The SVD can also be used to simplify some equations, which can give computational advantages.
Every matrix A of dimension $m \times n$ can be decomposed into $U\Sigma V^\top$, where U is an orthogonal matrix of dimension $m \times m$ and V is an orthogonal matrix of dimension $n \times n$. The matrix Σ has dimension $m \times n$ and its diagonal elements are the singular values of A, which are the square roots of the eigenvalues of $AA^\top$. The matrix Σ has as many nonzero values as the number of independent rows, which is equivalent to the rank of A.

Calculating the singular value decomposition of a matrix is a very common operation in linear algebra. Since singular value decomposition is so common, there are in most programming languages packages available that provide an implementation of SVD in a very low programming language. An example of such a package is the Linear Algebra PACKage (LAPACK) (Anderson et al., 1999). This package includes, amongst other functions, an implementation of singular value

decomposition in the programming language Fortran 90. This package is also implemented in other programming languages, for example in R, where it can be used with the function 'la.svd'.
Packages like LAPACK are typically very computationally efficient and there is therefore usually no reason to program an implementation of the singular value decomposition. Therefore, we do not go further into the details on how to calculate the SVD of a matrix.

Now to calculate A^+, we can make use of the singular value decomposition and Property 3 of the Moore-Penrose inverse. We have that $A^+ = (U\Sigma V^\top)^+ = (V^\top)^+\Sigma^+U^+ = V\Sigma^+U^\top$. As we mentioned before in Property 4, the pseudoinverse Σ^+ of matrices that consists of a diagonal matrix and a zero matrix are straightforward to calculate by transposing Σ by taking the inverse of the non-zero elements.
Furthermore SVD can also be used to simplify some multiplications which can have some numerical advantages. For example $A^+A = V\Sigma^+U^\top U\Sigma V^\top = V\Sigma^+\Sigma V^\top = VV^\top$.

2.3 Constrained gradient ascent

We have seen in Section 2.1 that a necessary condition for a point y^* needs to satisfy the KKT conditions (2.1) to be the global maximum of the function $S(y)$. The problem remains how to find such a point. In this section, we first briefly review how to find a KKT point for unconstrained nonlinear functions. Then we review some algorithms that can deal with constrained problems, like problem (1.1).
For unconstrained problems, there are several straightforward methods on how to find a global maximum to a function $S(y)$. One of these methods is unconstrained gradient ascent. Gradient ascent is a whole family of methods. We now briefly review a basic algorithm of unconstrained gradient ascent as an introduction to constrained gradient ascent algorithm that we review the remainder of this section.
The unconstrained gradient ascent algorithm consists of two steps. In this algorithm and all other algorithms that we review or present during this thesis, we use '$\leftarrow$' when assigning a value to a variable. We use '=', or any inequality sign when making a logical test.
We initialize the gradient ascent algorithm by taking any feasible point y^0 and $r \leftarrow 0$. Then $\nabla S(y^0)$ is the direction in which $S(y^0)$ increases the fastest. In the remainder of this section, we denote a direction that improves the objective function by $v \in \mathbb{R}^N$. In this case, $v = \nabla S(y^0)$. Now we go to Step 1 of the algorithm.

1. In Step 1 we check if $v = 0$. If $v = 0$, the algorithm stops. Otherwise, we go to Step 2.
2. In Step 2 we set $y^{r+1} \leftarrow y^r + \alpha v$, for some $\alpha > 0$ for which $S(y^1) > S(y^0)$. There are several methods to set the value for α. We analyze these methods in Section 2.4. Then we set $r \leftarrow r + 1$ and go to Step 1.

However, unconstrained gradient ascent does in general not result in a feasible solution for constrained problems, because for some feasible y and direction v, the point $y + \alpha v$ might not be feasible for any $\alpha > 0$. Therefore, a constrained gradient ascent algorithm is needed to solve

problems that are similar to problem (1.1).

There are several algorithms, for example, Zoutendijk's algorithm (Zoutendijk, 1960) and Rosen's projection algorithm (Rosen, 1960), that are specifically formulated for problems with linear constraints and any nonlinear objective function. Not all published algorithms for problems with linear constraints and any nonlinear objective function are proven to always converge to a point that satisfies the KKT conditions. For Zoutendijk's algorithm, there are examples for which there is no convergence to a KKT point (Bazaraa, Sherali, & Shetty, 2006). Even though the original proof of convergence of the algorithm by Rosen (1960) was not correct. it was later proven by He and Zhang (1986) that Rosen's algorithm reaches a KKT point. Rosen also formulated a variant of his own algorithm that involves nonlinear constraints (Rosen, 1961).
In the remainder of this section, we review Rosen's algorithm for linear constraints and in Section 4.2 we implement a modified version of Rosen's algorithm. In Section 4.3 we implement this algorithm for a number of random feasible starting points.

Rosen's algorithm is a gradient projection algorithm. This means that the best ascent direction $\nabla S(y)$ is projected by a projection matrix P on the set of feasible directions. This gives the steepest ascent direction that is also feasible.
Recall that we defined the set of active inequality constraints at a point y as $\mathcal{A}(y)$ in Section 2.1. These active inequality constraints combined with the equality constraints form the matrix $M = \left(B_{\mathcal{A}(y)}^{\top}, A^{\top}\right)^{\top}$, which contains all active constraints at the point y. Since we only consider the active inequality constraints $B_{\mathcal{A}(y)}$, we also have to change the notation for μ. We use that $\mu_{\mathcal{A}(y)}$ denotes the KKT multipliers of the active inequality constraints. Note that equation (2.1) implies that $\mu_l = 0 \ \forall l \notin \mathcal{A}(y)$.

The matrix $M^{\top}(MM^{\top})^{-1}M$ forms a projection matrix of size $N \times N$. This projection matrix $M^{\top}(MM^{\top})^{-1}M$ has the property that for every vector $x \in \mathbb{R}^N$, $M^{\top}(MM^{\top})^{-1}Mx = \nu$, where $\nu \in \mathbb{R}^N$ is such that it is a combination of the rows of M. In other words, $\nu = M^{\top}u$, for some u that has the same number of rows as M, so $u \in \mathbb{R}^{|\mathcal{A}(y)|+K}$. To see this, we first note that the vector x can be separated into two vectors ν and $z \in \mathbb{R}^N$, for which it holds that $x = \nu + z$. Here z is a vector that is in the null space of M, so $Mz = 0$. Now equation (2.6) shows a short proof of $M^{\top}(MM^{\top})^{-1}Mx = \nu$:

$$\begin{aligned} M^{\top}(MM^{\top})^{-1}Mx = M^{\top}(MM^{\top})^{-1}M(z+\nu) = M^{\top}(MM^{\top})^{-1}M\nu = \\ M^{\top}(MM^{\top})^{-1}MM^{\top}u = Mu = \nu \end{aligned} \tag{2.6}$$

Equation (2.6) also shows that the projection matrix $P = I - M^{\top}(MM^{\top})^{-1}M$, where I is an identity matrix of size $N \times N$, maps any vector x to a vector in the null space of M, so $MPx = Mz = 0$. In particular the vector $\nabla S(y)$ can be mapped to a feasible direction. We define this map as the direction $v = P\nabla S(y)$. We also note that this projection matrix P has the following properties: $P = PP$ and $P = P^{\top}$.

Note that $P = I - M^{\top}(MM^{\top})^{-1}M$ implies that $MM^{\top}$ needs to be nonsingular so that it can

be inverted. This implies that M needs to have full row rank and that means that there cannot be any redundant constraints in M. This means that there cannot be any constraints that are active at y, while they do not reduce the size of the polyhedron of the active constraints. We discuss in Subsection 2.3.1 when exactly this happens and how this can be dealt with.

Recall from the beginning of Section 2.3 that for the unconstrained gradient ascent algorithm v is a direction for which the objective function $S(y)$ increases in value. The linearly constrained problem adds the requirement that the direction v should also be feasible. Before we can show that these both hold, we need equation (2.7):

$$Mv = MP\nabla S(y) = M\nabla S(y) - MM^\top (MM^\top)^{-1} M\nabla S(y) = 0. \tag{2.7}$$

By the definition of M, $Mv = 0$ implies that $Av = 0$ and $B_{\mathcal{A}(y)}v = 0$, so v is a feasible direction for all active constraints. We also know that any direction is feasible for the nonactive constraints. This means that v is a feasible direction. For any direction v to be nondecreasing, $\nabla S(y)^\top v$ should be non-negative. We have that

$$\nabla S(y)^\top v = \nabla S(y)^\top P\nabla S(y) = \nabla S(y)^\top P^\top P\nabla S(y) = ||P\nabla S(y)|| \geq 0, \tag{2.8}$$

where $P = P^\top P$ because of the properties of a projection matrix.

Furthermore, we have that if $v = 0$ then v satisfies the first KKT condition of (2.1), because:

$$0 = v = P\nabla S(y) = (I - M^\top (MM^\top)^{-1} M)\nabla S(y) = \nabla S(y) - M^\top w = \nabla S(y) - B_{\mathcal{A}(y)}^\top \mu_{\mathcal{A}(y)} - A^\top \lambda, \tag{2.9}$$

where $w^\top = \left((MM^\top)^{-1} M\nabla S(y)\right)^\top = (\mu_{\mathcal{A}(y)}^\top, \lambda^\top)$.
Therefore, when defining v, we are also defining the values for the KKT multipliers $\mu_{\mathcal{A}(y)}$ and λ. Recall that the remaining KKT multipliers of μ that belong to the inactive constraints are set to zero. Since v is a feasible direction, Rosen's algorithm ensures that the point $y + \alpha v$ is feasible for some α if y is a feasible point. So we know that in addition to the first KKT condition, also all other KKT conditions are satisfied, except for $\mu \geq 0$, which might not be satisfied for some y. However, when this condition is satisfied, then this means that y is a KKT point.

Now we formulate Rosen's algorithm. The initialization of this algorithm is similar as in the unconstrained gradient ascent algorithm. So we initialize Rosen's algorithm by finding a feasible point y^0 and we set $r \leftarrow 0$.
Then we get two main steps of Rosen's algorithm.

1. In the first step we consider three possible options for the direction v. The three options are considered in order, so if the condition for Option (a) is not satisfied, then Option (b) is considered. If the condition for Option (b) is not satisfied then Option (c) is used.
   First we set $M \leftarrow \left(B_{\mathcal{A}(y^r)}^\top, A^\top\right)^\top$ and calculate the projection matrix P.
   (a) If M is empty, then the problem is not constrained at the point y^r and we therefore set $v \leftarrow \nabla S(y^r)$. We stop the algorithm if $v = 0$, because this means we have found a local optimum. Otherwise we go to Step 2.
   (b) For Option (b) we know that M is not empty, so the problem is constrained at y^r and we therefore need to project the gradient $\nabla S(y^r)$ with the projection matrix P. The

condition for Option (b) is that $||P\nabla S(y^r)|| > s_r|u|$, where s_r is some strictly positive constant for all r and u is the most negative KKT multiplier of the inequality constraints, so $u \leftarrow \min_{l \in \mathcal{A}(y)} \mu_l$. Recall that μ_l, $\forall l$ is calculated when $P\nabla S(y^r)$ is calculated. If there are no negative multipliers μ_l for any l, then $u \leftarrow 0$.
If the condition that $||P\nabla S(y^r)|| > s_r|u|$ is satisfied, then we set $v \leftarrow P\nabla S(y^r)$. If both the KKT multiplier $\mu \geq 0$ and $v = 0$, then we have satisfied the KKT conditions (2.1) and we have found a local optimum, so we stop the algorithm. Otherwise, we go to Step 2.

(c) If the conditions in Option (a) and (b) do not hold, then we first find $\hat{l} \leftarrow \text{argmin}_{l \in \mathcal{A}(y)} \mu_l$ and remove the corresponding row from the matrix of inequality constraints, so $B_{\mathcal{A}(y^r)\setminus\hat{l}}$. Then we set $M \leftarrow \left(B^\top_{\mathcal{A}(y^r)\setminus\hat{l}}, A^\top\right)^\top$ and we calculate a new projection matrix P. We use these new matrices to set $v \leftarrow P\nabla S(y^r)$ and we go to Step 2.

2. In the second step we maximize $S(y^r + \alpha v)$. For some α, such that $y^{r+1} \leftarrow y^r + \alpha v$ is a feasible point. Then we set $r \leftarrow r + 1$ and go to Step 1.

We further elaborate on Step 2 in Section 2.4.
It can be shown that even though a constraint is removed in Option (c), the direction v is still an ascent direction (Bazaraa et al., 2006). An intuitive reason for this can be given by the meaning of the KKT multipliers. If $\mu > 0$, then only directions that point to the outside of the feasible region will increase the objective value. When instead μ_l is negative for some l, then this shows that a move away from the corresponding constraint to the interior of the feasible region results in a better objective value. Therefore, if the restriction that said constraint has to be tight is dropped, then we are able to find a direction v that leads to an improvement.
There is no clear 'correct' value for the parameter s_r. It has been shown (He & Zhang, 1986) that Rosen's algorithm converges to a KKT point in finitely many steps for any nonlinear objective function and for $0 < s_r < \infty$. There are many examples, for example by Du, Wu, and Zhang (1990), that there is no convergence from some objective functions when $s_r = 0$ $\forall r$. However, Ritter (1980) showed that if $s_r = 0$ for some specific r, then the algorithm still converges.
In Subsection 4.3.4, we further analyze to what values we set the parameter s_r.

2.3.1 Redundant constraints

We mentioned before that $(MM^\top)^{-1}$ might not exist due to constraints being redundant. To fix this we have to take the following steps. It might be that there are inequality constraints for which we have $b_l y \geq q_l$ and $b_{\hat{l}} y \geq q_{\hat{l}}$, with $b_{\hat{l}} = -b_l$ and $q_{\hat{l}} = -q_l$ for some l and $\hat{l}$. This implies that both constraint l and $\hat{l}$ are always active and therefore they are always both in M. These two constraints have a rank of 1, so M does not have full row rank if both of these constraints are in M. To fix this, the above constraints should be replaced by a single equality constraint. The remaining set of inequalities are defined as $\hat{B}$ and $\hat{q}$

Now in the remaining constraints, we have to identify which of these constraints are redundant. There are several methods to remove redundant constraints (Paulraj & Sumathi, 2010). In this section, we review the method by Caron, McDonald, and Ponic (1989). We have to solve the

following problem:

$$\max_{y} \left\{ \hat{b}_l y \quad \text{s.t. } Ay = p, \quad \hat{B}_l y \geq \hat{q}, \quad \hat{b}_l y \geq \hat{q}_l - 1 \right\}. \tag{2.10}$$

Where $\hat{B}_l$ and $\hat{q}_l$ are $\hat{B}$ and $\hat{q}$, but with row l removed, which corresponds to the constraint $\hat{b}_l y \geq \hat{q}_l$.

Now we find that if $\hat{b}_l y^* > q_l$, where y^* is the solution to problem (2.10), then this implies that $b_l y \geq q_l$ is not a redundant constraint. We can see this because if a non-redundant constraint is relaxed, then the feasible region becomes bigger and so $b_l y^*$ will be bigger. If it is a redundant constraint it gets removed from the set of constraints and we repeat (2.10) for the next constraint until all constraints have been checked.

Note that the -1 is needed in problem (2.10). If it is not included, then there can still exist sets of equations where some constraints are redundant. For example the following set of equations in $\mathbb{R}^2$:

$$y_1 \geq 0, y_2 \geq 0, y_1 + y_2 \geq 0 \tag{2.11}$$

At the point $y_1 = y_2 = 0$ all three equations are tight, but the constraint $y_1 + y_2 \geq 0$ is still redundant. We can see this, because y_1 and y_2 still sum to zero, even when the constraint $y_1 + y_2 \geq 0$ is relaxed.

A similar approach can be taken for the equality constraints. If changing a constraint leads to an infeasible problem, then this constraint is removed from the set of constraints.

However, we still might not have full row rank at this point. For example the following inequalities in $\mathbb{R}^3$ result in a matrix M for which $MM^\top$ in $\mathbb{R}^{3\times 3}$ is not invertible, but that does have a feasible direction.

$$y_1 \geq 0, y_2 \geq 0, y_1 + y_2 \leq 0 \tag{2.12}$$

These constraints are always all tight, but the inequality matrix A is not invertible. Furthermore, the direction $v = (0, 0, 1)^\top$ is obviously a feasible direction.

To also allow directions such as v, we modify the projection matrix that we reviewed in Section 2.3 for the general version of Rosen's algorithm. We also use this new projection matrix in Section 4.2 in our modification of Rosen's algorithm. To change this projection we use the Moore-Penrose pseudoinverse that we reviewed in Section 2.2. This means that we replace $P = I - M^\top (MM^\top)^{-1} M$ by $\hat{P} = I - M^+ M$. We also denote the direction and the vector of multipliers that have been created with the Moore-Penrose pseudoinverse with $\hat{v} \in \mathbb{R}^N$ and $\hat{w} \in \mathbb{R}^{K+L}$ respectively. Recall that M^+ is the Moore-Penrose pseudoinverse and that $M^+ M$ is a square matrix. We also saw in equation (2.3b) in Section 2.2 that $M^+ = M^\top (MM^\top)^+$. We can therefore see that the following relation holds if $(MM^\top)^{-1}$ exists:

$$\hat{P} = I - M^+ M = M^\top (MM^\top)^+ M = M^\top (MM^\top)^{-1} M = P \tag{2.13}$$

If $(MM^\top)^{-1}$ does not exist, then the properties that we have seen in equations (2.7) and (2.9) also still hold. We have by property (2.3a) of the pseudoinverse, the following:

$$M\hat{v} = MP\nabla S(y) = M\nabla S(y) - MM^+ M\nabla S(y) = M\nabla S(y) - M\nabla S(y) = 0. \tag{2.14}$$

So $\hat{v}$ is a feasible direction, which means that equation (2.7) is satisfied.

For equation (2.9) we now get the following:

$$0 = \hat{v} = \hat{P}\nabla S(y) = (I - M^{+}M)\nabla S(y) = (I - M^{\top}(MM^{\top})^{+}M)\nabla S(y) = \nabla S(y) - M^{\top}\hat{w} = \nabla S(y) - B_{\mathcal{A}(y)}^{\top}\mu - A^{\top}\lambda \tag{2.15}$$

Where $\hat{w} = (MM^{\top})^{+}M\nabla S(y)$ is the vector of KKT multipliers and $\hat{w} = w$ if $(MM^{\top})^{-1}$ exists.

2.4 Line search

In Section 2.3, we mentioned in Step 2 of Rosen's projection algorithm that we need to maximize $S(y + \alpha v)$ for some α, such that $y + \alpha v$ is a feasible point. In this section, we assume that v is a feasible direction and y is a feasible point. This means that $y + \alpha v$ is also a feasible point for some $\alpha > 0$. Our constraints $Ay = p$ and $By \geq q$ are linear and so $A(y + \alpha v) = Ay + \alpha Av = Ay = p$. We can calculate for any constraint $l \in \{1, \ldots, L\}$ the smallest α for which we have $b_l(y + \alpha v) \leq q_l$. So $\alpha = \frac{q_l - b_l y}{b_l v}$. The smallest positive α for which any of the constraints are violated is denoted by $\alpha_{\max}$.
Now we know that we have to maximize $S(y + \alpha v)$, with $\alpha \in (0, \alpha_{\max}]$.

The method that is used to do this is called line search. We review several line search algorithms in this section. Later in Chapter 5, we implement all algorithms that we mention in this section and we analyze the computational advantages and disadvantages between the methods. First we describe the golden-section search, which was first discovered by Kiefer (1953). Later in this section, we also describe backtracking line search. Finally, a second order approximation of $S(y + \alpha v)$ is described, which can be used to approximate a local maximum $S(y + \alpha v)$ with regards to α in a single step.

2.4.1 Golden-section search

First, we review the basic implementation of golden-section search for functions with at most one local maximum in the feasible interval. Therefore, in the first part of this subsection we assume that $S(y + \alpha v)$ only has one local maximum for all $\alpha \in (0, \alpha_{\max}]$.
In the second part of this subsection, we consider golden-section search when there is more than one local maximum. If we would apply golden-section search as if the function has a single local maximum, then golden-section search might not increase the objective value. So we show how to modify golden-section search such that an improvement is always found.

The idea in golden-section search is that from any feasible interval of a one-dimensional function, two subsections are taken. We define the feasible interval as $[\alpha_a, \alpha_b]$, where $\alpha_a, \alpha_b \in \mathbb{R}_+$ and $\alpha_a < \alpha_b$. The subintervals are $[\alpha_a, \alpha_d]$ and $[\alpha_c, \alpha_b]$, where $\alpha_c, \alpha_d \in [\alpha_a, \alpha_b]$ and $\alpha_c < \alpha_d$. Note that the subintervals are overlapping each other. From these two intervals, the interval that contains the global maximum to the function $S(y + \alpha v)$ is selected. Then this process is repeated until the length of the interval is below a certain tolerance value. We then find the point α that maximizes $S(y + \alpha v)$.

The word 'golden' in golden-section search comes from the use of the golden ratio, which we denote by ϕ. It is implemented such that at every iteration both subsections are exactly the same size and that the ratio between a subsection and the whole interval is equal to $\frac{\phi}{\phi+1}$ during every iteration. This implies that either between two iterations only one new point needs to be evaluated, which reduces the computational cost.

We now describe the golden-section search algorithm in more detail. The golden-section search algorithm is initialized by setting $\alpha_a \leftarrow 0$ and $\alpha_b \leftarrow \alpha_{max}$. This means that the golden-section search algorithm starts with the interval $[\alpha_a, \alpha_b] = [0, \alpha_{\max}]$. Also the two subintervals are calculated. We have $\alpha_c \leftarrow \alpha_b - \frac{\alpha_b - \alpha_a}{\phi}$ and $\alpha_d \leftarrow \alpha_a + \frac{\alpha_b - \alpha_a}{\phi}$.
The golden-section search algorithm now consists of the following two steps:

1. In Step 1, two cases are determined. We denote these cases with Case (a) and Case (b).
   (a) If $S(y + \alpha_c v) \geq S(y + \alpha_d v)$, then we set $\alpha_b \leftarrow \alpha_d$, $\alpha_d \leftarrow \alpha_c$ and $\alpha_c \leftarrow \alpha_b - \frac{\alpha_b - \alpha_a}{\phi}$ and go to Step 2.
   (b) If the condition in Case (a) is not met, then we set $\alpha_a \leftarrow \alpha_c$, $\alpha_c \leftarrow \alpha_d$ and $\alpha_d \leftarrow \alpha_a + \frac{\alpha_b - \alpha_a}{\phi}$ and go to Step 2.
2. Stop the algorithm if the difference between α_a and α_b is smaller than the tolerance value, after which $\frac{\alpha_c + \alpha_d}{2}$ is outputted. Otherwise, go back to Step 1.

In the above algorithm if in Step 1 Case (a) is selected, so if $S(y + \alpha_c v) \geq S(y + \alpha_d v)$, then this implies that the global maximum has to be in the interval $[\alpha_a, \alpha_d]$. We know this, because $S(y + \alpha v)$ only has one maximum and if the global maximum would be in the other interval, then this would imply that the function has at least two local maximums.
If instead for Case (b) is selected, then $S(y + \alpha_c v) < S(y + \alpha_d v)$ and for the same reasons, the global maximum is in the interval $[\alpha_c, \alpha_b]$.
Notice that for Case (a) that the evaluated value of $S(y + \alpha_c v)$ that was already calculated can be reused as the value of $S(y + \alpha_d v)$ in the next iteration. For Case (b), this is the other way around.

A disadvantage of this golden-section search algorithm is that the algorithm only converges to a global maximum for functions with at most one local maximum. For functions with more than one local maximum it converges to a local maximum. This local maximum might have a lower value than $S(y)$. This is problematic since we know that we can reach a point with a higher value than $S(y)$, because v is an ascending direction.
However, the golden-section search algorithm can be easily changed to ensure that it converges to a local maximum with a higher objective value than $S(y)$. For this we need we use the property that v is an ascending direction, so we know that the first local maximum (the local maximum with the lowest positive value of α) has a higher objective value than $S(y)$. We can therefore make the restriction that the interval $[\alpha_c, \alpha_b]$, which leaves out α_a, is only selected when $S(y + \alpha_a v) < S(y + \alpha_d v)$, so a lower objective value is never obtained. Therefore, we replace Step 1, Case (a) with the following:

(a) If $S(y+\alpha_c v) \geq S(y+\alpha_d v)$ or $S(y+\alpha_a v) \geq S(y+\alpha_d v)$, then we set $\alpha_b \leftarrow \alpha_d$, $\alpha_d \leftarrow \alpha_c$ and $\alpha_c \leftarrow \alpha_b - \frac{\alpha_b - \alpha_a}{\phi}$ and go to Step 2.

For the golden-section search algorithm many evaluations of the objective function are needed. As we mentioned, the golden ratio ensures that the length of an interval after an additional step is equal to $\frac{\phi}{\phi+1} \approx 0.61$ times the length of the previous interval. Therefore, if the tolerance level is equal to 10^{-5} and the length of the initial interval is equal to 10^3, then it takes 38 steps to reach the tolerance level and therefore also 38 (+ 1 for the first iteration) evaluations of the objective function. When the objective function is expensive to evaluate, then this may not be a great approach, as it is often needed to do several line searches to find a local maximum to the objective function $S(y)$.

2.4.2 Backtracking line search

In this section, we review backtracking line search. The idea of backtracking line search is to introduce a minimum expectation of the improvement of the objective function $S(y+\alpha v)$ for all α. If the actually realized improvement for an α is bigger than this minimum improvement then this α is used and the line search is completed. If this is not the case, then α is decreased by some factor $\rho \in (0,1)$. This implies that $S(y+\alpha v)$ is not minimized, but instead a 'good enough' α is found. Finding a good enough α is often sufficient, because in the next iteration a new line search that moves again away from this point is calculated again. Therefore this method saves some computation time in exchange for finding a worse solution.

The backtracking line search algorithm is initialized by setting $\alpha = \alpha_{\max}$ and then consists of the following two steps:

1. If $S(y+\alpha v) \geq S(y) + C\alpha \nabla S(y)^\top v$, where $C \in (0,1)$ is some parameter, then stop and output α. Otherwise, go to Step 2.

2. Set $\alpha \leftarrow \rho\alpha$ and go to Step 1.

The condition $S(y+\alpha v) \geq S(y) + C\alpha \nabla S(y)^\top v$ is the Armijo condition (Armijo, 1966). Clearly $\nabla S(y)^\top v$ is the expected increase of the objective function at the point y and so $S(y) + \alpha \nabla S(y)^\top v$ is the first-order approximation at y for all α. This makes it clear that $S(y) + C\alpha \nabla S(y)^\top v$, for some $C \in (0,1)$ is indeed a condition for a minimum improvement over $S(y)$ that $S(y+\alpha v)$ should make for all α.

In the book by Nocedal and Wright (1999) the value $C = 10^{-4}$ is suggested. There is no specific suggestion made for ρ. When setting a value for ρ it should be taken into account that ρ is not too high, because that might mean that it takes a long time before an α is found that satisfies the Armijo condition. If ρ is set to a too low value, then many good values for α might be skipped. There are also several variants and extensions of the Armijo condition. A few examples of these are the Goldstein conditions and the Wolfe conditions (Wolfe, 1971), which adds the condition that the gradient at the new point $\nabla S(y+\alpha v)$ should be significantly smaller than the gradient of the old point, $\nabla S(y+\alpha v)$. We do not go further into detail of the Goldstein conditions and the Wolfe

conditions, but these conditions might be useful in possible improvements of the backtracking line search algorithm.

2.4.3 Quadratic Taylor approximation

The previous two line search methods are both linear methods. The method that we review in this section uses a quadratic function. This means that it is needed to calculate the Hessian of $S(y)$, which consists of N^2 values. This is, therefore, more expensive than the gradient, for which only N values need to be calculated. Therefore, this method might be too computationally expensive. As in the previous methods we have an ascending direction v and a step size α that we want to optimize, so $S(y+\alpha v)$. We now take the second order Taylor approximation of $T(\alpha)$ this function. We get the following equation:

$$S(y+\alpha v) \approx S(y) + \nabla S(y)^\top v\alpha + \frac{1}{2}v^\top \nabla^2 S(y) v\alpha^2 = T(\alpha) \tag{2.16}$$

We know that v is an ascending direction, so for some small α, the function $T(\alpha)$ will always increase. If the value of $\frac{1}{2}v^\top \nabla^2 S(y)v$ is negative, then $T(\alpha)$ is concave and so the maximum of $T(\alpha)$ is found by setting $\frac{\partial T(\alpha)}{\partial \alpha} = 0$. We get the following: $\frac{\partial T(\alpha)}{\partial \alpha} = \nabla S(y)^\top v + v^\top \nabla^2 S(y) v\alpha = 0$. This implies that $\alpha = -\frac{\nabla S(y)^\top v}{v^\top \nabla^2 S(y) v} > 0$.
If $\frac{1}{2}v^\top \nabla^2 S(y)v$ positive, then is $T(\alpha)$ is convex. We can then take $\alpha = \alpha_{\max}$ to find the maximum of the function. In both of the cases the found value of α can be used in calculating a new feasible point $y + \alpha v$ and a new iteration of quadratic approximation algorithm can be started.
If $T(\alpha)$ is concave, then it could be considered to find a better approximation of the α that minimizes $S(y+\alpha v)$ by iteratively approximating α. The α of each iteration is denoted by α^r. We initialize the algorithm by setting $\alpha^0 = 0$ and we set $r = 1$. Now we find the following iterative quadratic approximation algorithm:

1. In Step 1 we take the Taylor approximation $T(\alpha_k)$ of $S(y + \alpha^{r-1}v + \alpha^r v)$ around the point $y + \alpha^{r-1}v$. We get the following equation:
$$T(\alpha^r) = S(y + \alpha^{r-1}v) + \nabla S(y + \alpha^{r-1}v)^\top v\alpha^r + \frac{1}{2}v^\top \nabla^2 S(y + \alpha^{r-1}v) v\alpha^{r2} \tag{2.17}$$
2. In Step 2 we find α^r in the same way as was described before in this section, but taking into account that $\alpha^{r-1} + \alpha^r \in (0, \alpha_{\max})$. We stop the algorithm if $|\alpha^r - \alpha^{r-1}|$ is within some tolerance bound. Otherwise we set $\alpha^r \leftarrow \alpha^{r-1} + \alpha^r$ and $r \leftarrow r + 1$ and go to Step 1.

As we mentioned before in this section, the quadratic approximation algorithm is already quite computationally expensive compared to the algorithms in the previous sections. Therefore taking several iterations of the iterative quadratic approximation algorithm might result in a too high computational burden. In Chapter 5, we see that the computational burden of the regular version of the quadratic approximation algorithm is indeed too high to consider using the iterative version of the quadratic approximation algorithm.

2.5 Nonlinear programming

There are several methods to optimize NLP problems, we review two of these methods in this section. The first one is Sequential Quadratic Programming (SQP). It revolves around sequen-

tially optimizing a quadratic problem that has as objective a second order approximation of the Lagrangian of the objective of the NLP. Also, the constraints are linearized at each step.
The other method is the interior point method. There are many variants of this and they primarily revolve around constrained logarithmic barrier functions.

2.5.1 Sequential Quadratic Programming

Sequential Quadratic Programming (SQP) is a method to solve nonlinear programs by sequentially optimizing a Quadratic Programming problem, which at each sequential iteration is a closer approximation of the optimum of the original nonlinear problem. Therefore, "[SQP] can be viewed as the natural extension of Newton and quasi-Newton methods" (Boggs & Tolle, 1996).

In general in SQP theory any problem is first rewritten to a problem with only equality constraints and slack variables as in problem (1.2). Furthermore, the objective $S(y)$ is approximated by a quadratic function at the optimum of the previous iteration. We name this point point y_k. For the first iteration of the QP problem, any (not necessarily feasible) point can be taken. This gives the following equation:

$$S(y) = S(y_k) + \nabla S(y_k)^\top (y - y_k) + \tfrac{1}{2} (y - y_k)^\top H_k (y - y_k) \tag{2.18}$$

Where H_k is either equal to the Hessian of the Lagrangian of S in problem (1.2), evaluated at y_k and the Lagrange multipliers or $H_k = \nabla^2_{yy} S(y_k)$. These equations are generally not equivalent, but the can both be used. The second formulation only works because of the linear constraints in problem (1.2). If this is not the case, then there are examples where this breaks down (Boggs & Tolle, 1996).

From Equation (2.18) and problem (1.2), we can formulate the QP

$$\begin{aligned} y_{k+1} = \operatorname{argmax}_{y \in \mathbb{R}^n} \quad & \nabla S(y_k)^\top (y - y_k) + \tfrac{1}{2} (y - y_k)^\top H_k (y - y_k), \\ & Ay = p, \\ & y \geq 0, \end{aligned} \tag{2.19}$$

For general NLP, the constraints of problem (2.19) would be $C(y_k) + J(y_k)(y - y_k) = 0$, where $C(y_k)$ is are the constraint and $J(y_k)$ the Jacobian of the constraints both evaluated at y_k. However, we have linear constraints, so $C(y_k) + J(y_k)(y - y_k) = 0$ is equivalent to the constraints of problem (2.19).

This problem is clearly quadratic and for each y_k the objective is an approximation of the true objective. The constraints in (2.19) are the same as the constraints in (1.2), therefore every solution y_k of the subproblem (2.19) is also a the feasible point of the problem (1.2).

For a locally optimal point of problem (1.2), we need to satisfy the KKT necessary condition in (2.19). For a globally optimal point y^*, we also need, in addition to this condition, two sufficiency conditions. The first of these states that there is complementary slackness at y^*. We also need that for all feasible directions d from y^* the objective is concave. A direction d is feasible if the constraints are valid when tight inequality constraints do not change and when equality constraints also hold when moving in this direction. These conditions can be rewritten as the following

equations:

$$\begin{aligned}
&\nabla\mathcal{L}(y^*,\lambda^*,\mu^*)=0,\\
&\mu_i^*>0 \quad \text{if } y_i^*=0 \quad \forall i,\\
&d^\top\nabla^2_{yy}\mathcal{L}(y^*,\lambda^*,\mu^*)*d>0, \quad \forall d\neq 0 \text{ s.t. } Ad=0, \quad d_i=0 \text{ if } y_i^*=0 \quad \forall i
\end{aligned} \tag{2.20}$$

Where $\nabla_y\mathcal{L}(y^*,\lambda^*,\mu^*)=\nabla_y S(y_k)+\lambda^{*\top}(Ay^*-b)+\mu^{*\top}y$. Note that $\nabla^2_{yy}\mathcal{L}(y^*,\lambda^*,\mu^*)=\nabla^2_{yy}S(y_k)$ due to the constraints being linear. The last condition of (2.20) is also very similar to (2.2).
It can be shown that under these conditions, the condition that $\nabla^2_{yy}S(y^*)$ is not singular and conditions on converging properties of H_k, that SQP, if (y_k,λ_k,μ_k) is sufficiently close to (y^*,λ^*,μ^*), converges quadratically to (y^*,λ^*,μ^*).
For problems such as (2.19), there are many fast algorithms available when H_k is a positive definite or positive semi-definite matrix. However, finding a global optimum for a QP with an indefinite matrix is NP-hard (Sahni, 1974). Many commercially available solvers that can solve optimization problems do not support solving indefinite QP or instead try to find an as good as possible local optimum by using different techniques such as gradient descent or barrier methods, which we review in the next section. One of the few commercial solvers that can deal with these kinds of problems is CPLEX, in which it only got supported recently, since 2014 (IBM, 2013). Note that we do not have any conditions on the objective in problem (1.2), that would guarantee that H_k is positive definite or positive semi-definite.

2.5.2 Barrier functions

Next we review the interior point method and logarithmic barrier functions. We review the formulations by Forsgren, Gill, and Wright (2002).

First we review how to deal with inequality constraints and later in this section we review how to deal with equality constraints $Ay=p$.
The logarithmic barrier function takes care of the inequality constraint by using the penalty function $\sum_{l=1}^{L}\log\left(b_l^\top y-q_l\right)$. This penalty grows to $-\infty$ if $b_l^\top y-q_l$ goes to 0. Problem (1.1) is then rewritten to the following problem.

$$\max_y S(y)+\mu\sum_{l=1}^{L}\log\left(b_l^\top y-q_l\right) \tag{2.21}$$

Where μ is a penalty parameter. When μ is very small then this problem is equivalent to (1.1), without the equality constraints $Ay=p$.
Problem (1.1) is then solved by applying (2.21) in an iterative manner. First a point on the interior the constraints is found, so a point for which $b_l^\top y-q_l>0,\ \forall l$. An interior point is needed, because if for some l, $b_l^\top y-q_l=0$, then $\log\left(b_l^\top y-q_l\right)=-\infty$. Then sequentially equation (2.21) is optimized and the value of μ is decreased at each sequential step. This also means that value of the penalty is reduced, so a sequential move to the optimal point is taken.

Therefore, a different approach is needed for equality constraints. For example the following

formulation can be used:

$$\max_{y \in \mathbb{R}^n} f(y) + \mu \sum_{l=1}^{L} \log\left(b_l^\top y - q_l\right) - \frac{1}{2\mu} \sum_{k=1}^{K} \left(a_k^\top y - p_k\right)^2 \tag{2.22}$$

When μ goes to zero, $\frac{1}{2\mu}$ goes to infinity. Therefore the penalty $\left(a_k^\top y - p_k\right)$ grows the further the algorithm progresses. When the algorithm terminates, it should find $\log\left(b_k^\top y - q_k\right)^2 = 0, \ \forall k$.

There are also methods that specifically use the linear shape of the equality constraints. It might be possible to circumvent a formulation such as problem (2.22) and to instead use (2.21) with linear equality constraints.

3. CASE MODEL

Throughout the thesis, we use terminology from the business case of the health-care company that we mentioned in Chapter 1. Our objective is maximizing the number of sales and we optimize over a number of weeks and a number of media types.
Let m be the number of media types, and t the number of weeks in which we can allocate money to each of the media types. We have that $m \approx 10$ and in general we optimize over a whole year, so we have $t = 53$ weeks, but this parameter could be used to restrict the optimization to a specific sequential set of weeks in a year.

3.1 Decision variables and objective

To model the problem we present the decision variable $y_{ij} \in \mathbb{R}_+$, which indicates the allocated money in week i on media j, where $i = 1, \ldots, t$ and $j = 1, \ldots, m$.
Next we define the matrix $Y = (y_{ij}) \in \mathbb{R}^{t \times m}$. We also define $y_{i\bullet}$ as the i^{th} row of Y, so $y_{i\bullet} = [y_{i1}, \ldots, y_{im}]$, $\forall i$. Similarly, we define $y_{\bullet j}$ as the j^{th} column of Y, so $y_{\bullet j} = [y_{1j}, \ldots, y_{tj}]^\top$, $\forall j$.

We want to maximize the total sales $S(Y) = \sum_{i=1}^{t} S_i(y_{1\bullet}, \ldots, y_{i\bullet})$, where $S_i(y_{1\bullet}, \ldots, y_{i\bullet})$ denote the sales in week i. Note that for a fixed i, $S_i(y_{1\bullet}, \ldots, y_{i\bullet})$ does not depend on $y_{k\bullet}$ with $k > i$. We also make the assumption that $S_i(y_{1\bullet}, \ldots, y_{i\bullet}) > 0$, $\forall i$, because sales are (generally) positive. The objective function of the here considered problem is:

$$\max_Y S(Y) = \max_Y \sum_{i=1}^{t} S_i(y_{1\bullet}, \ldots, y_{i\bullet}). \tag{3.1}$$

We use S and S_i as short notation for $S(Y)$ and $S_i(y_{1\bullet}, \ldots, y_{i\bullet})$ respectively.

In the econometric market model by GfK, the individual S_i are defined by the following objective function:

$$\ln(S_i) = c_{i0} + c_{i\text{lag}} \ln(S_{i-1}) + \sum_{j=1}^{m} c_{ij} \ln(y_{ij}), \quad i = 1, \ldots, t, \tag{3.2}$$

where $c_{ij} \in (-1, 1)$, $\forall i, j$, $c_{i\text{lag}} \in (0, 1)$, $\forall i$ and $c_{i0} \in \mathbb{R}$, $\forall i$.
There is a single period lag for S_i that is denoted by S_{i-1} and we assume that $S_0 > 0$ is given. The coefficient $c_{i\text{lag}}$ corresponds to the single period lag S_{i-1}, $\forall i$. For y_{ij}, there is a corresponding parameter c_{ij}, $\forall i, j$ and there is a parameter c_{i0} that can change over the weeks $\forall i$. By applying

the exponential function on both sides of equation (3.2), we get the following equation:

$$S_i = d_i \left(S_{i-1}\right)^{c_{i\text{lag}}} \prod_{j=1}^{m} \left(y_{ij}\right)^{c_{ij}}, \quad i = 1, \ldots, t, \tag{3.3}$$

where $d_i = \exp(c_{i0})$, $\forall i$.

3.2 Constraints

The following inequalities are the constraints of our model.

1. If we allocate money to a type of medium, then there is a minimum and maximum allowable allocation during each week:

$$M_{ij}^{\min} \leq y_{ij} \leq M_{ij}^{\max}, \quad i = 1, \ldots, t, \quad j = 1, \ldots, m. \tag{3.4}$$

   The parameter $M_{ij}^{\min}$ (resp. $M_{ij}^{\max}$) is the minimum (resp. maximum) amount of money we can spend during week i on media j. We make the assumption that $M_{ij}^{\min}$ is greater or equal to 1, because $y_{ij}^{c_{ij}}$ with $c_{ij} < 0$ rapidly grows to infinity when $0 < y_{ij} < 1$ and because the value might not be real for some c_{ij} if $y_{ij} \leq 0$.

2. For every week i there is a minimum and maximum allowable expenditure:

$$M_i^{\min} \leq \sum_{j=1}^{m} y_{ij} \leq M_i^{\max}, \quad i = 1, \ldots, t. \tag{3.5}$$

   The parameter $M_i^{\min}$ (resp. $M_i^{\max}$) is the minimum (resp. maximum) amount of money we can allocate to week i.

3. There is a minimum and maximum allowable expenditure on a media type j:

$$M_j^{\min} \leq \sum_{i=1}^{t} y_{ij} \leq M_j^{\max}, \quad j = 1, \ldots, m. \tag{3.6}$$

   The parameter $M_j^{\min}$ (resp. $M_j^{\max}$) is the minimum (resp. maximum) amount of money we can allocate to media j.

4. The total amount of money that we can allocate to all media and all weeks together should be below the available total budget.

$$\sum_{i=1}^{t} \sum_{j=1}^{m} y_{ij} \leq D. \tag{3.7}$$

   The parameter D is the total budget.

We combine the objective (3.1) with the constraints (3.4) to (3.7) and we get the following problem:

$$\begin{aligned}
\max_Y S(Y) = \max_Y \quad & \sum_{i=1}^{t} S_i\left(y_{1\bullet}, \ldots, y_{i\bullet}\right) \\
\text{s.t.} \quad & M_{ij}^{min} \leq y_{ij} \leq M_{ij}^{max}, \quad i = 1, \ldots, t, \quad j = 1, \ldots, m, \\
& M_i^{min} \leq \sum_{j=1}^{m} y_{ij} \leq M_i^{max}, \quad i = 1, \ldots, t, \\
& M_j^{min} \leq \sum_{i=1}^{t} y_{ij} \leq M_j^{max}, \quad j = 1, \ldots, m, \\
& \sum_{i=1}^{t} \sum_{j=1}^{m} y_{ij} \leq D, \\
& y_{ij} \in \mathbb{R}_+, \quad i = 1, \ldots, t, \quad j = 1, \ldots, m.
\end{aligned} \tag{3.8}$$

We see that we have $N = t \times m$ variables and $L = 2tm + 2m + 2t + 1$ linear inequality constraints. We mentioned before that a typical instance has $m \approx 10$ and $t = 53$. This means that we have a total of roughly 500 variables and 1200 constraints.
In Chapter 4, we present several algorithms to solve problem (3.8). These algorithms should solve our problem, with at least 500 variables and 1200 constraints, in a reasonable time.

In Section 4.2, we use Rosen's projection algorithm (Section 2.3), which uses a vectorized notation. Therefore, to keep similarity in the formulation between Section 2.3 and Section 4.2 we do not use problem (3.8). Instead we use in Section 4.2 and the sections after Section 4.2 a vectorized version of problem (3.8) (problem (4.9)).

3.3 Shape of the objective

In this section, we analyze the different shapes that the objective can have depending on the parameter c_{ij}, $\forall i, j$. For this we create several figures. For each of these figures, we use the following objective function:

$$S = \prod_{j=1}^{2} (y_{1j})^{c_{1j}} \tag{3.9}$$

This equation is equivalent to equation (3.3) with the following variables: $t = 1$, $m = 2$, $S_0 = 1$ and $d_1 = 1$.
Furthermore, we take $y_{1j} \in [1, 40]$, $\forall j$. The figures help us to give an idea of which techniques we can and cannot use in solving problem (3.8).

First we analyze the case when $c_{1j} > 0$, $\forall j$ and $\sum_{j=1}^{2} c_{1j} \leq 1$. In Figure 3.1 we can clearly see that the shape of the objective function under these conditions is concave. Under these conditions, it is possible to use gradient ascent with line search to reach a global maximum because the constraints form a convex set.

When $c_{1j} > 0$, $\forall j$, but instead $\sum_{j=1}^{2} c_{1j} > 1$, then we can see in Figure 3.2 that the objective function is concave in some directions, for example in $(0, y_{12})$, while it is convex in (y_{11}, y_{12}).
On first sight, this does not seem much of an issue, because it still possible to use gradient ascent with line search to find a global optimum on graphs that are similar to Graph 3.1. However, when more variables are added, then it could be possible that gradient ascent gets stuck in a local maximum.

Fig. 3.1: Example of the objective function for $c_{11} = c_{12} = 0.4$

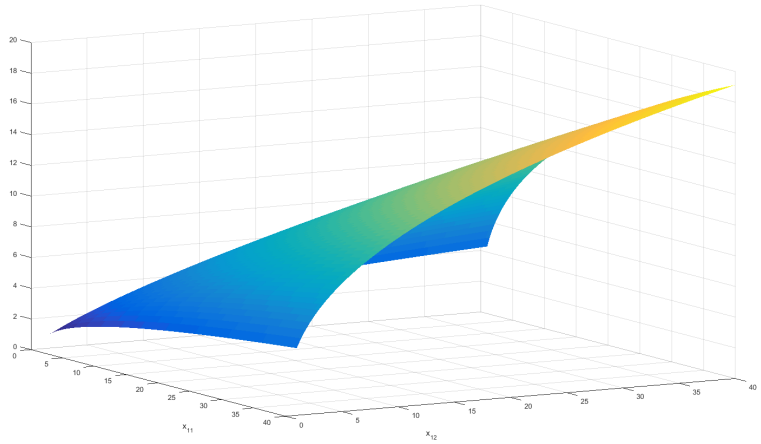
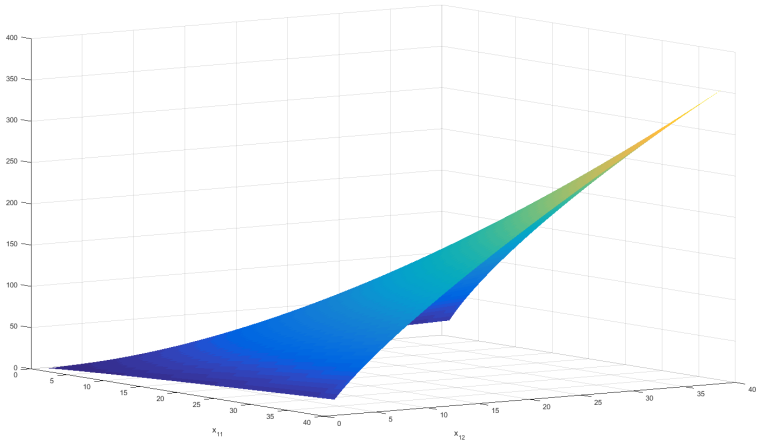


Fig. 3.2: Example of the objective function for $c_{11} = c_{12} = 0.8$



We also analyze the shape of the objective if $c_{1j} < 0, \ \forall j$. In this case the objective resembles Figure 3.3. This figure is clearly convex and this means that gradient ascent can get stuck in local optima. An example of such a local optima can occur if we add the constraints $y_{11} + y_{12} = 40$ and $y_{12} \leq 30$. In this case the point $(40, 0)$ is the global optimum and $(10, 30)$ is a local optimum. It is also not possible to reach the point $(40, 0)$ from the point $(10, 30)$ with gradient ascent. However, if the goal is to find a locally optimal point, then gradient ascent can of course still be used.

The final shape of the objective we analyze here is when the objective consists of both positive and negative coefficients c. If the objective has such coefficients, then we get, for the case with two variables, the graph that is shown in Figure 3.4. This graph is clearly concave in some directions and convex in other directions. There are similar issues regarding gradient ascent as in the case where $c_{1j} < 0, \ \forall j$.

Fig. 3.3: Example of the objective function for $c_{11} = c_{12} = -0.4$

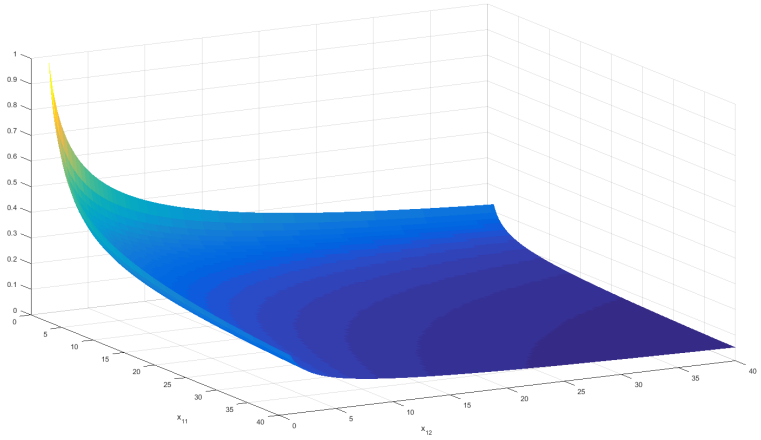
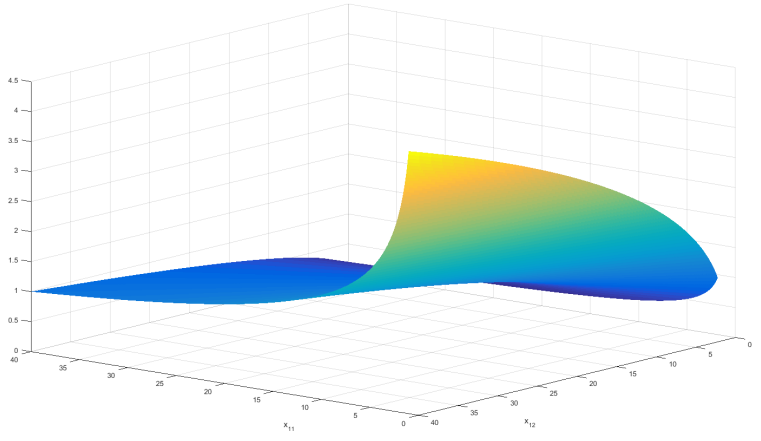


Fig. 3.4: Example of the objective function for $-c_{11} = c_{12} = 0.4$



4. PHASES OF THE ALGORITHM

In this chapter, we propose an algorithm to solve problem (3.8). We call this algorithm the complete algorithm.
The complete algorithm consists of two phases. In the first phase (Section 4.1) we first discard a few constraints of problem (3.8) to create a relaxed problem. In particular, the constraints on the maximum spending on a media type (3.6) and the maximum budget constraint (3.7), are dropped. This relaxation allows us to apply Lemma 2, which gives us a globally optimal solution to the relaxed problem.
In the second phase (Section 4.2) of the complete algorithm, we first add the constraints that were discarded in Phase 1. This means that the solution that we found in Phase 1 might be infeasible. Then we modify Rosen's projection algorithm such that it allows infeasible starting points. This modified version of Rosen's projection algorithm is used to find a feasible point for problem (3.8), using the point that was found in Phase 1. After this, the original version of Rosen's algorithm is used to improve this feasible point. After we have formulated the algorithm of Phase 2, we also present a few alternative versions of the algorithm of Phase 2 in Section 4.3. In Chapter 5, we test the complete algorithm on several datasets and we compare the results to the original version of Rosen's algorithm.
In the next section, we present Phase 1 of the complete algorithm.

4.1 Phase 1: Global maximum for a subset of the constraints

In this section, we present Phase 1 of the complete algorithm. This section consists of three subsections. In Subsection 4.1.1, we separate a relaxed version of problem (3.8) into subproblems that can be solved sequentially and that are easier to solve. In Subsection 4.1.2, we list the KKT conditions of problem (3.8).
In Subsection 4.1.3, we formulate some steps to solve the KKT conditions of problem (3.8). When solving the KKT conditions we split up the problem of solving the KKT conditions into three cases that are solved separately.

4.1.1 Creating subproblems

In this subsection, we present how we can sequentially solve subproblems of a relaxed version of problem (3.8). For Phase 1 we want to maximize the sales $S(Y)$, using only constraint (3.4) and

(3.5). We can write this in the following way:

$$\begin{array}{lll} \max_Y S(Y) = & \max_Y \sum_{i=1}^{t} S_i\left(y_{1\bullet}, \ldots, y_{i\bullet}\right) & \\ \text{s.t.} & M_{ij}^{\min} \leq y_{ij} \leq M_{ij}^{\max}, \quad i = 1, \ldots, t, \quad j = 1, \ldots, m, & \\ & M_i^{\min} \leq \sum_{j=1}^{m} y_{ij} \leq M_i^{\max}, \quad i = 1, \ldots, t, & \\ & y_{ij} \in \mathbb{R}_+, \quad i = 1, \ldots, t, \quad j = 1, \ldots, m. & \end{array} \tag{4.1}$$

To solve problem (4.1), we separate the problem in subproblems that are easier to solve. We use Lemma 2 to separate the problem in subproblems.

Lemma 2. *Assume that problem* (4.1) *is not infeasible, then maximization problem* (4.1) *can be solved in a sequential order starting at week 1.*

Proof. From equation (3.3), we know that $S_1\left(y_{1\bullet}\right) = d_1\left(S_0\right)^{c_{1\text{lag}}} \prod_{j=1}^{m} \left(y_{1j}\right)^{c_{1j}}$, where S_0 is given. Recall that $S_1\left(y_{1\bullet}\right)$ only depends on $y_{1\bullet}$ and not on $y_{i\bullet}$, $i = 2, \ldots, t$.
Let us denote the set of feasible points in problem (4.1) by Z, where

$$Z := \{Y | M_{ij}^{\min} \leq y_{ij} \leq M_{ij}^{\max}, M_i^{\min} \leq \sum_{j=1}^{m} y_{ij} \leq M_i^{\max}, \forall i, j\}.$$

We also see in problem (4.1), that in week 1, $y_{1\bullet}$ appears only in the following constraints: $Z_1 := \left\{ y_{1\bullet} | M_1^{\min} \leq \sum_{j=1}^{m} y_{1j} \leq M_1^{\max}, M_{1j}^{\min} \leq y_{1j} \leq M_{1j}^{\max}, \forall j \right\}$. Therefore, $y_{1\bullet}$ is independent of any constraint outside of week 1. This implies that, $\max_{Y \in Z} S_1\left(y_{1j}\right) = \max_{y_{1\bullet} \in Z_1} S_1\left(y_{1\bullet}\right)$.

For week 2, we know that $S_2\left(y_{2\bullet}\right) = d_2\left(S_1\left(y_{1\bullet}\right)\right)^{c_{2\text{lag}}} \prod_{j=1}^{m} \left(y_{2j}\right)^{c_{2j}}$. We see that $S_2\left(y_{2\bullet}\right)$ depends on S_1 and on $y_{2\bullet}$, $\forall j\}$. Analogous to week 1, $y_{2\bullet}$, $\forall j$ appears only in the following constraints: $Z_2 := \left\{ y_{2\bullet} | M_2^{\min} \leq \sum_{j=1}^{m} y_{2j} \leq M_2^{\max}, M_{2j}^{\min} \leq y_{2j} \leq M_{2j}^{\max}, \forall j \right\}$. Since $c_{i\text{lag}} > 0$, $\forall i$, it follows that $\left(\max_{y_{1\bullet} \in Z_1} [S_1\left(y_{1\bullet}\right)]\right)^{c_{2\text{lag}}} = \max_{y_{1\bullet} \in Z_1} \left(S_1\left(y_{1\bullet}\right)\right)^{c_{2\text{lag}}}$

Therefore, $\max_{Y \in Z} S_2\left(y_{1\bullet}, y_{2\bullet}\right) = d_2\left(\max_{Y \in Z} S_1\left(y_{1\bullet}\right)\right)^{c_{2\text{lag}}} \max_{y_{2\bullet} \in Z_2} \prod_{j=1}^{m} \left(y_{2j}\right)^{c_{2j}}$.
This implies that if we have calculated $\max_{Y \in Z} S_1$, then we can obtain $\max_{Y \in Z} S_2$, by only optimizing over $y_{2\bullet} \in Y_2$.

Using similar arguments, we can show that if we have $\max_{Y \in Z} S_{i-1}$ then we can obtain $\max_{Y \in Z} S_i$ for $i > 2$. We know that $\max_{Y \in Z} S_1$ does not depend on unknown past periods, only on the given S_0, so we can sequentially calculate all $\max_{Y \in Z} S_i$ for $i = 1, \ldots, t$. So $\max_{Y \in Z} \sum_{i=1}^{t} S_i = \sum_{i=1}^{t} \max_{Y \in Z} S_i$, which can be solved sequentially.

□

Lemma 2 proves that problem (4.1) can be solved sequentially. Therefore, if any week i is fixed, then we can use that S_{i-1} and $y_{1\bullet}, \ldots, y_{(i-1)\bullet}$ are known. We present a method to solve each of the subproblems in Subsection 4.1.2.

4.1.2 KKT conditions

In this section, we solve the subproblems that we have found in Subsection 4.1.1. First we fix week i. This allows us to use that S_{i-1} and $y_{1\bullet}, \ldots, y_{(i-1)\bullet}$ are known. Now we first show that if the KKT conditions of problem (4.1) are not infeasible, then we find a single KKT point $(y_{i\bullet}, \gamma^+, \mu_1^+, \ldots, \mu_m^+, \gamma^-, \mu_1^-, \ldots, \mu_m^-)$ that satisfies the KKT conditions.
The KKT multipliers $\gamma^- \in \mathbb{R}_+$ and $\gamma^+ \in \mathbb{R}_+$ belong to the the constraint $M_i^{\min} \leq \sum_{j=1}^{m} y_{ij}$ and the constraint $\sum_{j=1}^{m} y_{ij} \leq M_i^{\max}$ in week i, respectively. These constraints are the two parts of constraint (3.5) in week i.
For all $j = 1, \ldots, m$, we have that the KKT multipliers $\mu_j^- \in \mathbb{R}_+$ and $\mu_j^+ \in \mathbb{R}_+$ belong to the constraint $M_{ij}^{\min} \leq y_{ij}$ and the constraint $y_{ij} \leq M_{ij}^{\max}$ in week i, respectively. These constraints are the two parts of constraint (3.4) in week i. Now we find the following system of equations for any week i:

$$\nabla_{y_{i\bullet}} S_i(y_{1\bullet}, \ldots, y_{i\bullet}) = \mathbf{e}\left(\gamma^+ - \gamma^-\right) + \sum_{j=1}^{m} \mathbf{e}_j \left(\mu_j^+ - \mu_j^-\right), \tag{4.2a}$$

$$-y_{ij} \geq -M_{ij}^{\max}, \qquad j = 1, \ldots, m, \tag{4.2b}$$

$$y_{ij} \geq M_{ij}^{\min}, \qquad j = 1, \ldots, m, \tag{4.2c}$$

$$-\sum_{j=1}^{m} y_{ij} \geq -M_i^{\max}, \tag{4.2d}$$

$$\sum_{j=1}^{m} y_{ij} \geq M_i^{\min}, \tag{4.2e}$$

$$\mu_j^+, \mu_j^-, \gamma^+, \gamma^- \geq 0, \qquad j = 1, \ldots, m, \tag{4.2f}$$

$$\mu_j^+ \left(-y_{ij} + M_{ij}^{\max}\right) = 0, \qquad j = 1, \ldots, m, \tag{4.2g}$$

$$\mu_j^- \left(y_{ij} - M_{ij}^{\min}\right) = 0, \qquad j = 1, \ldots, m, \tag{4.2h}$$

$$\gamma^+ \left(-\sum_{j=1}^{m} y_{ij} + M_i^{\max}\right) = 0, \tag{4.2i}$$

$$\gamma^- \left(\sum_{j=1}^{m} y_{ij} - M_i^{\min}\right) = 0, \tag{4.2j}$$

where $\mathbf{e}$ is an m-dimensional vector or ones and $\mathbf{e}_j$ is an m-dimensional vector of zeros with a one at the j^{th} entry of the vector.

For equation (4.2a) we first need to calculate the derivative of S_i with regard to y_{ij}, $\forall j$ and the gradient of $S_i(y_{1\bullet}, \ldots, y_{i\bullet})$ with regard to $y_{i\bullet}$, where S_i is as in equation (3.3). This results in:

$$\frac{\partial S_i}{\partial y_{ij}} = S_i \frac{c_{ij}}{y_{ij}}, \quad \nabla_{y_{i\bullet}} S_i(y_{1\bullet}, \ldots, y_{i\bullet}) = \left(\frac{\partial S_i}{\partial y_{i1}}, \ldots, \frac{\partial S_i}{\partial y_{im}}\right)^\top = \left(S_i \frac{c_{i1}}{y_{i1}}, \ldots, S_i \frac{c_{im}}{y_{im}}\right)^\top \tag{4.3}$$

Remark 3. *We know that y_{ij} and S_i are positive, so if, for some j, c_{ij} is positive then $\frac{\partial S_i}{\partial y_{ij}}$ is positive. If, for some j, c_{ij} is negative then $\frac{\partial S_i}{\partial y_{ij}}$ is negative. If for some j, $c_{ij} = 0$ then $\frac{\partial S_i}{\partial y_{ij}} = 0$.*

Constraint (3.5) implies that $\sum_{j=1}^{m} y_{ij}$ is at least $M_i^{\min}$ and at most $M_i^{\max}$. We separate the possible solutions of the system of KKT equations (4.2a) to (4.2j) into three disjoint solution types depending on the value of $\sum_{j=1}^{m} y_{ij}$. In Subsection 4.1.3, we find solutions to the system (4.2a) to (4.2j) using these solution types. A solution of Type 1 satisfies $M_i^{\min} < \sum_{j=1}^{m} y_{ij} < M_i^{\max}$. A solution of Type 2 satisfies $\sum_{j=1}^{m} y_{ij} = M_i^{\max}$. Finally, a solution of Type 3 satisfies the only remaining option $\sum_{j=1}^{m} y_{ij} = M_i^{\min}$.

For clarity of the following section, we make the assumption here that $c_{ij} \neq 0$, $\forall j$ and $M_i^{\min} < M_i^{\max}$. Recall that i is still fixed. At the end of Subsection 4.1.3, we briefly analyze what happens when $c_{ij} \neq 0$ for some j or when $M_i^{\min} = M_i^{\max}$.

In the following three lemmas we prove some observations for each of the solution types. We start with the lemma for Solution Type 1.

Lemma 4. *Let i be a fixed value between 1 to t. Let $j = 1, \ldots, m$. Assume that the system (4.2a) to (4.2j) has a solution of Type 1. Then the following is the unique solution to (4.2a) to (4.2j) of Type 1: $\gamma^+, \gamma^- = 0$; $y_{ij} = M_{ij}^{\max}$, $\mu_j^+ = \frac{\partial S_i}{\partial y_{ij}}$ and $\mu_j^- = 0$, if $c_{ij} > 0$ for some j; $y_{ij} = M_{ij}^{\min}$, $\mu_j^+ = 0$ and $\mu_j^- = -\frac{\partial S_i}{\partial y_{ij}}$, if $c_{ij} < 0$ for some j.*

Proof. A solution of Type 1 implies $M_i^{\min} < \sum_{j=1}^{m} y_{ij} < M_i^{\max}$. This implies that the KKT conditions (4.2d), (4.2e) are satisfied, that (4.2i) and (4.2j) are only satisfied if $\gamma^+, \gamma^- = 0$ and that the KKT condition (4.2a) is now: $\nabla_{y_{i\bullet}} S_i(y_{1\bullet}, \ldots, y_{i\bullet}) = \sum_{j=1}^{m} \mathbf{e}_j \left(\mu_j^+ - \mu_j^-\right)$.

Therefore KKT condition (4.2a) and (4.2f) imply that if for some j, $\frac{\partial S_i}{\partial y_{ij}} > 0$, then $\frac{\partial S_i}{\partial y_{ij}} = \mu_j^+ > 0$. By Remark 3, we know that $\frac{\partial S_i}{\partial y_{ij}} > 0$ if $c_{ij} > 0$. Now (4.2g) implies $y_{ij} = M_{ij}^{\max}$ and (4.2h) implies $\mu_j^- = 0$.

Similarly if $\frac{\partial S_i}{\partial y_{ij}} < 0$, then $-\frac{\partial S_i}{\partial y_{ij}} = \mu_j^- > 0$. Now (4.2h) implies $y_{ij} = M_{ij}^{\min}$ and (4.2g) implies $\mu_j^+ = 0$. We find a feasible solution, so also the KKT conditions (4.2b) and (4.2c) are satisfied.

□

Now we state the lemma for Solution Type 2.

Lemma 5. *Let i be a fixed value between 1 to t. Let $j = 1, \ldots, m$. Assume that the system (4.2a) to (4.2j) has a solution of Type 2. Then the following is the solution to (4.2a) to (4.2j) of Type 2: $\gamma^- = 0$; $y_{ij} = M_{ij}^{\min}$, $\mu_j^+ = 0$ and $\mu_j^- = \gamma^+ - \frac{\partial S_i}{\partial y_{ij}}$, if $c_{ij} < 0$ for some j;*
If $c_{ij} > 0$, for some j, then:
either $\gamma^+ = \frac{\partial S_i}{\partial y_{ij}}$, $\mu_j^+ = \mu_j^- = 0$ and $M_{ij}^{\min} \leq y_{ij} \leq M_{ij}^{\max}$;
or $\gamma^+ = \frac{\partial S_i}{\partial y_{ij}} - \mu_j^+$, $\mu_j^+ > 0$, $\mu_j^- = 0$ and $y_{ij} = M_{ij}^{\max}$;
or $\gamma^+ = \frac{\partial S_i}{\partial y_{ij}} + \mu_j^-$, $\mu_j^+ = 0$, $\mu_j^- > 0$ and $y_{ij} = M_{ij}^{\min}$.

Proof. A solution of Type 2 implies $\sum_{j=1}^{m} y_{ij} = M_i^{\max}$. This means that KKT conditions (4.2d), (4.2e) and (4.2i) are satisfied. We also know that KKT condition (4.2j) implies that $\gamma^- = 0$. The

KKT multiplier $\gamma^- = 0$ implies that the following holds for equation (4.2a): $\nabla_{y_{i\bullet}} S_i(y_{1\bullet}, \ldots, y_{i\bullet}) = \mathbf{e}\gamma^+ + \sum_{j=1}^{m} \mathbf{e}_j \left(\mu_j^+ - \mu_j^-\right)$.

Therefore KKT condition (4.2a) and (4.2f) imply that if for some j, $\frac{\partial S_i}{\partial y_{ij}} < 0$, then $-\frac{\partial S_i}{\partial y_{ij}} + \gamma^+ = \mu_j^- > 0$. By Remark 3, we know that $\frac{\partial S_i}{\partial y_{ij}} < 0$ if $c_{ij} < 0$. KKT condition (4.2h) implies $y_{ij} = M_{ij}^{\min}$. Now (4.2g) implies that $\mu_j^+ = 0$.
If instead for some j, $\frac{\partial S_i}{\partial y_{ij}} > 0$, then KKT condition (4.2a) implies one of the following:
either $\frac{\partial S_i}{\partial y_{ij}} = \gamma^+$ and $\mu_j^+ = \mu_j^- = 0$, which satisfies the KKT conditions (4.2f), (4.2g) and (4.2h). Now KKT conditions (4.2b) and (4.2c) imply that $M_{ij}^{\min} \leq y_{ij} \leq M_{ij}^{\max}$;
or $\frac{\partial S_i}{\partial y_{ij}} = \gamma^+ + \mu_j^+$, with $\mu_j^+ > 0$ and so $y_{ij} = M_{ij}^{\max}$ by KKT condition (4.2g). KKT condition (4.2h) is satisfied by $\mu_j^- = 0$;
or $\frac{\partial S_i}{\partial y_{ij}} = \gamma^+ - \mu_j^-$, with $\mu_j^- > 0$ and so $y_{ij} = M_{ij}^{\min}$ by KKT condition (4.2h). KKT condition (4.2g) is satisfied by $\mu_j^+ = 0$.
We find a feasible solution, so also the KKT conditions (4.2b) and (4.2c) are satisfied.

□

In other words Lemma 5 proves that, if feasible, a solution of Type 2 needs to have the same value for $\frac{\partial S_i}{\partial y_{ij}}$ for all j for which $c_{ij} > 0$. This might result in y_{ij} for some j that are not feasible for KKT conditions (4.2b) or (4.2c). If some y_{ij} are infeasible, then the values of the KKT multipliers μ_j^+ or μ_j^- are chosen to be positive. Positive values for μ_j^+ or μ_j^- imply that y_{ij} is feasible and either equal to $M_{ij}^{\max}$ or $M_{ij}^{\min}$.
Whether $\mu_j^+ > 0$ and $\mu_j^- = 0$, $\mu_j^+ = 0$ and $\mu_j^- > 0$, or μ_j^+, $\mu_j^- = 0$ depends on the value of γ^+. In Subsection 4.1.3, we further analyze the solution of Type 2 and the KKT conditions.

Finally we state Lemma 6 for solutions of Type 3. Lemma 5 is similar to Lemma 6. Therefore, we only state the lemma and do not give a proof for Lemma 6.

Lemma 6. *Let i be a fixed value between 1 to t. Let $j = 1, \ldots, m$. $j = 1, \ldots, m$. Assume that the system* (4.2a) *to* (4.2j) *has a solution of Type 3. Then the following is the solution to* (4.2a) *to* (4.2j) *of Type 3: $\gamma^+ = 0$; $y_{ij} = M_{ij}^{\max}$, $\mu_j^+ = \gamma^+ + \frac{\partial S_i}{\partial y_{ij}}$ and $\mu_j^- = 0$, if $c_{ij} > 0$ for some j;*
If $c_{ij} > 0$, for some j, then:
either $\gamma^- = -\frac{\partial S_i}{\partial y_{ij}}$, $\mu_j^+ = \mu_j^- = 0$ and $M_{ij}^{\min} < y_{ij} < M_{ij}^{\max}$;
or $\gamma^- = -\frac{\partial S_i}{\partial y_{ij}} + \mu_j^+$, $\mu_j^+ > 0$, $\mu_j^- = 0$ and $y_{ij} = M_{ij}^{\max}$;
or $\gamma^- = -\frac{\partial S_i}{\partial y_{ij}} - \mu_j^-$, $\mu_j^+ = 0$, $\mu_j^- > 0$ and $y_{ij} = M_{ij}^{\min}$.

Lemma 7 shows that if we find a solution of Type 1 that solves the system (4.2a) to (4.2j), then there is no solution of Type 2 or Type 3.

Lemma 7. *Let i be any fixed value from 1 to t. Let $c_{ij} \neq 0$ for $j = 1, \ldots, m$. Let $y_{i\bullet}^{*\top} \in \mathbb{R}^m$, $\mu^{+*}, \mu^{-*} \in \mathbb{R}_+$ and $\mu_j^{+*}, \mu_j^{-*} \in \mathbb{R}_+$ for $j = 1, \ldots, m$.*
Let the point $\left(y_{i\bullet}^*, \mu^{+*}, \mu_1^{+*}, \ldots, \mu_m^{+*}, \mu^{-*}, \mu_1^{-*}, \ldots, \mu_m^{-*}\right)$ be a solution to the system (4.2a) *to* (4.2j) *of Type 1. Then there are no solutions to the system* (4.2a) *to* (4.2j) *of Type 2 or Type 3 and the point $\left(y_{i\bullet}^*, \mu^{+*}, \mu_1^{+*}, \ldots, \mu_m^{+*}, \mu^{-*}, \mu_1^{-*}, \ldots, \mu_m^{-*}\right)$ is the unique solution.*

Proof. Lemma 4 proved that if there is a solution of Type 1, then it is the only solution of Type 1. To prove Lemma 7 we have to show that there cannot be a solution of Type 2 or Type 3 if there

is a solution of Type 1. This proof is by contradiction.

Let the point $\left(\hat{y}_{i\bullet}, \hat{\mu}^{+}, \hat{\mu}_1^{+}, \ldots, \hat{\mu}_m^{+}, \hat{\mu}^{-}, \hat{\mu}_1^{-}, \ldots, \hat{\mu}_m^{-}\right)$ be a solution to the system (4.2a) to (4.2j) of either Type 2 of Type 3. So a solution that either satisfies the constraint $\sum_{j=1}^{m} y_{ij} = M_i^{\min}$ or the constraint $\sum_{j=1}^{m} y_{ij} = M_i^{\max}$.

If $\hat{y}_{i\bullet}$ satisfies $\sum_{j=1}^{m} \hat{y}_{ij} = M_i^{\max}$, then we can make the following observation.

$$M_i^{\max} = \sum_{j=1}^{m} \hat{y}_{ij} \leq \sum_{j|c_{ij}>0} M_{ij}^{\max} + \sum_{j|c_{ij}<0} M_{ij}^{\min} = \sum_{j=1}^{m} y_{ij}^{*} < M_i^{\max}. \tag{4.4}$$

The first inequality in equation (4.4) holds, because if $\frac{\partial S_i}{\partial y_{ij}} < 0$ (or equivalently $c_{ij} < 0$) for some j, then $\hat{y}_{ij} = M_{ij}^{\min}$ (Lemma 5). All other y_{ij} are at most $M_{ij}^{\max}$. The second equality follows from Lemma 4. The second inequality holds because $y_{i\bullet}^{*}$ is a solution of Type 1. Therefore, equation (4.4) gives a contradiction.

Similarly, if $\hat{y}_{i\bullet}$ satisfies $\sum_{j=1}^{m} \hat{y}_{ij} = M_i^{\min}$, then we can make the following observation.

$$M_i^{\min} = \sum_{j=1}^{m} \hat{y}_{ij} \geq \sum_{j|c_{ij}>0} M_{ij}^{\max} + \sum_{j|c_{ij}<0} M_{ij}^{\min} = \sum_{j=1}^{m} y_{ij}^{*} > M_i^{\min}.$$

This is also a contradiction.

So the point $\left(y_{i\bullet}^{*}, \mu^{+*}, \mu_1^{+*}, \ldots, \mu_m^{+*}, \mu^{-*}, \mu_1^{-*}, \ldots, \mu_m^{-*}\right)$ is the unique point that satisfies the system (4.2a) to (4.2j).

□

In a similar way to Lemma 7, it can be proven that if a solution of Type 2 is found that satisfies the system of KKT conditions (4.2a) to (4.2j), then there is no solution of Type 3 that satisfies system (4.2a) to (4.2j).

In Subsection 4.1.3, we use Lemma 4 to Lemma 7 and Solution Type 1, Type 2 and Type 3 to find a solution to the system of KKT conditions (4.2a) to (4.2j).

4.1.3 Solving the KKT conditions

In Subsection 4.1.2, we presented three disjoint solution types. In this subsection we solve each of these solution types separately. For each of the solution types, we want to find a KKT point $\left(y_{i\bullet}^{*}, \mu^{+*}, \mu_1^{+*}, \ldots, \mu_m^{+*}, \mu^{-*}, \mu_1^{-*}, \ldots, \mu_m^{-*}\right)$ that is a solution to the system (4.2a) to (4.2j).

First we consider Solution Type 1. This solution type satisfies $M_i^{\min} < \sum_{j=1}^{m} y_{ij}^{*} < M_i^{\max}$. Recall from Lemma 4 that if for some j, $c_{ij} > 0$, then $y_{ij}^{*} = M_{ij}^{\max}$ and if for some j, $c_{ij} < 0$, then $y_{ij}^{*} = M_{ij}^{\min}$.

We can quickly verify if $M_i^{\min} < \sum_{j=1}^{m} y_{ij}^{*} < M_i^{\max}$. If the point $y_{i\bullet}^{*} = (y_{i1}^{*}, \ldots, y_{im}^{*})$ lies within this constraint, then the point

$$\left(y_{i\bullet}, \gamma^{+}, \mu_1^{+}, \ldots, \mu_m^{+}, \gamma^{-}, \mu_1^{-}, \ldots, \mu_m^{-}\right) = \left(y_{i\bullet}^{*}, 0, \mu_1^{+*}, \ldots, \mu_m^{+*}, 0, \mu_1^{-*}, \ldots, \mu_m^{-*}\right)$$

satisfies the system of KKT conditions (4.2a) to (4.2j). Here $\mu_j^{+*} = \frac{\partial S_i}{\partial y_{ij}}$ and $\mu_j^{-*} = 0$ if $\frac{\partial S_i}{\partial y_{ij}} > 0$, otherwise $\mu_j^{+*} = 0$ and $\mu_j^{-*} = -\frac{\partial S_i}{\partial y_{ij}}$, $\forall j$. Recall from Lemma 7 that if we have found a KKT point for Solution Type 1, then there are no solutions of the other solution types. Therefore we can skip the Solution Type 2 Solution Type 3.
If the point $y_{i\bullet}^*$ does not satisfy the inequality $M_i^{\min} < \sum_{j=1}^{m} y_{ij} < M_i^{\max}$. Then we cannot find a KKT point with Solution Type 1, so we have to find a KKT point with Solution Type 2 or Solution Type 3.

Now we consider Solution Type 2. We first show that the system of KKT conditions (4.2a) to (4.2j) implies that y_{ij} has to satisfy the following function for any positive variable θ^+:

$$y_{ij}(\theta^+) = \begin{cases} M_{ij}^{\min} & \text{if } c_{ij}\theta^+ < M_{ij}^{\min} \\ c_{ij}\theta^+ & \text{if } M_{ij}^{\min} \leq c_{ij}\theta^+ \leq M_{ij}^{\max} \\ M_{ij}^{\max} & \text{if } c_{ij}\theta^+ > M_{ij}^{\max} \end{cases} \quad \forall j \in \mathcal{G}, \tag{4.5}$$

where $\mathcal{G} := \{j \,|c_{ij} > 0\}$.
After we have shown how to find function (4.5), we show how to use function (4.5) to find a point that satisfies the system of KKT conditions (4.2a) to (4.2j).

Remark 8. *Observe that function* (4.5) *is nondecreasing. We can see that y_{ij}, for all $j \in \mathcal{G}$ are linear functions of θ^+ We also know that c_{ij} is positive for all $j \in \mathcal{G}$. Therefore, y_{ij} is a nondecreasing function of θ^+ for all $j \in \mathcal{G}$.*

Note that Remark 8 implies that there is only a single point $(y_{i1}, \ldots, y_{im})$ that can satisfy the equation $\sum_{j=1}^{m} y_{ij} = M_i^{\max}$ of Solution Type 2. Therefore, the point $(y_{i1}^*, \ldots, y_{im}^*)$ is unique. This does not imply that the KKT multipliers $\left(\gamma^+, \mu_1^+, \ldots, \mu_m^+, \gamma^-, \mu_1^-, \ldots, \mu_m^-\right)$ have to be unique. Later in this section, we observe that if θ^+ can change without changing the point $(y_{i1}^*, \ldots, y_{im}^*)$, then the KKT point that satisfies the system (4.2a) to (4.2j) is not unique.

For a solution of Solution Type 2, we know that Lemma 5 holds. Recall from Lemma 5 that $\gamma^{-*} = 0$ and that for all j for which c_{ij} is negative, we can set $y_{ij}^* = M_{ij}^{\min}$ and $\mu_j^{+*} = 0$.
The set $\mathcal{G}$ is therefore also the set of all j for which no value is assigned to y_{ij}^*. We now fix the ratio $\frac{S_i}{\gamma^+} = \theta^+$. We use this ratio and Lemma 5 to construct each of the cases of function (4.5). Now we choose some $\hat{\theta}^+ > 0$.
We know from Lemma 5 that we can find three different solutions for all $j \in \mathcal{G}$. We first consider for some $j \in \mathcal{G}$ the solution for which $\gamma^+ = \frac{\partial S_i}{\partial y_{ij}} = \frac{c_{ij}S_i}{y_{ij}}$ and $M_{ij}^{\min} \leq y_{ij} \leq M_{ij}^{\max}$.
The equation $\frac{c_{ij}S_i}{y_{ij}} = \gamma^+$ implies that $y_{ij} = c_{ij}\hat{\theta}^+$. If $M_{ij}^{\min} \leq c_{ij}\hat{\theta}^+ \leq M_{ij}^{\max}$ then $\mu_j^+ = \mu_j^- = 0$ as implied by Lemma 5. If instead $c_{ij}\hat{\theta}^+ > M_{ij}^{\max}$ or $c_{ij}\hat{\theta}^+ < M_{ij}^{\min}$, then we have to consider one of the other two KKT solutions of Lemma 5.

Say that we have $c_{ij}\hat{\theta}^+ > M_{ij}^{\max}$. We now show that $y_{ij} = M_{ij}^{\max}$. We observe the following:

$$c_{ij}\theta^+ > M_{ij}^{\max} \implies \frac{c_{ij}S_i}{M_{ij}^{\max}}\theta^+ > \gamma^+ \tag{4.6}$$

We can combine equation (4.6) with the equality $\mu_j^+ = \frac{c_{ij}S_i}{y_{ij}} - \gamma^+$ to obtain the following inequality:

$$\mu_j^+ = \frac{c_{ij}S_i}{y_{ij}} - \gamma^+ > \frac{c_{ij}S_i}{y_{ij}} - \frac{c_{ij}S_i}{M_{ij}^{\max}} \geq 0 \tag{4.7}$$

Recall that if $\mu_j^+ > 0$, then $y_{ij} = M_{ij}^{\max}$, $\gamma^+ = \frac{\partial S_i}{\partial y_{ij}} - \mu_j^+$ and $\mu_j^- = 0$ (Lemma 5). Therefore, $c_{ij}\hat{\theta}^+ > M_{ij}^{\max}$ implies $y_{ij} = M_{ij}^{\max}$.
Similarly, if for any $j \in \mathcal{G}$ we have $c_{ij}\hat{\theta}^+ < M_{ij}^{\min}$, then $\mu_j^- > 0$. Therefore, $y_{ij} = M_{ij}^{\min}$, $\mu_j^+ = 0$ and $\gamma^- = -\frac{\partial S_i}{\partial y_{ij}} - \mu_j^-$ (Lemma 5).

Now we have shown all cases of function (4.5). What remains is to show how to calculate the KKT multipliers γ^+, μ_j^+ and μ_j^-. We can observe that function (4.5) and $\hat{\theta}^+$ imply that we know y_{ij}, for all $j \in \mathcal{G}$. Recall that $y_{1\bullet}, \ldots, y_{(i-1)\bullet}$ are known (Lemma 2) and that we also know y_{ij} for $j \notin \mathcal{G}$. This allows us to calculate $S_i(y_{1\bullet}, \ldots, y_{(i-1)\bullet}, y_{i\bullet})$. So we can calculate γ^+ with the equation $\gamma^+ = \frac{S_i}{\hat{\theta}^+}$. This implies that we can also calculate $\mu_j^+ = \frac{c_{ij}S_i}{y_{ij}} - \gamma^+$ and $\mu_j^- = \frac{c_{ij}S_i}{y_{ij}} + \gamma^+$. Now we know how to calculate all KKT multipliers for a given $\hat{\theta}^+$.
Using equation (4.5) and equation $\gamma^+ = \frac{S_i}{\theta^+}$, we observe that two different values of θ^+ can have the same value of y_{ij}, but different KKT multipliers. Recall that this implies that a solution to the system (4.2a) to (4.2j) might not be unique.

To summarize, we use function (4.5) and some value of θ^+ to find y_{ij} for all $j \in \mathcal{G}$. We use Lemma 5 to find the KKT multipliers. We now satisfy all the system of KKT conditions (4.2a) to (4.2j), except for (4.2i). We need a solution of Type 2, so we should find $\sum_{j=1}^{m} y_{ij}^* = M_i^{\max}$. If $\sum_{j=1}^{m} y_{ij}^* = M_i^{\max}$, then also the KKT condition (4.2i) is satisfied. If there is a value for θ^+ such that $\sum_{j=1}^{m} y_{ij} = M_i^{\max}$, then we define this value as θ^{+*}. Now we show that we find the following equation for θ^{+*}:

$$\theta^{+*} = \frac{M_i^{\max} - \sum_{j \in \mathcal{G}} y_{ij}(\theta^{+1}) - \sum_{j \notin \mathcal{G}} y_{ij}^*}{\sum_{j \in \mathcal{H}} c_{ij}}, \tag{4.8}$$

where $\mathcal{H} \subseteq \mathcal{G}$ and θ^{+1} is a special value of θ^+. Both $\mathcal{H}$ and θ^{+1} are more thoroughly defined later in this section.

We can use equation (4.5) to rewrite the equation of Solution Type 2, $\sum_{j=1}^{m} y_{ij} = M_i^{\max}$. We get $\sum_{j=1}^{m} y_{ij} = \sum_{j \in \mathcal{G}} y_{ij}(\theta^+) + \sum_{j \notin \mathcal{G}} y_{ij}^* = M_i^{\max}$. We know from Lemma 5 and equation (4.5) that if we find a solution of Type 2, then we find a KKT point. Equation (4.5) does not give us a solution of Type 2 for all θ^+, so we need to adjust θ^+ to find a solution of Type 2.

Recall that y_{ij} is a nondecreasing function of θ^+ for all $j \in \mathcal{G}$ (Remark 8). Therefore, for a

fixed week i, we can calculate all $2|\mathcal{G}|$ points where $\theta^+ = \frac{M_{ij}^{\min}}{c_{ij}}$ or where $\theta^+ = \frac{M_{ij}^{\max}}{c_{ij}}$, $\forall j \in \mathcal{G}$. Then we sort all $2|\mathcal{G}|$ points θ^+ and find for every point θ^+ all corresponding $y_{ij}(\theta^+)$, $\forall j \in \mathcal{G}$.
We can then verify for each of these θ^+ if they satisfy $\sum_{j\in\mathcal{G}} y_{ij}(\theta^+) + \sum_{j\notin\mathcal{G}} y_{ij}^* \leq M_i^{\max}$. We define θ^{+1} as the biggest of the $2|\mathcal{G}|$ points θ^+ for which $\sum_{j\in\mathcal{G}} y_{ij}(\theta^+) + \sum_{j\notin\mathcal{G}} y_{ij}^* < M_i^{\max}$. We also define θ^{+2} as the smallest of the $2|\mathcal{G}|$ points θ^+ for which $\sum_{j\in\mathcal{G}} y_{ij}(\theta^+) + \sum_{j\notin\mathcal{G}} y_{ij}^* \geq M_i^{\max}$. We know, because y_{ij} for all $j \in \mathcal{G}$ is a nondecreasing linear function of θ^+, that θ^{+*} lies on a straight line between θ^{+1} and θ^{+2}.
To find θ^{+*}, we evaluate equation (4.5) at the points θ^{+1} and θ^{+2} and identify the j's for which y_{ij} changed. In other words, we define the following set: $\mathcal{H} = \{j \mid \left|y_{ij}(\theta^{+1}) - y_{ij}(\theta^{+2})\right| > 0\}$. We have now that $\sum_{j\in\mathcal{H}} c_{ij}$ is the gradient of $\sum_{j\in\mathcal{G}} y_{ij}(\theta^+)$ to θ^+ on the interval between θ^{+1} and θ^{+2}.
We observe that the gap of the constraint $\sum_{j\in\mathcal{G}} y_{ij}(\theta^+) + \sum_{j\notin\mathcal{G}} y_{ij}^* \leq M_i^{\max}$, evaluated at the value $\theta^+ = \theta^{+1}$ is $M_i^{\max} - \sum_{j\in\mathcal{G}} y_{ij}(\theta^{+1}) - \sum_{j\notin\mathcal{G}} y_{ij}^*$. Therefore, $\theta^{+*} = \frac{M_i^{\max} - \sum_{j\in\mathcal{G}} y_{ij}(\theta^{+1}) - \sum_{j\notin\mathcal{G}} y_{ij}^*}{\sum_{j\in\mathcal{H}} c_{ij}}$. Now $y_{ij}^* = y_{ij}(\theta^{+*})$, $\forall j \in \mathcal{G}$ and so we know $y_{i\bullet}^*$.

It could be that θ^{+2} does not exist. In other words, there does not exist a value for θ^+, for which $\sum_{j\in\mathcal{G}} y_{ij}(\theta^+) + \sum_{j\notin\mathcal{G}} y_{ij}^* \geq M_i^{\max}$. So we cannot reach the constraint $\sum_{j=1}^{m} y_{ij} = M_i^{\max}$, which means that we cannot find a KKT solution of Type 2. Now we consider finding a solution of Type 3, so a solution where $\sum_{j=1}^{m} y_{ij} = M_i^{\min}$. Finding a solution of Type 3 is very similar to finding a solution of Type 2, so we do not specifically repeat it. As we mentioned before in Lemma 7, there cannot be a KKT solution of Type 2 and a KKT solution of Type 3. Therefore, if we find a solution of Type 2, then we stop. Otherwise, we attempt to find a solution of Type 3.

It could happen that we cannot find a solution of Type 1, Type 2 or Type 3. In this case, the problem is infeasible for a fixed i and this implies that problem (4.1) is infeasible. Therefore also problem (3.8) is infeasible, because problem (3.8) also needs to satisfy the constraints (3.6) and (3.7) in addition to the constraints that need to be satisfied by problem (4.1) and is, therefore, more restrictive.

Finally, we have to analyze what happens when $c_{ij} = 0$ for some j and what happens when $M_{ij}^{\min} = M_{ij}^{\max}$. First we analyze the differences in Phase 1 of the complete algorithm if $c_{ij} = 0$ for some j. When finding a solution of Type 1, we can have more than one KKT solution, because both $y_{ij} = M_{ij}^{\max}$ and $y_{ij} = M_{ij}^{\min}$ can be part of a KKT point. The objective value for S_i is the same, regardless of the value of y_{ij} if $c_{ij} = 0$ for some j. This also means that Lemma 7 does not apply anymore, so we have to consider all solution types to find a KKT point. We could also have more than one solution of Type 2 or Type 3.
If $M_{ij}^{\min} = M_{ij}^{\max}$, then the three solution types that were described in Subsection 4.1.2 would merge into one single solution type. The process of finding a KKT point for this solution type is similar to a process of finding a KKT point for a solution of Type 2.

We assumed that i is fixed and that S_{i-1} is known. By Lemma 2, we start at $i = 1$ and find $y^*_{1\bullet}$ and S_1 with Phase 1 of the complete algorithm that we have described in this section. We then use these results to calculate for $i = 2$, $y^*_{2\bullet}$ and S_2. We repeat this for all i. This gives us the solution for Phase 1, Y^{p1}. In Section 4.2, we use Y^{p1} as a starting point for a modified version of Rosen's projection algorithm.

4.2 Phase 2: Modified Rosen's projection algorithm

In Section 2.3, we reviewed Rosen's algorithm and we used projection matrices and singular value decomposition of matrices. Thus, it is more convenient to have an decision variable that is vector and not a matrix. For the remainder of the thesis we use $Y \in \mathbb{R}^N$ as a vector instead of the matrix $Y \in \mathbb{R}^{t\times m}$. Recall that $N = tm$. Each value in the vector $Y \in \mathbb{R}^N$ maps to a unique value in the matrix $Y \in \mathbb{R}^{t\times m}$. The other variables y_{ij}, $y_{i\bullet}$ and $y_{\bullet j}$ $\forall i, j$ do not change from their definition that was stated in Section 3.1.

In Phase 1 of the complete algorithm we have found a solution for problem (4.1). We call this solution Y^{p1}. This solution satisfies the $2mt$ inequality constraints $M^{min}_{ij} \leq y_{ij} \leq M^{max}_{ij}$ (constraint (3.4)) and the $2t$ inequality constraints $M^{min}_{i} \leq \sum_{j=1}^{m} y_{ij} \leq M^{max}_{i}$ (constraint (3.5)). However Y^{p1} might not be feasible for constraint (3.6) and (3.7), therefore $2m + 1$ constraints might not be satisfied. The point Y^{p1} already satisfies most constraints, so we use the solution Y^{p1} to derive a new solution that satisfies $M^{\min}_{j} \leq \sum_{i=1}^{t} y_{ij} \leq M^{\max}_{j}$, $\forall j$ and $\sum_{i=1}^{t}\sum_{j=1}^{m} y_{ij} \leq D$.
Note, it might happen that Y^{p1} satisfies the constraints (3.6) and (3.7). When this happens, the point Y^{p1} is also globally optimal for problem (3.8). We know this because the feasible region of problem (3.8) is a subset of the feasible region of problem (4.1). Therefore, if a point is optimal for problem (4.1) and feasible for problem (3.8), then this point is also optimal for problem (3.8).

In this section, we present a modified version of Rosen's projection algorithm (Section 2.3). The goal of our modified version of Rosen's projection algorithm is to find a KKT point that satisfies all constraints from problem (3.8) using the global optimal point of the relaxed problem (4.1), which we have found in Phase 1 of the complete algorithm. Therefore, in this section we are going to solve our main problem (3.8), using Y^{p1}. In Section 4.3, we present several alternative versions of our modified projection algorithm. The results from the version that is presented in this section and the alternatives versions are analyzed in Chapter 5.

Recall that Rosen's projection algorithm does not allow infeasible starting points. This means that we have to modify Rosen's projection algorithm to allow for these infeasible points. We also rewrite the constraints of problem (3.8) such that Y^{p1} is a feasible point.

We mentioned that it is more convenient to present our modified version of Rosen's projection algorithm with vector notation. Therefore, to keep similarity in the formulation between Section 2.3 and this section, we replace problem (3.8) with the following vectorized version of problem

(3.8):

$$\begin{aligned} &\max_Y S(Y) = \max_Y && \sum_{i=1}^{t} S_i\left(y_{1\bullet}, \ldots, y_{i\bullet}\right) \\ &\text{s.t.} && BY \geq q, \end{aligned} \tag{4.9}$$

where $B \in \mathbb{R}^{L \times N}$.

Now we need that Y^{p1} is a feasible result of $BY \geq q$. To make Y^{p1} feasible we introduce the slack parameter z. We replace the constraints of problem (4.9) by $BY - q \geq \min(0, BY^{p1} - q) = z$. This means that if the constraint $b_l Y^{p1} \geq q_l$ for some l is feasible then we do not change the constraint. If instead $b_l Y^{p1} < q_l$ for some constraint l, then we set the value of z_l such that $b_l Y^{p1} = z_l + q_l$. Therefore, z can be seen as a parameter vector of slack values. Note that $z \leq 0$. This reformulation implies that Y^{p1} is a feasible point.

Therefore, we get the following modified problem:

$$\begin{aligned} &\max_Y S(Y) = \max_Y && \sum_{i=1}^{t} S_i\left(y_{1\bullet}, \ldots, y_{i\bullet}\right), \\ &\text{s.t.} && BY \geq z + q. \end{aligned} \tag{4.10}$$

Remark 9. *Problem* (4.10) *is similar (but not equivalent) to the following problem with the slack variable* w*:*

$$\begin{aligned} &\max_{Y,w} S(Y) = \max_{Y,w} && \sum_{i=1}^{t} S_i\left(y_{1\bullet}, \ldots, y_{i\bullet}\right) + Me^{\top} w, \\ &\textit{s.t.} && BY \geq w + q, \\ & && w \leq 0, \end{aligned} \tag{4.11}$$

where M *is some big positive penalty parameter.*
The primary difference between problem (4.10) *and problem* (4.11) *is that problem* (4.11) *solves problem* (4.9)*, but with a penalty to violated constraints. Problem* (4.10) *is used to solve problem* (4.11) *in a stepwise manner by reducing* z *until* $z = 0$*.*

Next, we present how we move away from the slack constraints to the interior of the feasible region. We also present how we find a point where the KKT conditions hold and where the slack value $z = 0$.
First, we present a method where we can apply Rosen's algorithm to a direction that increases z and to a direction that improves the objective value $S(Y)$. These directions are then combined to find a single direction. We define the set of all constraints for which $z < 0$ as $\mathcal{B}(z) = \{l \,|z_l < 0\}$. Note that $\mathcal{B}(z) \subseteq \mathcal{A}(Y)$ (see Section 2.1).

Now we make the assumption that throughout this section the matrix $M = \left(B_{\mathcal{A}(Y)}^{\top}, A^{\top}\right)^{\top}$ has full row rank. The matrices $B_{\mathcal{A}(Y)}$ and A have therefore also full row rank, because $B_{\mathcal{A}(Y)}, A \in M$. The assumption of full row rank is a fair assumption to make, because the pseudoinverse can be applied in Rosen's algorithm when M does not have full row rank (Section 2.3 and Subsection 2.3.1). Furthermore, we do not have equality constraints $AY = p$ in problem (4.10). Therefore, matrix $A \in \mathbb{R}^{0 \times N}$, so A is an empty matrix. For the sake of generality of our modified version of Rosen's algorithm, we still include the matrix A in our calculations.

We now define the directions that are feasible and that are increasing the slack value z. Say we have a constraint $l \in \mathcal{B}(z)$, so a constraint with a strictly negative slack value. We remove this constraint from the active constraints $B_{\mathcal{A}(Y)}$, so we get $B_{\mathcal{A}(Y)\setminus l}$. We still do not want to violate this constraint $l \in \mathcal{B}(z)$. Therefore, we aim to find a direction v^l that is feasible with regards to this constraint and increases z_l as much as possible. An obvious choice is the direction αb_l for some strictly positive α. This direction is feasible with regards to the removed constraint, because $b_l^\top(Y + \alpha b_l) = b_l^\top Y + \alpha b_l^\top b_l = q_l + z_l + \alpha b_l^\top b_l > q_l + z_l$. However, it might not be feasible for all the other constraints. We know from Section 2.3 that we can use Rosen's algorithm to find a feasible direction $v^l = Pb_l^\top$. This direction v^l does not decrease the slack variable z_l (equation (2.8)).

We now show that for any $l \in \mathcal{B}(z)$ there exists a direction v^l that increases z_l, but that does not decrease z_l for all other l. In other words, we show that $b_l v^l > 0$.
We know that our constraint $b_l Y \geq q_l + z_l$ is perpendicular to the direction αb_l. Note that $b_l Y = q_l + z_l$, because $l \in \mathcal{B}(z)$ and therefore $l \in \mathcal{A}(Y)$. We also know that $z_l < 0$.
So for a direction v^l that increases z_l to exist, there should not be a constraint in the active set of constraints $B_{\mathcal{A}(Y)\setminus l}$ that is perpendicular to v^l. However, the only constraint that can meet this requirements is $-b_l Y \geq -q_l + z_l$. This constraint can only be active if $z_l = 0$. However, we just saw that $z_l < 0$, so this constraint is not active and therefore not in $B_{\mathcal{A}(Y)\setminus l}$. So there is always a direction v^l that increases z_l.

To summarize, in Section 2.3 we saw that Rosen's algorithm could be used to find a direction v that increases the objective. In this section, we have seen that Rosen's algorithm can be modified to find a feasible direction v^l that increases the slack variable z_l. There are several ways to pick l, which are analyzed in Subsection 4.3.2. We pick a random $l \in \mathcal{B}(z)$. The only topic that needs to be analyzed is how these directions can be combined.We denote the combined direction of v and v^l as V. There are several methods find V. In Subsection 4.3.2, we analyze some alternatives to V. In this section, we first normalize the directions v and v^l. Then we use the following equation for V:

$$V = \left(h^0 v + (1 - h^0) v^l h^l\right), \tag{4.12}$$

where the parameter $0 \leq h^0 < 1$ balances the importance of improving the current solution with increasing the slack variable z_l and h^l is some constant. We use that $h^l = 1$. This parameter h^l seems redundant for now, but we want to keep consistency between equation (4.12) and some alternatives to V, for example equation (4.14) and (4.15), that are presented in Subsection 4.3.2. The parameter h^l is analyzed further in Subsection 4.3.4. If we set $h^0 = 0$, then only improving the slack is important and any possible improvements are ignored. In equation (4.12) we see why it was needed to normalize v^l and v, because if we did not normalize it, then h^0 would not have a consistent meaning, because the length of v^l and v could change per iteration of our modified version of Rosen's algorithm algorithm.

In equation (4.12) we use the property that if there are multiple directions that are feasible, then any convex combination of directions is also feasible. The property holds because linear con-

straints form a convex set. Note that we are also always allowed to scale one direction relative to the other directions, as this does not influence the feasibility of the direction. We also note that $V \neq 0$, if $v \neq 0$ and $v^l \neq 0$. We know this because if $(1 - h_0)v^l \neq 0$, then $(1 - h_0)v^l$ is a direction that moves to the interior of the feasible region away from constraint l, so we know that $v = \frac{-(1-h_0)v^l}{h_0}$ is not a feasible direction, because it would be a move to the outside of constraint l so to the outside of the feasible region.

Now we modify Rosen's algorithm in the following way. As an initialization we set, as we did in the original formulation from Section 2.3, $r = 0$ and as initial point Y^0 we take Y^{p1}. Now our modified version of Rosen's algorithm performs the following three 3 steps. Step 1 in this algorithm is similar to Step 1 of Rosen's algorithm (Section 2.3), but the stopping criteria are different and the direction v is normalized. Step 2 is entirely new, but it is similar to Step 1. Step 3 in the modified algorithm is similar in function as Step 2 in the original algorithm. In both cases a step size α is found, but there are significant differences. We will omit a few minor details in Step 1 and Step 2 that we did mention in Section 2.3 for the original version of Rosen's algorithm.

1. In the first step, we consider three possible options for the direction v in the same way as in Section 2.3. First we set $M \leftarrow \left(B^\top_{\mathcal{A}(Y^r)}, A^\top\right)^\top$ and calculate the projection matrix P.

   (a) If M is empty, then we set $v \leftarrow \nabla S(Y^r)$. We have found a local optimum if $v = 0$ and $z = 0$, so we stop the algorithm. If $v = 0$ and $z \neq 0$, we go to Step 2. If $v \neq 0$ we set $v \leftarrow \frac{v}{\|v\|}$ and go to Step 2.

   (b) For Option (b) the gradient $\nabla S(Y^r)$ is projected with the projection matrix P. The condition for Option (b) is that $||P\nabla S(Y^r)|| > s_r|u|$ and $u \leftarrow \min_{l\in\mathcal{A}(Y)} \mu_l$. If there are no negative multipliers, then $u \leftarrow 0$.
   If the condition that $||P\nabla S(Y^r)|| > s_r|u|$ is satisfied, then we set $v \leftarrow P\nabla S(Y^r)$. If the KKT multiplier $\mu \geq 0$, $v = 0$ and $z = 0$, then we have satisfied the system of KKT conditions (4.2a) to (4.2j) and we have found a local optimum, so we stop the algorithm. If $v = 0$, but $\mu < 0$ or $z < 0$, then we go to Step 2. Otherwise, we set $v \leftarrow \frac{v}{\|v\|}$ and go to Step 2.

   (c) If the conditions in Option (a) and (b) do not hold, then we first find $\hat{l} \leftarrow \text{argmin}_{l\in\mathcal{A}(Y^r)} \mu_l$ and remove the corresponding row from the matrix of inequality constraints, so $B_{\mathcal{A}(Y^r)\setminus\hat{l}}$. Then we set $M \leftarrow \left(B^\top_{\mathcal{A}(Y^r)\setminus\hat{l}}, A^\top\right)^\top$ and we calculate a new projection matrix P. We use these new matrices to set $v \leftarrow \frac{P\nabla S(Y^r)}{\|P\nabla S(Y^r)\|}$ and we go to Step 2.

2. Step 2 is very similar to Step 1, but instead we find in this step the direction that increases the slack value for a random constraint $b_l Y^r \geq q_l + z_l$, for which $l \in \mathcal{B}(z)$. If $\mathcal{B}(z)$ is empty, then there are no negative slack variables z_l, so we set $v^l \leftarrow 0$ and we go to Step 3. Otherwise, we set $M \leftarrow \left(B^\top_{\mathcal{A}(Y^r)\setminus l}, A^\top\right)^\top$ and calculate P. Then we consider three options.

   (a) If M is empty, then we set $v^l \leftarrow \frac{b_l^\top}{\|b_l\|}$ and go to Step 3. Note that $\|b_l\| \neq 0$, because $B_{\mathcal{A}(Y^r)}$ has full row rank.

(b) Option (b) is very similar to Option (b) in Step 1. Only the stopping criterion is not included. For Option (b) we have that b_l is projected with the projection matrix P. The condition for Option (b) is that $||Pb_l^\top|| > s_r|u|$ and $u \leftarrow \min_{l \in \mathcal{A}(Y)} \mu_l$. If there are no negative multipliers, then $u \leftarrow 0$.
If the condition that $||Pb_l^\top|| > s_r|u|$ is satisfied, then we set $v^l \leftarrow \frac{Pb_l^\top}{\left\|Pb_l^\top\right\|}$ and we go to Step 3.

(c) Option (c) is very similar to Option (c) in Step 1. If the conditions in Option (a) and (b) do not hold, then we first find $\hat{l} = \text{argmin}_{l \in \mathcal{A}(Y^r) \setminus l} \mu_l$ and remove the corresponding row from the matrix of inequality constraints, so $B_{\mathcal{A}(Y^r)\setminus\{l,\hat{l}\}}$. Recall that row l was already removed from the set of inequality constraints at the beginning of Step 2. Then we set $M \leftarrow \left(B^\top_{\mathcal{A}(Y^r)\setminus\{l,\hat{l}\}}, A^\top\right)^\top$ and we calculate a new projection matrix P. We use these new matrices to set $v^l \leftarrow \frac{Pb_l^\top}{\left\|Pb_l^\top\right\|}$ and we go to Step 3.

3. In Step 3 we take a step in the direction $Y^{r+1} \leftarrow Y^r + \alpha \frac{V}{||V||}$ for some $0 < \alpha \leq \alpha_{\max}$. Where V is some direction that depends on both v and v^l, and $\alpha_{\max}$ is the maximum value for α, such that Y^{r+1} is still a feasible point. In this version of modified algorithm we take V as in equation (4.12). In Section 4.3, we analyze some alternatives to V.
For the choice of α we distinguish two options. Option (a) is used when it is not possible to use one of the line search methods that were reviewed in Section 2.4. Option (b) is used when it is possible to use one of these methods.

   (a) We are not able to use line search to find α if $v^l \neq 0$, because V might decrease the objective, such that $S\left(Y^r + \alpha \frac{V}{||V||}\right) < S(Y^r),\ \forall \alpha > 0$. Therefore, we just have to pick some value for α. We analyze in Subsection 4.3.4 what values of α we can use.

   (b) If $v^l = 0$, then we know that V is an ascending direction. This means that we can use line search, as was discussed in the original algorithm. We use backtracking line search here to find α, which was mentioned in Subsection 2.4.2.

   Then we set $r \leftarrow r + 1$ and we tighten the constraints that have been improved, so we set $z \leftarrow \min(0, BY^{r+1} - q)$ and go to Step 1.

In Subsection 4.3.3, we show how to calculate $\nabla S(Y)$.

This algorithm is the basic version of the modified version of Rosen's algorithm. In Section 4.3, we present several alternatives. Some alternatives focus on the initialization, some on the direction V and some change the type of line search that is used. These different types of line search were reviewed in Section 2.4. When comparing all the different algorithms in Chapter 5, we use 'base' to refer to the complete algorithm that uses the basic version of the modified version of Rosen's algorithm. Furthermore, if we replace in Step 3 of the algorithm the backtracking line search with golden section search that was reviewed in Subsection 2.4.1, then we denote this algorithm with 'gss'. If we replace backtracking line search with the quadratic Taylor approximation algorithm of Subsection 2.4.3, then we denote this algorithm with 'quadApprox'.

4.3 Alternatives to the algorithm

The modified version of Rosen's algorithm that was presented in Section 4.1 and Section 4.2 is just one possible way to solve problem (3.8). There are many variants of this modified algorithm possible. In this section, we present several of these variants.
First, we analyze how a feasible starting point could be found. This allows us to skip Phase 1 entirely and it also allows us to use the original version of Rosen's algorithm in Phase 2 of the complete algorithm instead of first having to find a feasible point before we could do that.
We also present some alternatives for calculating V, the combined direction that balances both the direction that improves the objective with the direction that increases the value of the slack variables z.

4.3.1 Initializing Rosen's algorithm

A way of finding a feasible point Y is solving the problem (4.9) by replacing the objective $S(Y)$ of that problem by a linear function. This allows us to use a linear program (LP) solver. There are many different LP solvers, for example, CPLEX (IBM, 2015) and lp_solve (Berkelaar, Eikland, & Notebaert, 2004) and these solvers can quickly give a feasible solution, especially when the number of constraints and variables does not exceed the tens of thousands. For example, a problem that is similar to problem (4.9) takes less than a second to solve on current computers. To initialize Rosen's algorithm, we solve the following problem:

$$\begin{aligned} \max_Y \quad & E^\top Y \\ \text{s.t.} \quad & BY \geq z, \end{aligned} \tag{4.13}$$

where $E^\top Y$ can be any approximation of the objective function $S(Y)$ and $E \in \mathbb{R}^N$.

We now discuss some possible choices for E. A straightforward choice for E is just setting $E = 0$. This might lead to a feasible starting point that is quite far away from the local optimum that we would want to reach. Therefore it is better to use some linearization of $S(Y)$. We present two approaches. The first approach involves linearizing equation (3.2). The second approach uses a first order Taylor approximation around a few different points.

We now present the first approach. We use equation (3.2) and replace $\log(y_{ij})$ with y_{ij} for all i, j. We ignore the lagged effect S_{i-1}. Using this as an objective function means that all y_{ij} with negative coefficients c_{ij} should be equal to their lower bound $M_{ij}^{\min}$, unless other lower bounds, such as $\sum_{j=1}^{m} y_{ij} \leq M_i^{\min}$ and $\sum_{i=1}^{t} y_{ij} \leq M_j^{\min}$ force some of them to be higher. Similarly, y_{ij} with positive coefficients should be equal to $M_{ij}^{\max}$, if that is possible.
The second approach is to use $E = \nabla S(Y)$. We need to evaluate $\nabla S(Y)$ at some point Y before we can use this. We use $y_{ij} = M_{ij}^{\min},\ \forall i, j$. This means that we are essentially using a first-order Taylor approximation of S around the point $y_{ij} = M_{ij}^{\min}, \forall i, j$. Two variants of setting $y_{ij} = M_{ij}^{\min}$, $\forall i, j$ are setting $y_{ij} = M_{ij}^{\max},\ \forall i, j$ or using any value for y_{ij} between $M_{ij}^{\min}$ and $M_{ij}^{\max}$, for example $y_{ij} = \frac{1}{2}(M_{ij}^{\min} + M_{ij}^{\max}),\ \forall i, j$. It is not required that any of these points are feasible for all constraints, although this would improve the accuracy of the Taylor approximation.

Up to now, we have ignored the lagged effect S_{i-1} in the calculation of the linear approximation E. We did not deal with the fact that equation (3.3) is a function of products either. Since equation (3.3) is a function of products, it is better to increase two variables y_{ij} with positive c_{ij} in the same week i (according to some fixed ratio) than to increase only one of them. We can also see this effect by looking at the Hessian in Subsection 4.3.3. Here we can see that the second derivative is positive if for two variables y_{ij} and y_{fg}, we have $c_{ij}, c_{fg} > 0,\ \forall i, j, f, g$. We also see that the second derivative decreases rapidly, depending on the difference between the index i and f and the value of $c_{i\text{lag}}$.
We can mimic the lagged effect S_{i-1} and the effect of increasing variables within a week by modifying E. If we first solve an LP for some E_1 and find that a certain week has a high number of sales, then we multiply the elements of E_1 that correspond to that week and the weeks before it by a certain factor. We call this modification E_2. Now we solve the same LP as before, but with E_2 instead of E_1 and find, possibly, a better approximation of $S(Y)$.

As we mentioned before, solving LP instances does not cost much computationally. Rosen's algorithm is however quite computationally expensive compared to solving LP's. Therefore, we can calculate the optimal point for all the different problems that we have mentioned in this section. Then use point with the highest objective as an initialization point of the original version of Rosen's algorithm. In Chapter 5, we use 'rosen' to refer to the version of the complete algorithm where we first find a good initial feasible point with all of the above methods and then use the original version of Rosen's algorithm. In the next section, we present some options to modify the ascent direction V.

4.3.2 Modifying the ascent direction

In our modified version of Rosen's algorithm in Section 4.2, we defined the direction V in equation (4.12). Recall that V is the direction that combines a direction v that improves the objective and a direction v^l that improves the value of a random slack variable z_l.
However, there are often several negative slack variables z_l. Therefore, it could be considered to repeat Step 2 of the modified projection algorithm in Section 4.2 for more than one direction. Recall that $\mathcal{B}(z)$ is the set of all constraints that have negative slack variables z, so $\sum_{l\in\mathcal{B}} v^l h^l$ is a linear combination of all directions that are feasible and that improve the slack variables. The direction $\sum_{l\in\mathcal{B}} v^l h^l$ is also feasible, because problem (4.9) has convex constraints. This gives us the following formula:

$$V = \left(h^0 v + (1 - h^0) \frac{\sum_{l\in\mathcal{B}} v^l h^l}{\left\| \sum_{l\in\mathcal{B}} v^l h^l \right\|} \right). \tag{4.14}$$

We normalize the direction $\sum_{l\in\mathcal{B}} v^l h^l$ to keep the interpretation of h^0 the same as in Section 4.2, which is the parameter that balances the best improving direction with some other direction that improves the slack variables. Now we also see that it is relevant to choose specific values for h^l, because this means that one direction v^l can be made more important than other directions. The question remains on how to exactly choose h^l. We discuss this in Subsection 4.3.4. Note that for

each direction v^l that is calculated a new projection matrix P is needed, so also a new singular value decomposition needs to be made. This is computationally expensive to do.

So we have now considered two options on which direction V to choose. The first option is the random direction that we used in Step 3 of our modified version of Rosen's algorithm (equation (4.12)). Here a direction is chosen that moves to the interior of the feasible region for some random constraint with a negative slack value. The second option is to consider a direction V that uses all constraints with negative slack values (equation (4.14)). We call this option 'allConstr' in Chapter 5.
A third option is to choose the direction that belongs to the constraint for the most negative z, so the constraint that is violated the most. We call this option 'maxConstr'. We can also combine Option 2 and 3 by taking any number of directions that increase the slack variable of the constraints. To keep a limited number of variants for the complete algorithm, we consider as a fourth option the option where we calculate the three directions v^l for which z_l is the most negative. We call this option '3maxConstr'. We define $\mathcal{N}(z)$ as the set of the index l of the three most negative z_l. So we use the following equation for V:

$$V = \left(h^0 v + (1 - h^0) \frac{\sum_{l \in \mathcal{N}(z)} v^l h^l}{\left\| \sum_{l \in \mathcal{N}(z)} v^l h^l \right\|} \right). \tag{4.15}$$

If there are fewer than three directions v^l with negative z_l, then we take all remaining directions v^l with negative z_l. In Chapter 5, we test each of these options for choosing V for a few datasets and compare the results between these options.

4.3.3 Gradient and Hessian

In this section, we show the calculation of the gradient $\nabla S(Y)$ of the objective function (3.1). This gradient is crucial, because it is used in the calculation of the feasible direction that improves the objective v in the original version of Rosen's algorithm in Section 2.3 and in modified version in Section 4.2. We also show how to calculate the Hessian in this section.

For the calculation of the gradient we use Lemma 10. We assume that if we have an empty product, so a product where no factors are multiplied, then this product is equal to 1. For example $\prod_{l=i}^{i-1} c_{l\text{lag}} = 1$.

Lemma 10. *Let S_k be the sales in week k as defined in (3.3) and $S = \sum_{k=1}^{t} S_k$. Then $\frac{\partial S}{\partial y_{ij}} = \frac{c_{ij}}{y_{ij}} \sum_{k \geq i}^{t} S_k \prod_{l=i}^{k-1} c_{llag}$, where $i = 1, \ldots, t$, $j = 1, \ldots, m$, $k = 1, \ldots, t$, $l = 1, \ldots, t$.*

Proof. We first define $u_{ijk} = \frac{\frac{\partial S_k}{\partial y_{ij}}}{S_k}$, $\forall i, j, k$. We know from equation (3.3) that $\forall i < k$, $\forall j$, S_k depends on y_{ij} through S_{k-1}. Therefore, we have $u_{ijk} = c_{i\text{lag}} \frac{\frac{\partial S_{k-1}}{\partial y_{ij}}}{S_{k-1}} = c_{i\text{lag}} u_{ij(k-1)}$, $\forall i < k$, $\forall j$. If $i = k$ then, as we have seen in equation (4.3), $u_{ijk} = \frac{c_{ij}}{y_{ij}}$, $\forall i = k$, $\forall j$.
We also know from Lemma 2, that $\frac{\partial S_k}{\partial y_{ij}} = 0$, $\forall i > k$, $\forall j$ and so $u_{ijk} = 0$, $\forall i > k$, $\forall j$. In particular,

$u_{ij0} = 0,\ \forall i, j.$
After combining the previous findings we have that $u_{ijk} = c_{i\text{lag}} u_{ij(k-1)} + \mathbb{1}_{i=k} \frac{c_{ij}}{y_{ij}},\ \forall i, j, k.$
Finally we have $\frac{\partial S}{\partial y_{ij}} = \sum_{k=1}^{t} \frac{\partial S_k}{\partial y_{ij}} = \sum_{k=1}^{t} u_{ijk} S_k = \sum_{k \geq i}^{t} \frac{c_{ij}}{y_{ij}} S_k \prod_{l=i}^{k-1} c_{l\text{lag}} = \frac{c_{ij}}{y_{ij}} \sum_{k \geq i}^{t} S_k \prod_{l=i}^{k-1} c_{l\text{lag}}.$ □

In Chapter 5, we use Lemma 10 to calculate $\nabla S(y)$ for a few different datasets, with different objective values. Lemma 10 can also be used to calculate the second derivative. We observe that $\frac{\partial^2 S}{\partial y_{ij} \partial y_{fg}} = \frac{c_{ij}}{y_{ij}} \sum_{k \geq i} \frac{\partial S_k}{\partial y_{fg}} \prod_{l=i}^{k-1} c_{l\text{lag}} = \frac{c_{ij}}{y_{ij}} \frac{c_{fg}}{y_{fg}} \sum_{k \geq \max(i,f)}^{t} S_k \prod_{l=i}^{k-1} c_{l\text{lag}} \prod_{l=f}^{k-1} c_{l\text{lag}}$, for $(f, g) \neq (i, j)$ and $f = 1, \ldots, t,\ g = 1, \ldots, m,\ i = 1, \ldots, t,\ j = 1, \ldots, m,\ k = 1, \ldots, t,\ l = 1, \ldots, t.$ Note that if $c_{ij} > 0$ and $c_{fg} > 0$, then this implies that $\frac{\partial^2 S}{\partial y_{ij} \partial y_{fg}} > 0\ \ \forall i, j, f, g,$ because it is assumed that $S_k > 0$ and $c_{i\text{lag}} > 0,\ \forall i, k.$ In other words, an increasing a variable with a positive coefficient increases, ceteris paribus, the gradient of all other variables with positive coefficients.
Furthermore, we have $\frac{\partial^2 S}{\partial y_{ij}^2} = c_{ij} \sum_{k \geq i}^{t} \frac{\partial \tilde{S_k}}{\partial y_{ij}} \prod_{l=i}^{k-1} c_{l\text{lag}} = \frac{c_{ij}(c_{ij}-1)}{y_{ij}^2} \sum_{k \geq i}^{t} S_k \left(\prod_{l=i}^{k-1} c_{l\text{lag}} \right)^2,\ \forall i, j, k, l,$ where $\tilde{S_k} = \frac{S_k}{y_{ij}}$. Note that $\tilde{S_k}$ only differers from S_k in coefficient c_{ij}. This coefficient is one unit smaller in $\tilde{S_k}$ than in S_k. The proof for these equations follows a very similar path as Lemma 10, because both $\sum_{k \geq i}^{t} S_k$ and $\sum_{k \geq i}^{t} \tilde{S_k}$ are very similar to $\sum_{k=1}^{t} S_k$.

We can see that the second derivative is quite complex to calculate. This also means that it is computationally expensive to calculate the Hessian of $S(Y)$. Therefore, line search algorithms that make use of the Hessian, for example, the quadratic Taylor approximation that was reviewed in Subsection 2.4.3, might use a lot of computational resources. In Chapter 5, we compare this quadratic Taylor approximation with the other line search methods that were reviewed in Section 2.4.

4.3.4 Parameter discussion

In this section, we further discuss all parameters that we have introduced in the previous sections and what their effects are on the optimization process. First, we consider the parameters that are in the original projection algorithm with linear constraints by Rosen (1960), which we reviewed in Section 2.3. We also consider all new parameters that we presented in Section 4.2 in the modified version of Rosen's algorithm. Then the parameters of all variants of the complete algorithm are considered and finally, we consider the parameters of the various line search algorithms. In Section 5.3, we do some sensitivity analysis for some of the parameters.

The original version of Rosen's algorithm with linear constraints that was presented in Section 4.2 only has a single parameter, s_r that can change between the iterations r. We briefly discussed this parameter before in Section 2.3. It is used in the inequality $||P\nabla S(Y^r)|| > s_r|u|$, where $u = \min_{l \in \mathcal{B}(z)} \mu_l$ is the smallest (and negative) multiplier of the inequality constraints of the KKT conditions. The vector $v = P\nabla S(Y^r)$ is the ascent direction in the case that there are no deleted constraints. This criterion $||P\nabla S(Y^r)|| > s_r|u|$ determines if $P\nabla S(Y^r)$ is a ascent direction that decreases the objective sufficiently. If u is very negative and the corresponding constraint $\hat{l} = \text{argmin}_{l \in \mathcal{B}(z)} \mu_l$ is dropped from M, then it will greatly improve the objective function. However, this comes at an additional computational cost of calculating another projection matrix

and another direction. So if the ascent direction v was already good enough, then the benefit of calculating another direction is small.
Therefore, the parameter s_r is a parameter that balances between incurring additional computational costs and getting a better ascent direction with less computational costs and a worse ascent direction.
Rosen (1960) suggested that s_r should not be higher than the square root of the smallest eigenvalue of $MM^\top$, which changes every iteration r. However, in our implementation this was significantly (roughly 50%) slower than setting $s_r = 1.25$ for all r. Recall that He and Zhang (1986) showed that there is convergence to a KKT point for any $s_r > 0$. So when comparing the different algorithms in Chapter 5, we set $s_r = 1.25$ for all r. In Section 5.3, we make some final sensitivity analysis for the choice of s_r.

In our modified version of Rosen's algorithm in Section 4.2, we used the parameters h^0 and h^l. Where h^0 is the parameter that balances the direction that improves the objective and direction that increases the value of z, the slack variable. We analyzed some aspects about h^0 and noted that $0 < h^0 \leq 1$. We set $h^0 = 0.5$ for the time being, as this implies that we give equal weight to improving the objective function $S(Y)$ and to improving the slack value z. In Section 5.3, we perform some sensitivity analysis on h^0.

The parameter h^l balances the importance of the directions v^l for $l \in \mathcal{B}(z)$. We saw that this parameter is important in equation (4.14) and equation (4.15). In Section 4.3, we presented several algorithms in which we changed the definition of V. The algorithm 'allConstr' uses equation (4.14) to determine V, so it uses all directions v^l for which $z_l < 0$. Therefore, it becomes relevant how h^l is chosen. We consider two options for the choice of h^l. For the algorithm 'allConstr', we use $h^l = 1$, $\forall l$. For the algorithm 'WallConstr' we weight h^l by taking $h^l = z_l$, $\forall l \in \mathcal{B}(z)$. Therefore, in the algorithm 'WallConstr', we put more weight on constraints that are violated the most for any point Y^r. For the algorithm '3maxConstr' that uses equation (4.15) we use the same approach. For this algorithm we use $h^l = 1$, $\forall l \in \mathcal{N}(z)$ and for 'W3maxConstr' we use $h^l = z_l$, $\forall l \in \mathcal{N}(z)$.

The final parameter for the modified version of Rosen's algorithm is the parameter α when $v^l \neq 0$ for some $l \in \mathcal{B}(z)$. In this case we noted that cannot use line search. We use $\alpha = 0.5\alpha_{\max}$. In Section 5.3, we compare some other choices for α.

In the golden section search algorithm (Subsection 2.4.1) we need to set some tolerance value τ. The value of τ determines when the interval $[\alpha_a, \alpha_b]$ is short enough to terminate the golden section search algorithm. If τ is too big, then the line search is not precise enough and occasionally it might result in finding a step size α for which $S(Y + \alpha v) < S(Y)$. This is problematic as we discussed in Subsection 2.4.1. However, too small values of τ incur a higher computational cost because more iterations of the golden section search algorithm are needed. We use $\tau = 10^{-5}$. This is quite computationally expensive, but also quite precise. In Section 5.3, we consider higher values for $\tau = 10^{-5}$.

For backtracking line search (Subsection 2.4.2) we have two parameters C and ρ. The parameter $C \in (0, 1)$ is used in calculating the minimum improvement that a step size should make. For this parameter the value $C = 10^{-4}$ is suggested (Nocedal & Wright, 1999). We use this value.
The parameter $\rho \in (0, 1)$ denotes the factor by which α is decreased during every iteration. Since the golden section search algorithm is a bit slow, but a relatively precise algorithm, we want to make the golden section search algorithm quite fast. Therefore, we do not mind if the algorithm is quite imprecise, so we initially set $\rho = 0.1$. In Section 5.3, C and ρ we perform some sensitivity analysis on C and ρ.

5. NUMERICAL RESULTS

In this chapter, we present the results of the algorithms that are mentioned in Chapter 4. In Section 5.1, we first summarize all versions of the complete algorithm that we have presented in the previous chapters. Then we present four different datasets and analyze the performance of all versions of the complete algorithm on these datasets. We also analyze the relative performance of the algorithms. For example, we analyze when the algorithm 'rosen' has advantages over the other algorithms and when it is the other way around. In Section 5.2, we analyze some trends and other observations that we noticed in the results of the datasets.
In Section 5.3, we do a sensitivity analysis on several parameters that were considered in the previous chapters of this thesis. In Figure 5.1, we summarize these results.

To calculate the results for all the algorithms and the sensitivity analysis, we use an Intel i5-3470S quad core processor running at 2.9 GHz and 8 GiB of memory. We only use a single core of this processor, so we do not use any parallelization of the code. Also, functions and packages that we used in the code (and that took a significant amount of time to calculate) did not use any parallelization. In Section 5.1, we state the advantages of using parallelization in the algorithm. All algorithms are implemented in R (v3.2.3) and we use the operating system Microsoft Windows 7.
The singular value decomposition, which we reviewed in Section 2.2 and which is needed for calculating projection matrices, is calculated using LAPACK (Anderson et al., 1999). To use LAPACK, we used the wrapper function La.svd(). This function is one of the functions that are included in the basic distribution of R. Furthermore, we use in R the software package lpSolve. This software package is a wrapper function for lp_solve (Berkelaar et al., 2004). We use lp_solve to solve linear programs, which is needed to find initial points for the algorithm 'rosen' (Subsection 4.3.1). We also use lp_solve to find constraints that are redundant (Subsection 2.3.1).

A slight disadvantage of programming is that there is some inherent randomness in the computation time of the versions of the complete algorithm. This randomness can be caused by various things, for example by an overheating processor that starts running at a lower frequency to reduce heat, which increases the computation time. Another example is that an operating system is also performing some tasks in the background, which slows down the optimization. We also had to measure real time, because it is not practically feasible to calculate the CPU time in R for a big algorithm, which also adds some additional randomness.
Unfortunately, the randomness is very unpredictable and tended to differ more when there was more time between the execution of two algorithms. We also noticed that when we took averages the variance in the computation time did not decrease much.

Therefore we executed all versions of the complete algorithm for a single dataset in quick succession. When interpreting the results, an uncertainty margin of approximately 5% should still be taken into account for all computation times.

5.1 Results datasets

In this section, we first analyze and compare numerical results on several datasets. Then we use the numerical results to analyze the relative performance of the versions of the complete algorithm that we have presented in the previous chapters.

Tab. 5.1: Description of versions of the complete algorithm

Name	Description	Section
base	Modified version of Rosen's algorithm	4.2 & 2.4.2
quadApprox	base & quadratic approximation for line search	4.2 & 2.4.3
gss	base & golden section search for line search	4.2 & 2.4.1
allConstr	base & direction V increases all negative z_l	4.2 & 4.3.2
WallConstr	allConstr & parameter h^l is weighted to z_l	4.2 & 4.3.2 & 4.3.4
maxConstr	base & direction V increases most negative z_l	4.2 & 4.3.2
3maxConstr	base & direction V increases three most negative z_l	4.2 & 4.3.2
W3maxConstr	3maxConstr & parameter h^l is weighted to z_l	4.2 & 4.3.2 & 4.3.4
rosen	Original version of Rosen's algorithm & LP starting point	2.3 & 4.3.1

Before we present the results, we first briefly summarize all the algorithms that we have presented in Chapter 4. This summary can be found in Table 5.1. In this table, we briefly repeat the names of the different algorithms, to which algorithms these names refer to and in which sections these algorithms can be found. Recall that 'base' is the algorithm that was described in Section 4.2 and that this algorithm uses backtracking line search.
In 'base' equation (4.12) is used to calculate the combined direction V. Therefore, V depends on an the direction v that improves the objective $S(Y)$ and a direction v^l that increases the slack variable z_l for some random $l \in \mathcal{B}(z)$. Due to this randomness, we have to repeat algorithm 'base', 'quadApprox' and 'gss' several times to get a more accurate estimate of the results. We execute the algorithm 'base' and 'gss' a total of 10 times for every dataset and report the averages. The algorithm 'quadApprox' is executed five times.

We do not report the variances in the run time of an algorithm because the variances are so small that they could not be accurately distinguished from the inherent randomness in the run time of an algorithm that we mentioned at the beginning of this chapter. We tested the variances by comparing the variances of the algorithms for which the direction V has some random elements with the variances of the algorithms for which the direction V does not have any randomness.
We do not report the variances of the objective either, because the difference of the objective values between the repeated algorithms is typically less than 0.01% of the value of the objective.
All other algorithms do not have any randomness, because the direction V is defined by either equation (4.14) or equation (4.15). Both these equations do not involve any randomness. These

other algorithms all use backtracking line search from Subsection 2.4.2.

The names of the algorithms in Table 5.1 also occur in the same order in Table A.2 to Table A.9. In these tables, we denote the results for each of the algorithms and we use a similar structure as in Table 5.1. We first denote the name of the algorithm and then we denote the results. The explanation of all the header names that are used in Table A.2 to Table A.9 are in Table A.1.

Now we present the datasets. We will present four datasets in this section. In Table 5.2 and Table 5.3 we tabulated a short summary of the differences between the datasets. In Table 5.2 the header #F denotes the number of free variables, so the number of variables y_{ij} for which $M_{ij}^{\max} > M_{ij}^{\min}$ for any i, j. The header $\#z_l$ denotes the number of violated constraints after Phase 1 of the complete algorithm, which we considered in Subsection 4.1.3. The header $\#z_l$ is therefore not applicable to the algorithm 'rosen'. The other headers in Table 5.2 and Table 5.3 should be self-explanatory.
The value of #F is a very important characteristic of the dataset because it can greatly influence the total computation time. We note that if the value of #F is smaller than the total number of variables $m \cdot t$, then we know that for some i and j, we have $M_{ij}^{\max} = M_{ij}^{\min} = y_{ij}$. Therefore, we know that any improving and feasible direction will not change that variable. This means that there is no point in including this variable in the calculation of the projection matrix P. Therefore, we can also exclude this variable from the matrix B. Note that we might have to modify the value of q for some constraints if we do this. Therefore, the smaller #F is, the smaller the matrix B and this allows us to calculate the singular value decomposition of B in a shorter amount of time.

Tab. 5.2: Dataset characteristics

Data	m	t	#F	$\#z_l$	Range c_{ij}	Range d_i	Range $c_{i\text{lag}}$	Range $M_{ij}^{\min}$	Range $M_{ij}^{\max}$
1	6	52	109	3	$[-.22, .15]$	$[6.5, 7.5]$	.27	$[1, 10^4]$	$[1, 3 \cdot 10^5]$
2	6	52	109	3	$[-.25, .25]$	$[6.5, 7.5]$	.27	$[1, 10^4]$	$[1, 3 \cdot 10^5]$
3	8	33	264	6	$[-.1, .15]$	$[5.3, 8.3]$	$[.2, .3]$	$[1, 10^5]$	$[2 \cdot 10^3, 6 \cdot 10^5]$
4	8	33	264	6	$[-.1, .15]$	$[5.3, 8.3]$	$[.2, .3]$	$[1, 10^5]$	$[2 \cdot 10^3, 6 \cdot 10^5]$

Tab. 5.3: Dataset characteristics, continuation

Data	Range $M_i^{\min}$	Range $M_i^{\max}$	Range $M_j^{\min}$	Range $M_j^{\max}$	D	S_0
1	$[5 \cdot 10^3, 3 \cdot 10^5]$	$[10^4, 5 \cdot 10^5]$	$[1, 10^3]$	$[2 \cdot 10^5, 2.5 \cdot 10^6]$	$4.8 \cdot 10^6$	$2.5 \cdot 10^4$
2	$[5 \cdot 10^3, 3 \cdot 10^5]$	$[3 \cdot 10^4, 10^6]$	$[1, 10^3]$	$[2 \cdot 10^5, 2.5 \cdot 10^6]$	$4.8 \cdot 10^6$	$2.5 \cdot 10^4$
3	$[5 \cdot 10^3, 2 \cdot 10^5]$	$[7 \cdot 10^4, 2 \cdot 10^5]$	$[7 \cdot 10^4, 2 \cdot 10^6]$	$[2 \cdot 10^5, 2.1 \cdot 10^6]$	$7.4 \cdot 10^6$	$2.5 \cdot 10^4$
4	$[5 \cdot 10^3, 2 \cdot 10^5]$	$[7 \cdot 10^4, 2 \cdot 10^5]$	$[2 \cdot 10^4, 1 \cdot 10^6]$	$[3 \cdot 10^5, 2.4 \cdot 10^6]$	$7.8 \cdot 10^6$	$2.5 \cdot 10^4$

In Chapter 1 we mentioned the GfK business case. Dataset 1 is based on this business case. This also implies that we cannot share the raw input data regarding the constraints of this dataset. Therefore, we only provide the characteristics of this dataset in Table 5.2 and Table 5.3.

Recall that when we use the results from Phase 1 of the complete algorithm in Phase 2 of the complete algorithm, then there are at most $m + 1$ constraints that are not satisfied. This means that if m is relatively small compared to t, then our modified version of Rosen's algorithm should have a starting point that is infeasible, but with relatively few violated constraints. This would also mean that it should take less time than the original version of Rosen's algorithm to find a local maximum. Similarly, when most constraints have a small feasibility gap, so if z_l is close to 0, $\forall l$, then the modified algorithm should also be faster than the original version of Rosen's algorithm.

For Dataset 1, we have $m = 6$, so this implies that there are at most seven violated constraints after Phase 1 of the complete algorithm. We can see in Table 5.2 that $\#z_l$, the number of constraints that are violated after Phase 1, is equal to 3. This means that the algorithms 'allConstr' and '3maxConstr' should give identical results, because 'allConstr' uses all directions with negative z_l, while 3maxConstr' uses the three directions with the most negative z_l. Therefore, algorithm 'allConstr' is equivalent to '3maxConstr'. The same thing holds for the algorithms 'WallConstr' and 'W3maxConstr'. Note that it might occur that due to numerical inaccuracies there are briefly more than three violated constraints. When there are more than three violated constraints, then it might happen that these algorithms give different results. (We actually see that the results are different in Table A.2, but they are identical in Table A.4).

The other datasets have one or more constraints or other parameters that are randomized. Dataset 2 is a dataset that is obtained from Dataset 1. Only two parameters are different. The parameter c_{ij} is a random value between -0.25 and 0.25 for all i and j. Furthermore, $M_j^{\max}$, $\forall j$ has been doubled from the value that $M_j^{\max}$ had in Dataset 1. Later in this section, we will see that this Dataset is relatively easy to solve with the original version of Rosen's algorithm, while our modified algorithm gives significantly worse results.
Dataset 3 is a dataset that we created ourselves. All constraints and parameters are randomly generated within the ranges that are reported in Table 5.2 and Table 5.3. We can also see in these tables that the number of free variables #F is about 2.5 times bigger than for Dataset 1 and Dataset 2 and that this dataset has a bigger value for m, so there are at most nine constraints that might be violated. We can see that $\#z_l = 6$, so there are six violated constraints after Phase 1.
Recall that for the algorithms 'allConstr' and 'WallConstr', the number of directions v and v^l that need to be calculated during an iteration equals $\#z_l + 1 = 7$, so these algorithms incur a high computational cost per iteration.
The final dataset that we use is Dataset 4. This dataset is nearly identical to Dataset 3. The only difference is that we created Dataset 4 such that all constraints that are violated after Phase 1 are at most violated by 10^5 units. Therefore, we had to slightly increase the value of the total budget D and the parameter $M_j^{\max}$. We also had to decrease $M_j^{\min}$. This dataset is created as an example of a dataset where the number of constraints that are violated after Phase 1 is high, but that is still relatively easy to solve for our modified version of Rosen's algorithm. Finally we note that Dataset 3 and 4 are available upon request [1].

[1] To request Dataset 3 and 4, please send a mail to erikvanderklooster@gmail.com.

Now we present the results for each of the datasets. All the results of Dataset 1 to Dataset 4 can be found in Appendix Section A.1 to Appendix Section A.4. For every dataset we have two tables. In the first table in each section, for example Table A.2, we report the results regarding the objective value and some other values. In the second table, for example, Table A.3, we report all results regarding the time it takes to run parts of the complete algorithm.

Any version of the complete algorithm is stopped once one of the stopping criteria is found. For the algorithm 'rosen', these stopping criteria are mentioned in the algorithm in Section 2.3. The stopping criteria of the other algorithms, that use our modified version of Rosen's projection algorithm, are mentioned in the algorithm in Subsection 4.1.2. We also stop any version of the complete algorithm if it takes more than 5 minutes before any stopping condition is met.

For Dataset 1 we have found in Table A.2 and Table A.3 that all algorithms from Table 5.1 terminate at the objective value $3.065 \cdot 10^5$. We can also see that all algorithms give a feasible result. Note we can see under 'AbsFeasGap' that for most algorithms for some constraint l we have that $b_l Y - q_l < 0$, which would mean that it is technically infeasible. However, for all algorithms we have for all l that $b_l Y - q_l > -9 \cdot 10^{-3}$, which means that all constraints are numerically and practically feasible. Also, note that relative gap 'RelFeasGap' is even closer to zero.
We do see major differences in the computation times. The computation time of the algorithm 'quadApprox' is by far the highest of all the algorithms. We see something similar for the other datasets. At the end of this section, we analyze why 'quadApprox' is performing so poorly compared to the other algorithms. Also, the algorithm 'maxConstr' is doing significantly worse than the other algorithms. However, it seems to be just random chance that this algorithm is performing worse, because in the results of the other datasets we do not see this difference. The algorithm 'rosen' is by far the fastest algorithm, both in the total number of iterations that are needed to find a local optimum and the computation time that was used. The other algorithms all have very similar computation times. We do see that the first feasible objective value of the algorithms that use the modified projection algorithm is far higher than the feasible point that 'rosen' uses, but we also see that it took far longer to find this feasible objective value than it took for the algorithm 'rosen' to finish its optimization.

The results of Dataset 2, that can be found in Table A.4 and A.5, are similar to the results of Dataset 1. Again we have that all results are numerically feasible and that all the versions of the complete algorithm terminate within the allowed time of 5 minutes. However, we also see in Table A.4 that the algorithm 'rosen' terminates at an objective value that is 0.15% higher than the objective value of the other algorithms. We also see that the initial value of the objective function that is found for the algorithm 'rosen' is very close to the final objective value of 'rosen'. This initial objective value is only about 1% smaller, while in the other datasets that we used, the initial point is at least 9% smaller than the final objective value. This means that the algorithm 'rosen' only has to improve a little bit from the starting point, meaning that also the computation times are very low, which we can see in Table A.5. So clearly, the algorithm 'rosen' is the algorithm that performs by far the best on this dataset.

Only the algorithm 'rosen' makes use of this initial point. All the other algorithms first have to find a feasible point with the approach that was presented in Subsection 4.1.3. These other algorithms perform relatively similar, except again for 'quadApprox', which is by far the slowest. As we briefly mentioned before, the results for the algorithm 'allConstr' and '3maxConstr' are the same, as are the results of the algorithm 'WallConstr' and 'W3maxConstr'. The computation times are different. We discussed this difference at the beginning of this chapter.

In the results of Dataset 3 in Table A.6 and A.7 we can see the effects of the higher number of free variables #F and the higher number of violated constraints $\#z_l$. Recall that the value of #F greatly influences the time it takes to find directions v and v^l. We also know that $\#z_l$ implies how many directions v^l that improve the violated constraints can be calculated. In addition to this, z_l is quite negative for several constraints l, so the initial solution that is found after Phase 1 is not even close to being feasible for some constraints.
The most noticeable results that can be seen in Table A.6 and A.7 are that the algorithms 'allConstr' and 'WallConstr' do not find a feasible result within the time limit. We can see that 'Slack', the time it takes to calculate directions v^l, takes about 4 minutes of the allowed 5 minutes for these algorithms. The time that is used calculate the direction v takes most of the remaining amount of time. We can also see that the number of iterations that the algorithms 'allConstr' and 'WallConstr' completed, is about half the number of iterations of the other algorithms. Therefore, the algorithms 'allConstr' and 'WallConstr' are not really suited for problems with too many possible directions v^l, many free variables #F and very negative values of z_l.

All other algorithms also reach the time limit of 5 minutes, but they all find a feasible result. It takes for these other algorithms at least 200 seconds to find such a point, except for the algorithm 'rosen' which starts of course at a feasible initial point. We see that especially the algorithm 'W3maxConstr' takes a long time to find a feasible result. This might be because algorithm 'W3maxConstr' weighs the directions v^l according to the value of z_l (equation (4.15)), so it favors decreasing constraints with a high value of z_l rather than decreasing constraints with a low value of z_l. This means that many constraints might stay infeasible for a long time, which increases the computational burden.
We see that the algorithm 'rosen' has the highest objective value. This algorithm is shortly followed by the algorithms 'base', 'maxConstr' and '3maxConstr'. The difference between these objective values is less than 1% of the value of the objective function. Therefore, for Dataset 3, the algorithm 'rosen' is again the best algorithm, but the algorithms 'base', 'maxConstr' and '3maxConstr' also give good results.

Now we continue with the final dataset, Dataset 4. Recall that this dataset is based on Dataset 3 and is created such that the value of z_l is never smaller than -10^5, so the initial point that is found in Phase 1 is relatively close to being a feasible point. We see that all variants of the complete algorithm still hit the time limit of 5 minutes, but that the final objective value of the algorithm 'rosen' is now lower than the final objective value of all other algorithms. Therefore the algorithm 'rosen' is for this dataset the worst algorithm. This is a big difference with the other

datasets, where 'rosen' generally gave the best results.
The other algorithms all give very similar results. All variants of the complete algorithm, except for the algorithm 'allConstr', find a feasible point within 19.5 to 23.5 seconds. The objective value of the feasible point that is found is also very similar for all variants of the complete algorithm. This is equal to $1.818 \cdot 10^8$ or $1.819 \cdot 10^8$. Notice that the difference between this initial feasible point and the final objective value of all variants of the complete algorithm is at most 0.2%. Therefore, if the computation time is also important when calculating a solution, then it could be considered to terminate the algorithms shortly after the first feasible point is found. This would save roughly 270 seconds, with only a minor loss of finding a 0.2% worse objective value.
We can see that the final point of the algorithm 'rosen' is also $1.819 \cdot 10^8$. Therefore, the algorithm 'rosen' takes 300 seconds to find an objective value that most other algorithms find within roughly 30 seconds. This again shows that the algorithm 'rosen' is not a good algorithm for this dataset.

Now we have analyzed most of the results of the datasets. In the next section, we analyze some final results regarding the line search algorithms, computational improvements when using parallelization and advantages that can be made when combining some of the different versions of the complete algorithm.

5.2 Line search, parallelization and combining algorithms

In this section, we present some results that did not fit in the previous sections. We first analyze whether backtracking line search or golden section search should be used. Then we analyze why quadratic line search performs so poorly compared to the other algorithms, which explains why the algorithm 'quadApprox' gives poor results for all datasets.
We did not use any parallelization in our code, so we also analyze in this section which computational advantages parallelization can bring when calculating the directions v and v^l. Finally, we analyze some advantages of combining one or more algorithms and in which cases which algorithm should be used.

For most datasets, we saw that the algorithm 'base' performed slightly better than the algorithm 'gss'. Recall that the difference between these algorithms is that 'base' uses backtracking line search, while 'gss' uses golden section search. In Dataset 1 and Dataset 2, we saw that the computation time for the algorithm 'base' was slightly lower than the computation time for 'gss'. In Dataset 3, both algorithms hit the time limit of 5 minutes, but we did see a difference in the objective value. The objective value of the algorithm 'base' is about 2% bigger than 'gss'. Therefore, we observe that backtracking line search works slightly better than golden section search for our problem, but there is not much of a difference.

There are two reasons why the algorithm 'quadApprox' is outperformed by all other algorithms on most datasets.
First, we observed some issues in the convergence of the algorithm 'quadApprox'. Recall from Subsection 2.4.3 that in this algorithm the step size α is calculated with the function $\alpha = -\frac{\nabla S(Y)^\top v}{v^\top \nabla^2 S(Y) v} >$

0. We noticed that $v^\top \nabla^2 S(Y)v$ tends to converge to zero when the algorithm converges. This leads to numerical instability due to rounding issues and therefore inaccurate step sizes α.
The second reason is that the calculation of the Hessian $\nabla^2 S(Y)$ takes a significantly longer time than evaluating the objective function $S(Y)$ several times, which is needed for the other line search algorithms. Therefore, the advantage of the algorithm 'quadApprox' that it gives a relatively accurate step size α does not weigh up against the additional computational costs.

We can show that all variants of the complete algorithm that are using our modified version of Rosen's projection algorithm, so all algorithms except for 'rosen', can be improved by using parallelization. First, we note that Step 1 and Step 2 of the algorithm 'base' (Section 4.2), can be calculated at the same time, because there is no dependence between Step 1 and Step 2. In other words, the directions v and v^l, $\forall l \in \mathcal{B}(z)$ can be calculated at the same time. Most modern computers consist of a CPU with more than one logical core. Each logical core can calculate a direction v or v^l at the same time as the other logical cores. This implies that if there are as many logical cores in a computer as there are directions to be calculated, then calculating all directions takes as long as the time that is needed to calculate the direction with the highest computational cost.
In other words, calculating v can take about as long as calculating v and v^l, $\forall l \in \mathcal{B}(z)$. We mentioned before that we only use a single core of our CPU. So if more than a single core is used, then a substantial amount of the time to calculate the slack directions 'Slack' can be subtracted from 'Total', the total amount of time that was used by an algorithm. This is especially advantageous for algorithms were many directions need to be calculated in each iteration, so especially the algorithms 'allConstr', 'WallConstr', '3maxConstr' and 'W3maxConstr'.
This is why we also mention the result 'Total-S' in Table A.3 to Table A.9. When we compare for Dataset 1 and Dataset 2 'Total-S' between all variants of the complete algorithm, then we can see that the algorithms that are using more than one direction per iteration perform better. This means that the algorithms 'allConstr', 'WallConstr', '3maxConstr' and 'W3maxConstr', still have a higher computation time for 'Total-S' than the algorithm 'rosen', but that the gap is definitely smaller. The algorithms 'allConstr', 'WallConstr', '3maxConstr' and 'W3maxConstr' also perform better than the other algorithms. For Dataset 3 and Dataset 4, it is harder to compare the results, because all algorithms were stopped after 5 minutes have passed. However, we can see that 'Total-S' is relatively small compared to 'Total' for the algorithms 'allConstr', 'WallConstr', '3maxConstr' and 'W3maxConstr'. Therefore, we expect that parallelization also increases the performance of these algorithms for Dataset 3 and 4.

In Table A.3, A.5, A.7 and A.9, we see that 'Init' is typically below 0.5 seconds. This implies that doing Phase 1 of the complete algorithm takes a small amount of time for all variants of the complete algorithms. For the algorithm 'rosen' the 'Init' time implies that finding an initial starting point only takes a very short time.
If it is noticed after Phase 1 that there are many violated constraints and that z_l is a big negative value for a few constraints l, then this means that the algorithms, that use our modified version of Rosen's projection algorithm, might take a very long time to find a feasible point. Therefore, in this

case it would be recommended to only use the original version of Rosen's projection algorithm and not our modified version. However, when z_l only a bit negative, then we might prefer our modified version of Rosen's projection algorithm. When $z_l = 0$ for all l after Phase 1, then the problem is feasible and globally optimal in Phase 2, so our modified algorithm is definitely preferred.
If there is a good estimation available about the optimal value of the objective function, then it can be considered to first find a few feasible starting points. If these starting points are close to the estimation of the optimal value of the objective function, then we start the algorithm 'rosen'. Otherwise, we pick any of the other algorithms.
Therefore, with the use of these strategies that select an algorithm based on the results on the initialization steps, it might be possible to select the algorithm that gives the best objective value in the shortest amount of time.

This concludes all the results that we have found. In the next section, we present some sensitivity analysis on the parameters. This sensitivity analysis can be used to fine-tune the algorithms, such that the running time (and due to the time limit, also the final objective value) of the algorithms can be improved.

5.3 Sensitivity of parameters

In this section, we present the sensitivity analysis for many of the parameters that we have discussed in the previous chapters. The sensitivity analysis is similar for most algorithms and datasets. Therefore we only present the sensitivity analysis for Dataset 1 with the algorithm 'allConstr'. We choose this algorithm, because there is no randomness in this algorithm. Dataset 1 is chosen because the algorithm 'allConstr' terminates within the time limit and because the line search part of the algorithm is not too short.
The sensitivity analysis that we do in this section only applies to problems with a similar objective function and similar constraints as the problem that we have defined in Chapter 3. If an entirely different objective function is chosen or if the constraints are radically different, then the best value of parameters might be slightly different for most of the algorithms.

We perform a sensitivity analysis on the parameters, α, h^0, s_r, ρ, C and τ. These parameters have also been considered in Chapter 4. We briefly recapitulate the interpretation of the parameters in Table 5.4. In Subsection 4.3.4, the parameters have been discussed more thoroughly.
Table 5.4 is very similar in interpretation to Table 5.1. The headers 'Parameter' and 'Description' should be self explanatory. The header 'Section' shows the sections in which there is an algorithm that uses the parameter.

The values that we use in the sensitivity analysis of the parameters can be found in Table 5.5. For each parameter, we consider three possible values. The original values that we gave each of the parameters in Subsection 4.3.4 are called 'Original'. The results for 'Original' have already been given in Section 5.1. The completion time of the algorithm is 20.24 seconds, which we have seen in Table A.5. The new values of the parameters are denoted by '–' and '+', where '–' is the

Tab. 5.4: Description of the algorithms

Parameter	Description	Section
α	Step size when direction $v^l \neq 0$ for some l.	4.2
h^0	Balances direction v with all directions v^l.	2.4
s_r	Factor used to decide if Option (c) should be used in Rosen's algorithm.	2.3 & 4.2
ρ	Factor that decreases α in backtracking line search	2.4.2
C	Factor that checks if the objective decreased enough in backtracking line search.	2.4.2
τ	Tolerance value for golden section search.	2.4.1

smallest value of the two new values. Note that the parameters α and h^0 are only relevant when a feasible solution is not yet found. The parameters ρ, C, and τ are only relevant when the line search part of the algorithm starts. The parameter s_r is relevant throughout the algorithm. This means that we expect more volatility in the sensitivity results for s_r than for the other parameters. We know that for Dataset 1, most time of the algorithm 'allConstr' is spent on finding the first feasible point. Therefore, we expect that α and h^0 have a bigger effect on the total computation time than ρ, C, and τ.

Tab. 5.5: Sensitivity analysis characteristics

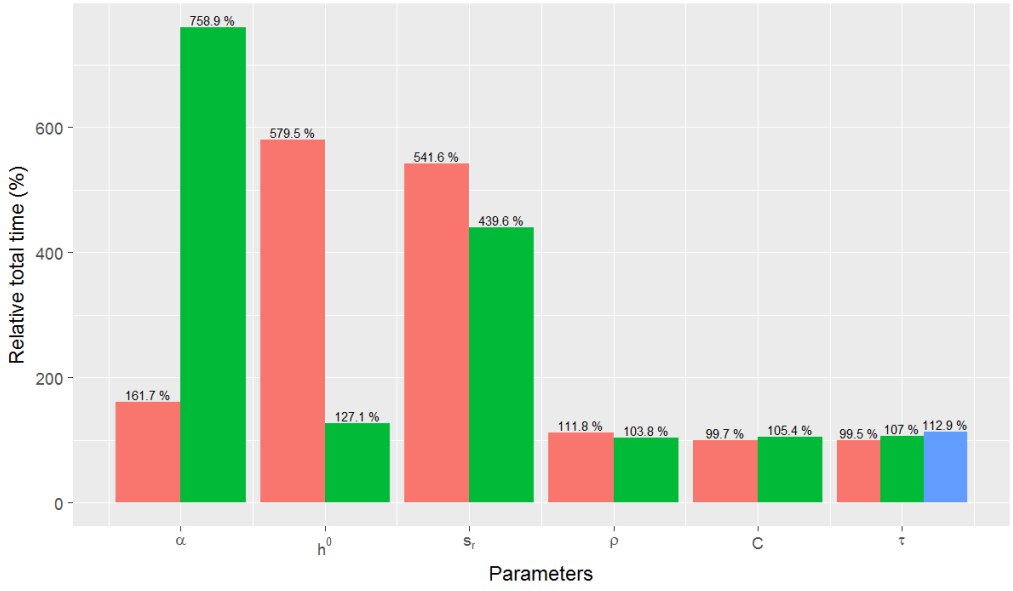
	α	h^0	s_r	ρ	C	τ
same	$\frac{\alpha_{max}}{2}$	0.5	1.25	0.1	10^{-5}	10^{-6}
$-$	$\frac{\alpha_{max}}{4}$	0.25	0.125	0.01	10^{-3}	10^{-3}
$+$	$\frac{3\alpha_{max}}{4}$	0.75	12.5	0.5	10^{-1}	$\min\left(1, \frac{\alpha_{\max}}{1000}\right)$

To do a sensitivity analysis on τ, we have to use golden section search. Recall that the algorithm 'allConstr' uses backtracking line search. Therefore, to compare the result of τ, we also have to use the algorithm 'allConstr' with the parameter set 'Original', but with the golden section search. This gives us one additional new result. In total, we calculate 13 different sensitivity analyses. Three for τ and two for each of the parameters α, h^0, s_r, ρ and C. We run each algorithm five times and take the average total time as the result.

We report the results of each of the 13 sensitivity analyses in Figure 5.1. Each bar in this figure shows the computation time of each sensitivity analysis relative to the computation time of 20.24 seconds of the 'allConstr' algorithm. The red bars correspond to the results for the '–' case. The green bars correspond to the '+' case. The single blue bar is the result of the algorithm 'allConstr' that uses the golden section search algorithm instead of the backtracking line search algorithm.

Now we analyze the results from Figure 5.1. We can see that changing the parameters α, h^0 or s_r can result in huge fluctuations in the total computation time, while the changes to ρ, C and τ result in minor changes to the total computation time. We already expected in the beginning of this section that α, h^0 or s_r would result in a bigger fluctuation in the computation time than ρ, C and

Fig. 5.1: Sensitivity of parameters



τ. However, we did not expect that relatively small changes to ρ, C and τ could result in computation times that are up to 7.6 times longer than the computation time of the algorithm 'allConstr'.

For the step size parameter α, we can see that too big step sizes are very detrimental to the total computation time. To see why, we had to analyze the individual step sizes. Here we that when a big step α is made during an iteration, then often in the next few iterations only small steps α are being made. This is, therefore, one possible reason for the increase in computation time.
We also see in Figure 5.1 that when $\alpha = \frac{\alpha_{\max}}{4}$ there is also an increase in the computation time. This is because not enough progress is made during each iteration, so more iterations are needed.

If the value of h^0 is reduced, then we can see in equation (4.12), (4.14) or (4.15) that for every iteration, the direction v is given far less weight than the direction v^l. Recall that the direction v improves the objective value $S(Y)$, while the direction v^l increases the value of the slack constraint z_l. We saw in our results that a small weight on v also implies that the step sizes remained very small during each iteration. This might be because increasing z_l too rapidly means that we get too close to other constraints, which limits the maximum step size.
When h^0 is big, then we noticed that z^l did not increase that much per iteration. This increases the time before the first feasible solution is found, so it increases the computation time.

For the parameter s_r we can see in Figure 5.1 that if $s_r = 0.125$, we get a huge increase in computation time. If s_r is small then this implies that the inequality $||Pb_l^\top|| > s_r|u|$ in Option (b) in Step 1 and Step 2 of our modified projection algorithm in Section 4.2 is often satisfied. This means that we often do not have to consider Option (c) in Step 1 or Step 2, so we save some computation time. However, we do lose a lot of computation time, because we take inefficient step

sizes. We know that if u, the most negative Lagrange multiplier μ_l, is a very negative value, then we can find better ascent direction by removing the constraint that belongs to u. So with a low value of s_r, we were skipping good ascent directions, which resulted in the increase in computation time.
If $s_r = 12.5$, so if we increase s_r, then we consider Option (c) in Step 1 or Step 2 of our modified projection algorithm more often and this increases the computation time.

The remaining parameters ρ, C and τ are only relevant when we have found the first feasible solution. When we set $\rho = 0.01$, then this implies that we skip a lot of good potential values for α, but we often find the step size α in less time. We see in Figure 5.1 that the time that we save, does not weigh up against the loss of some good potential values for α.
When we instead increase ρ, then we have the opposite effect. We generally consider more values for α, but it generally takes a longer time before we find a step size α that is good enough. We see in Figure 5.1 that the extra time that we spend on finding a better α does not give us the final result in a shorter amount of time. However, the difference between $\rho = 0.1$ and $\rho = 0.5$ is very small. It might be possible to slightly speed up the computation time of the backtracking line search algorithm by setting ρ to some value between 0.1 and 0.5.

The total computation time is not very sensitive to changes in C. The bigger the value of C, the more progress that the step size α should make. So it takes more time to find the step size α if C is large. We see that we have a minor improvement in the total computation time when $C = 10^{-3}$, so setting C between 10^{-3} and 10^{-5} slightly improves the speed of the backtracking line search algorithm. We can also see that increasing C to 10^{-1} implies that it either takes too much time to find a good α or that α is too small in each iteration.

Finally, we analyze the parameter τ. We can see in the blue bar in Figure 5.1 that using golden section search instead of the regular version of 'allConstr' results in a 12.9% increase in the computation time. We have also seen a similar difference in the results of the algorithms 'base' and 'gss' for Dataset 1 and Dataset 2, in Table A.3 and A.5. For both datasets, the algorithm 'base' is a few percent faster than the algorithm 'gss'.
The red and the green bars show that we can improve the computation time of the golden section search algorithms by increasing the value of τ. Recall that τ indicates how close the step size α should be to a local maximum. If τ is small, then we have to calculate more steps in the golden section search algorithm, but we get a more accurate value of α. It is the other way around if τ is big. Therefore, the results in Figure 5.1 show that the accuracy of the step size is not important for the calculation of the total computation time and that a lot of time is saved when we increase the value of τ.

This concludes the sensitivity analysis and with that, we also conclude the analysis of the results of the algorithms that we presented in Chapter 4. In Chapter 6, we present our conclusions and provide a number of possible extensions and improvements that could be made to improve the results or the computation times of the various algorithms.

6. CONCLUSIONS

In Chapter 2, we have reviewed some literature, in particular in Section 2.3 we have reviewed Rosen's projection algorithm and in Section 2.4 we have reviewed three line search algorithms. In Chapter 3, we presented problem (3.8) that we have solved in Chapter 4 with our complete algorithm that consists of two phases. In Phase 1 of our complete algorithm, we derived that we can find a globally optimal point to problem (4.1). This problem has the same objective function as problem (3.8), but it only has a subset of the total number of constraints. In Phase 2 of the complete algorithm, we modified Rosen's projection algorithm such that Rosen's projection algorithm could also start with infeasible starting points. Finally, in Chapter 5 we compared several versions of our modified version of Rosen's projection algorithm with the original version of Rosen's projection algorithm.
In this final Chapter, we give some conclusions about the various topics that we have presented in the previous chapters of this thesis. We end this chapter with some recommendations for future improvements to the algorithms.

Our most important result in this thesis is the complete algorithm that we derived in Chapter 4 to solve problem (3.8). In Subsection 4.1.3, we presented Phase 1 of the complete algorithm. In Phase 1 we found a globally optimal point to problem (4.1) by solving the system of KKT conditions (4.2a) to (4.2j). If this point is a feasible point of problem (3.8), then it is also a globally optimal point for this problem. We saw in Chapter 5 that a solution to problem (4.1) can typically be found in a very short time, which is an important result of Phase 1 of the complete algorithm. The other important result of Phase 1 is that we can use the result of Phase 1 in Phase 2 of the complete algorithm, if the result of Phase 1 is not a feasible point of problem (3.8).
We presented Phase 2 of the complete algorithm in Section 4.2. The important result from this section is that we modified Rosen's projection algorithm, such that our modified version of Rosen's projection algorithm can find a feasible solution with an infeasible starting point. In Chapter 4, we also presented several variants of the complete algorithm. These variants were used in Chapter 5 to solve several datasets.
In Chapter 5, we have seen several results for each of the versions of the complete algorithm for four datasets. We briefly repeat the most important results.
We have seen in Dataset 1 and Dataset 3 that some of the algorithms that use the modified version of Rosen's algorithm give comparable results to the original version of Rosen's algorithm. However, the original version of Rosen's algorithm was for both these datasets a bit faster. In Dataset 3 we saw that when there are many constraints that are violated after Phase 1 that the algorithms that make use of all possible directions v^l do not perform well.
We also saw that the original version of Rosen's algorithm is clearly better than all the other

versions of the complete algorithm for Dataset 2. However, the other algorithms clearly outperformed the original version of Rosen's algorithm for Dataset 4. Therefore, we conclude that our modifications definitely provide added value to Rosen's algorithm.
The final observation from Chapter 5 is that we saw in Section 5.3 that the computation time of the versions of the complete algorithm is very sensitive to changes in some parameters. If these parameters are tuned further, then it might be possible to slightly speed up the computation time, however, we do not expect that much improvement can be gained from this.

Now we continue this chapter by mentioning some possible improvements and extensions to the algorithms. We already mentioned one possible improvement in Section 5.1. In this section, we noted that we could speed up the computation time of variants of the complete algorithm that use our modified version of Rosen's algorithm by calculating the directions v and v^l in parallel instead of calculating them in order.

Another possible improvement is to use some kind of line search algorithm to calculate the step size α in our modified version of Rosen's algorithm in Section 4.2 if $v^l \neq 0$ for some $l \in \mathcal{B}(z)$. Currently, we use the fixed step size $\alpha = \frac{\alpha_{\max}}{2}$, but we believe that it might be possible to improve on this.
A possible improvement is to make a function that always increases for any feasible V and that balances the objective and the amount that z_l improves for all α. Then use a line search algorithm on this function.

We saw in Section 5.1 that a large part of the computation time of each of the versions of the complete algorithm is used to calculate the projection matrix P, even though the matrix M that generates P does not change much between iterations. The matrix M of Option (b) and Option (c) in Step 1 and Step 2 of our modified projection algorithm also only differs by one row. For most datasets and algorithms calculating P took roughly 80% of the total computation time. Therefore, if this calculation is made faster, then this means that there are significant improvements possible to the total computation time. In Chapter 3 of the original paper from Rosen (1960), there were some methods mentioned to update the projection matrix P between iterations. However, it is not clear if this method also works when the pseudoinverse is used to calculate the projection matrix. If this method does not work for pseudoinverses, then it is definitely worthwhile to investigate other methods, because huge improvements could be made.
A possible method that might speed up the computation time of calculating P is by using rank-1 updates. A rank-1 update of a matrix M results in the matrix $M' = M + vw^\top$, where $v \in \mathbb{R}^L$ and $w \in \mathbb{R}^N$. The paper by Meyer (1973) provides faster ways to calculate the pseudoinverse M'^+ if M^+ is known.

The following improvement might be interesting from a business point of view. In problem (3.8) we want to maximize the sales $S(Y)$ subject to the total budget constraint $\sum_{i=1}^{m}\sum_{j=1}^{t} y_{ij} \leq D$ and other constraints. However, a company might want to minimize the total budget $\sum_{i=1}^{m}\sum_{j=1}^{t} y_{ij}$,

subject to the constraint $S(Y) = R$, where $R \in \mathbb{R}_+$ is some parameter that denotes a target level of sales. The other constraints are taken as in problem (3.8).
This problem cannot be solved directly by the original version of Rosen's algorithm or by our modified version of Rosen's algorithm. However, we can use a combination of these algorithms to solve this problem. We first estimate some value of D for which we expect that it will result in a number of sales $S(Y)$ that is close to R. We can now formulate the following algorithm to solve the new problem:

1. In the first step we solve problem (3.8) as we described throughout the thesis. This gives the result Y^1. If $S(Y^1) = R$, then we stop the algorithm. Otherwise, we set $r \leftarrow 1$ and go to Step 2.

2. In Step 2, we consider Option (a) and Option (b).

   (a) If $S(Y^r) < R$, then we increase the parameter D by some step size. This means that the point Y^r is still feasible, but not optimal. We use Rosen's algorithm to calculate the point Y^{r+1} and we go to Step 3.

   (b) If $S(Y^r) > R$, then we decrease the parameter D by some step size. This makes the point Y^r infeasible. We use our modified version or Rosen's projection algorithm to calculate the point Y^{r+1} and we go to Step 3.

3. If $S(Y^r)$ is sufficiently close to R, then the algorithm is stopped. If $S(Y^r) = S(Y^{r+1})$, it is not possible to reach $S(Y^r) = R$ and the algorithm is stopped. Otherwise, we go to step 2.

We know from the results of Dataset 2, that when the original version of Rosen's algorithm starts close to a locally optimal point, then it quickly finds that locally optimal point. In Step 2, Option (a), the point Y^r is close to the point Y^{r+1}, because only the parameter D changes by some step size. Therefore, Step 2, Option (a) of this algorithm should be solved quickly.
In the results of Dataset 4, we observed that our modified version of Rosen's algorithm works well if the number of violated constraints is small and if the initial feasibility gap is small. If in Step 2, Option (b), the parameter D is decreased, then Y^r is an infeasible starting point. We know that z_l is only negative for a single index l that belongs to the constraint $\sum_{i=1}^{m}\sum_{j=1}^{t} y_{ij} \leq D$ and z_l is as negative as the step size, so the initial feasibility gap is small. Therefore our modified projection algorithm should quickly find a locally optimal point for Step 2, Option (b).
The step size in Step 2 should be taken such that $S(Y^r)$ moves closer to R. The exact step size is open for further research.

In Subsection 2.5.1 and Subsection 2.5.2, we reviewed Sequential Quadratic Programming and barrier functions. These optimization methods can also be used to solve problems that are similar to problem (3.8). We did not use these optimization methods in Chapter 4 of this thesis. However, it could be interesting to compare the performance of these algorithms with the performance of Rosen's algorithm or our modified version of Rosen's algorithm. It might also be possible to combine some ideas of Sequential Quadratic Programming and barrier functions with the ideas behind Rosen's algorithm to obtain an even better algorithm. Another option is to see if we can

identify for a dataset the algorithm that will likely give the best or the fastest results, as we analyzed for Rosen's algorithm and our modified algorithm in Section 5.2.

REFERENCES

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., ... Sorensen, D. (1999). *LAPACK users' guide* (Third ed.). Philadelphia, PA: Society for Industrial and Applied Mathematics.

Armijo, L. (1966). Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, *16*(1), 1–3.

Bazaraa, M. S., Sherali, H. D., & Shetty, C. M. (2006). *Nonlinear programming: Theory and algorithms* (3rd ed.). Wiley.

Berkelaar, M., Eikland, K., & Notebaert, P. (2004). *Open source (mixed-integer) linear programming system.* Retrieved 2017-07-04, from `http://lpsolve.sourceforge.net/5.5/.`

Bjerhammar, A. (1951). Application of calculus of matrices to method of least squares; with special references to geodetic calculations. *Transactions of the Royal Institute of Technology*, *49*, 1–86.

Boggs, P. T., & Tolle, J. W. (1996). Sequential quadratic programming. *Acte Numerica*, *5*, 1–52.

Boyd, S. P., Kim, S., Vandenberghe, L., & Hassibi, A. (2007). A tutorial on geometric programming. *Optimization and Engineering*, *8*, 67–127.

Caron, R., McDonald, J., & Ponic, C. (1989). A degenerate extreme point strategy for the classification of linear constraints as redundant or necessary. *Journal of Optimization Theory and Applications*, *62*, 225–237.

Du, D. Z., Wu, F., & Zhang, X. S. (1990). On rosen's gradient projection methods. *Annals of Operations Research*, *24*, 11–28.

Forsgren, A., Gill, E. P., & Wright, H. M. (2002). Interior methods for nonlinear optimization. *Society for Industrial and Applied Mathematics*, *44*, 525–597.

He, G. Z., & Zhang, X. S. (1986). A convergence theorem of rosen's gradient projection method. *Mathematical Programming*, *36*(2), 135–144.

IBM. (2013). *Cplex 12.6 announce.* Retrieved 2017-06-15, from `https://www.ibm.com/developerworks/community/blogs/jfp/entry/cplex_12_6.`

IBM. (2015). *Ibm ilog cplex optimization studio CPLEX user's manual* (6.3 ed.). Armonk, NY. Retrieved 2017-07-05, from `https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.6.3.`

Karush, W. (1939). *Minima of functions of several variables with inequalities as side constraints* (Unpublished master's thesis). University of Chicago, Chicago, Illinois.

Kiefer, R. (1953). Sequential minimax search for a maximum. In *Proceedings of the american mathematical society* (Vol. 4, pp. 502–506).

Kuhn, H. W., & Tucker, A. W. (1951, aug). Nonlinear programming. In J. Neyman (Ed.), *Proceedings of 2nd berkeley symposium* (p. 481-492). Berkeley: University of California Press.

Meyer, C. (1973). Generalized Inversion of Modified Matrices. *SIAM Journal on Applied Mathematics*, *24*, 9.

Moore, E. H. (1920). On the reciprocal of the general algebraic matrix. In *Bulletin of the american mathematical society* (Vol. 26, pp. 394–395).

Nocedal, J., & Wright, S. J. (1999). *Numerical optimization* (P. Glynn & S. M. Robinson, Eds.). Springer.

Paulraj, S., & Sumathi, P. (2010). A Comparative Study of Redundant Constraints Identification Methods in Linear Programming Problems. *Mathematical Problems in Engineering*, *2010*, 1–16.

Penrose, R. (1955). A generalized inverse for matrices. In *Mathematical proceedings of the cambridge philosophical society* (Vol. 51, pp. 406–413).

Ritter, K. (1980). Convergence and superlinear convergence of algorithms for linearly constrained minimization problems. In L. Dixon & S. G.P. (Eds.), *Nonlinear optimization: Theory and algorithms* (Vol. 2, pp. 221–251).

Rosen, J. B. (1960). The Gradient Projection Method for Nonlinear Programming. *Journal of the Society for Industrial and Applied Mathematics*, *8*(1), 181–217.

Rosen, J. B. (1961). The Gradient Projection Method for Nonlinear Programming. Part II. Nonlinear Constraints. *Journal of the Society for Industrial and Applied Mathematics*, *9*(4), 514–532.

Sahni, S. (1974). Computationally related problems. *SIAM Journal on Computing*, *3*, 262–279.

Wolfe, P. (1971). Convergence conditions for ascent methods. *SIAM Review*, *13*(2), 185–188.

Zoutendijk, G. (1960). *Methods of feasible directions: a study in linear and non-linear programming* (Vol. 51). Elsevier.

Appendix A

RESULTS DIFFERENT ALGORITHMS

The appendix shows the tables with the results from the various algorithms in Chapter 5. Table A.1 is the legend for all names that are used in Table A.2 to Table A.9.

Tab. A.1: Legend for Table A.2 to Table A.9

Name result	Description
Name	The name of the algorithm. For a summary of the algorithms, see Table 5.1
Objective	The final objective value found by the algorithm.
AbsFeasGap	The biggest absolute feasibility gap, so the absolute gap between $b_l Y^*$ and q for any violated constraint for which $b_l Y^* < q_l$.
RelFeasGap	The biggest relative feasibility gap, so the relative gap $\frac{b_l Y^* - q_l}{q_l}$ for any violated constraint for which $b_l Y^* < q_l$.
ObjFirstFeas	The first feasible objective value found by the algorithm.
Iter	The number of iterations of the projection, in other words, the number times a new V has been calculated.
Total	The total time in seconds when the algorithm terminates.
Total-S	The total time in seconds when the algorithm terminates minus the time in seconds that is spent on increasing the slack value z.
Feas	The time in seconds when the first feasible point is found.
Init	The time in seconds needed for all initialization steps before Rosen's algorithm starts. This corresponds to Phase 1 or finding an initial point.
Improve	The time in seconds in the algorithm that is spent on finding a direction v that improves the objective.
Slack	The time in seconds that is spent on increasing the slack value z.
Option (c)	The time in seconds that is spent on Option (c) in either the original version of Rosen's algorithm from Section 2.3 or in (either of the steps of) the modified version in Section 4.2 (Option (c)).
LS	The time in seconds used to do in the line search algorithm.
>-1E-15	A number is zero up to machine accuracy
NA	No number assigned, for example when no feasible result is found within the time limit.

All reported results are obtained when each algorithm finishes, except for 'ObjFeas', which is obtained when the first feasible solution is found. Note that in in the tables that report the calculation time we have that Total $\approx$ Init + Improve + Slack + LS. The time for Option (c) is a part of both Improve and Slack. There is some minor difference between Total and Init + Improve + Slack + LS. This difference is caused by some rounding errors in the calculation of the processing times.

A.1 Dataset 1

Tab. A.2: Results of the algorithms for Dataset 1

Name	Objective	ObjFirstFeas	AbsFeasGap	RelFeasGap	Iter
base	3.065E+05	3.062E+05	>-1E-15	>-1E-15	1295
quadApprox	3.065E+05	3.063E+05	>-1E-15	>-1E-15	1140
gss	3.065E+05	3.062E+05	-9.46E-11	-2.54E-15	1168
allConstr	3.065E+05	3.057E+05	-1.00E-08	-2.68E-13	1097
WallConstr	3.065E+05	3.042E+05	>-1E-15	>-1E-15	974
maxConstr	3.065E+05	3.036E+05	-1.16E-04	-2.20E-10	1491
3maxConstr	3.065E+05	3.061E+05	-9.67E-03	-2.01E-09	1045
W3maxConstr	3.065E+05	3.042E+05	>-1E-15	>-1E-15	917
rosen	3.065E+05	2.781E+05	-4.66E-08	-1.24E-13	656

Tab. A.3: Time results in seconds of the algorithms for Dataset 1

Name	Total	Total-S	Feas	Init	Improve	Slack	Option (c)	LS
base	17.20	11.96	14.77	0.28	7.70	5.24	1.16	2.39
quadApprox	46.44	41.76	12.76	0.29	6.33	4.68	1.07	34.01
gss	17.94	13.26	13.56	0.36	6.77	4.68	1.16	4.69
allConstr	20.24	11.07	17.24	0.36	6.72	9.17	1.26	2.42
WallConstr	20.76	9.84	16.11	0.31	6.05	10.92	1.27	2.40
maxConstr	29.91	19.65	24.05	0.33	14.79	10.26	8.15	2.99
3maxConstr	18.67	10.30	15.79	0.34	6.31	8.37	1.13	2.29
W3maxConstr	18.84	9.21	15.04	0.33	5.42	9.63	1.42	2.07
rosen	6.03	6.03	0.00	0.33	3.81	0.00	0.20	1.43

A.2 Dataset 2

Tab. A.4: Results of the algorithms for Dataset 2

Name	Objective	ObjFirstFeas	AbsFeasGap	RelFeasGap	Iter
base	3.988E+07	3.987E+07	>-1E-15	>-1E-15	652
quadApprox	3.988E+07	3.987E+07	-8.50E-05	-5.31E-11	654
gss	3.988E+07	3.987E+07	-1.00E-08	-4.35E-13	610
allConstr	3.988E+07	3.987E+07	-9.48E-05	-7.90E-11	554
WallConstr	3.988E+07	3.985E+07	>-1E-15	>-1E-15	563
maxConstr	3.988E+07	3.987E+07	-2.28E-03	-4.75E-10	721
3maxConstr	3.988E+07	3.987E+07	-9.48E-05	-7.90E-11	554
W3maxConstr	3.988E+07	3.985E+07	>-1E-15	>-1E-15	563
rosen	3.994E+07	3.953E+07	-4.49E-05	-3.24E-05	124

Tab. A.5: Time results in seconds of the algorithms for Dataset 2

Name	Total	Total-S	Feas	Init	Improve	Slack	Option (c)	LS
base	11.45	7.77	10.37	0.25	5.28	3.68	0.69	1.52
quadApprox	31.63	27.98	9.70	0.30	5.17	3.65	0.56	21.72
gss	12.01	8.45	10.17	0.32	5.14	3.56	0.50	2.26
allConstr	11.57	6.74	10.33	0.30	4.66	4.83	0.51	1.13
WallConstr	13.77	7.31	12.46	0.31	4.89	6.46	0.86	1.43
maxConstr	12.96	8.91	11.43	0.28	6.45	4.05	0.91	1.33
3maxConstr	11.03	6.27	9.89	0.29	4.31	4.76	0.68	1.22
W3maxConstr	13.30	6.92	12.03	0.31	4.67	6.38	0.72	1.22
rosen	1.81	1.81	0.00	0.31	1.09	0.00	0.15	0.26

A.3 Dataset 3

Tab. A.6: Results of the algorithms for Dataset 3

Name	Objective	ObjFirstFeas	AbsFeasGap	RelFeasGap	Iter
base	1.798E+08	1.700E+08	-4.15E-03	-2.01E-08	2309
quadApprox	1.720E+08	1.680E+08	-4.99E-03	-1.98E-08	1540
gss	1.758E+08	1.700E+08	-4.42E-03	-6.72E-07	2145
allConstr	NA	NA	-1.67E+05	-2.67E-01	789
WallConstr	NA	NA	-5.73E+04	-1.34E-01	580
maxConstr	1.796E+08	1.688E+08	-4.82E-03	-3.14E-08	2459
3maxConstr	1.790E+08	1.700E+08	-4.71E-03	-2.74E-08	1771
W3maxConstr	1.760E+08	1.698E+08	-4.95E-03	-2.21E-08	1386
rosen	1.802E+08	1.500E+08	-2.93E-06	-3.51E-07	4007

Tab. A.7: Time results in seconds of the algorithms for Dataset 3

Name	Total	Total-S	Feas	Init	Improve	Slack	Option (c)	LS
base	300.12	201.04	215.30	0.39	170.73	99.08	11.24	25.46
quadApprox	301.72	188.59	244.46	0.39	117.73	113.13	7.53	66.96
gss	300.14	199.92	218.55	0.41	161.41	100.22	9.19	33.47
allConstr	300.45	69.88	NA	0.48	60.77	230.57	8.89	6.81
WallConstr	300.85	50.44	NA	0.50	43.29	250.41	9.64	5.05
maxConstr	300.16	207.01	202.26	0.40	177.64	93.15	9.80	23.99
3maxConstr	300.10	148.01	211.73	0.39	128.24	152.09	9.96	15.73
W3maxConstr	300.09	118.03	253.84	0.39	102.19	182.06	10.02	12.40
rosen	300.22	300.22	0.00	0.50	258.27	0.00	8.71	34.05

A.4 Dataset 4

Tab. A.8: Results of the algorithms for Dataset 4

Name	Objective	ObjFirstFeas	AbsFeasGap	RelFeasGap	Iter
base	1.820E+08	1.819E+08	-4.02E-03	-3.37E-08	3541
quadApprox	1.819E+08	1.819E+08	-4.60E-03	-1.19E-08	460
gss	1.820E+08	1.819E+08	-4.94E-03	-9.20E-08	3074
allConstr	1.820E+08	1.818E+08	-4.41E-03	-1.05E-08	3288
WallConstr	1.821E+08	1.818E+08	-4.92E-03	-4.62E-08	3478
maxConstr	1.820E+08	1.819E+08	-4.87E-03	-1.60E-08	3590
3maxConstr	1.820E+08	1.818E+08	-4.91E-03	-2.41E-08	3527
W3maxConstr	1.821E+08	1.818E+08	-4.57E-03	-3.35E-08	3558
rosen	1.819E+08	1.519E+08	-9.86E-06	-6.54E-07	3724

Tab. A.9: Time results in seconds of the algorithms for Dataset 4

Name	Total	Total-S	Feas	Init	Improve	Slack	Option (c)	LS
base	300.14	289.60	23.47	0.41	246.84	10.54	6.76	35.40
quadApprox	301.25	290.67	23.48	0.39	33.58	10.58	2.85	255.60
gss	300.12	289.73	22.64	0.41	212.57	10.39	5.65	69.97
allConstr	300.12	268.63	41.60	0.44	229.00	31.49	7.78	32.53
WallConstr	300.12	283.40	20.66	0.40	239.40	16.72	7.08	37.02
maxConstr	300.10	290.13	22.45	0.39	247.06	9.97	6.19	35.43
3maxConstr	300.12	285.72	20.54	0.39	243.90	14.40	7.47	34.91
W3maxConstr	300.13	286.27	19.70	0.41	244.04	13.86	6.87	34.54
rosen	300.09	300.09	0.00	0.80	260.21	0.00	7.47	32.50