University of Southampton

GAUSS-NEWTON-TYPE METHODS FOR BILEVEL OPTIMIZATION

Andrey Tin

A thesis presented for the degree of
Doctor of Philosophy

School of Mathematical Sciences
Faculty of Social Sciences
University of Southampton
UK

November 2020

STATEMENT OF AUTHORSHIP

I, Andrey Tin, declare that the thesis entitled "Gauss-Newton-type methods for bilevel optimization" and the work presented in it are my own contribution and has not been submitted elsewhere for the award of any other degree. I confirm that:

- this work was done wholly while in candidature for a research degree at this University;
- where I have quoted from the work of others, the source is always given;
- with the exception of such quotation, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Part I of this thesis is an original work done by myself while guided by the supervisors. The content of Parts II-IV corresponds to three papers written jointly with my supervisors; i.e., as referred to in the thesis, Part II, III, and IV represent Paper 1, Paper 2, and Paper 3, respectively. For each of these papers, I drafted the initial version with the corresponding numerical results under the guidance of my supervisors. The work [Paper 1] presented in Part II has been accepted for publication with minor corrections by the journal "Computational Optimization and Applications". For this paper, my supervisors' Dr Alain Zemkoho and Prof Joerg Fliege provided guidance for the drafting and advice on specific aspects to focus on for the appropriate numerical experiments. Subsequently, they helped to improve the presentation of the manuscript before submission to the journal.

The other two works [Paper 2, Paper 3] presented in Part III and Part IV, respectively, have not yet been submitted for publication. But they are in the final form for submission and are planned to be submitted within a month from submission of this thesis. Part III partly arose as a natural improvement step from Part II as the Gauss-Newton method requires inverse calculations, which are not possible for many practical examples. However, part of the work there was motivated by [81], identifying classes of bilevel optimization problems that can be partially penalized thanks to the concept of partial calmness introduced there. Dr Zemkoho provided further guidance for drafting this paper and key aspects for experiments. He also helped in improving its presentation.

As for Part IV, it was motivated by the works [52, 84]. In [52], it was shown that bilevel problems with linear structure are partially calm. In [84] Alain Zemkoho and Shenglong Zhou provided a comparison of the KKT-based optimality conditions with LLVF-based optimality conditions for a general class of bilevel problems. Since approaches studied in [84] require partial calmness assumption, it seemed very logical and interesting to study similar comparison for the linear case. Alain Zemkoho suggested what measures to choose for the comparison and advised to build a new test set of examples based on examples from different class of problems. With this I was able to perform the desired comparison in Part IV, as well as performing very extensive experiments in Section 3 in Part IV. All these parts are included in this thesis to maintain completeness of the research contributions and to keep a natural flow of presentation of ideas.

Andrey Tin
November 2020

Approved by main supervisor, Dr Alain Zemkoho.

Signed: [illegible] Date: 16/11/2020

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to EPSRC (Engineering and Physical Sciences Research Council) and Presidential Scholarship programme for making this project possible and letting me to be part of it. Further, I would like to thank my supervisors Dr Alain Zemkoho and Professor Jörg Fliege for the thoughtful comments, recommendations and assistance in every step throughout the process of this work. Your insightful feedback brought my work to a higher level and allowed me to feel comfortable throughout writing this thesis. I am also thankful to the School of Mathematical Sciences and all its members of staff for clear guidance over these years. To conclude, I cannot forget to thank my family and friends for all the unconditional support and understanding over the years of working on this project.

ABSTRACT

This document is designed to provide an overview of the three papers that analyze application of Gauss-Newton-type methods to find solutions of bilevel programming problems. [Paper 1] and [Paper 2] consider the framework for nonlinear bilevel problems, while [Paper 3] extends the analysis to the linear class of bilevel problems. All of the papers are mainly based on the *lower-level value function* (LLVF) reformulation of bilevel programming problems. With some appropriate assumptions *optimality conditions* are stated in the context of this reformulation for nonlinear case in [Paper 1, Paper 2] and for linear case in [Paper 3]. In [Paper 3] LLVF reformulation is compared with Karush-Kuhn-Tucker (KKT) reformulation as the latter approach is very popular to solve linear bilevel problems (BLPs). There we also discuss which assumptions are automatically satisfied for these reformulations for the linear class of bilevel problems. In all three works NCP-functions are used to substitute complementarity constraints and state optimality conditions in the form of a system of equations. Smoothing is then discussed to be the best technique to obtain differentiability of such system. One of the main property of our framework is that introduced system is overdetermined. As Jacobian of such system is non-square, Gauss-Newton-type methods are considered to solve the system. In particular, [Paper 1] examines *Gauss-Newton method* and *Newton method with Moore-Penrose pseudo inverse* in the context of bilevel optimization. Levenberg-Marquardt method for nonlinear bilevel problems is discussed in [Paper 2] together with the detailed discussion on the parameters choice. In [Paper 3] Levenberg-Marquardt method is analyzed for the class of linear bilevel optimization problems. It is worth noting that these well-known methods have not yet been studied for bilevel optimization framework. We prove that all these methods can be well-defined for the introduced formulation of optimality conditions. In numerical part of the papers we present the implementation results of the methods for a good number of problems, choosing different value of penalty parameter λ on the go. The analysis has shown that all these methods perform well, recovering known optimal solutions for most of the tested examples with very small average CPU time required by the algorithms. This demonstrates that not only the methods are valid for the chosen framework, but also that they can compete with popular methods used in bilevel optimization. Further, results of [Paper 1] has shown that Newton method with pseudo inverse could very well be a better option to implement in practice than Gauss-Newton method in its classic formulation. Analysis in [Paper 2] has shown that introduced Levenberg-Marquardt algorithm is very sensitive to the choice of the penalty parameter. The results there lead to some non-trivial suggestions on choosing penalty parameter, which could be useful for anyone dealing with a framework of similar nature. In numerical part of [Paper 3], it has been shown that algorithm performs slightly better for LLVF-based approach than for KKT-based approach for the linear bilevel problems. However, the comparison was only done for 24 linear test problems of relatively small size. We believe that more extensive experiments would be more reliable. For this reason, we create the basis for such comparison by transforming 50 integer examples and 124 binary examples to ordinary linear bilevel problems. Finally, results of implementing Levenberg-Marquardt method for KKT and LLVF reformulations of these problems is presented in the final section of [Paper 3].

CONTENTS

I Summary 9
1 Introduction 9
2 Gauss-Newton method for bilevel optimization 16
3 Levenberg-Marquardt method and penalty parameter selection in bilevel optimization 21
4 Levenberg-Marquardt method for linear bilevel optimization 26
5 Conclusion 30

II Paper 1: Gauss-Newton-type Methods for Bilevel Optimization 33
1 Introduction 33
2 Optimality conditions and equation reformulation 34
3 Gauss-Newton-type methods under strict complementarity 37
4 Smoothing Gauss-Newton method 41
5 Numerical results 48
6 Final comments 53

III Paper 2: Levenberg-Marquardt method and partial exact penalty parameter selection in bilevel optimization 54
1 Introduction 54
2 Levenberg-Marquardt method for bilevel optimization 56
3 Partial exact penalty parameter selection 62
4 Performance comparison under fixed and varying penalty parameter 69
5 Final comments 73

IV Paper 3: Levenberg-Marquardt method for linear bilevel optimization 75
1 Introduction 75
2 Levenberg-Marquardt method for linear Bilevel Optimization 77
3 Numerical study 85
4 Final comments 94

A Objective functions values for transformed integer examples 96
B Objective functions values for transformed binary examples 99
References 106

SUPPORTING MATERIALS

Supplementary materials [Supp1, Supp2, Supp3] with the detailed results of the numerical experiments are available to support the last chapter of each of the papers, [Paper 1, Paper 2, Paper 3].

Part I.
Summary

1 INTRODUCTION

This document provides a summary of the papers [Paper 1, Paper 2, Paper 3]. In all three papers we aim to solve bilevel programming problems of the form

$$\min_{x,y} F(x,y) \ \text{ s.t. } \ G(x,y) \leqslant 0, \ y \in S(x) := \arg\min_{y} \{f(x,y) : \ g(x,y) \leqslant 0\}, \tag{BP}$$

where $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $G : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$ and $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$. This general formulation was analyzed in [Paper 1, Paper 2], while [Paper 3] considered all functions of (BP) to be linear. The first formulation of bilevel optimization problem draws back to Stackelberg in his monograph on market economy in 1934 [64]. It was proposed as a model for a leader-follower game in which two players try to minimize their individual objective functions $F(x,y)$ and $f(x,y)$, respectively, subject to a series of interdependent constraints. Bilevel optimization has a variety of applications in economics, defence, transportation, decision science, business, chemistry, engineering etc. The first mathematical model has been formulated by Bracken and McGill in 1972 [10]. Since then there has been a steady growth in investigations and applications of bilevel optimization. Formulated as a hierarchical game, two decision makers act in this problem. The leader minimizes his objective function subject to conditions partially composed by optimal decisions of the follower. The selection of the leader influences the feasible set and the objective function of the follower's problem, whose reaction has strong impact on the leader's payoff. The bilevel optimization problem is the leader's problem, formulated mathematically using the graph of the solution set of the follower's problem. The implicit structure of the bilevel optimization problems does not allow us to solve them directly. For each decision, x, of the upper-level decision maker, there is reaction set for the lower-level decision maker to optimise over his decision y. It is worth noting that bilevel optimization problems are known to be NP-hard [4, 36]. There are several ways to deal with the complexity of the problem. Earliest solution techniques draw back to implicit function approach and Karush-Kuhn-Tucker (KKT) reformulation. The methods based on implicit function approach rely on the insertion of the lower-level solution function or its approximation in the upper-level objective function; see, e.g., [17, 21] for related algorithms. Authors in [46, 47, 53, 58, 73] test global optimisation techniques as the solution method for (BP) and [50, 75, 76] consider special case of the problem (BP), where upper-level and lower-level constraints depend only on upper-level variable and lower-level variable respectively. We are going to discuss a variety of solution methods in more details below. Nevertheless, interested readers are referred to the books [3, 18] for more extensive review of bilevel optimization and on the approaches to deal with the complexity of this area.

The problem (BP) is the most widely studied form of a bilevel optimization problem. However, this is not the most general one. General bilevel optimization problem can be written as

$$\min_{x} F(x,y) \ \text{ s.t. } \ y \in S(x) := \arg\min_{y} \{f(x,y) : \ g(x,y) \leqslant 0\}. \tag{1.1}$$

This can lead to different lower-level solutions $y_1(x)$, $y_2(x)$, $y_3(x)$ etc. Clearly, if for a choice x there is uncertainty of the choice $y(x)$, it becomes challenging to determine optimal solution of the problem. One way to treat general bilevel optimization problem is to write it as the following set-valued optimization problem

$$\min_{x} F(x, S(x)) := \bigcup_{y \in S(x)} \{F(x,y)\}.$$

If $S(x)$ is single-valued for all $x \in X$ the problem above becomes a usual optimization problem

$$\min\{F(x, S(x)) | x \in X\}$$

The optimality conditions and algorithms based on this approach were presented in [18], with the main difficulty of such approach being implicit nature of the objective. As mentioned before, one of the earliest techniques to solve bilevel optimization problems is the implicit function approach. Such approach deals with solving the problem for a unique solution $y(x)$, that is

$$\min_{x} F(x, y(x)).$$

This allows to solve bilevel optimization problems with certain properties but this approach cannot be applied to all bilevel optimization problems.

The formulation of general bilevel optimization problem is uncertain from the view point of scalar-objective optimization. To overcome this, two approaches are typically considered in the literature – optimistic approach and pessimistic approach. Optimistic approach assumes that there is a cooperation between the leader (upper-level decision maker) and the follower (lower-level decision maker). In this scenario upper-level player is able to influence lower-level decision and the problem is then stated in the following form.

$$\min_{x} \min_{y \in S(x)} F(x, y). \tag{BP$_o$}$$

This way lower-level player is assumed to make the decision $y \in S(x)$ that is best for the leader. The problem (BP$_o$) is known as *optimistic bilevel optimization problem*. From economics viewpoint such problem corresponds to a situation where the follower participates in the profit of the leader. Special case of the optimistic scenario is considered in (BP), where we minimize F with respect to variables x and y. This could be thought of as a simplified version of the optimistic case. Since this is the most investigated formulation in the literature, we refer to (BP) as to "bilevel optimization problem". The overview on this problem is provided in [15]. In terms of global solutions of (BP) and (BP$_o$), these are equivalent. Further, local optimal solution of (BP$_o$) is equivalent to the local optimal solution of (BP). However, the local optimal of (BP) is not necessary a local optimal for (BP$_o$). It is worth noting that it is typical to have $G(x)$ instead of $G(x, y)$ in the literature, so having $G(x, y)$ in (BP) can be thought of as an approximation to this common formulation. However, it will discussed that this has no mathematical harm to consider $G(x, y)$, although this is not typical scenario due to the nature of the bilevel optimization problems.

Lastly, let us consider another scenario, which is somewhat opposite to the one introduced in (BP$_o$). It would not always be possible for the leader to convince the follower to make choice favourable for the leader. Hence, there is big interest in determining the bound of the damage from the worst case scenario. This motivates stating *pessimistic bilevel optimization problem* as follows.

$$\min_{x} \max_{y \in S(x)} F(x, y). \tag{BP$_p$}$$

This scenario assumes that follower plays against the leader and always chooses decision $y \in S(x)$ which is the worst case scenario from the leader's perspective. It is worth mentioning that (BP$_p$) is the special class of minimax problems. Although this class of problems has been widely investigated in the literature, this formulation faces many challenges when $S(x)$ stands for varying sets of solutions to another optimization problem. One major difficulty to handle this problem relates to the fact that the objective function in (BP$_p$) is usually only upper semicontinuous, which makes it hard to detect optimal solutions. Secondly, most of classic techniques for minimax problems cannot be applied to pessimistic bilevel optimization problem. This is due to the fact that the corresponding inner problem $\max_{y \in S(x)} F(x, y(x))$ violates the imposed constraint qualifications (CQs).

Several authors contributed in deriving optimality conditions for bilevel optimization, precisely for (BP). For the case with stable lower-level problem, optimality conditions were obtained by Dempe [16] and Outrata [56] via the implicit function approach. The first result, based on LLVF reformulation, was due to Ye and Zhu [81]. Chen and Florian [12] was one of the first papers to consider optimality conditions based on KKT and LLVF single-level reformulations. In general, there are three major one-level reformulations of the bilevel problem. They are LLVF, KKT and

OPEC (optimization problem with generalized equation constraints) reformulations. For the OPEC lower-level problem is assumed to be convex in the sense that $f(x,.)$ and $g(x,.)$ are convex for all $x \in X$, then S takes the generalized equation form

$$S(x) = \{y \in \mathbb{R}^m | 0 \in \nabla_y f(x,y) + N_{K(x)}(y)\},$$

where $K(x) := \{y \in \mathbb{R}^m | g(x,y) \leqslant 0\}$ and $N_{K(x)}(y)$ denotes the normal cone to $K(x)$ at y. Hence, (BP) can be interpreted as an optimization problem with generalized equation constraint (OPEC):

$$\min_{x,y} \{F(x,y) | x \in X, 0 \in \nabla_y f(x,y) + N_{K(x)}(y)\}.$$

Interested reader is referred to [54, 79] for the problem studied in this form. To avoid making this convexity assumption and to get away from some possibly unwanted behaviour that may occur due to using the optimal value reformulation, Ye and Zhu [82] suggested a combination of the KKT and the optimal value reformulations in order to obtain optimality conditions for the bilevel optimization problem. Concretely, it is assumed in [82] that the KKT conditions of the lower level problem are satisfied without necessarily requiring the convexity of the lower level problem.

Before moving on to the approach studied in this thesis let us first discuss some of the most popular solution methods. One of the earliest techniques to solve bilevel optimization problems draws back to *implicit function technique*. As discussed earlier, this technique considers one $y(x)$ for each decision x. The next class of methods considers *single-level reduction* of bilevel optimization problems. For instance, when the lower level problem is convex and sufficiently regular, it is possible to replace the lower level optimization problem with its Karush-Kuhn-Tucker (KKT) conditions. The KKT conditions appear as Lagrangian and complementarity constraints, and reduce the overall bilevel optimization problem to a single-level constrained optimization problem. The other possibility is the lower level value function (LLVF) reformulation, which also reduces bilevel optimization problem to single-level problem. Next class of the solution techniques is *descent methods*. A descent direction in bilevel optimization leads to decrease in upper level function value while keeping the new point feasible. Given that a point is considered feasible only if it is lower level optimal, finding the descent direction can be quite challenging. Another proposed technique to solve bilevel optimization problems is *penalty function methods*. In penalty function methods the bilevel optimization problem is handled by solving a series of unconstrained optimization problems. The unconstrained problem is generated by adding a penalty term that measures the extent of violation of the constraints. The penalty term often requires a parameter and takes the value zero for feasible points and positive value for infeasible points. In the bilevel context penalty methods are usually incorporated together with single-level transformation of the (BP). Another class of methods incorporates extracting gradient information of the lower-level problem. Arguments common to sensitivity analysis in parametric nonlinear programming can provide information on the gradient and the directional derivative of the optimal solution $y^*(x)$ of the lower-level problem. Using $\nabla y^*(x)$ or the directional derivative $Dy^*(x,d)$, it is possible to derive optimality conditions of (BP). Other techniques to solve bilevel optimization problems include trust region methods, evolutionary algorithms, branch and bound, vertex enumeration and selection function approach. Some of them are designed to solve linear bilevel optimization problems, some of them are considered to be heuristics. Nevertheless, interested reader is referred to [3, 18] for a detailed review of solution methods for bilevel optimization.

We take the approach of reformulating the problem using optimality conditions. It is worth saying that the most popular reformulation is the replacement of lower-level problem by KKT optimality conditions. Such approach is strongly linked to MPECs (mathematical problems with equilibrium constraints). Interested readers are referred to [1, 23, 41] and references therein, for results and methods based on KKT transformation. The first two papers, [Paper 1, Paper 2], aimed to analyze different type of reformulation, known as *lower-level value function reformulation* (LLVF). As discussed in [23], LLVF reformulation has a link to the mentioned KKT approach. These two reformulations were compared in [84] for the general case. We compare them for the linear class of bilevel problems in [Paper 3]. The recent studies of LLVF-based approach for bilevel programs include [46, 47, 53, 58, 73], where authors develop global optimization techniques for (BP) based on LLVF reformulation. Moreover, some works (e.g. [50, 75, 76]) propose algorithms computing stationary points for LLVF-based optimality conditions in the case where the upper-level and lower-level feasible sets do not depend on the lower-level and upper-level variable, respectively. The closest work to the approach considered here is [31], where semi-smooth Newton method for the LLVF reformulation of bilevel programs is studied. Nevertheless, the approach of solving LLVF-based optimality conditions for bilevel optimization is understudied in the literature in comparison to KKT-based approach. Further, it is more standard to introduce assumptions that lead to a square system, making it possible to implement classic Newton method. Extra variables are sometimes introduced to make such system square (e.g. [31]). We make assumptions that lead to a relatively simple system, which has a disadvantage of being overdetermined. To solve overdetermined system we study Gauss-Newton-type methods in theoretical and numerical manner. To our knowledge such approach was not tested in the context of bilevel optimization framework. The main scope of this work is to analyze theoretically and practically validity and effectiveness of such methods for bilevel optimization. We further discuss Newton method with pseudo inverse as a good alternative to implementing Gauss-Newton method in [Paper 1], give detailed overview of the penalty parameter selection in [Paper 2] and compare LLVF-based reformulation with KKT-based reformulation for linear bilevel problems in [Paper 3] in the context of implementing quadratic method, which has also not been studied before.

In [Paper 1, Paper 2] in order to obtain optimality conditions for nonlinear bilevel optimization, we are using the lower-level value function formulation of the problem (BP):

$$\min_{x,y} F(x,y) \;\; \text{s.t.} \;\; G(x,y) \leqslant 0, \;\; g(x,y) \leqslant 0, \;\; f(x,y) \leqslant \varphi(x), \tag{1.2}$$

where the optimal value function is defined by

$$\varphi(x) := \min \{f(x,y) \,|\, g(x,y) \leqslant 0\}. \tag{1.3}$$

The problem with LLVF (lower-level value function) formulation is that the presence of the value function in the problem violates classic constraint qualifications to hold. Because of this we need to assume lower-level regularity and partial calmness condition around optimal solution to be able to state optimality conditions. These assumptions allows us to shift the value function to the upper-level problem as the penalty term with parameter λ as follows.

$$\min_{x,y} F(x,y) + \lambda(f(x,y) - \varphi(x)) \;\; \text{s.t.} \;\; G(x,y) \leqslant 0, \;\; g(x,y) \leqslant 0. \tag{1.4}$$

Further challenge is that value function is not differentiable in the usual sense. Hence, as presented in [Paper 1, Paper 2] we are making some appropriate assumptions to estimate subdifferential of φ and state the desired optimality conditions. This is well-known approach and there is a variety of ways to estimate subdifferential presented in the literature. Depending on the assumptions made, the estimation of the subdifferential would be slightly different, leading to a variety of optimality concepts. One is referred to [20, 23, 24, 81] for different estimations of subdifferential of value function in the context of obtaining optimality conditions for bilevel programs based on lower-level value function reformulation. In fact, due to this property different notions of stationarity are discussed in [22, 83]. For our formulation we use assumption of convexity of the lower-level problem and lower-level regularity at $(\bar{x}, \bar{y})$. The common alternative assumptions include *inner semicontinuity* or *inner semicompactness* of the optimal solution set-valued mapping S, which are briefly discussed in [Paper 1], with the interested reader referred to [20, 23, 24] for the results. The approach that we take leads us to the estimation of $\partial\varphi$ introduced in [20, Theorem 4.2], where subdifferential is estimated as

$$\partial\varphi(\bar{x}) \subseteq \bigcup_{u \in \Lambda(\bar{x},\bar{y})} \left\{ \nabla_x f(\bar{x},\bar{y}) + \sum_{i=1}^{p} u_i \nabla_x g_i(\bar{x},\bar{y}) \right\}, \tag{1.5}$$

where $\Lambda(\bar{x}, y)$ is the set of Lagrange multipliers for lower level, when parameter is fixed at $\bar{x}$. Using the subdifferential of the value function in the form of (1.5), we use the rule of Lagrange inclusion to define stationarity conditions, which are the conditions that should be satisfied for the point to be an optimal solution of the value function penalty formulation of a bilevel program. The optimality conditions are stated in Theorem 2.2 in [Paper 1] and Theorem 2.3 in [Paper 2]. Most of the assumptions required to state optimality conditions are satisfied automatically for the class of problems where all functions defining (BP) are linear. This brings a big advantage to the conditions analyzed in [Paper 3]. The LLVF-based optimality conditions for the linear class of bilevel problems are stated in Theorem 2.3 in [Paper 3], in conjunction with KKT-based optimality conditions. The considered framework allows us to obtain system that is easy to handle. For instance, the conditions introduced in [Paper 1, Paper 2] are easier to handle than the more general case presented in [24, Theorem 3.5] or in [20, Theorem 3.1]. The main reason for this being that the convexity assumption allows us to get rid of the convex hull in the generalized subdifferential of $\varphi(\bar{x})$. The other advantage of conditions in [Paper 1, Paper 2] is that, unlike the system studied in [31], our conditions do not introduce second lower-level variable.

The main idea behind the assumptions made for our framework is that they allow us to get the formulation of $\partial\varphi$ as defined by (1.5). Under full convexity of the lower-level problem, the value function φ is locally Lipschitz continuous around $\bar{x}$. Partial calmness implies the possibility of restating the problem by (1.4). Then we claim the following Lagrangian inclusion as in (3.25) of [20],

$$0 \in \nabla F(\bar{x},\bar{y}) + \lambda\nabla f(\bar{x},\bar{y}) + (\lambda\partial(-\varphi(\bar{x}),0) + \nabla g(\bar{x},\bar{y})^\top u + \nabla G(\bar{x},\bar{y})^\top v, \tag{1.6}$$

where $u \in \mathbb{R}_+{}^p$ and $v \in \mathbb{R}_+{}^q$ are lower-level and upper-level Lagrange multipliers respectively. Due to the convexity of $\partial\varphi(\bar{x})$ we observe that

$$\partial(-\varphi(\bar{x})) \subset -\partial\varphi(\bar{x}). \tag{1.7}$$

With this property we can restate (1.6) as

$$\nabla F(\bar{x},\bar{y}) + \lambda\nabla f(\bar{x},\bar{y}) + \nabla g(\bar{x},\bar{y})^\top u + \nabla G(\bar{x},\bar{y})^\top v \in (\lambda\partial(\varphi(\bar{x}),0). \tag{1.8}$$

Substituting $\partial\varphi(\bar{x})$ defined by (1.5) in (1.8) gives the system of optimality conditions considered in [Paper 1, Paper 2] for nonlinear bilevel problems and in [Paper 3] for the linear case. Comparing to the approach of redefining lower-level problem as KKT conditions, our conditions are easier to handle as they do not involve third derivatives of original functions or extra variables to have square system. To proceed with our framework, Fischer-Burmeister NCP-function is used to substitute all complementarity constraints in the optimality conditions to get rid of inequalities. This enables us to present optimality conditions in the following form of the single system of equations, as presented in Section 2 in [Paper 1, Paper 2].

$$\Upsilon^\lambda(z) := \begin{pmatrix} \nabla_x F(x,y) + \nabla_x g(x,y)^\top(u-\lambda w) + \nabla_x G(x,y)^\top v \\ \nabla_y F(x,y) + \nabla_y g(x,y)^\top(u-\lambda w) + \nabla_y G(x,y)^\top v \\ \nabla_y f(x,y) + \nabla_y g(x,y)^\top w \\ \sqrt{u^2+g(x,y)^2} - u + g(x,y) \\ \sqrt{v^2+G(x,y)^2} - v + G(x,y) \\ \sqrt{w^2+g(x,y)^2} - w + g(x,y) \end{pmatrix} = 0, \tag{1.9}$$

where $z := (x,y,u,v,w)$ with $N := n+m+2p+q$ total number of the variables in the system, and

$$\sqrt{u^2+g(x,y)^2} - u + g(x,y) := \begin{pmatrix} \sqrt{u_1^2+g_1(x,y)^2} - u_1 + g_1(x,y) \\ \vdots \\ \sqrt{u_p^2+g_p(x,y)^2} - u_p + g_p(x,y) \end{pmatrix}. \tag{1.10}$$

$\sqrt{v^2+G(x,y)^2} - v + G(x,y)$ and $\sqrt{w^2+g(x,y)^2} - w + g(x,y)$ are defined as in (1.10). Similar conditions are stated in [Paper 3] for the linear bilevel problems. Newton method is known to be one of the most popular methods to solve nonlinear systems similar to (1.9). For our case classic

Newton method cannot be used as the system (1.9) is clearly overdetermined, having $N + m$ equations and N unknowns. Overdetermined systems have non-square Jacobian, which means that we cannot compute inverse of the Jacobian in the usual sense. Hence, we consider Gauss-Newton-type methods that can be applied for the systems with non-square Jacobian. The methods considered are *Gauss-Newton method* and *Newton method with Moore-Penrose pseudo inverse* in [Paper 1], and *Levenberg-Marquardt method* in [Paper 2, Paper 3]. Approach to implement these methods to solve optimality conditions based on LLVF reformulation of bilevel programs has not been yet considered in the literature. As the methods have not yet been studied in the context of bilevel optimization framework, it could be the case that the methods considered in [Paper 1, Paper 2, Paper 3] are not appropriate to implement for the considered problem. For instance, it could possibly be the case that the methods are not well-defined or cannot converge in the context of solving bilevel optimization problems, especially for the linear case. Hence, one of the main scopes of the papers was to show that methods make sense in the context of our framework both theoretically and practically. The well-definedness of the methods, convergence results and numerical tests are presented in all three papers [Paper 1, Paper 2, Paper 3]. One of the most interesting properties of the considered framework is the dependence of the introduced formulation on the penalty parameter λ. The choice of the penalty parameter and some tests on the behaviour of λ were analyzed in [Paper 2]. This has shown to be different to what one would expect based on the common literature discussions on the theoretical properties of the penalty parameter. Some important advices based on the behaviour of the algorithm with increasing λ were given in [Paper 2], which we hope will become useful for some readers. Further observation was that the method analyzed in [Paper 2] always converges, which was one of the reasons to consider Levenberg-Marquardt method in [Paper 3].

For the numerical tests in [Paper 1, Paper 2], we present results of extensive experiments based on 124 nonlinear problems from BOLIB (Bilevel Optimization Library) [85]. The results are then compared with known solutions of the problems to check if obtained stationary points are actually optimal solutions of the problems or not. Such a number of experiments allows our work to be a good basis for comparison with the other solution methods for bilevel optimization. Further, this allowed us to draw conclusions on the behaviour of the algorithm for different choices of the involved parameters. For the numerical results in [Paper 3], we base the analysis of the performance of the method on 24 linear bilevel problems from BOLIB [85]. We further define transformation of 50 integer and 124 binary examples from [32] into classic linear bilevel programs (BLPS). We then present the results of implementing our method for these transformed examples, hoping that this will provide a good basis for comparison in the future. The results obtained in [Paper 1, Paper 2, Paper 3] show that chosen algorithms perform fairly fast and recover a good amount of solutions for the examples where optimal solutions are known. The overall picture of the work done for this thesis is summarized in Figure 1 below.

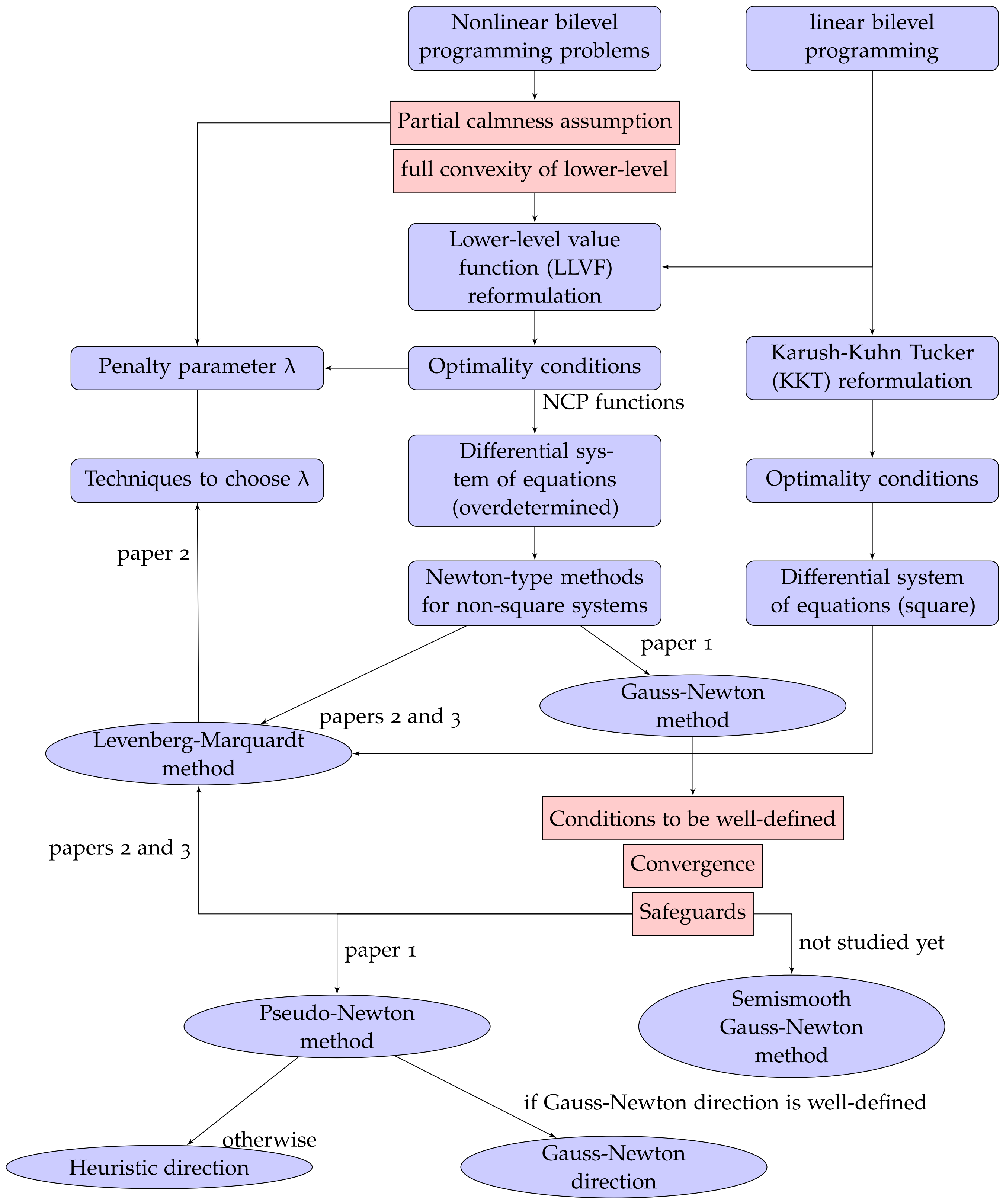

Nonlinear bilevel programming problems
linear bilevel programming
Partial calmness assumption
full convexity of lower-level
Lower-level value function (LLVF) reformulation
Penalty parameter λ
Optimality conditions
Karush-Kuhn Tucker (KKT) reformulation
NCP functions
Techniques to choose λ
Differential system of equations (overdetermined)
Optimality conditions
paper 2
Newton-type methods for non-square systems
Differential system of equations (square)
paper 1
Gauss-Newton method
papers 2 and 3
Levenberg-Marquardt method
Conditions to be well-defined
papers 2 and 3
Convergence
Safeguards
not studied yet
paper 1
Pseudo-Newton method
Semismooth Gauss-Newton method
if Gauss-Newton direction is well-defined
otherwise
Heuristic direction
Gauss-Newton direction

Figure 1: Overview diagram of the work done

The upcoming sections will be structured as follows. We are first going to look at the classic Gauss-Newton method and Newton method with pseudo inverse in Section 2, which is based on [Paper 1]. There we will define the methods, conditions for them to be well-defined for bilevel framework and convergence results. We will then summarize the main outcomes of the experiments. We are then going to look at Levenberg-Marquardt method for bilevel optimization in Section 3, which is based on [Paper 2]. We will briefly introduce the method's nature with the link to the Gauss-Newton method considered in [Paper 1]. We will discuss the main advantages of the Levenberg-Marquardt method and show that it can converge for bilevel framework. This will be followed by the detailed discussion of the choice of penalty parameter λ, as this plays the key role in the definition of optimality conditions and affects performance of the algorithm dramatically, as shown in [Paper 2]. We then present results of implementing the method and discuss the main observations of the experiments. Finally, we are going to look at KKT and LLVF reformulations of linear bilevel problems in Section 4, which is based on [Paper 3]. We will give the overview of optimality conditions and theoretical advantages of the linear framework for both reformulations. We then will discuss the validity of the implementation of Levenberg-Marquardt method to solve the systems of optimality conditions based on KKT and LLVF reformulations. This will be followed by the discussion of convergence results of the algorithm for linear bilevel framework. To finalize this section, reasoning behind the numerical experiments will be given and the main outcomes of the tests will be reviewed.

2 GAUSS-NEWTON METHOD FOR BILEVEL OPTIMIZATION

This section refers to methodology used in [Paper 1], which is based around the methods of the nature similar to the popular Newton method. Generally, methods of this nature are designed to find the solution z of the system $H(z) = 0$ by going through iterative steps of the form

$$z^{k+1} = z^k + td^k, \tag{2.1}$$

where d^k is the direction vector, z is a vector of the variables and t is the step size. More generally, Newton-type methods are minimization methods, where we can consider the problem defined by $H(z) = 0$ to be solved by minimizing the least squares problem i.e.

$$\min\ \Phi(z), \quad \text{where}\ \ \Phi(z) := \frac{1}{2}\left\|H(z)\right\|^2. \tag{2.2}$$

The methods of this nature use information about the gradient of the function to move along *descent direction* towards minimizing the function Φ. Classic Newton method would compute direction in (2.1) by $d^k := -\nabla H(z^k)^{-1}H(z^k)$. There has been recent study testing Newton method for optimality conditions of bilevel programming problems (see [31]) with extensive experiments. The advantage of our system is that we do not need to introduce extra lower-level variable to make the system square (see variable z in [31]). In our case $H(z)$ is defined by (1.9), which has non-square Jacobian $\nabla\Upsilon(z^k) \in \mathbb{R}^{(N+m)\times N}$. For this reason we choose to analyze *Gauss-Newton method* to solve (1.9). The trick that benefits Gauss-Newton method is that the product of $\nabla H(z^k)$ with its transpose results in a square matrix. The step of *Gauss-Newton method*, as defined in [25, 35, 55], is

$$d^k = -(\nabla H(z^k)^\top \nabla H(z^k))^{-1}\nabla H(z^k)^\top H(z^k), \tag{2.3}$$

where clearly $\nabla H(z^k)^\top \nabla H(z^k)$ is a square matrix. For the method taking steps (2.1) it is crucial that direction is descent. For the direction to be descent, d^k should satisfy $\Phi(z^k + td^k) < \Phi(z^k)$ for any iteration k. It is known that unmodified Gauss-Newton algorithm has descent direction. For our case we are dealing with the system $H(z) := \Upsilon^\lambda(z)$ and hence with the merit function $\Phi^\lambda(z) = \frac{1}{2}\left\|\Upsilon^\lambda(z)\right\|^2$. Then for algorithm to be well-defined to solve (1.9) one needs matrix $\nabla\Upsilon^\lambda(z)^\top\nabla\Upsilon^\lambda(z)$ to be invertible, making the direction of the method to be well-defined. This becomes crucial part of the analysis under two scenarios in Section 3 and Section 4 in [Paper 1]. These results demonstrate that the method can be theoretically well-defined for the framework of bilevel programming problems.

The step (2.1) with $t = 1$ is known as a pure step. Due to the novelty of Gauss-Newton method for bilevel programming it was decided to test *pure Gauss-Newton method* to solve (1.9) in [Paper 1], alongside with *pure Newton method with Moore-Penrose pseudo inverse* and Matlab solver *fsolve*. By analysing pure method we get the idea of how appropriate is the method to find optimal solutions for bilevel programming problems. However, it should be noted that technique to control a step size, known as *line search*, could possibly further benefit the method. Line search is designed to choose optimal step length to avoid over-going an optimal solution in the direction d^k and also to globalize the convergence of the methods. Line search technique was out of scope of [Paper 1], but

Levenberg-Marquardt method with line search to solve (1.9) was implemented in [Paper 2] and to solve optimality conditions for the linear case in [Paper 3]. It is worth saying that Levenberg-Marquardt method considered in [Paper 2, Paper 3] falls under the same class of descent direction methods and can be thought of as regularization of Gauss-Newton method, which will be discussed later.

In terms of the nature of Gauss-Newton method it is often referred to as approximation of Newton method. The following link between Gauss-Newton and Newton method is discussed in [55, 69]. The direction of the Newton method to minimize $\Phi(z) := \frac{1}{2}\|H(z)\|^2$ can be written as

$$d^k := -(\nabla H(z_k)^\top \nabla H(z_k) + T(z_k))^{-1} \nabla H(z_k) H(z_k),$$

where $T(z_k) := \sum_{i=1}^{N} H_i(z_k)\nabla^2 H_i(z_k)$ is the term that is omitted in the Gauss-Newton direction (2.3). It is well known that the Gauss-Newton method converges with the same rate as Newton method if for an optimal point $\bar{z}$, the term $T(\bar{z})$ is small enough in comparison to the term $\nabla H(\bar{z})^\top \nabla H(\bar{z})$; see, e.g., [55, 69]. This became the basis of the convergence result in Theorem 3.5 in [Paper 1]. We claim that Gauss-Newton method converges quadratically if $T(\bar{z}) = 0$ and Q-Linearly if $T(\bar{z})$ is small relative to $\nabla H(\bar{z})^\top \nabla H(\bar{z})$. We note that to satisfy $T(\bar{z}) = 0$ one would either need to be able to solve the system exactly (i.e. $H_i(\bar{z}) = 0$ for all i), or have very sparse Hessian $\nabla^2 H_i(\bar{z})$. Such properties can be satisfied for small residuals problems and for the problems that are not too nonlinear, as discussed in [69].

This is interesting to see if the method can be well-defined and converge as Gauss-Newton method to solve (1.9) was not tested before. In fact, the method was not tested for bilevel optimization at all, which makes the work in [Paper 1] unique. As discussed, methods of this nature can deal with non-square systems, which motivates implementation of the method for considered optimality conditions. One of the challenges of the method is that Jacobian of (1.9) needs to be defined everywhere to be able to calculate direction d^k. However, Fischer-Burmeister functions in the last lines of (1.9) are not differentiable at $(0, 0)$ in the usual sense, as this leads to the division $0/0$. One way to deal with this problem is by assuming that arguments of NCP-functions cannot simultaneously equal to zero, which is known as *strict complementarity assumption*. This scenario was considered in Section 3 in [Paper 1]. There we have shown that non-singularity condition of $\nabla\Upsilon^\lambda(z)^\top \nabla\Upsilon^\lambda(z)$ may hold theoretically ([Paper 1, Theorem 3.3]) and practically ([Paper 1, Example 3.4]). This was followed by the convergence result of the method ([Paper 1, Theorem 3.5]). To show non-singularity condition we have shown that columns of $\nabla\Upsilon^\lambda$ are linearly independent, which is sufficient due to the following result.

Lemma 2.1. *For an arbitrary matrix $A \in \mathbb{R}^{(N+m)\times N}$, the matrix $A^\top A$ has full rank if and only if the columns of A are linearly independent.*

It was shown in part (c) of the proof of [40, Theorem 7.2.10] that a Gram matrix, defined by the product of the original matrix with its transpose, is of the same rank as the original matrix. Recalling that $\nabla\Upsilon^\lambda(z) \in \mathbb{R}^{(N+m)\times N}$, the lemma above then shows that linear independence of the columns of $\nabla\Upsilon^\lambda(z)$ is a sufficient condition for invertibility of $\nabla\Upsilon^\lambda(z)^\top \nabla\Upsilon^\lambda(z)$. We then state linear independence of the columns of Jacobian matrix under mild assumptions in Theorem 3.3 in [Paper 1], which is the key theorem of Section 3 of [Paper 1]. This result is important as it shows that the method can indeed be well-defined for our framework. To claim convergence of Gauss-Newton method for bilevel optimization in Theorem 3.5 [Paper 1] we assume that $\nabla\Upsilon^\lambda$ is invertible for each step of the Gauss-Newton method and that elements of $\nabla\Upsilon^\lambda(z)$ are Lipschitz continuous. The result that well-definiteness of the Newton-type methods guarantees convergence is used by many authors to state convergence of Gauss-Newton method (e.g. [2, 13, 49]). Some of the authors, e.g. [63], use the geometric approach with the notion of curvature to prove convergence of Gauss-Newton method. Another way to prove convergence is demonstrated in [28, 38], where the majorant condition is assumed to ensure convergence. Typically, these convergence results depend on so known *Robinson condition*, which implies convergence assumptions made in Theorem 3.5 in [Paper 1], meaning that convergence result discussed in [Paper 1] is at least as good as the mentioned ones.

One of the main outcomes of Section 3 of [Paper 1] is that Gauss-Newton method can be well-defined for the bilevel optimization framework. However, we are not guaranteed that the matrix $\nabla\Upsilon^\lambda(z)^\top\nabla\Upsilon^\lambda(z)$ would always be nonsingular in practice. For the case when the direction for the Gauss-Newton method could not be calculated, we are motivated to look at the alternative method of the similar nature, which we have chosen to be *Newton method with generalized inverse*. There have been studies (e.g. [57]) on the approach of using Newton method with different notions of pseudo inverse to calculate direction. In this paper we decided to test Newton method with *Moore-Penrose pseudo inverse* as this notion of generalized inverse is not widely used in the literature in the context of applying Newton-type methods. As stated in [39], the standard way to determine Moore-Penrose pseudo inverse is through checking the following conditions.

Definition 1. (Pseudo inverse) Let $A \in \mathbf{R}^{m\times n}$ be an arbitrary matrix, $A^+ \in \mathbf{R}^{n\times m}$ is the pseudo inverse of the matrix A if it satisfies the four *Moore-Penrose* conditions:

$$1.\ AA^+A = A \tag{2.4}$$

$$2.\ A^+AA^+ = A^+ \tag{2.5}$$

$$3.\ (AA^+)^\top = AA^+ \tag{2.6}$$

$$4.\ (A^+A)^\top = A^+A \tag{2.7}$$

The further property states that when a real matrix A has *linearly independent columns*, pseudo inverse A^+ can be computed as

$$A^+ = (A^\top A)^{-1}A^\top. \tag{2.8}$$

Such pseudo inverse can be treated as left inverse of A as $A^+A = (A^\top A)^{-1}A^\top A = I$. With the standard SVD formulation (see Section 5.4.5 in [39]), pseudo inverse $\Sigma^+ \in \mathbb{R}^{n\times m}$ of Σ is defined by

$$\Sigma^+ = diag(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, ..., \frac{1}{\sigma_r}, 0, ..., 0), \tag{2.9}$$

where $r = rank(A)$. Let us define $U \in \mathbb{R}^{(N+m)\times(N+m)}$ and $V \in \mathbb{R}^{N\times N}$ to be orthogonal matrices. We can then write the pseudo inverse A^+ of A by

$$A^+ = V\Sigma^+U^\top,$$

which satisfies (2.4)-(2.7). Clearly, $\exists\ \nabla\Upsilon^\lambda(z)^+$ such that (2.4)-(2.7) hold for Jacobian of (1.9). An iteration of the Newton method with pseudo inverse to solve (1.9) is then stated as

$$z^{k+1} = z^k - \nabla\Upsilon(z^k)^+\Upsilon(z^k). \tag{2.10}$$

We denote (2.10) as an iteration of *Pseudo-Newton method*. The convergence of Newton method with generalized inverse is demonstrated in [37] where the method is discussed in the context of *rank forcing* of a matrix. Making assumption that $\nabla\Upsilon^\lambda(z_k)$ has linearly independent columns and using the property (2.8), we can write the pseudo inverse of the Jacobian of $\Upsilon^\lambda(z_k)$ by

$$(\nabla\Upsilon^\lambda(z_k))^+ = (\nabla\Upsilon^\lambda(z_k)^\top\nabla\Upsilon^\lambda(z_k))^{-1}\nabla\Upsilon^\lambda(z_k)^\top.$$

Then Pseudo-Newton method for (1.9) would take the direction defined by

$$d^k = -(\nabla\Upsilon^\lambda(z_k)^\top\nabla\Upsilon^\lambda(z_k))^{-1}\nabla\Upsilon^\lambda(z_k)^\top\Upsilon^\lambda(z_k), \tag{2.11}$$

which is exactly the *Gauss-Newton* iteration defined by (2.3) with $H(z) := \Upsilon^\lambda(z)$. Interestingly, it turns out that whenever Gauss-Newton is well-defined the direction of Gauss-Newton and Newton method with Moore Penrose pseudo inverse are equivalent. However, there is a number of questions regarding this equivalence. Would this hold in practice? Which computation of the direction is faster? Would the direction based on pseudo-inverse lead to good solutions when Gauss-Newton direction is not well-defined? To answer these questions it was decided to treat Gauss-Newton method and Pseudo-Newton method as two separate methods in [Paper 1]. The big theoretical advantage of Pseudo-Newton method is that it should converge to a number for all problems as Moore-Penrose pseudo-inverse is always well-defined. It was further observed that some papers (e.g. [38, 63]) define Gauss-Newton method to be the Newton method with Moore-Penrose pseudo inverse, which does not seem accurate for several reasons. Firstly, the direction for the methods is computed in a different numerical manner, i.e. the Gauss-Newton step is computed by inverting the product of the Jacobian with its transpose, whenever Moore-Penrose pseudo inverse is normally calculated using SVD of the matrix. Secondly, these methods only have the same direction whenever the Jacobian matrix has full column rank, which cannot be guaranteed in practice. For these reasons methods are treated separately in [Paper 1] with the conjecture that the methods should produce the same solutions when Jacobian is full rank. For the case when Gauss-Newton direction is ill-conditioned Newton method with Moore-Penrose pseudo inverse is expected to converge to a number, as stated in [34]. In the numerical study, the aim was to test the conjecture and see the behaviour of both methods in terms of quality of produced solutions and computation time.

Although we have shown that the method could be well-defined and converge under reasonable assumptions, the scenario considered in Section 3 of [Paper 1] has a big disadvantage. As discussed in [Paper 1], strict complementarity is a strong assumption to make. For each family of Lagrange multipliers, it leads to solving a problem for two cases, namely for $\{u, g(x,y) | u = 0, g(x,y) < 0\}$ and $\{u, g(x,y) | u > 0, g(x,y) = 0\}$; whereas in reality we have the third case $\{u, g(x,y) | u = 0, g(x,y) = 0\}$. This approach could cause problems if the assumption does not hold in practice for any of the iterations when using Gauss-Newton method. This has been practically verified that implementing Gauss-Newton algorithm under this scenario would lead to divergent solutions for almost half of the examples tested in [Paper 1]. If one wants to avoid the *strict complementarity* assumption, another option to deal with differentiability is to use *smoothing technique* for Fischer-Burmeister function, which is analyzed in Section 4 in [Paper 1]. The technique uses a trick of adding perturbation $2\mu > 0$ under the square root of Fischer-Burmeister functions in the system (1.9), where μ is a vector of appropriate dimensions with sufficiently small positive elements. Classically, elements of μ are defined to be a sequence $\mu_k \downarrow 0$, such that $\Upsilon^\lambda_\mu(z)$ converges to the original system $\Upsilon^\lambda(z)$. In general, a function $G(z,\epsilon) : \mathbb{R}^n \to \mathbb{R}^m$ is a smoothing function for $H(z) : \mathbb{R}^n \to \mathbb{R}^m$ if $\|H(z) - G(z,\epsilon)\| \to 0$ as $\epsilon \downarrow 0$ (see [71]). This enables us to solve $H(z) = 0$ using Gauss-Newton algorithm with the step

$$z^{k+1} = z^k - (\nabla G(z^k,\mu^k)^\top \nabla G(z^k,\mu^k))^{-1} \nabla G(z^k,\mu^k)^\top H(z^k). \tag{2.12}$$

The main point of the smoothing technique is that $(0,0)$ falls out of the range of the approximated Fischer-Burmeister function. As the value of μ decreases we get closer to our original Fischer-Burmeister function. Smoothing technique is typically used to get rid of the problem with differentiability of non-smooth functions. The smoothed system Υ^λ_μ was presented in Section 4 in [Paper 1]. Since all functions in the system $\Upsilon^\lambda_\mu(z)$ are continuous, it is easy to see that

$$\lim_{\mu_k \to 0} \Upsilon^\lambda_{\mu_k}(z^k) = \Upsilon^\lambda(z^k).$$

Then for $k \to \infty$ the sequence of solutions $\{z^k\}$ generated by solving the smoothed system $\Upsilon^\lambda_\mu(z) = 0$ converges to the solution z^* for the system of equations (1.9). Hence, finding $z^* = (x^*, y^*, u^*, v^*, w^*)$ satisfying $\Upsilon^\lambda_{\mu_k}(z^*) = 0$, we get (x^*, y^*) to be a stationary point of (BP), given k is sufficiently large. The benefit of smoothing technique is that, having differentiable system of equations, we can now introduce the step of Gauss-Newton method for the smoothed optimality conditions $\Upsilon^\lambda_\mu(z)$ as

$$z^{k+1} = z^k - (\nabla \Upsilon^\lambda_{\mu^k}(z^k)^\top \nabla \Upsilon^\lambda_{\mu^k}(z^k))^{-1} \nabla \Upsilon^\lambda_{\mu^k}(z^k)^\top \Upsilon^\lambda_\mu(z^k).$$

Clearly, the direction of the method considered in Section 4 of [Paper 1] requires invertibility of $\nabla \Upsilon^\lambda_\mu(z)^\top \nabla \Upsilon^\lambda_\mu(z)$. With the use of smoothing technique we do not need to make strict complementarity assumption. However, this assumption was also used to show that the matrix $\nabla \Upsilon^\lambda(z)$ has full column rank in general. For this reason, we needed some other technique to show that $\nabla \Upsilon^\lambda_\mu(z)$ has full column rank to ensure non-singularity of $\nabla \Upsilon^\lambda_\mu(z)^\top \nabla \Upsilon^\lambda_\mu(z)$. Two scenarios were considered in Section 4 in [Paper 1]. Under the first scenario we assumed that $\nabla^2 L^\lambda(\bar{z})$ is positive definite and penalty parameter λ satisfies $0 < \lambda < \frac{\kappa_j^\mu}{\theta_j^\mu}\frac{\tau_j^\mu}{\gamma_j^\mu}$. For the second scenario it was assumed that each row of $\left[\nabla^2 L^\lambda(z)^T \quad \nabla(\nabla_y \mathcal{L}(z))^T \quad \nabla g(x,y)^T \quad \nabla G(x,y)^T\right]$ is a nonzero vector and the diagonal elements of the matrix $\nabla \Upsilon_\mu^\lambda(z)^T \nabla \Upsilon_\mu^\lambda(z)$ dominate the other terms row-wise. It has been shown that the assumptions made in both theorems can hold for an example of bilevel problem. Showing these two sufficient scenarios verifies that the method can be well-defined for the introduced framework. It is worth noting that scenarios are sufficient but not necessary, meaning that $\nabla \Upsilon_\mu^\lambda(z)^\top \nabla \Upsilon_\mu^\lambda(z)$ could possibly be non-singular even if the introduced assumptions are not satisfied.

In terms of convergence of the smoothed version of Gauss-Newton algorithm, it is enough to know that we satisfy *Jacobian consistency property* for $\Upsilon_\mu^\lambda(z)$ and semi-smoothness of $\Upsilon^\lambda(z)$ according to [43]. In the end of Section 4 of [Paper 1] Jacobian consistency property was discussed. This property demonstrates that, whenever the smoothing parameter μ is getting close to zero, the Jacobian $\nabla \Upsilon_\mu^\lambda(z)$ converges to the generalized derivative $W \in \partial \Upsilon^\lambda(z)$. Jacobian consistency property has been shown to hold for our framework in [Paper 1], referring to [43] for the technique. Further, as the proof of Theorem 3.5 in [Paper 1] did not involve specifying the structure of Fischer-Burmeister function, it could be considered to be a general convergence result of Gauss-Newton method. Hence, same convergence result holds for the smoothed algorithm with the only difference that we ensure differentiability by smoothing instead of assuming strict complementarity.

It is worth saying that another option to deal with non-differentiability of the Fischer-Burmeister NCP-functions would be to introduce the *generalized Jacobian* of the system. This would mean that whenever we arrive at the point where the function is not differentiable generalized derivative of that function would be used. In this case, one would use semi-smooth Gauss-Newton method with *generalized Jacobian* defined as

$$\nabla \Upsilon^\lambda(z^k) = W_k, \tag{2.13}$$

where $W_k \in \partial \Upsilon^\lambda(z^k)$. Under this scenario smoothing would not be required and we would be dealing with generalized Jacobian of Υ^λ used whenever arguments of any of the NCP-functions are $(0,0)$. This approach is the alternative approach to the assumption of strict complementarity and smoothing technique. Although, such approach is quite popular to use for Newton method (e.g. [72, 31], we have not analyzed it in [Paper 1] as one would need a separate convergence result for the generalized Jacobian scenario. Otherwise, such method would be a heuristic. Hence, we stick with a smoothing technique as the most robust and sensible way to maintain differentiability of (1.9). It was decided to compare the method to the smoothed Pseudo-Newton method, with all discussed properties of the latter method holding for the smoothed scenario. Smoothed Pseudo-Newton method would then take the step (2.1) with the direction defined by $d^k = -\nabla \Upsilon_\mu(z^k)^+ \Upsilon_\mu(z^k)$. For implementation built-in function *pinv(Υ^λ)* in MATLAB was used to calculate Moore-Penrose pseudo inverse of a matrix Υ^λ. Before doing experiments, we could not be entirely sure if such implementation would result in Gauss-Newton method for invertible $\nabla \Upsilon^\lambda(z_k)^\top \nabla \Upsilon^\lambda(z_k)$, and also whether or not it would solve the problem in some way for non-invertible $\nabla \Upsilon^\lambda(z_k)^\top \nabla \Upsilon^\lambda(z_k)$. This further motivated the comparison of two methods implemented separately.

The results of implementing Gauss-Newton method and Pseudo-Newton method to solve 124 instances of nonlinear bilevel problems from BOLIB (Bilevel Optimization Library [85]) is presented in Section 5 of [Paper 1]. Such extensive experiments are not common in the literature and we hope that our experiments will provide a good basis for comparison for other authors in the field. Both methods has shown to perform very well to solve bilevel problems from the test set. However, we still observed some divergence of Gauss-Newton method for a few examples, typically for one or two values of λ. Sensitivity to the starting point, which was expected for the pure method, brings another disadvantage of the method. Finally, the choice of parameter λ remains a heuristic, although choosing several fixed values of λ of different magnitudes seemed to perform well in the sense that solutions were recovered for majority of examples for at least one such value of λ. This motivated using line search technique and analyzing penalty parameter λ in [Paper 2].

The comparison of the solutions to known ones demonstrated that the methods considered in [Paper 1] are appropriate to be used for bilevel programs, recovering optimal solutions for most of the tested problems. The numerical results verified that for the well-defined problems (without ill conditions for Gauss-Newton direction), Gauss-Newton method and Pseudo-Newton method produced very similar results by the means of all performance measures. Further, Pseudo-Newton method always converged to a number, while Gauss-Newton method diverged for some values of λ for some examples. Finally, the time taken by Pseudo-Newton algorithm was not worse than the time used by Gauss-Newton algorithm. Studying methods separately, allowed us to build performance profiles as the measure to compare the methods, where we have seen slightly better performance of Pseudo-Newton method. Based on the results observed in [Paper 1], implementing Pseudo-Newton method seems more attractive than Gauss-Newton method in its classic form. At least this was clearly observed for the test set of 124 nonlinear bilevel problems. This suggestion is likely to hold for other classes of the problems, as Pseudo-Newton method always converges and is equivalent to Gauss-Newton method whenever the latter method converges. Further advantage of Pseudo-Newton method is that it could also be used for square systems. For Square systems with nonsingular Jacobian, Pseudo-Newton method would take exactly the Newton direction as in this case $H^{+}(z) = H^{-1}(z)$ for some $H \in \mathbb{R}^{n \times n}$. Similarly to our study, Pseudo-Newton method would always converge to a number, which is possibly a reasonable solution, even when H^{-1} is not well-defined.

Summarizing [Paper 1], the optimality conditions were presented and formulated as a system of equations (1.9). The aim then was to test Gauss-Newton to solve the system as the system is overdetermined and the method has not yet been tested in considered framework. To obtain differentiability of the system two scenarios were considered, where we discussed that smoothing scenario is a clear winner to implement in practice. Alongside with standard Gauss-Newton method, Newton method with pseudo inverse (Pseudo-Newton method) was introduced. This was verified theoretically and practically that the methods are equivalent if the Gauss-Newton method is well-defined. In the experiments we have shown that solving introduced LLVF-based optimality conditions is a valid approach. Gauss-Newton method and Pseudo-Newton method performed better than *fsolve*. We finally discussed the advantages of implementing Pseudo-Newton method in comparison to classic Gauss-Newton method. For the further study of the topic, semi-smooth Gauss-Newton method and Levenberg-Marquardt method could be considered as regularized versions of Gauss-Newton method. Line search technique could also be analyzed to improve convergence and robustness of the method.

3 LEVENBERG-MARQUARDT METHOD AND PENALTY PARAMETER SELECTION IN BILEVEL OPTIMIZATION

In this section we are going to look at the *Levenberg-Marquardt method* analyzed in [Paper 2] together with the selection of partial exact penalty parameter λ considered there. As before, we aim to solve (BP) and to do so we consider LLVF-based optimality conditions (1.9). Similarly to Gauss-Newton method, Levenberg-Marquardt method is typically discussed in the context of solving least squares problem (2.2) (see e.g. [25, 55, 69]). Levenberg-Marquardt method to solve a problem $H(z) = 0$ falls under the class of descent direction methods with the step defined by (2.1) and direction given by

$$d^k = -\left(\nabla H(z^k)^\top \nabla H(z^k) + \alpha(z^k) I\right)^{-1} \nabla H(z^k)^\top H(z^k). \tag{3.1}$$

where $\alpha(z^k) > 0$ is the *Levenberg-Marquardt parameter* and I is the identity matrix of appropriate dimension. Levenberg-Marquardt method could be thought of as a regularization of the Gauss-Newton method. Remind ourselves that the direction of the Gauss-Newton method is given by (2.3) and requires non-singularity of $\nabla H(z^k)^\top \nabla H(z^k)$. The benefit of the Levenberg-Marquardt method is that adding positive term to the main diagonal of $\nabla H(z^k)^T \nabla H(z^k)$ allows us to avoid the possible problem with singularity of the matrix being inverted. The way it works is as follows. The matrix $\nabla H(z^k)^\top \nabla H(z^k)$ is clearly positive semidefinite as for any vector $d \neq 0$ of appropriate dimension we have

$$d^T \nabla H(z^k)^T \nabla H(z^k) d = \left\|\nabla H(z^k) d\right\|^2.$$

Adding a positive perturbation on main diagonal ensures that such matrix becomes positive definite. For our case we aim to solve $H(z) := \Upsilon^\lambda(z) = 0$ as defined by (1.9). Given $\alpha(z^k) > 0$, we claim that $(\nabla\Upsilon^\lambda(z^k)^T\nabla\Upsilon^\lambda(z^k) + \alpha(z^k)I)$ is positive definite and hence nonsingular. The obvious benefit of Levenberg-Marquardt method is that it is more robust than Gauss-Newton method, as direction of the method is always well-defined. However, adding perturbation results in having the additional challenge of calculating the parameter $\alpha(z^k)$. Further, it turns out that the choice of $\alpha(z^k)$ affects convergence of the method.

In some sense Levenberg-Marquardt method in [Paper 2] analyzes the extension of the Gauss-Newton method considered in [Paper 1], with the safeguard preventing non-invertibility issue. Although the method is very popular in optimization, it has not been tested to solve bilevel optimization problems. This brings the novelty of such approach and follows logically to be the next step of analysis done in [Paper 1]. To deal with differentiability of NCP-functions only smoothing technique is considered in [Paper 2] as strict complementarity has shown to be too strong assumption to often hold in practice. Levenberg-Marquardt method has been used by many authors to solve least squares problems of different nature. For the selection of the LM parameter α_k, there are various options that have been considered in the literature; see [6, 26, 27, 44, 78]. Based on the choice of α_k various convergence results have been presented in the literature for Levenberg-Marquardt method in the context of solving a problem $H(z) = 0$. For instance, authors in [25] show quadratic convergence of the method if the Jacobian is nonsingular at the solution point and α_k is chosen with the order $O(\|\nabla H(z^k)^T H(z^k)\|)$. Yamashita and Fukushima, [78], chose the LM parameter $\alpha_k = \|H(z^k)\|^2$ and showed that under the *local error bound condition* the Levenberg-Marquardt method converges quadratically to the solution set of $H(z) = 0$. Fan and Yuan in [27] considered the choice $\alpha_k = \|H(z^k)\|^\eta$ with $\eta \in (0, 2]$ and showed that under the same conditions, the Levenberg-Marquardt method converges quadratically to some solution of $H(z) = 0$ when $\eta \in [1, 2]$ and super-linearly when $\eta \in (0, 1)$. In [26] Fan and Pan combine these ideas to present more complex choice of stepsize α_k, claiming potential global convergence. Such choice of stepsize α has some implementation sensitivity and we cannot be sure that it would behave well for our framework. For this reason, to implement Levenberg-Marquardt method to solve (1.9) we decided to stick with the choice $\alpha_k := \|\Upsilon^\lambda(z^k)\|^\eta$, where $\eta \in [1, 2]$. The convergence followed from [27] and with the observed behaviour of the method seemed to fit well for the framework. Apart from the choice of the parameter α_k, convergence of Levenberg-Marquardt requires further conditions, mainly *Error Bound condition*. Error bound condition (Assumption 2 in the convergence theorem in [Paper 2]) is the standard assumption required to show convergence of Levenberg-Marquardt algorithm. Yamashita and Fukushima, [78], discuss that error bound assumption is weaker than Jacobian non-singularity assumption. As stated there, non-singularity of the Jacobian matrix implies that solution is isolated and error bound condition holds. Remind ourselves that non-singularity of $\nabla\Upsilon^\lambda_\mu(\bar{z})$ is the condition we needed for the Gauss-Newton method to be well-defined. As $(\nabla\Upsilon^\lambda_\mu(z^k)^T\nabla\Upsilon^\lambda_\mu(z^k) + \alpha(z^k)I)$ is positive definite matrix, we do not require full column rank of $\nabla\Upsilon^\lambda_\mu(z)$ for well-definedness of the Levenberg-Marquardt method. This shows that Levenberg-Marquardt method converges under weaker assumptions than Gauss-Newton method. Hence, not only Levenberg-Marquardt direction is always well-defined, the method also requires weaker convergence assumptions than Gauss-Newton method.

To solve (1.9) we implement Levenberg-Marquardt method with *line search*. In terms of the choice of Levenberg-Marquardt parameter we use $\alpha := \|\Upsilon^\lambda_\mu\|$. Although the choice $\alpha := \|\Upsilon^\lambda_\mu\|^\eta$ with $\eta \in (1, 2]$ is widely used by many authors (e.g. [30, 44, 78]) we demonstrated that with $\alpha := \|\Upsilon^\lambda_\mu\|$ (initially introduced in [45]), algorithm defined in [Paper 2] has a better performance, based on the observed behaviour of the algorithm. The algorithm with $\eta > 1$ would typically diverge after 80-120 iterations in the context of solving 124 test problems. It is interesting as Fan and Yuan [27] discussed that the choice $\alpha_k := \|\Upsilon^\lambda_\mu(z^k)\|^2$ has some potential implementation problems, which was likely the case for our problem. There authors state that when the sequence is close to the solution set, $\alpha_k := \|\Upsilon^\lambda_\mu(z^k)\|^2$ could become smaller than the machine precision and lose its role as a result. On the other hand, when the sequence is far away from the solution set, $\alpha_k := \|\Upsilon^\lambda_\mu(z^k)\|^2$ may be very large, making movement to the solution set to be very slow. Our work further verifies the possibilities of such problems with $\alpha_k := \|\Upsilon^\lambda(z^k)\|^2$ and brings a suggestion that in the context of solving bilevel problems $\alpha_k := \|\Upsilon^\lambda(z^k)\|$ performs much better. The quadratic convergence for Levenberg-Marquardt method with Armijo condition line search has been shown in [27] for any choice of the parameter $\alpha_k := \|\Upsilon^\lambda(z^k)\|^\eta$ with $\eta \in [1, 2]$. Clearly, the result holds for the choice $\eta = 1$, as presented in Section 2 in [Paper 2].

In [Paper 2] we studied parameters selection for the Levenberg-Marquardt method to fit the structure of bilevel programming problems and to solve (1.9) efficiently. This involved selecting penalty parameter λ, smoothing parameter μ, Levenberg-Marquardt parameter α and stopping criteria. For selection of the penalty parameter, two approaches were considered: keeping λ as fixed constant with the test values $\lambda \in \{10^6, 10^5, \ldots, 10^{-2}\}$ or varying λ as increasing sequence $\lambda_k := 0.5 \times 1.05^k$, where k is the number of iterations. The smoothing parameter μ_k was taken as decreasing sequence $\mu_k := 0.001/(1.5^k)$, which is in line to what it should be theoretically. As discussed earlier, Levenberg-Marquardt parameter was chosen to be $\alpha_k := \|\Upsilon^\lambda(z^k)\|$. The additional stopping criteria was needed to ensure that algorithm is not running for too long. This is particularly important for the case with varying λ due to the danger of ill-conditioning, which could occur if λ is too large. Further, we observed the pattern that we recover solution earlier than algorithm stops. This appears due to the nature of the overdetermined system. Quite often we cannot solve the system with the precision $\epsilon = 10^{-5}$, as one would prefer. We do not know beforehand the tolerance with which we can solve (1.9) for each example. To avoid algorithm running for too long and to prevent λ to become too large, we impose additional stopping criterion, as defined in the end of Section 2 in [Paper 2]. The motivation behind this stopping criteria was the observation of the typical behaviour of the algorithm with varying λ, defined as $\lambda := 0.5 \times 1.05^k$. The behaviour of the algorithm for majority of the examples had the following pattern.

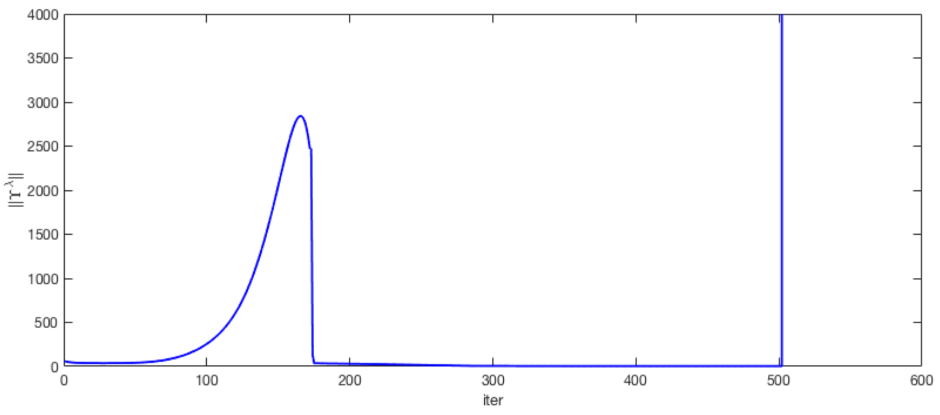


Figure 2: Typical behaviour of the algorithm in [Paper 2]

Typically, for our problems we would get a good solution for small value of λ, then there would be a jump in the value of the $\text{Error}_k := \|\Upsilon^\lambda(z^k)\|$ for some iteration k and we would then come back to a good solution at some point later, clearly with the larger value of λ. Then the solution would be retained for a good number of iterations (e.g. 200-500 iterations in Figure 2), but at some point Error would blow up without coming back to reasonable values later on. It was discussed in [Paper 2] that it could be the case that the system becomes ill-conditioned due to the large value of the penalty parameter, which is known to be a possible issue with penalization methods. For almost all of the examples we observe that after 100-150 iterations we obtain the value reasonably close to the solution. Further, a quick check has shown that ill-conditioning issue typically takes place after 500 iterations for majority of the problems. As discussed in [Paper 2] these observations motivated additional stopping criteria introduced there. It is worth noting that stopping criteria incorporated in the actual experiments in [Paper 2] worked well as a safeguard against ill behaviour. Ill behaviour was observed to affect results for 3/124 (2.4%) of the problems with the incorporated stopping criteria. If we relax all stopping criteria and run the algorithm for $1,000$ iterations for each example, ill behaviour is observed for 91/124 (73.4%) problems.

The main challenge with selecting parameters for the algorithm was the choice of the penalty parameter λ. This parameter clearly has a big impact on the system (1.9). As suggested by Lemma 2.2 in [Paper 2] large values of λ should theoretically be good to recover solutions. As opposed to that, the authors in [33, 51, 59, 74] suggest that too large values of penalty parameter could lead to ill-condition (so known zig-zagging issue). Hence, it becomes a tricky question what is the best choice of the penalty parameter λ. Firstly, in Section 3 of [Paper 3] we test possibility of ill-behaviour issue for all examples in the test set of 124 nonlinear problems from BOLIB [85]. Since we are only increasing λ throughout iterations it is likely, but not definitely, that the explanation of the ill behaviour is the mentioned issue of ill conditioning of the penalty functions. Ill condition refers to one eigenvalue of the Hessian being much larger than the other eigenvalue, which affects the curvature in the negative way for gradient methods [62]. To proceed, we reran the algorithm for all examples for $1,000$ iterations with no stopping criteria, letting λ vary indefinitely. We then looked at which iteration algorithm blows up and recorded the value of λ there. We Denoted the first value of λ for which ill behaviour was observed for each example by λ_{ill}. From the analysis of this, we have seen that ill behaviour typically occurs after about 500 iterations (where $\lambda_{ill} \approx 10^{10}$). For 34/124 of the tested problems ill-condition was not observed under the scope of 1000 iterations. As discussed in [Paper 2], it could potentially be that the parameter λ does not get large enough after $1,000$ iterations to cause problems for these examples. It could also be that the eigenvalues of the Hessian are not affected by large values of λ for these examples. It was observed that for more than half of such problems key elements of the Hessian vanish due to linearity of g w.r.t. (x, y)-variables. Finally, for most of the tested problems (72/124) ill-behaviour was observed for $10^9 < \lambda < 10^{11}$. Algorithm has shown to behave well for the values of penalty parameter $\lambda < 10^9$ with only 7/124 examples demonstrating ill behaviour for such λ. This makes the choice of very large values of λ not attractive at all. We further recommend that for our method $\lambda \leqslant 10^7$ is very safe choice, which supports the choices of fixed values of λ considered in the experiments in [Paper 2].

We then move on to discuss which magnitudes of λ seem to perform better for our method. There we extend the choices of fixed values $\lambda \in \{10^2, 10^1, 10^0, 10^{-1}, 10^{-2}\}$ considered in [Paper 1] to fixed values $\lambda \in \{10^6, 10^5, 10^4, 10^3, 10^2, 10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}\}$. We further analyze the case with varying λ defined by $\lambda := 0.5 \times 1.05^k$, where k is the number of iterations. As discussed in [Paper 2], such choice of λ comes from the danger that algorithm diverges for λ that grows too aggressively. This choice has been observed to perform well for our framework, with the algorithm recovering more than half of the optimal solutions in the test set from BOLIB [85]. The aim this part of analysis of [Paper 2] was to link practical observations to Lemma 2.2 there. Finding inflection point $\bar{\lambda}$ for all examples would give the idea of what values of λ were optimal for the examples in the test set. As the test set is quite large (containing 124 problems) this could provide a solid conjecture on what magnitude of λ could be the best for (1.9). Attempting to find the inflection point, we relaxed all stopping criteria defined before and have set the new one, namely to stop once we get $\mathtt{Error} < 1.1\mathtt{Error}^*$, where $\mathtt{Error}^*$ is the value of the $\mathtt{Error}$ obtained by the algorithm in the actual experiments. We would then run the algorithm with varying λ for all examples in the test set with this new stopping criterion. This way algorithm stopped once we get $\mathtt{Error}$ close enough to what is suggested to be optimal by the algorithm. However, doing so, algorithm quite often stopped early and reported small value λ for which good enough solution with $\mathtt{Error} < 1.1\mathtt{Error}^*$ was already obtained after a few iterations. For instance, looking at Figure 2 we aimed to find the inflection point around 180-190 iterations. Instead, we get $\bar{\lambda}$ reported to be λ after 15 iterations. The reason for this is that behaviour of the algorithm for the majority of the examples is similar to the one shown in Figure 2. As can be seen in the figure, it is possible that we obtain the solution very close to the optimal one after a few iterations. It is interesting as the theory suggests that large values of λ should be the best choice but we observe that this is not the case for many problems.

To find the inflection point it was decided to reconsider the approach and to find λ^*, which is obtained in the same way as $\bar{\lambda}$ but once at least 50 iterations are made. That is we stop if $\mathtt{Error} < 1.1\mathtt{Error}^*$ and $\mathtt{iter} > 50$. As it is interesting to see for how many examples we get $\bar{\lambda}$

different from λ^*, we decided to perform the search for both of them. When these are different it means that behaviour shown in Figure 2 holds, i.e. solution is recovered early for small values of λ, then blows up and comes back to the optimal value at λ^* and is retained afterwards until possibly ill-behaviour happens and the value blows up. Of course, we only tested the examples where solutions were recovered, as $\mathtt{Error}^*$ would not be reliable otherwise. As the result of the comparison we then get $\bar{\lambda}$ that stands for the first (smallest) value of λ for which optimal solution was obtained, while λ^* represent the actual threshold after which solution is retained for further iterations with $\lambda > \lambda^*$. One of the main observations of this section is that small threshold is smaller than large threshold ($\bar{\lambda} < \lambda^*$) for 59/72 (82%) problems. This clearly shows that for majority of the problems, for which we recover solution with $\lambda := 0.5 \times 1.05^k$, we obtain a good solution for small λ as well as for large λ. This demonstrates that small λ could in principle be good for the method. For the rest 18% of the problems we have $\bar{\lambda} = \lambda^*$, meaning that good solution was not obtained for $\lambda < 6$ for these examples. This also means that we typically obtain a good solution for small values of λ and for large values of λ, but not for the medium values ($\bar{\lambda} < \lambda < \lambda^*$). As for the observations on the magnitudes of λ at the inflection point, for 42/72 (58.33%) examples we observed that the large threshold λ^* is located somewhere in between $90 - 176$ iterations with $40 < \lambda^* < 2680$, for 7/72 problems threshold is in the range $6.02 < \lambda^* \leqslant 40$, and for only 4/72 problems $\lambda^* > 1.1 \times 10^4$. This justifies that typically λ does not need to be large.

As further observed in Section 4 of [Paper 2], we could actually argue that smaller values of λ work better for our method not only for varying λ but also for fixed λ. Together with the fact that we often have the behaviour as demonstrated in Figure 2, it follows that small λ could be more attractive for the method we implement. We even get better values of $\mathtt{Error}$ and better solutions for small values of λ for some examples. Hence, we draw the conclusion that small values of λ can generate good solutions for our framework. Since it is typical to use large values of λ for other penalization methods (e.g. in [8, 11, 59, 61]), it is interesting to see that small λ turned out to work better for our case. This could be due to the specific nature of the method, or due to the fact that we do not do full penalization in the usual sense, or due to the structure of the problems in the test set. This could also possibly be the case that small values of penalty parameter could be good for some other penalty methods and optimization problems of different nature. At least for our framework we claim that λ needs not to be large, which is counter-intuitive to what penalization theory suggests. This could highlight important property of partial exact penalization for bilevel framework, as well as suggest the possibility of such behaviour for other frameworks with methods of similar nature being implemented.

In the last part of Section 3 in [Paper 2] we look closely at the behaviour of the algorithm with varying λ for 7 nonlinear examples from BOLIB [85] that are known to be partially calm. These are guaranteed to be partially calm due to their structure, according to [52]. Partially calm examples fit the theoretical structure behind the penalty approach that we take. These examples are meant to follow the pattern of retaining solution after some threshold, i.e. for $\lambda > \lambda^*$. However, this turns out not to always be the case for the tested partially calm examples from BOLIB [85]. To proceed, we relaxed the stopping criteria and reran the algorithm with varying λ for 1000 iterations for each of these examples. Three different scenarios were observed. In the first scenario, algorithm was performing well, retaining the solution for the number of iteration, but then blowed up at some point after 500 iterations and never came back to reasonable solution values. Three considered examples follow this pattern. In the second scenario we see the zig-zagging pattern. Algorithm blows up at some point and starts zig-zagging away from the solution after obtaining it for a smaller value of λ. This is somewhat similar to scenario 1. However, we put this separately as zig-zagging issue is often referred to as the danger that could be caused by ill-conditioning of a penalty function. Such pattern is observed for two of the seven considered examples. We then refer to *ill behaviour* if scenario 1 or scenario 2 was observed for some example. Finally, In the third scenario, ill-conditioning was not observed throughout running algorithm for 1000 iterations. It could be possible that algorithm would blow up after more iterations if we keep increasing λ. It could also be possible that ill-condition does not occur for these example at all. Out of the examples with the specified structure two examples follow this pattern.

In terms of the numerical experiments, the results of implementing the algorithm with all the specified parameters for 124 nonlinear problems from BOLIB [85] are presented in the last section of [Paper 2]. This creates reproducible extensive experiments for a good variety of the choices of the penalty parameter, being ten fixed values $\lambda \in \{10^6, 10^5, \dots, 10^{-3}\}$ and the option of varying parameter $\lambda := 0.5 \times 1.05^k$. The measures for the comparison in [Paper 2] were accuracy of upper-level objective, lower-level feasibility, experimental order of convergence and line-search stepsize. With these measures we clearly observed that if one could optimally choose the best value of λ for each problem, fixing λ outperforms the approach of varying λ for the chosen framework. However, varying λ has shown reasonably strong performance, competing with random choice of a fixed value $\lambda \in \{10^6, 10^5, \dots, 10^{-3}\}$. Clearly, if for some framework solutions to the problems are not known, it could be complicated to decide which fixed value of λ was the best even after obtaining the results. For instance, this challenge was faced in the experiments for modified examples in [Paper 3]. In this case varying λ could very well be more attractive choice than fixing λ.

In [Paper 2] we analyzed Levenberg-Marquardt method for bilevel optimization. We observed that the method is very sensitive to the choice of the parameters and gave suggestions that worked well in the context of solving bilevel optimization problems. Most importantly, we provided a detailed discussion on choosing the exact penalty parameter. Interestingly, we observed that penalty parameter λ does not need to be large for the considered problem (1.9), which contradicts penalization theory in some way. Based on the analysis, safeguard stopping criteria was introduced to prevent algorithm running for too long when not necessary. In the final section of the paper, it was shown that algorithm performs well for the test set of nonlinear problems from BOLIB [85]. We observed that Levenberg-Marquardt method in [Paper 2] always converges and recovers majority of the known solutions. Algorithm has also demonstrated to be very fast, with average CPU time for all separate fixed values of λ being 0.24 seconds and average CPU time for varying λ being 0.53 seconds. Experiments also give the idea of the performance of the algorithm for different choices of penalty parameter λ, where varying λ seems to be as good as a random fixed λ. However, if one is able to choose the best fixed value from the tested values the approach of fixing λ outperforms that of varying λ. For both approaches of choosing λ the method has been shown to be appropriate for bilevel framework, and even could be competitive with the known solution methods for bilevel optimization.

4 LEVENBERG-MARQUARDT METHOD FOR LINEAR BILEVEL OPTIMIZATION

In the first two papers [Paper 1, Paper 2] we have based our analysis on solving nonlinear bilevel problems. This section summarizes the work done in [Paper 3], where this approach was extended to linear bilevel optimization problems (BLPs). For this class of problems we are dealing with (BP), where all functions are linear, i.e.

$$\begin{aligned} \min_{x,y} \quad & F(x,y) := c_1^\top x + d_1^\top y \\ \text{s.t.} \quad & A_1 x + B_1 y - b_1 \leqslant 0, \\ & y \in S(x) := \arg\min_y \left\{ f(x,y) = c_2^\top x + d_2^\top y : \ A_2 x + B_2 y - b_2 \leqslant 0 \right\}, \end{aligned} \tag{4.1}$$

where $c_1 \in \mathbb{R}^{n\times 1}$, $d_1 \in \mathbb{R}^{m\times 1}$, $A_1 \in \mathbb{R}^{q\times n}$, $B_1 \in \mathbb{R}^{q\times m}$, $b_1 \in \mathbb{R}^{q\times 1}$, $c_2 \in \mathbb{R}^{n\times 1}$, $d_2 \in \mathbb{R}^{m\times 1}$, $A_2 \in \mathbb{R}^{p\times n}$, $B_2 \in \mathbb{R}^{p\times m}$, and $b_2 \in \mathbb{R}^{p\times 1}$. The most common methodologies to solve BLPs include enumerative algorithms, implicit function approach, Simplex method and lower-level KKT reformulation. Enumerative algorithm proceeds with enumerating extreme points of the polyhedron and chooses the best one with respect to the upper-level objective function. However, such method is known to be very slow as feasible set is not convex. One of the most popular approaches to solve BLPs deals with transforming problem (4.1) into a single level problem using *Karush-Kuhn-Tucker* (KKT) conditions. The approach substitutes lower-level problem by KKT conditions, transforming bilevel problem into single-level problem. As this is one of the most popular approach and has the link to LLVF reformulation we aim to compare LLVF-based approach to the KKT-based one. This was also motivated by the comparison of KKT and LLVF reformulations made in [84] for the general case. According to [84], for a nonlinear problem it is not obvious which reformulation is more beneficial in terms of the assumptions needed for the reformulation to be obtained. In [Paper 3] comparison was studied for the linear class of bilevel optimization problems. What is interesting about this class of problems is that some main assumptions needed to state optimality conditions for both KKT and LLVF reformulations hold automatically due to linearity of (4.1). Clearly, one does not need to assume continuity of the original functions defining (4.1). Further, full convexity of the lower-level problem and partial calmness condition hold for this framework. In this paper we study second order method introduced in [Paper 2] to solve (4.1). It is worth saying, that second order methods are not typically studied for linear problems. We believe that Levenberg-Marquardt method fits well for the framework of linear bilevel problems, as discussed in [Paper 3]. In [Paper 3] we have shown the theoretical simplifications arising due to the linear structure of (4.1) for the chosen solution method. We also demonstrate that the method can converge for problem (4.1) theoretically and practically. As observed in numerical part of the analysis in [Paper 3] the method has shown to be very fast in comparison to the known solution methods for linear bilevel optimization.

As in [Paper 1, Paper 2] to implement the method we require optimality conditions to be in the form of a system of equations. Once again, we are using NCP-functions to state the system (1.9) for the linear case, which takes the form

$$\Upsilon^{\lambda}_{LLVF}(z) := \begin{pmatrix} c_1 + A_2^\top(u - \lambda w) + A_1^\top v \\ d_1 + B_2^\top(u - \lambda w) + B_1^\top v \\ d_2 + B_2^\top w \\ \sqrt{u^2 + (A_2x + B_2y - b_2)^2} - u + A_2x + B_2y - b_2 \\ \sqrt{v^2 + (A_1x + B_1y - b_1)^2} - v + A_1x + B_1y - b_1 \\ \sqrt{w^2 + (A_2x + B_2y - b_2)^2} - w + A_2x + B_2y - b_2 \end{pmatrix} = 0. \tag{4.2}$$

Similarly, KKT-based optimality conditions with the use of NCP-function was stated as the system

$$\Upsilon^{\lambda}_{KKT}(z) := \begin{pmatrix} c_1 + A_2^\top(u - \lambda w) + A_1^\top v \\ d_1 + B_2^\top(u - \lambda w) + B_1^\top v \\ d_2 + B_2^\top w \\ -\lambda(A_2x + B_2y - b_2) + B_2 s + \eta \\ \sqrt{u^2 + (A_2x + B_2y - b_2)^2} - u + A_2x + B_2y - b_2 \\ \sqrt{v^2 + (A_1x + B_1y - b_1)^2} - v + A_1x + B_1y - b_1 \\ \sqrt{\eta^2 + w^2} - w + \eta \end{pmatrix} = 0. \tag{4.3}$$

It was noted that the two systems have a lot of similarities. We provide the link between the systems in Proposition 2.4 in [Paper 3], where we present conditions under which KKT-based optimality conditions and LLVF-based optimality conditions are equivalent in the context of linear bilevel optimization. However, comparing them separately allows us to study if one is possibly better than the other. The first obvious difference is that KKT-based system (4.3) is a square system, meaning the number of equations is the same as the number of the variables, which is not the case for LLVF-based system (4.2). To proceed, Levenberg-Marquardt method to solve (4.2) and (4.3) was introduced in Section 2.2 of [Paper 3]. The method takes the direction as defined in (3.1). As before, we use smoothing technique to achieve differentiability of the systems. With the method of this nature, instead of directly solving the systems, we are dealing with the minimization of the following least-squares problem,

$$\Phi^{\lambda,\mu}_{KKT}(z) = \frac{1}{2}\left\|\Upsilon^{\lambda,\mu}_{KKT}(z)\right\|^2, \quad \Phi^{\lambda,\mu}_{LLVF}(z) = \frac{1}{2}\left\|\Upsilon^{\lambda,\mu}_{LLVF}(z)\right\|^2, \tag{4.4}$$

It was further discussed that Levenberg-Marquardt method is essentially a combination of Gauss-Newton method and *gradient descent method*. Clearly, Levenberg-Marquardt step (3.1) takes the Gauss-Newton (2.3) for $\alpha(z^k) \to 0$ and gradient descent direction for $\alpha(z^k) \to \infty$. This is interesting as gradient descent method is the first order method that is commonly used to solve linear problems. This means that we use combination of first order method and second order method to solve linear bilevel problems. This makes a lot of sense as functions defining (4.1) are linear and we have nonlinear terms in the introduced systems (4.2) and (4.3), appearing due to complementarity constraints. Similarly to [Paper 2], we define Levenberg-Marquardt method with line search and state the convergence result from [27]. For the framework in [Paper 3] we note that Lipschitz-continuity assumption needed for the convergence is automatically satisfied for (4.2) and (4.3). This further benefits the framework of implementing Levenberg-Marquardt method to find stationary points of linear bilevel problems (4.1) compared to the general case (BP). However, convergence of the method also requires *error bound condition*, which becomes a little bit tricky. It is known that error bound condition holds for piecewise linear functions according to [78]. However, due to NCP-function present in (4.2) and (4.3), we cannot exploit this property for linear bilevel programming problems, at least for considered reformulations. To show that condition can hold for (4.1) we aimed to demonstrate that columns of the Jacobians $\nabla\Upsilon^{\lambda,\mu}_{LLVF}$ and $\nabla\Upsilon^{\lambda,\mu}_{KKT}$ are linearly independent under appropriate assumptions. As already discussed in [Paper 2], according to [78] this is sufficient for the required error bound condition to hold.

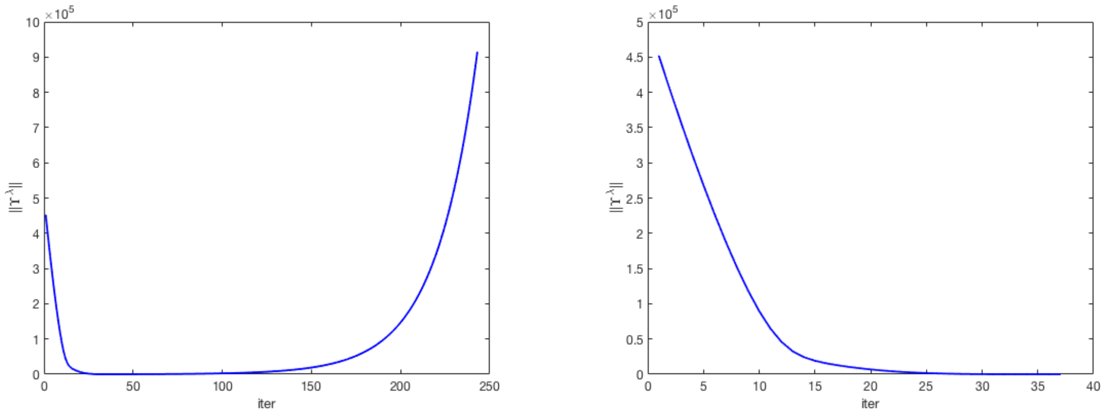
Although linear framework simplifies problem a lot, linear independence of the columns of Jacobian matrices is in some sense harder to preserve due to some key terms vanishing from the Jacobian matrices. Nevertheless, convergence results for LLVF and KKT frameworks are stated in Theorem 2.8 and Theorem 2.9 in [Paper 3] respectively. Two scenarios are considered to show full rank of the Jacobian $\nabla\Upsilon^{\lambda,\mu}_{LLVF}$. For $\nabla\Upsilon_{KKT}$ one scenario of obtaining required condition is analyzed. The first scenario of both convergence theorems is somewhat similar. This scenario considers problems with the number of lower-level constraints being not bigger than the number of lower-level variables. Especially $p = m$ is common class of problems, where the set of lower-level constraints spans the dimension of lower-level variables. The case $p < m$ is less common but it is still interesting that such property of a problem allows to state convergence proof with less assumptions than more natural scenario with $p > m$, which is considered in the second scenario for $\nabla\Upsilon^{\lambda,\mu}_{LLVF}$. This scenario in Theorem 2.8 in [Paper 3] imposes the upper bound on λ, which is interesting as this suggests that smaller values of λ could be better for the method in terms of its convergence. This suggestion is in line with one of the main conjectures of [Paper 2], where we discussed that algorithm performs well with small values of λ, and also that large values of λ could cause issues for the algorithm. The first scenario in both convergence theorems require full rank condition of the matrix B_2 defined in (4.1). Further, the following assumption is used to prove the theorems (see [Paper 3, Assumption 2.7]).

Assumption 4.1. $q + p \geqslant n + m$ and the matrix $\begin{bmatrix} A_1 & B_1 \\ A_2 & B_2 \end{bmatrix}$ is full rank.

This is the main assumption used for both convergence theorems. It is easy to see from (4.1) that this assumption is the same as assuming that the family of the vectors $\{\nabla G_j \cup \nabla g_j\}$ is linearly independent, which is very standard assumption to show that columns of Jacobian matrix are linearly independent for the general framework (e.g. in [84]). The assumption on the rank of $B_2^\top$ in scenario 1 in both theorems is the assumption that we made to ensure lower-level regularity holds (see Assumption 2.2 in [Paper 3]). Nicely, for both proofs we exploited Lemma 4.2 from [Paper 1], where we have shown that some elements of the Jacobians $\nabla\Upsilon^{\lambda,\mu}_{LLVF}$ and $\nabla\Upsilon^{\lambda,\mu}_{KKT}$ are strictly positive, and some elements are strictly negative. To finish off the convergence discussion, we highlight that due to similarity of the systems, convergence result shown for one system could be applied to the other one if the systems are equivalent. For instance, if conditions described in Proposition 2.4 in [Paper 3] are fulfilled, then systems are equivalent and full rank condition shown for $\nabla\Upsilon^{\lambda,\mu}_{LLVF}$ would imply full rank of $\nabla\Upsilon^{\lambda,\mu}_{KKT}$ and vice versa. What is further interesting about the proofs is that the stated conditions ensuring full rank of the Jacobian matrices have not yet been studied in the literature. The introduced conditions could be exploited for the benefit of theoretical properties of the other methods of similar nature (e.g. Newton method or Gauss-Newton method) to solve (4.2) and (4.3). We hope that this brings a unique impact for the linear framework of KKT-based and LLVF-based optimality conditions for bilevel optimization.

Once we have shown that the method can converge in theory, we move on to implementation of the algorithm. As observed in [Paper 2] Levenberg-Marquardt algorithm for bilevel optimization is sensitive to the choice of parameters. The choices considered there were for nonlinear case. For [Paper 3] these were carefully reevaluated and it turns out that most of the choices in [Paper 2] fit well for the linear framework, compared to the alternative choices. Similarly to [Paper 2], the choices $\alpha(z^k) := \left\|\Upsilon^{\lambda,\mu}_{KKT}\right\|$ and $\alpha(z^k) := \left\|\Upsilon^{\lambda,\mu}_{LLVF}\right\|$ perform well for the linear framework. Smoothing parameter is also taken as $\mu_k := 0.001/(1.5^k)$ as in [Paper 2]. Further, the choice of penalty parameter is almost the same with fixed values $\lambda \in \{10^5, 10^4, 10^3, 10^2, 10^1, 10^0, 10^{-1}, 10^{-2}$ and varying option $\lambda := 0.5 \times 1.05^k$, where k is the number of iterations. The reason we drop $\lambda \in \{10^6, 10^{-3}\}$ considered in [Paper 2] is because these values show exactly the same performance as $\lambda \in \{10^5, 10^{-2}\}$. In general, adjacent fixed values of λ show more or less similar performance once it gets $\lambda > 10^3$ or $\lambda < 10^0$.

Even though the choice of parameters is similar to [Paper 2], it was observed that the typical behaviour of the algorithm is somewhat different for the instances of linear problems considered in [Paper 3] to nonlinear problems considered in [Paper 2]. It is observed that in the context of algorithm with varying λ, the solution is typically found quite quickly (after 20-100 iterations) for both fixed and varying λ. However, for the varying λ it seems to be always the case that the value of the Error blows up at some point between 200-400 iterations as shown in Figure 3 (a) below, where the behaviour of the algorithm with varying λ is demonstrated for `Bblp_20_15_50_10_1` example.



(a) Typical behaviour of the algorithm for linear problems

(b) Implementing stopping criteria

Figure 3: Typical behaviour of the algorithm and new stopping criteria for linear bilevel problems

After blowing up, the value of the Error typically keeps growing and does not come back to reasonable values. Interestingly, for the linear examples our algorithm with varying λ typically performs as demonstrated in Figure 3 (a) above for both LLVF and KKT reformulations. As algorithm obtains a good solution and retains it for many iterations and then blows up without getting back to reasonable values, we aimed to provide the criteria to stop earlier. For this reason it makes sense to set maximum number of iterations to $K := 200$ with additional stopping criteria

$$\textbf{STOP if} \quad \left\|\Upsilon^\lambda(z^{k-1})\right\| - \left\|\Upsilon^\lambda(z^k)\right\| < 10^{-7} \;\;\&\;\; \mathtt{iter} > 5,$$

that is if the improvement from step to step becomes less than ϵ by the two orders of magnitude, OR if the direction becomes not descent ($\left\|\Upsilon^\lambda(z^{k-1})\right\| - \left\|\Upsilon^\lambda(z^k)\right\| < 0$) we stop. We also impose that a few iterations is made, i.e. $\mathtt{iter} > 5$ in case one of the first iterations takes non-descent direction. The implementation of such stopping criteria is demonstrated in 3 (b) above. It seems a good strategy as we do not risk the value of the Error to blow up this way. Also as the solution is recovered early, this would not harm results in any way to set $K := 200$. Condition on λ in the second scenario of Theorem 2.8 in [Paper 3] also suggests that λ is not too large. This justifies stopping criteria with the maximum number of iterations being relatively small. The behaviour of Figure 3 (a) was observed for most of the linear problems in the test set for both KKT-based and LLVF-based approach. Hence, we impose the defined stopping criteria for both reformulations.

It is worth noting that usually algorithms for linear bilevel problems are tested for very small problems. Further, it seems that literature suffers from the lack of linear bilevel test problems. Methods are typically tested for less than 20 instances, which might not be sufficient to make valid conclusions about an algorithm, as suggested in [7]. Hence, we decided to construct a large test set of linear bilevel problems, using mixed integer problems from [32] as a basis. For these examples we consider the transformation, where we drop integer constraints and replace binary constraints with the bounds on the corresponding variables (please see supplementary materials [Supp3] for more details). However, these examples can be considered to be new examples and hence solutions to these are unknown. Hence, we cannot have valid performance measures of the algorithm to compare solving (4.2) and (4.3). The transformed examples are of medium size and they play the role of the extensive basis for the comparison for other authors in the future. We provide all details on implementation of the method to make provided results reproducible and comparable with the other solution methods. To our knowledge there are no works providing such an extensive results of experiments for the linear bilevel programming problems. Importantly, algorithm introduced in [Paper 3] always converges, which means we were able to provide values obtained by the algorithm for each example. For the comparison purpose of KKT-based and LLVF-based approaches we use test set of linear examples from BOLIB [85]. So far this contains 24 linear problems of small size, but we hope that 174 transformed examples from [32] could be included in [85] in the future versions of the library. We have already observed in [Paper 2] that solutions obtained by the algorithm with varying λ could somewhat be reasonable for more than half of nonlinear examples. We observed that for the linear case varying λ could be almost as good as best fixed λ. As the approach of fixing λ has a few disadvantages, this brings an important conjecture that if one wants to use Levenberg-Marquardt method to solve linear bilevel optimization problems, varying λ could be the best option to consider. Further, as we mentioned in [Paper 2], varying λ is more in line with what penalization theory suggests to do, that is to set λ as an increasing sequence. In terms of the comparison, LLVF-based approach has shown stronger overall numerical performance than KKT one. There is a small concern that it has slightly higher feasibility error but this could be explained by the overdetermined nature of LLVF system (4.2), as opposed to the square system of equations (4.3) being solved for KKT reformulation. Nevertheless, algorithm has shown slightly better performance for LLVF-based system.

5 CONCLUSION

In [Paper 1] we tested novel approach of implementing Gauss-Newton to solve system of LLVF-based optimality conditions for bilevel optimization. First property being studied was differentiability of the system that is required for Gauss-Newton direction to be calculated. To preserve differentiability two scenarios were considered in the context of implementing Gauss-Newton method. The first scenario involved assumption of strict complementarity, which was discussed to be rather strong assumption to hold in practice. Nevertheless, the study of the framework has shown that Gauss-Newton method could be well-defined and converge under this scenario. For the second scenario we introduced the framework where differentiability of the system was obtained with the use of smoothing technique. Smoothed Gauss-Newton was shown to be well-defined and converge under this scenario. Alongside with standard Gauss-Newton method, Newton method with pseudo inverse (Pseudo-Newton method) was discussed. This was verified theoretically and practically that the methods are equivalent if the Gauss-Newton direction is well-defined. In the experiments both methods were compared with MATLAB built-in solve fsolve. It has been shown that solving introduced LLVF-based optimality conditions with methods of this class is a valid approach, recovering optimal solutions for most of the tested examples from BOLIB [85]. Gauss-Newton method and Pseudo-Newton method performed better than fsolve with all the measures considered in the numerical section in [Paper 1]. We have also shown that Pseudo-Newton method is at least as good as

Gauss-Newton method. The main difference between the methods is that Pseudo-Newton would always produce a number as an output, i.e. it would never diverge completely. Together with similar speed of the methods, this brings a suggestion that Pseudo-Newton method could be a better choice to implement in practice. It was also discussed that for a square system Pseudo-Newton method would perform at least as good as the classic Newton method. For the further step, it would be interesting to see if observed behaviour of the methods would hold for larger real-life problems. It would also be interesting to look at the different notions of stationarity to see what optimality conditions are the best for bilevel optimization. Finally, different safeguard measures could be interesting to study to improve implementing such methods for problems that could be more challenging than in the analyzed test set.

In [Paper 2] we introduced a lot of non-trivial choices of the parameters and specifications required for the Levenberg-Marquardt method to fit well to bilevel optimization framework. One of the outcomes of the work is the suggestion that the choice of the Levenberg-Marquardt parameter $\alpha(z^k) := \|\Upsilon^\lambda(z^k)\|$ performs well for introduced bilevel framework both theoretically and numerically. We have further discussed the challenge of choosing the exact penalty parameter λ. Some numerical study has been performed to understand the behaviour of the algorithm for the case of varying penalty parameter. Interestingly, this analysis has resulted in the suggestion that penalty parameter λ does not need to be large for the considered problem (1.9). Further disadvantage of large values of λ was discussed to be the possibility of the ill-conditioning occurring for large values of λ. To deal with this issue additional stopping criteria was discussed as a way to provide a safeguard for the algorithm. Although introduced stopping criteria performed very well for the considered test set, it cannot be guaranteed that ill behaviour would be always avoided by the algorithm. For anyone considering varying choice of penalty parameter, we suggest non-aggressive choice of the sequence and safeguard measures to prevent penalty parameter growing too large. The implementation of the algorithm on the test set of nonlinear BOLIB [85] problems has shown that algorithm performs well for our framework for both fixed and varying λ. It is worth saying that unlike Gauss-Newton method considered in [Paper 1], Levenberg-Marquardt method in [Paper 2] always converges. In terms of the time algorithm was observed to be very fast, which makes it competing with other solution methods in the field. It is further concluded that if one has a way to choose best fixed value of λ, then fixing λ outperforms approach of varying λ. However, due to reasonably strong performance of varying λ, varying λ could be better option if one does not want to deal with the choice of different fixed values of λ. However, one should be cautious that with such approach there is a possibility that ill-behaviour could influence the algorithm to produce solutions that are not reasonable at all. As the further step, it would be interesting to study what leads to the observed behaviour of the algorithm for different choice of penalty parameter λ. For instance, it could be the case that small λ was good for examples with certain structure or that algorithm could have some other tuning so that ill behaviour is avoided completely. It would also be interesting to study the method for real-life problems and also to test if the suggestions on choosing penalty parameter could be valid for other classes of problems.

In paper [Paper 3] we have considered two types of optimality conditions for linear bilevel problems (4.1). Motivation of the approach was that many assumptions needed to state optimality conditions are satisfied automatically due to linearity of the function defining (4.1). The optimality conditions based on KKT and LLVF reformulations were stated in the form of systems of equations (4.3) and (4.2). It was noted that systems have a lot of similarities and it was shown that there is strong link between the systems. In terms of theoretical properties it remained unclear which one is better. To compare KKT-based and LLVF-based approaches, Levenberg-Marquardt method to solve such systems was discussed. It was mentioned that Levenberg-Marquardt method is essentially a combination of second order and first order method, which also motivated such method to be studied for the framework of linear bilevel problems. Interestingly, convergence of the method could be shown to hold under the full rank condition of Jacobian matrices $\nabla\Upsilon^{\lambda,\mu}_{KKT}$ and $\nabla\Upsilon^{\lambda,\mu}_{LLVF}$. We have shown that for both reformulations there are scenarios under which Jacobian of the system being solved has full column rank. Hence, the introduced method was shown to be theoretically appropriate to solve linear bilevel problems (4.1) in the sense of theoretical convergence. In terms of practical comparison, algorithm for LLVF-based approach has shown slightly better performance than for KKT-based approach, based on testing the method for 24 linear problems from BOLIB [85]. We further present the results of implementing the algorithm for 174 transformed examples from [32]. For these examples it was observed that large fixed values of λ, that is $\lambda \in \{10^5, 10^4, 10^3\}$ produce similar results among themselves. The likely explanation takes the roots in the theory of partial calmness that was discussed in [Paper 2]. We know that theoretically there exists threshold value $\bar{\lambda}$ such that if the optimal solution is obtained for $\bar{\lambda}$ it will hold for any $\lambda > \bar{\lambda}$. This is likely to be the case that similar behaviour for fixed $\lambda > 10^3$ due to this property. Nevertheless, this observation is somewhat similar to what was observed in the experiments for nonlinear BOLIB examples in [Paper 1, Paper 2]. We hope that for transformed examples from [32] other authors could judge our solutions based on the results of implementing their algorithms to the same problems. Most importantly, quadratic method considered in [Paper 3] has not yet been tested in the context of solving linear bilevel problems. It is worth noting that method always converges and requires average computation time of less than 1 second. Together with the good rate of recovering solutions, this method seems to compete very well with other methodologies to solve linear bilevel optimization problems. It would further be interesting to test if other quadratic methods could be efficient to solve linear bilevel optimization problems, especially problems of large size.

Part II.

Paper 1: Gauss-Newton-type Methods for Bilevel Optimization

This article studies Gauss-Newton-type methods for over-determined systems to find solutions to bilevel programming problems. To proceed, we use the lower-level value function reformulation of bilevel programs and consider necessary optimality conditions under appropriate assumptions. First, under strict complementarity for upper- and lower-level feasibility constraints, we prove the convergence of a Gauss-Newton-type method in computing points satisfying these optimality conditions under additional tractable qualification conditions. Potential approaches to address the shortcomings of the method are then proposed, leading to alternatives such as the pseudo or smoothing Gauss-Newton-type methods for bilevel optimization. Our numerical experiments conducted on 124 examples from the recently released Bilevel Optimization LIBrary (BOLIB) compare the performance of our method under different scenarios and show that it is a tractable approach to solve bilevel optimization problems with continuous variables.

1 INTRODUCTION

We aim to solve the bilevel programming problem

$$\min_{x,y} F(x,y) \ \text{ s.t. } \ G(x,y) \leqslant 0, \ y \in S(x) := \arg\min_{y} \{f(x,y) : \ g(x,y) \leqslant 0\}, \tag{1.1}$$

where $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $G : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$, and $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$. As usual, we refer to F (resp. f) as the upper-level (resp. lower-level) objective function and G (resp. g) as the upper-level (resp. lower-level) constraint function. Solving problem (1.1) is very difficult because of the implicit nature of the lower-level optimal solution mapping $S : \mathbb{R}^n \rightrightarrows \mathbb{R}^m$ defined in (1.1).

There are several ways to deal with the complex nature of problem (1.1). Earliest techniques were based the implicit function and Karush-Kuhn-Tucker (KKT) reformulations. The implicit function approaches rely on the insertion of the lower-level solution function or its approximation in the upper-level objective function; see, e.g., [17, 21] for related algorithms. As for the KKT reformulation, it is strongly linked to MPECs (mathematical programs with equilibrium constraints), see, e.g., [19], which are not necessarily easy to handle, in part due to the extra variables representing the lower-level Lagrangian multipliers. Interested readers are referred to [1, 23, 41] and references therein, for results and methods based on this transformation. More details on methods based on the KKT reformulation and other approaches to deal with bilevel optimization can be found in the books [3, 18]. In this paper, we are going to use the lower-level value function reformulation (LLVF)

$$\min_{x,y} F(x,y) \ \text{ s.t. } \ G(x,y) \leqslant 0, \ \ g(x,y) \leqslant 0, \ \ f(x,y) \leqslant \varphi(x), \tag{1.2}$$

where the optimal value function is defined by

$$\varphi(x) := \inf \{f(x,y) \,|\, g(x,y) \leqslant 0\}, \tag{1.3}$$

to transform problem (1.1) into a single-level optimization problem. As illustrated in [31], this approach can provide tractable opportunities to develop second order algorithms for the bilevel optimization problem, as it does not involve first order derivatives for the lower-level problem, as in the context of the KKT reformulation.

There are recent studies on solution methods for bilevel programs, based on the LLVF reformulation. For example, [46, 47, 53, 58, 73] develop global optimization techniques for (1.1) based on (1.2)–(1.3). [50, 75, 76] propose algorithms computing stationary points for (1.2)–(1.3), in the case where the upper-level and lower-level feasible sets do not depend on the lower-level and upper-level variable, respectively. [31] is the first paper to propose a Newton-type method for the LLVF reformulation for bilevel programs. Numerical results there show that the approach can be very successful. The system of equations considered there is quadratic while our method in this paper is based a non-quadratic overdetermined system of equations. Hence, there is a need to develop Gauss-Newton-type techniques to capture certain classes of bilevel optimization stationarity points.

One of the main problems in solving (1.2) is that its feasible set systematically fails many constraint qualifications (see, e.g., [24]). To deal with this issue, we will use the partial calmness condition [81], to shift the value function constraint $f(x, y) \leqslant \varphi(x)$ to the upper-level objective function, as a penalty term with penalty parameter λ. The other major problem with the LLVF reformulation is that φ is typically non-differentiable. This will be handled by using upper estimates of the subdifferential of the function; see, e.g., [24, 20, 23, 81]. Our Gauss-Newton-type scheme proposed in this paper is based on a relatively simple system of optimality conditions that depend on λ.

To transform these optimality conditions into a system of equations, we substitute the corresponding complementarity conditions using the standard Fischer-Burmeister function [29]. To deal with the non-differentiability of the Fischer-Burmeister function, we consider two approaches in this paper. The first one is to assume strict complementarity for the constraints involved in the upper- and lower-level feasible sets. As second option to avoid non-differentiability; here we investigate a smoothing technique by adding a perturbation inside the Fischer-Burmeister function.

Another important aspect of the aforementioned system of equations is that it is *overdetermined*. Since overdetermined systems have non-square Jacobian, we cannot use a classical Newton-type method as in [31]; for a Newton method addressing the closely related semi-infinite programming problem, interested readers are referred to [65, 66]. Gauss-Newton and Newton-type methods with Moore-Penrose pseudo inverse are both introduced in Section 3. It will be shown that these methods are well-defined for solving bilevel programs from the perspective of the LLVF reformulation (1.2). In particular, our framework ensuring that the Gauss-Newton method for bilevel optimization is well-defined does not require any assumption on the lower-level objective function. Links between the two methods are then discussed and it is shown that they should perform very similarly for most problems. However, it is expected that the Newton-type method with pseudo inverse will be more robust, as the evidence from the numerical experiments (Section 5) confirms.

In Section 5, we present results of extensive experiments on the methods and comparisons with the MATLAB built-in function *fsolve*. Considering the complicated nature of the feasible set of problem (1.1), the results from the algorithms are compared with known solutions of the problems to check if obtained stationary points are optimal solutions of the problems or not. For 124 tested problems (taken from the BOLIB library [85]), more than 80% are solved satisfactorily in the sense that we recover known solutions with $< 10\%$ error, or obtain better ones by all methods with CPU time being less than half a second. The number of recovered solutions as well as the performance profiles and feasibility check show that Gauss-Newton and Newton method with pseudo inverse outperform *fsolve*. It is worth mentioning here that it is not typical to conduct such a large number of experiments in the bilevel optimization literature. The conjecture of the similarity of the performance of the tested methods is verified numerically, also showing that Newton's method with pseudo inverse is indeed more robust than the classic Gauss-Newton one. However, the technique for choosing the penalty parameter λ is still a heuristic, and the result might depend on the structure of the corresponding problem.

2 OPTIMALITY CONDITIONS AND EQUATION REFORMULATION

Let us start with some definitions required to state the main theorem of this section. Define *full convexity* of the lower-level problem as convexity of the lower-level objective function, as well as all lower-level constraints, with respect to (x, y). Further on, a point $(\bar{x}, \bar{y}) \in \mathbb{R}^n \times \mathbb{R}^m$, feasible for lower-level problem, is said to be *lower-level regular* if there exists a direction $d \in \mathbb{R}^m$ such that

$$\nabla_y g_i(\bar{x}, \bar{y})^T d < 0 \quad \text{for } i \in I_g(\bar{x}, \bar{y}) := \{i : g_i(\bar{x}, \bar{y}) = 0\}. \tag{2.1}$$

It is clear that (2.1) corresponds to the Mangasarian-Fromowitz constraint qualification for the lower-level constraint at the point $\bar{y}$, when the upper-level variable is fixed at $x := \bar{x}$. Similarly, the point $(\bar{x}, \bar{y}) \in \mathbb{R}^n \times \mathbb{R}^m$ satisfying the upper- and lower-level inequality constraints is *upper-level regular* if there exists a direction $d \in \mathbb{R}^{n+m}$ such that

$$\begin{aligned} \nabla G_j(\bar{x}, \bar{y})^T d < 0 \quad & \text{for } j \in I_G(\bar{x}, \bar{y}) := \{j : G_j(\bar{x}, \bar{y}) = 0\}, \\ \nabla g_j(\bar{x}, \bar{y})^T d < 0 \quad & \text{for } j \in I_g(\bar{x}, \bar{y}) := \{j : g_j(\bar{x}, \bar{y}) = 0\}. \end{aligned} \tag{2.2}$$

Finally, to describe necessary optimality conditions for problem (1.2), it is standard to use the following partial calmness concept [81]:

Definition 2.1. *Let $(\bar{x}, \bar{y})$ be a local optimal solution of problem (1.2). This problem is partially calm at $(\bar{x}, \bar{y})$ if there exists $\lambda > 0$ and a neighbourhood U of $(\bar{x}, \bar{y}, 0)$ such that*

$$F(x, y) - F(\bar{x}, \bar{y}) + \lambda|u| \geqslant 0, \ \ \forall (x, y, u) \in U : \ G(x, y) \leqslant 0, \ g(x, y) \leqslant 0, \ f(x, y) - \varphi(x) - u = 0.$$

According to [81, Proposition 3.3], problem (1.2)–(1.3) being partially calm at one of its local optimal solution $(\bar{x}, \bar{y})$ is equivalent to the existence of a parameter $\lambda > 0$ such that the point $(\bar{x}, \bar{y})$ is also a local optimal solution of problem

$$\min_{x, y} F(x, y) + \lambda(f(x, y) - \varphi(x)) \ \text{ s.t. } \ G(x, y) \leqslant 0, \ \ g(x, y) \leqslant 0. \tag{2.3}$$

Partial calmness has been the main tool to derive optimality conditions for (1.1) from the perspective of the optimal value function; see, e.g., [20, 23, 24, 81]. It is automatically satisfied if the upper-level feasible set is independent from y and the lower-level problem is defined by

$$f(x, y) := c^\top y \quad \text{and} \quad g(x, y) := A(x) + By,$$

where $A : \mathbb{R}^n \to \mathbb{R}^p$, $c \in \mathbb{R}^m$, and $B \in \mathbb{R}^{p \times m}$. More generally, various sufficient conditions ensuring that partial calmness holds have been studied in the literature; see [81] for the seminal work on the subject. More recently, the paper [52] has revisited the condition, proposed a fresh perspective, and established new dual-type sufficient conditions for partial calmness to hold.

It is clear that problem (2.3) is a penalization of (1.2) only w.r.t. the constraint $f(x, y) - \varphi(x) \leqslant 0$ with penalty parameter λ. Hence, the problem is a usually labelled as a *partial exact penalization* of problem (1.2)–(1.3). With this reformulation it is now reasonable to assume standard constraint qualifications to derive optimality conditions. Based on this, we have the following result, see, e.g., [24, 20, 23, 81], based on a particular estimate of the subdifferential of φ (1.3).

Theorem 2.2. *Let $(\bar{x}, \bar{y})$ be a local optimal solution to (1.2)–(1.3), where all functions are assumed to be differentiable, φ is finite around $\bar{x}$ and the lower-level problem is fully convex. Further assume that the problem is partially calm at $(\bar{x}, \bar{y})$, the lower-level regularity is satisfied at $(\bar{x}, \bar{y})$ and upper-level regularity holds at $\bar{x}$. Then there exist $\lambda > 0$, and Lagrange multipliers u, v, and w such that*

$$\nabla F(\bar{x}, \bar{y}) + \nabla g(\bar{x}, \bar{y})^T (u - \lambda w) + \nabla G(\bar{x}, \bar{y})^T v = 0, \tag{2.4}$$

$$\nabla_y f(\bar{x}, \bar{y}) + \nabla_y g(\bar{x}, \bar{y})^T w = 0, \tag{2.5}$$

$$u \geqslant 0, \ \ g(\bar{x}, \bar{y}) \leqslant 0, \ \ u^T g(\bar{x}, \bar{y}) = 0, \tag{2.6}$$

$$v \geqslant 0, \ \ G(\bar{x}, \bar{y}) \leqslant 0, \ \ v^T G(\bar{x}, \bar{y}) = 0, \tag{2.7}$$

$$w \geqslant 0, \ \ g(\bar{x}, \bar{y}) \leqslant 0, \ \ w^T g(\bar{x}, \bar{y}) = 0. \tag{2.8}$$

Remark 2.3. There are important classes of functions that satisfy the full convexity assumption imposed on the lower-level problem in Theorem 2.2; cf. [48]. However, when it is not possible to guarantee that this assumption is satisfied, there are at least two alternative scenarios to obtain the same optimality conditions. The first is to replace the full convexity assumption by the *inner semicontinuity* of the optimal solution set-valued mapping S (1.1). Secondly, note that a much weaker qualification condition known as *inner semicompactness* can also be used here. However, under the latter assumption, it will additionally be required to have $S(\bar{x}) = \{\bar{y}\}$ in order to arrive at the optimality conditions (2.4)–(2.8). The concept of inner semicontinuity (resp. semicompactness) of S is closely related to the lower semicontinuity (resp. upper semicontinuity) of set-valued mappings; for more details on these notions and their ramifications on bilevel programs, see [20, 23, 24].

Remark 2.4. For the result above it is typical to assume that upper-level constraint does not depend on lower-level y. However, it has been shown in [80] that introducing optimality conditions with $G(x, y)$ is mathematically valid. Due to the nature of the problem one would normally have $G(x)$ depending only on upper-level variable, and hence $\nabla_y G(x, y) = 0$ in (2.4). We present more general result with $G(x, y)$ to cover the possibility of G depending on y

Depending on the assumptions made, we can obtain optimality conditions different from the above. The details of different stationarity concepts can be found in the references provided in the remark above, as well as in [83]. Weaker assumptions will typically lead to more general conditions. However, making stronger assumptions allows us to obtain systems that are easier to handle. For instance, it is harder to deal with more general conditions introduced in [24, Theorem 3.5] or [20, Theorem 3.1] because of the presence of the convex hull in the corresponding estimate of the subdifferential of φ [24, 20, 23, 81]. The other advantage of (2.4)-(2.8) is that, unlike the system studied in [31], these conditions do not require to introduce a new lower-level variable.

The above optimality conditions involve the presence of complementarity conditions (2.6)-(2.8), which result from inequality constraints present in (1.2)–(1.3). In order to reformulate the complementarity conditions in the form of a system of equations, we are going use the concept of NCP-functions; see, e.g., [68]. The function $\phi : \mathbb{R}^2 \to \mathbb{R}$ is said to be a NCP-function if we have

$$\phi(a, b) = 0 \iff a \geqslant 0, \ b \geqslant 0, \ ab = 0.$$

In this paper, we use $\phi(a, b) := \sqrt{a^2 + b^2} - a - b$, known as the *Fischer-Burmeister* function [29]. This leads to the reformulation of the optimality conditions (2.4)–(2.8) into the system of equations:

$$\Upsilon^\lambda(z) := \begin{pmatrix} \nabla_x F(x,y) + \nabla_x g(x,y)^T(u - \lambda w) + \nabla_x G(x,y)^T v \\ \nabla_y F(x,y) + \nabla_y g(x,y)^T(u - \lambda w) + \nabla_y G(x,y)^T v \\ \nabla_y f(x,y) + \nabla_y g(x,y)^T w \\ \sqrt{u^2 + g(x,y)^2} - u + g(x,y) \\ \sqrt{v^2 + G(x,y)^2} - v + G(x,y) \\ \sqrt{w^2 + g(x,y)^2} - w + g(x,y) \end{pmatrix} = 0, \tag{2.9}$$

where we have $z := (x, y, u, v, w)$ and

$$\sqrt{u^2 + g(x,y)^2} - u + g(x,y) := \begin{pmatrix} \sqrt{u_1^2 + g_1(x,y)^2} - u_1 + g_1(x,y) \\ \vdots \\ \sqrt{u_p^2 + g_p(x,y)^2} - u_p + g_p(x,y) \end{pmatrix}. \tag{2.10}$$

$\sqrt{v^2 + G(x,y)^2} - v + G(x,y)$ and $\sqrt{w^2 + g(x,y)^2} - w + g(x,y)$ are defined as in (2.10). The superscript λ is used to emphasize the fact that this number is a parameter and not a variable for equation (2.9). One can easily check that this system consists of $n + 2m + p + q + p$ real-valued equations with $n + m + p + q + p$ variables. Clearly, this means that (2.9) is an over-determined system and the Jacobian of $\Upsilon^\lambda(z)$, where it exists, is a non-square matrix.

3 GAUSS-NEWTON-TYPE METHODS UNDER STRICT COMPLEMENTARITY

To solve equation (2.9), we use a Gauss-Newton-type method, as the system is over-determined. Hence, it is necessary to compute the Jacobian of $\Upsilon^\lambda(z)$ (2.9). However, the function is not differentiable at any point where one of the pairs

$$(u_i,\ g_i(x,y)),\ i=1,\dots,p,\ \ (v_j,\ G_j(x,y)),\ j=1,\dots,q,\ \ \text{and}\ \ (w_i,\ g_i(x,y)),\ i=1,\dots,p$$

vanishes. To avoid this situation, we assume throughout this section that the strict complementarity condition holds:

Assumption 3.1. The strict complementarity condition holds at (x,y,u,v,w) if $(u_i,\ g_i(x,y)) \neq 0$ and $(w_i,\ g_i(x,y)) \neq 0$ for all $i=1,\dots,p$ and $(v_j,\ G_j(x,y)) \neq 0$ for all $j=1,\dots,q$.

Under this assumption, the Jacobian of Υ^λ is well-defined everywhere and hence, the *Gauss-Newton step* to solve equation (2.9) can be defined as

$$d^k = -(\nabla\Upsilon^\lambda(z^k)^T\nabla\Upsilon^\lambda(z^k))^{-1}\nabla\Upsilon^\lambda(z^k)^T\Upsilon^\lambda(z^k), \tag{3.1}$$

provided that the involved inverse matrix exists; see, e.g., [35, 55]. This leads to the following algorithm tailored to equation (2.9):

Algorithm 3.2. *Gauss-Newton Method for Bilevel Optimization*

```
Step 0: Choose $\lambda > 0$, $\epsilon > 0$, $K > 0$, $z^0 := (x^0, y^0, u^0, v^0, w^0)$, and set $k := 0$.
Step 1: If $\|\Upsilon^\lambda(z^k)\| < \epsilon$ or $k \geqslant K$, then stop.
Step 2: Calculate Jacobian $\nabla\Upsilon^\lambda(z^k)$ and compute the direction $d^k$ using (3.1).
Step 3: Set $z^{k+1} := z^k + d^k$, $k := k+1$, and go to Step 1.
```

In Algorithm 3.2, ϵ denotes the tolerance and K is the maximum number of iterations. Additionally, it is possible to implement a line search technique to control the step size at each iteration, as done in [31] for a Newton-type method. The impact of line search will be analyzed in a separate work, as our main aim here is to study the core step (3.1) of the method. It is clear from (3.1) that for Algorithm 3.2 to be well-defined, the matrix $\nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$ needs to be non-singular. In the next subsection, we provide tractable conditions ensuring that this is possible.

3.1 Nonsingularity of $\nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$ and Convergence

We begin this subsection by noting that as the Jacobian $\nabla\Upsilon^\lambda(z)$ is a $(n+2m+2p+q)\times(n+m+2p+q)$ matrix with m more rows than columns, and linear independence of its columns ensures that $\nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$ is non-singular. It therefore suffices for us to provide conditions guarantying the linear independence of the columns of $\nabla\Upsilon^\lambda(z)$.

To present the Jacobian of the system (2.9) in a compact form, let the *upper-level* and *lower-level Lagrangian* functions be defined by

$$L^\lambda(z) := F(x,y) + g(x,y)^T(u-\lambda w) + G(x,y)^T v \ \text{ and } \ \mathcal{L}(z) := f(x,y) + g(x,y)^T w,$$

respectively. As we need the derivatives of these functions in the sequel, we denote the appropriate versions of the Hessian matrices of L^λ and $\mathcal{L}$, w.r.t. (x,y), by

$$\nabla^2 L^\lambda(z) := \begin{bmatrix} \nabla^2_{xx}L^\lambda(z) & \nabla^2_{yx}L^\lambda(z) \\ \nabla^2_{xy}L^\lambda(z) & \nabla^2_{yy}L^\lambda(z) \end{bmatrix} \quad \text{and} \quad \nabla(\nabla_y\mathcal{L}(z)) := \begin{bmatrix} \nabla^2_{xy}\mathcal{L}(z) & \nabla^2_{yy}\mathcal{L}(z) \end{bmatrix} \tag{3.2}$$

respectively. Furthermore, by denoting $\nabla g(x,y)^T := \begin{bmatrix} \nabla_x g(x,y)^T \\ \nabla_y g(x,y)^T \end{bmatrix}$ and $\nabla G(x,y)^T := \begin{bmatrix} \nabla_x G(x,y)^T \\ \nabla_y G(x,y)^T \end{bmatrix}$, we can easily check that the Jacobian of $\Upsilon^\lambda(z)$ w.r.t. z can be written as

$$\nabla\Upsilon^\lambda(z) = \begin{bmatrix} \nabla^2 L^\lambda(z) & \nabla g(x,y)^T & \nabla G(x,y)^T & -\lambda\nabla g(x,y)^T \\ \nabla(\nabla_y\mathcal{L}(z)) & O & O & \nabla_y g(x,y)^T \\ \mathcal{T}\nabla g(x,y) & \Gamma & O & O \\ \mathcal{A}\nabla G(x,y) & O & \mathcal{B} & O \\ \Theta\nabla g(x,y) & O & O & \mathcal{K} \end{bmatrix} \tag{3.3}$$

with $\mathcal{T} := \text{diag}\{\tau_1, \ldots, \tau_p\}$, $\Gamma := \text{diag}\{\gamma_1, \ldots, \gamma_p\}$, $\mathcal{A} := \text{diag}\{\alpha_1, \ldots, \alpha_q\}$, $\mathcal{B} := \text{diag}\{\beta_1, \ldots, \beta_q\}$, $\Theta := \text{diag}\{\theta_1, \ldots, \theta_p\}$, and $\mathcal{K} := \text{diag}\{\kappa_1, \ldots, \kappa_p\}$, where the pair (τ_j, γ_j), $j := 1, \ldots p$ is defined by

$$\tau_j := \frac{g_j(x,y)}{\sqrt{u_j^2 + g_j(x,y)^2}} + 1 \text{ and } \gamma_j := \frac{u_j}{\sqrt{u_j^2 + g_j(x,y)^2}} - 1, \text{ for } j = 1, \ldots p. \tag{3.4}$$

The pairs (α_j, β_j), $j = 1, \ldots, q$ and (θ_j, κ_j), $j = 1, \ldots, p$ are defined similarly in terms of $(G_j(x,y), v_j)$, $j = 1, \ldots, q$ and $(g_j(x,y), w_j)$, $j = 1, \ldots, p$, respectively. Additionally, analogously to the lower-level (resp. upper-level) regularity condition in (2.1) (resp. (2.2)), we will need the lower-level (resp. upper-level) linear independence constraint qualification denoted by LLICQ (resp. ULICQ) that will be said to hold at a point $(\bar{x}, \bar{y})$ if the family of gradients

$$\{\nabla_y g_i(\bar{x},\bar{y}),\ i \in I_g(\bar{x},\bar{y})\} \ \left(\text{resp. } \left\{\nabla g_i(\bar{x},\bar{y}),\ i \in I_g(\bar{x},\bar{y}),\ \nabla G_j(\bar{x},\bar{y}),\ j \in I_G(\bar{x},\bar{y})\right\}\right) \tag{3.5}$$

is linearly independent. Finally, to construct the second order condition necessary in the formulation of the next result, we also need the following index sets:

$$\begin{array}{lllll} \nu^1 & := & \nu^1(\bar{x},\bar{y},\bar{u}) & := & \left\{j \mid \bar{u}_j > 0,\ g_j(\bar{x},\bar{y}) = 0\right\}, \\ \nu^2 & := & \nu^2(\bar{x},\bar{y},\bar{v}) & := & \left\{j \mid \bar{v}_j > 0,\ G_j(\bar{x},\bar{y}) = 0\right\}, \\ \nu^3 & := & \nu^3(\bar{x},\bar{y},\bar{w}) & := & \left\{j \mid \bar{w}_j > 0,\ g_j(\bar{x},\bar{y}) = 0\right\}. \end{array}$$

We use these sets to introduce the following cone of feasible directions:

$$\mathcal{C}(\bar{x},\bar{y}) := \left\{ d \,\middle|\, \begin{array}{l} \nabla g_j(\bar{x},\bar{y})^\top d = 0 \text{ for } j \in \nu^1 \\ \nabla G_j(\bar{x},\bar{y})^\top d = 0 \text{ for } j \in \nu^2 \\ \nabla g_j(\bar{x},\bar{y})^\top d = 0 \text{ for } j \in \nu^3 \end{array} \right\}.$$

Theorem 3.3. *Let the point $\bar{z} = (\bar{x},\bar{y},\bar{u},\bar{v},\bar{w})$ satisfy (2.9) for some $\lambda > 0$. Suppose that Assumption 3.1 holds at $(\bar{x},\bar{y},\bar{u},\bar{v},\bar{w})$, while LLICQ and ULICQ are satisfied at $(\bar{x},\bar{y})$. Furthermore, suppose that we have*

$$d^\top \nabla^2 L^\lambda(\bar{z}) d > 0 \quad \textit{for all} \quad d \in \mathcal{C}(\bar{x},\bar{y}) \setminus \{0\}. \tag{3.6}$$

Then, the columns of the Jacobian matrix $\nabla \Upsilon^\lambda(\bar{z})$ are linearly independent.

Proof. Consider an arbitrary vector $d := (d_1^\top, d_2^\top, d_3^\top, d_4^\top)^\top$ such that $\nabla\Upsilon^\lambda(\bar{z})d = 0$ with the components $d_1 \in \mathbb{R}^{n+m}$, $d_2 \in \mathbb{R}^p$, $d_3 \in \mathbb{R}^q$, and $d_4 \in \mathbb{R}^p$. Then we have

$$\nabla^2 L^\lambda(\bar{z})d_1 + \nabla g(\bar{x},\bar{y})^\top d_2 + \nabla G(\bar{x},\bar{y})^\top d_3 - \lambda \nabla g(\bar{x},\bar{y})^\top d_4 = 0, \tag{3.7}$$

$$\mathcal{T}\nabla g(\bar{x},\bar{y})d_1 + \Gamma d_2 = 0, \tag{3.8}$$

$$\mathcal{A}\nabla G(\bar{x},\bar{y})d_1 + \mathcal{B}d_3 = 0, \tag{3.9}$$

$$\Theta\nabla g(\bar{x},\bar{y})d_1 + \mathcal{K}d_4 = 0, \tag{3.10}$$

$$\nabla(\nabla_y \mathcal{L}(\bar{z}))d_1 + \nabla_y g(\bar{x},\bar{y})^\top d_4 = 0. \tag{3.11}$$

On the other hand, it obviously follows from (3.4) that

$$(\tau_j - 1)^2 + (\gamma_j + 1)^2 = 1 \quad \text{for} \quad j = 1, \ldots p. \tag{3.12}$$

Hence, the indices of the pair (τ, γ) satisfying (3.12) can be partitioned into the sets

$$P_1 := \{j : \tau_j > 0,\ \gamma_j < 0\}, \quad P_2 := \{j : \tau_j = 0\}, \quad \text{and} \quad P_3 := \{j : \gamma_j = 0\}.$$

Similarly, define index sets Q_1, Q_2, and Q_3 for the pair (α, β) and T_1, T_2, and T_3 for (θ, κ). Next, consider the following componentwise description of (3.8), (3.9), and (3.10),

$$\tau_j \nabla g_j(\bar{x},\bar{y})^\top d_1 + \gamma_j d_{2_j} = 0 \quad \text{for } j = 1, ..., p, \tag{3.13}$$

$$\alpha_j \nabla G_j(\bar{x},\bar{y})^\top d_1 + \beta_j d_{3_j} = 0 \quad \text{for } j = 1, ..., q, \tag{3.14}$$

$$\theta_j \nabla g_j(\bar{x},\bar{y})^\top d_1 + \kappa_j d_{4_j} = 0 \quad \text{for } j = 1, ..., p. \tag{3.15}$$

For $j \in P_2$, equation (3.13) becomes $\gamma_j d_{2_j} = 0$. Additionally, it follows from (3.12) that for $j \in P_2$, $\gamma_j \neq 0$. Hence $d_{2_j} = 0$ for $j \in P_2$. For $j \in P_3$, (3.13) leads to $\tau_j \nabla g_j(\bar{x},\bar{y})^T d_1 = 0$, which due to the property above translates into $\nabla g_j(\bar{x},\bar{y})^T d_1 = 0$, as $\tau_j \neq 0$. Finally, for $j \in P_1$ equation (3.13) takes the form $\nabla g_j(\bar{x},\bar{y})^T d_1 = -\frac{\gamma_j}{\tau_j} d_{2_j}$, where by definition of P_1 we know that $-\frac{\gamma_j}{\tau_j} > 0$. Following the same logic, we respectively have from (3.14) and (3.15) that

$$\begin{array}{lll} d_{3_j} = 0 \text{ for } j \in Q_2, & \nabla G_j(\bar{x},\bar{y})^T d_1 = 0 \text{ for } j \in Q_3, & \nabla G_j(\bar{x},\bar{y})^T d_1 = -\frac{\beta_j}{\alpha_j} d_{3_j} \text{ for } j \in Q_1, \\ d_{4_j} = 0 \text{ for } j \in T_2, & \nabla g_j(\bar{x},\bar{y})^T d_1 = 0 \text{ for } j \in T_3, & \nabla g_j(\bar{x},\bar{y})^T d_1 = -\frac{\kappa_j}{\theta_j} d_{4_j} \text{ for } j \in T_1, \end{array}$$

with $-\frac{\beta_j}{\alpha_j} > 0$ for $j \in Q_1$ and $-\frac{\kappa_j}{\theta_j} > 0$ for $j \in T_1$. Multiplying (3.7) by d_1^T,

$$d_1^T \nabla^2 L^\lambda(\bar{z}) d_1 + d_1^T \nabla g(\bar{x},\bar{y})^T d_2 + d_1^T \nabla G(\bar{x},\bar{y})^T d_3 - \lambda d_1^T \nabla g(\bar{x},\bar{y})^T d_4 = 0. \tag{3.16}$$

Considering the cases defined above, we know that for $j \in P_2$, $j \in Q_2$, and $j \in T_2$, the terms d_{2_j}, d_{3_j}, and d_{4_j}, respectively, disappear. For $j \in P_3$, $j \in Q_3$, and $j \in T_3$, the terms $\nabla g_j(\bar{x},\bar{y})^T d_1$, $\nabla G_j(\bar{x},\bar{y})^T d_1$, and $\nabla g_j(\bar{x},\bar{y})^T d_1$ also vanish. This leads to the equation (3.16) being simplified to

$$d_1^T \nabla^2 L^\lambda(\bar{z}) d_1 + \sum_{j \in P_1} \left(-\frac{\gamma_j}{\tau_j}\right) d_{2_j}^2 + \sum_{j \in Q_1} \left(-\frac{\beta_j}{\alpha_j}\right) d_{3_j}^2 - \lambda \sum_{j \in T_1} \left(-\frac{\kappa_j}{\theta_j}\right) d_{4_j}^2 = 0. \tag{3.17}$$

One can easily check that thanks to Assumption 3.1, the sets P_1, Q_1, and T_1 are empty. Hence, (3.17) reduces to $d_1^T \nabla^2 L^\lambda(\bar{z}) d_1 = 0$. Moreover, one can easily check that $\nu^1 \subseteq P_3$, $\nu^2 \subseteq Q_3$, and $\nu^3 \subseteq T_3$. Hence, $d_1 \in \mathcal{C}(\bar{x},\bar{y})$ and therefore, it follows from (3.6) that $d_1 = 0$.

We have shown that $d_{2_j} = 0$, $d_{3_j} = 0$ and $d_{4_j} = 0$ for $j \in P_2$, $j \in Q_2$ and $j \in T_2$, and $d_1 = 0$. Let us use these results to simplify equations (3.7) and (3.11) as follows

$$\sum_{j \in P_3} \nabla g_j(\bar{x},\bar{y})^T d_{2_j} + \sum_{j \in Q_3} \nabla G_j(\bar{x},\bar{y})^T d_{3_j} - \lambda \sum_{j \in T_3} \nabla g_j(\bar{x},\bar{y})^T d_{4_j} = 0, \tag{3.18}$$

$$\sum_{j \in T_3} \nabla_y g_j(\bar{x},\bar{y})^T d_{4_j} = 0. \tag{3.19}$$

Equation (3.19) implies that $d_{4_j} = 0$ for all $j \in T_3$, given that $T_3 \subseteq I_g(\bar{x},\bar{y})$ and the LLICQ holds at $(\bar{x},\bar{y})$. Then (3.18) becomes

$$\sum_{j \in P_3} \nabla g_j(\bar{x},\bar{y})^T d_{2_j} + \sum_{j \in Q_3} \nabla G_j(\bar{x},\bar{y})^T d_{3_j} = 0,$$

which implies $d_{2_j} = 0$ and $d_{3_j} = 0$ for $j \in P_3$ and $j \in Q_3$, given that the ULICQ holds at $(\bar{x},\bar{y})$. This completes the proof as we have shown that $\nabla \Upsilon^\lambda(\bar{z}) d = 0$ only if $d = 0$. □

Example 3.4. We consider an instance of problem (1.1) taken from the BOLIB Library [85] with

$$\begin{array}{l} F(x,y) := (x-3)^2 + (y-2)^2, \\ f(x,y) := (y-5)^2, \end{array} \quad G(x,y) := \begin{pmatrix} x-8 \\ -x \end{pmatrix}, \quad g(x,y) := \begin{pmatrix} -2x+y-1 \\ x-2y-2 \\ x+2y-14 \end{pmatrix}.$$

The point $\bar{z} = (\bar{x},\bar{y},\bar{u}_1,\bar{u}_2,\bar{u}_3,\bar{v}_1,\bar{v}_2,\bar{w}_1,\bar{w}_2,\bar{w}_3) = (1,3,4\lambda-2,0,0,0,0,4,0,0)$ satisfies equation (2.9) for any $\lambda > 1/2$. Obviously, strict complementarity holds at this point, for $\lambda > 1/2$. and the family of vectors $\{\nabla g_j(\bar{x},\bar{y}), j \in I_g(x,y), \nabla G_j(\bar{x},\bar{y}), j \in I_G(x,y)\}$ is linearly independent, as $I_g(x,y) = \{1\}$, $I_G(x,y) = \emptyset$. It is easy to see that ULICQ holds as $\nabla g_1(\bar{x},\bar{y})^T = (-2,1)^T \neq 0$, and LLICQ holds as $\nabla_y g_1(\bar{x},\bar{y})^T = 1 \neq 0$. Finally, we obviously have that $\nabla^2 L^\lambda(\bar{z}) = 2e$, where e is the identity matrix of $\mathbb{R}^{2\times 2}$, is positive definite. In conclusion, this example shows that all assumptions of Theorem 3.3 can hold for a bilevel program and therefore the Gauss-Newton method in (3.2) is well-defined.

Based on the result above, we can now state the convergence theorem for our Gauss-Newton Algorithm 3.2. To proceed, first note that by implementing the Gauss-Newton method to solve (2.9), leads to a solution to the least-square problem

$$\min_z \Phi^\lambda(z) := \sum_{i=1}^{N+m} \Upsilon_i^\lambda(z)^2, \tag{3.20}$$

where we define $N := n + m + 2p + q$. The direction of the Newton method for problem (3.20) can be written as

$$d^k := -(\nabla\Upsilon^\lambda(z_k)^T\nabla\Upsilon^\lambda(z_k) + T(z_k))^{-1}\nabla\Upsilon^\lambda(z_k)\Upsilon(z_k),$$

where $T(z_k) := \sum_{i=1}^{N} \Upsilon_i^\lambda(z_k)\nabla^2\Upsilon_i^\lambda(z_k)$ is the term that is omitted in the Gauss-Newton direction (3.1). It is well known that the Gauss-Newton method converges with the same rate as Newton method if the term $T(\bar{z})$ is small enough in comparison with the term $\nabla\Upsilon^\lambda(\bar{z})^T\nabla\Upsilon^\lambda(\bar{z})$; see, e.g., [25, 55, 69]. This is the basis of the following convergence result of Algorithm 3.2.

Theorem 3.5. *Let the assumptions in Theorem 3.3 hold at the point $\bar{z} = (\bar{x}, \bar{y}, \bar{u}, \bar{v}, \bar{w})$ (for some $\lambda > 0$), assumed to be a local optimal solution of problem (3.20). Also, let $\{z^k\}$ be a sequence of points generated by Algorithm 3.2 and assume that it converges to $\bar{z}$. Furthermore, suppose that $\nabla^2 L^\lambda$ and $\nabla(\nabla_y)\mathcal{L}$ are well-defined and Lipschitz continuous in a neighbourhood of $\bar{z}$. Then we have*

$$\|z^{k+1} - \bar{z}\| \leqslant \left\|\left(\nabla\Upsilon^\lambda(\bar{z})^T\nabla\Upsilon^\lambda(\bar{z})\right)^{-1}\right\| \|T(\bar{z})\| \, \|z^k - \bar{z}\| + O\left(\|z_k - \bar{z}\|^2\right). \tag{3.21}$$

Proof. Start by recalling that under the assumptions of Theorem 3.3, the matrix $\nabla\Upsilon^\lambda(\bar{z})$ is of full column rank. Hence, it is positive definite. Furthermore, under the strict complementarity condition and well-definedness of the functions $\nabla^2 L^\lambda$ and $\nabla(\nabla_y\mathcal{L})$ near $\bar{z}$, all components of $z \mapsto \Upsilon^\lambda(z)$ and $z \mapsto \nabla\Upsilon^\lambda(z)$ are differentiable near $\bar{z}$. Hence, the term $T(z)$ is well-defined near $\bar{z}$. Furthermore, the local Lipschitz continuity of $\nabla^2 L^\lambda$ and $\nabla(\nabla_y\mathcal{L})$ imply that the same holds for $z \mapsto \nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$ and $z \mapsto T(z)$. Hence, the function $z \mapsto \nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z) + T(z)$ is Lipschitz continuous around $\bar{z}$. Next, note that $z \mapsto \nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$ is differentiable near $\bar{z}$, as the same is satisfied for $z \mapsto \nabla\Upsilon^\lambda(z)$. We also know that $\nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$ is non-singular for all z in some neighbourhood of $\bar{z}$, under the assumptions made in Theorem 3.3. By the inverse function theorem, the latter ensures that $\left(\nabla\Upsilon^\lambda\right)^\top\nabla\Upsilon^\lambda$ is a diffeomorphism, and hence has a differentiable inverse around $\bar{z}$. Thus, the function $z \mapsto \left(\nabla\Upsilon^\lambda(z)^\top\nabla\Upsilon^\lambda(z)\right)^{-1}$ is Lipschitz continuous around the point $\bar{z}$ and hence, inclusion (3.21) follows by the application of [69, Theorem 7.2.2]. □

It is clear from this theorem that the Gauss-Newton method converges quadratically if $T(\bar{z}) = 0$ and Q-Linearly if $T(\bar{z})$ is small relative to $\nabla\Upsilon^\lambda(\bar{z})^T\nabla\Upsilon^\lambda(\bar{z})$. Such properties can be satisfied for small residuals problems and for the problems that are not too nonlinear. For problems with small residuals we have that the components $\Upsilon_i^\lambda(\bar{z})$ are small for all i, which makes the term $T(\bar{z})$ small. For the problems with not too much nonlinearity the components $\nabla^2\Upsilon_i^\lambda(\bar{z})$ are small for all i, which also results in small $T(\bar{z})$. If it turns out that we can obtain an exact solution $\Upsilon^\lambda(\bar{z}) = 0$, then $T(\bar{z}) = 0$, and we have quadratic convergence. It is worth noting that in general we cannot always have $\Upsilon_i^\lambda(\bar{z}) = 0$ for all i as the system is overdetermined, but minimizing the sum of the squares of $\Upsilon_i^\lambda(\bar{z})$ we obtain a solution point $\bar{z}$, at which $\sum_{i=1}^{N+m}(\Upsilon_i^\lambda(z))^2$ is as small as possible. If the problem has small residuals, then small $\Upsilon_i^\lambda(\bar{z})$ are naturally obtained by implementing Algorithm 3.2 as this is designed to minimize $\sum_{i=1}^{N+m}\Upsilon_i^\lambda(z)^2$. In terms of having small components $\nabla^2\Upsilon_i^\lambda(\bar{z})$ we observe that $\nabla^2\Upsilon^\lambda(\bar{z})$ will involve third derivatives of $F(x,y), G(x,y), f(x,y)$ and $g(x,y)$. Hence, if the original problem is not too nonlinear, the Hessian of the system (2.9) should be small. As a result, if there exists a solution with $\Upsilon_i^\lambda(z) \approx 0$ for all i or if the original problem (1.1) is not too nonlinear, then we expect that the Gauss-Newton for Bilevel Programming converges Q-linearly.

The first drawback of Algorithm 3.2 is the requirement of strict complementarity in Assumption 3.1, to help ensure the differentiability of the function Υ^λ. Assumption 3.1 is rather strong; for the test problems used for our numerical experiments in Section 5, it did not hold at the last iteration for at least one value of λ for a total of 54 out of 124 problems considered. If one wants to avoid the *strict complementarity* assumption, one option is to use *smoothing technique* for Fischer-Burmeister function, to be discussed in Section 4. Before we move to this, it is worth mentioning that a second issue faced by our Algorithm 3.2 is the requirement that the matrix $\nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$ is nonsingular at each iteration. To deal with this, one option is to execute a Newton step, where the generalized inverse of $\nabla\Upsilon^\lambda(z)^T\nabla\Upsilon^\lambda(z)$, which always exists, is used. Such an approach is briefly discussed in the next subsection.

3.2 Newton method with Moore-Penrose pseudo inverse

Indeed, one of the most challenging aspects of the Gauss-Newton step in Algorithm 3.2 is the computation of the inverse of the matrix $\nabla\Upsilon^\lambda(z_k)^T\nabla\Upsilon^\lambda(z_k)$, as this quantity might not exist at some iterations. To deal with situations where the inverse of the matrix does not exist, various concepts of generalized inverse have been used in the context of Newton's method; see, e.g., [57] for related details. Although we do not directly compute $\left(\nabla\Upsilon^\lambda(z_k)^T\nabla\Upsilon^\lambda(z_k)\right)^{-1}$ in our implementation of Algorithm 3.2 in Section 5, we would like to compare the *pure* Gauss-Newton-type method presented in the previous subsection with the Newton method using the Moore-Penrose pseudo inverse. Hence, we present the later approach here and its relationship to Algorithm 3.2.

For an arbitrary matrix $A \in \mathbb{R}^{m\times n}$, its *Moore-Penrose pseudo inverse* (see, e.g., [39]) is defined by

$$A^+ := V\Sigma^+U^\top,$$

where $V\Sigma^+U^\top$ represents a singular value decomposition of A, where Σ^+ corresponds to the pseudo-inverse of Σ that can be given by

$$\Sigma^+ = \text{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \dots, \frac{1}{\sigma_r}, 0, \dots, 0\right) \quad \text{with} \quad r = \text{rank}\,(A).$$

if A has full column rank, we have an additional property that

$$A^+ := \left(A^T A\right)^{-1} A^\top.$$

Based on this definition, an iteration of the Newton method with pseudo inverse for system (2.9) can be then stated as

$$z^{k+1} = z^k - \nabla\Upsilon(z^k)^+\Upsilon(z^k). \tag{3.22}$$

We are now going to refer to (3.22) as iteration of the *Pseudo-Newton method*. The Pseudo-Newton method for bilevel programming can be defined in the same fashion as Algorithm 3.2 with the difference that direction would be given by $d^k = -\nabla\Upsilon^\lambda(z^k)^+\Upsilon^\lambda(z^k)$. Clearly, the Pseudo-Newton method is always well-defined, unlike the Gauss-Newton method, and hence, it will produce some result in the case when the Gauss-Newton method diverges [34]. Based on this general behaviour and interplay between the two approaches, we will be comparing them in the numerical section. For details on the convergence of Newton-type methods with pseudo-inverse, the interested reader in referred to [37].

4 SMOOTHING GAUSS-NEWTON METHOD

In this section, we relax the strict complementarity assumption, considering the fact that it often fails for many problems as illustrated in the previous section. However, to ensure the smoothness of the function Υ^λ (2.9), the Fischer-Burmeister function is replaced with the *smoothing Fischer-Burmeister function* (see [42]) defined by

$$\phi^\mu_{g_j}(x,y,u) := \sqrt{u_j^2 + g_j(x,y)^2 + 2\mu} - u_j + g_j(x,y) \quad \text{for} \quad j = 1,\dots,p, \tag{4.1}$$

where the perturbation parameter $\mu > 0$ helps to guarantee its differentiability at points (x, y, u) satisfying $u_j = g_j(x, y) = 0$. It is well-known (see latter references) that

$$\phi^{\mu}_{g_j}(x, y, u) = 0 \iff \left[u_j > 0,\ -g_j(x, y) > 0,\ -u_j g_j(x, y) = \mu\right]. \tag{4.2}$$

The smoothing system of optimality conditions becomes

$$\Upsilon^{\lambda}_{\mu}(z) := \begin{pmatrix} \nabla_x F(x,y) + \nabla_x g(x,y)^T(u - \lambda w) + \nabla_x G(x,y)^T v \\ \nabla_y F(x,y) + \nabla_y g(x,y)^T(u - \lambda w) + \nabla_y G(x,y)^T v \\ \nabla_y f(x,y) + \nabla_y g(x,y)^T w \\ \sqrt{u^2 + g(x,y)^2 + 2\mu} - u + g(x,y) \\ \sqrt{v^2 + G(x,y)^2 + 2\mu} - v + G(x,y) \\ \sqrt{w^2 + g(x,y)^2 + 2\mu} - w + g(x,y) \end{pmatrix} = 0, \tag{4.3}$$

following the convention in (2.10), where μ is a vector of appropriate dimensions with sufficiently small positive elements. Under the assumption that all the functions involved in problem (1.1) are continuously differentiable, Υ^{λ}_{μ} is also a continuously differentiable function for any $\lambda > 0$ and $\mu > 0$. Additionally, we can easily check that

$$\|\Upsilon^{\lambda}_{\mu}(z) - \Upsilon^{\lambda}(z)\| \longrightarrow 0 \quad \text{as} \quad \mu \downarrow 0.$$

Following the smoothing scheme discussed, for example, in [70], our aim is to consider a sequence $\{\mu_k\}$ decreasing to 0 such that equation (2.9) is approximately solved:

$$\Upsilon^{\lambda}_{\mu^k}(z) = 0, \quad k = 0, 1, \ldots$$

for a fixed value of $\lambda > 0$. Hence, we consider the following algorithm for system (4.3):

Algorithm 4.1. *Smoothing Gauss-Newton Method for Bilevel Optimization*

Step 0: *Choose* $\lambda > 0$, $\mu_0 \in (0, 1)$, $z^0 := (x^0, y^0, u^0, v^0, w^0)$, $\epsilon > 0$, $K > 0$, *set* $k := 0$.
Step 1: *If* $\|\Upsilon^{\lambda}(z^k)\| < \epsilon$ *or* $k \geqslant K$, *then stop.*
Step 2: *Calculate Jacobian* $\nabla\Upsilon^{\lambda}_{\mu_k}(z^k)$ *and find the direction*

$$d^k = -(\nabla\Upsilon^{\lambda}_{\mu_k}(z^k)^T \nabla\Upsilon^{\lambda}_{\mu_k}(z^k))^{-1} \nabla\Upsilon^{\lambda}_{\mu_k}(z^k)^T \Upsilon^{\lambda}(z^k). \tag{4.4}$$

Step 3: *Calculate* $z^{k+1} = z^k + d^k$.
Step 4: *Update* $\mu_{k+1} = \mu_k^{k+1}$.
Step 5: *Set* $k := k + 1$ *and go to Step 1.*

To implement Algorithm 4.1 numerically, we compute the direction by solving

$$\nabla\Upsilon^{\lambda}_{\mu_k}(z^k)^T \nabla\Upsilon^{\lambda}_{\mu_k}(z^k) d^k = -\nabla\Upsilon^{\lambda}_{\mu_k}(z^k)^T \Upsilon^{\lambda}(z^k). \tag{4.5}$$

The Pseudo-Newton algorithm for the smoothed optimality conditions (4.3) will be the same as Algorithm 4.1 apart from **Step 2**, where the corresponding direction is given by

$$d^k = -\nabla\Upsilon^{\lambda}_{\mu_k}(z^k)^{+} \Upsilon^{\lambda}(z^k).$$

Another way to deal with the non-differentiability of the Fischer-Burmeister NCP-function is to introduce a generalized *generalized Jacobian* concept for the system (2.9). A semismooth Newton-type method for bilevel optimization following this type of approach is developed in [31]. However, we will not consider this approach here.

Similarly to (3.4)–(3.3), we introduce the matrices $\mathcal{T}^{\mu}$, Γ^{μ}, $\mathcal{A}^{\mu}$, $\mathcal{B}^{\mu}$, Θ^{μ}, and $\mathcal{K}^{\mu}$, where for instance, the pair $(\mathcal{T}^{\mu}, \Gamma^{\mu})$ is defined by $\mathcal{T}^{\mu} := \mathrm{diag}\,\{\tau^{\mu}_1, .., \tau^{\mu}_p\}$ and $\Gamma^{\mu} := \mathrm{diag}\,\{\gamma^{\mu}_1, .., \gamma^{\mu}_p\}$ with

$$\tau^{\mu}_j := \frac{g_j(x,y)}{\sqrt{u_j^2 + g_j(x,y)^2 + 2\mu}} + 1 \ \text{ and } \ \gamma^{\mu}_j := \frac{u_j}{\sqrt{u_j^2 + g_j(x,y)^2 + 2\mu}} - 1, \ j = 1, \ldots p. \tag{4.6}$$

With this notation, we can easily check that for $\lambda > 0$ and $\mu > 0$, the Jacobian of Υ^λ_μ is

$$\nabla\Upsilon^\lambda_\mu(z) = \begin{bmatrix} \nabla^2 L^\lambda(z) & \nabla g(x,y)^T & \nabla G(x,y)^T & -\lambda\nabla g(x,y)^T \\ \nabla(\nabla_y \mathcal{L}(z)) & O & O & \nabla_y g(x,y)^T \\ \mathcal{T}^\mu \nabla g(x,y) & \Gamma^\mu & O & O \\ \mathcal{A}^\mu \nabla G(x,y) & O & \mathcal{B}^\mu & O \\ \Theta^\mu \nabla g(x,y) & O & O & \mathcal{K}^\mu \end{bmatrix} \tag{4.7}$$

The fundamental difference between the framework here and the one in the previous section is that for the pair $(\tau^\mu_j, \gamma^\mu_j)$, $j = 1,\dots,p$, for instance, we have the strict inequalities

$$(\tau^\mu_j - 1)^2 + (\gamma^\mu_j + 1)^2 < 1, \;\; j = 1,\dots p$$

instead of equalities in the context of (τ_j, γ_j), $j = 1,\dots,p$ (3.4). The next lemma illustrates a further difference between the new coefficients in this section and the ones in (3.4).

Lemma 4.2. *For a point $z := (x,y,u,v,w)$ such that $\Upsilon^\lambda_\mu(z) = 0$ with $\lambda > 0$ and $\mu > 0$, it holds that*

$$\begin{aligned} \tau^\mu_j > 0, \quad \gamma^\mu_j < 0, \quad j = 1,\dots,p, \\ \alpha^\mu_j > 0, \quad \beta^\mu_j < 0, \quad j = 1,\dots,q, \\ \theta^\mu_j > 0, \quad \kappa^\mu_j < 0, \quad j = 1,\dots,p. \end{aligned}$$

Proof. We prove that $\tau^\mu_j > 0$ and $\gamma^\mu_j < 0$ for $j = 1,\dots,p$; the other cases can be done similarly. For $j = 1,\dots,p$, it follows from (4.2) that $g_j(x,y) = -\frac{\mu}{u_j}$. Hence, we can rewrite τ^μ_j and γ^μ_j as

$$\tau^\mu_j = 1 - \frac{\mu}{u_j\sqrt{u_j^2 + \frac{\mu^2}{u_j^2} + 2\mu}} \quad \text{and} \quad \gamma^\mu_j = \frac{u_j}{\sqrt{u_j^2 + \frac{\mu^2}{u_j^2} + 2\mu}} - 1, \tag{4.8}$$

respectively. Next, we consider the following three scenarios:

Case 1 Suppose that $u_j = \mu$. Substituting this value into (4.8), we arrive at

$$\tau^\mu_j = 1 - \frac{1}{\mu + 1} > 0 \;\text{ and }\; \gamma^\mu_j = \frac{\mu}{\mu+1} - 1 < 0 \;\text{ as }\; \mu > 0.$$

Case 2 Suppose that $u_j = \mu + \delta$ for some $\delta > 0$ and substituting this in (4.8) leads to

$$\tau^\mu_j = 1 - \frac{\mu}{(\mu+\delta)\sqrt{(\mu+\delta)^2 + \frac{\mu^2}{(\mu+\delta)^2} + 2\mu}} = 1 - \frac{1}{\sqrt{\frac{(\mu+\delta)^4}{\mu^2} + 1 + 2\frac{(\mu+\delta)^2}{\mu}}} > 0,$$

$$\gamma^\mu_j = \frac{\mu+\delta}{\sqrt{(\mu+\delta)^2 + \frac{\mu^2}{(\mu+\delta)^2} + 2\mu}} - 1 = \frac{1}{\sqrt{1 + \frac{\mu^2}{(\mu+\delta)^4} + 2\frac{\mu}{(\mu+\delta)^2}}} - 1 < 0.$$

Case 3 Finally, suppose that $u_j = \mu - \delta$ for some $\delta > 0$. Then substituting this in (4.8),

$$\tau^\mu_j = 1 - \frac{\mu}{(\mu-\delta)\sqrt{(\mu-\delta)^2 + \frac{\mu^2}{(\mu-\delta)^2} + 2\mu}} = 1 - \frac{1}{\sqrt{\frac{(\mu-\delta)^4}{\mu^2} + 1 + 2\frac{(\mu-\delta)^2}{\mu}}} > 0,$$

$$\gamma^\mu_j = \frac{\mu-\delta}{\sqrt{(\mu-\delta)^2 + \frac{\mu^2}{(\mu-\delta)^2} + 2\mu}} - 1 = \frac{1}{\sqrt{1 + \frac{\mu^2}{(\mu-\delta)^4} + 2\frac{\mu}{(\mu-\delta)^2}}} - 1 < 0.$$

Note that $u_j = \mu - \delta > 0$ for in the third case, which can be used to ensure that $\mu - \delta = \sqrt{(\mu-\delta)^2}$. □

Next, we use this lemma to provide a condition ensuring that the matrix $\nabla\Upsilon^\lambda_\mu(z)^T\nabla\Upsilon^\lambda_\mu(z)$ is nonsingular. This will allow the smoothed Gauss-Newton step (4.4) to be well-defined. As in the previous section, it suffices to show that the columns of $\nabla\Upsilon^\lambda_\mu(\bar{z})$ are linearly independent.

Theorem 4.3. *For a point $\bar{z} := (\bar{x}, \bar{y}, \bar{u}, \bar{v}, \bar{w})$ verifying (4.3) for some $\mu > 0$ and $0 < \lambda < \frac{\kappa_j^\mu}{\theta_j^\mu}\frac{\tau_j^\mu}{\gamma_j^\mu}$, $j = 1, \ldots, p$, suppose that $\nabla^2 L^\lambda(\bar{z})$ is positive definite. Then, the columns of the matrix $\nabla \Upsilon_\mu^\lambda(\bar{z})$ are linearly independent.*

Proof. Similarly to the proof of Theorem 3.3, $\nabla \Upsilon_\mu^\lambda(\bar{z}) \left(d_1^\top, d_2^\top, d_3^\top, d_4^\top\right)^\top = 0$ is equivalent to

$$\nabla^2 L^\lambda(\bar{z}) d_1 + \nabla g(\bar{x}, \bar{y})^\top d_2 + \nabla G(\bar{x}, \bar{y})^\top d_3 - \lambda \nabla g(\bar{x}, \bar{y})^\top d_4 = 0, \tag{4.9}$$

$$\tau_j^\mu \nabla g_j(\bar{x}, \bar{y})^\top d_1 + \gamma_j^\mu d_{2_j} = 0, \tag{4.10}$$

$$\alpha_j^\mu \nabla G_j(\bar{x}, \bar{y})^\top d_1 + \beta_j^\mu d_{3_j} = 0, \tag{4.11}$$

$$\theta_j^\mu \nabla g_j(\bar{x}, \bar{y})^\top d_1 + \kappa_j^\mu d_{4_j} = 0, \tag{4.12}$$

$$\nabla(\nabla_y \mathcal{L}(\bar{z})) d_1 + \nabla_y g(\bar{x}, \bar{y})^\top d_4 = 0, \tag{4.13}$$

where $j = 1, ..., p$ in (4.10) and (4.12), while $j = 1, ..., q$ in (4.11). Thanks to Lemma 4.2, we can rewrite equations (4.10), (4.11), and (4.12) as

$$\nabla g_j(\bar{x}, \bar{y})^\top d_1 = -\frac{\gamma_j^\mu}{\tau_j^\mu} d_{2_j}, \quad \nabla G_j(\bar{x}, \bar{y})^\top d_1 = -\frac{\beta_j^\mu}{\alpha_j^\mu} d_{3_j}, \quad \text{and} \quad \nabla g_j(\bar{x}, \bar{y})^\top d_1 = -\frac{\kappa_j^\mu}{\theta_j^\mu} d_{4_j}, \tag{4.14}$$

respectively, with $-\frac{\gamma_j^\mu}{\tau_j^\mu} > 0$, $-\frac{\beta_j^\mu}{\alpha_j^\mu} > 0$, and $-\frac{\kappa_j^\mu}{\theta_j^\mu} > 0$. Now, let us multiply (4.9) by $d_1^\top$:

$$d_1^\top \nabla^2 L^\lambda(\bar{z}) d_1 + d_1^\top \nabla g(\bar{x}, \bar{y})^\top d_2 + d_1^\top \nabla G(\bar{x}, \bar{y})^\top d_3 - \lambda d_1^\top \nabla g(\bar{x}, \bar{y})^\top d_4 = 0. \tag{4.15}$$

Using the results above, the equation (4.15) can be written as

$$d_1^\top \nabla^2 L^\lambda(\bar{z}) d_1 + \sum_{j=1}^p \left(-\frac{\gamma_j^\mu}{\tau_j^\mu}\right) d_{2_j}^2 + \sum_{j=1}^q \left(-\frac{\beta_j^\mu}{\alpha_j^\mu}\right) d_{3_j}^2 - \lambda \sum_{j=1}^p \left(-\frac{\kappa_j^\mu}{\theta_j^\mu}\right) d_{4_j}^2 = 0. \tag{4.16}$$

Furthermore, it follows from the first and last items of (4.14) that

$$d_{4_j} = -\frac{\theta_j^\mu}{\kappa_j^\mu} \nabla g_j(\bar{x}, \bar{y})^\top d_1 = \frac{\theta_j^\mu \gamma_j^\mu}{\kappa_j^\mu \tau_j^\mu} d_{2_j}. \tag{4.17}$$

Substituting (4.17) into (4.16)

$$d_1^\top \nabla^2 L^\lambda(\bar{z}) d_1 + \sum_{j=1}^p \left(-\frac{\gamma_j^\mu}{\tau_j^\mu}\right) d_{2_j}^2 + \sum_{j=1}^q \left(-\frac{\beta_j^\mu}{\alpha_j^\mu}\right) d_{3_j}^2 - \lambda \sum_{j=1}^p \left(-\frac{\kappa_j^\mu}{\theta_j^\mu}\right) \left(\frac{\theta_j^\mu \gamma_j^\mu}{\kappa_j^\mu \tau_j^\mu}\right)^2 d_{2_j}^2 = 0. \tag{4.18}$$

Rearranging this equation, we get

$$d_1^\top \nabla^2 L^\lambda(\bar{z}) d_1 + \sum_{j=1}^q \left(-\frac{\beta_j^\mu}{\alpha_j^\mu}\right) d_{3_j}^2 + \sum_{j=1}^p \left(1 - \lambda \frac{\theta_j^\mu}{\kappa_j^\mu}\frac{\gamma_j^\mu}{\tau_j^\mu}\right) \left(-\frac{\gamma_j^\mu}{\tau_j^\mu}\right) d_{2_j}^2 = 0. \tag{4.19}$$

Then, with Lemma 4.2, and under the assumptions that $\nabla^2 L^\lambda(\bar{z})$ is positive definite and $\lambda < \frac{\kappa_j^\mu}{\theta_j^\mu}\frac{\tau_j^\mu}{\gamma_j^\mu}$ for $j = 1, \ldots, p$, equation (4.19) is the sum of non-negative terms, which can only be a sum of zeros if all components of d_1, d_2 and d_3 are zeros. Since all components of d_2 are zeros, we can look back to (4.17) or (4.12) to deduce that $d_{4_j} = 0$ for $j = 1, \ldots p$, completing the proof. □

It is important to note that the assumption that $\lambda < \frac{\kappa_j^\mu}{\theta_j^\mu}\frac{\tau_j^\mu}{\gamma_j^\mu}$ does not necessarily conflict with the requirement that λ be strictly positive, as due to Lemma 4.2, we have $\frac{\kappa_j^\mu}{\theta_j^\mu}\frac{\tau_j^\mu}{\gamma_j^\mu} > 0$. In Subsection 5.5, a numerical analysis of this condition is conducted. Next, we provide an example of bilevel program where the assumptions made in the Theorem 4.3 are satisfied.

Example 4.4. Consider the instance of problem (1.1) from the BOLIB Library [85] with

$$\begin{array}{l} F(x,y) := x^2 + (y_1 + y_2)^2, \\ f(x,y) := y_1, \end{array} \quad G(x,y) := -x + 0.5, \quad g(x,y) := \begin{pmatrix} -x - y_1 - y_2 + 1 \\ -y \end{pmatrix}.$$

The point $\bar{z} = (\bar{x}, \bar{y}_1, \bar{y}_2, \bar{u}_1, \bar{u}_2, \bar{u}_3, \bar{v}, \bar{w}_1, \bar{w}_2, \bar{w}_3) = (0.5, 0, 0.5, 1, \lambda, 0, 0, 0, 1, 0)$ satisfies equation (2.9) for any $\lambda > 0$. Strict complementarity does not hold at this point as $(\bar{v}, G(\bar{x},\bar{y})) = (0,0)$ and $(\bar{w}_1, g_1(\bar{x},\bar{y})) = (0,0)$. We observe that $\nabla^2 L^\lambda(\bar{z}) = 2e$, where e is the identity matrix of $\mathbb{R}^{3\times 3}$, is positive definite. As for the conditions $\lambda < \frac{\kappa_j^\mu}{\theta_j^\mu}\frac{\tau_j^\mu}{\gamma_j^\mu}$, $j = 1,2,3$, they hold for any value of λ such that

$$0 < \lambda < \min\left\{ \frac{1}{1 - 1/(2\mu+1)^{1/2}}, \ \frac{1 - 1/(2\mu+1)^{1/2}}{1 - \lambda/(\lambda^2 + 2\mu)^{1/2}}, \ 1 \right\}$$

This is automatically the case if, for example, we set $\mu = 2 \times 10^{-2}$ and $\lambda = 10^{-2}$. □

There is at least one other way to show that $\nabla \Upsilon_\mu^\lambda(\bar{z})^T \nabla \Upsilon_\mu^\lambda(\bar{z})$ is nonsingular. The approach is based on the structure of the matrix, as it will be clear in the next result. To proceed, we need the following two assumptions.

Assumption 4.5. Each row of the following matrix is a nonzero vector:

$$\left[\nabla^2 L^\lambda(z)^T \quad \nabla(\nabla_y \mathcal{L}(z))^T \quad \nabla g(x,y)^T \quad \nabla G(x,y)^T\right].$$

Assumption 4.6. For $\lambda > 0$ and $\mu > 0$, the diagonal elements of the matrix $\nabla \Upsilon_\mu^\lambda(z)^T \nabla \Upsilon_\mu^\lambda(z)$ dominate the other terms row-wise; i.e., $a_{ii} > \sum_{j=1,\, j\neq i}^N |a_{ij}|$ for $i = 1, \ldots, N$, where a_{ij} denotes the element in the cell (i,j) of $\nabla \Upsilon_\mu^\lambda(z)^T \nabla \Upsilon_\mu^\lambda(z)$ for $i = 1, \ldots, N$ and $j = 1, \ldots, N$.

Lemma 4.7. *Let Assumption 4.5 hold at the point $z := (x,y,u,v,w)$. Then for any $\lambda > 0$ and $\mu > 0$, the diagonal elements of the matrix $\nabla \Upsilon_\mu^\lambda(z)^T \nabla \Upsilon_\mu^\lambda(z)$ are strictly positive.*

Proof. Considering the Jacobian matrix in (4.7), its transpose can be written as

$$\nabla \Upsilon_\mu^\lambda(z)^T = \begin{bmatrix} \nabla^2 L^\lambda(z)^T & \nabla(\nabla_y \mathcal{L}(z))^T & \nabla g(x,y)^T \mathcal{T}^{\mu T} & \nabla G(x,y)^T \mathcal{A}^{\mu T} & \nabla g(x,y)^T \Theta^{\mu T} \\ \nabla g(x,y) & O & \Gamma^{\mu T} & O & O \\ \nabla G(x,y) & O & O & \mathcal{B}^{\mu T} & O \\ -\lambda \nabla g(x,y) & \nabla_y g(x,y) & O & O & \mathcal{K}^{\mu T} \end{bmatrix}.$$

Denote by r_i, $i = 1, \ldots, 4$, respectively, the first, second, third, and fourth row-block of this matrix. Then the desired product can be represented as

$$\nabla \Upsilon_\mu^\lambda(z)^T \nabla \Upsilon_\mu^\lambda(z) = \begin{bmatrix} r_1 r_1^T & r_1 r_2^T & r_1 r_3^T & r_1 r_4^T \\ r_2 r_1^T & r_2 r_2^T & r_2 r_3^T & r_2 r_4^T \\ r_3 r_1^T & r_3 r_2^T & r_3 r_3^T & r_3 r_4^T \\ r_4 r_1^T & r_4 r_2^T & r_4 r_3^T & r_4 r_4^T \end{bmatrix}.$$

Obviously, the diagonal elements of $\nabla \Upsilon_\mu^\lambda(z)^T \nabla \Upsilon_\mu^\lambda(z)$ are the diagonal elements of $r_1 r_1^T$, $r_2 r_2^T$, $r_3 r_3^T$, and $r_4 r_4^T$. We can check that for $j = 1, \ldots, n+m$, a diagonal element of $r_1 r_1^T$ has the form

$$\begin{aligned} (r_1 r_1^T)_{jj} \ = \ & \textstyle\sum_{k=1}^{n+m} \nabla_{j,k}^2 L^\lambda(z)^T \nabla_{j,k}^2 L^\lambda(z) + \sum_{k=1}^m \nabla_j(\nabla_{y_k} \mathcal{L}(z))^T \nabla_j(\nabla_{y_k} \mathcal{L}(z)) \\ & + \textstyle\sum_{k=1}^p \nabla_j g_k(x,y)^T \nabla_j g_k(x,y) (\tau_k^\mu)^2 + \sum_{k=1}^q \nabla G_k(x,y)^T \nabla G_k(x,y) (\alpha_k^\mu)^2 \\ & + \textstyle\sum_{k=1}^p \nabla g_k(x,y)^T \nabla g_k(x,y) (\theta_k^\mu)^2, \end{aligned}$$

where ∇_j stands for the jth element of $\nabla := (\nabla_{x_1}, \ldots, \nabla_{x_n}, \nabla_{y_1}, \ldots, \nabla_{y_m})$ and $\nabla_{j,k}^2$ corresponds to an element in the jth row and kth column of

$$\nabla^2 := \begin{bmatrix} \nabla_{x_1 x_1} & \ldots \nabla_{x_1 x_n} & \nabla_{x_1 y_1} & \ldots \nabla_{x_1 y_m} \\ \vdots & \ddots & \ddots & \vdots \\ \nabla_{x_n x_1} & \ldots \nabla_{x_n x_n} & \nabla_{x_n y_1} & \ldots \nabla_{x_n y_m} \\ \nabla_{y_1 x_1} & \ldots \nabla_{y_1 x_n} & \nabla_{y_1 y_1} & \ldots \nabla_{y_1 y_m} \\ \vdots & \ddots & \ddots & \vdots \\ \nabla_{y_m x_1} & \ldots \nabla_{y_m x_n} & \nabla_{y_m y_1} & \ldots \nabla_{y_m y_m} \end{bmatrix}.$$

Combining Assumption 4.5 and Lemma 4.2, it is clear that $(r_1 r_1^T)_{jj} > 0$ for $j = 1, \ldots, n+m$. Similarly, the diagonal elements of $r_2 r_2^T$, $r_3 r_3^T$, and $r_4 r_4^T$ can respectively be written as

$$\begin{aligned}
(r_2 r_2^T)_{jj} &= \nabla g_j(x,y)\nabla g_j(x,y)^T + (\gamma_j^\mu)^2 \quad \text{for} \quad j = 1, \ldots, p,\\
(r_3 r_3^T)_{jj} &= \nabla G_j(x,y)\nabla G_j(x,y)^T + (\beta_j^\mu)^2 \quad \text{for} \quad j = 1, \ldots, q,\\
(r_4 r_4^T)_{jj} &= \nabla g_j(x,y)\nabla g_j(x,y)^T + (\kappa_j^\mu)^2 \quad \text{for} \quad j = 1, \ldots, p.
\end{aligned}$$

Thanks to Lemma 4.2, it is also clear that these items are all strictly positive. □

Theorem 4.8. *Let $z = (x,y,u,v,w)$ be a stationary point of the system (4.3) for some $\lambda > 0$ and $\mu > 0$. If Assumptions 4.5 and 4.6 are satisfied, then the matrix $\nabla\Upsilon_\mu^\lambda(z)^T\nabla\Upsilon_\mu^\lambda(z)$ is nonsingular.*

Proof. It is known that the matrix is positive definite if it is symmetric, its diagonal elements are strictly positive, and diagonal elements dominate elements of the matrix in the corresponding row. This property is the consequence of the *Gershgorin circle theorem* [39, page 320]. As $\nabla\Upsilon_\mu^\lambda(z)^T\nabla\Upsilon_\mu^\lambda(z)$ is symmetric, then combining Assumptions 4.5 and 4.6 to Lemma 4.7, we have the result. □

Next, we provide an example where the assumptions required for Theorem 4.8 are satisfied.

Example 4.9. Consider the instance of problem (1.1) taken from the BOLIB Library [85] with

$$F(x,y) := (x-1)^2 + y^2, \quad f(x,y) := x^2 y, \quad g(x,y) := y^2$$

and no upper-level constraint. For this problem, the function Υ^λ (2.9) can be written as

$$\Upsilon^\lambda(z) = \left(2x-2,\ 2y+2yu-2\lambda yw,\ x^2+2yw,\ \sqrt{u^2+y^4}-u+y^2,\ \sqrt{w^2+y^4}-w+y^2\right)^T.$$

The first item of note about this example is that the optimal solution $(\bar{x},\bar{y}) = (1,0)$ does not satisfy the optimality conditions (2.4)–(2.8), given that $\Upsilon^\lambda(\bar{x},\bar{y},\bar{u},\bar{w}) \neq 0$ for any values of $\bar{u}$ and $\bar{w}$. However, Algorithm 4.1 identifies the solution for λ taking the values 0.6, 0.7 or 0.8 with the smoothing parameter set to $\mu = 10^{-11}$. Indeed, the convergence of the method seems to be justified as for this problem, we can easily check that for $\bar{x} = 1$ and $\bar{y} = 0$,

$$\begin{aligned}
&\nabla^2 L^\lambda(\bar{z}) = 2\text{diag}\left(1,\ 1+\bar{u}-\lambda\bar{w}\right), \quad \nabla(\nabla_y\mathcal{L}(\bar{z})) = (2\ \ 2\bar{w}), \quad \nabla g(\bar{x},\bar{y}) = (0\ \ 0),\\
&\Gamma^\mu = \tfrac{\bar{u}}{\sqrt{\bar{u}^2+2\mu}} - 1, \quad \text{and} \quad \mathcal{K}^\mu = \tfrac{\bar{w}}{\sqrt{\bar{w}^2+2\mu}} - 1
\end{aligned}$$

and subsequently, we have the product

$$\nabla\Upsilon_\mu^\lambda(\bar{z})^T\nabla\Upsilon_\mu^\lambda(\bar{z}) = \begin{bmatrix} 8 & 4\bar{w} & 0 & 0\\ 4\bar{w} & 4\bar{w}^2+(2\bar{u}-2\lambda\bar{w})^2 & 0 & 0\\ 0 & 0 & \left(\frac{\bar{u}}{\sqrt{\bar{u}^2+2\mu}}-1\right)^2 & 0\\ 0 & 0 & 0 & \left(\frac{\bar{w}}{\sqrt{\bar{w}^2+2\mu}}-1\right)^2 \end{bmatrix}.$$

Hence, Assumption 4.5 is clearly satisfied; for Assumption 4.6 to hold, we need

$$8 > 4\bar{w},\ \ 4\bar{w}^2+(2\bar{u}-2\lambda\bar{w})^2 > 4\bar{w},\ \left(\frac{\bar{u}}{\sqrt{\bar{u}^2+2\mu}}-1\right)^2 > 0,\ \left(\frac{\bar{w}}{\sqrt{\bar{w}^2+2\mu}}-1\right)^2 > 0,$$

which holds for any $\mu > 0$, $\lambda > 0$, $\bar{u} > 0$, and $1 < \bar{w} < 2$. □

We further note that the assumptions made in Theorem 4.3 hold for the problem in this example. Firstly, we observe that $\nabla^2 L^\lambda(\bar{z})$ is positive definite if $\lambda\bar{w} < \bar{u}+1$. Subsequently, we can check that both assumptions of Theorem 4.3 are satisfied if

$$\lambda < \min\left\{\frac{\bar{u}+1}{\bar{w}},\ \frac{\left(\frac{\bar{w}}{\sqrt{\bar{w}^2+2\mu}}-1\right)}{\left(\frac{\bar{u}}{\sqrt{\bar{u}^2+2\mu}}-1\right)}\right\} \quad \text{with} \quad \bar{w} \neq 0.$$

For instance, choosing $\bar{u} = \sqrt{8} \times 10^{-6}$, $\bar{w} = 10^{-6}$, $\mu = 4 \times 10^{-12}$, and $\lambda < 2.25$ gives the result.

To conclude this section, we would like to analyse the *Jacobian consistency* of Υ^λ. Recall that according to [14], the Jacobian consistency property will hold for Υ^λ if this mapping is Lipschitz continuous and there exists a constant $\epsilon > 0$ such that for any $z \in \mathbb{R}^N$ and $\mu \in \mathbb{R}_+$, we have

$$\left\|\Upsilon^\lambda_\mu(z) - \Upsilon^\lambda(z)\right\| \leqslant \mu\epsilon \quad \text{and} \quad \lim_{\mu \downarrow 0} \text{dist}\left(\nabla\Upsilon^\lambda_\mu(z),\ \partial_C \Upsilon^\lambda(z)\right) = 0. \tag{4.20}$$

Here, *dist* represents the standard distance between a point and a set while $\partial_C \Upsilon^\lambda(z)^T$ denotes the C-subdifferential

$$\partial_C \Upsilon^\lambda(z)^T := \partial\Upsilon^\lambda_1(z) \times ... \times \partial\Upsilon^\lambda_{N+m}(z), \tag{4.21}$$

commonly used in this context; see, e.g., [43]. In (4.21), $N := n + m + 2p + q$ and $\partial\Upsilon^\lambda_i$, $i = 1, \dots, N + m$ represents the subdifferential in the sense of Clarke. Note that $\partial_C \Upsilon^\lambda(z)^T$ contains the generalized Jacobian in the sense of Clarke of the function Υ^λ. Roughly speaking, the Jacobian consistency property (4.20) translates to a framework ensuring that when the smoothing parameter μ converges to zero, the Jacobian $\nabla\Upsilon^\lambda_\mu(z)$ converges to an element in the C-subdifferential $\partial_C \Upsilon^\lambda(z)$. This property is important in determining the accuracy of the smoothing method in Algorithm 4.1.

Based on (4.21), at all points $z := (x, y, u, v, w)$ satisfying Assumption 3.1,

$$\partial_C \Upsilon^\lambda(z)^T := \left\{\nabla\Upsilon^\lambda(z)^T\right\}, \tag{4.22}$$

where $\nabla\Upsilon^\lambda(z)$ is defined by (3.3). For the case when strict complementarity does not hold, elements of $\partial_C \Upsilon^\lambda(z)$ have the same structure as in (3.3) with the only differences being in the terms τ_j, γ_j, α_j, β_j, θ_j, and κ_j for indices j where strict complementarity does not hold. We still have $\partial\Upsilon^\lambda_i(z)$ is the same as $\nabla\Upsilon^\lambda_i(z)$ for rows $i = 1, ..., n + 2m$. To determine the remaining rows, consider

$$\begin{aligned}
\Omega_1 &:= \{j : (u_j, g_j(x, y)) = (0, 0)\},\\
\Omega_2 &:= \{j : (v_j, G_j(x, y)) = (0, 0)\},\\
\Omega_3 &:= \{j : (w_j, g_j(x, y)) = (0, 0)\}.
\end{aligned}$$

Obviously τ_j and γ_j introduced in (3.4) are not well-defined for $j \in \Omega_1$. Similarly, the same holds for α_j and β_j for $j \in \Omega_2$ and θ_j and κ_j for $j \in \Omega_3$. Following the same procedure as in [43, Proposition 2.1], we define

$$\begin{aligned}
&\tau_k := \zeta_k + 1,\ \ \gamma_k := \rho_k - 1 \ \text{ for some } \ (\zeta_k, \rho_k) \in \mathbb{R}^2 \ \text{ such that } \ \|(\zeta_k, \rho_k)\| \leqslant 1 \ \text{ if } \ k \in \Omega_1,\\
&\alpha_k := \sigma_k + 1,\ \ \beta_k := \delta_k - 1 \ \text{ for some } \ (\sigma_k, \delta_k) \in \mathbb{R}^2 \ \text{ such that } \ \|(\sigma_k, \delta_k)\| \leqslant 1 \ \text{ if } \ k \in \Omega_2,\\
&\theta_k := \iota_k + 1,\ \ \kappa_k := \eta_k - 1 \ \text{ for some } \ (\iota_k, \eta_k) \in \mathbb{R}^2 \ \text{ such that } \ \|(\iota_k, \eta_k)\| \leqslant 1 \ \text{ if } \ k \in \Omega_3.
\end{aligned}$$

Since we do not assume strict complementarity here, then in contrast to Subsection 3.1, we have

$$(\tau_j - 1)^2 + (\gamma_j + 1)^2 \leqslant 1, \ \ (\alpha_j - 1)^2 + (\beta_j + 1)^2 \leqslant 1, \ \ (\theta_j - 1)^2 + (\kappa_j + 1)^2 \leqslant 1.$$

Theorem 4.10. *For $\lambda > 0$, the Jacobian consistency property holds for the approximation Υ^λ_μ of Υ^λ.*

Proof. First of all, note that Υ^λ is locally Lipschitz continuous. Proceeding as in [43, Corollary 2.4], we can easily check that we have

$$\left\|\Upsilon^\lambda_\mu(z) - \Upsilon^\lambda(z)\right\| \leqslant \epsilon\sqrt{\mu} \quad \text{with} \quad \epsilon := 2\sqrt{2p} + \sqrt{2q}.$$

$$\lim_{\mu \downarrow 0} \nabla\Upsilon^\lambda_\mu(z) = \lim_{\mu \downarrow 0} \begin{bmatrix} \nabla^2 L^\lambda(z) & \nabla g(x,y)^T & \nabla G(x,y)^T & -\lambda\nabla g(x,y)^T \\ \nabla(\nabla_y \mathcal{L}(z)) & O & O & \nabla_y g(x,y)^T \\ \mathcal{T}^\mu \nabla g(x,y) & \Gamma^\mu & O & O \\ \mathcal{A}^\mu \nabla G(x,y) & O & \mathcal{B}^\mu & O \\ \Theta^\mu \nabla g(x,y) & O & O & \mathcal{K}^\mu \end{bmatrix}, \tag{4.23}$$

where it is easy to see that the first two rows of (4.23) are the same as the first two rows of (3.3), as these are not involving perturbation μ. For the rest of Jacobian, we observe that

$$\lim_{\mu\downarrow 0}\left[\tau_j^\mu \nabla g_j(x,y), \quad \gamma_j^\mu, \quad 0, \quad 0\right] = \begin{cases} \begin{pmatrix}\tau_j \nabla g_j(x,y) & \gamma_j & 0 & 0\end{pmatrix} & \text{for } j \notin \Omega_1 \\ \begin{pmatrix}\nabla g_j(x,y) & -1 & 0 & 0\end{pmatrix} & \text{for } j \in \Omega_1 \end{cases} \tag{4.24}$$

and similarly for $\lim_{\mu\downarrow 0}\left[\begin{pmatrix}\alpha_j^\mu \nabla G_j(x,y) & \beta_j^\mu & 0 & 0\end{pmatrix}\right]$ and $\lim_{\mu\downarrow 0}\left[\begin{pmatrix}\theta_j^\mu \nabla g_j(x,y) & \kappa_j^\mu & 0 & 0\end{pmatrix}\right]$. This leads to $\lim_{\mu\downarrow 0} \operatorname{dist}(\nabla \Upsilon_\mu^\lambda(z), \partial_C \Upsilon^\lambda(z)) = 0$ as $\partial_C \Upsilon^\lambda(z)$ has the same form in (3.3) with corresponding adjustments to $\zeta_j, \rho_j, \sigma_j, \delta_j, \iota_j$ and η_j for the cases when strict complementarity does not hold. □

5 NUMERICAL RESULTS

The focus of our experiments in this section will be on the smoothing system (4.3), where we set $\mu := 10^{-11}$ constant throughout all iterations. Based on this system, we test and compare the Gauss-Newton method, the Pseudo-Newton method, and the MATLAB built-in method called *fsolve* (with Levenberg-Marquardt chosen as option). The examples used for the experiments are from the Bilevel Optimization LIBrary of Test Problems (BOLIB) [85], which contains 124 nonlinear examples. The experiments are run in MATLAB, version R2016b, on a MACI64. Here, we present a summary of the results obtained; more details for each example are reported in [Supp1].

For Step 0 of Algorithm 4.1 and the corresponding smoothed Pseudo-Newton algorithm, we set the tolerance to $\epsilon := 10^{-5}$ (see Subsection 5.4 for a justification) and the maximum number of iterations to be $K := 1000$. As for stopping criterion of `fsolve`, the tolerance is set to 10^{-5} as well. For the numerical implementation we calculate the direction d^k by solving (4.5) with Gaussian elimination. Five different values of the penalty parameter are used for all the experiments; i.e., $\lambda \in \{100, 10, 1, 0.1, 0.01\}$, see [Supp1] for details of the values of each solution for a selection of λ. The motivation of using different values of λ comes from the idea of not over-penalizing and not under-penalizing deviation of lower-level objective values from the minimum, as bigger (resp. smaller) values of λ seem to perform better for small (resp. big) values of lower-level objective. After running the experiments for all values of $\lambda \in \{100, 10, 1, 0.1, 0.01\}$, the best one is chosen (see Table 1 in [Supp1]), i.e. for which the best feasible solution is produced for the particular problem by the tested algorithms. Later in this section, we present the comparison of the performance of the algorithms for the best value of λ. The experiments have shown that the algorithms perform much better if the starting point (x^0, y^0) is feasible. As a default setup, we start with $x^0 = 1_n$ and $y^0 = 1_m$. If the default starting point does not satisfy at least one constraint, we choose a feasible starting point; see [Supp1]. Subsequently, the Lagrange multipliers are initialised at $u^0 = \max\left\{0.01, \ -g(x^0, y^0)\right\}$, $v^0 = \max\{0.01, \ -G(x,y)\}$, and $u^0 = w^0$.

5.1 Performance profiles

Performance profiles are widely used to compare characteristics of different methods. In this section we consider performance profiles, where $t_{i,s}$ denotes the CPU time to solve problem i by algorithm s. If the optimal solution of problem i is known but it cannot be solved by algorithm s (i.e., upper-level objective function error > 60%), we set $t_{i,s} := \infty$. We then define the *performance ratio* by

$$r_{i,s} := \frac{t_{i,s}}{\min\{t_{i,s} : s \in S\}},$$

where S is the set of solvers. The performance ratio is the ratio of how algorithm s performed to solve problem i compared to the performance of the best performed algorithm from the set S. The *performance profile* can be defined as the cumulative distribution function of the performance ratio:

$$\rho_s(\tau) := \frac{\left|\{i \in P : r_{i,s} \leqslant \tau\}\right|}{n_p},$$

where P is the set of problems and τ is the number measuring performance ratio. The performance profile, $\rho_s(\tau)$, is counting the number of examples for which the performance ratio of the algorithm s is better (smaller) than τ. The performance profile $\rho_s : \mathfrak{R} \to [0, 1]$ is a non-decreasing function, where the value of $\rho_s(1)$ shows the fraction of the problems for which solver s was the best.

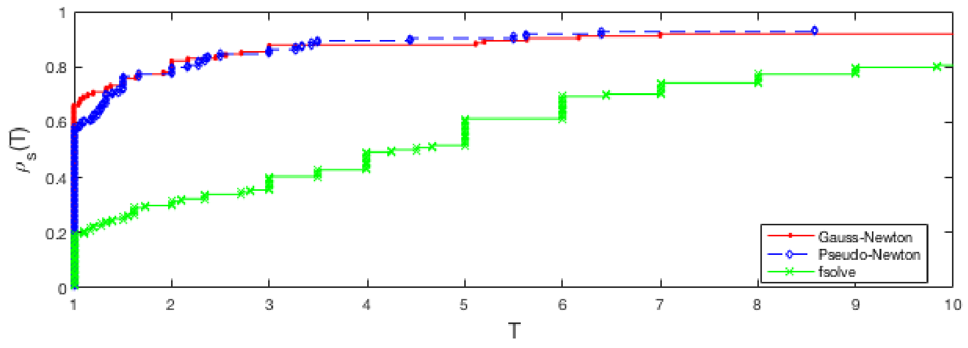


Figure 4: Performance profiles of the methods for 124 problems

Comparing line-graphs of the performance profiles (cf. Figure 4), a higher position of a graph indicates better performance of the corresponding algorithm. The value on the y-axis shows the fraction of examples for which the performance ratio is better than T (presented on the x-axis). Figure 4 clearly shows that the Gauss-Newton and Pseudo-Newton method perform better than fsolve. Since the variable for the comparison was CPU time, based on the values of $\rho_s(1)$, we can claim that Gauss-Newton was the fastest algorithm for about 70% of the problems, Pseudo-Newton for about 60% of the problems and fsolve was the quickest for about 20% of the problems. From the graph, one can also see that Gauss-Newton and Pseudo-Newton methods have $\rho_s(2) = 80\%$, while fsolve only has the value $\rho_s(2) = 30\%$, meaning that fsolve was more than twice worse than the best algorithm for 70% of the problems. Approaching $T = 6$, the Gauss-Newton and Pseudo-Newton methods obtain a performance ratio close to $\rho_s(T) = 90\%$, where fsolve obtains $\rho_s(6) \approx 65\%$. This shows that Gauss-Newton and Pseudo-Newton methods show quite similar performance in terms of CPU time. Clearly, from the perspective of the performance profiles discussed in this subsection, both of these algorithms clearly outperform fsolve for solving our test problems.

5.2 Feasibility check

Considering the structure of the feasible set of problem (1.2), it is critical to check whether the points computed by our algorithms satisfy the value function constraint $f(x, y) \leqslant \varphi(x)$, as it is not explicitly included in the expression of Υ^λ (2.9). If the lower-level problem is convex in y and a solution generated by our algorithms satisfies (2.5) and (2.8), then it will verify the value function constraint. Conversely, to guaranty that a point (x, y) such that $y \in S(x)$ satisfies (2.5) and (2.8), a constraint qualification (CQ) is necessary. Note that conditions (2.5) and (2.8) are incorporated in the stopping criterion of Algorithm 4.1. To check whether the points obtained are feasible, we first identify the BOLIB examples, where the lower-level problem is convex w.r.t. y; see the summary of these checks in Table 1. It turns out that a significant number of test examples have linear lower-level constraints. For these examples, the lower-level convexity is automatically satisfied.

$f(\cdot, y)$	$g_i(\cdot, y), i = 1, \dots, p$	**Total count**
Convex	Convex (linear)	55
Convex	Convex (nonlinear)	14
Convex	No constraints	5
Convex	Convex (with CQ satisfied)	74

Table 1: Convexity of the lower-level functions

There are 14 problems, where $f(\cdot, y)$ and $g_i(\cdot, y)$ with $i = 1, \ldots, p$, are convex, but the constraints are not all linear w.r.t. y. For these examples, the lower-level regularity (2.1) has been shown to hold at the points computed by our algorithms. The rest of the problems have non-convex lower-level objective or some of the lower-level constraints are non-convex. For these examples, we compare the solutions obtained with those known from the literature. Let f_A stand for $f(\bar{x}, \bar{y})$, the lower-level function value obtained by one of the algorithms tested at convergence, and let f_K be the known optimal value of lower-level objective function. In the graph below we plot the lower-level relative error, $(f_A - f_K)/(1 + |f_K|)$ on the y-axis, against different problems. The errors are plotted in increasing order. Note that for the 25 problems with smallest relative error the error is smaller than 5%.

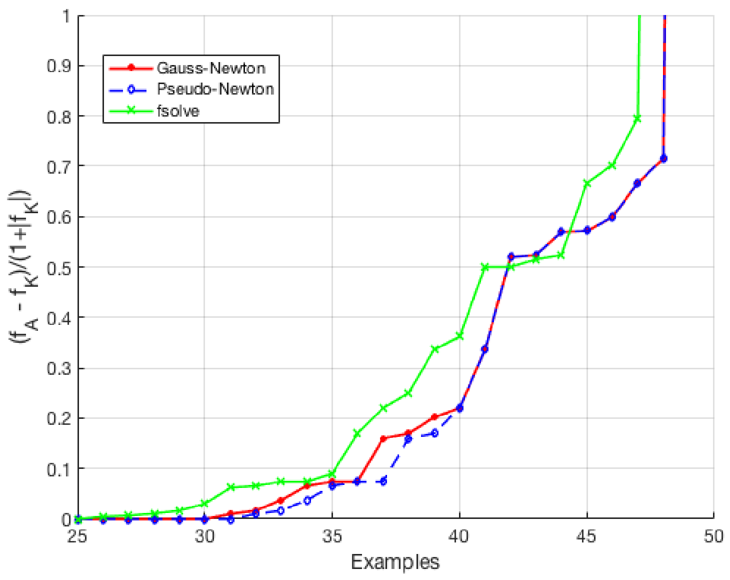


Figure 5: Lower-level optimality check for examples with a nonconvex lower-level problem

From the figure above we can see that for 30 problems the relative error of lower-level objective is negligible ($< 5\%$) for all three methods. For almost all of the remaining 19 examples, the Gauss-Newton and Pseudo-Newton methods obtain smaller errors than fsolve, while the Pseudo-Newton method seems to obtain slightly smaller errors than the Gauss-Newton method for some of the examples. We have seen that convexity and a constraint qualification hold for the lower-level problem hold for 74 test examples. Accordingly, (2.5) and (2.8) are guarantied to hold and hence, the corresponding points are feasible for our bilevel programs. If we allow for a feasibility error of up to 20%, feasibility is satisfied for 113 (91.13%) problems through the Gauss-Newton and Pseudo-Newton methods, and for 110 (88.71%) problems if we proceed with fsolve.

5.3 Accuracy of the upper-level objective function

Here, we compare the values of the upper-level objective functions at points computed by the algorithms under study, i.e., Gauss-Newton and Pseudo-Newton algorithms and fsolve. For this comparison, we focus our attention only on 116 BOLIB examples [85], as solutions are not known for six of them and the Gauss-Newton algorithm diverges for *NieEtal2017e*, possibly due the singularity of the matrix $\nabla\Upsilon^{\lambda}_{\mu}(z)^T\nabla\Upsilon^{\lambda}_{\mu}(z)$; see [Supp1] for more details. To proceed, let $\bar{F}_A$ be the value of upper-level objective function at the point $(\bar{x}, \bar{y})$ at which the algorithm under consideration stops, and let $\bar{F}_K$ the value of this function at the known best solution point reported in the literature (see corresponding references in [85]). The comparison is shown in Figure 6, where we plot the relative error $(\bar{F}_A - \bar{F}_K)/(1 + |\bar{F}_K|)$ on the $y - axis$ against examples on the x-axis. Again, the graph is plotted in the order of increasing error. The 85 test cases with smallest error all exhibit a relative error smaller than 5%.

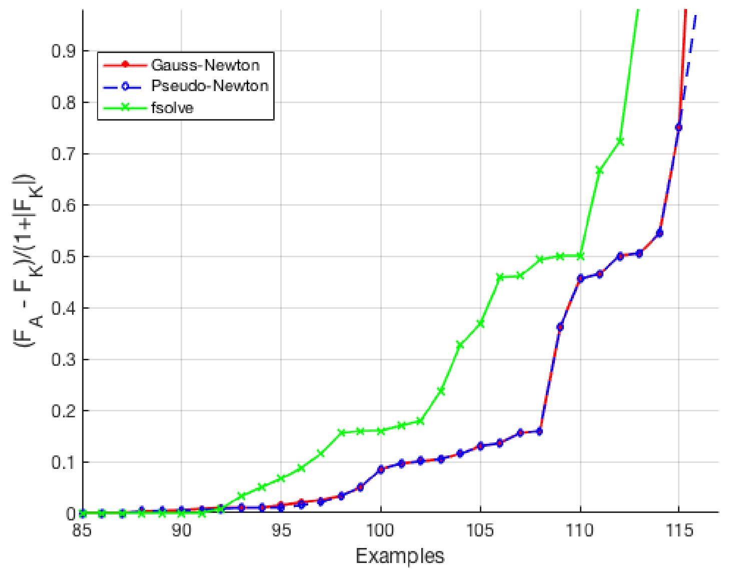


Figure 6: Comparison of upper-level objective values for examples with known solutions

From Figure 6, we can see that most of the known best values of upper-level objective functions were recovered by all the methods, as the relative error is close to zero. For 93 of the tested problems, the upper-level objective function error is negligible (i.e., less than 5%) for the solutions obtained by all the three methods. For the remaining 23 examples, it is clear that the errors resulting from fsolve are much higher than the ones from the Gauss-Newton and Pseudo-Newton methods. It is worth noting that algorithms perform fairly well for most of the problems. With the accuracy error $\leqslant 20\%$ our algorithms recovered solutions for 92.31% of the problems, while fsolve recovered only 88.03% of the solutions.

5.4 Variation of the tolerance in the stopping criterion

Let us now evaluate the performance of Algorithm 4.1 as we relax the tolerance in the stopping criterion. Setting $\epsilon := 10^{-8}$ (as opposed to $\epsilon := 10^{-5}$ used so far) it turns out that for most of the examples our algorithms stop at the maximum number of iterations, or when the gap between improvement from step to step is too small.

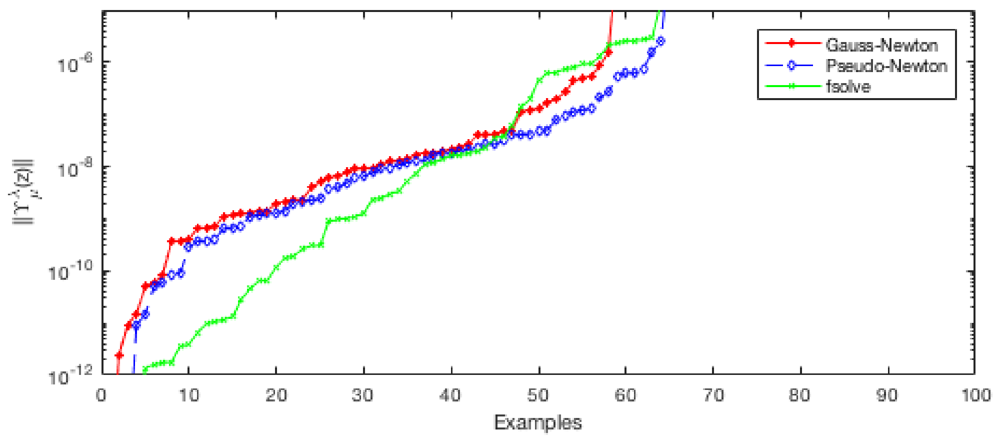


Figure 7: Performance of the methods in terms of solving $\Upsilon^{\lambda}_{\mu}(z) = 0$

The values of $\|\Upsilon^{\lambda}_{\mu}(z)\|$ produced by the algorithms are presented in increasing order in Figure 7. We can see that fsolve performs slightly better for 40 examples, where all algorithms achieve $\|\Upsilon^{\lambda}_{\mu}(z)\| \leqslant 10^{-8}$. For those examples, fsolve recovered solutions with better tolerance than the

Gauss-Newton and Pseudo-Newton algorithms. This shows that whenever we are able to solve a problem almost exactly, fsolve's stopping criteria is more strict and obtains slightly better values for the system. This can be explained by an additional stopping criteria that we use for Gauss-Newton and Pseudo-Newton methods if the improvement between the steps of the algorithms gets too small. Accordingly, whenever we are able to solve the system $\Upsilon^{\lambda}_{\mu}(z) = 0$ almost exactly, fsolve's stopping criteria is somewhat less strict and keeps iterating to produce values of $\|\Upsilon^{\lambda}_{\mu}(z)\|$ that are closer to 0 than for the other two methods. The explanation could be that due to an additional stopping criteria for the Gauss-Newton and Pseudo-Newton algorithms stop earlier once significant improvement of the solution is not observed from step to step. If we now look at the performance of the algorithms with tolerances between 10^{-8} and 10^{-6}, the Pseudo-Newton method shows better performance than the other algorithms and fsolve is the weakest among the three methods. This means that if we want to solve a problem with the tolerance of 10^{-6} or better, the Pseudo-Newton algorithm is more likely to recover solutions than the other two methods. The other important observation from Figure 7 is that choosing $\epsilon := 10^{-5}$ is the most sensible tolerance for our problem set, as better tolerance only allow about 50% of the examples to be solved (i.e., just over 60 examples as we can see from the graph).

5.5 Checking assumption on λ

Considering the importance of the requirement that $\lambda < \frac{\kappa^{\mu}_j}{\theta^{\mu}_j}\frac{\tau^{\mu}_j}{\gamma^{\mu}_j}$ for $j = 1, \dots, p$ in Theorem 4.3, let us analyze its ramifications regarding the solution point generated from Algorithm 4.1 for each value of $\lambda \in \{100, 10, 1, 0.1, 0.01\}$. To simplify the analysis, we introduce

$$c^{\mu}(\lambda) := \min_{j=1,\dots,p} \frac{\kappa^{\mu}_j}{\theta^{\mu}_j}\frac{\tau^{\mu}_j}{\gamma^{\mu}_j}.$$

It suffices to check that $\lambda < c^{\mu}(\lambda)$. We set $\lambda - c^{\mu}(\lambda) := 100$ if the difference is undefined, and we do not consider problems with no lower-level constraints. (Because the assumption on λ is not necessary in Theorem 4.3 for problems with no lower-level constraint.) Let us also introduce the notions of the best λ and optimal λ, whereby the best λ we mean the values of λ for which $\lambda - c^{\mu}(\lambda)$ is the smallest and by the optimal we mean the values of λ that were the best to obtain the solution according to [Supp1]. In the figure below we present the difference $\lambda - c^{\mu}(\lambda)$ on the y-axis and number of the example on the x-axis, following ascending order w.r.t. the values on the y-axis.

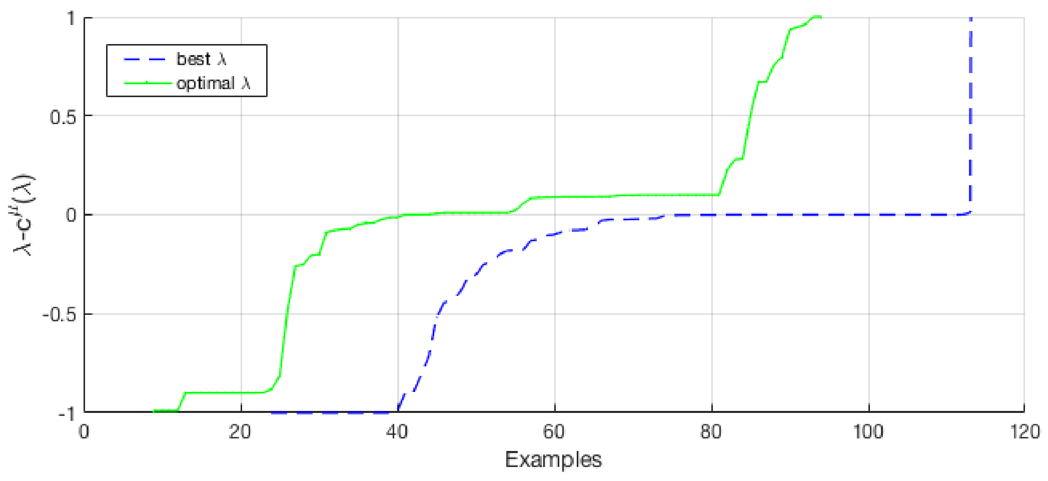


Figure 8: Checking assumption $\lambda < c^{\mu}(\lambda)$ for best and optimal values of λ

Clearly, condition $\lambda < c^{\mu}(\lambda)$ holds for the values of $\lambda - c^{\mu}(\lambda)$ lying below the x-axis. From Figure 8, we can see that the assumption holds for 42 (out of 116) problems for the optimal λ. This means that the condition can hold for many examples. But, obviously, as it is not a necessary condition, our Algorithm 4.1 still converges for many other problems, where the condition is not necessarily satisfied. For the best values of λ, condition $\lambda < c^{\mu}(\lambda)$ holds for 101 problems. Hence, showing that for most of the examples, there is at least one value of $\lambda \in \{100, 10, 1, 0.1, 0.01\}$ for which the condition is satisfied.

6 FINAL COMMENTS

In this paper, a class of the lower-level value function-based optimality conditions for the bilevel optimization problems has been reformulated as a system of equations using the Fischer-Burmeister function. We have shown that the Gauss-Newton method for such systems can be well-defined and provide a framework for convergence. We then test the method and its smoothed version numerically, alongside with Newton's method with Moore-Penrose pseudo inverse. The comparison of solutions obtained with known best ones shows that the methods are appropriate to be used for bilevel programs, as they recover optimal solutions (when known) for most of the problems tested.

It is worth mentioning that whenever $\nabla\Upsilon^{\lambda}_{\mu}(z)^T\nabla\Upsilon^{\lambda}_{\mu}(z)$ is non-singular throughout all iterations, Gauss-Newton and Pseudo-Newton methods produced the same results as expected. More interestingly, for the 38 test problems for which the Gauss-Newton method could not be implemented due to singularity of the direction matrix for one or more values of λ (see [Supp1]), our conjecture that the Pseudo-Newton would produce reasonable solutions for these cases was correct for 14 problems (e.g., *CalamaiVicente1994c*, *DempeDutta2012b*, and *DempeFranke2011a* in [Supp1]), but fails for the remaining 24 examples (e.g., *Bard1988c*, *Colson2002BIPA3*, and *DempeDutta2012a* in [Supp1]). Nevertheless, we can conclude that our proposed Pseudo-Newton method is indeed somewhat more robust than the Gauss-Newton method. Overall, the Gauss-Newton and Pseudo-Newton methods are more efficient at recovering solutions for BOLIB test problems [85] than the MATLAB built-in function fsolve. We also show that our methods are better at producing feasible solutions (i.e., optimal points for the lower-level problem) than fsolve; cf. Subsection 5.2.

ACKNOWLEDGEMENTS

The authors would like to thank the two anonymous referees for their constructive remarks that have helped us to improve the quality of this paper. The work of the second author was supported by the University of Southampton Presidential Scholarship and that of third author was funded by an EPSRC grant with reference number EP/P022553/1.

Part III.

Paper 2: Levenberg-Marquardt method and partial exact penalty parameter selection in bilevel optimization

This article examines the application of smoothed Levenberg-Marquardt method to find solutions of bilevel programming problems. To proceed, we use the lower-level value function reformulation of bilevel programs and consider necessary optimality conditions under appropriate assumptions. In particular, we discuss partial calmness assumption used to derive the optimality conditions. Partial calmness is strongly linked to the exact penalization parameter for our case. This turns out to play important role for our method. The method is analyzed for the two options of selecting penalty parameter. First approach implements the method with several fixed values of penalty parameter. The second approach defines penalty parameter as increasing sequence throughout iterations of the method. We further present the algorithm with the required convergence assumptions and parameters specifications. Numerical experiments conducted on 124 examples from the recently released Bilevel Optimization Library (BOLIB) compare the performance of our method under two different scenarios of selecting penalty parameter.

1 INTRODUCTION

We aim to solve the bilevel programming problem

$$\min_{x,y} F(x,y) \ \text{ s.t. } \ G(x,y) \leqslant 0, \ y \in S(x) := \arg\min_{y} \{f(x,y) : \ g(x,y) \leqslant 0\}, \tag{1.1}$$

where $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, $G : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$, and $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$. As usual, we refer to F (resp. f) as upper-level (resp. lower-level) objective function and G (resp. g) stands for upper-level (resp. lower-level) constraint function. Solving problem (1.1) is very difficult because of the implicit nature of the lower-level optimal solution mapping $S : \mathbb{R}^n \rightrightarrows \mathbb{R}^m$ defined in (1.1).

There are several ways to deal with the complex nature of problem (1.1). One popular technique is to replace the lower-level problem with its Karush-Kuhn-Tucker (KKT) conditions. Interested readers are referred to [1, 23, 41] and references therein, for results and methods based on this transformation. As mentioned in [Paper 1] such reformulation is strongly linked to MPECs, see, e.g., [19], which are not necessarily easy to handle due in part to the extra variables representing the lower-level Lagrangian multipliers. In this paper, we are going to use the lower-level value function reformulation (LLVF)

$$\min_{x,y} F(x,y) \ \text{ s.t. } \ G(x,y) \leqslant 0, \ g(x,y) \leqslant 0, \ f(x,y) \leqslant \varphi(x), \tag{1.2}$$

where the optimal value function is defined by

$$\varphi(x) := \min \{f(x,y) \,|\, g(x,y) \leqslant 0\}, \tag{1.3}$$

to transform problem (1.1) into a single-level optimization problem. As illustrated in [31, Paper 1], this approach can provide tractable opportunities to develop algorithms for the bilevel optimization problem. The benefit of such reformulation is that it does not involve first order derivatives for lower-level problem, in contrast to the KKT reformulation.

There are recent studies on solution methods for bilevel programs, based on the LLVF reformulation. For example, [46, 47, 53, 58, 73] develop global optimization techniques for (1.1) based on (1.2)–(1.3). [50, 75, 76] propose algorithms computing stationary points for (1.2)–(1.3), in the case where the upper-level and lower-level feasible sets do not depend on the lower-level and upper-level variable, respectively. [31, Paper 1] are the first papers to propose Newton-type methods for the LLVF reformulation for bilevel programs. Numerical results there show that the methods could be well-defined and produce successful results with reasonable frequency. The study of Gauss-Newton-type techniques to capture certain classes of bilevel optimization stationarity points was already carried out in [Paper 1]. This paper considers extension of Gauss-Newton method, known as *Levenberg-Marquardt method*. Levenberg-Marquardt method is more robust than Gauss-Newton method as the possibility of the singularity problem is avoided by adding a perturbation on the main diagonal of the matrix.

One of the main problems in solving (1.2) is that its feasible set systematically fails many constraint qualifications (see, e.g., [24]). To deal with this issue, we will use the partial calmness condition [81], to shift the value function constraint $f(x,y) \leqslant \varphi(x)$ to the upper-level objective function, as the penalty term with parameter λ. The choice of the penalty parameter is a broad area, where authors use different methods to optimally choose the parameter, [8, 11, 33, 51, 59, 61]. For this reason, this paper will study partial calmness assumption and penalization approach for our framework in details. The other difficulty with the LLVF reformulation is that φ is typically non-differentiable function. This will be handled by using upper estimates of the subdifferential of the function; see, e.g., [24, 20, 23, 81]. This estimate will lead us to a relatively simple system of optimality conditions, which depend on λ. However, the control of the penalty parameter λ is not trivial. Two approaches in the context of applying Levenberg-Marquardt scheme will be discussed in the paper. The more theoretical approach suggests varying λ as a sequence, increasing the value of λ at each iteration. However, this approach has a disadvantage for implementing the algorithm. In particular, It will be shown that if we let penalty parameter to be too large an issue of ill-conditioning could occur. Hence, the approach of choosing fixed values of λ is discussed as well. As we do not know the appropriate order of magnitude of λ for each example beforehand, it makes sense to test the algorithm for several values of fixed λ.

In section 2 we start by transforming the optimality conditions into a system of equations. To do so we substitute the corresponding complementarity conditions by the standard Fischer-Burmeister function [29]. To deal with the non-differentiability of the Fischer-Burmeister function, we implement a smoothing technique by adding a perturbation, μ, inside the Fischer-Burmeister function. We are then going to introduce Smoothed Levenberg-Marquardt method for Bilevel Optimization. It will be shown that the method can converge for solving bilevel programs from the perspective of the LLVF reformulation (1.2). *Line search* is then introduced, where we use Armijo condition to control stepsize of the algorithm. This ensures global convergence of the method and brings in an extra parameter to the method. As we are combining several techniques, involving the task of choosing parameters efficiently, we conclude Section 2 by discussing parameters selection for the algorithm. For the numerical tests in Section 4, we present results of extensive experiments for testing the method. The numerical analysis will be in the similar framework as in [Paper 1]. Additionally, we are going to study the behaviour of the method with the penalty parameter defined as increasing sequence. In Section 3 we are going to focus on partial calmness and exact penalization, including discussion of the choice of penalty parameter and ill-conditioning issue. This analysis plays an important role in the behaviour of the method and hence discussed in details, including the ways to avoid ill-conditioning of the method. Further, we are going to present some results on *experimental order of convergence* (EOC) of the method and line search stepsize parameter at the last iteration. The results in Section 4 are compared with known solutions of the problems to check the performance of the method under two different scenarios of choosing penalty parameter.

2 LEVENBERG–MARQUARDT METHOD FOR BILEVEL OPTIMIZATION

We start this section with some definitions necessary to state the optimality conditions of the bilevel program (1.2). The lower-level problem is *fully convex* if the functions f and g_i, $i = 1, \dots, p$ are convex in (x, y). A point $(\bar{x}, \bar{y})$ is said to be *lower-level regular* if there exists d such that

$$\nabla_y g_i(\bar{x}, \bar{y})^\top d < 0 \quad \text{for} \quad i \in I_g(\bar{x}, \bar{y}) := \{i : \; g_i(\bar{x}, \bar{y}) = 0\}. \tag{2.1}$$

Obviously, this is the *Mangasarian-Fromovitz constraint qualification* (MFCQ) for the lower-level problem in (1.2). Similarly, a point $(\bar{x}, \bar{y})$ is *upper-level regular* if there exists d such that

$$\begin{aligned} \nabla g_i(\bar{x}, \bar{y})^\top d < 0 \quad &\text{for} \quad j \in I_g(\bar{x}, \bar{y}), \\ \nabla G_j(\bar{x}, \bar{y})^\top d < 0 \quad &\text{for} \quad j \in I_G(\bar{x}, \bar{y}) := \left\{j : \; G_j(\bar{x}, \bar{y}) = 0\right\}. \end{aligned} \tag{2.2}$$

Finally, to write the necessary optimality conditions for problem (1.2), it is standard to use the following partial calmness concept [81]:

Definition 2.1. *Let $(\bar{x}, \bar{y})$ be a local optimal solution of problem (1.2). This problem is partially calm at $(\bar{x}, \bar{y})$ if there exists $\lambda > 0$ and a neighbourhood U of $(\bar{x}, \bar{y}, 0)$ such that*

$$F(x, y) - F(\bar{x}, \bar{y}) + \lambda |u| \geqslant 0, \;\; \forall (x, y, u) \in U : \; G(x, y) \leqslant 0, \; g(x, y) \leqslant 0, \; f(x, y) - \varphi(x) - u = 0.$$

The following relationship shows that partial calmness enables the penalization of the optimal value function constraint $f(x, y) - \varphi(x) \leqslant 0$ to generate a tractable optimization problem.

Theorem 2.2 ([52])**.** *Let $(\bar{x}, \bar{y})$ be a local minimizer of (1.2). Then, this problem is partially calm at $(\bar{x}, \bar{y})$ if and only if there is some $\bar{\lambda} > 0$ such that for any $\lambda \geqslant \bar{\lambda}$, the point $(\bar{x}, \bar{y})$ is also a local minimizer of*

$$\min_{x, y} F(x, y) + \lambda(f(x, y) - \varphi(x)) \;\; \text{s.t.} \;\; G(x, y) \leqslant 0, \;\; g(x, y) \leqslant 0. \tag{2.3}$$

Problem (2.3) is known as the *partial exact penalization* problem of (1.2), as only the optimal value constraint is penalized. Next, we state the necessary optimality conditions for problems based on the aforementioned qualification conditions (see, e.g., [24, 20, 23, 81]) based on an estimate of the subdifferential of the optimal value function φ (1.3).

Theorem 2.3. *Let $(\bar{x}, \bar{y})$ be a local optimal solution of problem (1.2), where all involved functions are assumed to be continuously differentiable, φ is finite around $\bar{x}$, and the lower-level problem is fully convex. Furthermore, suppose that the problem is partially calm at $(\bar{x}, \bar{y})$, and the lower- and upper-level regularity conditions are both satisfied at $(\bar{x}, \bar{y})$. Then, there exist $\lambda > 0$, as well as $u, w \in \mathbb{R}^p$ and $v \in \mathbb{R}^q$ such that*

$$\nabla F(\bar{x}, \bar{y}) + \nabla g(\bar{x}, \bar{y})^T (u - \lambda w) + \nabla G(\bar{x}, \bar{y})^T v = 0, \tag{2.4}$$

$$\nabla_y f(\bar{x}, \bar{y}) + \nabla_y g(\bar{x}, \bar{y})^T w = 0, \tag{2.5}$$

$$u \geqslant 0, \;\; g(\bar{x}, \bar{y}) \leqslant 0, \;\; u^T g(\bar{x}, \bar{y}) = 0, \tag{2.6}$$

$$v \geqslant 0, \;\; G(\bar{x}, \bar{y}) \leqslant 0, \;\; v^T G(\bar{x}, \bar{y}) = 0, \tag{2.7}$$

$$w \geqslant 0, \;\; g(\bar{x}, \bar{y}) \leqslant 0, \;\; w^T g(\bar{x}, \bar{y}) = 0. \tag{2.8}$$

Remark 2.4. Most of the literature such as [20, 23, 24, 81] consider the case where upper-level constraint G does not depend on lower-level variable y. Hence, the result there would have $\nabla_y G(x, y) = 0$. However, it has been shown in [80] that introducing optimality conditions with $G(x, y)$ is mathematically valid and we consider this scenario here as it is more general.

In this result, partial calmness and full convexity are essential and fundamentally related to the nature of the bilevel optimization. Hence, it is important to highlight a few classes of problems satisfying these assumptions. Partial calmness has been the main tool to derive optimality conditions for (1.1) from the perspective of the optimal value function; see, e.g., [20, 23, 24, 81]. It automatically holds if G is independent from y and the lower-level problem is defined by

$$f(x, y) := c^\top y \quad \text{and} \quad g(x, y) := A(x) + By, \tag{2.9}$$

where $A : \mathbb{R}^n \to \mathbb{R}^p$, $c \in \mathbb{R}^m$, and $B \in \mathbb{R}^{p\times m}$. More generally, various sufficient conditions ensuring that partial calmness holds have been studied in the literature; see [81] for the seminal work on the subject. More recently, the paper [52] has revisited the condition, proposed a fresh perspective, and established new dual-type sufficient conditions for partial calmness to hold.

As for full convexity, it will be automatically be satisfied for the class of problem defined in (2.9) provided that each component of the function A is a convex function. Another class of nonlinear fully convex lower-level problem is given in [48]. Note however when it is not possible to guarantee that this assumption is satisfied, there are at least two alternative scenarios to obtain the same optimality conditions. The first is to replace the full convexity assumption by the *inner semicontinuity* of the optimal solution set-valued mapping S (1.1). Secondly, note that a much weaker qualification condition known as *inner semicompactness* can also be used here. However, under the latter assumption, it will additionally be required to have $S(\bar{x}) = \{\bar{y}\}$ in order to get the optimality conditions in (2.4)–(2.8). The concept of inner semicontinuity (resp. semicompactness) of S is closely related to the lower semicontinuity (resp. upper semicontinuity) of set-valued mappings; for more details on these notions and their implementation on bilevel programs, see [20, 23, 24].

It is important to mention that various other necessary optimality conditions, different from the above ones, can be obtained, depending on the assumptions made. Details of different stationarity concepts for (1.2) can be found in the latter references, as well as in [83].

2.1 The algorithm and its convergence analysis

To reformulate the complementarity conditions (2.6)–(2.8) into a system of equations, we use the well-known *Fischer-Burmeister* function [29] $\phi(a,b) := \sqrt{a^2+b^2} - a - b$, which ensures that

$$\phi(a,b) = 0 \iff a \geqslant 0,\ \ b \geqslant 0,\ \ ab = 0.$$

This leads to the reformulation of the optimality conditions (2.4)–(2.8) into the system of equations:

$$\Upsilon^\lambda(z) := \begin{pmatrix} \nabla_x F(x,y) + \nabla_x g(x,y)^T(u - \lambda w) + \nabla_x G(x,y)^T v \\ \nabla_y F(x,y) + \nabla_y g(x,y)^T(u - \lambda w) + \nabla_y G(x,y)^T v \\ \nabla_y f(x,y) + \nabla_y g(x,y)^T w \\ \sqrt{u^2 + g(x,y)^2} - u + g(x,y) \\ \sqrt{v^2 + G(x,y)^2} - v + G(x,y) \\ \sqrt{w^2 + g(x,y)^2} - w + g(x,y) \end{pmatrix} = 0, \tag{2.10}$$

where we have $z := (x,y,u,v,w)$ and

$$\sqrt{u^2 + g(x,y)^2} - u + g(x,y) := \begin{pmatrix} \sqrt{u_1^2 + g_1(x,y)^2} - u_1 + g_1(x,y) \\ \vdots \\ \sqrt{u_p^2 + g_p(x,y)^2} - u_p + g_p(x,y) \end{pmatrix}. \tag{2.11}$$

$\sqrt{v^2 + G(x,y)^2} - v + G(x,y)$ and $\sqrt{w^2 + g(x,y)^2} - w + g(x,y)$ are defined as in (2.11). The superscript λ is used to emphasize the fact that this number is a parameter and not a variable for equation (2.10). One can easily check that this system is made of $n + 2m + p + q + p$ real-valued equations and $n + m + p + q + p$ variables. Clearly, this means that (2.10) is an *overdetermined* system and the Jacobian of $\Upsilon^\lambda(z)$, when it exists, is a non-square matrix.

To focus our attention on the main ideas of this paper, we smoothen the function Υ^λ (2.10) using the *smoothed Fischer-Burmeister function* (see [42]) defined by

$$\phi_j^\mu(x,y,u) := \sqrt{u_j^2 + g_j(x,y)^2 + 2\mu} - u_j + g_j(x,y),\ j = 1,\dots,p, \tag{2.12}$$

where the perturbation $\mu > 0$ helps to guaranty its differentiability at points (x,y,u), where $u_j = g_j(x,y) = 0$ for $j = 1,\dots,p$. It is well-known (see latter reference) that for $j = 1,\dots,p$,

$$\phi_j^\mu(x,y,u) = 0 \iff \left[u_j > 0,\ -g_j(x,y) > 0,\ -u_j g_j(x,y) = \mu\right]. \tag{2.13}$$

The smoothed version of system (2.10) then becomes

$$\Upsilon^\lambda_\mu(z) = \begin{pmatrix} \nabla_x F(x,y) + \nabla_x g(x,y)^T(u-\lambda w) + \nabla_x G(x,y)^T v \\ \nabla_y F(x,y) + \nabla_y g(x,y)^T(u-\lambda w) + \nabla_y G(x,y)^T v \\ \nabla_y f(x,y) + \nabla_y g(x,y)^T w \\ \sqrt{u^2 + g(x,y)^2 + 2\mu} - u + g(x,y) \\ \sqrt{v^2 + G(x,y)^2 + 2\mu} - v + G(x,y) \\ \sqrt{w^2 + g(x,y)^2 + 2\mu} - w + g(x,y) \end{pmatrix} = 0, \tag{2.14}$$

following the convention in (2.11), where μ is a vector of appropriate dimensions with sufficiently small positive elements. Under the assumption that all the functions involved in problem (1.1) are continuously differentiable, Υ^λ_μ is also a continuously differentiable function for any $\lambda > 0$ and $\mu > 0$. We can easily check that for a fixed value of $\lambda > 0$,

$$\|\Upsilon^\lambda_\mu(z) - \Upsilon^\lambda(z)\| \quad \longrightarrow 0 \quad \text{as} \quad \mu \downarrow 0. \tag{2.15}$$

Hence, based on this scheme, our aim is to consider a sequence $\{\mu_k\}$ decreasing to 0 such that equation (2.10) is approximately solved while leading to

$$\Upsilon^\lambda_{\mu^k}(z^k) \quad \longrightarrow 0 \quad \text{as} \quad k \uparrow \infty$$

for a fixed value of $\lambda > 0$. In order to proceed, let us define the least squares problem

$$\min \; \Phi^\lambda_\mu(z) := \frac{1}{2}\left\|\Upsilon^\lambda(z)\right\|^2. \tag{2.16}$$

Before we introduce the smoothed Levenberg-Marquardt method, that will be one of the main focus points of this paper, note that for fixed $\lambda > 0$ and $\mu > 0$,

$$\nabla\Upsilon^\lambda_\mu(z) = \begin{bmatrix} \nabla^2 L^\lambda(z) & \nabla g(x,y)^T & \nabla G(x,y)^T & -\lambda\nabla g(x,y)^T \\ \nabla(\nabla_y \mathcal{L}(z)) & O & O & \nabla_y g(x,y)^T \\ \mathcal{T}^\mu \nabla g(x,y) & \Gamma^\mu & O & O \\ \mathcal{A}^\mu \nabla G(x,y) & O & \mathcal{B}^\mu & O \\ \Theta^\mu \nabla g(x,y) & O & O & \mathcal{K}^\mu \end{bmatrix} \tag{2.17}$$

with the pair $(\mathcal{T}^\mu, \Gamma^\mu)$ defined by $\mathcal{T}^\mu := \text{diag}\,\{\tau^\mu_1, .., \tau^\mu_p\}$ and $\Gamma^\mu := \text{diag}\,\{\gamma^\mu_1, .., \gamma^\mu_p\}$, where

$$\tau^\mu_j := \frac{g_j(x,y)}{\sqrt{u_j^2 + g_j(x,y)^2 + 2\mu}} + 1 \;\text{ and }\; \gamma^\mu_j := \frac{u_j}{\sqrt{u_j^2 + g_j(x,y)^2 + 2\mu}} - 1, \;\; j = 1, \dots p. \tag{2.18}$$

The pairs $(\mathcal{A}^\mu, \mathcal{B}^\mu)$ and $(\Theta^\mu, \mathcal{K}^\mu)$ are defined in a similar way, based on $(v_j, G_j(x,y))$, $j = 1, \dots, q$ and $(w_j, g_j(x,y))$, $j = 1, \dots, p$ respectively.

We now move on to present some definitions that will help us state the algorithm with *line search*. It is well-known that line search helps to choose optimal step length to avoid over-going an optimal solution in the direction d^k and also to globalize the convergence of the method, i.e., have more flexibility on the starting point z^0. We need to calculate the optimal step-length, γ_k. This can be done through minimizing $\Phi^\lambda(z^k + \gamma_k d^k)$, with respect to γ_k, such that

$$\Phi^\lambda(z^k + \gamma_k d^k) \leqslant \Phi^\lambda(z^k) + \sigma\gamma_k \nabla\Phi^\lambda(z^k)^T d^k \;\text{ for }\; 0 < \sigma < 1.$$

That is, we are looking for $\gamma_k = \arg\min_{\gamma\in\mathbb{R}} \|\Upsilon^\lambda(z^k + \gamma d^k)\|^2$. To implement line search, it is standard to use *Armijo condition* that guarantees a decrease at the next iterate.

Definition 2.5. *Fixing d and z, consider the function $\phi_\lambda(\gamma) := \Phi^\lambda(z + \gamma d)$. Then, the* Armijo *condition will be said to hold if $\phi_\lambda(\gamma) \leqslant \phi(0) + \gamma\sigma\phi'_\lambda(\gamma)$ for some $0 < \sigma < 1$.*

The practical implementation of the Armijo condition is based on backtracking.

Definition 2.6. *Let $\rho \in (0,1)$ and $\bar{\gamma} > 0$.* Backtracking *is the process of checking over a sequence $\bar{\gamma}$, $\rho\bar{\gamma}$, $\rho^2\bar{\gamma}, \dots$, until a number γ is found satisfying the Armijo condition.*

Line search is widely used in continuous optimization; see, e.g., [55] for more details. For the implementation in this paper, we start with stepsize $\gamma_0 := 1$; then, if the condition

$$\left\|\Upsilon^\lambda(z^k + \gamma_k d^k)\right\|^2 < \left\|\Upsilon^\lambda(z^k)\right\|^2 + \sigma\gamma_k \nabla\Upsilon^\lambda_\mu(z^k)^T\Upsilon^\lambda(z^k)d^k,$$

is not satisfied, we set $\gamma_k = \gamma_k/2$ and check again until the condition above is satisfied. Sufficient conditions to ensure *global convergence* of the method are known to be satisfied for *backtracking line search*. The algorithm then proceeds as follows:

Algorithm 2.7. *Smoothed Levenberg-Marquardt Method for Bilevel Optimization*

Step 0: *Choose $(\lambda, \mu, K, \epsilon, \alpha_0) \in \left(\mathbb{R}^*_+\right)^5$, $(\rho, \sigma, \gamma_0) \in (0,1)^3$, $z^0 := (x^0, y^0, u^0, v^0, w^0)$, and set $k := 0$.*
Step 1: *If $\left\|\Upsilon^\lambda_\mu(z^k)\right\| < \epsilon$ or $k \geqslant K$, then stop.*
Step 2: *Calculate the Jacobian $\nabla\Upsilon^\lambda_\mu(z^k)$ and subsequently find a vector d^k satisfying*

$$\left(\nabla\Upsilon^\lambda_\mu(z^k)^\top\nabla\Upsilon^\lambda_\mu(z^k) + \alpha_k I\right) d^k = -\nabla\Upsilon^\lambda_\mu(z^k)^\top\Upsilon^\lambda_\mu(z^k), \tag{2.19}$$

where I denotes the identity matrix of appropriate size.
Step 3: ***While*** *$\left\|\Upsilon^\lambda_\mu(z^k + \gamma_k d^k)\right\|^2 \geqslant \left\|\Upsilon^\lambda_\mu(z^k)\right\|^2 + \sigma\gamma_k\nabla\Upsilon^\lambda_\mu(z^k)^T\Upsilon^\lambda_\mu(z^k)d^k$,* ***do*** *$\gamma_k \leftarrow \rho\gamma_k$* ***end.***
Step 4: *Set $z^{k+1} := z^k + \gamma_k d^k$, $k := k+1$, and go to Step 1.*

Note that in Step 0, $\mathbb{R}^*_+ := (0, \infty)$. Before we move on to focus our attention on the practical implementation details of this algorithm, we present its convergence result, which is based on the following selection of the Levenberg-Marquardt (LM) parameter α_k:

$$\alpha_k = \|\Upsilon^\lambda_\mu(z^k)\|^\eta \ \text{ for any choice of } \eta \in [1, 2]\,. \tag{2.20}$$

Theorem 2.8 ([27]). *Consider Algorithm 2.7 with fixed values for the parameters $\lambda > 0$ and $\mu > 0$ and let α_k be defined as in (2.20). Then, the sequence $\{z^k\}$ generated by the algorithm converges quadratically to $\bar{z}$ satisfying $\Upsilon^\lambda_\mu(\bar{z}) = 0$, under the following assumptions:*

1. *$\Upsilon^\lambda_\mu : \mathbb{R}^N \to \mathbb{R}^{N+m}$ is continuously differentiable and $\nabla\Upsilon^\lambda_\mu : \mathbb{R}^N \to \mathbb{R}^{(N+m)\times(N)}$ is locally Lipschitz continuous in a neighbourhood of $\bar{z}$.*

2. *There exists some $C > 0$ and $\delta > 0$ such that*

$$C\text{dist}(z, Z^\lambda_\mu) \leqslant \left\|\Upsilon^\lambda_\mu(z)\right\| \quad \textit{for all} \quad z \in \mathcal{B}(\bar{z}, \delta),$$

where dist *denotes the distance function and Z^λ_μ corresponds to the solution set of equation (2.14).*

For fixed values of $\lambda > 0$ and $\mu > 0$, assumption 1 in this theorem is automatically satisfied if all the functions involved in problem (1.1) are twice continuously differentiable. According to [78], assumption 2 of Theorem 2.8 is fulfilled if the matrix $\nabla\Upsilon^\lambda_\mu$ has a full column rank. Various conditions guarantying that $\nabla\Upsilon^\lambda_\mu$ has a full rank have been developed in [Paper 1]. Below, we present an example of bilevel program satisfying the first and second assumptions of Theorem 2.8.

Example 2.9. Consider the following instance of problem (1.1) from the BOLIB library [85, LamprielloSagratelli2017Ex33]:

$$\begin{array}{rcl} F(x,y) & := & x^2 + (y_1 + y_2)^2, \\ G(x,y) & := & -x + 0.5, \\ f(x,y) & := & y_1, \\ g(x,y) & := & \begin{pmatrix} -x - y_1 - y_2 + 1 \\ -y \end{pmatrix}. \end{array}$$

Obviously, assumption 1 holds. According to [Paper 1], for this example, the columns of $\nabla\Upsilon^\lambda_\mu$ are linearly independent at the solution point

$$\bar{z} := (\bar{x},\, \bar{y}_1,\, \bar{y}_2,\, \bar{u}_1,\, \bar{u}_2,\, \bar{u}_3,\, \bar{v},\, \bar{w}_1,\, \bar{w}_2,\, \bar{w}_3) = (0.5,\, 0,\, 0.5,\, 1,\, \lambda,\, 0,\, 0,\, 0,\, 1,\, 0)$$

with the parameters chosen as $\mu = 2 \times 10^{-2}$ and $\lambda = 10^{-2}$. □

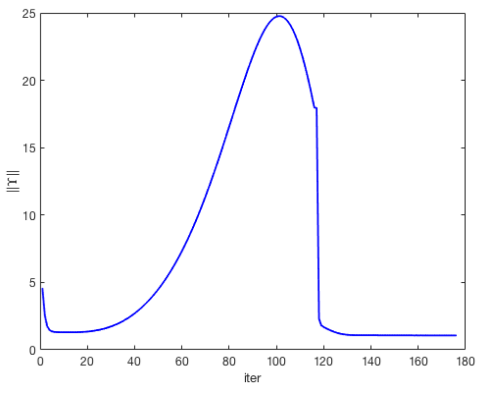
2.2 Practical implementation details

On the selection of the LM parameter α_k, a preliminary analysis based on the BOLIB library test set [85]. It was observed that for almost all the corresponding examples, the choice $\alpha_k := \|\Upsilon^\lambda_\mu(z^k)\|^\eta$ with $\eta \in]1, 2]$ leads to a very poor performance of Algorithm 2.7. The typical behaviour of the algorithm for $\eta \in]1, 2]$ is shown in the following example.

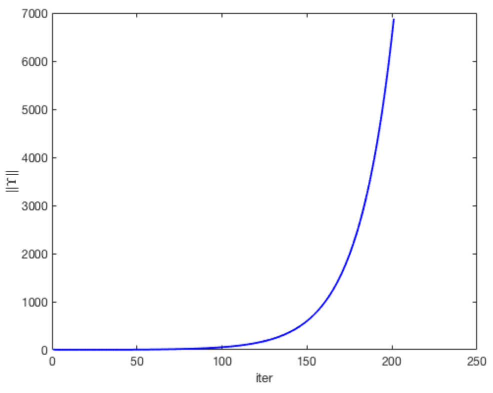
Example 2.10. Consider the following scenario of problem (1.1) from [85, AllendeStill2013]:

$$\begin{aligned}
F(x,y) &:= x_1^2 - 2x_1 + x_2^2 - 2x_2 + y_1^2 + y_2^2,\\
G(x,y) &:= \begin{pmatrix} -x \\ -y \\ x_1 - 2 \end{pmatrix},\\
f(x,y) &:= y_1^2 - 2x_1y_1 + y_2^2 - 2x_2y_2,\\
g(x,y) &:= \begin{pmatrix} (y_1-1)^2 - 0.25 \\ (y_2-1)^2 - 0.25 \end{pmatrix}.
\end{aligned}$$

Figure 9 shows the progression of $\|\Upsilon^\lambda(z^k)\|$ generated from Algorithm 2.7 with α_k selected as in (2.20) while setting $\eta = 1$ and $\eta = 2$, respectively. Clearly, after about 100 iterations $\|\Upsilon^\lambda(z^k)\|$ blows up relatively quickly when $\eta = 2$, while it falls and stabilizes within a certain tolerance for $\eta = 1$.



(a) $\alpha_k := \|\Upsilon^\lambda(z^k)\|$



(b) $\alpha_k := \|\Upsilon^\lambda(z^k)\|^2$

Figure 9: Typical behaviour of Algorithm 2.7 for two scenarios of LM parameter

It is worth noting that the scale on the y-axis of Figure 9(b) is quite large. Hence, it might not be apparent that solutions at the early iterations of the algorithm are much better for $\eta = 1$ compared to the ones obtained in the case where $\eta = 2$. □

Although Algorithm 2.7 is designed to have descent direction, we cannot guarantee that direction is indeed descent at every iteration of the algorithm. In the above example we observe that the algorithm can sometimes make non-descending iterations. There could be a variety of reasons for such behaviour. The most obvious ones are nature of the problem, complexity of the algorithm and specifications of the parameters of the method. However, the most likely explanation for non-descending behaviour is the procedure of varying the penalty parameter λ between the iterations, as this results in solving a different problem at each iteration. Adding this to overdetermined structure of the system and tuning required for the convergence of the algorithm (such as smoothing technique and line-search) could lead to surprising behaviour at some iterations. Note that for almost all of the examples in the BOLIB test set [85], we have a behaviour of Algorithm (2.7) similar to that of Figure 9(b) when $\alpha_k = \|\Upsilon^\lambda(z^k)\|^\eta$ for many different choices of $\eta \in]1, 2]$. It is important to mention that the behaviour of the algorithm is not surprising, as it is well-known in the literature that with $\alpha_k := \|\Upsilon^\lambda_\mu(z^k)\|^2$, the Levenberg–Marquardt faces some potential implementation problems. When the sequence is close to the solution set, $\alpha_k := \|\Upsilon^\lambda_\mu(z^k)\|^2$ could become smaller than the machine precision and hence lose its role as a result of this. On the other hand, when the sequence is far away from the solution set, $\|\Upsilon^\lambda_\mu(z^k)\|^2$ may be very large; making movement to the solution set to be very slow. Hence, from now on, we use $\alpha_k := \|\Upsilon^\lambda(z^k)\|$ $(k = 0, 1, \dots)$ for all the analysis of Algorithm 2.7 conducted in this paper. Note however that there are various other approaches to select α_k in the literature; see, e.g., [5, 6, 26, 27, 44, 78].

It is worth noting that Step 0 of Algorithm 2.7 is crucial for the convergence and performance of the algorithm. There, K represents the maximum number of iteration that we set to $K := 1000$. As for γ, ρ, and σ, they correspond to the parameters of the line search technique described in Definitions 2.5 and 2.6. To be precise, we use the values $\gamma_0 := 1$, $\rho = 0.5$, and $\sigma = 10^{-2}$. The selection of the remaining parameters of the algorithm is going to be carefully addressed in the sequel.

Smoothing parameter μ. First, note that for the practical implementation of Algorithm 2.7, the smoothing process, i.e., the use of the parameter μ is used only where the derivative calculation is necessary. That is, precisely in Step 2 and Step 3. Where the derivative evaluation for Υ^λ_μ is necessary, fixing μ to be a small constant for all iterations can perform better than setting $\mu_k \downarrow 0$ to be a decreasing sequence. We have tried both approaches and it was observed is that our algorithms is indifferent of both of the options. For our algorithms we are going to define decreasing sequence $\mu_k := 0.001/(1.5^k)$. As the idea of the smoothing parameter is to be as small as possible to approximate the original function, we start with the small value and decrease it with every iteration drawing the value of μ to zero, i.e. $\mu_k \downarrow 0$. Smoothing is well-known and popular approach to use for Newton-type methods. Some of the papers (e.g. [31, Paper 1]) report difficulties with running the algorithm with a smoothing parameter being a sequence $\mu_k \downarrow 0$. In such cases authors fix the parameter μ to be a small number (e.g. $\mu := 10^{-11}$) and run algorithm keeping μ constant throughout all iterations. Interestingly, testing the behaviour of constant smoothing parameter and smoothing parameter as decreasing sequence for our algorithm has shown that both options lead to obtaining the same results. This was observed to hold for the algorithm with varying λ, as well as for the algorithm with fixed values of λ . We obtain the same results fixing μ to be a small constant ($\mu := 10^{-11}$) or choosing $\mu_k := 0.001/(1.5^k)$. In the light of this, we stick to what is theoretically right, that is the smoothing sequence $\mu_k := 0.001/(1.5^k)$.

Descent direction check and update. If $\|\Upsilon^\lambda(z^k + \gamma_k d^k)\|^2 < \|\Upsilon^\lambda(z^k)\|^2 + \sigma\gamma_k \nabla\Upsilon^\lambda(z^k)^T\Upsilon^\lambda(z^k)d^k$ we redefine $\gamma_k = \gamma_k/2$ and check again. Remind ourselves that Levenberg-Marquardt direction can be interpreted as a combination of Gauss-Newton direction and steepest descent direction. When $\alpha_k = 0$ Levenberg-Marquardt direction is equivalent to Gauss-Newton direction. Similarly as $\alpha_k \to \infty$ the direction d^k tends to steepest descent direction. With this approach, if the Levenberg-Marquardt direction is not descent at some iteration, we give more weight to the steepest descent direction. Hence, when $\|\Upsilon^\lambda(z^k)\| > \|\Upsilon^\lambda(z^{k-1})\|$ setting $\alpha_{k+1} := 10000\,\|\Upsilon^\lambda(z^k)\|$ has led to an oeverall good performance of Algorithm 2.7 for test set used in this paper.

Stopping Criteria. The primary stopping criteria is that the algorithm is that $\|\Upsilon^\lambda_\mu(z^k)\| < \epsilon$, as requested in Step 1. However, as it will be discussed in detail in the next sections, robust safeguards are needed to deal with ill-behaviours typically due to the size of the penalty parameter λ. Hence, for the practical implementation of the method, we set $\epsilon = 10^{-5}$ and **stop** if one of the following six conditions is satisfied:

1. $\|\Upsilon^\lambda(z^k)\| < \epsilon$,
2. $\big|\, \|\Upsilon^\lambda(z^{k-1})\| - \|\Upsilon^\lambda(z^k)\| \,\big| < 10^{-9}$,
3. $\big|\, \|\Upsilon^\lambda(z^{k-1})\| - \|\Upsilon^\lambda(z^k)\| \,\big| < 10^{-4}$ and iter > 200,
4. $\|\Upsilon^\lambda(z^{k-1})\| - \|\Upsilon^\lambda(z^k)\| < 0$ and $\|\Upsilon^\lambda(z^k)\| < 10$ and iter > 175,
5. $\|\Upsilon^\lambda(z^k)\| < 10^{-2}$ and iter > 500,

6. $\|\Upsilon^\lambda(z^k)\| > 10^2$ and $\mathtt{iter} > 200$.

The additional stopping criteria is important to ensure that algorithm is not running for too long. The danger of running algorithm for too long is that ill-conditioning could occur. Further, we typically observe the pattern that we recover solution earlier than algorithm stops. This appears due to the nature of the overdetermined system. We do not know beforehand the tolerance with which we can solve for $\|\Upsilon^\lambda(z)\|$ as Υ^λ is overdetermined system. Hence, it is hard to select ϵ that would fit all examples and allow to solve examples with efficient tolerance. With the stopping criteria defined above we avoid running unnecessary iterations, retaining the obtained solution. To avoid algorithm running for too long and to prevent λ to become too large, we impose additional stopping criterion 3, 5 or 6 above. The motivation behind this stopping criteria was the observation of the behaviour of the algorithm. For almost all of the examples we observe that after 100-150 iterations we obtain the value reasonably close to the solution but we cannot know beforehand what would be the tolerance of $\|\Upsilon^\lambda(z^k)\|$ to stop. Choosing small ϵ would not always work due to the overdetermined nature of the system being solved. Choosing ϵ too big would lead to worse solutions and possibly not recover some of the solutions. Further, a quick check has shown that ill-conditioning issue typically takes place after 500 iterations for majority of the problems. For these reasons we stop if the improve of the $\mathtt{Error}$ value from step to step becomes too small, $|\,\|\Upsilon^\lambda(z^{k-1})\| - \|\Upsilon^\lambda(z^k)\|\,| < 10^{-4}$, after the algorithm has performed 200 iterations. Since ill-conditioning is likely to happen after 500 iterations we stop if by that time we obtain a reasonably small $\mathtt{Error}$, $\|\Upsilon^\lambda(z^k)\| < 10^{-2}$. Finally, if it turns out that system cannot be solved with a good tolerance, such that we would obtain a reasonably small value of the $\mathtt{Error}$, we stop if the Error after 200 iterations is big, $\|\Upsilon^\lambda(z^k)\| > 10^2$. This way additional stopping criteria plays the role of safeguard to prevent ill-conditioning and also does not allow the algorithm to keep running for too long once a good solution is obtained.

Starting point. The experiments have shown that the algorithm performs much better if the starting point (x^0, y^0) is feasible. As a default setup, we start with $x^0 = 1_n$ and $y^0 = 1_m$. If the default starting point does not satisfy at least one constraint and algorithm diverges, we choose a feasible starting point; see [Supp2] for such choices. To be more precise, if for some i, $G_i(x^0, y^0) > 0$ or for some j we have $g_j(x^0, y^0) > 0$ and the algorithm does not converge to a reasonable solution, we generate a starting point such that $G_i(x^0, y^0) = 0$ or $g_j(x^0, y^0) = 0$. Subsequently, the Lagrange multipliers are initialised at $u^0 = \max\left\{0.01, -g(x^0, y^0)\right\}$, $v^0 = \max\{0.01, -G(x, y)\}$, and $u^0 = w^0$.

To conclude this section, it is important to recall that from the perspective of bilevel optimization, the partial exact penalization parameter λ is the main element of Algorithm 2.7, as it originates from the penalization of the value function constraint $f(x, y) \leqslant \varphi(x)$. Additionally, unlike the other parameters involved in the algorithm, which have benefited from many years of research, as reported above, it remains unknown what is the best way to select λ while solving problem (1.1) via the value function reformulation (1.2). Hence, the focus of the remaining parts of this paper will be focused on the selection of λ and impact on Algorithm 2.7.

3 PARTIAL EXACT PENALTY PARAMETER SELECTION

The aim of this section is to explore the best way to select the penalization parameter λ. Based on Theorem 2.2, we should be getting the solution for some threshold value $\bar{\lambda}$ of the penalty parameter and the algorithm should be returning the value of the solution for any $\lambda \geqslant \bar{\lambda}$. Hence, increasing λ at every iteration seems to be the ideal approach to follow this logic to obtain and retain the solution. Hence, to proceed, we take the increasing sequence $\lambda_k := 0.5 * (1.05)^k$, where k is the number of iterations of the algorithm. The main reason of this choice is that the final value of λ needs not to be too small to recover solutions and not too large to avoid the issue of ill-conditioning. It was observed that going too aggressive with the sequence (e.g., with $\lambda_k := 2^k$) forces the algorithm to diverge. Also, it was observed that for fixed and small values of λ (i.e., $\lambda < 1$), Algorithm 2.7 performs well for many examples. This justifies choosing the starting value for varying parameter

λ to be $\lambda < 1$. Overall, the aim here is vary λ as mentioned above and assess what could be the potential best ranges for selection of the parameter based on our test set from BOLIB [85].

3.1 How far can we go with the value of λ?

We start here by acknowledging that it is very difficult to check that partial calmness holds in practice. Nevertheless, we would like to consider Theorem 2.2 as the base of our analysis, and ask ourselves how large does λ need to be for Algorithm 2.7 to converge. Intuitively, one would think that taking λ as whatever large number should be fine. However, this is usually not the case in practice. One of the main reasons for this is that for too large values of λ algorithm does not behave well. In particular, if we run Algorithm 2.7 with varying λ for too many iterations, the value of the Error blows up at some point and algorithm stops descending. Since we are only increasing λ throughout iterations it is likely, but not definitely, that the explanation of such *ill behaviour* of the algorithm is the issue known as ill conditioning of the penalty functions. Remind ourselves that ill condition refers to one eigenvalue of the Hessian being much larger than the other eigenvalue, which affects the curvature in the negative way for gradient methods [62]. To analyze the ill behaviour in this section, we are going to run the algorithm for $1,000$ iterations with no stopping criteria and let λ vary indefinitely. We would then look at which iteration algorithm blows up and record the value of λ there. Let us define λ_{ill} to be the first value of λ for which ill behaviour is observed for each example, and present for how many examples λ_{ill} took certain values in the table below

Table 2: Starting λ for ill behaviour

λ_{ill}	$\lambda_{ill} < 10^7$	$10^7 < \lambda_{ill} < 10^9$	$10^9 < \lambda_{ill} < 10^{11}$	$10^{11} < \lambda_{ill} < 10^{20}$	Not observed
Examples	1	6	72	11	34

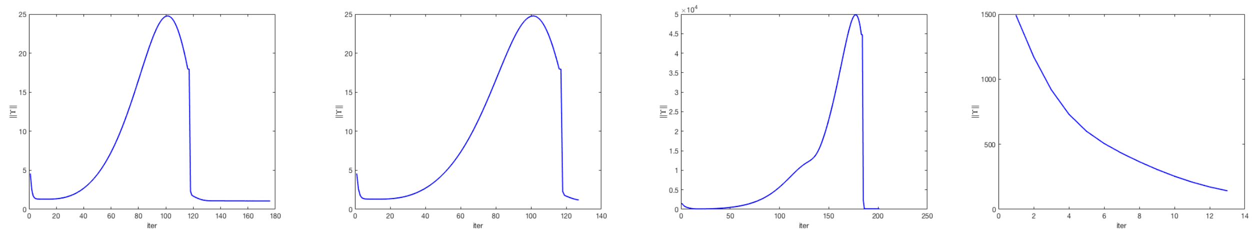
Ill behaviour seems to typically occur after about 500 iterations (where $\lambda_{ill} \approx 10^{10}$), as seen in the table above. For 34 problems ill behaviour was not observed under the scope of 1000 iterations. We also see that for most of the problems (72/124) ill-conditioning happens for $10^9 < \lambda < 10^{11}$. We further observe that algorithm has shown to behave well for the values of penalty parameter $\lambda < 10^9$ with only 7/124 examples demonstrating ill behaviour for such λ. This makes the choice of very large values of λ not attractive at all. Mainly, the analysis shows that choosing $\lambda > 10^9$ could cause algorithm to diverge. We could further recommend that for our method $\lambda \leqslant 10^7$ is very safe choice. This is useful for the choice of fixed λ as we can choose values smaller than 10^7. For the case of varying λ the values are controlled by the introduced stopping criteria. The complete results on the values of λ_{ill} for each example will be presented in Table 3.

Interestingly, 34 out of 124 test problems do not demonstrate signs of any ill behaviour even if we run the algorithm for $1,000$ iterations with $\lambda = 0.5 * 1.05^{iter}$. A potential reason why ill-behaviour is not observed for these examples could just be that the parameter λ does not get large enough after $1,000$ iterations to cause problems for these examples. However, it could also be that the eigenvalues of the Hessian are not affected by large values of λ for these examples. Possibly, elements of the Hessian do not depend on λ at all, as for 20/34 problems the function g is linear in (x, y) or not present in these problems. Let us now proceed to discussing which magnitudes of the penalty parameter λ seem to perform the best for our method.

3.2 Do the values of λ really need to be large?

It is clear from the previous subsection that to reduce the chances for Algorithm 2.7 to diverge or exhibit some ill-behaviour, we should approximately select $\lambda < 10^7$. However, it is still unclear whether only large values of λ would be sensible to ensure a good behaviour of the algorithm. In other words, it is important to know whether relatively small values of λ can lead to a good behaviour for Algorithm 2.7. To assess this, we attempt here to identify inflection points, i.e., values of k where we have $\left\|\Upsilon_{\mu}^{\lambda_k}(z^k)\right\| < \epsilon$ as λ_k varies increasingly as described in the previous subsection. We would then record the value of λ_k at these points. Ideally, we want to get the threshold $\bar{\lambda}$

such that solution is retained for all $\lambda > \bar{\lambda}$ in the sense of Theorem 2.2. To proceed, we extract the information on the final $\text{Error}^* := \|\Upsilon^{\lambda}(z^*)\|$ for each example from [Supp2] and then rerun the algorithm with varying penalty parameter $\lambda := 0.5 * 1.05^k$ with new stopping criterion (i.e., $\text{Error} \leqslant 1.1\text{Error}^*$) while relaxing all of the previous stopping criteria. This way we stop once we observe $\text{Error}_k := \|\Upsilon^{\lambda_k}(z)\|$ close to the Error^* that we obtained in our experiments [Supp2]. It is worth noting, that it would make sense to test only 72/117 (61.54%) of examples, for which algorithm performed well and produced a good solution. This approach can be thought of as finding the inflection point. For instance, if we have an algorithm running as below, we want to stop at the inflection point after 125-130 iterations. The illustration of how we aim stop at the inflection point is presented in Figure 10 (a) and (b) below, where we have $\|\Upsilon^{\lambda}(z)\|$ on the y-axis and iterations on the x-axis.



(a) Complete run of algorithm for `AllendeStill2013` **(b)** Stop at the inflection point for `AllendeStill2013` **(c)** Complete run of algorithm for `Anetal2009` **(d)** Stop at the inflection point for `Anetal2009`

Figure 10: Illustrating the inflection point identification for Examples `AllendeStill2013` and `Anetal2009` from BOLIB [85]

It was observed that for some of the examples it was the case that we got better Error than initial Error^*. For these cases we stopped very early as $\text{Error}_k \leqslant 1.1\text{Error}^*$ was typically met at an early iteration k (where λ was still small), as demonstrated in Figure 10 (c) and (d) above. This demonstrates the disadvantage of λ being an increasing sequence. If the algorithm makes many iterations, the parameter λ keeps increasing without possibility to go back to the smaller values. It turns out that for some examples the smaller value of λ was as good as large values, or even better to recover a solution. For such cases we get good enough Error to satisfy $\text{Error} \leqslant 1.1\text{Error}^*$ when the value of λ is relatively small. The reason for this could be that in our initial experiments algorithm could make too many iterations, where descent direction was not always guaranteed. In practice it seems quite often that small λ is the better option for the algorithm. This further justifies the choice to start from the small value of λ, that is $\lambda_0 < 1$ and increase it slowly.

With the setting to stop whenever $\text{Error}_k \leqslant 1.1\text{Error}^*$ we often stop very early. This way we do not get $\bar{\lambda}$ that represents the inflection point, which we aimed to get. This scenario is demonstrated in Figure 10 (c) an (d) above, where picture (c) has a scale of 10^4 on the y-axis. In this example we were able to obtain $\text{Error} = 136$ after 12 iterations, where $\bar{\lambda}$ is small, while the value of the error when algorithm stopped was $\text{Error}^* = 124$. Although, we can clearly see from Figure 10 (c) that inflection point lies around 190-200 iterations, where the value of λ is much bigger. It is clear that we stopped earlier due to having small value of Error after 12-45 iterations. It was observed that such scenario is typical for the examples in the considered test set. For this reason, we want to introduce λ^* as the *large threshold* λ. The value of λ^* will be used to represent the value of the penalty parameter at the inflection point, where solution starts to be recovered for large values of λ ($\lambda > 6.02$), while we also record $\bar{\lambda}$ as the first (smallest) value of λ for which good solution was obtained. For instance, with λ defined as $\lambda := 0.5 \times 1.05^k$ in Figure 10 (c) we have $\bar{\lambda} = 0.5 \times 1.05^{12}$ and $\lambda^* = 0.5 \times 1.05^{190}$ as we obtain good solution for small λ after 12 iterations and for large λ after 190 iterations. We shall note that the value $\lambda > 6.02$ is the value of the penalty parameter after we make at least 50 iterations as for the case with varying λ we have $\lambda = 0.5 \times 1.05^{51} = 6.02$. The complete results of detecting $\bar{\lambda}$ and λ^* is presented in Table 3 below. It was observed that the behaviour of the method follows the same pattern for majority of the examples. Typically, we get a good solution retaining for a few small values of λ, then value of the Error blows up and takes some iterations to start decreasing, coming back to obtaining and retaining a good solution for large values of λ. Such pattern is clearly demonstrated in Figure 10 (c). Such behaviour is interesting as classically only large values of penalty parameter are considered to be good [11, 59, 61], which coincides with the result of Lemma 2.2. As mentioned in [33] some methods require penalty parameter to increase to infinity to obtain convergence.

Table 3: Ill behaviour and two thresholds for λ

Problem number	**Problem name**	**iter$_{ill}$**	λ_{ill}	$\bar{\lambda}$	λ^*	iter for λ^*
1	AiyoshiShimizu1984Ex2	495	1.54e+10	$9.92 * 10^4$	$9.92 * 10^4$	250
2	AllendeStill2013	Not observed	NA	245	245	127
3	AnEtal2009	Not observed	NA	-	-	-
4	Bard1988Ex1	520	5.22e+10	0.608	245	127
5	Bard1988Ex2	536	1.14e+11	-	-	-
6	Bard1988Ex3	Not observed	NA	0.525	54.1	96
7	Bard1991Ex1	Not observed	NA	0.608	183	121
8	BardBook1998	502	2.17e+10	-	-	-
9	CalamaiVicente1994a	476	6.1e+09	0.608	27.3	82
10	CalamaiVicente1994b	490	1.21e+10	0.739	79.9	104
11	CalamaiVicente1994c	490	1.21e+10	-	-	-
12	CalveteGale1999P1	538	1.26e+11	0.525	$1.57 * 10^3$	165
13	ClarkWesterberg1990a	520	5.22e+10	-	-	-
14	Colson2002BIPA1	510	3.2e+10	792	792	151
15	Colson2002BIPA2	900	5.88e+18	1.33	$1.65 * 10^3$	166
16	Colson2002BIPA3	110	107	-	-	-
17	Colson2002BIPA4	Not observed	NA	0.551	$2.68 * 10^3$	176
18	Colson2002BIPA5	550	2.25e+11	-	-	-
19	Dempe1992a	Not observed	NA	-	-	-
20	Dempe1992b	Not observed	NA	0.551	$1.29 * 10^3$	161
21	DempeDutta2012Ex24	Not observed	NA	-	-	-
22	DempeDutta2012Ex31	Not observed	NA	0.525	137	115
23	DempeEtal2012	470	4.55e+09	-	-	-
24	DempeFranke2011Ex41	492	1.33e+10	0.704	44.5	92
25	DempeFranke2011Ex42	495	1.54e+10	0.67	54.1	96
26	DempeFranke2014Ex38	495	1.54e+10	0.551	79.9	104
27	DempeLohse2011Ex31a	502	2.17e+10	0.525	6.02	51
28	DempeLohse2011Ex31b	510	3.2e+10	-	-	-
29	DeSilva1978	495	1.54e+10	0.551	69	101
30	FalkLiu1995	440	1.05e+09	$1.1 * 10^4$	$1.1 * 10^4$	205
31	FloudasEtal2013	505	2.51e+10	0.525	183	121
32	FloudasZlobec1998	510	3.2e+10	-	-	-
33	GumusFloudas2001Ex1	530	8.5e+10	-	-	-
34	GumusFloudas2001Ex3	510	3.2e+10	-	-	-
35	GumusFloudas2001Ex4	502	2.17e+10	-	-	-
36	GumusFloudas2001Ex5	495	1.54e+10	3.04	38.4	89
37	HatzEtal2013	Not observed	NA	0.608	6.02	51
38	HendersonQuandt1958	Not observed	NA	$5.64 * 10^7$	$5.64 * 10^7$	380
39	HenrionSurowiec2011	Not observed	NA	0.551	6.02	51
40	IshizukaAiyoshi1992a	495	1.54e+10	-	-	-
41	KleniatiAdjiman2014Ex3	445	1.34e+09	-	-	-
42	KleniatiAdjiman2014Ex4	485	9.46e+09	0.739	42.4	91
43	LamparSagrat2017Ex23	Not observed	NA	0.525	137	115
44	LamparSagrat2017Ex31	Not observed	NA	0.525	6.02	51
45	LamparSagrat2017Ex32	Not observed	NA	0.551	284	130
46	LamparSagrat2017Ex33	495	1.54e+10	0.551	102	109
47	LamparSagrat2017Ex35	Not observed	NA	0.525	298	131
48	LucchettiEtal1987	495	1.54e+10	0.525	6.02	51
49	LuDebSinha2016a	Not observed	NA	0.943	6.02	51
50	LuDebSinha2016b	Not observed	NA	0.67	6.02	51
51	LuDebSinha2016c	Not observed	NA	-	-	-
52	LuDebSinha2016d	890	3.61e+18	-	-	-
53	LuDebSinha2016e	900	5.88e+18	-	-	-
54	LuDebSinha2016f	Not observed	NA	-	-	-
55	MacalHurter1997	Not observed	NA	0.551	6.02	51

56	Mirrlees1999	Not observed	NA	0.579	6.02	51
57	MitsosBarton2006Ex38	398	1.36e+08	6.64	7.32	55
58	MitsosBarton2006Ex39	400	1.5e+08	-	-	-
59	MitsosBarton2006Ex310	470	4.55e+09	56.8	56.8	97
60	MitsosBarton2006Ex311	452	1.89e+09	-	-	-
61	MitsosBarton2006Ex312	470	4.55e+09	0.99	6.02	51
62	MitsosBarton2006Ex313	505	2.51e+10	0.579	54.1	96
63	MitsosBarton2006Ex314	460	2.79e+09	0.855	11.9	65
64	MitsosBarton2006Ex315	470	4.55e+09	26	56.8	97
65	MitsosBarton2006Ex316	Not observed	NA	1.33	6.02	51
66	MitsosBarton2006Ex317	420	3.97e+08	3.7	6.02	51
67	MitsosBarton2006Ex318	Not observed	NA	1.04	6.02	51
68	MitsosBarton2006Ex319	398	1.36e+08	-	-	-
69	MitsosBarton2006Ex320	485	9.46e+09	0.943	6.32	52
70	MitsosBarton2006Ex321	452	1.89e+09	1.09	10.3	62
71	MitsosBarton2006Ex322	470	4.55e+09	0.99	20.4	76
72	MitsosBarton2006Ex323	505	2.51e+10	-	-	-
73	MitsosBarton2006Ex324	495	1.54e+10	0.855	6.02	51
74	MitsosBarton2006Ex325	505	2.51e+10	-	-	-
75	MitsosBarton2006Ex326	505	2.51e+10	-	-	-
76	MitsosBarton2006Ex327	475	5.81e+09	1.26	7.32	55
77	MitsosBarton2006Ex328	510	3.2e+10	-	-	-
78	MorganPatrone2006a	500	1.97e+10	0.525	6.02	51
79	MorganPatrone2006b	505	2.51e+10	0.551	6.02	51
80	MorganPatrone2006c	470	4.55e+09	56.8	56.8	97
81	MuuQuy2003Ex1	Not observed	NA	-	-	-
82	MuuQuy2003Ex2	Not observed	NA	-	-	-
83	NieEtal2017Ex34	520	5.22e+10	0.551	118	112
84	NieEtal2017Ex52	Not observed	NA	-	-	-
85	NieEtal2017Ex54	495	1.54e+10	-	-	-
86	NieEtal2017Ex57	850	5.13e+17	-	-	-
87	NieEtal2017Ex58	780	1.69e+16	-	-	-
88	NieEtal2017Ex61	Not observed	NA	-	-	-
89	Outrata1990Ex1a	505	2.51e+10	0.608	192	122
90	Outrata1990Ex1b	510	3.2e+10	0.551	223	125
91	Outrata1990Ex1c	495	1.54e+10	0.99	718	149
92	Outrata1990Ex1d	495	1.54e+10	-	-	-
93	Outrata1990Ex1e	495	1.54e+10	0.855	873	153
94	Outrata1990Ex2a	520	5.22e+10	0.551	245	127
95	Outrata1990Ex2b	470	4.55e+09	1.78	6.02	51
96	Outrata1990Ex2c	510	3.2e+10	0.551	72.5	102
97	Outrata1990Ex2d	520	5.22e+10	-	-	-
98	Outrata1990Ex2e	470	4.55e+09	0.898	6.02	51
99	Outrata1993Ex31	520	5.22e+10	0.551	144	116
100	Outrata1993Ex32	680	1.28e+14	-	-	-
101	Outrata1994Ex31	910	9.58e+18	-	-	-
102	OutrataCervinka2009	505	2.51e+10	-	-	-
103	PaulaviciusEtal2017a	400	1.5e+08	107	107	110
104	PaulaviciusEtal2017b	490	1.21e+10	-	-	-
105	SahinCiric1998Ex2	510	3.2e+10	-	-	-
106	ShimizuAiyoshi1981Ex1	495	1.54e+10	46.7	46.7	93
107	ShimizuAiyoshi1981Ex2	520	5.22e+10	-	-	-
108	ShimizuEtal1997a	910	9.58e+18	0.638	212	124
109	ShimizuEtal1997b	Not observed	NA	-	-	-
110	SinhaMaloDeb2014TP3	Not observed	NA	0.525	651	147
111	SinhaMaloDeb2014TP6	530	8.5e+10	-	-	-
112	SinhaMaloDeb2014TP7	Not observed	NA	$5.75 * 10^{10}$	$5.75 * 10^{10}$	522
113	SinhaMaloDeb2014TP8	520	5.22e+10	-	-	-
114	SinhaMaloDeb2014TP9	505	2.51e+10	-	-	-
115	SinhaMaloDeb2014TP10	480	7.41e+09	-	-	-
116	TuyEtal2007	495	1.54e+10	2.9	16.8	72
117	Vogel2002	Not observed	NA	-	-	-
118	WanWangLv2011	520	5.22e+10	-	-	-

119	YeZhu2010Ex42	Not observed	NA	0.814	6.02	51
120	YeZhu2010Ex43	Not observed	NA	4.49	6.02	51
121	Yezza1996Ex31	460	2.79e+09	223	223	125
122	Yezza1996Ex41	490	1.21e+10	0.608	46.7	93
123	Zlobec2001a	360	2.12e+07	-	-	-
124	Zlobec2001b	495	1.54e+10	-	-	-

We are now going to proceed with finding the threshold of λ for which we start getting a solution and retain the value for larger values of λ. We are going to proceed with the technique of finding small threshold $\bar{\lambda}$ and large threshold λ^*, which was briefly discussed earlier. To find the threshold we first extract the value of the final Error^* for each example from [Supp2]. To find $\bar{\lambda}$ we run the algorithm with λ being defined as $\lambda := 0.5 \times 1.05^k$, with the new stopping criteria:

$$\textbf{Stop if} \quad \mathrm{Error} \leqslant 1.1\mathrm{Error}^*.$$

Of course, we also relax all of the previous stopping criteria, as Error is the main measure here and we know that desired value of Error exists. We then define $\bar{\lambda} := 0.5 \times 1.05^{\bar{k}}$, where $\bar{k}$ is the number of iterations after stopping whenever we get $\mathrm{Error} < 1.1\mathrm{Error}^*$. For most of the cases we detect $\bar{\lambda}$ early due to a good solution after the first few iterations in the same manner as shown in Figure 10 (c). Since for many examples we satisfy condition $\mathrm{Error} \leqslant 1.1\mathrm{Error}^*$ for early iterations (before the inflection point is achieved), we further introduce λ^*, *the large threshold* λ. The value of λ^* will be used to represent the value of the penalty parameter at the inflection point, where solution starts to be recovered for large values of λ. This will be obtained in the same way as $\bar{\lambda}$ with the only difference that we additionally impose the condition to stop after at least 50 iterations. To obtain λ^* we run the algorithm with $\lambda := 0.5 \times 1.05^k$ and the following stopping criteria:

$$\textbf{Stop if} \quad \mathrm{Error} \leqslant 1.1\mathrm{Error}^* \quad \& \quad \mathrm{iter} > 50.$$

Then the large threshold is defined as $\lambda^* := 0.5 \times 1.05^{k^*}$, where k^* is the number of iterations after stopping whenever we get $\mathrm{Error} < 1.1\mathrm{Error}^*$ and $k > 50$. This way $\bar{\lambda}$ represents the first (smallest) value of λ for which good solution was obtained, while λ^* represent the actual threshold after which solution is retained for large values of λ. The demonstration of stopping at the inflection point for large threshold λ^* was shown in Figure 10 (b) and for small threshold $\bar{\lambda}$ in Figure 10 (d).

It makes sense to test only the examples where algorithm performed well and produced a good solution. For the rest of the examples, the value of Error^* would not make sense as the measure to stop, and we do not obtain good solutions for these examples by the algorithm anyway. From the optimistic perspective we could consider recovered solutions to be the solutions for which the optimal value of upper-level objective was recovered with some prescribed tolerance. Taking the tolerance of 20%, the total amount of recovered solutions by the method with varying λ is 72/117 (61.54%) for the cases where solution was reported in BOLIB [85]. This result will be shown in more details in Section 4.1. Let us look at the thresholds $\bar{\lambda}$ and λ^* for these examples in the figure below, where the value of λ is shown on $y - axis$ and the example numbers on the $x - axis$.

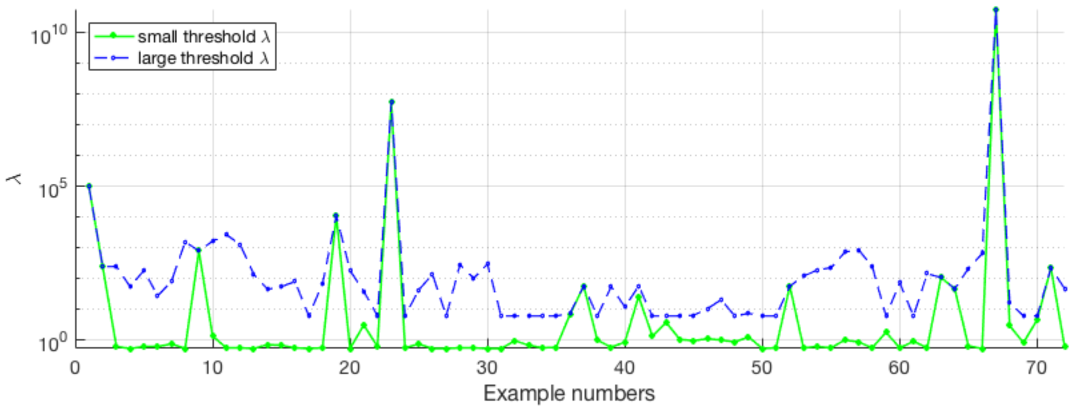


Figure 11: Small threshold $\bar{\lambda}$ and large threshold λ^* for examples with good solutions

For 68/72 problems we observe that large threshold of the penalty parameter has the value $\lambda^* \leqslant 10^4$, which shows that we usually can obtain a good solution before 205 iterations. This further justifies stopping algorithm after 200 iterations if there is no significant step to step improvement. As for the main outcome of Figure 11, we observe that small threshold is smaller than large threshold ($\bar{\lambda} < \lambda^*$) for 59/72 (82%) problems. This clearly shows that for majority of the problems, for which we recover solution with $\lambda := 0.5 \times 1.05^k$, we obtain a good solution for small λ as well as for large λ. This demonstrates that small λ could in principle be good for the method. For the rest 18% of the problems we have $\bar{\lambda} = \lambda^*$, meaning that good solution was not obtained for $\lambda < 6$ for these examples. This also means that we typically obtain good solution for small values of λ and for large values of λ, but not for the medium values ($\bar{\lambda} < \lambda < \lambda^*$).

As for the general observations of Figure 11, for 42/72 (58.33%) examples we observed that the large threshold λ^* is located somewhere in between $90 - 176$ iterations with $40 < \lambda^* < 2680$. For 7/72 problems threshold is in the range $6.02 < \lambda^* \leqslant 40$, and for only 4/72 problems $\lambda^* > 1.1 \times 10^4$. Once again, this justifies that typically λ does not need to be large. It also suggests the optimal values of λ for the tested examples, at least for our solution method. We observe that for 19/72 problems we get $\lambda^* = 51$, which should be treated carefully as this could mean that the inflection point could possibly be achieved before 50 for these examples. Nevertheless, $\lambda^* = 6.02$ is still a good value of penalty parameter for these examples as solutions are retained for $\lambda > \lambda^*$.

As going to be observed in Section 4 we could actually argue that smaller values of λ work better for our method not only for varying λ but also for fixed λ. Together with the fact that we often have the behaviour as demonstrated in Figure 10 (c), it follows that small λ could be more attractive for the method we implement. We even get better values of $\mathtt{Error}$ and better solutions for small values of λ for some examples. Hence we draw the conclusion that small values of λ can generate good solutions. Since it is typical to use large values of λ for other penalization methods (e.g. in [8, 11, 59, 61]), it is interesting what could be the reasons that small λ worked better for our case. This could be due to the specific nature of the method, or due to the fact that we do not do full penalization in the usual sense. Other reason could come from the structure of the problems in the test set. The exact reason of why such behaviour was observed remains an open question. What is important is that this could possibly be the case that small values of λ would be good for some other penalty methods and optimization problems of different nature. This result contradicts typical choice of large penalty parameter for general penalization methods for optimization problems. As the conclusion for our framework, we can claim that for our method λ needs not to be large.

3.3 Partially calm examples

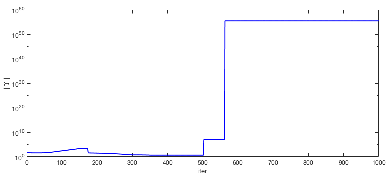
Intuitively, one would think that for partially calm examples, Algorithm 2.7 would behave well, in the sense that varying λ increasingly would lead to a good convergence behaviour. To show that it is not necessarily the case, we start by considering the following result identifying a class of bilevel program of the form (1.1) that is automatically partially calm.

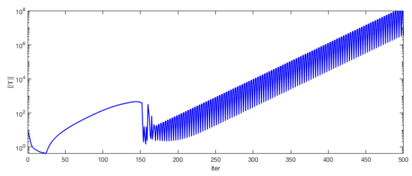
Theorem 3.1 ([52]). *Consider a bilevel program (1.1), where G is independent of y and the lower-level optimal solution map is defined as follows, with $c \in \mathbb{R}^m$, $d \in \mathbb{R}^p$, $B \in \mathbb{R}^{p\times m}$, and $A : \mathbb{R}^n \to \mathbb{R}^p$:*

$$S(x) := \arg\min_y \left\{c^T y |\ A(x) + By \leqslant d\right\}. \tag{3.1}$$

In this case, problem (1.1) is partially calm at any of its local optimal solutions.

Examples 8, 40, 43, 45, 46, 188, and 123 in the BOLIB library (see Table 3) are of the form described in this result. The expectation is that these examples will follow the pattern of retaining solution after some threshold, that is for $\lambda > \lambda^*$, as they fit the theoretical structure behind the penalty approach as described in Theorem 2.2. Note that all of these examples follow the pattern shown in Figure 9(a). However, if we relax the stopping criteria used to mitigate the effects of ill-conditioning, as discussed in the previous two subsections, varying λ for 1000 iterations for these seven partially calm examples leads to the 3 typical scenarios demonstrated in Figure 12.


(a) Example 8


(b) Example 123

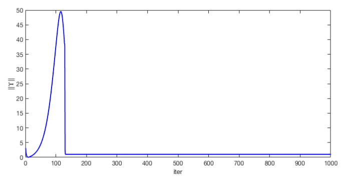

(c) Example 45

Figure 12: (a) and (b) are obtained for 1000 iterations, while (c) is based on 500 iterations.

In the first case of Figure 12, we can clearly see the algorithm is performing well, retaining the solution for the number of iteration, but then blows up at one point (after 500 iterations) and never goes back to reasonable solution values. Examples 40 and 46 also follow this pattern. Example 123 (second picture in Figure 12) shows a slightly different picture, where the zig-zagging pattern is observed. Algorithm 2.7 blows up at some point and starts zig-zagging away from the solution after obtaining it for a smaller value of λ. Zig-zagging is very common issue in penalty methods and often caused by ill-conditioning [55]. Note that Example 118 exhibits a similar behaviour. This is somewhat similar to scenario 1. However, we put this separately as zig-zagging issue is often referred to as the danger that could be caused by ill-conditioning of a penalty function. The last picture of Figure 12 shows a case where Algorithm 2.7 runs very well without any ill-behaviour observed for all the 1000 iterations. It could be possible that the algorithm could blow up after more iterations if we keep increasing λ. It could also be possible that ill-conditioning does not occur for this example at all, as the Hessian of Υ^λ (2.10) is not affected by λ. Out of the seven BOLIB problems considered here, only examples 43 and 45 follows this pattern.

4 PERFORMANCE COMPARISON FOR THE LEVENBERG–MARQUARDT METHOD UNDER FIXED AND VARYING PARTIAL EXACT PENALTY PARAMETER

Considering the behaviour of Algorithm 2.7 based on the nature of the parameter λ, the aim of this section is to provide a thorough comparison of two scenarios: (a) varying λ and (b) fixed values of λ. As in the previous section, the examples used for the experiments are from the Bilevel Optimization LIBrary of Test Problems (BOLIB) [85], which contains 124 nonlinear examples. The experiments are run in MATLAB, version R2016b, on MACI64. Here, we present a summary of the results obtained; more details for each example are reported in the supplementary material [Supp2]. It is important to mention that algorithm always converges, that is algorithm never diverges and always produces an output. This brings us to a strong advantage of the method compared to the method considered in [Paper 1], where algorithm could diverge for some values of λ for some examples.

For Step 0 of Algorithm 2.7 we set the tolerance to $\epsilon := 10^{-5}$ and the maximum number of iterations to be $K := 1000$. We also choose $\alpha_0 := \left\|\Upsilon^\lambda(z^0)\right\|$, $\gamma_0 := 1$, $\rho = 0.5$, and $\sigma = 10^{-2}$. The selection of σ is based on the overall performance of the algorithm while the other parameters are standard in the literature. For the numerical implementation we calculate the direction d^k by solving (2.19) with Gaussian elimination. Following the discussion from Section 3, two approaches for selecting penalty parameter are tested. For the approach with varying λ we define penalty parameter as $\lambda := 0.5 \times 1.05^k$, where k is the number of iterations. For the approach with fixed values of penalty parameter, ten different values of the penalty parameter are used for all the experiments; i.e., $\lambda \in \{10^6, 10^5, ..., 10^{-3}\}$, see [Supp2] for details of the values of each solution for a selection of λ. For the fixed values of λ one could choose best λ for each example to see if at least one of the selected values worked well to recover the solution. After running the experiments for all values of $\lambda \in \{10^6, 10^5, 10^4, ..., 10^{-3}\}$, the best one is chosen i.e. for which the best feasible solution is produced for the particular problem by the tested algorithms. We judge best λ by the best feasible value of upper-level objective function.

4.1 Accuracy of the upper-level objective function

Here, we compare the values of the upper-level objective functions at points computed by the Levenberg-Marquardt algorithm with fixed λ and varying λ. For the comparison purpose, we focus our attention only on 117 BOLIB examples [85], as solutions are not known for the other seven problems. To proceed, let $\bar{F}_A$ be the value of upper-level objective function at the point $(\bar{x}, \bar{y})$ obtained by the algorithm and $\bar{F}_K$ the value of this function at the known best solution point reported in the literature (see corresponding references in [85]). We consider all fixed $\lambda \in \{10^6, 10^5, ..., 10^{-3}\}$ and varying λ in one graph and present the results in Figure 13 below, where we have the relative error $(\bar{F}_A - \bar{F}_K)/(1 + |\bar{F}_K|)$ on the $y-axis$ and number of examples on the x-axis, starting from 30th example. We further plot the results for the best fixed value of λ. The graph is plotted in the order of increasing error.

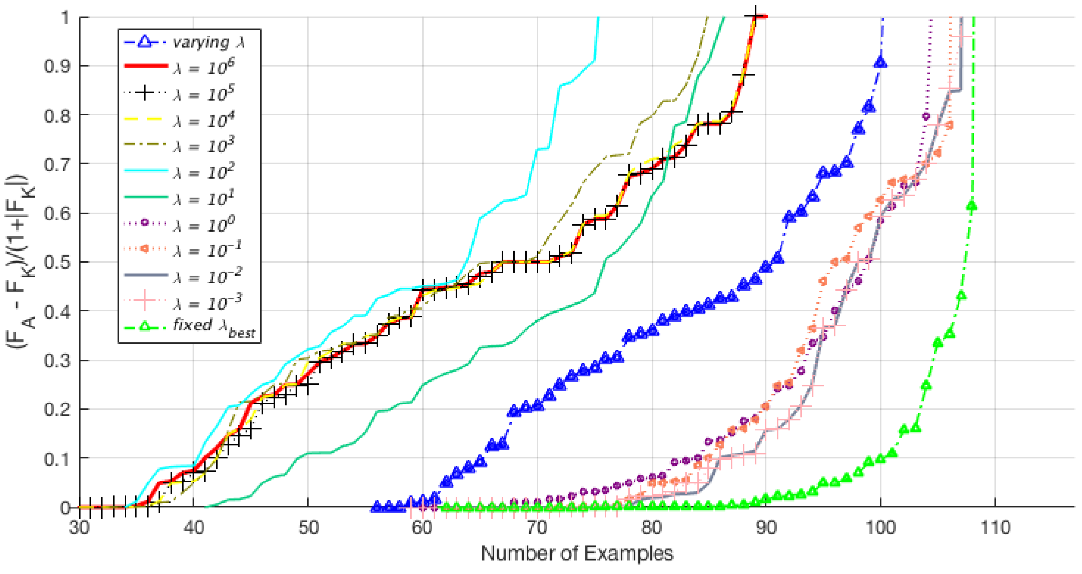


Figure 13: Error of the upper-level objective value for examples with known solutions

From the Figure 13 above we can clearly see that much more solutions were recovered for the small values of fixed λ than for large values. For instance with the allowable accuracy error of $\leqslant 20\%$ we recover solutions for 78.63% for fixed $\lambda \in \{10^{-2}, 10^{-3}\}$, while for $\lambda \in \{10^6, 10^5, 10^4, 10^3, 10^2\}$ we recover at most 40.17% solutions. Interestingly, the worst performance is observed for fixed $\lambda = 100$. With the varying λ we observe that algorithm performed averagely in comparison between large and small fixed values of λ, recovering 59.83% of the solutions with the accuracy error of $\leqslant 20\%$. It is worth saying that implementing Algorithm 2.7 with $\lambda := 0.5 \times 1.05^k$ still recovers over half of the solutions, which is not too bad. However, fixing λ to be small recovers way more solutions, which shows that varying λ is not the most efficient option for our case.

It was further observed that for some examples only $\lambda \geqslant 10^3$ performed well, while for others small values ($\lambda < 1$) showed good performance. If we were able to pick best fixed λ for each example, we would obtain negligible (less than 10%) error for upper-level objective function for 85.47% of the tested problems. With the accuracy error of $\leqslant 25\%$ our algorithm recovered solutions for 88.9% of the problems for the best fixed λ and for 61.54% with varying λ. This means that if one can choose the best fixed λ from the set of different values, fixing λ is much more attractive for the algorithm. It was further observed that for some examples only $\lambda \geqslant 10^3$ performed well, while for others small values of λ ($\lambda < 1$) showed good performance. However, if one does not have a way to choose the best value or a set of potential values cannot be constructed efficiently for certain problems, varying λ could be a better option to choose. Nevertheless, for the test set of small problems from BOLIB [85], fixing λ to be small performed much better than varying λ as increasing sequence. Further, if one could run the algorithm for all fixed λ and was able to choose the best one, the algorithm with fixed λ performs extremely well compared to varying λ. In other words, algorithm almost always finds a good solution for at least one value of fixed $\lambda \in \{10^6, 10^5, ..., 10^{-3}\}$.

4.2 Feasibility check

Considering the structure of the feasible set of problem (1.2), it is critical to check whether the points computed by our algorithms satisfy the value function constraint $f(x, y) \leqslant \varphi(x)$, as it is not explicitly included in the expression of Υ^λ (2.10). If the lower-level problem is convex in y and a solution generated by our algorithms satisfies (2.5) and (2.8), then it will verify the value function constraint. Conversely, to guaranty that a point (x, y) such that $y \in S(x)$ satisfies (2.5) and (2.8), a constraint qualification (CQ) is necessary. Note that conditions (2.5) and (2.8) are incorporated in the stopping criterion of Algorithm 2.7. To check whether the points obtained are feasible, we first identify the BOLIB examples, where the lower-level problem is convex w.r.t. y. As shown in [Paper 1] it turns out that a significant number of test examples have linear lower-level constraints. For these examples, the lower-level convexity is automatically satisfied. We detect 49 examples for which some of these assumptions are not satisfied, that is the problems having non-convex lower-level objective or some of the lower-level constraints being nonconvex. For these examples, we compare the obtained solutions with the known ones from the literature. Let f_A stand for $f(\bar{x}, \bar{y})$ obtained by one of the tested algorithms and f_K to be the known optimal value of lower-level objective function. In the graph below we have the lower-level relative error, $(f_A - f_K)/(1 + |f_K|)$, on the y-axis, where the error is plotted in increasing order. In Figure 14 below we present results for all fixed $\lambda \in \{10^6, 10^5, ..., 10^{-3}\}$ as well as varying λ defined as $\lambda := 0.5 \times 1.05^k$.

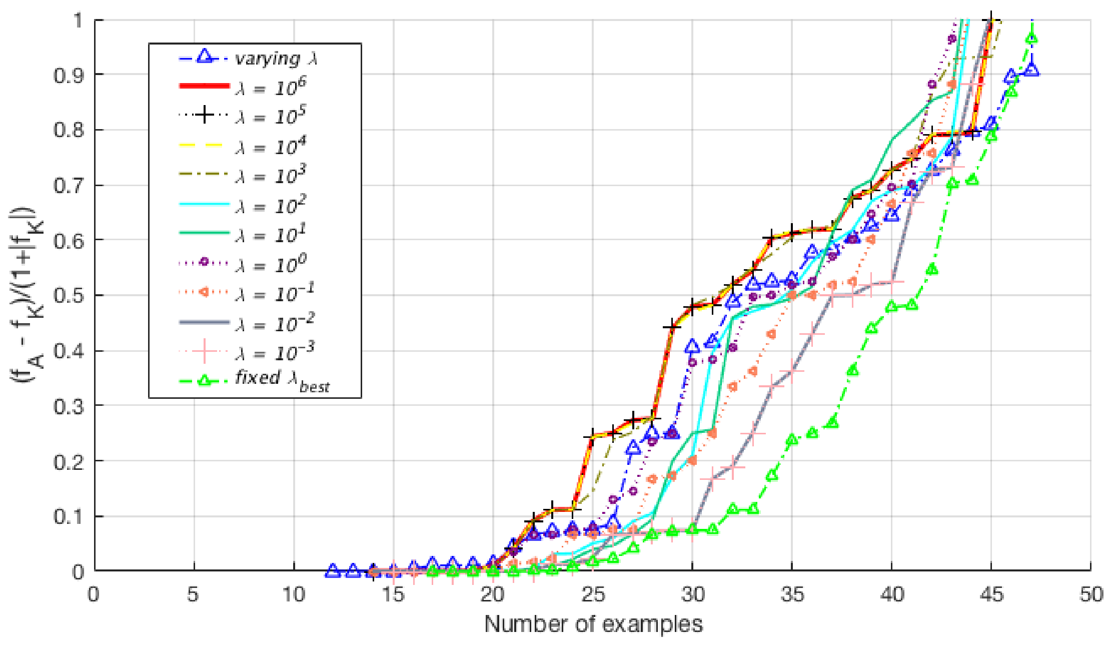


Figure 14: Feasibility error for the lower-level problem in increasing order

From the Figure 14 above we can see that for 20 problems the relative error of lower-level objective is negligible ($< 5\%$) for all values of fixed λ and varying λ. We have seen that convexity and a CQ hold for the lower-level hold for 74 test examples. We consider solutions for these problems to be feasible for the lower-level problem. Taking satisfying feasibility error to be $< 20\%$ and using information from the graph above, we claim that feasibility is satisfied for at most 100 (80.65%) problems for fixed $\lambda \in \{10^6, 10^5, 10^4, 10^3\}$, for $101 - 104$ ($81.45 - 83.87\%$) problems for $\lambda \in \{10^3, 10^2, 10^1, 10^0, 10^{-1}\}$ and for 106 (85.48%) problems for $\lambda \in \{10^{-2}, 10^{-3}\}$. We further observe that feasibility is satisfied for 101 (81.4%) problems for varying λ. Considering we could choose best fixed λ for each of the examples, we could also claim that feasibility is satisfied for 108 (87.1%) problems for best fixed λ.

From Figure 14 we note that slightly better feasibility was observed for smaller values of fixed λ than for the big ones and that varying λ has shown average performance between these magnitudes in terms of the feasibility.

4.3 Experimental order of convergence

Recall that the experimental order of convergence (EOC) is defined by

$$\mathrm{EOC} := \max\left\{\frac{\log\|\Upsilon^{\lambda}(z^{K-1})\|}{\log\|\Upsilon^{\lambda}(z^{K-2})\|}, \frac{\log\|\Upsilon^{\lambda}(z^{K})\|}{\log\|\Upsilon^{\lambda}(z^{K-1})\|}\right\},$$

where K is the number of the last iteration [31]. If $K = 1$, no EOC will be calculated (EOC$= \infty$). EOC is important to estimate the local behaviour of the algorithm and to show whether this practical convergence reflects the theoretical convergence result stated earlier. Let us consider EOC for fixed $\lambda \in \{10^6, ..., 10^{-3}\}$ and for varying λ ($\lambda = 0.5 \times 1.05^k$) in Figure 15 below.

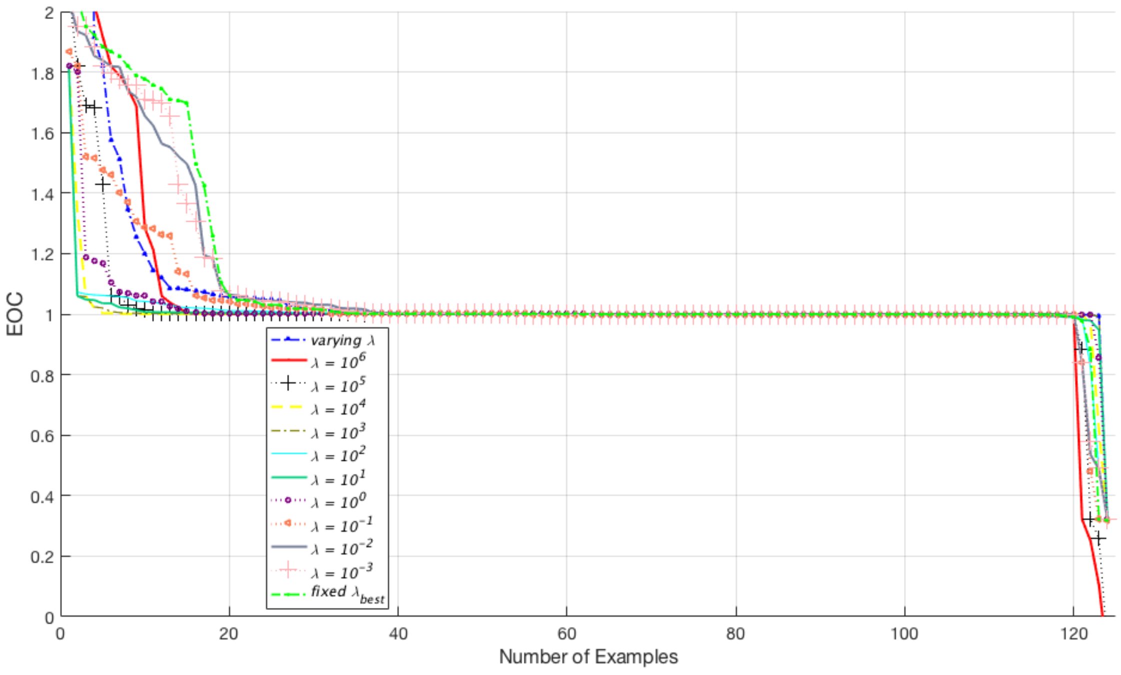


Figure 15: Observed EOC at the last iterations for all examples (in decreasing order)

It is clear from this picture that for most of the examples our method has shown linear experimental convergence. This is slightly below the quadratic convergence established by Theorem 2.8. It is however important to note that the method always converges, although the obtained solution is not necessarily optimal for the problem. There are a few examples that shown better convergence for each value of λ, with the best ones being $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^6\}$ as seen in the figure above. These fixed values have shown slightly better EOC performance than varying λ. Varying λ showed slightly better convergence than fixed $\lambda \in \{10^0, 10^1, 10^2, 10^3, 10^4, 10^5\}$. EOC bigger than 1.2 has been obtained for less than 5 (4.03 %) examples for fixed $\lambda \in \{10^0, 10^1, 10^2, 10^3, 10^4, 10^5\}$, while varying λ showed such EOC for 11 (8.87%) examples. Fixed $\lambda = 10^6$ has shown almost the same result as varying λ with rate of convergence greater than 1.2 for 12 (9.67%) examples, while $\lambda = 10^{-1}$ has demonstrated such EOC for 14 (11.29%) examples and $\lambda \in \{10^{-2}, 10^{-3}\}$ for 17 (13.71 %) examples. Finally, in the graph above, we can see that for all values of λ only a few ($\leqslant 4/124$) examples have worse than linear convergence.

4.4 Line search stepsize

Let us now look at the line search stepsize, γ_k, at the last step of the algorithm for each example. Consider all fixed λ and varying λ in Figure 16 below. This is quite important to know two things. Firstly, how often line search was used at the last iteration, that is how often implementation of line search was clearly important. Secondly, as main convergence results are for the pure method this would be demonstrative to note how often the pure (full) step was made at the last iteration. This can then be compared with the experimental convergence results in the previous subsection, namely with Figure 15.

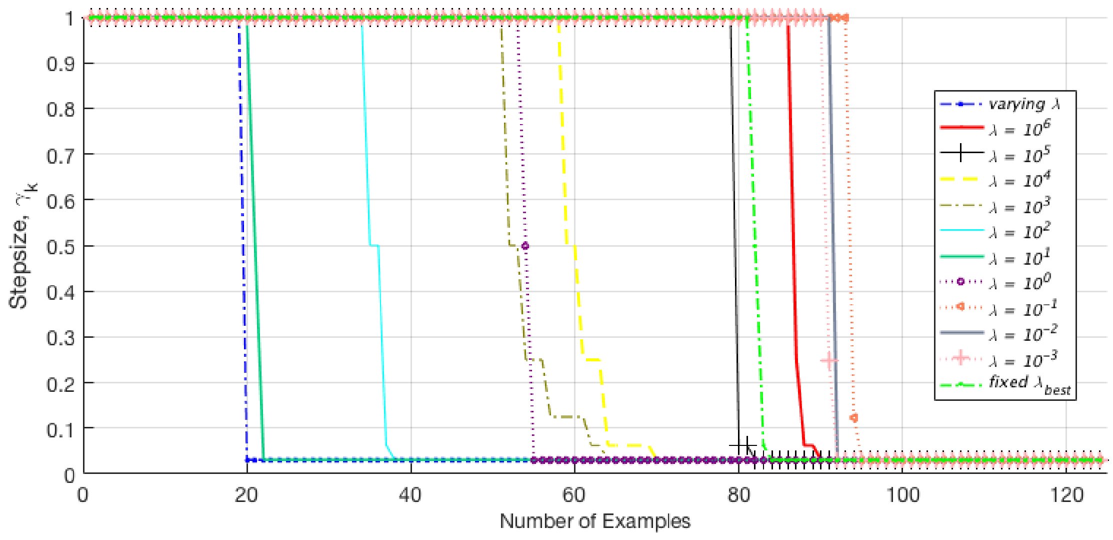


Figure 16: Stepsize made at the last iteration for all examples (in decreasing order)

In the figure above whenever stepsize on the y-axis is equal to 1 it means the full step was made at the last iteration. For these cases the convergence results shown in Theorem 2.8 could be considered valid. From the graphs above we observe that stepsize at the last iteration was rather $\gamma_k = 1$ or $\gamma_k < 0.05$. We observe that for varying λ algorithm would typically do a small step at the last iteration. It seems that algorithm with varying λ benefits more from line search technique than the algorithm with fixed λ. Possibly, pure Levenberg-Marquardt method with varying λ would not converge for most of the problems. Interestingly, for fixed values of λ stepsize was $\gamma_k < 0.05$ at the last iteration much more often for the values of λ that showed worse performance in terms of recovering solutions (i.e. $\lambda \in \{10^1, 10^2\}$). We also observe that for medium values of λ ($\lambda \in \{10^4, 10^3, 10^2, 10^1, 10^0\}$) full stepsize was made for less than half of the examples. For large values $\lambda \in \{10^5, 10^5\}$ full step was made for 63.71% and 70.16% of the problems respectively. Further on, small values of λ for which more solutions were recovered would do the full step at the last iteration for most of the examples. For instance, with $\lambda = 10^{-3}$ and $\lambda = 10^{-2}$ full step was made at the last iteration for 73.39% of the problems, while for $\lambda = 10^{-1}$ full step was made for 75.81% of the problems. In terms of the fixed λ_{best} it is interesting to observe that full step was used only for 82/124 (66.13%) of the problems, meaning that for a third of the problems line-search was implemented in the last step for the best tested value of λ. This also coincides with the results of Figure 15 where with smaller values of λ algorithm has shown faster than linear convergence for more examples than for big values of λ. This is likely to be the case that small steps were made in the other instances due to the non-efficient direction of the method at the last iteration.

5 FINAL COMMENTS

In this paper, a class of the LLVF-based optimality conditions for bilevel optimization problems has been reformulated as a system of equations using the Fischer-Burmeister function. Smoothed

Levenberg-Marquardt method was proposed to solve the system. We have seen that penalty parameter could affect the method to diverge. That is why we discussed the penalty parameter selection and partial calmness in details. Surprisingly, small values of penalty parameter have shown good properties for our method. This demonstrated that small penalty parameter could possibly be good for our framework, although classic literature on exact penalization suggests selection of the large penalty parameter. Further, we have discussed that implementing the method with line search to control step-size. We have then provided the framework of choosing all required parameters for the method. The results of the numerical experiments were presented, where we demonstrated the results for implementing smoothed Levenberg-Marquardt method for bilevel optimization with the two approaches of choosing λ. The method with fixing λ showed to strongly outperform the method with varying λ, which coincides with the discussion in Section 3. Further, numerical experiments approved that small λ could be a good choice of penalty parameter for our framework, showing even better performance than large values of λ for both approaches. Medium values of λ were the worst for the method to converge to a solution. We further observe that with both approaches the method showed linear experimental order of convergence for most of the examples.

Average CPU time for all fixed λ is 0.243 seconds, average CPU time for fixed λ_{best} is 0.193 seconds, average CPU time for varying λ is 0.525 seconds. Algorithm with varying λ turns out to be more than twice slower than for fixed λ. However, running algorithm for all fixed values of λ would take way more time than just for varying λ.

Part IV.
Paper 3: Levenberg-Marquardt method for linear bilevel optimization

This article examines the application of smoothed Levenberg-Marquardt method to find solutions of linear bilevel programming problems (BLPs). It will be shown that for this class of problems some key assumptions needed to state optimality conditions are satisfied automatically. Two reformulations of bilevel programming problems will be considered, namely Karush-Kuhn-Tucker (KKT) and lower-level value function (LLVF) reformulations. The optimality conditions for both reformulations will be stated for the linear class of bilevel problems. In particular, we will discuss how the linear structure of the problem affects the introduced optimality conditions. We will then compare the basic properties of the reformulations to show the theoretical differences and similarities of the KKT and LLVF formulations for the linear case. With the use of NCP-functions and smoothing technique optimality conditions for both reformulations will be stated as the differentiable systems of equations. We then present Levenberg-Marquardt algorithm for linear bilevel optimization with the required convergence assumptions and parameters specifications for the linear framework. Finally, we compare the two reformulations in the context of implementing Levenberg-Marquardt algorithm for the instances of linear bilevel programming problems. The comparison will be performed for the test set of 24 linear problems from BOLIB [85]. To extend numerical tests, we further present transformation of mixed integer and binary examples from [32] and provide results obtained by our algorithm for these transformed examples.

1 INTRODUCTION

In this paper we are going to focus on the optimistic linear bilevel problems of the form

$$\begin{aligned} \min_{x,y} \quad & F(x,y) := c_1^\top x + d_1^\top y \\ \text{s.t.} \quad & A_1 x + B_1 y - b_1 \leqslant 0, \\ & y \in S(x) := \arg\min_y \left\{ f(x,y) = c_2^\top x + d_2^\top y : \ A_2 x + B_2 y - b_2 \leqslant 0 \right\}, \end{aligned} \tag{1.1}$$

where $c_1 \in \mathbb{R}^{n\times 1}$, $d_1 \in \mathbb{R}^{m\times 1}$, $A_1 \in \mathbb{R}^{q\times n}$, $B_1 \in \mathbb{R}^{q\times m}$, $b_1 \in \mathbb{R}^{q\times 1}$, $c_2 \in \mathbb{R}^{n\times 1}$, $d_2 \in \mathbb{R}^{m\times 1}$, $A_2 \in \mathbb{R}^{p\times n}$, $B_2 \in \mathbb{R}^{p\times m}$, and $b_2 \in \mathbb{R}^{p\times 1}$. As usual, we refer to F (resp. f) as upper-level (resp. lower-level) objective function. Interested reader is referred to [3] for detailed overview of the problem. Problems of the form of (1.1) are known to be NP-hard. In terms of solution methods, there are several ways to deal with linear bilevel programming problems. The methodologies include k^{th} best algorithm, KKT approach, Simplex method etc. The class of exact methods take advantage of specific properties of LB problem. These methods have been classified as *enumerative algorithms*. One of the most well-known algorithms of this class is the *Kth best algorithm*. The methodology is to enumerate extreme points and examine all extreme of the polyhedron and choose the best one with respect to the upper-level objective function. Further on, [77] achieved similar results under weaker assumptions via a penalty function approach. Another category of algorithms deals with transforming LB problem into a single level problem using *Karush-Kuhn-Tucker* (KKT) conditions. For linear optimization problems KKT conditions are known to be necessary and sufficient for optimality. The approach substitutes lower-level problem by KKT conditions, transforming bilevel problem into single-level problem. This is then typically solved by mixed-integer linear programming or by branch-and-bound algorithms, or penalty approaches. The third subcategory includes algorithms that apply classical optimization theory. One of the first algorithms to solve LB problem was introduced by Bialas and Karwan [9]. They develop the Sequential Linear Complementarity Problems (SLCP) algorithm. Finally, there are some metaheuristic algorithms, such as evolutionary algorithms, but we are not going to touch heuristics in details in this paper. The discussed methods are known to have some disadvantages. As it was mentioned in [60] the k^{th} best method is computationally costly in comparison to KKT conditions, especially for large problems. The KKT approach does not always guarantee an optimal solutions. Further, with the certain structure one can only guarantee local optimality of the solution by the methods based on this reformulation. Nevertheless, this is the most popular approach to solve linear bilevel programming problems. For this reason, we aim to compare KKT approach with *lower level value function* (LLVF) reformulation for the class of linear bilevel programming problems.

To solve (1.1) from the perspective of standard constrained optimization, we are going to look at two approaches to reformulate the problem as a single-level optimization problem. We are firstly going to consider *Karush-Kuhn-Tucker* (KKT) reformulation, where lower-level problem is replaced with its KKT optimality conditions. The second reformulation is to formulate the problem as exact penalization problem with the lower-level problem being replaced by the value function penalty term as done in [Paper 1, Paper 2]. This will be referred to as *lower-level value function* (LLVF) reformulation. In this paper we will compare theoretical and numerical properties of KKT and LLVF reformulations for the linear bilevel problems. Both reformulations take advantage of the linearity of the problem as some main assumptions needed to state optimality conditions for both KKT and LLVF reformulations hold automatically in the context of linear framework. We are going to show what are the theoretical simplifications arising due to the linear structure of (1.1) for the chosen solution methods. According to [84], for a general problem it is not obvious which reformulation is more beneficial in terms of the assumptions needed for the reformulation to be obtained. The KKT reformulation of problem (1.1) can be written as

$$\begin{aligned} \min_{x,y,w} \quad & c_1^\top x + d_1^\top y \\ \text{s.t.} \quad & A_1x + B_1y - b_1 \leqslant 0,\ d_2 + B_2^\top w = 0, \\ & A_2x + B_2y - b_2 \leqslant 0,\ w \geqslant 0,\ w^\top(A_2x + B_2y - b_2) = 0. \end{aligned} \tag{KKT}$$

The second single-level reformulation of (1.1) is lower-level value function reformulation (LLVF)

$$\begin{aligned} \min_{x,y} \quad & c_1^\top x + d_1^\top y \\ \text{s.t.} \quad & A_1x + B_1y - b_1 \leqslant 0, \\ & A_2x + B_2y - b_2 \leqslant 0,\ \ c_2^\top x + d_2^\top y \leqslant \varphi(x), \end{aligned} \tag{LLVF}$$

where the optimal value function is defined by

$$\varphi(x) := \inf \left\{ c_2^\top x + d_2^\top y \mid A_2x + B_2y - b_2 \leqslant 0 \right\}. \tag{1.2}$$

We assume throughout that $S(x) \neq \emptyset$ for all $x \in \mathbb{R}^n$ to ensure that φ is well-defined. We will see that partial calmness is the crucial condition to state optimality conditions for both reformulations. According to [52], problems of the structure defined by (1.1) are partially calm due to the linearity in the lower-level problem. Clearly, it is a big advantage that partial calmness is automatically satisfied for the linear bilevel problems (1.1). However, this approach still leads to the formulations involving penalty parameter, which remains hard to choose. Authors use different methods to optimally choose penalty parameter in the context of their framework, [8, 11, 33, 51, 59, 61]. In [Paper 2] some properties of the parameter were studied for LLVF reformulation and some interesting results were observed in the context of implementing Levenberg-Marquardt method for nonlinear bilevel optimization problems. It is interesting whether the observed behaviour of the penalty parameters will hold for the problems with linear structure considered in this paper.

In the next section we are going to define optimality conditions for KKT and LLVF reformulations and discuss what are the simplifications for the linear class of bilevel problems in comparison to the general case. In terms of the method to solve the optimality conditions based on the KKT and LLVF reformulations, we will implement Levenberg-Marquardt method in Section 2.2. As Levenberg-Marquardt method is essentially the combination of Gauss-Newton method and gradient descent method, the link to these methods will be discussed. We will note that some elements of the Jacobian of the system of optimality conditions will vanish due to the linear structure, which creates a small concern about Levenberg-Marquardt direction that will be discussed. However, we will discuss that due to the other elements involved to compute direction this does not create a big danger in terms of implementing the method for the linear bilevel problems. We will further show under what conditions Levenberg-Marquardt method converges for both reformulations of a linear bilevel problem. For this we will mainly be touching possible scenarios for which *error bound condition* holds, which is the main property required to establish convergence of the method. This will show the method is theoretically appropriate to solve problems of the type (1.1). We are also going to show that framework of the method is relatively robust, as assumptions required for the convergence of the method are easier to satisfy for the linear framework than for the general case.In Section 3 we are going to present the implementation results of the smoothed Levenberg-Marquardt method for both KKT and LLVF reformulations on two different test sets of linear bilevel problems. Firstly, we are going to compare performance of the methods for 24 small linear bilevel problems from BOLIB [85], where solutions are known. We are then going to define transformations of 50 integer and 124 binary problems from [32] to the general BLPs. As these transformations can be considered to be new problems, we simply present the obtained results, hoping to create basis for the comparison for other authors.

2 LEVENBERG-MARQUARDT METHOD FOR LINEAR BILEVEL OPTIMIZATION

2.1 Optimality conditions and equation reformulation

It is well-known that because of the complementarity conditions in (KKT) and the value function constraint $c_2^\top x + d_2^\top y \leqslant \varphi(x)$ in (LLVF), standard constraint qualifications cannot hold for the corresponding problems. Hence, to deal with this issue, we start here by building corresponding penalization models that are more tractable optimization problems.

Theorem 2.1. *Assuming that $B_1 = 0$, the following statements hold true:*

(i) *If the point $(\bar{x}, \bar{y})$ is a local optimal solution of problem (LLVF), then there exists some $\bar{\lambda} > 0$ such that this point is also locally optimal for the following problem for all $\lambda > \bar{\lambda}$:*

$$\begin{aligned} \min_{x,y} \quad & c_1^\top x + d_1 y + \lambda(c_2^\top x + d_2^\top y - \varphi(x)) \\ \text{s.t.} \quad & A_1 x + B_1 y - b_1 \leqslant 0, \;\; A_2 x + B_2 y - b_2 \leqslant 0. \end{aligned} \tag{λ-LLVF}$$

Any local optimal solution of (λ-LLVF) with $\lambda > \bar{\lambda}$ with respect to the neighborhood of $(\bar{x}, \bar{y})$ in which $(\bar{x}, \bar{y})$ is a local minimum are also local minima of (LLVF).

(ii) *If the point $(\bar{x}, \bar{y}, \bar{w})$ is a local optimal solution of problem (KKT), then there exists some $\bar{\lambda} > 0$ such that this point is also locally optimal for the following problem for all $\lambda > \bar{\lambda}$:*

$$\begin{aligned} \min_{x,y} \quad & c_1^\top x + d_1^\top y - \lambda w^\top (A_2 x + B_2 y - b_2) \\ \text{s.t.} \quad & A_1 x + B_1 y - b_1 \leqslant 0, \; A_2 x + B_2 y - b_2 \leqslant 0, \; d_2 + B_2^\top w = 0, \; w \geqslant 0. \end{aligned} \tag{λ-KKT}$$

Any local optimal solution of (λ-KKT) with $\lambda > \bar{\lambda}$ with respect to the neighbourhood of $(\bar{x}, \bar{y}, \bar{w})$ in which $(\bar{x}, \bar{y}, \bar{w})$ is a local minimum are also local minima of (KKT).

Proof. For (i), let $(\bar{x}, \bar{y})$ be a local optimal solution of (1.2) but not for (λ-LLVF) for all $\lambda > 0$. Then, there is a sequence (x^k, y^k) with

$$A_1 x^k \leqslant b_1, \;\; A_2 x^k + B_2 y^k \leqslant b_2, \;\; \text{and} \;\; \left\|(x^k, y^k) - (\bar{x}, \bar{y})\right\| < \frac{1}{k}$$

such that we have the following sequence of inequalities:

$$0 < c_2^\top x^k + d_2^\top y^k - \varphi(x^k) < \frac{1}{k}\left[c_1^\top \left(\bar{x} - x^k\right) + d_1^\top \left(\bar{y} - y^k\right)\right]. \tag{2.1}$$

Hence, $\lim\limits_{k\to\infty}\left[c_2^\top x^k + d_2^\top y^k - \varphi(x^k)\right] = 0$ considering the fact that $\left\|(x^k, y^k) - (\bar{x}, \bar{y})\right\| < \frac{1}{k}$. However, the second inequality in (2.1) contradicts the existence of $\delta > 0$ and $\kappa > 0$ such that

$$c_1^\top(\bar{x} - x) + d_1^\top(\bar{y} - y) \leqslant \kappa|u| \tag{2.2}$$

for all $(x, y, u) \in \mathbb{B}_\delta(\bar{x}, \bar{y}, 0)$ satisfying

$$A_1 x \leqslant b_1,\;\; c_2^\top x + d_2^\top y - \varphi(x) \leqslant u,\;\; A_2 x + B_2 y - b_2 \leqslant 0, \tag{2.3}$$

which is guarantied under the assumption that $B_1 = 0$ based on the partial calmness concept [52]. The remaining part of the proof of item (i) is obvious. As for item (ii), note that it follows in a similar, while noting that for any vector $(x, y, w, u) \in \mathbb{B}_\delta(\bar{x}, \bar{y}, \bar{w}, 0)$, for some $\delta > 0$, such that

$$A_1 x \leqslant b_1,\;\; -w^\top(A_2 x + B_2 y - b_2) \leqslant u,\;\; A_2 x + B_2 y - b_2 \leqslant 0,\;\; d_2 + B_2^\top w = 0,\;\; w \geqslant 0,$$

condition (2.3) is satisfied. Hence, ensuring that (2.2) also holds in the context of problem (KKT). □

Although we assumed $B_1 = 0$ in Theorem 2.3, we believe that having B_1 present in the problem does not break mathematical structure of the reformulations. It has been verified in [80] that introducing optimality conditions of this nature with upper-level constraints depending on lower-level variable is a mathematically valid approach. The assumption that upper-level constraints do not depend on lower-level variables is common in bilevel optimization due to the nature of the problem being solved. Hence, most sources construct the proof for the case when $B_1 = 0$. However, our initial formulation (1.1) considers more general case with B_1 being present in the problem and we aim to construct the analysis that allows the possibility for the scenario when upper-level constraints could depend on y. Next, we derive necessary optimality conditions for (KKT) and (LLVF) based on (λ-KKT) and (λ-LLVF), respectively. To proceed, we need the following assumption:

Assumption 2.2. The matrix $B_2 \in \mathbb{R}^{p\times m}$ in (1.1) is full rank.

Theorem 2.3. *The following statements hold true:*

(i) *If (x, y, w) is a local optimal solution of problem (λ-KKT) for some $\lambda > 0$, then there exist vectors $(u, \eta, w) \in \mathbb{R}^{3p}$, $v \in \mathbb{R}^q$, and $s \in \mathbb{R}^m$ such that*

$$c_1 + A_2^\top(u - \lambda w) + A_1^\top v = 0, \tag{2.4}$$
$$d_1 + B_2^\top(u - \lambda w) + B_1^\top v = 0, \tag{2.5}$$
$$d_2 + B_2^\top w = 0, \tag{2.6}$$
$$-\lambda(A_2 x + B_2 y - b_2) + B_2 s + \eta = 0, \tag{2.7}$$
$$u \geqslant 0, (A_2 x + B_2 y - b_2) \leqslant 0, u^\top(A_2 x + B_2 y - b_2) = 0, \tag{2.8}$$
$$v \geqslant 0, (A_1 x + B_1 y - b_1) \leqslant 0, v^\top(A_1 x + B_1 y - b_1) = 0, \tag{2.9}$$
$$\eta \leqslant 0, w \geqslant 0, \eta^\top w = 0. \tag{2.10}$$

(ii) *If (x, y, w) is a local optimal solution of problem (λ-LLVF) for some $\lambda > 0$, and Assumption 2.2 holds. Then there exist Lagrange multipliers $(u, w) \in \mathbb{R}^{2p}$ and $v \in \mathbb{R}^q$ such that*

$$c_1 + A_2^\top(u - \lambda w) + A_1^\top v = 0, \tag{2.11}$$
$$d_1 + B_2^\top(u - \lambda w) + B_1^\top v = 0, \tag{2.12}$$
$$d_2 + B_2^\top w = 0, \tag{2.13}$$
$$u \geqslant 0,\;\; A_2 x + B_2 y - b_2 \leqslant 0,\;\; u^\top(A_2 x + B_2 y - b_2) = 0, \tag{2.14}$$
$$v \geqslant 0,\;\; A_1 x + B_1 y - b_1 \leqslant 0,\;\; v^\top(A_1 x + B_1 y - b_1) = 0, \tag{2.15}$$
$$w \geqslant 0,\;\; A_2 x + B_2 y - b_2 \leqslant 0,\;\; w^\top(A_2 x + B_2 y - b_2) = 0. \tag{2.16}$$

Proof. The optimality conditions in (i) follow easily from the standard well-known Lagrange multipliers rule, without any constraint qualification, as the feasible set of problem (λ-KKT) is defined only by linear constraints. In the case of (ii), we might just want to observe that the optimal value function φ (1.2) is convex, as $S(x) \neq \emptyset$ for all $x \in \mathbb{R}^n$, and hence locally Lipschitz continuous. Therefore, the Lagrange multipliers rule for Lipschitz continuous optimization for (λ-LLVF) also holds, but while noting that the subdifferential of φ, in the sense of convex analysis, can be estimated as

$$\partial\varphi(x) \subseteq \left\{c_2 + A_2^\top w \,\middle|\, d_2 + B_2^\top w = 0,\ w \geqslant 0,\ A_2x + B_2y \leqslant b_2,\ w^\top (A_2x + B_2y - b_2) = 0\right\},$$

thanks to the fulfilment of Assumption 2.2. □

First observe that the optimality conditions in (i) have more variables and constraints than the ones in (ii) and more importantly, as we will see later in this section the system (2.4)–(2.10) will lead to a square system of equations, while (2.11)–(2.16) will generate an overdetermined one. However, as we show below, there is a close connection between the two systems.

Proposition 2.4. *If the point (x, y, u, v, w, η, s) satisfies (2.4)–(2.10) with either*

$$w^\top B_2 s \geqslant 0 \quad \text{or} \quad w^\top (A_2x + B_2y - b_2) \geqslant 0, \tag{2.17}$$

then, the point (x, y, u, v, w) fulfils conditions (2.11)–(2.16).

Conversely, if the point (x, y, u, v, w) satisfies (2.11)–(2.16) and there exists some s such that

$$w^\top B_2 s \leqslant 0 \quad \text{and} \quad \lambda(A_2x + B_2y - b_2) \leqslant B_2 s, \tag{2.18}$$

then, we can find some η such that the point (x, y, u, v, w, η, s) fulfills conditions (2.4)–(2.10).

Proof. For a point (x, y, u, v, w, η, s) satisfying (2.4)–(2.10), we have

$$\eta = \lambda(A_2x + B_2y - b_2) - B_2 s. \tag{2.19}$$

Hence, combining this with (2.17) and (2.10), we have

$$0 \leqslant \frac{1}{\lambda} w^\top B_2 s = \frac{1}{\lambda}\left(w^\top \eta + w^\top B_2 s\right) = w^\top (A_2x + B_2y - b_2) \leqslant 0,$$

thus ensuring that the complementarity system (2.16) is satisfied.

Conversely, consider a point (x, y, u, v, w) satisfying (2.11)–(2.16) and some s such that (2.18) holds. Then defining η as in (2.19), it follows that

$$0 \leqslant -w^\top B_2 s = w^\top [\lambda(A_2x + B_2y - b_2) - B_2 s] = w^\top \eta \leqslant 0$$

while considering the fact that $w^\top (A_2x + B_2y - b_2) = 0$ from (2.16). Combining this with the aforementioned definition of η, we clearly have (2.7) and (2.10). Thus, concluding the proof. □

Conditions (2.4)-(2.10) and (2.12)-(2.16) involve presence of complementarity conditions (2.8)-(2.10) and (2.14)-(2.16) respectively. In order to reformulate the complementarity conditions in the form of a system of equations, we are going make use of NCP-functions; see, e.g., [68]. The function $\phi : \mathbb{R}^2 \to \mathbb{R}$ is said to be a NCP-function if we have

$$\phi(a, b) = 0 \iff a \geqslant 0,\ b \geqslant 0,\ ab = 0.$$

In this paper, we use $\phi(a, b) := \sqrt{a^2 + b^2} - a - b$, known as the *Fischer-Burmeister* function [29]. This leads to the reformulation of the optimality conditions (2.4)–(2.10) into the system of equations:

$$\Upsilon^\lambda_{KKT}(z) := \begin{pmatrix} c_1 + A_2^\top(u - \lambda w) + A_1^\top v \\ d_1 + B_2^\top(u - \lambda w) + B_1^\top v \\ d_2 + B_2^\top w \\ -\lambda(A_2x + B_2y - b_2) + B_2 s + \eta \\ \sqrt{u^2 + (A_2x + B_2y - b_2)^2} - u + A_2x + B_2y - b_2 \\ \sqrt{v^2 + (A_1x + B_1y - b_1)^2} - v + A_1x + B_1y - b_1 \\ \sqrt{\eta^2 + w^2} - w + \eta \end{pmatrix} = 0. \tag{2.20}$$

With the use of NCP functions, optimality conditions (2.12)–(2.16) result into the following system.

$$\Upsilon^{\lambda}_{LLVF}(z) := \begin{pmatrix} c_1 + A_2^\top(u-\lambda w) + A_1^\top v \\ d_1 + B_2^\top(u-\lambda w) + B_1^\top v \\ d_2 + B_2^\top w \\ \sqrt{u^2 + (A_2x+B_2y-b_2)^2} - u + A_2x + B_2y - b_2 \\ \sqrt{v^2 + (A_1x+B_1y-b_1)^2} - v + A_1x + B_1y - b_1 \\ \sqrt{w^2 + (A_2x+B_2y-b_2)^2} - w + A_2x + B_2y - b_2 \end{pmatrix} = 0. \tag{2.21}$$

It is possible to transform (2.20) by substituting η in the last line of (2.20) by the expression $\eta = \lambda(A_2x + B_2y - b_2) - B_2s$ from (2.7). This would make systems (2.20) and (2.21) look very similar and easy to see how Proposition 2.4 implies the equivalence of the reformulations. However, it is beneficial to keep (2.20) as it is, due to the system being square this way. To solve the systems of optimality conditions (2.20) and (2.21) we are going to implement Levenberg-Marquardt method with line search. With the method of this nature, instead of directly solving (2.20) and (2.21), respectively, we are dealing with the minimization of the following least-squares problem,

$$\Phi^{\lambda}_{KKT}(z) = \frac{1}{2}\left\|\Upsilon^{\lambda,\mu}_{KKT}(z)\right\|^2 \quad \text{and} \quad \Phi^{\lambda}_{LLVF}(z) = \frac{1}{2}\left\|\Upsilon^{\lambda,\mu}_{LLVF}(z)\right\|^2. \tag{2.22}$$

The Levenberg-Marquardt method requires Υ^{λ}_{KKT} and $\Upsilon^{\lambda}_{LLVF}$ to be differentiable. The only problem with the differentiability occurs for the NCP-functions whenever Lagrange multiplier and the corresponding constraint are $(0,0)$ for some component. As discussed in [Paper 1] there are two main approaches to deal with the differentiability: *strict complementarity assumption* and *smoothing technique*. As was discussed in [Paper 1] the first approach has a big practical disadvantage as strict complementarity is too strong assumption to often hold in practice. Smoothing technique works well theoretically and practically to deal with the differentiability of the NCP functions. For this reason we are going to proceed with the smoothing technique. For the smoothing technique we will add the perturbation 2μ, where $\mu \downarrow 0$, under the square root of Fischer-Burmeister function to ensure differentiability of the systems (2.20) and (2.21) as follows. With the smoothing, the KKT system (2.20) becomes

$$\Upsilon^{\lambda,\mu}_{KKT}(z) := \begin{pmatrix} c_1 + A_2^\top(u-\lambda w) + A_1^\top v \\ d_1 + B_2^\top(u-\lambda w) + B_1^\top v \\ d_2 + B_2^\top w \\ -\lambda(A_2x+B_2y-b_2) + B_2s + \eta \\ \sqrt{u^2 + (A_2x+B_2y-b_2)^2 + 2\mu} - u + A_2x + B_2y - b_2 \\ \sqrt{v^2 + (A_1x+B_1y-b_1)^2 + 2\mu} - v + (A_1x + B_1y - b_1) \\ \sqrt{\eta^2 + w^2 + 2\mu} - \eta + w \end{pmatrix}. \tag{2.23}$$

Applying smoothing to LLVF system, (2.21) takes the form

$$\Upsilon^{\lambda,\mu}_{LLVF}(z) := \begin{pmatrix} c_1 + A_2^\top(u-\lambda w) + A_1^\top v \\ d_1 + B_2^\top(u-\lambda w) + B_1^\top v \\ d_2 + B_2^\top w \\ \sqrt{u^2 + (A_2x+B_2y-b_2)^2 + 2\mu} - u + A_2x + B_2y - b_2 \\ \sqrt{v^2 + (A_1x+B_1y-b_1)^2 + 2\mu} - v + A_1x + B_1y - b_1 \\ \sqrt{w^2 + (A_2x+B_2y-b_2)^2 + 2\mu} - w + A_2x + B_2y - b_2 \end{pmatrix}. \tag{2.24}$$

We can easily check that

$$\|\Upsilon^{\lambda,\mu}_{KKT}(z) - \Upsilon^{\lambda}_{KKT}(z)\| \longrightarrow 0 \quad \text{and} \quad \|\Upsilon^{\lambda,\mu}_{LLVF}(z) - \Upsilon^{\lambda}_{LLVF}(z)\| \longrightarrow 0 \quad \text{as} \quad \mu \downarrow 0.$$

Following the principle of the introduced smoothing scheme (see e.g. [70]), our aim is to consider a sequence $\{\mu_k\}$ decreasing to 0 such that equations (2.20) and (2.21) are approximately solved:

$$\Upsilon^{\lambda,\mu^k}_{KKT}(z) = 0 \text{ and } \Upsilon^{\lambda,\mu^k}_{KKT}(z) = 0, \quad k = 0, 1, \ldots \tag{2.25}$$

for a fixed value of $\lambda > 0$. The implementation of the smoothing technique will be discussed later in the context of implementing Levenberg-Marquardt method for bilevel optmization. In the framework with smoothing, the Jacobians of $\Upsilon^{\lambda,\mu}_{KKT}$ and $\Upsilon^{\lambda,\mu}_{LLVF}$ are well-defined everywhere. Hence, for $\lambda > 0$, $\mu > 0$ and $z := (x, y, u, v, w, s, \eta)$ the Jacobian $\nabla\Upsilon^{\lambda,\mu}_{KKT}$ can be expressed as

$$\nabla\Upsilon^{\lambda,\mu^k}_{KKT}(z) = \begin{bmatrix} O & O & A_2^\top & A_1^\top & -\lambda A_2^\top & O & O \\ O & O & B_2^\top & B_1^\top & -\lambda B_2^\top & O & O \\ O & O & O & O & B_2^\top & O & O \\ -\lambda A_2 & -\lambda B_2 & O & O & O & B_2 & I \\ \mathcal{T}^\mu A_2 & \mathcal{T}^\mu B_2 & \Gamma^\mu & O & O & O & O \\ \mathcal{A}^\mu A_1 & \mathcal{A}^\mu B_1 & O & \mathcal{B}^\mu & O & O & O \\ O & O & O & O & \Psi^\mu & O & \Omega^\mu \end{bmatrix} \tag{2.26}$$

with $\mathcal{T}^\mu := \operatorname{diag}\{\tau^\mu_1, \dots, \tau^\mu_p\}$ and $\Gamma^\mu := \operatorname{diag}\{\gamma^\mu_1, \dots, \gamma^\mu_p\}$ representing a pair of diagonal matrices, where the pair $(\tau^\mu_j, \gamma^\mu_j)$, $j := 1, \dots p$ is given by

$$\tau^\mu_j := \frac{A_{2_j}x + B_{2_j}y - b_{2_j}}{\sqrt{u_j^2 + (A_{2_j}x + B_{2_j}y - b_{2_j})^2 + 2\mu}} + 1, \;\; j = 1, \dots p \tag{2.27}$$

and

$$\gamma^\mu_j := \frac{u_j}{\sqrt{u_j^2 + (A_{2_j}x + B_{2_j}y - b_{2_j})^2 + 2\mu}} - 1, \;\; j = 1, \dots p, \tag{2.28}$$

respectively. The other pairs are similarly defined by

$$\left.\begin{array}{l} \mathcal{A}^\mu := \operatorname{diag}\{\alpha^\mu_1, \dots, \alpha^\mu_q\} \\ \mathcal{B}^\mu := \operatorname{diag}\{\beta^\mu_1, \dots, \beta^\mu_q\} \end{array}\right\} \text{ and } \left\{\begin{array}{l} \Psi^\mu := \operatorname{diag}\{\psi^\mu_1, \dots, \psi^\mu_p\} \\ \Omega^\mu := \operatorname{diag}\{\omega^\mu_1, \dots, \omega^\mu_p\} \end{array}\right. \tag{2.29}$$

with the pairs $(\alpha^\mu_j, \beta^\mu_j)$, $j = 1, \dots, q$ and $(\psi^\mu_j, \omega^\mu_j)$, $j = 1, \dots, p$ are defined similarly to (2.27)-(2.28) in terms of $(A_{1_j}x + B_{1_j}y - b_{1_j}, v_j)$, $j = 1, \dots, q$ and in terms of (η_j, w_j), $j = 1, \dots, p$, respectively.

For $\lambda > 0$, $\mu > 0$ and the variables $z := (x, y, u, v, w)$, the Jacobian of $\Upsilon^\lambda_\mu(z)$ can be written as

$$\nabla\Upsilon^{\lambda,\mu^k}_{LLVF}(z) = \begin{bmatrix} O & O & A_2^\top & A_1^\top & -\lambda A_2^\top \\ O & O & B_2^\top & B_1^\top & -\lambda B_2^\top \\ O & O & O & O & B_2^\top \\ \mathcal{T}^\mu A_2 & \mathcal{T}^\mu B_2 & \Gamma^\mu & O & O \\ \mathcal{A}^\mu A_1 & \mathcal{A}^\mu B_1 & O & \mathcal{B}^\mu & O \\ \Theta^\mu A_2 & \Theta^\mu B_2 & O & O & \mathcal{K}^\mu \end{bmatrix} \tag{2.30}$$

with $\mathcal{T}^\mu := \operatorname{diag}\{\tau^\mu_1, \dots, \tau^\mu_p\}$, $\Gamma^\mu := \operatorname{diag}\{\gamma^\mu_1, \dots, \gamma^\mu_p\}$, $\mathcal{A}^\mu := \operatorname{diag}\{\alpha^\mu_1, \dots, \alpha^\mu_q\}$, $\mathcal{B}^\mu := \operatorname{diag}\{\beta^\mu_1, \dots, \beta^\mu_q\}$, $\Theta^\mu := \operatorname{diag}\{\theta^\mu_1, \dots, \theta^\mu_p\}$, and $\mathcal{K}^\mu := \operatorname{diag}\{\kappa^\mu_1, \dots, \kappa^\mu_p\}$, where the pair $(\tau^\mu_j, \gamma^\mu_j)$, $j := 1, \dots p$ is defined by (2.27)-(2.28). The pairs $(\alpha^\mu_j, \beta^\mu_j)$, $j = 1, \dots, q$ and $(\theta^\mu_j, \kappa^\mu_j)$, $j = 1, \dots, p$ are defined similarly in terms of $(A_{1_j}x + B_{1_j}y - b_{1_j}, v_j)$, $j = 1, \dots, q$ and $(A_{2_j}x + B_{2_j}y - b_{2_j}, w_j)$, $j = 1, \dots, p$, respectively. The following lemma from [Paper 1] clearly holds for the linear case and will be useful in upcoming convergence analysis.

Lemma 2.5 ([Paper 1]). *For a point $z := (x, y, u, v, w)$ such that $\Upsilon^{\lambda,\mu}_{LLVF}(z) = 0$ with $\lambda > 0$ and $\mu > 0$, it holds that*

$$\begin{array}{lll} \tau^\mu_j > 0, & \gamma^\mu_j < 0, & j = 1, \dots, p, \\ \alpha^\mu_j > 0, & \beta^\mu_j < 0, & j = 1, \dots, q, \\ \theta^\mu_j > 0, & \kappa^\mu_j < 0, & j = 1, \dots, p. \end{array}$$

Further, for a point $z := (x, y, u, v, w, s, \eta)$ such that $\Upsilon^{\lambda,\mu}_{KKT} = 0$ with $\lambda > 0$ and $\mu > 0$ the signs of $\tau^\mu_j, \gamma^\mu_j, \alpha^\mu_j, \beta^\mu_j$ above hold true and the following is satisfied

$$\psi^\mu_j > 0, \;\; \omega^\mu_j < 0, \;\; j = 1, \dots, p.$$

2.2 The algorithm and convergence analysis

We present an algorithm to compute the stationary points for problems (λ-KKT) and (λ-LLVF); i.e., precisely to solve the approximated systems (2.23) and (2.24), respectively. To make sure that the algorithm is suited for both systems, we use Υ^{λ} to represent Υ^{λ}_{KKT} (2.20) or $\Upsilon^{\lambda}_{LLVF}$ (2.21). Similarly, $\Upsilon^{\lambda,\mu}$ correspond to $\Upsilon^{\lambda,\mu}_{KKT}$ (2.23) or $\Upsilon^{\lambda,\mu^k}_{LLVF}$ (2.24).

Algorithm 2.6 (General method with line search)**.**

Step 0: *Choose* $\lambda > 0$, $\mu > 0$, $\epsilon > 0$, $K > 0$, $(\gamma, \rho, \sigma) \in (0,1)^3$, $\alpha_0 > 0$, z^0, *and set* $k := 0$.
Step 1: *If* $\|\Upsilon^{\lambda}(z^k)\| < \epsilon$ *or* $k \geqslant K$, *then stop.*
Step 2: *Calculate Jacobian* $\nabla\Upsilon_{\lambda,\mu}(z^k)$ *and find* d^k *such that*

$$\left(\nabla\Upsilon^{\lambda,\mu}(z^k)^{\top}\nabla\Upsilon^{\lambda,\mu}(z^k) + \alpha_k I\right) d^k = -\nabla\Upsilon^{\lambda,\mu}(z^k)^{\top}\Upsilon^{\lambda,\mu}(z^k), \tag{2.31}$$

where I *denotes the identity matrix of appropriate size.*
Step 3: ***While*** $\|\Upsilon^{\lambda,\mu}(z^k + \gamma_k d^k)\|^2 \geqslant \|\Upsilon^{\lambda,\mu}(z^k)\|^2 + \sigma\gamma_k \nabla\Upsilon^{\lambda,\mu}(z^k)^T\Upsilon^{\lambda,\mu}(z^k) d^k$, ***do*** $\gamma_k \leftarrow \rho\gamma_k$ ***end***.

Step 4: *Set* $z^{k+1} := z^k + \gamma_k d^k$, $k := k+1$, $\mu_{k+1} = \mu_k^{k+1}$ *and go to Step 1.*

Obviously, this algorithm is an adaptation of the well-known Levenberg–Marquardt method. We are going to choose a fixed number $\lambda \in \{10^5, 10^4, ..., 10^{-2}\}$ as the first approach and varying λ as increasing sequence $\lambda = 0.5 \times 1.05^k$ as another approach. It is worth noting that Levenberg-Marquardt method is essentially a combination of Gauss-Newton method and gradient descent method with the weight given to one or another depending on Levenberg-Marquardt parameter α. Whenever $\alpha^k \to 0$ the direction d^k tends to Gauss-Newton direction, while if $\alpha(z^k) \to \infty$ the direction taken is essentially gradient descent direction. Gradient descent method is known to converge well for linear problems but loses to Newton-like methods for quadratic and higher order problems. As we consider all original functions to be linear and add nonlinear NCP-functions to the system, it makes a lot of sense to implement the method that is combination of the method that deals well with linear problems and the higher order method. There is slight concern that some terms in the Jacobians (2.26) and (2.30) vanished due to the linearity of the problem (1.1). In particular, second derivatives of Lagrangians L^{λ} and $\mathcal{L}^{\lambda}$ vansish, which leads to (2.26) and (2.30) not to depend on the objective functions $F(x,y)$ and $f(x,y)$ at all. However, the direction d^k of the Levenberg-Marquardt method defined by (2.31) involves the actual systems Υ^{λ}_{KKT} and Υ^{λ}. Obviously, these systems do depend on $F(x,y)$ and $f(x,y)$ of (1.1) and hence Levenberg-Marquardt method does not ignore objective functions. We are going to show that the method can convergence for both KKT and LLVF frameworks, and compare their numerical performance in the next section.

Line search implemented in Step 3 of the algorithm above ensures that Levenberg-Marquardt algorithm converges globally with the choices $\alpha(z^k) := \left\|\Upsilon^{\lambda_1}_{KKT}(z^k)\right\|$ and $\alpha(z^k) := \|\Upsilon^{\lambda}(z^k)\|$, according to [27]. Before stating the convergence results, let us quickly discuss some of the initialization details of the algorithm. Step 0 is crucial for the convergence and performance of the algorithm. There, K represents the maximum number of iteration that we set to $K := 200$. As for γ, ρ, and σ, they correspond to the parameters of the line search techniqueWe use the values $\gamma_0 := 1$, $\rho = 0.5$, and $\sigma = 10^{-2}$. The selection of the remaining parameters of the algorithm is going to be carefully addressed in Section 3. This will ensure that the following convergence results holds. The following assumption is needed to proceed.

Assumption 2.7. $q + p \geqslant n + m$ and the matrix $\begin{bmatrix} A_1 & B_1 \\ A_2 & B_2 \end{bmatrix}$ is full rank.

Theorem 2.8. *Consider Algorithm 2.6 with fixed values for the parameters* $\lambda > 0$ *and* $\mu > 0$ *and let* $\alpha_k = \|\Upsilon^{\lambda,\mu}_{LLVF}(z^k)\|^{\ell}$ *for any choice of* $\ell \in [1, 2]$. *Then, the sequence* $\{z^k\}$ *converges quadratically to a point* $\bar{z}$ *such that* $\Upsilon^{\lambda,\mu}_{LLVF}(\bar{z}) = 0$ *if one of the following set of conditions is satisfied:*

(i) *Assumption 2.7 holds and rank* $(B_2^{\top}) = p \leqslant m$.

(ii) *Assumption 2.7 holds and* $0 < \lambda < \frac{\tau_j^\mu}{\gamma_j^\mu}\frac{\kappa_j^\mu}{\theta_j^\mu}$.

Proof. Considering the fact that $\Upsilon^{\lambda,\mu}_{LLVF}$ (2.23) is twice continuously differentiable, according to [27], it suffices to show that the *error bound condition* holds under the assumptions made in the theorem. The condition is known to be fulfilled if the columns of the Jacobian matrix $\nabla\Upsilon^{\lambda,\mu}_{LLVF}(\bar{z})$ are linearly independent [78]. To proceed, consider an arbitrary vector $d := (d_1^\top, d_2^\top, d_3^\top, d_4^\top, d_5^\top)^\top$ such that $\nabla\Upsilon^{\lambda,\mu}_{LLVF}(\bar{z})d = 0$ with $d_1 \in \mathbb{R}^n$, $d_2 \in \mathbb{R}^m$, $d_4 \in \mathbb{R}^q$, and $(d_3, d_5) \in \mathbb{R}^{2p}$,

$$A_2^\top d_3 + A_1^\top d_4 - \lambda A_2^\top d_5 = 0, \tag{2.32}$$

$$B_2^\top d_3 + B_1^\top d_4 = 0, \tag{2.33}$$

$$\mathcal{T}^\mu A_2 d_1 + \mathcal{T}^\mu B_2 d_2 + \Gamma^\mu d_3 = 0, \tag{2.34}$$

$$\mathcal{A}^\mu A_1 d_1 + \mathcal{A}^\mu B_1 d_2 + \mathcal{B}^\mu d_4 = 0, \tag{2.35}$$

$$\Theta^\mu A_2 d_1 + \Theta^\mu B_2 d_2 + \mathcal{K}^\mu d_5 = 0, \tag{2.36}$$

$$B_2^\top d_5 = 0. \tag{2.37}$$

Under (i), (2.37) implies that $d_5 = 0$. Hence, the system (2.32)–(2.37) reduces to

$$A_2^\top d_3 + A_1^\top d_4 = 0, \tag{2.38}$$

$$B_2^\top d_3 + B_1^\top d_4 = 0, \tag{2.39}$$

$$\mathcal{T}^\mu A_2 d_1 + \mathcal{T}^\mu B_2 d_2 + \Gamma^\mu d_3 = 0, \tag{2.40}$$

$$\mathcal{A}^\mu A_1 d_1 + \mathcal{A}^\mu B_1 d_2 + \mathcal{B}^\mu d_4 = 0, \tag{2.41}$$

$$\Theta^\mu A_2 d_1 + \Theta^\mu B_2 d_2 = 0. \tag{2.42}$$

Multiplying (2.38) by $d_1^\top$ and (2.39) by $d_2^\top$ and summing them up, we get

$$d_1^\top A_2^\top d_3 + d_1^\top A_1^\top d_4 + d_2^\top B_2^\top d_3 + d_2^\top B_1^\top d_4 = 0. \tag{2.43}$$

Furthermore multiplying (2.40) by $d_3^\top$ and (2.41) by $d_4^\top$, we get

$$d_3^\top A_2 d_1 + d_3^\top B_2 d_2 = \sum_{j=1}^{p}\Big(-\frac{\gamma_j^\mu}{\tau_j^\mu}\Big)d_{3_j}^2, \quad d_4^\top A_1 d_1 + d_4^\top B_1 d_2 = \sum_{j=1}^{q}\left(-\frac{\beta_j^\mu}{\alpha_j^\mu}\right)d_{4_j}^2. \tag{2.44}$$

Substituting (2.44) into (2.43) leads to

$$\sum_{j=1}^{p}\left(-\frac{\gamma_j^\mu}{\tau_j^\mu}\right)d_{3_j}^2 + \sum_{j=1}^{q}\left(-\frac{\beta_j^\mu}{\alpha_j^\mu}\right)d_{4_j}^2 = 0, \tag{2.45}$$

where $-\frac{\gamma_j^\mu}{\tau_j^\mu} > 0$ and $-\frac{\beta_j^\mu}{\alpha_j^\mu} > 0$ due to Lemma 2.5 and hence $d_3 = 0$ and $d_4 = 0$. Inserting these values in (2.35) and (2.34), respectively, the resulting system ensures $d_1 = 0$ and $d_2 = 0$ thanks to Assumption 2.7.

In the context of scenario (ii), let us start by multiplying (2.34) and (2.36) by $d_5^\top$. Then, thanks to (2.37), we respectively have the following equations:

$$\mathcal{T}^\mu d_5^\top A_2 d_1 + d_5^\top \Gamma^\mu d_3 = 0, \tag{2.46}$$

$$\Theta^\mu d_5^\top A_2 d_1 + d_5^\top \mathcal{K}^\mu d_5 = 0. \tag{2.47}$$

Multiplying (2.32) by $d_1^\top$ and (2.33) by $d_2^\top$ and adding them up, we get

$$d_1^\top A_2^\top d_3 + d_1^\top A_1^\top d_4 + d_2^\top B_2^\top d_3 + d_2^\top B_1^\top d_4 - \lambda d_1^\top A_2^\top d_5 = 0, \tag{2.48}$$

Substituting $d_5^\top A_2 d_1 = -\sum_{j=1}^{p}\frac{\kappa_j^\mu}{\theta_j^\mu}d_{5_j}^2$ from (2.47) in equation (2.48),

$$d_1^\top A_2^\top d_3 + d_1^\top A_1^\top d_4 + d_2^\top B_2^\top d_3 + d_2^\top B_1^\top d_4 + \lambda\sum_{j=1}^{p}\frac{\kappa_j^\mu}{\theta_j^\mu}d_{5_j}^2 = 0, \tag{2.49}$$

where $\frac{\kappa_j^\mu}{\theta_j^\mu} < 0$. Then multiplying (2.34) by $d_3^\top$ and (2.35) by $d_4^\top$, we get the expressions in (2.44). Replacing these values in (2.49), we have

$$\sum_{j=1}^{p}\left(-\frac{\gamma_j^\mu}{\tau_j^\mu}\right) d_{3_j}^2 + \sum_{j=1}^{q}\left(-\frac{\beta_j^\mu}{\alpha_j^\mu}\right) d_{4_j}^2 + \lambda \sum_{j=1}^{p} \frac{\kappa_j^\mu}{\theta_j^\mu} d_{5_j}^2 = 0, \tag{2.50}$$

Further, due to (2.34) and (2.36) we have the relationship $d_{3_j} = \frac{\tau_j^\mu}{\gamma_j^\mu}\frac{\kappa_j^\mu}{\theta_j^\mu} d_{5_j}$ for $j = 1,\dots,p$, so that equation (2.50) becomes

$$\sum_{j=1}^{p}\left(-\frac{\tau_j^\mu}{\gamma_j^\mu}\right)\left(\frac{\kappa_j^\mu}{\theta_j^\mu}\right)^2 d_{5_j}^2 + \sum_{j=1}^{q}\left(-\frac{\beta_j^\mu}{\alpha_j^\mu}\right) d_{4_j}^2 + \lambda \sum_{j=1}^{p} \frac{\kappa_j^\mu}{\theta_j^\mu} d_{5_j}^2 = 0, \tag{2.51}$$

where $\frac{\tau_j^\mu}{\gamma_j^\mu} < 0$, $\frac{\beta_j^\mu}{\alpha_j^\mu} < 0$, and $\frac{\kappa_j^\mu}{\theta_j^\mu} < 0$ due to Lemma 2.5. Then with $0 < \lambda < \frac{\tau_j^\mu}{\gamma_j^\mu}\frac{\kappa_j^\mu}{\theta_j^\mu}$ we have that

$$\sum_{j=1}^{p}\left[-\frac{\tau_j^\mu}{\gamma_j^\mu}\left(\frac{\kappa_j^\mu}{\theta_j^\mu}\right)^2 + \lambda\frac{\kappa_j^\mu}{\theta_j^\mu}\right] > 0.$$

Then we can deduce from (2.51) that $d_4 = 0$ and $d_5 = 0$. Inserting these values in (2.35) and (2.36), it also follows from Assumption 2.7 that $d_1 = 0$ and $d_2 = 0$. Finally, inserting these latter values in (2.34), we have $d_3 = 0$, considering the nature of Γ^μ being diagonal matrix with non-zero elements on the main diagonal, thanks to Lemma 2.5 once again. □

Theorem 2.9. *Consider Algorithm 2.6 with fixed values for the parameters $\lambda > 0$ and $\mu > 0$ and let $\alpha_k = \|\Upsilon^{\lambda,\mu}_{KKT}(z^k)\|^\ell$ for any choice of $\ell \in [1, 2]$. Then, the sequence $\{z^k\}$ converges quadratically to a point $\bar{z}$ such that $\Upsilon^{\lambda,\mu}_{KKT}(\bar{z}) = 0$ if Assumption 2.7 holds and rank $\left(B_2^\top\right) = p = m$.*

Proof. Clearly, $\Upsilon^{\lambda,\mu}_{KKT}$ (2.23) is twice continuously differentiable. Similarly to the proof of Theorem 2.8, it then suffices to show that the columns of the Jacobian matrix $\nabla\Upsilon^{\lambda,\mu}_{KKT}(\bar{z})$ are linearly independent under the assumptions made. Considering an arbitary vector

$$d := (d_1^\top, d_2^\top, d_3^\top, d_4^\top, d_5^\top, d_6^\top, d_7^\top)^\top \text{ with } d_1 \in \mathbb{R}^n, (d_2, d_6) \in \mathbb{R}^{2m}, (d_3, d_5, d_7) \in \mathbb{R}^{3p}, \text{ and } d_4 \in \mathbb{R}^q.$$

Then, having $\nabla\Upsilon^{\lambda,\mu}_{KKT}(\bar{z})d = 0$ is equivalent to the system of equations

$$A_2^\top d_3 + A_1^\top d_4 - \lambda A_2^\top d_5 = 0, \tag{2.52}$$
$$B_2^\top d_3 + B_1^\top d_4 = 0, \tag{2.53}$$
$$-\lambda A_2 d_1 - \lambda B_2 d_2 + B_2 d_6 + d_7 = 0, \tag{2.54}$$
$$\mathcal{T}^\mu A_2 d_1 + \mathcal{T}^\mu B_2 d_2 + \Gamma^\mu d_3 = 0, \tag{2.55}$$
$$\mathcal{A}^\mu A_1 d_1 + \mathcal{A}^\mu B_1 d_2 + \mathcal{B}^\mu d_4 = 0, \tag{2.56}$$
$$\Psi^\mu d_5 + \Omega^\mu d_7 = 0, \tag{2.57}$$
$$B_2^\top d_5 = 0. \tag{2.58}$$

Under assumption rank $\left(B_2^\top\right) = p \leqslant m$, we have $d_5 = 0$ from (2.58). Subsequently, $\Omega^\mu d_7 = 0$ from (2.57). Since $\omega_j^\mu < 0$ for all $j = 1,\dots,p$ due to Lemma 2.5, we deduce that $d_7 = 0$. Hence, system (2.52)–(2.56) becomes

$$A_2^\top d_3 + A_1^\top d_4 = 0, \tag{2.59}$$
$$B_2^\top d_3 + B_1^\top d_4 = 0, \tag{2.60}$$
$$-\lambda A_2 d_1 - \lambda B_2 d_2 + B_2 d_6 = 0, \tag{2.61}$$
$$\mathcal{T}^\mu A_2 d_1 + \mathcal{T}^\mu B_2 d_2 + \Gamma^\mu d_3 = 0, \tag{2.62}$$
$$\mathcal{A}^\mu A_1 d_1 + \mathcal{A}^\mu B_1 d_2 + \mathcal{B}^\mu d_4 = 0, \tag{2.63}$$

Multiplying (2.62) by d_3^T and (2.63) by d_4^T, we respectively get

$$d_3^T A_2 d_1 + d_3^T B_2 d_2 = -\sum_{j=1}^{p} \frac{\gamma_j^\mu}{\tau_j^\mu} d_{3_j}^2, \tag{2.64}$$

$$d_4^T A_1 d_1 + d_4^T B_1 d_2. = -\sum_{j=1}^{q} \frac{\beta_j^\mu}{\alpha_j^\mu} d_{4_j}^2. \tag{2.65}$$

Multiplying (2.59) and (2.60) by d_1^T, d_2^T respectively and adding the resulting equations we get

$$d_1^T A_2^T d_3 + d_1^T A_1^T d_4 + d_2^T B_2^T d_3 + d_2^T B_1^T d_4 = 0.$$

Using (2.64) and (2.65) this becomes

$$\sum_{j=1}^{p} \Big(-\frac{\gamma_j^\mu}{\tau_j^\mu}\Big) d_{3_j}^2 + \sum_{j=1}^{q} \Big(-\frac{\beta_j^\mu}{\alpha_j^\mu}\Big) d_{4_j}^2 = 0. \tag{2.66}$$

Since $-\frac{\gamma_j^\mu}{\tau_j^\mu} > 0$ for all $j = 1, \dots, p$ and $-\frac{\beta_j^\mu}{\alpha_j^\mu} > 0$ for all $j = 1, \dots, q$, this clearly means that $d_3 = 0$ and $d_4 = 0$ for all components. Hence, we are left with the following equations:

$$-\lambda A_2 d_1 - \lambda B_2 d_2 + B_2 d_6 = 0, \tag{2.67}$$

$$\mathcal{T}^\mu A_2 d_1 + \mathcal{T}^\mu B_2 d_2 = 0, \tag{2.68}$$

$$\mathcal{A}^\mu A_1 d_1 + \mathcal{A}^\mu B_1 d_2 = 0. \tag{2.69}$$

Multiplying (2.68) by λ, we get $\lambda A_2 d_1 = -\lambda B_2 d_2$. Using this for (2.67) yields $B_2 d_6 = 0$, meaning that $d_6 = 0$ due to Assumption of the full rank of B_2. Finally, under Assumption 2.7, it follows from (2.68) or (2.69) that $d_1 = 0$ and $d_2 = 0$. This completes the proof. □

Remark 2.10. Scenario considered in Theorem 2.9 imposes the same conditions on the problem (1.1) as the first scenario in Theorem 2.8. This demonstrates a strong link between KKT and LLVF reformulations for the linear case. Further, if any conditions from Proposition 2.4 are fulfilled and full rank of the Jacobian could be established for one of the systems then it holds for the other one.

3 NUMERICAL STUDY

In this section we are going to analyze the performance of the algorithm in the context of KKT-based and LLVF-based approaches for the instances of the problem (1.1). We split the results between two test sets of linear examples. The first test set contains 24 linear problems from BOLIB [85], for which solutions are known apart from one problem, for which there is no optimal solution. The second test set is constructed by modifying mixed integer problems from Fischetti, [32], msinnl.github.io/bilevel/MIPLIB. This test set contains 174 problems, where we have 50 transformed integer examples and 124 transformed binary examples. These problems are modified by dropping integer constraints and replacing binary integer constraints with the bound constraints on the variables, i.e. $0 \leqslant x_i \leqslant 1$ and $0 \leqslant y_j \leqslant 1$ for all $i = 1, ..., n$ and $j = 1, ..., m$. Since problems have been modified, we do not know the optimal solutions to these problems. A good number of solutions are expected to be reasonably close to optimal as the considered framework benefits a lot from the structure of the problems. Mainly, partial calmness, lower-level convexity and continuity of original functions of (1.1) are automatically satisfied. We hope that our study will provide a benchmark of the solutions, which could play a role of comparison basis for other algorithms. All experiments are run in MATLAB, version R2016b, on MACI64. Here, we present a summary of the results obtained; more details for each example are reported in the supplementary material.

Algorithm proposed in this paper is based on the systems of optimality conditions, which depend on λ. The choice of the parameter λ is not trivial and might depend on the structure of the problems.

It has been shown in [Paper 2] that small values of λ perform well for the nonlinear problems of small scale. It is interesting to see if such conjecture holds for the problems with linear structure discussed in this paper. The focus of our experiments in this section will be on the smoothed systems $\Upsilon^{\lambda,\mu}_{KKT}(z)$ and $\Upsilon^{\lambda,\mu}_{LLVF}(z)$, where we set $\mu := 0.001/(1.5^k)$ with k being the number of iterations. For Step 0 of Algorithm 2.6 we set the tolerance to $\epsilon := 10^{-5}$ and the maximum number of iterations to be $K := 200$. We also choose $\alpha_0 := \|\Upsilon^\lambda(z^0)\|$, $\gamma_0 := 1$, $\rho = 0.5$ and $\sigma = 10^{-2}$. For the numerical implementation we calculate the direction d^k by solving (2.31) with Gaussian elimination. In terms of the starting point $(x_0, y_0) = (1_n, 1_m)$ is used for all BOLIB examples and for all transformed integer examples from [32]. For transformed binary examples $(x_0, y_0) = (0.5_n, 0.5_m)$ is used. Two approaches for selecting penalty parameter are tested. For the approach with varying λ we define penalty parameter as $\lambda := 0.5 \times 1.05^k$, where k is the number of iterations. For the approach with fixed values of penalty parameter, eight different values are used for all experiments; i.e., $\lambda \in \{10^5, 10^4, ..., 10^{-2}\}$. After running the experiments for all fixed values of penalty parameter $\lambda \in \{10^5, 10^4, 10^3, ..., 10^{-2}\}$, the best one is chosen i.e. for which the best feasible solution is produced for the particular problem by the tested algorithms. Such best λ for each example will allow us to see if at least one of the selected values worked well to recover the solution. The stopping criteria is defined by the following conditions. Algorithm stops if

1. $\|\Upsilon^\lambda\| < \epsilon$. If the system is solved with prescribed tolerance ϵ.
2. $\|\Upsilon^\lambda(z^k)\| - \|\Upsilon^\lambda(z^{k-1})\| < 10^{-7}$ & $iter > 5$. If the value of the $\|\Upsilon^\lambda\|$ stops improving after a few iterations is done. Specifically algorithm will stop if algorithm stops descending, which prevents algorithm going away from the solution once solution is obtained.
3. **K** iterations performed by the algorithm. The number of iterations reaches predefined maximum number of iterations.

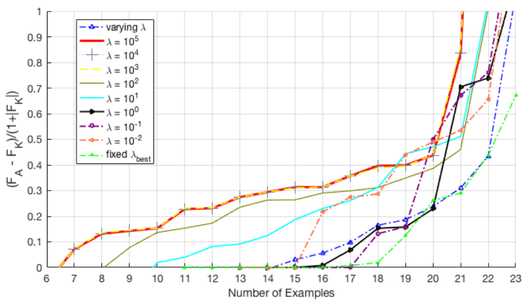
Finally, it is important to mention that for both reformulations algorithm always converges to some solution, that is algorithm never diverges and always produces an output.

3.1 Bilevel Optimization LIBrary (BOLIB) examples

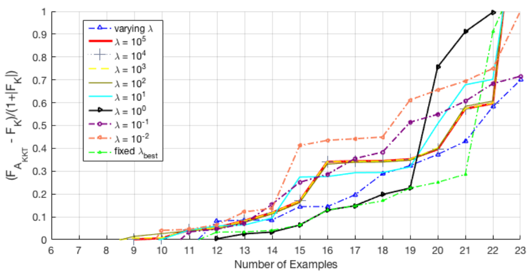
In this section we are going to study the performance of Algorithms 2.6 for 24 linear problems from BOLIB [85]. We are going to compare implementation of Levenberg-Marquardt method for KKT and LLVF reformulations of (1.1). The measures of the comparison will be how to close are upper-level objective values to the known optimal values, how feasible are they in terms of the lower-level objective values, how fast was experimental convergence rate and with what error the actual systems (2.23) and (2.24) are solved. Number of iterations and time required by the algorithms will be reported to finish the comparison. As discussed earlier, fixed choices of λ will include $\lambda \in \{10^5, 10^4, ..., 10^{-2}\}$, while varying λ will be set to be $\lambda := 0.5 \times 1.05^k$. The linear problems in BOLIB [85] are of small dimensions, with the maximum dimensions being $[n, m, q, p] = [10, 6, 12, 13]$.

3.1.1 *Accuracy of the upper-level objective values*

Let us compare the values of the upper-level objective functions at the points computed by algorithm 2.6 with fixed λ and varying λ. As problem 18 ('`MershaDempe2006Ex1`') has no global optimal solution it is not considered in this section, and we focus our attention on the remaining 23 linear BOLIB examples [85]. Let $\bar{F}_{A_{KKT}}$ and $\bar{F}_A$ be the values of upper-level objective function at the point $(\bar{x}, \bar{y})$ obtained by algorithm 2.6 by solving (2.20) and (2.21) systems respectively. Further, let $\bar{F}_K$ be the value of this function at the known best solution point reported in the literature (see corresponding references in [85]). We consider all fixed $\lambda \in \{10^5, 10^4, ..., 10^{-2}\}$ and varying λ in one graph and present the results in Figure 17 below, where we have the relative error on the $y-axis$ and number of examples on the x-axis, starting from $6th$ example. We further plot the results for the best fixed value of λ, discussed earlier. The graphs are plotted in the order of increasing error.



(a) Upper-level error for solving LLVF reformulation



(b) Upper-level error for solving KKT reformulation

Figure 17: Upper-level objective accuracy for LLVF and KKT systems

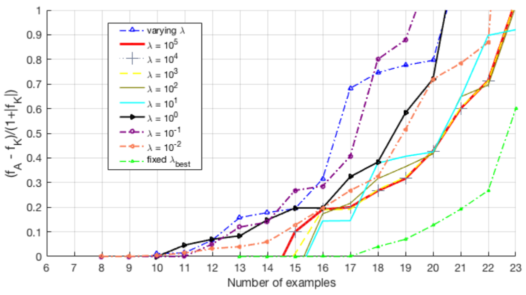
From the Figure 17 we can clearly see that for both reformulations fixed $\lambda = 10^0$ showed very strong performance, recovering upper-level objective value with accuracy error of $\leqslant 20\%$ for 19/23 (82.61%) problems for LLVF case and for 18/23 (78.26%) for KKT case. Interestingly, for fixed large value of penalty parameter, $\lambda \in \{10^5, 10^4, 10^3, 10^2\}$, algorithm showed very poor performance for LLVF reformulation, recovering only 12/23 (52.17%) of the values with the allowable accuracy error of $\leqslant 25\%$. Apart from $\lambda = 10^0$, fixed $\lambda = 10^{-1}$ and varying $\lambda := 0.5 \times 1.05^k$ showed the strongest performance for LLVF reformulation, recovering 19/23 (82.61%) upper-level objective values with accuracy of $\leqslant 20\%$. Finally, we can see that for the best fixed λ we have error of $< 30\%$ for 21/23 problems (91.30%), with the remaining two examples having error of $< 70\%$. This means that if we can pick the best λ we could obtain reasonable solution for almost all of the problems. It is worth noting that varying λ performed very well in the context of LLVF reformulation for the test set of linear problems from BOLIB [85]. It can be seen from Figure 17 (a) that it is almost as good as fixed λ. The situation is very different for KKT reformulation. For large values of the penalty parameter, $\lambda \in \{10^5, 10^4, 10^3, 10^2\}$, algorithm has shown reasonable performance, recovering 15/23 (65.22%) solutions with allowable accuracy error of $\leqslant 20\%$. We further observe that $\lambda \in \{10^1, 10^{-1}, 10^{-2}\}$ recovered only 14/23 (60.87%) of values. Once again, fixed $\lambda = 10^0$ showed the strongest performance, recovering 18/23 (78.26%) solutions. Varying λ for KKT reformulation showed reasonably strong performance, recovering 17/23 (73.91%) solutions. For the best fixed λ we observe error of $< 30\%$ for 21/23 problems (91.30%), which is the same as for LLVF reformulation. However, for the KKT case the remaining two examples have error of $> 90\%$.

With the varying λ we observe that algorithms performed very well in comparison to any particular fixed value $\lambda \in \{10^5, 10^4, ..., 10^{-2}\}$, recovering more than 73% of the solutions with the accuracy error of $\leqslant 20\%$. With the implementation of Algorithm 2.6, solving LLVF reformulation leads to recovering more values of upper-level objective function than solving KKT reformulation. We note that Levenberg-Marquardt algorithm almost always finds a good solution for at least one value of fixed $\lambda \in \{10^5, 10^4, \ldots, 10^{-2}\}$, and always converges as expected. Although best fixed λ showed very strong strong performance, the approach of fixing λ has some disadvantages. Firstly, one would need to consume time, running algorithm for many values of λ. Further, if one does not have a way to choose the best value, or a set of potential values cannot be constructed efficiently, varying λ could be a better option to choose. Varying λ showed almost as good performance as fixed λ_{best} and so could very well be more attractive option to choose rather than fixing λ.

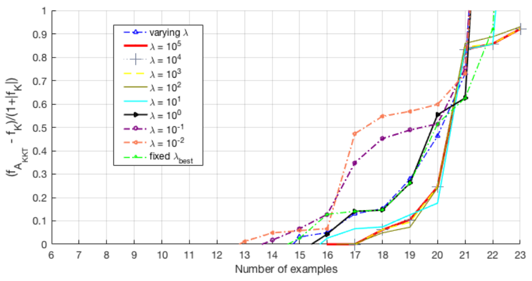
3.1.2 *Lower-level feasibility*

Although due to linearly of (1.1) lower-level problem is convex w.r.t. x and y for all examples in the test set, we cannot guarantee that lower-level feasibility at the computed points would be satisfied for all examples. First of all, Constraint qualification might not be satisfied for some of the problems not satisfying Assumption 2.2. Secondly, the points obtained in practice might have some feasibility error due to overdetermined nature of the systems. For this reason, we aim to compare feasibility with respect to the value of the lower-level objective value in this section. Let $f_{A_{KKT}}$ stand for $f(\bar{x}, \bar{y})$ obtained by Algorithm 2.6 by solving (2.20), f_A stand for $f(\bar{x}, \bar{y})$ obtained by Algorithm

2.6 by solving (2.21) and f_K to be the known optimal value of lower-level objective function. In Figure 18 below we have the lower-level relative error on the y-axis, where the error is plotted in increasing order. Results are presented for all fixed $\lambda \in \{10^5, 10^4, ..., 10^{-2}\}$ as well as for varying λ.



(a) Lower-level error for solving LLVF



(b) Lower-level error for solving KKT

Figure 18: lower-level feasibility for LLVF and KKT systems

From the figure above we can see that the error is negligible ($< 5\%$) for 11 examples for LLVF reformulation and for 14 problems for KKT reformulation for all values of fixed λ and varying λ. From Figure 18 (a) we can see that solutions of LLVF system with $\lambda \in \{10^5, 10^4, 10^3, 10^1\}$ have smaller feasibility error than for the other fixed values of λ. For these values we have feasibility error $\leqslant 20\%$ for 17/23 (73.91%) of the test problems. Surprisingly, feasibility error for LLVF formulation is higher for small fixed values of λ as well as for varying λ. For $\lambda = 10^{-0}$ and for varying λ we have error below 20% for 15/23 (65/22%), while $\lambda \in \{10^2, 10^{-2}\}$ showed smaller feasibility error with the value below 20% for 16/23 (69.57%) test problems. We can further see that feasibility error was much smaller if we could choose best fixed λ. Fixed λ_{best} has less than 20% error for 21/23 (91.30%) problems with the remaining two problems having $\leqslant 60\%$ error. From Figure 18 (b) we observe that for KKT reformulation the feasibility error is on average smaller than for LLVF reformulation. Similar to LLVF, small values of λ performed worse in terms of preserving feasibility of the lower-level objective value. We can see that for $\lambda \in \{10^5, 10^4, 10^3, 10^2, 10^1\}$ the feasibility error is below 20% for 19/23 (82.61%) examples. For $\lambda \in \{10^{-1}, 10^{-2}\}$ the same error was obtained only for 16/23 (69.57%) examples. In terms of varying λ it performed similar to fixed $\lambda = 10^0$ with the error below 20% for 18/23 (78.26%). Interestingly, fixed λ_{best} has shown the same performance as varying λ, making varying λ seemingly strong to implement in the context of KKT reformulation.

3.1.3 *Experimental order of convergence*

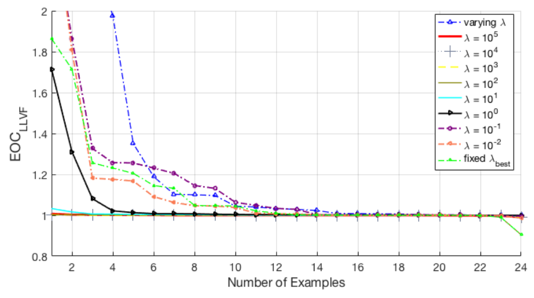
Let us proceed define *Experimental order of convergence* (EOC) for LLVF reformulation to be

$$\text{EOC} := \max\left\{\frac{\log \|\Upsilon^\lambda(z^{K-1})\|}{\log \|\Upsilon^\lambda(z^{K-2})\|}, \frac{\log \|\Upsilon^\lambda(z^{K})\|}{\log \|\Upsilon^\lambda(z^{K-1})\|}\right\},$$

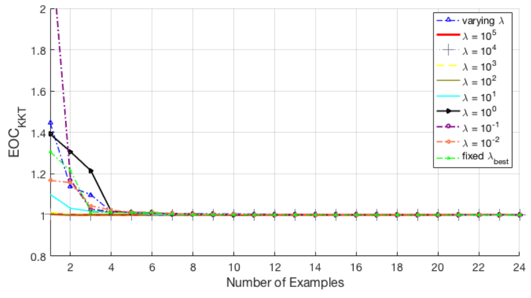
where K is the number of the last iteration [31]. Similarly, experimental order of convergence for KKT reformulation will be defined as

$$\text{EOC}_{KKT} := \max\left\{\frac{\log \|\Upsilon^\lambda_{KKT}(z^{K-1})\|}{\log \|\Upsilon^\lambda_{KKT}(z^{K-2})\|}, \frac{\log \|\Upsilon^\lambda_{KKT}(z^{K})\|}{\log \|\Upsilon^\lambda_{KKT}(z^{K-1})\|}\right\}.$$

If $K = 1$, no EOC will be calculated (EOC$= \infty$). EOC is important to estimate the local behaviour of the algorithm. Let us consider EOC for both reformulation for fixed $\lambda \in \{10^5, ..., 10^{-2}\}$ and for varying λ in Figure 19 below. The values are plotted in the decreasing order of EOC on the y-axis.



(a) EOC for solving LLVF system



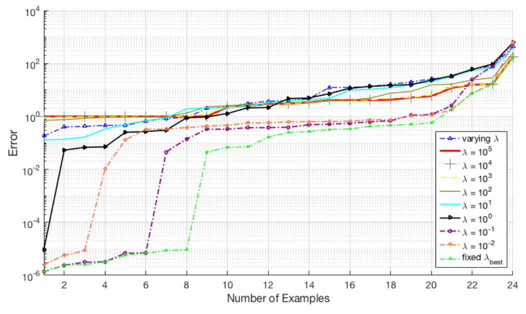
(b) EOC for solving KKT system

Figure 19: Experimental Order of Convergence for LLVF and KKT systems

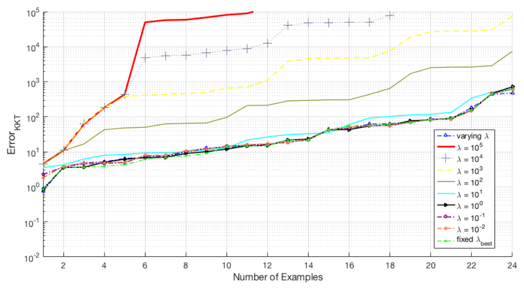
For most of the examples our method has shown linear experimental convergence. This suggests that our method converges linearly, which coincides with the theory behind convergence of the algorithm with line search. It is important to note that the method always converges, although sometimes the output might not be the optimal point for the problem. For LLVF reformulation, in Figure 19 (a), there are a few examples that shown better than linear convergence for each value of λ, with the best ones being fixed values $\lambda \in \{10^0, 10^{-1}, 10^{-2}\}$ and varying λ. For varying λ and for fixed $\lambda = 10^{-1}$ EOC was bigger than 1 for 14/24 problems. For $\lambda = 10^{-2}$ EOC was higher than 1 for 10/24 problems, which is similar to what is observed for best fixed λ. For $\lambda = 10^0$ EOC > 1 was observed for 8/24 problems. The rest of the values show linear convergence for all examples. For $\lambda = 10^6$ one example has shown slightly worse than linear convergence. For KKT formulation, in Figure 19 (b), we can see that for just 4/24 examples convergence rate was better than linear for varying λ and for all values of fixed λ, except large values $\lambda \in \{10^5, 10^4, 10^3, 10^2\}$ for which convergence was linear for all examples. We claim that convergence rate has some more potential to be better than linear for LLVF scenario rather than KKT scenario.

3.1.4 *Comparison of the Error of the systems at the last iteration*

Let us now move on to the comparison of the Error at the last iteration of the Algorithm 2.6. This will show with what tolerance (2.20) and (2.21) were solved and allow us to judge how well algorithm was able to solve the considered systems of optimality conditions for the examples from the test set. In the graph below Error and Error$_{\text{KKT}}$ are defined by $\|\Upsilon^\lambda_{LLVF}(\hat{z})\|$ and $\|\Upsilon^\lambda_{KKT}(\hat{z})\|$ respectively. In the graph below Error is plotted in the increasing order.



(a) The value of the Error $:= \|\Upsilon^\lambda_{LLVF}(\bar{z})\|$ at the solution point of Algorithm 2.6



(b) The value of the Error$_{\text{KKT}}$ $:= \|\Upsilon^\lambda_{KKT}(\hat{z})\|$at the solution point of Algorithm 2.6

Figure 20: Error for LLVF and KKT systems

Firstly, we observe that on average much smaller error was obtained for LLVF reformulation in Figure 20 (a) than for KKT reformulation in Figure 20 (b). For the LLVF reformulation in Figure 20 (a), algorithm with varying λ and with fixed $\lambda \in \{10^5, 10^4, 10^3, 10^2, 10^1\}$ reports the value of the Error in the range $10^{-1} \leqslant$ Error $\leqslant 10^1$ for 15/24 examples and Error $\leqslant 10^2$ for 22/24 examples. For $\lambda = 10^0$ we observe $10^{-5} \leqslant$ Error $\leqslant 10^{-1}$ for 4/24 examples and Error $\leqslant 10^1$ for 15/24 examples.

Finally, the strongest performance in the sense of the LLVF $\mathtt{Error}$ at the last iteration is observed for small fixed values $\lambda \in \{10^{-1}, 10^{-2}\}$ with $\mathtt{Error} \leqslant 10^0$ for 20/24 examples. Best fixed λ shows even better performance for LLVF case, lying below the plotted values of $\mathtt{Error}$ for each value of tested penalty parameter λ. For KKT reformulation in Figure 20 (b) we see that no $\mathtt{Error}$ better than 10^0 was observed. This observation on its own makes a strong point that LLVF reformulation performed better in the sense of measuring $\mathtt{Error}$ at the solution point. This could possibly be due to the extra equation (2.7) containing λ, comparing to LLVF system. Large values of λ, i.e. $\lambda \in \{10^5, 10^4, 10^3\}$, result into very high error for KKT reformulation with $\mathtt{Error}_{KKT} > 10^2$ for 20/24 examples. Interestingly, $\lambda = 10^2$ produces larger error than the smaller values of λ. In terms of the other values $\lambda \in \{10^1, 10^0, 10^{-1}, 10^{-2}\}$ and varying λ, they all show similar performance with the values $10^0 \leqslant \mathtt{Error}_{KKT} \leqslant 10^1$ for 9/24 examples and $\mathtt{Error}_{KKT} \leqslant 10^2$ for 20/24 examples. Best fixed λ also shows the same performance as these value of λ, making the point that varying λ for such reformulation could be a better choice even stronger. With the values of $\mathtt{Error}$ and $\mathtt{Error}_{KKT}$ observed in Figure 20 above LLVF reformulation is a clear winner under this measure.

3.1.5 *Final comments on the numerical performance for BOLIB linear examples*

With the analysis considered in this section we observe that Levenberg-Marquardt method to solve LLVF reformulation outperforms Levenberg-Marquardt method to solve KKT reformulation by almost all of the measures considered. The only slight advantage of KKT reformulation was feasibility of the lower-level problem. However, feasibility obtained for LLVF approach was not dramatically worse than for KKT approach, meaning there is no big concern regarding feasibility of the obtained solutions. To finalize this section, let us compare the average number of iterations required to solve each reformulation for each choice of λ, as well as the average time taken by the algorithm.

λ		$\lambda = 10^5$	$\lambda = 10^4$	$\lambda = 10^3$	$\lambda = 10^2$	$\lambda = 10^1$	$\lambda = 10^0$	$\lambda = 10^{-1}$	$\lambda = 10^{-2}$	λ_{var}
Average	**LLVF**	109.58	109.08	150.71	144.04	73.08	50.25	111.08	106.25	23.75
Iter	**KKT**	167.92	167.92	164.17	118.12	63.54	20.38	18.92	18.46	12.21
Average	**LLVF**	0.21	0.09	0.09	0.09	0.05	0.04	0.07	0.08	0.14
Time	**KKT**	0.19	0.13	0.11	0.07	0.04	0.02	0.02	0.02	0.07
Average	**LLVF**	0.0019	0.0008	0.0006	0.0006	0.0007	0.0007	0.0006	0.0007	0.0059
Time/Iter	**KKT**	0.0011	0.0008	0.0007	0.0006	0.0007	0.0011	0.0011	0.0011	0.0058

Table 4: Number of iterations and time used by the algorithms for BOLIB linear examples

From the table above we can see that algorithms show very similar speed and similar number of iterations required to converge to a solution. We observe that more iterations are taken by KKT approach for fixed $\lambda \geqslant 10^3$, while for the rest of the choices of λ LLVF approach used more iterations to converge. We shall not forget that LLVF produced better solutions and hence smaller number of iterations taken by KKT approach could be for the cases where KKT system could not produce solution reasonably close to the optimal. The speed of each iteration is almost the same for both approaches. Interestingly, time needed to do an iteration for varying λ is more than 5 times bigger than needed for any fixed value of λ for both reformulations. However, much less iterations are needed to converge for varying λ, which makes the total average time taken by the algorithm almost the same as for any fixed λ.

Summarizing, LLVF-based optimality conditions for linear bilevel optimization were compared with KKT-based optimality conditions for linear bilevel optimization. With regards to theoretical aspects we could not say which approach is better. We have used NCP-functions and smoothing technique to state optimality conditions of both reformulations in the form of differentiable systems of equations. Smoothed Levenberg-Marquardt method was then proposed to solve the systems. We have shown that Levenberg-Marquardt method could converge to solve such systems under mild assumptions. The results of the numerical experiments were then presented, where we demonstrated the results of implementing smoothed Levenberg-Marquardt method for linear bilevel optimization in the context of solving KKT and LLVF reformulations. Two approaches of choosing λ were considered for each of the two reformulations. The method with varying λ showed strong performance for the linear framework in comparison to the nonlinear scenario considered in [Paper 2]. We further observe that the method has shown linear experimental order of convergence for most of the examples for both reformulations. In terms of the comparison, LLVF-based approach outperformed KKT-based approach in terms of recovering upper-level objective value, obtaining smaller Error at the solution point and demonstrating faster order of convergence. This extends similar observation made for nonlinear bilevel problems in the comparison analyzed in [84].

3.2 Fischetti et al. based examples

In this section we transform mixed integer bilevel linear examples from [32] to linear bilevel optimization examples. We do so by dropping integer constraints for examples with integer variables and we impose bounds on the variables to be between 0 and 1 for binary examples. The examples from [32] are initially in the format of mps files with the following structure

$$\begin{aligned} &\min_x c^\top x && (3.1)\\ &\text{s.t.:}\ Aineq * x \leqslant bineq && (3.2)\\ &\quad Aeq * x = beq && (3.3)\\ &\quad lb \leqslant x \leqslant ub && (3.4)\\ &\quad x(i) \text{ is integer}, && (3.5)\end{aligned}$$

where $i \in intcon$ (integer constraints). The whole test set from [32] has integer constraints on all variables, as $i = 1$ for all components of $intcon$. Further, some examples have binary variables, i.e. $x_i \in \{0, 1\}$. Apart from mps-files, some infromation is stored in AUX files. The information there includes dimension of lower-level variable "N", indices required for lower-level objective "LO" and indices for which variables and constraints are lower-level. It is worth noting that all constraints involving both upper-level and lower-level variables are considered to be lower-level constraints. We extract information from mps files in MATLAB, using the command *mpsread*, obtaining the matrices $c, Aineq, bineq, Aeq, beq, lb, ub$ as described by (3.1)-(3.5) above. This information from mps-files is combined with the information from aux files as follows. In the sense of bilevel problem, the variable x above is the combination of upper-level variable x and lower-level variable y. We split the variable into upper-level and lower-level by using the information from aux file. Most importantly, we extract the dimension of y denoted by N and indices of y for lower-level objective denoted by LO. We then split the bounds (3.4) into upper-level and lower-level constraints based on the split of the variables into upper-level and lower-level ones. We define upper-level objective as $F(x, y) := c^\top x = c_x^\top x + c_y^\top y$, where $c^\top x$ comes from (3.1). All inequality constraints involving both upper-level and lower-level variables are treated as lower-level constraints. Hence, we move (3.2) to be lower-level constraints together with bounds on y coming from (3.4). Although it was observed that there are no equality constraints in the test set, our transformation includes redefining any equality constraints (3.3) as inequality constraints and move it to upper-level constraints $G(x, y)$ together with the bounds on variable x resulting from (3.4). Finally, we drop integer constraint (3.5) and define lower-level objective as suggested by aux file (coefficients of $f(x, y)$ are given by LO). Then the resulting system is

$$\begin{aligned} &\min_{x,y} c_x^\top x + c_y^\top y && (3.6)\\ &\text{s.t.:}\ Aeq * (x, y) - beq \leqslant 0 && (3.7)\\ &\quad beq - Aeq * (x, y) \leqslant 0 && (3.8)\\ &\quad lb \leqslant x \leqslant ub && (3.9)\\ &\quad \min_y\ LO * y && (3.10)\\ &\quad \text{s.t.:}\ Aineq * (x, y) - bineq \leqslant 0 && (3.11)\\ &\quad lb \leqslant y \leqslant ub && (3.12)\end{aligned}$$

For all 174 considered examples there are no equality constraints, so we do not have (3.7)-(3.8) in the model. For our transformation, coefficients LO from aux file are defined as the vector $LO = LO_1, LO_2, .., LO_m$ for each example in the corresponding m-file of the example, such that LO above is the vector of coefficients for y. Further, if an example is binary, we relax the binary constraints as the bound constraints $0 \leqslant x_i \leqslant 1$ and $0 \leqslant y_j \leqslant 1$ for all $i = 1, ..., n$ and $j = 1, ..., m$. The bound constraints are then stated as upper-level constraints G for variables x_i and as lower-level constraints g for variables y_i.

Since examples have been modified, they are considered to be new examples with not-known solutions. We denote solutions obtained by Algorithm 2.6 by $(\hat{x}, \hat{y})$ and $\lambda_{var} := 0.5 \times 1.05^k$, where k is the number of the iteration. In Table 8 (Appendix A) we provide the solutions obtained by our algorithms for 50 transformed integer examples from [32] (initial mixed integer examples are available in "RAND_BILEVEL" folder from msinnl.github.io/bilevel/MIPLIB). The starting point for these examples was $(x_0, y_0) = (1_n, 1_m)$. For the names of these examples we have replaced "mi" (standing for "mixed integer") by capital letter "B". In Table 9 (Appendix B) we provide solution obtained for 124 transformed binary examples from [32] (initial binary examples are available in "TedConverted" folder from msinnl.github.io/bilevel/MIPLIB). The starting point for this examples was $(x_0, y_0) = (0.5_n, 0.5_m)$. The names of these examples were changed by adding capital "B" at the start of the name of each example. The dimension of the examples are summarized as follows.

No	n	m	q	p
1–10	5	10	10	40
11–40	15	5	30	30
41–50	5	15	10	50

(a) Dimensions of the transformed integer examples

No	n	m	q	p
1–25	25	25	50	96
26–44	10	10	20	32
45–64	20	20	40	62
65–84	30	30	60	92
85–104	40	40	80	122
105–114	50	50	100	152
115–124	20	20	40	62

(b) Dimensions of the transformed binary examples

Table 5: Dimensions of $x, y, G(x, y)$ and $g(x, y)$ of the transformed problems from [32]

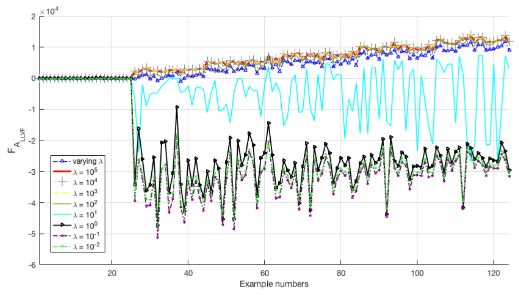
We hope the results in Table 8 (Appendix A) and Table 9 (Appendix B) will provide a basis for comparison for anyone attempting to solve the examples with different algorithms. As the solutions are not known we cannot claim which solutions in tables 8 and 9 are the best. Clearly, the smallest value of upper-level objective might not be the best solution as such solutions could possibly be infeasible. Further, the values of $\text{Error} := \left\|\Upsilon^{\lambda}(\hat{z})\right\|$ and $\text{Error}_{KKT} := \left\|\Upsilon^{\lambda}_{KKT}(\hat{z})\right\|$ are also not great measures as these directly depend on the value of λ, meaning that smaller values of λ would lead to smaller values of error at the solution point, but the quality of the solutions is not necessary better than for bigger values of λ. Hence, we do not have a good unbiased way to choose the best solutions. We hope the provided values of $F(\hat{x}, \hat{y})$ and $f(\hat{x}, \hat{y})$ will enable other authors to make conclusions about their own solution methods. Let us also provide a summary of the performance and speed of the smoothed Levenberg-Marquardt algorithm in the context of solving LLVF-based and KKT-based optimality conditions for these examples. For the relaxed integer examples presented in Table 8 (Appendix A) we observe the following performance of Algorithm 2.6 in terms of the iterations made and CPU time used.

λ		$\lambda = 10^5$	$\lambda = 10^4$	$\lambda = 10^3$	$\lambda = 10^2$	$\lambda = 10^1$	$\lambda = 10^0$	$\lambda = 10^{-1}$	$\lambda = 10^{-2}$	λ_{var}
Average	**LLVF**	140	130	150	200	190	170	180	200	37
iter	**KKT**	160	140	130	100	200	200	170	160	21
Average	**LLVF**	1.3	0.91	1	1.4	1.3	1.2	1.2	1.4	0.43
time	**KKT**	1.3	1.1	0.99	0.78	1.5	1.5	1.3	1.3	0.28
Average	**LLVF**	0.009	0.007	0.007	0.007	0.007	0.007	0.007	0.007	0.012
time/iter	**KKT**	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.008	0.013

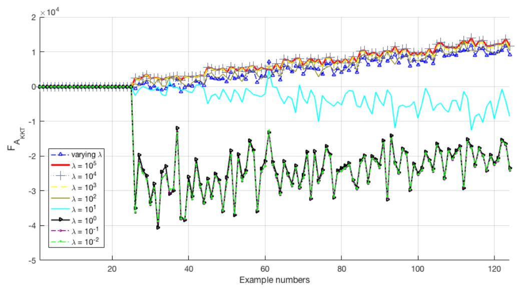
Table 6: Time and iterations required to solve LLVF and KKT reformulations of relaxed integer problems

For both reformulations Levenberg-Marquardt algorithm shows similar speed for solving relaxed integer examples. For LLVF system slightly more iterations were made but iterations are also less expensive, leading to very similar average time required to solve one system or another. It is interesting to observe that algorithm with varying λ takes significantly more time to make an iteration, which is in line to what was observed for linear examples from BOLIB [85]. In terms of experimental order of convergence (EOC), for both reformulations algorithm has shown super-linear convergence for all fixed values of λ for all examples from Table 8 (Appendix A). For varying λ the convergence was slightly faster with $1 \leqslant EOC_{LLVF} \leqslant 1.1$ for 15/50 examples and $1 \leqslant EOC_{KKT} \leqslant 1.05$ for 5/50 examples. As we can see in the table above, there is no much difference in terms of computation time of the algorithm but some difference in EOC suggests that algorithm to solve LLVF-based system converges slightly faster than for KKT-based approach.

The values $F(\hat{x}, \hat{y})$ and $f(\hat{x}, \hat{y})$ for the relaxed binary examples from [32] are presented in Table 9 (Appendix B). There are 124 such examples that we have transformed by replacing binary constraints by the bounds on the variables. It is interesting to observe that for the examples **No** $26 - 124$ in Table 9 (Appendix B) for fixed $\lambda \in \{10^0, 10^{-1}, 10^{-2}\}$ the algorithms produced solutions $(\hat{x}, \hat{y})$ with very small values of upper-level objective $F(\hat{x}, \hat{y}) < -10^4$ and very large values of of the lower-level objective $f(\hat{x}, \hat{y}) \geqslant 10^4$. In contrast, for fixed $\lambda \in \{10^5, 10^4, 10^3, 10^2\}$ algorithm returns rather large upper-level values $F(\hat{x}, \hat{y})$ and small lower-level values $f(\hat{x}, \hat{y})$ for both reformulations. This is demonstrated in Figures 21 and 22 below, where we have the values of the objective functions at the solution point on the $y - axis$ with the scale of 10^4.

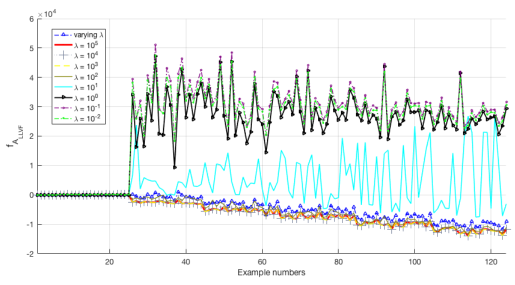


(a) Upper-level objective for LLVF reformulation

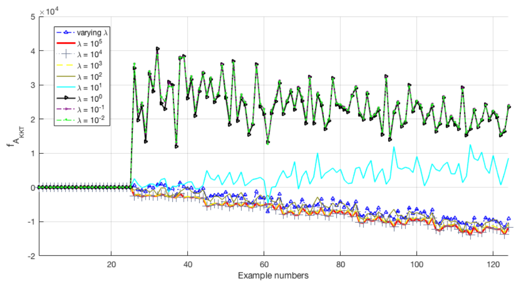


(b) Upper-level objective for KKT reformulation

Figure 21: Upper-level objective values for LLVF and KKT systems for relaxed binary examples



(a) lower-level objective for LLVF reformulation



(b) lower-level objective or KKT reformulation

Figure 22: Lower-level objective values for LLVF and KKT systems for relaxed binary examples

Due to the large observed values, It is very likely that solutions for $\lambda \in \{10^0, 10^{-1}, 10^{-2}\}$ are infeasible for the lower-level problem. This would mean that solutions for the LLVF and KKT reformulations with small values of λ might be infeasible for majority of the considered transformed binary examples (in Appendix B). Further, solutions for $\lambda \in \{10^5, 10^4, 10^3, 10^2\}$ for transformed binary examples are likely to be bad solutions from the optimistic perspective as the values of upper-level objective are large. It is interesting that solutions for varying λ seem similar to solutions for large values of λ, but slightly better. Finally, solutions for $\lambda = 10^1$ lie in the middle between those two extreme scenarios and seem to be the most reasonable solutions reported in Appendix B. With these observations our conjecture is that it could be the case that the problem was *under-penalized* for small values of λ and *over-penalized* for large values of λ, making medium value $\lambda = 10^1$ and varying λ more attractive choices. This is in line with the initial motivation of using different values of λ with the idea of not over-penalizing and not under-penalizing deviation of lower-level objective values from the minimum. In terms of iterations and time used by the algorithms, we observe the following performance for the relaxed binary examples presented in Appendix B.

λ		$\lambda = 10^5$	$\lambda = 10^4$	$\lambda = 10^3$	$\lambda = 10^2$	$\lambda = 10^1$	$\lambda = 10^0$	$\lambda = 10^{-1}$	$\lambda = 10^{-2}$	λ_{var}
Average iter	**LLVF**	200	180	160	92	85	170	170	170	41
	KKT	9.9	8.6	8.2	9.7	76	160	160	160	49
Average time	**LLVF**	4.2	3.5	3.1	1.5	1.8	3.2	3.2	3.2	1
	KKT	0.32	0.23	0.21	0.24	2	3.7	3.7	3.7	1.3
Average time/iter	**LLVF**	0.021	0.019	0.02	0.016	0.021	0.02	0.019	0.019	0.025
	KKT	0.033	0.027	0.026	0.024	0.026	0.023	0.023	0.023	0.026

Table 7: Time and iterations required to solve LLVF and KKT reformulations of relaxed binary problems

For both reformulations Levenberg-Marquardt algorithm demonstrated similar performance on average but very different for different magnitudes of λ. The small number of iterations for large values of λ for KKT system indicates that algorithm stopped descending quite early, which could either be a problem with the direction for the reformulation, or a good solution was obtained very early. It is interesting to see that time needed to make an iteration for relaxed binary examples (from Appendix B) is more than twice more expensive than for integer examples (from Appendix A).

4 FINAL COMMENTS

We have shown that the assumptions needed to reformulate linear bilevel programming problem as the single-level KKT and LLVF reformulations are much weaker than for the nonlinear case. However, even with this it is still hard to tell which reformulation is theoretically better for the linear bilevel optimization framework. To proceed, we have used NCP-functions and smoothing technique to formulate KKT-based and LLVF-based optimality conditions in the form of differentiable systems of equations. We then focused on testing the implementation of Levenberg-Marquardt method to solve these systems. We have shown that the method can theoretically converge for the linear bilevel framework under mild assumption. We have then presented the numerical comparison of implementing Levenberg-Marquardt method for KKT-based and LLVF-based approaches on the test of linear bilevel problems from BOLIB, [85]. It was observed that Levenberg-Marquardt method for LLVF reformulation has shown slightly better numerical performance than Levenberg-Marquardt method for KKT system. However, the test set was not very large, having just 24 problems of small size. This motivated creating the basis for comparison for the larger test set of linear bilevel problems. In the final part of the paper, we have transformed mixed integer and binary problems from [32]. We have presented the obtained results in tables 8 and 9 in terms of the objective functions values at the solution points for both reformulations. Finally, we have provided general observations on the obtained results. It is worth saying that we avoided any suggestions on which results are the best, as the solutions to the modified problems are clearly unknown.

A OBJECTIVE FUNCTIONS VALUES AT THE SOLUTION POINT FOR TRANSFORMED INTEGER EXAMPLES FROM [32]

λ			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
No	Problem names		$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$
1	**Bblp_20_15_50_10_1**	**LLVF**	-613	-142	-611	-141	-590	-136	-628	-150	156	122	1670	745	635	-542	229	-464	144	-17
		KKT	63	-530	63	-530	63	-531	65	-535	78	-546	298	-547	203	-489	103	-254	-102	-455
2	**Bblp_20_15_50_10_2**	**LLVF**	-234	132	-232	132	-201	133	330	134	441	74	1432	-630	521	482	-105	405	383	14
		KKT	-179	414	-179	414	-179	414	-180	418	-196	421	-293	387	-245	308	-161	263	-281	320
3	**Bblp_20_15_50_10_3**	**LLVF**	79	-93	78	-93	66	-97	-21	-343	-217	-482	-506	-3014	170	-622	57	-280	49	-136
		KKT	204	-542	204	-542	204	-543	206	-547	209	-576	226	-854	169	-204	3	57	120	-11
4	**Bblp_20_15_50_10_4**	**LLVF**	-351	-217	-344	-214	-344	-217	-387	-400	-269	-687	-249	485	442	2643	46	1847	-278	61
		KKT	-26	692	-26	692	-26	692	-25	701	32	762	236	823	-200	143	-243	128	-81	354
5	**Bblp_20_15_50_10_5**	**LLVF**	-153	141	-155	140	-183	132	-467	59	-217	-291	112	-698	77	-887	154	-357	260	-628
		KKT	-230	-375	-230	-375	-230	-375	-232	-378	-196	-389	-95	-359	444	50	-91	-189	-30	-323
6	**Bblp_20_15_50_10_6**	**LLVF**	-132	-119	-134	-119	-152	-114	-308	-18	-97	30	-181	4537	551	668	597	105	244	353
		KKT	329	-195	329	-195	329	-195	332	-196	378	-175	369	-147	384	-36	165	-161	429	-141
7	**Bblp_20_15_50_10_7**	**LLVF**	-146	69	-145	70	-132	81	-74	127	-87	83	-396	-543	-931	-1698	-414	-172	-208	-171
		KKT	-396	117	-396	117	-396	117	-398	117	-440	109	-618	81	-426	33	-124	85	-486	110
8	**Bblp_20_15_50_10_8**	**LLVF**	78	90	78	90	76	85	52	40	-772	-770	-2397	-2149	-4672	-2386	-2939	-1440	-1191	-755
		KKT	-117	199	-117	199	-117	199	-119	201	-76	244	-50	372	4	106	74	233	25	290
9	**Bblp_20_15_50_10_9**	**LLVF**	-302	-3	-300	-2	-285	5	-473	75	227	1249	2492	-520	1419	-817	182	-271	115	180
		KKT	-232	-191	-232	-191	-232	-191	-234	-193	-272	-198	-460	-108	-574	-203	-261	-226	-400	-272
10	**Bblp_20_15_50_10_10**	**LLVF**	435	379	436	378	447	371	427	270	-3198	-973	-1127	-253	-428	690	472	722	-144	181
		KKT	483	545	483	545	483	546	487	550	444	563	489	640	579	808	472	551	171	13
11	**Bblp_20_20_50_5_1**	**LLVF**	-389	204	-380	176	-1136	282	-1121	176	-237	-96	1074	-923	142	-517	33	-346	406	-161
		KKT	-51032	-12083	-44140	-10554	-32917	-7957	-26312	-6432	-14551	-3775	-609	-719	447	-288	286	-133	444	-213
12	**Bblp_20_20_50_5_2**	**LLVF**	-403	-52	-401	-52	-372	-56	-111	-83	214	-239	5577	167	8832	-949	1257	-1103	151	-807
		KKT	157	-480	157	-480	157	-481	160	-485	115	-525	-697	-643	-793	-577	-57	-135	33	-285
13	**Bblp_20_20_50_5_3**	**LLVF**	-474	-118	-206	-92	-223	-92	-440	-104	-305	-94	-184	-414	0	-451	-7	-352	10	-68
		KKT	-3569	-736	-3569	-736	-3573	-737	-3607	-743	-3863	-788	-1175	-430	225	-134	280	-84	249	92
14	**Bblp_20_20_50_5_4**	**LLVF**	-702	-238	-702	-238	-701	-239	-589	-335	-491	-316	-162	-349	-510	-316	-452	-294	-598	-319
		KKT	-153	-282	-153	-282	-153	-282	-153	-283	-142	-290	-283	-369	-330	-273	-334	-242	-247	-244
15	**Bblp_20_20_50_5_5**	**LLVF**	-128	-56	-93	-16	-134	-50	32	61	671	65	4934	-655	1940	826	47	-334	-12	-52
		KKT	-492	111	-493	111	-493	111	-489	109	-522	103	-127	-292	-76	-7	-4	17	-86	-33
16	**Bblp_20_20_50_5_6**	**LLVF**	-413	86	-415	86	-436	91	-597	142	-529	-913	6117	-2865	-71	-1440	-829	-1417	-222	101
		KKT	-1607	-847	-1608	-847	-1609	-848	-1570	-805	-1679	-954	15	-995	-173	-281	-201	59	-232	61
17	**Bblp_20_20_50_5_7**	**LLVF**	-65	-199	-66	-198	-74	-197	-125	-196	-1594	-751	3549	-1403	-751	-945	-930	-811	-92	-1131

			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
			$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$
17	**Bblp_20_20_50_5_7**	**KKT**	231	-102	231	-102	231	-102	233	-103	241	-120	-514	-126	-409	-519	164	-214	165	-219
18	**Bblp_20_20_50_5_8**	**LLVF**	-578	-58	-576	-57	-559	-50	-362	83	-1808	1403	1732	1877	1622	2603	450	318	-177	706
		KKT	-2566	1381	-2567	1382	-1911	1050	-1134	664	-932	537	-88	-1027	-327	-193	-319	-66	43	-145
19	**Bblp_20_20_50_5_9**	**LLVF**	-205	41	-214	39	-110	177	-389	190	-1326	318	-1256	-250	-859	-297	-490	-254	-372	-169
		KKT	-5774	-2687	-5775	-2687	-5810	-2702	-5835	-2713	-6167	-2875	-1378	-907	1	-90	92	-44	91	-50
20	**Bblp_20_20_50_5_10**	**LLVF**	-20	82	-22	83	-41	95	-220	167	-421	394	-855	611	-195	392	-116	276	-26	166
		KKT	-329	782	-330	782	-330	782	-325	767	-377	861	-438	597	72	21	193	-65	249	-84
21	**Bblp_20_20_50_5_11**	**LLVF**	497	-56	498	-56	499	-49	478	19	3016	546	5083	3290	-858	261	-140	86	59	-138
		KKT	-50054	-50027	-23217	-23280	-6054	-6220	-2531	-2727	-2919	-3165	516	-218	-16	-195	223	32	277	-83
22	**Bblp_20_20_50_5_12**	**LLVF**	318	81	318	81	315	82	234	92	919	158	-2189	-685	-334	-73	1288	-409	693	-84
		KKT	-136180	63604	-55240	26397	-24810	12108	-17439	8663	-5879	3335	293	405	1158	-31	394	208	447	66
23	**Bblp_20_20_50_5_13**	**LLVF**	-42	109	-41	108	-43	109	991	133	-584	279	1313	-186	-232	-160	-211	-213	51	-80
		KKT	-857	-227	-857	-227	-858	-227	-869	-230	-946	-237	-1432	-233	-559	-223	-192	9	-556	6
24	**Bblp_20_20_50_5_14**	**LLVF**	-1384	-912	-1384	-911	-1378	-904	-1365	-858	-672	-1173	-758	-1574	-13	-492	-18	-542	-1143	-419
		KKT	-2023	-1584	-1846	-1488	-781	-879	-169	-522	-529	-743	-236	-433	-455	-849	22	-110	-202	-414
25	**Bblp_20_20_50_5_15**	**LLVF**	408	314	380	286	403	305	409	230	-710	483	-553	254	1318	-531	1671	-782	956	-153
		KKT	82760	-56171	51699	-35131	32923	-22448	26333	-17997	6775	-4788	841	-523	667	-104	70	42	651	-134
26	**Bblp_20_20_50_5_16**	**LLVF**	-160	101	-158	100	-140	100	-38	96	290	137	3593	1949	3214	640	333	-17	456	-258
		KKT	-177	232	-177	233	-177	233	-177	235	-232	251	-223	225	-830	-210	-56	-3	129	31
27	**Bblp_20_20_50_5_17**	**LLVF**	-309	168	-311	168	-330	174	-563	247	-3272	421	-13207	4038	-3738	1789	-1677	496	-1542	418
		KKT	-595	-539	-595	-539	-595	-540	-581	-515	-660	-636	27	-332	-425	230	-279	162	-285	-43
28	**Bblp_20_20_50_5_18**	**LLVF**	-6	18	-5	18	-9	20	-139	-3	-132	-466	2284	-7512	636	-2565	172	-206	120	-405
		KKT	-331	-412	-331	-412	-332	-413	-339	-420	-364	-410	-177	-74	-584	-103	149	20	61	6
29	**Bblp_20_20_50_5_19**	**LLVF**	486	-15	487	-15	498	-17	672	-58	778	528	-737	7209	109	3407	499	270	547	104
		KKT	1388	45	1388	45	1390	45	1389	46	293	53	163	-69	435	-46	59	-36	487	107
30	**Bblp_20_20_50_5_20**	**LLVF**	-67	201	-68	200	-70	197	-94	181	-477	-20	3705	-595	3719	-422	-373	-137	-485	-354
		KKT	419	-29	419	-29	419	-29	422	-29	407	-36	99	4	-581	-335	81	56	70	5
31	**Bblp_20_20_50_10_1**	**LLVF**	-57	74	-58	76	-56	75	101	635	257	259	1218	-91	-2429	-83	-1723	-17	-58	-68
		KKT	-1834	-34	-1834	-34	-1835	-34	-1823	-30	-1914	-26	-1396	149	-359	-175	-165	72	-581	147
32	**Bblp_20_20_50_10_2**	**LLVF**	-49	-64	-49	-65	-45	-65	242	104	-255	398	116	-357	638	-399	-392	-545	28	-157
		KKT	16423	-776	16390	-774	7075	-461	2249	-278	5903	-394	983	-3	-261	-427	12	-109	49	-55
33	**Bblp_20_20_50_10_3**	**LLVF**	-429	264	-429	257	-409	185	-613	487	-892	-270	-735	-557	-700	-383	-164	-190	-337	-32
		KKT	-253	-228	-253	-228	-253	-229	-257	-229	-217	-250	-184	-507	-126	-224	-180	-33	-175	34
34	**Bblp_20_20_50_10_4**	**LLVF**	-800	-322	-796	-321	-752	-310	-628	-365	-359	-880	-1808	-2894	11	-53	421	-423	-137	-58
		KKT	-17639	1666	-16029	1502	-14481	1345	-14248	1321	-2099	50	-96	-364	130	-382	-318	-104	-296	-76

			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
			$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$
35	**Bblp_20_20_50_10_5**	**LLVF**	-596	209	-428	236	-431	236	-1145	116	-1474	-2557	1378	-1398	3915	-1491	1771	-732	-230	85
		KKT	123753	-60630	110821	-54191	48611	-23459	29791	-14123	31169	-14845	3434	-1277	602	-335	259	123	701	22
36	**Bblp_20_20_50_10_6**	**LLVF**	-54	-766	-58	-763	-98	-721	-354	-421	-893	853	-918	527	-3019	3517	-1785	1005	-850	87
		KKT	-1799	19676	-1789	19475	-1636	15009	-1059	2182	-1300	6454	-1408	683	-643	-68	-227	-102	-674	-86
37	**Bblp_20_20_50_10_7**	**LLVF**	-94	409	-88	401	-94	408	-434	251	-759	2348	8161	5558	506	571	1338	905	-126	472
		KKT	63	-699	63	-699	63	-699	-28	-594	85	-777	-252	-112	116	232	-11	161	-144	260
38	**Bblp_20_20_50_10_8**	**LLVF**	-140	-335	-141	-335	-278	-252	-1204	-242	-729	-117	-1607	-28	-1558	-860	-286	-888	-365	-365
		KKT	32302	-7342	31815	-7233	29369	-6679	29511	-6708	8716	-1910	-159	-6	71	-624	-140	-346	-139	-379
39	**Bblp_20_20_50_10_9**	**LLVF**	413	-418	413	-418	415	-419	463	-1381	-789	-2050	-981	-490	-819	-1331	-175	-285	-116	-103
		KKT	20666	-20859	25634	-25824	8867	-9140	2375	-2691	6274	-6656	522	-1392	103	-246	82	178	4	136
40	**Bblp_20_20_50_10_10**	**LLVF**	238	-59	239	-58	240	-59	698	-116	1962	-480	-1673	3032	-1171	1097	-265	177	-144	181
		KKT	1689	24	1689	24	1691	24	1678	27	1771	32	579	274	164	-31	203	-22	171	13
41	**Bblp_20_20_50_15_1**	**LLVF**	-1288	-562	-1288	-563	-1290	-573	-1296	-680	-1673	-831	-2262	-22	-1694	-16	-1386	18	-186	-48
		KKT	-866	196	-866	196	-868	196	-883	198	-626	296	-469	411	-302	147	-121	212	103	87
42	**Bblp_20_20_50_15_2**	**LLVF**	87	568	78	561	-23	501	-389	108	-680	-625	-1186	-2176	-116	-235	-282	233	-35	30
		KKT	-273	359	-273	359	-273	360	-276	362	-341	403	-173	299	-48	418	132	415	260	398
43	**Bblp_20_20_50_15_3**	**LLVF**	-124	22	-123	24	-109	45	-66	373	260	103	-427	583	-416	23	-270	-90	-143	81
		KKT	149	198	149	198	149	198	152	200	124	103	-341	-130	-195	-58	-163	-119	-297	-188
44	**Bblp_20_20_50_15_4**	**LLVF**	-231	-164	-206	-189	-209	-191	-634	-319	-1364	-538	-3351	-390	-2491	-1141	-1407	-958	-173	-123
		KKT	-9201	-15140	-6847	-11316	-4664	-7774	-3600	-6045	-1573	-2927	-122	-256	64	-156	-14	-200	239	-97
45	**Bblp_20_20_50_15_5**	**LLVF**	232	255	228	268	233	254	336	982	-45	1163	-163	427	322	-113	-112	-144	46	111
		KKT	-301	161	-301	161	-302	161	-305	162	30	-148	118	-228	226	-49	288	-118	125	-104
46	**Bblp_20_20_50_15_6**	**LLVF**	745	88	745	88	747	90	891	1081	-373	2813	-890	940	-880	640	-30	136	167	-144
		KKT	-10027	14672	-10028	14673	-9310	13549	-8548	12345	-7006	9599	-1378	1816	291	259	394	244	-48	606
47	**Bblp_20_20_50_15_7**	**LLVF**	-377	137	-367	141	-388	127	-1138	-365	-503	287	-322	873	-2310	-573	-1708	-289	-687	71
		KKT	-5526	3140	-5152	3248	-4149	3045	-3772	3033	-2928	2630	-962	671	-315	390	-175	323	-244	327
48	**Bblp_20_20_50_15_8**	**LLVF**	-216	353	-215	352	-207	343	28	-881	2898	-5799	1547	-3916	94	-4026	-608	-1213	-301	-88
		KKT	131	-1863	131	-1864	131	-1865	133	-1883	174	-1986	-272	-706	-608	328	-370	364	-384	382
49	**Bblp_20_20_50_15_9**	**LLVF**	-174	370	-178	369	-219	357	-656	506	-235	2377	2059	1197	1206	-92	367	353	-171	251
		KKT	-2546	8970	-2381	8228	-2136	7181	-1062	2603	-1426	4305	-588	832	-500	303	-332	395	-553	310
50	**Bblp_20_20_50_15_10**	**LLVF**	-96	-161	-62	38	-152	-124	-546	155	-208	-582	-2024	-593	-705	261	-437	410	-37	73
		KKT	-143344	33578	-135842	31834	-80697	18906	-51524	12074	-33645	7781	-2739	365	-301	262	-145	384	22	242

Table 8: Upper-level and lower-level objective values at the solution point of algorithms **??** and 2.6 for relaxed integer examples from [32]

B OBJECTIVE FUNCTIONS VALUES AT THE SOLUTION POINT FOR TRANSFORMED BINARY EXAMPLES FROM [32]

λ			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
No	Problem names		$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$	$F(\hat{x},\hat{y})$	$f(\hat{x},\hat{y})$
1	**B2AP05-1**	**LLVF**	-34	43	-36	40	-60	58	-91	87	-92	87	-42	-33	-99	-11	-104	-4	-11	12
		KKT	76	238	-7	104	20	161	2	133	-19	78	-43	38	-90	66	-94	70	-52	52
2	**B2AP05-2**	**LLVF**	-39	37	-32	38	-54	55	-81	81	-82	83	-82	-82	-151	-28	-165	-20	-23	23
		KKT	-203	43	-93	44	-143	49	-121	49	-70	42	-46	34	-94	64	-98	67	-54	54
3	**B2AP05-3**	**LLVF**	-42	38	-46	38	-68	57	-102	86	-103	87	-10	-76	-94	3	-105	25	-30	30
		KKT	65	7	-23	36	3	31	-16	37	-35	38	-53	40	-106	77	-110	80	-64	64
4	**B2AP05-4**	**LLVF**	-36	34	-37	37	-55	60	-83	89	-84	90	-3	-92	-105	-32	-123	-11	-7	7
		KKT	62	-141	-14	-8	10	-54	-7	-23	-26	14	-45	35	-79	68	-86	72	-52	52
5	**B2AP05-5**	**LLVF**	-40	40	-41	39	-60	59	-90	88	-91	89	-34	-64	-99	-20	-109	-12	-18	18
		KKT	11	102	-32	61	-21	81	-31	73	-37	52	-47	36	-89	69	-93	73	-55	55
6	**B2AP05-6**	**LLVF**	-33	39	-36	38	-53	57	-80	86	-81	87	-38	-63	-103	-15	-99	3	-8	8
		KKT	45	137	-19	72	0	102	-14	89	-28	59	-39	35	-75	62	-78	65	-46	46
7	**B2AP05-7**	**LLVF**	-37	37	-37	37	-59	59	-88	88	-89	89	84	-84	-42	42	-82	82	-13	13
		KKT	-108	108	-59	59	-77	77	-69	69	-49	49	-45	45	-84	84	-87	87	-52	52
8	**B2AP05-8**	**LLVF**	-46	46	-47	47	-73	73	-96	96	-97	97	78	-78	-48	48	-79	79	-25	25
		KKT	-48	48	-53	53	-57	57	-58	58	-51	51	-53	53	-111	111	-115	115	-69	69
9	**B2AP05-9**	**LLVF**	-52	52	-52	52	-84	84	-125	125	-125	125	56	-56	-69	69	-98	98	-39	39
		KKT	3	-3	-46	46	-36	36	-47	47	-52	52	-65	65	-132	132	-137	137	-78	78
10	**B2AP05-10**	**LLVF**	-47	47	-46	46	-65	65	-98	98	-99	99	85	-85	-86	86	-105	105	-27	27
		KKT	-241	241	-100	100	-150	150	-123	123	-73	73	-53	53	-107	107	-111	111	-65	65
11	**B2AP05-11**	**LLVF**	-49	49	-50	50	-92	92	-106	106	-107	107	51	-51	-41	41	-78	78	-28	28
		KKT	144	-144	-11	11	31	-31	108	-108	-34	34	-58	58	-108	108	-113	113	-68	68
12	**B2AP05-12**	**LLVF**	-47	47	-45	45	-58	58	-86	86	-88	88	18	-18	-76	76	-83	83	-38	38
		KKT	-135	135	-75	75	-99	99	-88	88	-62	62	-51	51	-88	88	-94	94	-61	61
13	**B2AP05-13**	**LLVF**	-39	47	-39	46	-59	65	-89	98	-90	99	-4	-82	-100	-21	-112	-1	-48	-7
		KKT	-12	241	-37	100	-32	150	-38	123	-38	73	-47	45	-88	80	-92	84	-54	49
14	**B2AP05-14**	**LLVF**	-40	42	-43	40	-57	52	-92	84	-94	86	22	-97	-122	-29	-130	-5	-51	-8
		KKT	45	86	-24	57	-5	72	-19	67	-34	49	-53	40	-102	78	-106	82	-61	47
15	**B2AP05-15**	**LLVF**	-52	52	-52	52	-71	71	-115	115	-116	116	43	-43	-81	81	-100	100	-44	44
		KKT	-34	34	-55	55	-54	54	-59	59	-55	55	-60	60	-128	128	-133	133	-77	77
16	**B2AP05-16**	**LLVF**	-31	31	-33	33	-70	70	-93	93	-94	94	129	-129	-31	31	-90	90	-3	3
		KKT	-21	21	-36	36	-35	35	-40	40	-35	35	-43	43	-91	91	-95	95	-51	51
17	**B2AP05-17**	**LLVF**	-49	49	-50	50	-68	68	-100	100	-102	102	102	-102	-25	25	-87	87	-23	23

			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
17	**B2AP05-17**	**KKT**	-52	52	-55	55	-58	58	-59	59	-53	53	-60	60	-118	118	-125	125	-71	71
18	**B2AP05-18**	**LLVF**	-37	37	-37	37	-60	60	-90	90	-91	91	91	-91	-24	24	-70	70	-6	6
18	**B2AP05-18**	**KKT**	-7	7	-34	34	-30	30	-36	36	-37	37	-44	44	-88	88	-95	95	-55	55
19	**B2AP05-19**	**LLVF**	-52	52	-51	51	-78	78	-103	103	-104	104	12	-12	-95	95	-101	101	-49	49
		KKT	-128	128	-79	79	-104	104	-93	93	-68	68	-61	61	-109	109	-113	113	-70	70
20	**B2AP05-20**	**LLVF**	-38	38	-38	38	-64	64	-93	93	-94	94	65	-65	-52	52	-79	79	-14	14
		KKT	-39	39	-44	44	-46	46	-49	49	-42	42	-43	43	-90	90	-94	94	-52	52
21	**B2AP05-21**	**LLVF**	-39	39	-40	40	-63	63	-92	92	-93	93	80	-80	-33	33	-77	77	-9	9
		KKT	15	-15	-30	30	-19	19	-29	29	-34	34	-44	44	-96	96	-101	101	-54	54
22	**B2AP05-22**	**LLVF**	-56	56	-56	56	-74	74	-108	108	-109	109	63	-63	-82	82	-105	105	-31	31
		KKT	-35	35	-54	54	-53	53	-58	58	-54	54	-58	58	-120	120	-125	125	-73	73
23	**B2AP05-23**	**LLVF**	-42	60	-44	57	-68	71	-102	104	-102	104	-6	-32	-95	35	-101	57	-51	28
		KKT	-14	54	-42	60	-37	63	-44	65	-44	57	-50	55	-102	97	-107	101	-64	65
24	**B2AP05-24**	**LLVF**	-44	53	-43	53	-68	73	-102	107	-103	108	-25	-85	-110	-25	-123	-13	-59	-3
		KKT	34	-7	-29	40	-11	29	-26	40	-38	45	-53	46	-104	93	-108	98	-63	56
25	**B2AP05-25**	**LLVF**	-48	48	-45	45	-63	63	-82	82	-84	84	62	-62	-84	84	-117	117	-20	20
		KKT	-289	289	-119	119	-190	190	-154	154	-87	87	-51	51	-92	92	-96	96	-56	56
26	**BK5010W02KNP**	**LLVF**	2457	-2457	2458	-2458	2477	-2477	1509	-1509	-6773	6773	-34040	34040	-39428	39428	-38200	38200	-126	126
		KKT	2447	-2447	2420	-2420	2314	-2314	826	-826	-2575	2575	-34903	34903	-36196	36196	-36084	36084	-678	678
27	**BK5010W03KNP**	**LLVF**	2005	-2005	2009	-2009	2062	-2062	1050	-1050	-23788	23788	-16253	16253	-26129	26129	-25985	25985	469	-469
		KKT	2004	-2004	1988	-1988	1931	-1931	203	-203	-893	893	-19756	19756	-21845	21845	-22088	22088	687	-687
28	**BK5010W04KNP**	**LLVF**	2754	-2754	2759	-2759	2826	-2826	1511	-1511	-2020	2020	-25969	25969	-31992	31992	-30462	30462	1080	-1080
		KKT	2749	-2749	2738	-2738	2687	-2687	1194	-1194	57	-57	-23956	23956	-24730	24730	-24845	24845	790	-790
29	**BK5010W05KNP**	**LLVF**	3443	-2319	3446	-2321	3496	-2347	1350	-821	-8975	5874	-36082	16497	-41094	21121	-38753	20200	1158	-1158
		KKT	3431	-2311	3421	-2296	3376	-2234	1911	-399	-427	300	-25705	13407	-26480	15832	-26733	16181	1151	-1151
30	**BK5010W06KNP**	**LLVF**	2318	-2318	2321	-2321	2359	-2359	704	-704	-5149	5149	-34304	34304	-40737	40737	-39110	39110	-174	174
		KKT	2313	-2313	2303	-2303	2256	-2256	1458	-1458	-1029	1029	-33235	33235	-34035	34035	-33911	33911	-509	509
31	**BK5010W07KNP**	**LLVF**	2506	-2506	2508	-2508	2534	-2534	1514	-1514	-4697	4697	-25324	25324	-32947	32947	-31545	31545	657	-657
		KKT	2502	-2502	2485	-2485	2420	-2420	580	-580	-982	982	-28038	28038	-29222	29222	-29136	29136	99	-99
32	**BK5010W08KNP**	**LLVF**	2498	-2498	2501	-2501	2553	-2553	1120	-1120	-4739	4739	-47226	47226	-51167	51167	-47365	47365	-741	741
		KKT	2490	-2490	2481	-2481	2470	-2470	2154	-2154	-147	147	-40567	40567	-39028	39028	-38788	38788	-866	866
33	**BK5010W09KNP**	**LLVF**	1925	-1925	1929	-1929	1982	-1982	444	-444	-2476	2476	-20745	20745	-29234	29234	-29066	29066	79	-79
		KKT	1921	-1921	1901	-1901	1866	-1866	148	-148	-2314	2314	-24576	24576	-26437	26437	-26534	26534	-935	935
34	**BK5010W10KNP**	**LLVF**	2362	-2362	2366	-2366	2415	-2415	151	-151	-1655	1655	-20310	20310	-27871	27871	-27725	27725	566	-566
		KKT	2349	-2349	2331	-2331	2208	-2208	130	-130	-2628	2628	-22932	22932	-25175	25175	-25356	25356	-647	647
35	**BK5010W11KNP**	**LLVF**	2692	-2692	2695	-2695	2737	-2737	2274	-2274	151	-151	-36820	36820	-43110	43110	-39824	39824	931	-931

			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
35	**BK5010W11KNP**	**KKT**	2694	-2694	2694	-2694	2703	-2703	2744	-2744	1345	-1345	-30859	30859	-30593	30593	-29560	29560	451	-451
36	**BK5010W12KNP**	**LLVF**	2798	-2798	2804	-2804	2885	-2885	1850	-1850	-284	284	-30622	30622	-38747	38747	-36862	36862	733	-733
		KKT	2797	-2797	2791	-2791	2770	-2770	2151	-2151	-389	389	-30001	30001	-29471	29471	-29347	29347	337	-337
37	**BK5010W13KNP**	**LLVF**	2958	-2958	2965	-2965	3050	-3050	2228	-2228	85	-85	-9261	9261	-18389	18389	-18345	18345	1945	-1945
		KKT	2962	-2962	2956	-2956	2922	-2922	2282	-2282	-257	257	-12023	12023	-13075	13075	-13307	13307	2256	-2256
38	**BK5010W14KNP**	**LLVF**	1762	-1762	1763	-1763	1781	-1781	831	-831	-4512	4512	-34120	34120	-41377	41377	-39870	39870	-939	939
		KKT	1757	-1757	1744	-1744	1693	-1693	1219	-1219	-1083	1083	-37898	37898	-38132	38132	-37987	37987	-1433	1433
39	**BK5010W15KNP**	**LLVF**	2952	-2952	2954	-2954	2982	-2982	1991	-1991	-4130	4130	-42813	42813	-46348	46348	-43286	43286	20	-20
		KKT	2947	-2947	2935	-2935	2861	-2861	454	-454	-842	842	-38422	38422	-38150	38150	-37958	37958	-162	162
40	**BK5010W16KNP**	**LLVF**	2719	-2719	2721	-2721	2755	-2755	236	-236	-2389	2389	-26362	26362	-33198	33198	-31971	31971	767	-767
		KKT	2710	-2710	2677	-2677	2618	-2618	-97	97	-1569	1569	-26062	26062	-27777	27777	-27793	27793	245	-245
41	**BK5010W17KNP**	**LLVF**	2947	-2947	2949	-2949	2977	-2977	1220	-1220	-8750	8750	-34210	34210	-39064	39064	-37224	37224	599	-599
		KKT	2936	-2936	2906	-2906	2792	-2792	1250	-1250	-1683	1683	-31591	31591	-32540	32540	-32446	32446	187	-187
42	**BK5010W18KNP**	**LLVF**	2745	-2745	2749	-2749	2803	-2803	1199	-1199	-3019	3019	-25729	25729	-33197	33197	-31552	31552	1017	-1017
		KKT	2740	-2740	2734	-2734	2701	-2701	1650	-1650	445	-445	-20975	20975	-22568	22568	-22421	22421	1470	-1470
43	**BK5010W19KNP**	**LLVF**	3185	-3185	3188	-3188	3228	-3228	1793	-1793	-2825	2825	-34271	34271	-38467	38467	-36029	36029	1384	-1384
		KKT	3180	-3180	3170	-3170	3119	-3119	1748	-1748	354	-354	-28176	28176	-29045	29045	-28829	28829	1027	-1027
44	**BK5010W20KNP**	**LLVF**	3082	-3082	3086	-3086	3146	-3146	1617	-1617	-3604	3604	-38049	38049	-42471	42471	-39786	39786	606	-606
		KKT	3081	-3081	3068	-3068	3002	-3002	643	-643	-638	638	-33355	33355	-33768	33768	-33539	33539	352	-352
45	**BK5020W01KNP**	**LLVF**	5560	-5560	5567	-5567	5667	-5667	4663	-4663	-11012	11012	-30035	30035	-33852	33852	-32832	32832	3217	-3217
		KKT	5546	-5546	5487	-5487	5304	-5304	3453	-3453	-4988	4988	-26526	26526	-27291	27291	-27264	27264	3183	-3183
46	**BK5020W02KNP**	**LLVF**	5265	-5265	5269	-5269	5325	-5325	3831	-3831	-4567	4567	-36370	36370	-39364	39364	-37502	37502	2824	-2824
		KKT	5256	-5256	5242	-5242	5045	-5045	4007	-4007	-2676	2676	-31594	31594	-32213	32213	-32056	32056	2418	-2418
47	**BK5020W03KNP**	**LLVF**	4688	-4688	4693	-4693	4767	-4767	1696	-1696	-642	642	-26333	26333	-31724	31724	-30511	30511	2709	-2709
		KKT	4679	-4679	4666	-4666	4584	-4584	2506	-2506	-2127	2127	-24994	24994	-25700	25700	-25763	25763	2311	-2311
48	**BK5020W04KNP**	**LLVF**	4872	-4872	4879	-4879	4977	-4977	3962	-3962	-10548	10548	-28938	28938	-33538	33538	-32577	32577	2435	-2435
		KKT	4859	-4859	4807	-4807	4618	-4618	2665	-2665	-3706	3706	-26869	26869	-27517	27517	-27495	27495	2125	-2125
49	**BK5020W05KNP**	**LLVF**	5259	-5259	5262	-5262	5317	-5317	2464	-2464	-10215	10215	-45227	45227	-47092	47092	-44193	44193	2389	-2389
		KKT	5243	-5243	5235	-5235	5096	-5096	3574	-3574	-725	725	-35903	35903	-35911	35911	-35624	35624	2263	-2263
50	**BK5020W06KNP**	**LLVF**	4107	-4107	4111	-4111	4173	-4173	3342	-3342	-8246	8246	-26940	26940	-32157	32157	-31406	31406	1966	-1966
		KKT	4097	-4097	4065	-4065	3950	-3950	2578	-2578	-2560	2560	-26242	26242	-27577	27577	-27535	27535	1381	-1381
51	**BK5020W07KNP**	**LLVF**	5669	-5669	5674	-5674	5740	-5740	4802	-4802	-14337	14337	-19139	19139	-23781	23781	-23405	23405	3984	-3984
		KKT	5650	-5650	5627	-5627	5345	-5345	1811	-1811	-4489	4489	-18447	18447	-19695	19695	-19747	19747	4016	-4016
52	**BK5020W08KNP**	**LLVF**	5146	-5146	5150	-5150	5217	-5217	2224	-2224	-4428	4428	-45315	45315	-48468	48468	-45470	45470	2154	-2154
		KKT	5141	-5141	5122	-5122	5038	-5038	4370	-4370	-848	848	-36932	36932	-36781	36781	-36448	36448	2258	-2258
53	**BK5020W09KNP**	**LLVF**	4559	-4559	4564	-4564	4641	-4641	3628	-3628	-101	101	-20111	20111	-26648	26648	-26033	26033	2830	-2830

			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
53	**BK5020W09KNP**	**KKT**	4550	-4550	4518	-4518	4478	-4478	2629	-2629	-2112	2112	-19456	19456	-20840	20840	-20883	20883	2556	-2556
54	**BK5020W10KNP**	**LLVF**	5487	-5487	5492	-5492	5558	-5558	5082	-5082	-82	82	-27954	27954	-32746	32746	-31636	31636	3119	-3119
		KKT	5488	-5488	5447	-5447	5276	-5276	1430	-1430	-4552	4552	-25999	25999	-27197	27197	-27122	27122	3237	-3237
55	**BK5020W11KNP**	**LLVF**	4351	-4351	4355	-4355	4422	-4422	3183	-3183	-6080	6080	-24605	24605	-31105	31105	-29893	29893	2586	-2586
		KKT	4344	-4344	4320	-4320	4230	-4230	3663	-3663	-485	485	-24174	24174	-24916	24916	-24804	24804	2167	-2167
56	**BK5020W12KNP**	**LLVF**	5740	-5740	5748	-5748	5849	-5849	5605	-5605	-2513	2513	-17638	17638	-23740	23740	-23024	23024	4252	-4252
		KKT	5730	-5730	5705	-5705	5611	-5611	4071	-4071	-1324	1324	-15549	15549	-16406	16406	-16426	16426	4510	-4510
57	**BK5020W13KNP**	**LLVF**	5899	-5899	5909	-5909	6042	-6042	4473	-4473	-4047	4047	-21434	21434	-27249	27249	-26144	26144	4184	-4184
		KKT	5886	-5886	5860	-5860	5763	-5763	4862	-4862	-2330	2330	-18319	18319	-18766	18766	-18812	18812	4473	-4473
58	**BK5020W14KNP**	**LLVF**	4579	-4579	4582	-4582	4632	-4632	2043	-2043	-3804	3804	-37741	37741	-41822	41822	-40125	40125	1653	-1653
		KKT	4571	-4571	4536	-4536	4367	-4367	3246	-3246	-2572	2572	-35913	35913	-35740	35740	-35551	35551	1280	-1280
59	**BK5020W15KNP**	**LLVF**	5421	-5421	5426	-5426	5494	-5494	3910	-3910	-12330	12330	-25687	25687	-30288	30288	-29336	29336	3421	-3421
		KKT	5408	-5408	5360	-5360	5183	-5183	2148	-2148	-2952	2952	-23683	23683	-24459	24459	-24415	24415	3077	-3077
60	**BK5020W16KNP**	**LLVF**	5093	-5093	5098	-5098	5161	-5161	4302	-4302	-2077	2077	-24125	24125	-29163	29163	-28323	28323	3227	-3227
		KKT	5079	-5079	5057	-5057	4923	-4923	3229	-3229	-2086	2086	-21543	21543	-22982	22982	-23005	23005	2968	-2968
61	**BK5020W17KNP**	**LLVF**	6849	-6849	6873	-6873	6978	-6978	7428	-7428	3775	-3775	-14465	14465	-20390	20390	-20324	20324	6862	-6862
		KKT	4841	-4841	5864	-5864	6009	-6009	2278	-2278	4646	-4646	-13191	13191	-13036	13036	-12699	12699	7128	-7128
62	**BK5020W18KNP**	**LLVF**	4942	-4942	4947	-4947	5017	-5017	3812	-3812	-4265	4265	-24641	24641	-30206	30206	-28987	28987	3286	-3286
		KKT	4934	-4934	4912	-4912	4830	-4830	3892	-3892	-79	79	-21638	21638	-22523	22523	-22393	22393	3107	-3107
63	**BK5020W19KNP**	**LLVF**	6547	-6547	6551	-6551	6612	-6612	5103	-5103	-1410	1410	-35173	35173	-38691	38691	-35868	35868	4526	-4526
		KKT	6550	-6550	6507	-6507	6328	-6328	4362	-4362	-199	199	-25171	25171	-26161	26161	-25877	25877	4687	-4687
64	**BK5020W20KNP**	**LLVF**	4885	-4885	4890	-4890	4965	-4965	3856	-3856	-4405	4405	-33780	33780	-38134	38134	-36751	36751	2222	-2222
		KKT	4871	-4871	4851	-4851	4740	-4740	2988	-2988	-2984	2984	-30441	30441	-31197	31197	-31052	31052	2002	-2002
65	**BK5030W01KNP**	**LLVF**	7940	-7940	7948	-7948	8068	-8068	6951	-6951	-15198	15198	-26445	26445	-29990	29990	-29022	29022	5701	-5701
		KKT	7922	-7922	7851	-7851	7604	-7604	5573	-5573	-4975	4975	-21552	21552	-22308	22308	-22268	22268	5964	-5964
66	**BK5030W02KNP**	**LLVF**	7507	-7507	7513	-7513	7602	-7602	6819	-6819	-12542	12542	-29634	29634	-32701	32701	-31399	31399	5309	-5309
		KKT	7494	-7494	7458	-7458	7333	-7333	6157	-6157	-2811	2811	-24913	24913	-25564	25564	-25407	25407	5254	-5254
67	**BK5030W03KNP**	**LLVF**	6901	-6901	6906	-6906	6980	-6980	6114	-6114	-1220	1220	-31665	31665	-35313	35313	-34008	34008	4440	-4440
		KKT	6883	-6883	6858	-6858	6699	-6699	4921	-4921	-3415	3415	-27993	27993	-28607	28607	-28496	28496	4168	-4168
68	**BK5030W04KNP**	**LLVF**	8102	-8102	8111	-8111	8233	-8233	7707	-7707	1166	-1166	-27272	27272	-30478	30478	-29616	29616	5784	-5784
		KKT	8084	-8084	8047	-8047	7551	-7551	4295	-4295	-7569	7569	-22749	22749	-23545	23545	-23507	23507	5796	-5796
69	**BK5030W05KNP**	**LLVF**	8082	-8082	8086	-8086	8150	-8150	6229	-6229	-11085	11085	-40250	40250	-42200	42200	-39463	39463	5755	-5755
		KKT	8074	-8074	8063	-8063	7940	-7940	6880	-6880	479	-479	-28986	28986	-29409	29409	-28995	28995	5684	-5684
70	**BK5030W06KNP**	**LLVF**	6053	-6053	6058	-6058	6142	-6142	5719	-5719	-10419	10419	-28308	28308	-32322	32322	-31439	31439	3714	-3714
		KKT	6043	-6043	6028	-6028	5860	-5860	5157	-5157	-3266	3266	-25350	25350	-26385	26385	-26279	26279	3554	-3554
71	**BK5030W07KNP**	**LLVF**	7170	-7170	7175	-7175	7254	-7254	6413	-6413	2944	-2944	-20976	20976	-24882	24882	-24281	24281	5290	-5290

			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
71	**BK5030W07KNP**	**KKT**	7147	-7147	7114	-7114	6907	-6907	4999	-4999	-4909	4909	-18733	18733	-19891	19891	-19912	19912	5386	-5386
72	**BK5030W08KNP**	**LLVF**	8110	-8110	8116	-8116	8217	-8217	5525	-5525	-9915	9915	-42358	42358	-44095	44095	-41596	41596	5418	-5418
		KKT	8111	-8111	8095	-8095	7974	-7974	7780	-7780	-962	962	-32346	32346	-32381	32381	-31967	31967	5422	-5422
73	**BK5030W09KNP**	**LLVF**	6904	-6904	6910	-6910	6998	-6998	6195	-6195	-13675	13675	-22053	22053	-26580	26580	-25908	25908	4999	-4999
		KKT	6888	-6888	6867	-6867	6614	-6614	4801	-4801	-3095	3095	-18668	18668	-19871	19871	-19874	19874	5192	-5192
74	**BK5030W10KNP**	**LLVF**	7877	-7877	7884	-7884	7981	-7981	7182	-7182	1261	-1261	-29650	29650	-32816	32816	-31877	31877	5220	-5220
		KKT	7873	-7873	7808	-7808	7520	-7520	2770	-2770	-10106	10106	-27003	27003	-27699	27699	-27615	27615	4988	-4988
75	**BK5030W11KNP**	**LLVF**	5910	-5910	5916	-5916	5999	-5999	3883	-3883	-10504	10504	-25698	25698	-30666	30666	-29744	29744	3725	-3725
		KKT	5897	-5897	5879	-5879	5666	-5666	4718	-4718	-2562	2562	-24048	24048	-24886	24886	-24801	24801	3466	-3466
76	**BK5030W12KNP**	**LLVF**	8221	-8221	8230	-8230	8352	-8352	7686	-7686	4709	-4709	-21010	21010	-25475	25475	-24493	24493	6331	-6331
		KKT	8203	-8203	8178	-8178	8033	-8033	5471	-5471	-4115	4115	-17137	17137	-17884	17884	-17881	17881	6550	-6550
77	**BK5030W13KNP**	**LLVF**	8380	-8380	8390	-8390	8526	-8526	7364	-7364	4498	-4498	-24104	24104	-28313	28313	-27164	27164	6336	-6336
		KKT	8378	-8378	8346	-8346	8026	-8026	5917	-5917	-4841	4841	-19521	19521	-20082	20082	-20052	20052	6539	-6539
78	**BK5030W14KNP**	**LLVF**	6558	-6558	6564	-6564	6647	-6647	4782	-4782	-526	526	-33525	33525	-37252	37252	-36067	36067	3653	-3653
		KKT	6540	-6540	6515	-6515	6144	-6144	4550	-4550	-5679	5679	-31888	31888	-31873	31873	-31715	31715	3320	-3320
79	**BK5030W15KNP**	**LLVF**	7595	-7595	7601	-7601	7684	-7684	5643	-5643	1355	-1355	-28870	28870	-32299	32299	-31044	31044	5296	-5296
		KKT	7578	-7578	7550	-7550	7153	-7153	5674	-5674	-4355	4355	-24167	24167	-24778	24778	-24688	24688	5158	-5158
80	**BK5030W16KNP**	**LLVF**	6798	-6798	6805	-6805	6905	-6905	6044	-6044	957	-957	-25571	25571	-29711	29711	-29017	29017	4468	-4468
		KKT	6774	-6774	6738	-6738	6516	-6516	4223	-4223	-5309	5309	-23434	23434	-24319	24319	-24294	24294	4066	-4066
81	**BK5030W17KNP**	**LLVF**	7932	-7932	7936	-7936	8003	-8003	7032	-7032	-19368	19368	-28058	28058	-31277	31277	-30064	30064	5749	-5749
		KKT	7918	-7918	7899	-7899	7682	-7682	4901	-4901	-2224	2224	-22725	22725	-23579	23579	-23412	23412	5733	-5733
82	**BK5030W18KNP**	**LLVF**	6744	-6744	6750	-6750	6849	-6849	6315	-6315	-6560	6560	-25560	25560	-30301	30301	-29230	29230	4876	-4876
		KKT	6733	-6733	6721	-6721	6576	-6576	5993	-5993	-1131	1131	-22055	22055	-22694	22694	-22541	22541	4689	-4689
83	**BK5030W19KNP**	**LLVF**	8665	-8665	8671	-8671	8771	-8771	7337	-7337	-10640	10640	-35810	35810	-38329	38329	-36204	36204	6274	-6274
		KKT	8645	-8645	8605	-8605	8465	-8465	7131	-7131	-1299	1299	-26901	26901	-27196	27196	-26893	26893	6362	-6362
84	**BK5030W20KNP**	**LLVF**	7046	-7046	7051	-7051	7134	-7134	6224	-6224	-1617	1617	-33025	33025	-36267	36267	-35139	35139	4191	-4191
		KKT	7023	-7023	6991	-6991	6793	-6793	4640	-4640	-5604	5604	-28964	28964	-29620	29620	-29465	29465	4137	-4137
85	**BK5040W01KNP**	**LLVF**	10302	-10302	10311	-10311	10447	-10447	9335	-9335	-17724	17724	-27373	27373	-30091	30091	-28996	28996	7927	-7927
		KKT	10279	-10279	10245	-10245	9869	-9869	8730	-8730	-5926	5926	-21374	21374	-21911	21911	-21816	21816	8132	-8132
86	**BK5040W02KNP**	**LLVF**	10398	-10398	10406	-10406	10524	-10524	10301	-10301	3434	-3434	-25227	25227	-28062	28062	-26918	26918	8343	-8343
		KKT	10388	-10388	10368	-10368	10101	-10101	9330	-9330	-2608	2608	-19822	19822	-20437	20437	-20249	20249	8291	-8291
87	**BK5040W03KNP**	**LLVF**	9055	-9055	9061	-9061	9146	-9146	8805	-8805	-17864	17864	-30959	30959	-33903	33903	-32800	32800	6431	-6431
		KKT	9035	-9035	9008	-9008	8695	-8695	7695	-7695	-5887	5887	-27418	27418	-27880	27880	-27700	27700	6203	-6203
88	**BK5040W04KNP**	**LLVF**	9784	-9784	9795	-9795	9942	-9942	9529	-9529	3289	-3289	-24670	24670	-27777	27777	-26992	26992	7420	-7420
		KKT	9757	-9757	9718	-9718	9163	-9163	7649	-7649	-7936	7936	-20056	20056	-20758	20758	-20717	20717	7570	-7570
89	**BK5040W05KNP**	**LLVF**	9647	-9647	9654	-9654	9762	-9762	9068	-9068	3477	-3477	-27233	27233	-30474	30474	-28897	28897	7592	-7592

			$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
89	**BK5040W05KNP**	**KKT**	9638	-9638	9619	-9619	9356	-9356	8668	-8668	-1558	1558	-20945	20945	-21500	21500	-21289	21289	7641	-7641
90	**BK5040W06KNP**	**LLVF**	8353	-8353	8360	-8360	8465	-8465	8101	-8101	-13102	13102	-25992	25992	-29562	29562	-28736	28736	6158	-6158
		KKT	8333	-8333	8312	-8312	8054	-8054	7176	-7176	-3997	3997	-22571	22571	-23396	23396	-23288	23288	6060	-6060
91	**BK5040W07KNP**	**LLVF**	10091	-10091	10100	-10100	10220	-10220	9205	-9205	6212	-6212	-19586	19586	-22786	22786	-22068	22068	8231	-8231
		KKT	10068	-10068	10029	-10029	9493	-9493	8096	-8096	-5512	5512	-15495	15495	-16311	16311	-16300	16300	8391	-8391
92	**BK5040W08KNP**	**LLVF**	10324	-10324	10329	-10329	10440	-10440	7302	-7302	-14019	14019	-43823	43823	-44474	44474	-41949	41949	7548	-7548
		KKT	10312	-10312	10302	-10302	10205	-10205	10118	-10118	-934	934	-32413	32413	-32180	32180	-31708	31708	7341	-7341
93	**BK5040W09KNP**	**LLVF**	9838	-9838	9846	-9846	9956	-9956	9540	-9540	-15261	15261	-18997	18997	-22480	22480	-21818	21818	8131	-8131
		KKT	9818	-9818	9792	-9792	9487	-9487	7765	-7765	-2571	2571	-14040	14040	-15041	15041	-15033	15033	8399	-8399
94	**BK5040W10KNP**	**LLVF**	11603	-11603	11612	-11612	11738	-11738	10939	-10939	5836	-5836	-26586	26586	-28902	28902	-27824	27824	9025	-9025
		KKT	11560	-11560	11500	-11500	11122	-11122	6376	-6376	-11961	11961	-21794	21794	-22367	22367	-22272	22272	9200	-9200
95	**BK5040W11KNP**	**LLVF**	9419	-9419	9427	-9427	9539	-9539	8755	-8755	2765	-2765	-28204	28204	-31390	31390	-30336	30336	6950	-6950
		KKT	9401	-9401	9368	-9368	8917	-8917	8323	-8323	-5789	5789	-24679	24679	-25127	25127	-24948	24948	6835	-6835
96	**BK5040W12KNP**	**LLVF**	10711	-10711	10721	-10721	10860	-10860	10001	-10001	5799	-5799	-23787	23787	-26992	26992	-25829	25829	8636	-8636
		KKT	10691	-10691	10662	-10662	10256	-10256	8088	-8088	-5339	5339	-17916	17916	-18408	18408	-18315	18315	8735	-8735
97	**BK5040W13KNP**	**LLVF**	10870	-10870	10880	-10880	11034	-11034	10149	-10149	-16929	16929	-25231	25231	-28154	28154	-27075	27075	8684	-8684
		KKT	10854	-10854	10831	-10831	10578	-10578	9227	-9227	-5917	5917	-19029	19029	-19504	19504	-19418	19418	8863	-8863
98	**BK5040W14KNP**	**LLVF**	9048	-9048	9054	-9054	9154	-9154	7916	-7916	972	-972	-32551	32551	-35713	35713	-34539	34539	6336	-6336
		KKT	9027	-9027	8994	-8994	8531	-8531	6902	-6902	-5564	5564	-29845	29845	-29724	29724	-29505	29505	5908	-5908
99	**BK5040W15KNP**	**LLVF**	9874	-9874	9880	-9880	9972	-9972	8514	-8514	2980	-2980	-28285	28285	-31165	31165	-29900	29900	7446	-7446
		KKT	9850	-9850	9816	-9816	9325	-9325	7765	-7765	-4338	4338	-22157	22157	-22796	22796	-22664	22664	7593	-7593
100	**BK5040W16KNP**	**LLVF**	9141	-9141	9150	-9150	9270	-9270	8264	-8264	-23343	23343	-28258	28258	-31134	31134	-30354	30354	6559	-6559
		KKT	9114	-9114	9076	-9076	8644	-8644	7371	-7371	-9075	9075	-24958	24958	-25435	25435	-25346	25346	6292	-6292
101	**BK5040W17KNP**	**LLVF**	9926	-9926	9930	-9930	9999	-9999	9544	-9544	923	-923	-28701	28701	-31346	31346	-30204	30204	7512	-7512
		KKT	9903	-9903	9866	-9866	9640	-9640	7788	-7788	-5286	5286	-23414	23414	-24217	24217	-24043	24043	7455	-7455
102	**BK5040W18KNP**	**LLVF**	9023	-9023	9030	-9030	9131	-9131	9113	-9113	-7566	7566	-22612	22612	-26933	26933	-25952	25952	7215	-7215
		KKT	9013	-9013	8999	-8999	8839	-8839	8315	-8315	-432	432	-18710	18710	-19330	19330	-19142	19142	7085	-7085
103	**BK5040W19KNP**	**LLVF**	10883	-10883	10891	-10891	11020	-11020	9885	-9885	-15655	15655	-28746	28746	-31654	31654	-30174	30174	8701	-8701
		KKT	10862	-10862	10844	-10844	10614	-10614	9180	-9180	-2761	2761	-21094	21094	-21499	21499	-21289	21289	8782	-8782
104	**BK5040W20KNP**	**LLVF**	9035	-9035	9042	-9042	9141	-9141	8469	-8469	-21209	21209	-28251	28251	-31235	31235	-30416	30416	6412	-6412
		KKT	9010	-9010	8977	-8977	8595	-8595	7394	-7394	-6929	6929	-23643	23643	-24454	24454	-24341	24341	6572	-6572
105	**BK5050W01KNP**	**LLVF**	12205	-12205	12214	-12214	12357	-12357	11479	-11479	6158	-6158	-22196	22196	-25054	25054	-24168	24168	10051	-10051
		KKT	12181	-12181	12136	-12136	11539	-11539	10063	-10063	-5284	5284	-16386	16386	-17077	17077	-17012	17012	10264	-10264
106	**BK5050W02KNP**	**LLVF**	12697	-12697	12706	-12706	12838	-12838	12618	-12618	4733	-4733	-24752	24752	-26956	26956	-25939	25939	10518	-10518
		KKT	12677	-12677	12649	-12649	12247	-12247	11313	-11313	-4773	4773	-18593	18593	-19198	19198	-19019	19019	10487	-10487
107	**BK5050W03KNP**	**LLVF**	11624	-11624	11630	-11630	11725	-11725	11650	-11650	1802	-1802	-30290	30290	-33068	33068	-31880	31880	8924	-8924

		$\lambda = 10^5$		$\lambda = 10^4$		$\lambda = 10^3$		$\lambda = 10^2$		$\lambda = 10^1$		$\lambda = 10^0$		$\lambda = 10^{-1}$		$\lambda = 10^{-2}$		λ_{var}	
107 **BK5050W03KNP**	**KKT**	11596	-11596	11556	-11556	10991	-10991	9372	-9372	-6489	6489	-26310	26310	-26688	26688	-26441	26441	8702	-8702
108 **BK5050W04KNP**	**LLVF**	11146	-11146	11157	-11157	11320	-11320	11057	-11057	4526	-4526	-22751	22751	-25759	25759	-25112	25112	8878	-8878
	KKT	11120	-11120	11084	-11084	10569	-10569	9331	-9331	-7397	7397	-17987	17987	-18679	18679	-18624	18624	8955	-8955
109 **BK5050W05KNP**	**LLVF**	11490	-11490	11497	-11497	11611	-11611	10979	-10979	4616	-4616	-27336	27336	-29720	29720	-28353	28353	9190	-9190
	KKT	11472	-11472	11446	-11446	11080	-11080	10093	-10093	-4014	4014	-20453	20453	-21030	21030	-20834	20834	9284	-9284
110 **BK5050W06KNP**	**LLVF**	10870	-10870	10879	-10879	11007	-11007	10691	-10691	-14230	14230	-23601	23601	-26687	26687	-25850	25850	8649	-8649
	KKT	10852	-10852	10826	-10826	10543	-10543	9731	-9731	-3966	3966	-18432	18432	-19180	19180	-19067	19067	8872	-8872
111 **BK5050W07KNP**	**LLVF**	12209	-12209	12219	-12219	12363	-12363	11813	-11813	7028	-7028	-22318	22318	-24939	24939	-24063	24063	10138	-10138
	KKT	12187	-12187	12151	-12151	11647	-11647	10513	-10513	-7063	7063	-17153	17153	-17738	17738	-17621	17621	10151	-10151
112 **BK5050W08KNP**	**LLVF**	13200	-13200	13207	-13207	13351	-13351	10928	-10928	1027	-1027	-41503	41503	-41958	41958	-39608	39608	10595	-10595
	KKT	13212	-13212	13200	-13200	13037	-13037	13103	-13103	-1416	1416	-29265	29265	-29124	29124	-28613	28613	10404	-10404
113 **BK5050W09KNP**	**LLVF**	12439	-12439	12447	-12447	12560	-12560	12346	-12346	-19639	19639	-21002	21002	-23752	23752	-22939	22939	10571	-10571
	KKT	12410	-12410	12377	-12377	11982	-11982	9949	-9949	-3336	3336	-15221	15221	-16064	16064	-15951	15951	10600	-10600
114 **BK5050W10KNP**	**LLVF**	14008	-14008	14019	-14019	14174	-14174	13343	-13343	-26821	26821	-22534	22534	-24439	24439	-23605	23605	11714	-11714
	KKT	13964	-13964	13905	-13905	13235	-13235	10518	-10518	-12587	12587	-17198	17198	-17762	17762	-17696	17696	11837	-11837
115 **BK5050W11KNP**	**LLVF**	11570	-11570	11579	-11579	11703	-11703	11181	-11181	-21220	21220	-26023	26023	-28749	28749	-27867	27867	9043	-9043
	KKT	11544	-11544	11510	-11510	11121	-11121	10108	-10108	-8507	8507	-22620	22620	-23022	23022	-22838	22838	8973	-8973
116 **BK5050W12KNP**	**LLVF**	13096	-13096	13107	-13107	13270	-13270	12411	-12411	7559	-7559	-24064	24064	-26523	26523	-25395	25395	10851	-10851
	KKT	13072	-13072	13038	-13038	12555	-12555	10423	-10423	-7871	7871	-17730	17730	-18089	18089	-17956	17956	10870	-10870
117 **BK5050W13KNP**	**LLVF**	13255	-13255	13266	-13266	13430	-13430	12383	-12383	7407	-7407	-25623	25623	-28069	28069	-26925	26925	10915	-10915
	KKT	13238	-13238	13198	-13198	12686	-12686	11656	-11656	-10329	10329	-19659	19659	-19989	19989	-19854	19854	10866	-10866
118 **BK5050W14KNP**	**LLVF**	11686	-11686	11694	-11694	11825	-11825	10989	-10989	-21346	21346	-28335	28335	-31265	31265	-30212	30212	9131	-9131
	KKT	11660	-11660	11631	-11631	11286	-11286	10316	-10316	-5715	5715	-24202	24202	-24312	24312	-24078	24078	8925	-8925
119 **BK5050W15KNP**	**LLVF**	11659	-11659	11665	-11665	11771	-11771	10623	-10623	-21291	21291	-25796	25796	-28331	28331	-27302	27302	9325	-9325
	KKT	11632	-11632	11596	-11596	11193	-11193	10049	-10049	-4094	4094	-19677	19677	-20320	20320	-20188	20188	9432	-9432
120 **BK5050W16KNP**	**LLVF**	12079	-12079	12088	-12088	12224	-12224	11374	-11374	4649	-4649	-26414	26414	-29007	29007	-28113	28113	9397	-9397
	KKT	12047	-12047	11991	-11991	11228	-11228	9378	-9378	-9383	9383	-22010	22010	-22484	22484	-22361	22361	9461	-9461
121 **BK5050W17KNP**	**LLVF**	12276	-12276	12281	-12281	12363	-12363	12254	-12254	-26743	26743	-26764	26764	-28950	28950	-27933	27933	9982	-9982
	KKT	12245	-12245	12209	-12209	11790	-11790	9576	-9576	-4807	4807	-20579	20579	-21401	21401	-21187	21187	9927	-9927
122 **BK5050W18KNP**	**LLVF**	12209	-12209	12217	-12217	12341	-12341	12394	-12394	-10579	10579	-20683	20683	-24403	24403	-23393	23393	10527	-10527
	KKT	12193	-12193	12173	-12173	11945	-11945	11355	-11355	-702	702	-15272	15272	-15848	15848	-15642	15642	10384	-10384
123 **BK5050W19KNP**	**LLVF**	13708	-13708	13718	-13718	13862	-13862	12983	-12983	7197	-7197	-23447	23447	-26301	26301	-25023	25023	11734	-11734
	KKT	13687	-13687	13658	-13658	13239	-13239	12331	-12331	-4258	4258	-16395	16395	-16853	16853	-16665	16665	11680	-11680
124 **BK5050W20KNP**	**LLVF**	11789	-11789	11796	-11796	11908	-11908	11076	-11076	3075	-3075	-29470	29470	-31584	31584	-30542	30542	9033	-9033
	KKT	11754	-11754	11705	-11705	11000	-11000	10291	-10291	-8561	8561	-23546	23546	-24101	24101	-23930	23930	9175	-9175

Table 9: Upper-level and lower-level objective values at the solution point of algorithms **??** and 2.6 for relaxed binary examples from [32]

REFERENCES

[1] G.B. Allende and G. Still. *Solving bi-level programs with the KKT-approach*, Mathematical Programming 131:37-48 (2012)

[2] I. K. Argyros and S. Hilout. *Improved generalized differentiability conditions for Newton-like methods*, Journal of Complexity, 26(3): 316-333 (2010)

[3] J.F. Bard. *Practical bilevel optimization: algorithms and applications*. Kluwer Academic Publishers (1998)

[4] J .F. Bard. *Some Properties of the Bilevel Programming Problem*, Jounal of Optimization Theory and Applications 68(2): 371-378 (1991)

[5] R. Behling, A. Fischer, M. Herrich, A. Lusem and Y. Ye. *A Levenberg-Marquardt method with approximate projections*, Journal of Computational and Applied Mathematics 59:5-26 (2014)

[6] R. Behling, A. Fischer, G. Haeser, A. Ramos and K. Schönefeld. *On the constrained error bound condition and the projected Levenberg-Marquardt method*, A Journal of Mathematical Programming and Operations Research (2016)

[7] V. Beiranvand, W.Hare and Y. Lucet. *Best practices for comparing optimization algorithms*, Optimization and Engineering 18: 815-848 (2017)

[8] D.P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*, Academic Press, New York (1982)

[9] W.F. Bialas and M.H. Karwan. *Two-level linear programming*, Management Science 30: 1004-1024 (1984)

[10] J. Bracken and J. McGill. *Mathematical programs with optimization problems in the constraints*, Operational Research 21: 37-44 (1973)

[11] J.V. Burke. *An exact penalization viewpoint of constrained optimization*, SIAM journal on control and optimization 29(4): 968-998 (1991)

[12] Y. Chen and M. Florian. *The nonlinear bilevel programming problem: Formulations, regularity and optimality conditions*, Optimization 32(3): 193-209 (1995)

[13] J. Chen and W. Li. *Convergence of Gauss-Newton's method and uniqueness of the solution*, Applied Mathematics and Computation, 170(1): 686-705 (2005)

[14] X. Chen, L. Qi and D. Sun. *Global and superlinear convergence of the smoothing Newton method and its application to general box constrained variational inequalities*, Mathematics of Computation 67(222): 519-540 (1998)

[15] B. Colson, P. Marcotte and G. Savard. *An overview of bilevel optimization*, Annals of Operational Research 153: 235-256 (2007)

[16] S. Dempe. *A necessary and a sufficient optimality condition for bilevel programming problems*, Optimization 25(4): 341-354 (1992)

[17] S. Dempe. *A bundle algorithm applied to bilevel programming problems with non-unique lower level solutions*, Computational Optimization and Applications, 15(2): 145-166 (2000)

[18] S. Dempe. *Foundations of bilevel programming*. Kluwer Academic Publishers (2002)

[19] S. Dempe and J.Dutta. *Is bilevel programming a special case of mathematical programming with equilibrium constraints?* Mathematical Programming 131: 37-48 (2010)

[20] S. Dempe, J. Dutta and B.S. Mordukhovich. *New necessary optimality conditions in optimistic bilevel programming*, Optimization 56 (5-6): 577-604 (2007)

[21] S. Dempe, and H. Schmidt. *On an algorithm solving two-level programming problems with nonunique lower level solutions*, Computational Optimization and Applications, 6(3): 227-249 (1996)

[22] S. Dempe and A.B. Zemkoho. *KKT Reformulation and Necessary Conditions for Optimality in Nonsmooth Bilevel Optimization*, SIAM Journal on Optimization, 24(4): 1639-1669 (2014)

[23] S. Dempe and A.B. Zemkoho. *The bilevel programming problem: reformulations, constraint qualification and optimality conditions*, Mathematical Programming 138: 447-473 (2013)

[24] S. Dempe and A.B. Zemkoho. *The generalized Mangasarian-Fromowitz constraint qualification and optimality conditions for bilevel programs*, Journal of Optimization Theory and Applications 148(1): 46-68 (2011)

[25] J.E. Dennis and R.B. Schnabel. Numerical methods for unconstrained optimization and nonlinear equations, SIAM Classics in Applied Mathematics, 1996

[26] J. Fan and J. Pan. *A note on the Levenberg-Marquardt parameter*, Applied Mathematics and Computation 207:351-359 (2009)

[27] J.Y. Fan and Y.X. Yuan. *On the quadratic convergence of the Levenberg-Marquardt method without nonsingularity assumption*, Computing 74: 23-39 (2005)

[28] O.P. Ferreira, M.L.N. Goncalves and P.R. Oliveira. *Local convergence analysis of the Gauss-Newton method under a majorant condition*, Journal of Complexity, 27(1): 111-125 (2011)

[29] A. Fischer. *A special Newton-type optimization method*, Optimization 24(3): 269-284 (1992)

[30] A. Fischer and P.K. Shukla. *Levenberg-Marquardt algorithm for unconstrained multicriteria optimization*, Operations Research Letters 36:643-646 (2008)

[31] A. Fischer, A.B. Zemkoho and S. Zhou. *Semismooth Newton-type method for bilevel optimization: global convergence and extensive numerical experiments*, Optimization and Control, arXiv:1912.07079 (2019)

[32] M. Fischetti, I. Ljubič, M. Monaci and M. Sinnl. *Instances and solver software for mixed-integer bilevel linear problems*, msinnl.github.io/pages/bilevel (2017)

[33] R. Fletcher. *An Ideal Penalty Function for Constrained Optimization* Journal Inst. Maths Applies 15: 319-342 (1975)

[34] R. Fletcher. *Generalized inverse methods for the best least squares solution of systems of non-linear equations*, The Computer Journal 10(4): 392-399 (1968)

[35] R. Fletcher. Practical methods of optimization (2nd Ed.), John Wiley (1987)

[Paper 1] J. Fliege, A. Tin and A.B. Zemkoho. *Gauss-Newton-type methods for bilevel optimization* School of Mathematical Sciences, University of Southampton, UK (2020)

[Supp1] J. Fliege, A. Tin, and A.B. Zemkoho. *Supplementary material for "Gauss-Newton-type methods for bilevel optimization"*, Computational Optimization and Applications, doi.org/10.1007/s10589-020-00254-3 (2021)

[36] R.G. Jeroslow. *The Polynomial Hierarchy and a Simple Model for Competitive Analysis*, Mathematical Programming 32: 146-164 (1985)

[37] S.Yu. Gatilov. *Using low-rank approximation of the Jacobian matrix in the Newton-Raphson method to solve certain singular equations*, Journal of Computational and Applied Mathematics, 272: 8-24 (2014)

[38] M.L.N. Goncalves. *Local convergence of the Gauss-Newton method for injective-overdetermined systems of equations under a majorant condition*, Computers & Mathematics with Applications, 66(4): 490-499 (2013)

[39] G.H. Golub and C.F. Van Loan. *Matrix computations*, The John Hopkins University Press (1996)

[40] R.A. Horn and C.R. Johnson. *Matrix Analysis (2nd edition)*, Cambridge University Press: 441-442 (2013)

[41] Y. Jiang, X. Li, C. Huang and X. Wu. *W. Application of particle swarm optimization based on CHKS smoothing function for solving nonlinear bi-level programming problem*, Applied Mathematics and Computation 219: 4332-4339 (2013)

[42] C. Kanzow. *Some noninterior continuation methods for linear complementarity problems*, SIAM Journal on Matrix Analysis and Applications 17(4):851-868 (1996)

[43] C. Kanzow and H. Pieper. *Jacobian smoothing methods for general nonlinear complementarity problems*, SIAM Journal on Optimization 9: 342-372 (1999)

[44] C. Kanzow, N. Yamashita and M. Fukushima, *Levenberg-Marquardt methods with strong local convergence properties for solving nonlinear equations with convex constraints*, Journal of Computational and Applied Mathematics 172: 375-397 (2004)

[45] C.T. Kelley. *Iterative methods for optimization*, Frontiers in Applied Mathematics 18. Philadelphia: SIAM (1999)

[46] P. Kleniati and C.S. Adjiman. *Branch-and-sandwich: a deterministic global optimization algorithm for optimistic bilevel programming problems. Part I: Theoretical development*, Journal of Global Optimization 60(3): 425-458 (2014)

[47] P. Kleniati and C.S. Adjiman. *Branch-and-sandwich: a deterministic global optimization algorithm for optimistic bilevel programming problems. Part II: Convergence analysis and numerical results*, Journal of Global Optimization 60(3): 459-481 (2014)

[48] L. Lampariello and S. Sagratella. *Numerically tractable optimistic bilevel problems*, Computational Optimization and Applications 76(2): 277-303 (2020)

[49] C. Li, Wen-Hong Zhang and Xiao-Qing Jin. *Convergence and uniqueness properties of Gauss-Newton's method*, Computers & Mathematics with Applications, 47(6-7): 1057-1067 (2004)

[50] G.-H. Lin, M. Xu and J.J. Ye. *On solving simple bilevel programs with a nonconvex lower level program*, Mathematical Programming 144(1-2):277-305 (2014)

[51] N. Maratos. *Exact Penalty Function Algorithms for Finite Dimensional and Control Optimization Problems, Thesis* Department of Computing and Control, Imperial College of Science and Technology, University of London (1978)

[52] P. Mehlitz, L.I. Minchenko and A.B. Zemkoho. *A note on partial calmness for bilevel optimization problems with linear structures at the lower level* Optimization letters, doi.org/10.1007/s11590-020-01636-6 (2020)

[53] A. Mitsos, P. Lemonidis and P.I. Barton. *Global solution of bilevel programs with a nonconvex inner program*, Journal of Global Optimization 42(4): 475-513 (2008)

[54] B.S. Mordukhovich. *Variational analysis and generalized differentiation. I: Basic theory. II: Applications*, Berlin: Springer (2006)

[55] J. Nocedal and S.J. Wright. Numerical optimization, Springer (1999)

[56] J.V. Outrata. *Necessary optimality conditions for Stackelberg problems*, Optimization Theory and Applications 76(2): 305-320 (1993)

[57] V. Pan and R. Schreiber. *An improved newton iteration for the generalized inverse of a matrix, with applications*, SIAM Journal on Scientific and Statistical Computing 12(5):1109-1130 (1991)

[58] R. Paulavicius, J. Gao, P. Kleniati and C.S. Adjiman. *BASBL: Branch-and-sandwich bilevel solver. Implementation and computational study with the BASBLib test sets*, Computers & Chemical Engineering 132 (2020)

[59] G.Di Pillo and L. Grippo. *Exact penalty functions in constrained optimization*, SIAM journal on control and optimization 27 (6): 1333-1360 (1989)

[60] S. Pineda, H. Bylling and J.M. Morales. *Efficiently solving linear bilevel programming problems using off-the-shelf optimization software* Optimization and Engineering 19: 187-211 (2018)

[61] M. J. D. Powell. *In Optimization (ed. R. Fletcher)* London: Academic Press, chapter 19 (1969)

[62] W.Press, S.Teukolsky, W. Vetterling and B.Flannery. *Numerical recipes in C*, Cambridge University Press, Cambridge (1992)

[63] R.W. Siregar, Tulus and M. Ramli. *Analysis Local Convergence of Gauss-Newton Method*, et al 2018 IOP Conf. Ser.: Mater. Sci. Eng. 300 012044 (2018)

[64] H.v. Stackelberg, Marktform und Gleichgewicht *English Translated: The Theory of the Market Economy*, Springer, Berlin (1934)

[65] O. Stein and A.Tezel. *The semismooth approach for semi-infinite programming under the Reduction Ansatz*, Journal of Global Optimization, 41(2): 245-266 (2008)

[66] O. Stein and A.Tezel. *The semismooth approach for semi-infinite programming without strict complementarity*, SIAM Journal on Optimization, 20(2): 1052-1072 (2009)

[67] W. Sun and Y.-X. Yuan. Optimization Theory and Methods, Springer, 2006

[68] D. Sun and L. Qi. *On NCP Functions*, Computational Optimization and Applications 13(1-3): 201-220 (1999)

[69] W. Sun and Y.-X. Yuan. *Optimization Theory and Methods*, Springer (2006)

[70] L. Qi and D. Sun. *A survey of some nonsmooth equations and smoothing Newton methods.* In Progress in optimization 30: 121-146, Springer (1999)

[71] L. Qi and D. Sun. *Smoothing Functions and Smoothing Newton Method for Complementarity and Variational Inequality Problems*, Journal of Optimization Theory and Applications, 113: 121-147 (2002)

[72] L. Qi and J. Sun. *A Nonsmooth version of Newton's method*, Mathematical Programming, 58: 353-367 (1993)

[73] W. Wiesemann, A. Tsoukalas, P. Kleniati and B. Rustem. *Pessimistic bilevel optimization*, SIAM Journal on Optimization 23(1): 353-380 (2013)

[74] D.J. White and G. Anandalingam. *A penalty function approach for solving bi-level linear programs* Journal of Global Optimization, 3: 397-419 (1993)

[75] M. Xu and J.J. Ye. *A smoothing augmented lagrangian method for solving simple bilevel programs*, Computational Optimization and Applications 59(1-2): 353-377 (2014)

[76] M. Xu, J.J. Ye and L. Zhang. *Smoothing sqp methods for solving degenerate nonsmooth constrained optimization problems with applications to bilevel programs*, SIAM Journal on Optimization 25(3): 1388-1410 (2015)

[77] Z.K. Xu. *Deriving the properties of linear bilevel programming via a penalty function approach*, Journal of Optimization Theory and Applications, 103: 441-456 (1999)

[78] N. Yamashita and M. Fukushima. *On the rate of convergence of the Levenberg-Marquardt method*, Computing 15: 237-249 (2001)

[79] J.J. Ye and X.Y. Ye. *Necessary optimality conditions for optimization problems with variational inequality constraints*, Mathematics of Operations Research 22(4): 977-997 (1997)

[80] J.J. Ye, D.L. Zhu and Q.J. Zhu. *Exact penalization and necessary optimality conditions for generalized bilevel programming problems*, SIAM Journal on optimization, May 7(2): 481-507 (1997)

[81] J.J. Ye and D.L. Zhu. *Optimality conditions for bilevel programming problems*, Optimization 33: 9-27 (1995)

[82] J.J. Ye and D.L. Zhu. *New necessary optimality conditions for bilevel programs by combining the MPEC and value function approache,*. SIAM Journal on Optimization 20(4): 1885-1905 (2010)

[83] A.B. Zemkoho. *Bilevel programming: reformulations, regularity and stationarity*, PhD thesis, Department of Mathematics and Computer Science, TU Bergakademie Freiberg, Freiberg, Germany (2012)

[Paper 2] A. Tin and A.B. Zemkoho. *Solving bilevel programs with the Levenberg-Marquardt method*, School of Mathematical Sciences, University of Southampton, UK (2020)

[Paper 3] A. Tin and A.B. Zemkoho. *Levenberg-Marquardt method for linear bilevel programming*, School of Mathematical Sciences, University of Southampton, UK (2020)

[Supp2] A. Tin and A.B. Zemkoho. *Supplementary material for "Solving bilevel programs with the Levenberg-Marquardt method"*, School of Mathematical Sciences, University of Southampton, UK (2020)

[Supp3] A. Tin and A.B. Zemkoho. *Supplementary material for "Levenberg-Marquardt method for linear bilevel optimization"*, School of Mathematical Sciences, University of Southampton, UK (2020)

[84] A.B. Zemkoho and S. Zhou. *theoretical and numerical comparison of the karush-kuhn-tucker and value function reformulations in bilevel optimization*, Computational Optimization and Applications doi.org/10.1007/s10589-020-00250-7 (2021)

[85] S. Zhou, A.B. Zemkoho, and A. Tin. *BOLIB: Bilevel Optimization LIBrary of Test Problems*, In: S. Dempe and A.B. Zemkoho (eds.) Bilevel Optimization: Advances and Next Challenges, Springer, Berlin (2020)