Trajectory-Based Methods for Solving Nonlinear and Mixed Integer Nonlinear Programming Problems

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

Terry-Leigh Oliphant

School of Computational and Applied Mathematics
University of the Witwatersrand, Johannesburg.

This dissertation is submitted for the degree of
Doctor of Philosophy

November 2015

I would like to dedicate this thesis to my loving parents and brothers, Michael, Bridgette, Michael and Barry-more, and my precious grandmother Deannah.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Terry-Leigh Oliphant
November 2015

Acknowledgements

I would like to acknowledge a number of people who contributed towards the completion of this thesis. Firstly, I thank my supervisor Professor Montaz Ali for his patience, enthusiasm, guidance and teachings. The skills I have acquired during this process have infiltrated every aspect of my life. I remain forever grateful. Secondly, I would like to say a special thank you to Professor Jan Snyman for his assistance, which contributed immensely towards this thesis. I would also like to thank Professor Dominque Orban for his willingness to assist me for countless hours with the installation of CUTEr, as well as Professor Jose Mario Martinez for his email correspondence. A heartfelt thanks goes out to my family and friends at large, for their prayers, support and faith in me when I had little faith in myself. Thank you also to my colleagues who kept me sane and motivated, as well as all the support staff who played a pivotal roll in this process. Above all, I would like to thank God, without whom none of this would have been possible.

Abstract

Trajectory-based methods for solving constrained nonlinear programming problems (CNLPs) and mixed integer nonlinear programming problems (MINLPs) are proposed in this thesis. The trajectory-based method for unconstrained nonlinear programming problems (UNLPs) was proposed by Snyman [115]. The algorithms developed for CNLPs and MINLPs in thesis, which are extensions of Snyman's method [115], are however novel. First we develop a trajectory-based local algorithm for solving general CNLPs and then an adaptation of this algorithm for MINLPs is designed around the definition of a local minimum for MINLPs proposed by Newby [90]. In the development of the algorithms, the augmented Lagrangian function is used to convert the constrained problem into an equivalent unconstrained problem. Several contributions are made, including a new scheme for updating the penalty parameter, the implementation of an adaptive step size routine and a scaling mechanism for badly scaled problems. Global and local convergence properties of TACNLP are stablished.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Greek Symbols

δ	The maximum allowable size for the space steps Δx and $\Delta\lambda$
δ_1	Used for the magnification of Δt_x and Δt_λ
$\hat{\varepsilon}$	Tolerance for the discretization error of Euler's method
ε	Tolerance for the trajectory-based algorithms
$\hat{\lambda}$	The Lagrange multiplier estimate obtained using modified Euler's method
λ	The Lagrange multiplier
Λ	Used to denote the Lagrange multiplier in certain instances
γ	The decreasing factor for the conventional penalty parameter update
μ	The penalty parameter
μ_{max}	The maximum value which μ can attain
μ_{min}	The minimum value which μ can attain
Ω_c	The feasible region of the general CNLP problem
Ω_d	The feasible region of the general INLP problem
Ω_m	The feasible region of the general MINLP problem
ϕ_A	The augmented Lagrangian for equality and inequality constraints
$\nabla_x \phi_A$	The gradient of the augmented Lagrangian with respect to x
$\nabla_\lambda \phi_A$	The gradient of the augmented Lagrangian with respect to λ

Δt The time step

Δt_{max} The maximum value Δt can attain

Δt_{min} The minimum value Δt can attain

τ Scaling parameter

Δt_{λ} The time step with respect to λ

Δt_{μ} The time step with respect to μ

Δt_{x} The time step with respect to x

Δx The space step

Superscripts

k superscript index

Subscripts

i subscript index

j subscript index

Other Symbols

$A(x)$ The set of active constraints at x

$a(x)$ The force acting on the particle x

A The matrix describing the linear equality constraints, $Ax = b$

$A(x)^c$ The complement of $A(x)$

$A_s(x)$ The set of strongly active constraints at x

b The vector describing the linear equality constraints, $Ax = b$

b^* The optimal solution for the Lagrangian dual problem

$B_{\varepsilon}(x_c)$ The open ball $\{x \in \mathbb{R}^{n_c} : |x - x_c| < \varepsilon\}$

$c_i,\ i \in E$ The number of equality constraints

$c_i,\ i \in I$ The number of inequality constraints

∇c_i The gradient of the ith constraint, also known as the constraint normal

$(c_i)_s$ The mean value obtained for the constraint, c_i

$(\nabla c_i)_s$ The mean value obtained for the constraint normal, ∇c_i

d The direction vector used in several algorithms

d_i The coordinate directions used to generate trial points using the rule of PS, $d_i,\ i = 1,\ldots,n_s$

E The finite index set of equality constraints

f The objective function

$f^M(x,y_d)$ The feasible continuous manifold obtained by fixing the integer variables to the value y_d

∇f The gradient of f

$(\nabla f)_s$ The mean value obtained for the gradient of the objective function, ∇f

g The vector describing the quadratic function, $q(x) = \frac{1}{2}z^T Hz + g^T z$

H The matrix describing the quadratic function, $q(x) = \frac{1}{2}z^T Hz + g^T z$

I The finite index set of inequality constraints

i_λ Used to ensure that the descent condition, (4.6), is satisfied

i_x Used to ensure that the descent condition, (4.6), is satisfied

j_λ Used to ensure that the descent condition, (4.6), is satisfied

j_x Used to ensure that the descent condition, (4.6), is satisfied

∇L The gradient of the Lagrangian function, L

L The Lagrangian function

$\hat{M}$ Used for the adaptive step size update for μ

$\hat{m}$ Used to keep track of the desired progress of the trajectory computed in TAUNLP

$\hat{m}_\lambda$ Used to keep track of the desired progress of the trajectory computed wrt λ

$\hat{m}_x$ Used to keep track of the desired progress of the trajectory computed wrt x

$M(x)$ The set of active and infeasible inequality constraints

$M(x)^c$ The complement of $M(x)$

m_E The number of equality constraints in the general CNLP and MINLP

m_I The number of of inequality constraints in the general CNLP and MINLP

m The number of constraints in the general CNLP and MINLP

$\mathcal{N}(x)$ A continuous neighborhood of x.

$\mathcal{N}(y)$ A discrete neighborhood of y

n The total number of decision variables in the general CNLP and MINLP

$\mathcal{N}_{comb}(x,y)$ The neighborhood used in the definition of a combined local minimum

n_c The number of continuous decision variables in the general MINLP

$\mathcal{N}_d(x,y)$ A discrete user defined neighborhood

n_d The number of discrete decision variables in the general MINLP

$\mathcal{N}_r(x,y)$ The neighborhood used in the definition of a local minimum of a mixed integer problem

n_s A user prescribed parameter, indicating the number of integer points to include in the search space of the MINLP trajectory-based algorithm

p^* The best lower bound obtained from the Lagrangian dual function

$\hat{P}$ The number of integer-feasible trial points generated during the first phase of TAMINLP

$\underline{P}$ The number of points which satisfy the inequality (8.32)

p_λ Used for the magnification of Δt_λ

p_x Used for the magnification of Δt_x

Q The vector describing the linear function, $Q^T z$

q The number of random decision variables generated for scaling purposes

$\bar{q}$ The vector describing the linear function $l(x) = \bar{q}^T x$

$\underline{q}$ The vector describing the linear function $l(z) = \underline{q}^T z$

r convergence parameter

S_λ The discretization error estimate with respect to λ

S_x The discretization error estimate with respect to x

$f(x)$ Used to denote the potential energy of the particle x

$T(x)$ Used to denote the kinetic energy of the particle x

$t_{(\lambda,1)}$ The factor Δt_λ is decreased by during the step size routine

$t_{(x,1)}$ The factor Δt_x is decreased by during the step size routine

$t_{(\lambda,2)}$ The factor Δt_λ is increased by during the step size routine

$t_{(x,2)}$ The factor Δt_x is increased by during the step size routine

v The velocity of the particle x

w The velocity of the particle λ

W^k The Hessian matrix of the Lagrangian function L, evaluated at the kth iteration of the SQP algorithm, $W^k = \nabla^2_{xx} L(x^k, \lambda^k)$

x The vector of continuous decision variables

x^0 The user prescribed initial point for x

y The vector of integer decision variables

y^0 The user prescribed initial point for y

$\bar{y}$ The integer value obtained by rounding and fixing the solution, y^*_c, of the relaxed MINLP problem (8.1)

$\bar{y}^i$ The trial points generated about the centre $\bar{y}$, using the rule of PS

z The vector of continuous and discrete decision variables, $z = (x^T, y^T)^T$

z^*_c The solution to problem $\underline{\mathscr{M}}$, obtained during the first phase of TAMINLP, $z^*_c = (x^*_c, y^*_c)^T$

z_f^* The best solution to problem $\bar{\mathscr{M}}(\bar{y}^i),\ i = 1,\ldots,n_s$, obtained during the second phase of TAMINLP, $z_f^* = (x_f^*, y_f^*)^T$

z^* The local minimum obtained during the third and final phase of TAMINLP, $z^* = (x^*, y^*)^T$

$\lceil . \rceil$ The ceiling function

$\lfloor . \rfloor$ The floor function

Acronyms / Abbreviations

AL Augmented Lagrangian

ATAUNLP An adaptation of TAUNLP, for constrained nonlinear programming problems

BB Branch and Bound

$\alpha - BB$ α Branch and Bound

CNLPs Constrained nonlinear programming problems

ECP Extended Cutting Plane

$\alpha - ECP$ α Extended Cutting Plane

GBD Generalized Benders decomposition

INLPs Integer nonlinear programming problems

KKT Karush-Kuhn Tucker

LB Log Barrier

LICQ The Linear independence constraint qualification

MFCQ The Mangasarian-Fromovitz constraint qualification

MILPs Mixed integer linear programming problems

MINLPs Mixed integer nonlinear programming problems

MIQPs Mixed integer quadratic programming problems

NLPs Nonlinear programming problems

OA Outer approximation

QP Quadratic Penalty

SCQ Slater's Constraint Qualification

SQP Sequential Quadratic Programming

TAMINLP Trajectory-based method for mixed integer nonlinear programming problems

TACNLP Trajectory-based method for constrained nonlinear programming problems

TAUNLP Trajectory-based method for unconstrained nonlinear programming problems

UNLPs Unconstrained nonlinear programming problems

Chapter 1

Introduction

The aim of this chapter is to provide a brief introduction to the work contained in this thesis. Section 1.1 provides a description of general nonlinear programming problems (NLPs), including unconstrained NLPs (UNLPs) and constrained NLPs (CNLPs). In Section 1.2, we provide a description of general mixed integer nonlinear programming problems (MINLPs) considered in this thesis. Section 1.3 presents our motivation for the development of trajectory-based methods [115, 116, 118] for CNLPs and MINLPs. Finally, Section 1.4 outlines the layout of the rest of the thesis.

1.1 Nonlinear Programming Problems

The first class of problems we consider in this thesis are NLPs. NLPs are classified as either UNLPs or CNLPs. General UNLPs can be expressed as follows:

$$\min_{x \in \mathbb{R}^n} \quad f(x), \tag{1.1}$$

where $f : X \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$. In this thesis, however, we are concerned with CNLPs. Constrained nonlinear optimization continues to be a critical area of optimization with applications arising naturally in finance [34], water supply systems [37], [70], [72], automotive design [111], and all areas of engineering [13], [122]. Constrained nonlinear optimization, unlike its unconstrained counterpart, involves the minimization of an objective function subject to constraints. We consider the general CNLP of the form:

$$\mathscr{P}\begin{cases} \min\limits_{x\in\mathbb{R}^n} & f(x), \\ \text{s.t.} & c_i(x)=0, i\in E, \\ & c_i(x)\geq 0, i\in I, \\ & x\in\Omega_c, \end{cases} \tag{1.2}$$

where

$$\Omega_c=\{x\in\mathbb{R}^n | c_i(x)=0 \ \forall\, i\ \in E,\ c_i(x)\geq 0 \ \forall\, i\ \in I\},$$

$f: X\subseteq\mathbb{R}^n\longrightarrow\mathbb{R}$ and $c: X\subseteq\mathbb{R}^n\longrightarrow\mathbb{R}^m$ are twice continuously differentiable functions, and I and E are finite index sets of inequality and equality constraints respectively. The nonlinearity of the problem arises because at least one of $f(x)$ or $c_i(x)$ is nonlinear in the argument x. The total number of constraints in (1.2) is denoted by m. When there are only equality constraints present, the number of constraints is denoted by m_E, i.e. $m=m_E$. Similarly when there are only inequality constraints present, we denote the number of constraints by m_I, i.e. $m=m_I$.

Most solution approaches for this type of problem require first and second derivative information. The use of second derivative information usually renders second-order methods computationally expensive as opposed to first order methods [130]. The extension and adaptation of the trajectory-based method [115], for solving NLPs of type (1.2), is therefore crucial because it requires first derivative information only.

1.2 Mixed Integer Nonlinear Programming Problems

The second class of problems we consider in this thesis are MINLPs. When CNLPs of the form (1.2) contain continuous as well as integer variables, then they are referred to as MINLPs. MINLPs are defined as follows:

$$\mathscr{M}\begin{cases} \min\limits_{z} & f(z), \\ \text{s.t.} & c_i(z)=0, i\in E, \\ & c_i(z)\geq 0, i\in I, \\ & z\in X\times Y, \end{cases} \tag{1.3}$$

where $f: X\times Y\longrightarrow\mathbb{R}$ and $c: X\times Y\longrightarrow\mathbb{R}^m$ are real valued, nonlinear functions, z is the vector made up of continuous and integer variables, $z=(x^T,y^T)^T$, $X\subseteq\mathbb{R}^{n_c}$ and $Y\subseteq\mathbb{Z}^{n_d}$ is a

polyhedral set of integer points, and T is the transpose of a vector. The number of decision variables, n, in the general MINLP, is made up of the number of continuous variables, n_c, and the number of discrete variables, n_d, i.e. $n = n_c + n_d$. The variables, y, are the integer variables while the variables, x, are the continuous variables; y can either take on general integer values, binary values or a combination of the two. Throughout the rest of the thesis we write $z = (x,y)^T$ to mean $z = (x^T, y^T)^T$. Furthermore, any continuous vector x, will be written as $x = (x_1,\ x_2, \ldots,\ x_{n_c})^T$, and any discrete vector y will be written as $y = (y_1,\ y_2, \ldots,\ y_{n_d})^T$. Let Ω_m denote the feasible region of (1.3), then the point z is feasible if $z \in \Omega_m$. When no continuous variables are present in (1.3), the problem reduces to an integer nonlinear programming problem (INLP) and the feasible region Ω_m reduces to the discrete part only. We denote this discrete feasible region by Ω_d. Similarly, when no discrete variables are present in (1.3), the problem reduces to (1.2) and the feasible region Ω_m reduces to the continuous part only. We denote this feasible part, as in problem (1.2), by Ω_c.

In this thesis, we deal only with smooth objective functions $f(x)$ and $f(z)$ and smooth constraint functions $c_i(x)$ and $c_i(z)$. The problem (1.2) and the general MINLP problem (1.3) generally have more than one local minimum. We are interested in locating a local solution to (1.2) as well as a local solution to (1.3). Most existing solution processes for (1.3) are designed to solve convex MINLPs [27], [36], [41], [49], [55], [125], while the trajectory-based method proposed here adequately solves non-convex problems locally. Clearly the trajectory-based method can be applied to solve convex problems also.

1.3 Motivation

The main objective of this thesis is to develop a trajectory-based algorithm to solve (1.2) and adapt it to solve (1.3). Trajectory optimization is gaining some momentum since its inception [5, 74]. In the literature a few trajectory-based methods exist for solving UNLPs. These include first-order-in-time trajectory-based methods [10, 32] as well as second-order-in time trajectory-based methods [3, 4, 8, 9, 11, 26, 100, 115, 116, 118].

The method of Zirilli et al. [3] involves rewriting a system of second order ODEs as a set of first-order equations which are then linearized and solved using the implicit Euler method [3, 26].

On the other hand Antipin [8] uses the gradient projection method of first and second order to obtain a solution to the unconstrained problem (1.1). The problem (1.1) is formulated on drawing a trajectory which is obtained as a solution to the system of differential equations given by (1.4):

$$\dot{x} = -x + \pi_Q \times (x - \alpha \nabla f(x)),\ x(0) = x^0, \tag{1.4}$$

where $\pi_Q(\cdot)$ is the projection operator of a vector onto the set Q and $\alpha > 0$ is a step length parameter. The right hand side of (1.4) is chosen to satisfy necessary and sufficient optimality conditions at x^* [8].
The method of Alvarez [4] solves (1.1) by considering the solution of a second order evolution equation with linear damping and convex potential:

$$\ddot{x} + \gamma\dot{x} = -\nabla\Phi(x), \tag{1.5}$$

where $\Phi : \mathbb{H} \longrightarrow \mathbb{R}$, $\mathbb{H}$ is a real Hilbert space and γ is a positive real number, $\gamma > 0$. The system (1.5) is commonly referred to as a nonlinear oscillator with damping. The solution to (1.5) is guided to the minimum x^* because it is a dissipative system. Here a dynamical approach is used for the iterative framework of the proposed algorithm [4]. This methodology was then further extended by Alvarez and Attouch [9] to solve the convex constrained version of (1.2) by considering the gradient-projection dynamical system

$$\ddot{x} + \gamma\dot{x} + x = \pi_C(x - \mu\nabla\Phi(x)), \tag{1.6}$$

where $\pi_C(\cdot)$ is the projection operator of a vector onto C which is a closed convex subset of $\mathbb{H}$.
A review of the methods of Snyman [115, 116], Snyman and Fatti [118] and the generalized descent method of Griewank [62] can be found in Diener et al. [39]. The basic idea of Snyman's method [115, 116] is to solve the differential equation

$$\ddot{x} = -\nabla f(x),\ x(0) = x^0,\ \dot{x}(0) = 0, \tag{1.7}$$

where $f(x)$ is the real valued unconstrained function to be minimized over some n-dimensional box which contains all minimizers as interior and isolated, and ∇f is the gradient of f. This local method was then extended by Snyman and Fatti [118] to solve unconstrained global optimization problems. The solution process of Snyman and Fatti's method involves performing multiple searches to locate the global minimizer of the problem, with a certain probability. The method was also extended by Snyman to solve constrained problems via penalty function formulations [117]
On the other hand, the generalized descent method of Griewank [62] solves the system

$$\max[0, f(x) - c]\,\ddot{x} + e(I - \dot{x}\dot{x}^T)\nabla f(x) = 0,\ x(0) = x_0,\ \dot{x}(0) = 0, \tag{1.8}$$

where c and e are parameters. The method is parameter dependent and convergence to the global minimizer can be guaranteed for a certain class of functions only. Unfortunately, not

much numerical testing with the generalized descent is known in the literature.

The most common solution processes for (1.2) use first and second derivate information to reach an optimal solution. Some benchmarking software packages, namely LANCELOT, which uses the augmented Lagrangian method [33], and LOQO, based on an interior point algorithm [121], requires first and second-order information. Other software packages, namely SNOPT, which uses the sequential quadratic programming method [61], [56], [57] and MINOS, which is based on a combination of quasi-Newton's method, a projected Lagrangian method, the simplex method and the reduced-gradient method [89], only requires first order information. The use of second-order information generally reduces the number of iterations, but increases the CPU time, rendering the method computationally expensive. Depending on the nature of the problem, it may not even be possible to obtain second-order information. Even when no second-order information is required, sometimes line search techniques are needed to ensure convergence to an optimal solution. Line search procedures however, usually require a significant amount of function evaluations to monitor the progress of a sequence of iterates to a solution [52], [130]. Based on the fact that existing second-order methods are usually computationally expensive, and alternative methods, which use line-search strategies, require function evaluations, we develop a trajectory-based method for CNLPs.

In the trajectory-based method presented here solution trajectories are computed for Augmented Lagrangian formulations of the constrained problem (1.2) and the method uses first order information only and requires no function evaluations throughout the solution process. This method is not only an extension of the trajectory-based method proposed by Snyman [115, 116], but includes some novel updates to the existing trajectory-based method. These include:

- The implementation of an adaptive step-size routine.
- A new technique for updating the penalty parameter μ.
- A mechanism used to circumvent the effects of badly-scaled problems.

In addition, we have provided convergence proofs which were not given in the original papers [115, 116, 118], dealing with UNLPs.

In this thesis, our first objective is to develop a trajectory-based algorithm for CNLPs, which we denote as TACNLP. Our second objective is to extend this and develop a trajectory-based algorithm for MINLPs. This algorithm is denoted as TAMINLP.

Mixed integer optimization has applications arising naturally in process and engineering design [75], heat exchanger networks [21], production planning and control [88, 120] location - allocation [81], pump configurations [97], water transmission networks, finance [82], process flowsheets [76] and scheduling [66, 109]. Due to these numerous applications of MINLPs [21, 66, 75, 76, 81, 82, 88, 97, 109, 120] several methods have been developed to solve convex MINLPs. These are mostly deterministic methods, viz., the Branch and Bound method (BB), the Outer approximation method (OA), the Generalized Benders Decomposition method (GBD), and the extended cutting plane method (ECP).
These methods usually combine methods from CNLPs and mixed integer linear programming problems (MILPs) [27, 36, 41, 49, 55, 125]. For MINLPs, a large number of binary variables may results in a potentially large combinatorial problem [50]. If binary as well as general integer variables are present, then this poses an even harder problem. Furthermore, if the MINLP is non-convex, then it generally contains more than one solution and the deterministic methods for convex MINLPs mentioned above can not be applied to solve it. A number of global MINLP algorithms have, however, been developed for non-convex problems with specific structures. These algorithms use convex under-estimators for the original non-convex problem [2, 102]. Unfortunately, very little research can be found in the literature for general non-convex MINLPs. Developing an algorithm which is able to solve general MINLPs will therefore have a significant impact on the advancement of existing methods for MINLPs. At present no trajectory-based algorithm for MINLPs exists. It is with this objective in mind that we develop a local trajectory-based algorithm for general MINLPs.

1.4 Organization of the thesis

The remainder of the thesis is organized as follows. In Chapter 2, we outline some important theoretical concepts pertaining to the solution of CNLPs and summarize some of the most common nonlinear programming methods used for solving CNLPs. Chapter 3 contains a summary of the most common solution approaches used to solve general MINLPs. In Chapter 4, we provide a review of Snyman's trajectory-based algorithm for UNLPs [115], which is denoted as TAUNLP throughout the rest of this thesis. Chapter 5 contains an outline of the framework of our trajectory-based algorithm for CNLPs, TACNLPs. In Chapter 6, we outline and compare TACNLP with an adaptation of TAUNLP for CNLPs. We denote this adaptation as ATAUNLP. The comparison between TACNLP and ATAUNLP highlights the fundamental differences between TACNLP and ATAUNLP. Chapter 7 contains local and global convergence discussions of TACNLP. Chapter 8 outlines the trajectory-based algorithm for MINLPs, denoted as TAMINLP. The overview of an important advancement in

the definition of a local minimum of an MINLP [90] is also presented in Chapter 8. We will use this definition to aid in the convergence discussion of the trajectory-based algorithm for MINLPs. Computational results demonstrating the performance of TACNLP and ATAUNLP are given in Chapter 9. In Chapter 10, we present numerical results for TAMINLP. Finally, some concluding remarks are made in Chapter 11.

Chapter 2

Review of Some Solution Techniques for CNLPs

In this chapter, we provide a review of various existing methods for continuous nonlinear programming problems. In Section 2.1, we provide a review on theories of nonlinear programming problems for continuous variables. We then provide an overview of some of the most common solution techniques used for solving problem (1.2), in Section 2.2.

2.1 Theories of Constrained Nonlinear Optimization

Problem (1.2) contains both equality and inequality constraints. Inequality constraints are classified as either active or inactive at the optimal solution. We refer to the following definition for the differentiation between active and inactive constraints.

Definition 2.1.1. *(Active and Inactive constraints)*
For the set of inequality constraints $c_i(x) \geq 0$, the i-th constraint is said to be active at some feasible point $\hat{x}$, if $c_i(\hat{x}) = 0$, and inactive if $c_i(\hat{x}) > 0$. If $c_i(\bar{x}) < 0,\ i \in I$, then $c_i(x)$ is said to be violated at some infeasible point $\bar{x}$ [52]. Clearly all equality constraints are active at a feasible solution $\hat{x}$. The active index set is defined at any feasible x as follows:

$$A(x) = E \cup \{i \in I | c_i(x) = 0\}. \tag{2.1}$$

CNLPs are often characterized by a nonlinear function and/or nonlinear constraints. A direct implication of this is that more than one solution to (1.2) may exist. In this thesis, we are only interested in obtaining a local solution to (1.2). The distinction between a local and a global solution is provided in the definitions.

Definition 2.1.2. *(Local minimizer)*
A vector $x^* \in \Omega_c$ is a local minimizer of (1.2)*, if there exists a neighborhood $\mathcal{N}(x^*)$ of x^*, such that*

$$f(x^*) \leq f(x),\ \forall\ x \in \mathcal{N}(x^*) \cap \Omega_c.$$

Definition 2.1.3. *(Global minimizer)*
A vector $x^* \in \Omega_c$ is a global minimizer of (1.2)*, if*

$$f(x^*) \leq f(x),\ \forall\ x \in \Omega_c.$$

Problem (1.2) is referred to as the primal problem, since it is defined in terms of the primal variables x. Recall that (1.2) involves the minimization of an objective function subject to constraints. To incorporate the effect of the constraints on the minimization of the objective function, we introduce the Lagrangian function:

$$L(x, \lambda) = f(x) - \sum_{i \in E \cup I} \lambda_i c_i(x) = f(x) - \lambda^T c(x). \tag{2.2}$$

The impact of a particular constraint, $c_i(x)$, on the minimization of the objective function is quantified by its corresponding Lagrangian multiplier λ_i, which is also known as the dual variable associated with problem (1.2)[1]. To demonstrate this, we notice that at the optimal solution, $\{x^*,\ \lambda^*\}^T$, the gradient of the Lagrangian (2.2) vanishes:

$$\nabla_x L(x^*, \lambda^*) = \nabla f(x^*) - \sum_{i \in E \cup I} \lambda_i^* \nabla c_i(x^*) = 0. \tag{2.3}$$

Depending on whether (1.2) contains equality or inequality constraints, (2.3) must be interpreted accordingly. When only equality constraints are present, an implication of (2.3) is that the gradient of the objective function can be written as a linear combination of the gradient of the constraints or constraint normals, at the optimal solution $(x^*,\ \lambda^*)^T$:

$$\nabla f(x^*) = \sum_{i \in E} \lambda_i^* \nabla c_i(x^*) = \nabla c(x^*) \lambda^*$$

where λ^*, is the vector of Lagrange multipliers $\lambda_i^*, i \in E$. When only inequality constraints are present in (1.2), the following cases arise:

[1]The dual problem associated with problem (1.2) will be defined later in this section

Case 1: Consider the case when all $c_i(x)$ are active at x^*, i.e. $c_i(x^*) = 0, \forall i \in I$. In this case, as with the equality constrained problem, (2.3) implies that

$$\nabla f(x^*) = \sum_{i \in I \cap A(x^*)} \lambda_i^* \nabla c_i(x^*), \tag{2.4}$$

but the non-negative restriction on the Lagrange multiplier, i.e. $\lambda_i^* \geq 0, \forall i \in I$, is imposed. If (2.4) were satisfied with negative values for λ_i^*, then λ^*, the vector of Lagrange multipliers $\lambda_i^*, i \in I$, would not be optimal. This is because, satisfying (2.3) with negative values for λ_i^*, implies that a decrease in the current objective function value at the corresponding point x^*, is still possible [130].

Case 2: Consider now the case when all $c_i(x)$ are inactive at x^*. Here $c_i(x^*) > 0, \forall i \in I$, and these constraints do not have an impact on the minimization of $f(x)$. In this case the corresponding Lagrange multipliers are 0, i.e. $\lambda_i^* = 0, \forall i \in I$, and (2.3) reduces to the unconstrained optimality condition:

$$\nabla f(x^*) = 0.$$

Remark 2.1.1. When both equality and inequality constraints are present, we have that

$$\nabla f(x^*) = \sum_{i \in A(x^*)} \lambda_i^* \nabla c(x^*).$$

Here, inactive inequality constraints are not included in the right hand summation since, as we have shown, they have null Lagrange multipliers.

We observe that λ_i^* describes how sensitive the optimal objective function value, $f(x^*)$, is to the presence of the constraint $c_i(x^*)$ [130]. With reference to the following definition, we can now classify constraints in terms of their corresponding Lagrange multipliers.

Definition 2.1.4. *(Strongly active constraints and weakly active constraints)*
A constraint is said to be strongly active if its corresponding Lagrange multiplier is not equal to 0. Conversely, a constraint is said to be weakly active at some point $\bar{x}$, if $c_i(\bar{x}) = 0$ and its corresponding Lagrange multiplier $\bar{\lambda}_i$ is also 0.

The set of strongly active constraints, which we denote by $A_s(x)$, are those in $A(x)$, which are strongly active, as per **Definition 2.1.4**. For a strongly active equality constraint, λ_i may be negative or positive, but the Lagrange multiplier λ_i corresponding to a strongly active inequality constraint must be strictly positive. We reiterate that all inactive inequality

constraints have null Lagrange multipliers, $\lambda_i = 0$. If $\lambda_i = 0$, then the corresponding constraint has no effect on the minimization of $f(x)$.

Before we summarize the first order optimality conditions for constrained nonlinear optimization, we introduce what is known as a constraint qualification. Constraint qualifications are necessary for establishing optimality conditions. We illustrate this with an example. Consider the following problem:

$$\begin{cases} \min & f(x) = x_1 + x_2, \\ \text{s.t.} & c_1(x) = (x_1^2 + x_2^2 - 1)^3 = 0. \end{cases}$$

Notice that, $\nabla c_1(x) = (0,\ 0)^T$ for any feasible point, and at the minimizer, $x^* = (-\sqrt{\frac{1}{2}}, -\sqrt{\frac{1}{2}})^T$, $\nabla L(x^*, \lambda^*) = 0$, reduces to $\nabla f(x^*) = 0$, which is a fallacy. A constraint qualification ensures that for any problem (1.2), this degeneracy does not occur, i.e. the normal to the constraint, $\nabla c_i(x)$, does not vanish at x^* [52], [130]. One such constraint qualification is defined below.

Definition 2.1.5. *(Linear independence constraint qualification (LICQ))*
If we consider the set of constraints in (1.2) *at some point x^*, then the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients $\{\nabla c_i(x^*), i \in A(x^*)\}$ is linearly independent [52].*

We can now summarize the first order optimality conditions for the general CNLP (1.2).

Theorem 2.1.6. *(Karush-Kuhn-Tucker (KKT) conditions)*
Suppose x^* is a local solution of (1.2) *and that the LICQ holds at x^*, then x^* is a KKT point or equivalently, x^* satisfies the first order necessary optimality (KKT) conditions if there exists a Lagrange multiplier vector λ^* such that:*

(2.1.6a) $\nabla_x L(x^*, \lambda^*) = 0$,

(2.1.6b) $c_i(x^*) = 0,\ \forall\ i \in E$,

(2.1.6c) $c_i(x^*) \geq 0,\ \forall\ i \in I$,

(2.1.6d) $\lambda_i^* \geq 0,\ \forall\ i \in I$,

(2.1.6e) $\lambda_i^* c_i(x^*) = 0,\ \forall\ i \in E \cup I$,

where L is defined as in (2.2).

Proof. See Nocedal and Wright [130]. □

The condition, $(2.1.6e)$, is called the complementarity condition. It implies that the Lagrange multiplier $\lambda_i^*, i \in I$, can be strictly positive if and only if $c_i(x^*), i \in I$, is strongly active. When this is not the case, i.e. when $c_i(x),\ i \in I$ is weakly active or inactive, then the corresponding Lagrange multiplier λ_i is null. If strict complementarity of the constraint and multiplier pair is satisfied, then $(6d)$ implies $\lambda_i^* > 0, \forall\ i \in I \cap A(x^*)$ and $\lambda_i^* = 0,\ \forall i \in I \setminus A(x^*)$. An implication of this is that all active constraints are strictly active at x^*, i.e. $A(x^*) = A_s(x^*)$. A formal definition of strict complementarity is given in **Definition 2.1.7**.

Definition 2.1.7. *(Strict complementarity)*
Strict complementarity holds at a Karush-Kuhn-Tucker (KKT) point x^*, if there is a multiplier λ^* satisfying the KKT conditions such that $\lambda_i^* > 0\ \forall i \in I \cap A_s(x^*)$ [52].

The *KKT conditions* can be depicted geometrically. We do so by considering the following problem:

$$\begin{cases} \min & f(x) = (x_1 - 1)^2 + (x_2 - 1)^2, \\ \text{s.t.} & c_1(x) = -0.5x_1 - x_2 - 1.07 \geq 0, \\ & c_2(x) = 4.25 - (x_1 + 2)^2 - (x_2 + 2)^2 \geq 0, \end{cases}$$

which has solution $x^* = (0.5086\ -1.336)^T$. Figure 2.1 represents the *KKT conditions* of this problem. This figure was adapted from Emet [43]. The feasible region is the shaded area. It is clear from the figure that at x^*, $\nabla f(x^*)$ can be written as a linear combination of $\nabla c_1(x^*)$ and $\nabla c_2(x^*)$, since $\nabla f(x^*)$ points into the cone spanned by $\nabla c_1(x^*)$ and $\nabla c_2(x^*)$. This is inline with the the *KKT conditions*, where $\nabla_x L(x^*, \lambda^*) = \nabla f(x^*) - \sum_{i \in E \cup I} \lambda_i^* \nabla c_i(x^*) = 0$. Moreover, since $\nabla c_1(x^*)$ and $\nabla c_2(x^*)$ are both strictly active at x^* and $\{\nabla c_1(x^*), \nabla c_2(x^*)\}$ is linearly independent, it follows that $\lambda_1^*, \lambda_2^* > 0$, i.e. strict complementarity holds.

The *KKT conditions* can also be defined in terms of the weaker Mangasarian-Fromovitz constraint qualification defined below:

Definition 2.1.8. *(Mangasarian-Fromovitz constraint qualification (MFCQ))*
Given the point x^* and the active set $A(x^*)$, the Mangasarian-Fromovitz constraint qualification (MFCQ) holds if there exists a vector $d \in \mathbb{R}^n$, such that

$$\begin{aligned} &\nabla c_i(x^*)^T d > 0, \quad \forall i \in (A(x^*) \cap I), \\ &\nabla c_i(x^*)^T d = 0, \quad \forall i \in E, \end{aligned}$$

and the set of equality constraint gradients is linearly independent [130].

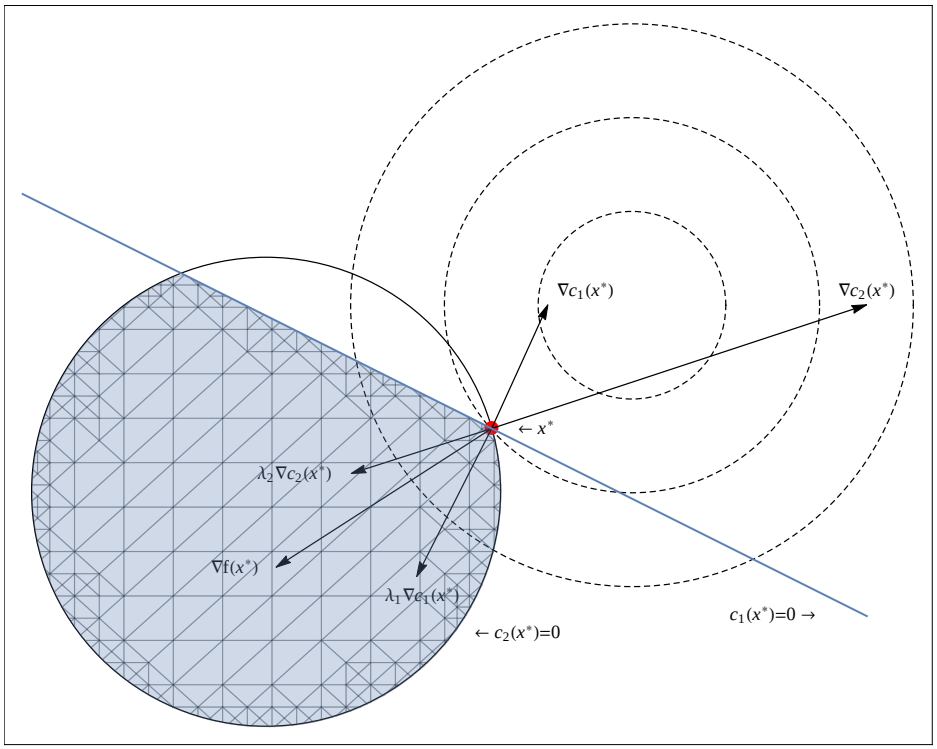


Figure 2.1 Illustration of a KKT point.

It is possible to prove a version of **Theorem 2.1.6** with MFCQ replacing LICQ. MFCQ has the useful property that it is equivalent to the boundedness of the set of Lagrange multiplier vectors λ^* for which the *KKT* conditions of **Theorem 2.1.6** (with MFCQ replacing LICQ) are satisfied [130].
LICQ is a stronger constraint qualification than MFCQ. To demonstrate this consider a problem with the feasible region characterized by the following constraints

$$c_1(x) = 2 - (x_1 - 1)^2 - (x_2 - 1)^2 \geq 0,$$
$$c_2(x) = 2 - (x_1 - 1)^2 - (x_2 + 1)^2 \geq 0,$$
$$c_3(x) = x_1 \geq 0.$$

At the point $x^* = (0,\ 0)^T$ all three constraints are active. In order to satisfy MFCQ we therefore need only find some direction $d = (d_1,\ d_2)^T$ such that $\nabla c_i(x^*)^T d > 0,\ i = 1,\ldots,3$. If we choose $d = (1,0)^T$ then:

$$\nabla c_1(x^*)^T d = \begin{pmatrix} 2 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 2 > 0,$$

$$\nabla c_2(x^*)^T d = \begin{pmatrix} 2 & -2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 2 > 0,$$

$$\nabla c_3(x^*)^T d = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1 > 0.$$

MFCQ is therefore satisfied at x^* with $d = (1,0)^T$. We have shown that all three constraints are active at x^*, therefore LICQ is satisfied if we can prove that the set of constraint gradients is linearly independent. If we choose $k_1 = 1$, $k_2 = 1$ and $k_3 = -4$ however, then $\sum_{i=1}^{3} k_i \nabla c_i(x^*) = (0,0)^T$. Hence LICQ is not satisfied at x^*. This proves that LICQ is indeed stronger than MFCQ.

The last constraint qualification we consider, is a special case of MFCQ [16]. This constraint qualification is known as Slater's constraint qualification (SCQ) for convex problems, and is associated with strong duality. Before we present Slater's constraint qualification, we need some background on the theory of duality.

2.1.1 Duality

Problem $\mathscr{M}$ defined by (1.2), is usually difficult to solve directly. As an alternative, one can solve its dual problem. Recall the Lagrangian function defined by (2.2) and the associated dual variable λ. We now define the Lagrange dual function $D : \mathbb{R}^m \longrightarrow \mathbb{R}$, as the minimum possible value of L over its range:

$$D(\lambda) = \inf_{x \in \mathscr{D}} L(x, \lambda) = \inf_{x \in \mathscr{D}} (f(x) - \sum_{i \in E \cup I} \lambda_i c_i(x)), \tag{2.5}$$

where D is a concave function and $\mathscr{D} = \cap_{i=1}^{m} dom(c_i)$ [16], [25].

Let p^* denote the optimal value of (1.2), that is, the optimal value of f at the feasible optimal solution x^*. Then, provided $D(\lambda)$ is bounded below, for $\lambda_i \geq 0,\ i \in I$, the Lagrange dual function yields lower bounds on this optimal value p^* of (1.2):

$$D(\lambda) \leq p^*. \tag{2.6}$$

We now verify this. For some feasible point $\bar{x}$ of the problem (1.2) and $\lambda_i \geq 0,\ i \in I$, we have

$$\sum_{i \in E \cup I} \lambda_i c_i(x) \geq 0,$$

since each term in the sum is non-negative. It therefore follows that:

$$L(\bar{x},\lambda) = f(\bar{x}) - \sum_{i\in E\cup I} \lambda_i c_i(\bar{x}) \leq f(\bar{x}).$$

Now since $D(\lambda) \leq L(\bar{x},\lambda)$ for every feasible point $\bar{x}$, it follows that (2.6) holds. The inequality (2.6) only holds when $\lambda_i \geq 0, i \in I$ and $D(\lambda) > -\infty$. By (2.6), for $\lambda_i \geq 0, i \in I$ the Lagrange dual function yields a lower bound on the optimal value p^*.

Solving the following problem, we obtain the best lower bound of the Lagrange dual function:

$$DP \begin{cases} \max & D(\lambda), \\ \text{s.t.} & \lambda_i \geq 0,\ i \in I. \end{cases} \tag{2.7}$$

This is called the Lagrange dual problem associated with (1.2). Even when (1.2) is non-convex, DP is convex since $D(\lambda)$ is concave and the constraint $\lambda_i \geq 0,\ i \in I$ is linear. We can now define weak and strong duality in terms of the Lagrange dual problem.

Definition 2.1.9. *(Weak Duality)*
Denote the optimal solution to (2.7) as $\bar{b}^*$. This is the best lower bound on p^* that can be obtained from the Lagrange dual function. When the following equality is satisfied:

$$\bar{b}^* \leq p^*, \tag{2.8}$$

we say that weak duality holds. This inequality holds even when the original problem is non-convex.

The difference between p^* and $\bar{b}^*$ is known as the optimal duality gap of the original problem since it describes the gap between the solution of the primal problem (1.2) and the dual problem (2.7). Strong duality holds when there is no duality gap.

Definition 2.1.10. *(Strong Duality)*
Strong duality holds when the equality

$$\bar{b}^* = p^*,$$

is satisfied, i.e., the optimality gap is zero. Naturally, when the primal problem is convex, the duality gap is zero.

Strong duality on a convex problem can also be verified by satisfying constraint qualifications [25]. One such constraint qualification is Slater's constraint qualification.

Definition 2.1.11. *(Slater's constraint qualification (SCQ))*
Given the convex problem

$$\begin{cases} \min\limits_{x\in\mathbb{R}^n} & f(x), \\ s.t. & c_i(x) \geq 0, i \in I, \\ & Ax = b, \end{cases} \tag{2.9}$$

where $f : X \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$ and $c : X \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}^{m_I}$ are convex, $A \in \mathbb{R}^{m_E \times n}$ and $b \in \mathbb{R}^{m_E}$, Slater's constraint qualification (SCQ) is satisfied if there exists an $x \in relint(\Omega_c)$, where relint is the relative interior of Ω_c, such that

$$c_i(x) > 0, i \in I$$

and

$$Ax = b.$$

By convexity of the problem, *SCQ* implies that strong duality holds [16], [25].

Remark 2.1.2. From **Theorem 2.1.9**, it follows that if $\mathscr{P}$ is unbounded below, (that is if $p^* = -\infty$), then *DP* is infeasible. Also, if *DP* is unbounded above (that is if $b^* = \infty$), then $p^* = \infty$, that is $\mathscr{P}$ is infeasible [93].

Theorem 2.1.12. *If there exists $x \in \Omega_c$ and λ such that*

$$f(x) = D(\lambda), \tag{2.10}$$

then,

$$f(x) = p^* = D(\lambda) = \bar{b}^*. \tag{2.11}$$

This implies that x is optimal and λ is optimal.

Proof. This proof is adapted from [93]. We know that the following relations hold:

$$D(\lambda) \leq \bar{b}^*, \tag{2.12}$$

and

$$p^* \leq f(x). \tag{2.13}$$

From **Theorem 2.1.12**, and the relations (2.12) and (2.13), we have

$$D(\lambda) \leq \bar{b}^* \leq p^* \leq f(x), \tag{2.14}$$

Combining (2.10) and (2.14), we get (2.11), which completes the proof. □

Theorem 2.1.13. *If strong duality holds, and $\mathscr{P}$ is a convex optimization problem, then the KKT conditions are sufficient, that is, every KKT point is an optimal point and every dual variable such that $\{x^*, \lambda^*\}$ satisfy (2.1.6) is an optimal pair for DP.*

Proof. The conditions $(2.1.6b)$ and $(2.1.6c)$ imply that the point x^* is primal feasible. Since the functions f, c_i, $i \in E \cup I$ are convex, L, defined by (2.2), is also convex. The conditions $(2.1.6a)$ shows that x^* minimizes $L(x, \lambda^*)$ over $\mathscr{D}$. Thus $D(\lambda^*) = L(x^*, \lambda^*)$. Furthermore, by applying $(2.1.6b)$ and $(2.1.6c)$, we have $D(\lambda^*) = f(x)^*$. It follows by **Theorem 2.1.12** , that $\bar{b}^* = D(\lambda^*) = f(x^*) = p^*$. So x^* is a primal optimal point and λ^* is dual optimal. □

Assuming that f and c are twice continuously differentiable, second order sufficient conditions for (1.2) are presented in the following theorem [130].

Theorem 2.1.14. *(Second-Order Sufficient Conditions)*
Suppose that x^* is a local solution of (1.2) *and that LICQ is satisfied. Furthermore assume that λ^* satisfies the KKT conditions in* **Theorem 2.1.6** *and let $d \in S(x^*)$, where*

$$S(x^*) = \begin{cases} & \nabla c_i(x^*)^T d = 0, \quad i \in E \\ d \in \mathbb{R}^n | d \neq 0, & \nabla c_i(x^*)^T d = 0, \quad i \in A_s(x^*) \cap I, \\ & \nabla c_i(x^*)^T d \geq 0, \quad i \in (A(x^*) \setminus A_s(x^*)) \cap I, \end{cases} \tag{2.15}$$

then

$$d^T \nabla^2_{xx} L(x^*, \lambda^*) d > 0, \ \ \forall\, d \in S(x^*), \tag{2.16}$$

where $A(x^*)$ and $A_s(x^*)$ are defined as in **Definitions 2.1.1** *and* **2.1.4** *respectively.*

Proof. See Nocedal and Wright [78, 130]. □

Generally practical iterative methods are used to solve (1.2). The algorithmic framework of the solution process is to formulate subproblems at each iteration and carry out an approximate minimization with respect to the primal variable x, until the sequence of iterates $\{x^k\}$ converges to x^*, where k is the iteration counter of the algorithm. Convergence can be characterized both locally and globally. The local convergence of an algorithm describes the rate of convergence in some neighborhood of $x^* \in \Omega$, while the global convergence of an

algorithm tells us how efficiently the algorithm converges independent of the initial point x^0. Note that for an algorithm to be globally convergent, it is sufficient that it converges from any point x^0 to a stationary point x^*, where x^* need not be a global minimizer. The basis for local convergence analysis of an algorithm can be summarized below.

a The algorithm is said to converge linearly if it generates the sequence $\{x^k\}$ which converges to x^* such that for some $\alpha \in (0,\ 1)$ and $k_0 \geq 0$

$$||x^{k+1} - x^*|| \leq \alpha ||x^k - x^*||,\ \ k \geq k_0.$$

b The algorithm is said to converge superlinearly if it generates the sequence $\{x^k\}$ which converges to x^* such that for some sequence $\{\alpha^k\} \to 0$

$$||x^{k+1} - x^*|| \leq \alpha^k ||x^k - x^*||,\ \ k = 0, 1\ \ldots$$

c The algorithm is said to converge quadratically if it generates the sequence $\{x^k\}$ which converges to x^* such that for some non-negative α, not necessarily less than 1

$$||x^{k+1} - x^*|| \leq \alpha ||x^k - x^*||^2,\ \ k = 0, 1\ \ldots$$

Within the iterative scheme, (1.2) may be modified or reformulated into an equivalent unconstrained problem. We will now consider some of the most common reformulations of (1.2).

2.2 Review of Constraints Handling in Constrained Nonlinear Programming

For general constrained optimization problems we expect the presence of constraints to increase the level of difficulty of the problem. This can however be circumvented by reformulating the problem. In the most commonly used solution approaches, the problem $\mathscr{P}$, given by (1.2), is reformulated into a penalty function [15, 35, 38, 47, 53, 58, 85, 130]. The most prominent penalty methods used include the quadratic penalty method (QP) [35], the logarithmic (log) barrier method (LB) [47, 53] and the augmented Lagrangian method (AL) [15, 67, 107], also known as the method of multipliers. We now provide a brief outline of the aforementioned.

2.2.1 Penalty methods for nonlinear constrained optimization

The QP method

The quadratic penalty method (QP) was first proposed by Courant [35]. It is defined for equality constraints $c_i(x)$, $i \in E$. It reformulates the original problem, defined by:

$$\begin{cases} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & c_i(x) = 0, i \in E \end{cases} \tag{2.17}$$

into the QP function:

$$Q(x;\mu) = f(x) + \frac{1}{2\mu}\sum_{i\in E} c_i^2(x), \tag{2.18}$$

where $\mu > 0$ is a penalty parameter. The methods based on QP solve (2.17) iteratively for each k, by generating a decreasing sequence of values for μ which penalize the constraint violations, by driving μ to 0, $\mu_k \to 0$, as $k \to \infty$. One of the draw-backs of QP is that the equality constraints $c_i(x) = 0$, $i \in E$ are not satisfied to exactness, rather they satisfy

$$c_i(x^k) = -\mu_k \lambda_i^*, \ \forall\, i \in E, \tag{2.19}$$

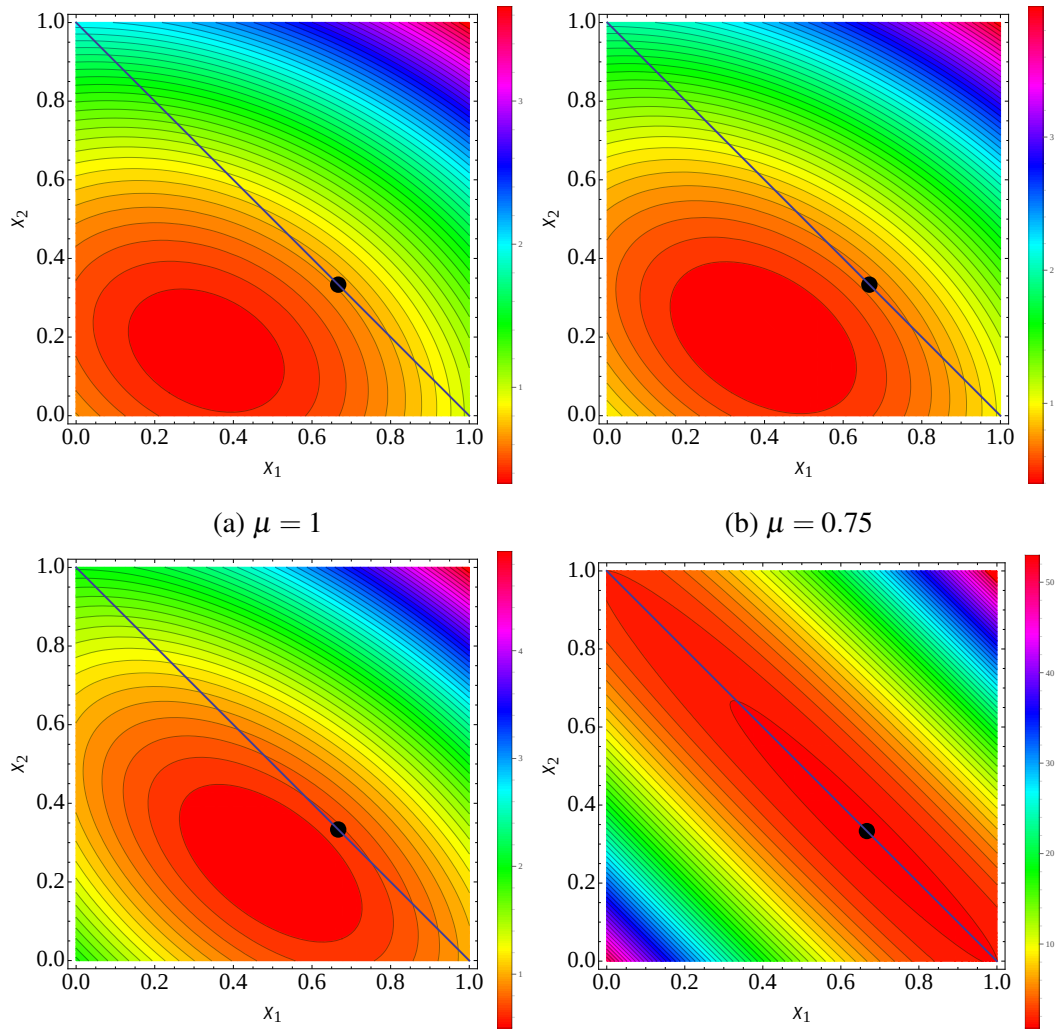
where λ^* satisfies the *KKT conditions* in **Theorem 2.1.6**. See Nocedal and Wright [130] for the proof. As $\mu \to 0$, however, the approximation improves, but unless search directions are carefully chosen, this results in ill conditioning of the Hessian matrix $\nabla^2_{xx} Q(x;\mu)$ for small values of μ. Figure 2.2 illustrates this ill conditioning. Consider the problem

$$\begin{cases} \min & f(x) = x_1^2 + 2x_2^2 \\ \text{s.t.} & c_1(x) = x_1 + x_2 - 1 = 0, \end{cases}$$

which has the solution $x^* = (\frac{2}{3}, \frac{1}{3})^T$. The QP function for this problem is

$$Q(x;\mu) = x_1^2 + 2x_2^2 + \frac{1}{2\mu}|x_1 + x_2 - 1|^2.$$

$Q(x^*;\mu)$ is illustrated in Figure 2.2 for different values of μ. The constraint $c_1(x)$ as well as the optimal solution x^* depicted by the bold black point on $c_1(x)$ are also included in each figure.



(a) $\mu = 1$ (b) $\mu = 0.75$

(c) $\mu = 0.5$ (d) $\mu = 0.01$

Figure 2.2 Quadratic penalty function for min $x_1^2 + 2x_2^2$ subject to $x_1 + x_2 = 1$.

We notice in Figure 2.2d, that the minimum corresponding to the QP function, is a fairly accurate representation of the actual minimum of the problem only when $\mu = 0.01$. Notice also however that unlike the contours in 2.2a, 2.2b and 2.2c, the contours of the QP function in 2.2d are quite elongated and less ellipsoidal which is indicative of a poorly scaled QP function. This results in the ill conditioning of the approximation of the Hessian matrix [130].

The LB method

The log barrier method (LB) was first proposed by Frisch [53]. Fiacco and McCormack [47] later published an in-depth study on the method. Unlike QP, LB is defined for inequality

constraints $c_i(x)$, $i \in I$ only, i.e.

$$\begin{cases} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & c_i(x) \geq 0,\ i \in I. \end{cases} \tag{2.20}$$

This method reformulates (2.20) by introducing a logarithmic term which acts as a barrier, in that it keeps feasible iterates well within the boundary of the feasible region. This LB function is defined as:

$$P(x;\mu) = f(x) - \mu \sum_{i\in I} \log(c_i(x)), \tag{2.21}$$

where μ is the barrier parameter and $-\sum_{i\in I}\log(c_i(x))$ is the logarithmic barrier term. Notice that the barrier parameter appears explicitly as μ in the LB function and not $\frac{1}{\mu}$ as with the penalty parameter in the QP function. We have done this for consistency, because we present the effects of driving μ to 0, on the different penalty methods. By decreasing μ, the minimizer of $P(x;\mu)$, $x(\mu)$, tends to the solution of (2.20). The search for the minimizer $x(\mu)$ can be carried out using any method for unconstrained minimization, provided that the method produces feasible iterates only. Even though (2.21) is defined for inequality constraints only, this method can be modified for equality constraints. Note that $P(x;\mu)$ is undefined at the boundaries of the feasible region, which makes it a non-smooth function. We use Figure 2.3 to illustrate how smaller values for μ tend to yield better approximations of the objective function. Consider the problem

$$\begin{cases} \min & f(x) = (x_1+0.5)^2 + (x_2-0.5)^2 \\ \text{s.t.} & 0 \leq (x_1,x_2) \leq 1, \end{cases} \tag{2.22}$$

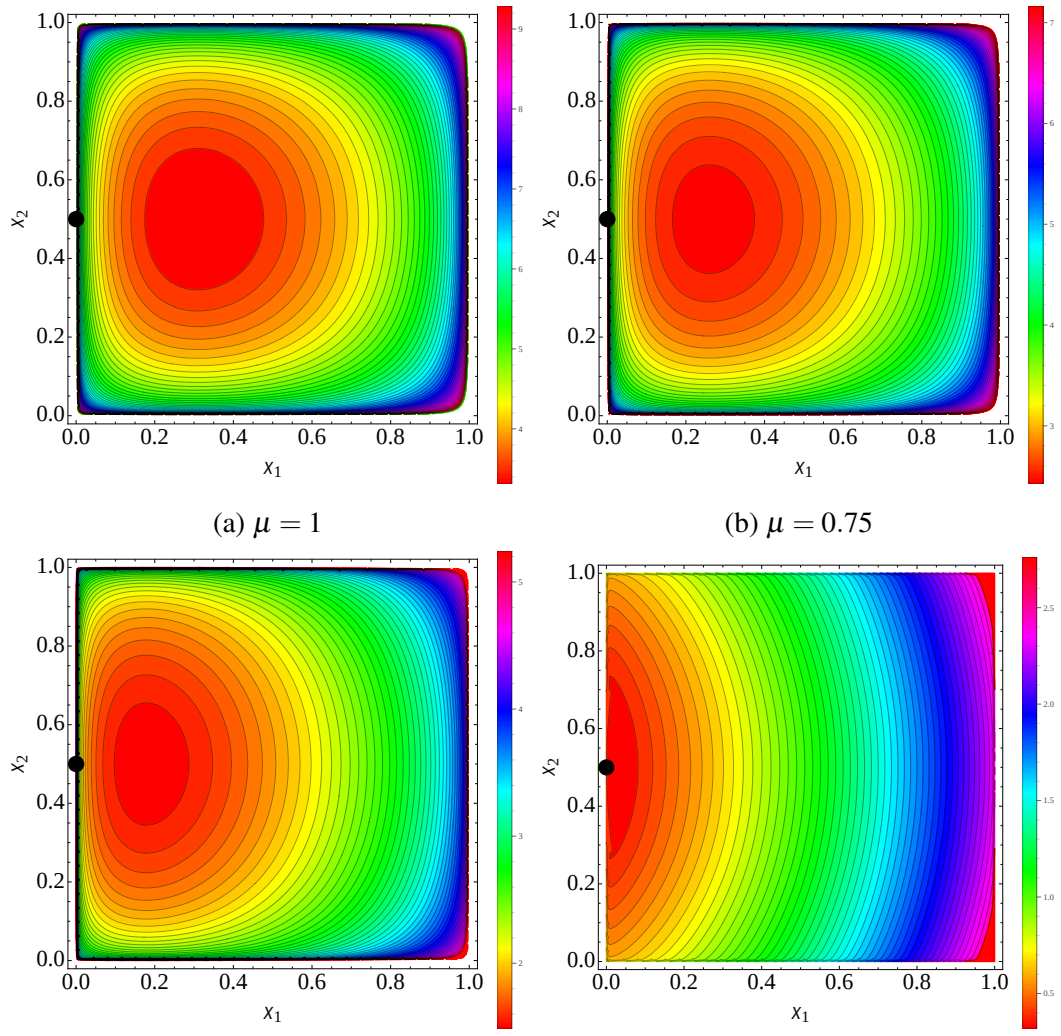
which has solution $x^* = (0,\ 0.5)^T$. The problem (2.22) can be written explicitly as:

$$\begin{cases} \min & f(x) = (x_1+0.5)^2 + (x_2-0.5)^2 \\ \text{s.t.} & c_1(x) = x_1 \geq 0, \\ & c_2(x) = x_2 \geq 0, \\ & c_3(x) = 1-x_1 \geq 0, \\ & c_4(x) = 1-x_2 \geq 0. \end{cases}$$

The corresponding LB function is given by:

$$P(x;\mu) = (x_1+0.5)^2 + (x_2-0.5)^2 - \mu(\log(x_1)+\log(x_2)+\log(1-x_1)+\log(1-x_2))$$

and is illustrated in Figure 2.3 for different values of μ. Each figure also includes the optimal point x^* depicted by the bold black point.



(a) $\mu = 1$ (b) $\mu = 0.75$

(c) $\mu = 0.5$ (d) $\mu = 0.01$

Figure 2.3 Log barrier function for min $(x_1 + 0.5)^2 + (x_2 - 0.5)^2$ subject to $0 \leq x_1, x_2 \leq 1$.

When $\mu = 0.01$, the LB function is a fairly accurate representation of the actual minimum of the problem, see Figure 2.3d, but the same kind of contour elongation which was displayed in Figure 2.2d is displayed. This implies that depending on $f(x)$ and $c_i(x)$, $P(x, \mu)$ may exhibit the same kind of ill conditioning as the QP function, for small values of μ.

The AL method

The augmented Lagrangian method (AL) was first introduced by Hestenes [67] and Powell [107]. Here, the AL function is minimized iteratively where at each iteration the solutions x^k as well as the dual variables λ^k and the parameter μ^k are updated. The choice of AL is

motivated by its smoothness and ability to reduce or eliminate ill-conditioning associated with small values of the penalty parameter μ [130]. As we have seen in Figures 2.2 and 2.3, alternative methods like QP [35, 38, 58, 85, 130], and LB [38, 47, 53, 58, 85, 130] are susceptible to either ill-conditioning, non-smoothness or both of these short-comings [130].

The augmented Lagrangian (AL) for equality constraints is given by:

$$L_A(x,\lambda;\mu) = f(x) - \sum_{i\in E}\lambda_i c_i(x) + \frac{1}{2\mu}\sum_{i\in E} c_i^2(x), \tag{2.23}$$

where λ_i is the classical Lagrangian multiplier and μ is the penalty parameter used to penalize constraint violations [6, 7, 15, 18–20, 28, 38, 58, 59, 67, 85, 87, 107, 110, 130]. Consider again the problem:

$$\begin{cases} \min & f(x) = x_1^2 + 2x_2^2 \\ \text{s.t.} & c_1(x) = x_1 + x_2 - 1 = 0, \end{cases}$$

with solution $x^* = (\frac{2}{3},\ \frac{1}{3})^T$. We use Figure 2.4 to illustrate that, unlike QP and LB, μ does not have to be driven to 0 in order for the AL function to be an accurate representation of the problem. With the fixed value of $\mu = 1$, the AL function adequately represents the problem. Judging by the ellipsoidal contours of the function, L_A is not susceptible to the ill conditioning we have seen in Figures 2.2 and 2.3. This means that when $\mu = 1$, the minimizer corresponding to AL is a fairly good approximation to the minimum of the original problem. To verify this, the constraint $c_1(x)$ as well as the optimal solution x^*, depicted by the bold black point on $c_1(x)$, are included in Figure 2.4.

Although (2.23) is defined for equality constraints only, it can be extended for inequality constraints by introducing the slack variables s_i [130]. These variables convert the inequality constraints

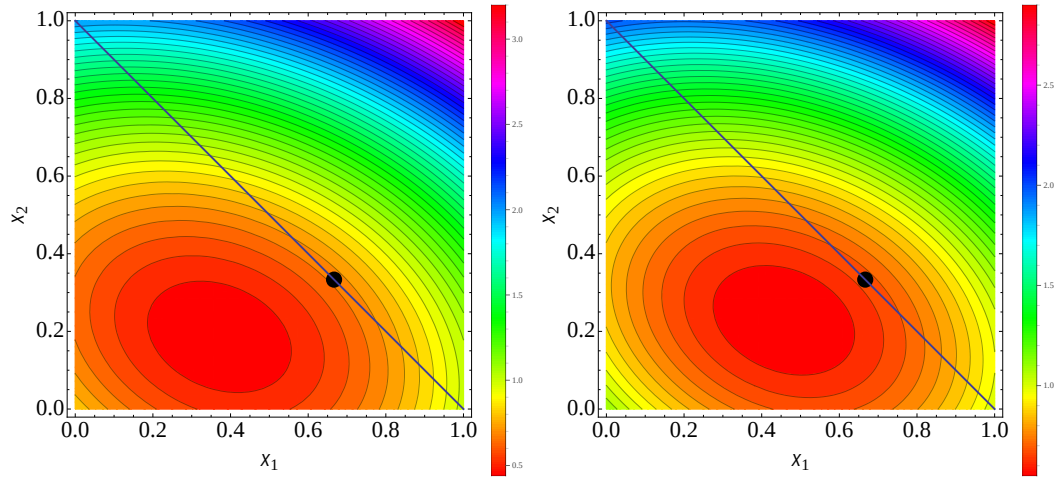
$$c_i(x) \geq 0$$

into equality constraints:

$$c_i(x) - s_i = 0, \quad s_i \geq 0 \quad \forall i \in I. \tag{2.24}$$

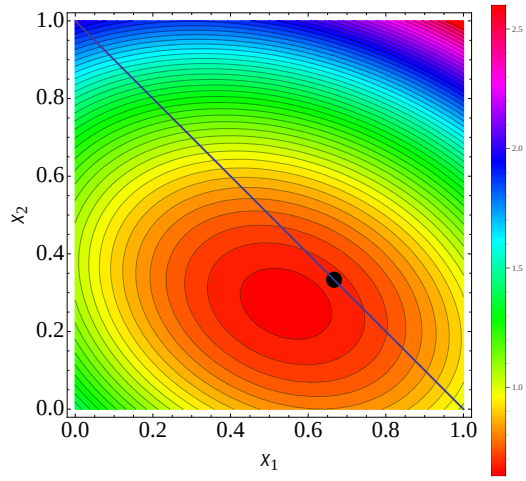
Defining AL in terms of these constraints yields a smooth function which depends on μ^k and λ^k. Iterative methods used to solve (1.2), use the construction of this AL at each k-th iteration:

$$\min_{x,s} f(x) - \sum_{i\in I}\lambda_i^k(c_i(x) - s_i) + \frac{1}{2\mu^k}\sum_{i\in I}(c_i(x) - s_i)^2, \tag{2.25}$$

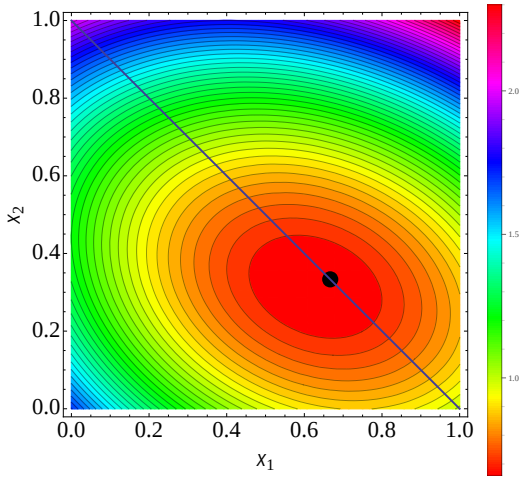


(a) $\lambda = 0.3, \mu = 1$

(b) $\lambda = 0.6, \mu = 1$



(c) $\lambda = 0.9, \mu = 1$



(d) $\lambda = \lambda^* = \frac{4}{3}, \mu = 1$

Figure 2.4 Augmented Lagrangian function for min $x_1^2 + 2x_2^2$ subject to $x_1 + x_2 = 1$, with fixed $\mu = 1$.

$$\text{subject to } s_i \geq 0, \forall i \in I.$$

An optimal value for s_i can then be found analytically since (2.25) is separable in s_i. The solution is given by

$$s_i = \max\{c_i(x) - \mu^k \lambda_i^k, 0\} \quad \forall i \in I, \tag{2.26}$$

where the positive restriction on s_i follows from (2.24). Combining (2.25) and (2.26), the following AL function for inequality constraints is obtained

$$Ł_A(x, \lambda^k; \mu^k) = f(x) + \begin{cases} \sum\limits_{i \in I} -\lambda_i^k c_i(x) + \sum\limits_{i \in I} \frac{1}{2\mu^k} c_i^2(x), & c_i(x) - \mu^k \lambda_i^k \leq 0; \\ \sum\limits_{i \in I} -\frac{\mu^k}{2} (\lambda_i^k)^2, & c_i(x) - \mu^k \lambda_i^k > 0. \end{cases} \tag{2.27}$$

A strong argument supporting the derivation of this AL can be found in Nocedal and Wright [92].

Suppose one were to minimize (2.27) up to the k-th iteration, where x^k, λ_i^k and μ^k are known. Conventionally, the iterative procedure progresses by updating (2.27) with λ^k and μ^k and then minimizing (2.27) again to get x^{k+1}. One then updates λ_i^{k+1} and μ^{k+1} using appropriate formulae and the iterative procedure continues [130]. The method proposed in this thesis, however, uses a different strategy. Here, x^k, λ_i^k and μ^k, are simultaneously updated in the optimization of (2.27).

It is important to note that (2.27) is a smooth function with respect to x, but a discontinuity in the Hessian with respect to x occurs whenever $c_i(x^k) = \mu^k \lambda_i^k$ for some $i \in I$ [130]. If strict complementarity holds, then this problem is circumvented. To verify this, consider the case when $c_i(x^k)$ is active. By strict complementarity the corresponding Lagrange multiplier λ_i^k is non-zero. When $c_i(x^k)$ is inactive, for k large enough $\lambda_i^k \approx 0$. In both cases $c_i(x^k) = \mu^k \lambda_i^k$ is not satisfied and the discontinuous region is avoided [130]. The trajectory-based method which we develop in this thesis does not require second-order information, so the discontinuity in the Hessian does not impact on the solution process.

The AL function in (2.27) is not explicitly defined for active or inactive inequality constraints. To investigate which inequality, i.e. $c_i(x) \leq \mu^k \lambda_i^k$ and $c_i(x) > \mu^k \lambda_i^k$ corresponds to which type of inequality constraints, we define the following set

$$M(x^k) = \{i \in I | c_i(x^k) - \mu^k \lambda_i^k \leq 0\}.$$

Since weakly active constraints do not impact on the minimization of $f(x)$, we focus on constraints which are strongly active. By extension, the equality $\mu^k \lambda_i^k = c_i(x^k)$ can not be satisfied. We therefore assume strict complementarity and redefine $M(x^k)$ as follows:

$$M(x^k) = \{i \in I | c_i(x^k) - \mu^k \lambda_i^k < 0\}, \tag{2.28}$$

where $c_i(x^k) < \mu^k \lambda_i^k$ can be satisfied in two ways. The first is with $\lambda_i^k > 0, i \in I$. By the strict complementarity assumption, it follows that $c_i(x^k)$ is active, i.e. $c_i(x^k) = 0$. The second way in which $c_i(x^k) < \mu^k \lambda_i^k$ can be satisfied, is with $\lambda_i^k = 0, i \in I$. An implication of this is that $c_i(x^k)$ is infeasible, i.e. $c_i(x^k) < 0$. $M(x^k)$ therefore contains all strongly active inequality constraints, as well as inequality constraints which are violated at x^k, should they exist. In Chapters 5 and 8, we develop trajectory-based algorithms, based on AL, for which local and global convergence properties are established (in Chapters 7 and 8). As $k \to \infty$, therefore, we may assume that all constraints which are initially violated in AL, converge to feasibility. For k large enough, we may therefore define $M(x^k)$ as the set of constraints which are strongly active at some feasible x:

$$M(x) = \{i \in I | c_i(x) \in A_s(x)\}.$$

We now consider the other set of constraints in (2.27), i.e. the set of constraints which satisfy the following inequality

$$c_i(x^k) > \mu^k \lambda_i^k, \ i \in I.$$

Since $\lambda_i \geq 0, \ \forall i \in I$, the only way that the condition $c_i(x^k) > \mu^k \lambda_i^k$ can be satisfied, is if $c_i(x^k)$ is inactive, i.e. $c_i(x^k) > 0$. The set of inequality constraints corresponding to AL in equation (2.27) can thus neatly be defined in terms of the two sets $A_s(x)$ and $I \setminus A_s(x)$:

$$Ł_A(x, \lambda^k; \mu^k) = f(x) + \begin{cases} \sum\limits_i -\lambda_i^k c_i(x) + \sum\limits_i \frac{1}{2\mu^k} c_i^2(x), & i \in I \cap A_s(x); \\ \sum\limits_i -\frac{\mu^k}{2} (\lambda_i^k)^2, & i \in I \setminus A_s(x). \end{cases} \tag{2.29}$$

Observe that for all inequality constraints, $c_i(x)$, such that $i \in I \cap A_s(x)$, equation (2.29) is exactly the same as AL in (2.23) for $c_i(x), \ i \in E$. Since for any x, the constraints, $c_i(x)$, are either strongly active at x i.e. $i \in A_s(x)$, or inactive, i.e. $i \in I \setminus A_s(x)$, we now define a general AL, where constraints of all types at x are integrated:

$$\phi_A(x, \lambda; \mu) = f(x) - \sum_{i \in E \cup (I \cap A_s(x))} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i \in E \cup (I \cap A_s(x))} c_i^2(x) + \psi(x, \lambda; \mu) \tag{2.30}$$

where

$$\psi(x,\lambda;\mu) = -\sum_{i\in I\setminus A_s(x)} \frac{\mu}{2}{\lambda_i}^2,$$

The second and third terms containing index sets corresponding to active inequality constraints are identical to that of the AL function in (2.23). The last term in (2.30) corresponds to all inactive inequality constraints, contained in $I \setminus A_s(x)$. In the implementation of AL, all types of constraints are included in the summation in (2.30), but for the theoretical discussion, we write (2.30) excluding weakly active constraints.

Many of the problems considered in this thesis contain a combination of equality and inequality constraints. Since all the inequality constraints have been converted to equality constraints, we can define the general AL function as (2.30). The multiplier estimates for inequality constraints however, must be strictly non-negative. We therefore still treat inequality and equality constraints separately. This is to make sure that $\lambda_i \not< 0$ for inequality constraints.

The penalty parameter μ in (2.30), is conventionally updated using various techniques to ensure that constraint violation is penalized at each iteration, particularly if maintaining feasibility is an important criteria for the solution process [130]. In the current literature μ is either decreased by a factor γ or kept constant, depending on whether or not constraint violation is improved [6, 7], [18, 19], [20]. In this thesis, however, we update μ by solving a differential equation. The details of this penalty parameter update scheme are provided in Sections 5.1 and 5.2.

We now briefly describe some popular algorithms for CNLPs.

2.2.2 Algorithms for CNLPs

Once the problem (1.2) is reformulated into either (2.18), (2.21) or (2.30), practical iterative methods are used to locate an optimal solution x^* [60], [130]. The general framework of the solution process is to formulate one of these subproblems at the kth iteration and carry out the minimization with respect to x using methods for UNLPs. We take a brief look at two of the most popular methods for CNLPs, namely interior point methods [25], [93], [130] and sequential quadratic programming methods (SQP) [24, 45, 61, 86, 95, 96, 130, 104–106, 128].

Interior point methods

Interior point methods solve a special case of (1.2):

$$\begin{cases} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & c_i(x) \geq 0,\ i = 1,\ldots,m_I, \\ & Ax = b, \end{cases} \tag{2.31}$$

where $f : X \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$ and $c : X \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}^{m_I}$ are convex and twice continuously differentiable, $b \in \mathbb{R}^{m_e}$ and $A \in \mathbb{R}^{m_E \times n}$, with $\text{rank}(A) = m_E < n$. The problem is assumed strictly feasible, i.e. all equality constraints are satisfied and all inequality constraints are satisfied as $c_i(x) > 0, \forall i \in I$. Interior point methods are designed to solve linearly constrained problems with both equality and inequality constraints by reformulating the original problem into a sequence of equality constrained problems [25]. We present a particular interior-point method, namely the barrier method. The basic idea of the barrier method is to reformulate (2.31) into the LB function, i.e. the function presented by (2.21), subject to the equality constraint $Ax = b$:

$$\begin{cases} \min\limits_{x\in\mathbb{R}^n} & P(x;\mu) = f(x) - \mu \sum\limits_{i\in I} \log(c_i(x)) \\ \text{s.t.} & Ax = b. \end{cases} \tag{2.32}$$

Since (2.32) is twice continuously differentiable, an adaptation of Newton's method for equality constrained problems can be applied to solve it [25]. The problem (2.32) is just an approximation of problem (2.31). As we have demonstrated in Section 2.2, as μ^k decreases this approximation become more accurate, but ill conditioning can occur when μ^k is small.

Within the algorithmic framework, a sequence of problems of the form (2.32) are solved. At each iteration μ^k is gradually decreased. At the $(k+1)$-th iteration, Newton's method is initialized from the previous solution, x^k, obtained using μ^k. The barrier parameter, μ^{k+1}, is then updated and x^{k+1} is found using Newton's method. The iterative procedure continues in this way until the optimal solution is found.

An alternative interior-point method is known as the primal-dual interior-point method. Primal-dual interior-point methods are usually more accurate than barrier methods since they exhibit better convergence [25]. We now consider one such method.

Primal-Dual Interior Point Method for CNLPs

We derive the primal-dual interior point method for nonlinear programming, based on the assumption that $f, c_i,\ i \in E \cup I$ are twice continuously differentiable. We also assume that strong duality holds. We now describe a feasible primal-dual interior point method for solving $\bar{\mathscr{P}}$, where feasibility of the iterates is maintained throughout the solution process. Here $\bar{\mathscr{P}}$ is defined as:

$$\bar{\mathscr{P}} \begin{cases} \min\limits_{x \in \mathbb{R}^n} & f(x), \\ \text{s.t.} & c_i(x) = 0, i \in E, \\ & c_i(x) \geq 0, i \in I. \end{cases} \tag{2.33}$$

The Lagrangian $\bar{L}$ is defined as follows:

$$\bar{L}(x, \nu, \lambda) = f(x) - c_E(x)^T \nu - c_I(x)^T \lambda, \tag{2.34}$$

where $c_E(x) = (c_i(x))_{i \in E}$ and $c_I(x) = (c_i(x))_{i \in I}$ are equality and inequality constraints respectively, and ν and λ are the corresponding Lagrange multipliers. The first and second derivatives of L with respect to x are defined respectively, as:

$$\nabla_x L(x, \nu, \lambda) = \nabla f(x) - J_E(x)\nu - J_I(x)\lambda,$$

and

$$\nabla_{xx} L(x, \nu, \lambda) = \nabla^2 f(x) - \sum_{i \in E} \nu_i \nabla^2 c_i(x) - \sum_{i \in I} \lambda_i \nabla^2 c_i(x),$$

where J_E and $J_I(x)$ are the Jacobian matrices of equality and inequality constraints respectively:

$$J_E(x) = (\nabla c_1(x), \nabla c_2(x), \ldots, \nabla c_{m_E}(x))^T, \tag{2.35}$$

$$J_I(x) = (\nabla c_1(x), \nabla c_2(x), \ldots, \nabla c_{m_I}(x))^T, \tag{2.36}$$

and $J_E(x) \in \mathbb{R}^{m_E \times n}$, $m_E < n$ and $J_I(x) \in \mathbb{R}^{m_I \times n}$.

The *KKT* conditions for $\mathscr{P}$ are:

(2.35a) $\nabla_x L(x^*, \nu^*, \lambda^*) = 0,$

(2.35b) $c_E(x^*) = 0,$

(2.35c) $C(x^*)\lambda^* = 0,$

(2.35d) $\lambda^* \geq 0,\ c_I(x^*) \geq 0,$

where $C(x^*) = diag(c_I(x^*))$.
The perturbed *KKT* system $KKT(\mu^k),\ \mu^k > 0$ is then defined as:

(2.36a) $\nabla_x L(x^*, \nu^*, \lambda^*) = 0,$

(2.36b) $c_E(x^*) = 0$

(2.36c) $C(x^*)\lambda^* - \mu^k e = 0,$

(2.36d) $\lambda^* \geq 0,\ c_I(x^*) \geq 0,$

where $e = (1, 1, \ldots, 1)^T \in \mathbb{R}^{m_E}$.
The primal-dual interior point method approximately solves the sequence of systems $KKT(\mu^k)$ in the variable $\{x, \nu, \lambda\}$, for a sequence of positive values $\mu^k > 0$, where $\mu^k \to 0$ as $k \to \infty$. For each parameter μ^k, Newton's method approximates the solution (x^k, ν^k, λ^k) of the system of equations defined by $(2.36a) - (2.36c)$. To maintain feasibility, the Newton iterates are chosen such that $(2.36d)$ is satisfied. In this way, (x^k, ν^k, λ^k) converges to the solution x^*, ν^*, λ^* as $k \to \infty$ [92], [108].

We obtain the Newton search directions $(\Delta x, \Delta\nu, \Delta\lambda)$ for the system *KKT*(μ^k), by solving

$$\begin{bmatrix} \nabla^2_{xx} L(x) & -(J_E^k)^T & (J_I^k)^T \\ J_E^k & 0 & 0 \\ \Lambda J_I^k & C & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\nu \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla_x L \\ c_E(x) \\ C\lambda - \mu^k e \end{bmatrix}, \tag{2.37}$$

where $\Lambda = diag(\lambda)$. Given a point (x^k, ν^k, λ^k), Newton's method computes the new iterate $(x^{k+1}, \nu^{k+1}, \lambda^{k+1})$ as follows:

$$x^{k+1} = x^k + \alpha\Delta x,$$

$$\nu^{k+1} = \nu^k + \alpha\Delta\nu$$

$$\lambda^{k+1} = \lambda^k + \alpha\Delta\lambda,$$

where $(\Delta x, \Delta\nu, \Delta\lambda)$ is given by (2.37) and α is chosen such that condition $(2.36d)$ is satisfied. A suitable choice for α is:

$$\alpha = \min\{1, \tau^k \times \min_{i \in I}\{-\frac{\lambda_i}{\Delta\lambda_i} : (\Delta\lambda)_i < 0\}, \tau_k \min_{i \in I} \alpha^i\}$$

where $\alpha^i \leq \min\{\beta > 0 : c_i(x^k + \beta\Delta x) = 0\}$ [108], [93].

Sequential Quadratic Programming (SQP)

SQP methods were first proposed by Wilson [128] and was later developed by Garcia et al.[86], Han [95, 96] and Powell [104–106]. SQP methods are second order methods as they require second derivative information. They construct a series of quadratic approximations to (1.2) iteration by iteration. We consider first the SQP method for equality constrained problems. The basic idea is to solve a problem consisting of a quadratic approximation of the objective function subject to linear approximations of the constraints, at every iteration. The subproblem at the current iteration x^k is given by:

$$\begin{cases} \min\limits_{d} & \frac{1}{2}d^T W^k d + (\nabla f^k)^T d \\ \text{s.t.} & (\nabla c_i^k)^T d + (c_i^k) = 0,\ i \in E, \end{cases} \tag{2.38}$$

where $W^k = \nabla^2_{xx} L(x^k, \lambda^k), \nabla f^k = \nabla f(x^k)$, $\nabla c_i^k = \nabla c_i(x^k)$, $c_i^k = c_i(x^k)$ and L is defined as in (2.2). The iterate generated by solving (2.38) is equivalent to the iterate generated by applying Newton's method to the *KKT conditions* described in **Theorem 2.1.6** [130]. To demonstrate this, let $J(x)$ denote the Jacobian matrix of the constraints:

$$J(x) = (\nabla c_1(x), \nabla c_2(x), \ldots, \nabla c_m(x))^T, \tag{2.39}$$

where $J \in \mathbb{R}^{m \times n}$, $m < n$. The problem (2.38) can now be written equivalently as

$$\begin{cases} \min\limits_{d} & \frac{1}{2}d^T W^k d + (\nabla f^k)^T d \\ \text{s.t.} & J^k d + c^k = 0 \end{cases} \tag{2.40}$$

where $J^k = J(x^k)$ and $c^k = c(x^k)$ is the vector made up of components $c_i(x^k), i \in E$. Denote the Lagrangian of (2.40) by

$$L = \frac{1}{2}d^T W^k d + (\nabla f^k)^T d - \Lambda^T (J^k d + c^k), \tag{2.41}$$

where Λ is the Lagrangian multiplier. By applying the *KKT conditions* to (2.41), and assuming that $J(x)$ has full row rank, (2.40) has a unique solution (d^k, Λ^k) that satisfies

$$\begin{aligned} &W^k d^k + \nabla f^k - (J^k)^T \Lambda^k = 0, \\ &J^k d^k + c^k = 0, \end{aligned} \tag{2.42}$$

where the first equation in (2.42) corresponds to $\nabla_d L = 0$ and the second equation in (2.42) corresponds to $\nabla_\Lambda L = 0$.

We can rewrite (2.42) as

$$\begin{bmatrix} W^k & -(J^k)^T \\ J^k & 0 \end{bmatrix} \begin{bmatrix} d^k \\ \Lambda^k \end{bmatrix} = - \begin{bmatrix} \nabla f^k \\ c^k \end{bmatrix}. \tag{2.43}$$

Now recall that the Lagrangian function for problem (1.2) is defined by (2.2). By applying the *KKT conditions* to (2.2) with equality constraints only, we obtain the following system:

$$\nabla L = \begin{bmatrix} \nabla_x L \\ \nabla_\lambda L \end{bmatrix} = \begin{bmatrix} \nabla f(x) - \nabla c(x)\lambda \\ c(x) \end{bmatrix} = 0. \tag{2.44}$$

We can define (2.44) as a function of x and λ

$$F(x,\lambda) = \begin{bmatrix} \nabla f(x) - \nabla c(x)\lambda \\ c(x) \end{bmatrix} = 0, \tag{2.45}$$

and apply Newton's method to obtain a solution to (2.45), by solving the following system

$$F(x^k,\lambda^k) + (\nabla F(x^k,\lambda^k))^T \begin{bmatrix} d_x^k \\ d_\lambda^k \end{bmatrix} = 0. \tag{2.46}$$

The left hand side of (2.46) expands to

$$\begin{bmatrix} \nabla f^k - \nabla c^k \lambda^k \\ c^k \end{bmatrix} + \left[\nabla \begin{pmatrix} \nabla_x L \\ \nabla_\lambda L \end{pmatrix}\right]^T \begin{bmatrix} d_x^k \\ d_\lambda^k \end{bmatrix}$$

$$= \begin{bmatrix} \nabla f^k - \nabla c^k \lambda^k \\ c^k \end{bmatrix} + \begin{bmatrix} \nabla^2_{xx} L & \nabla^2_{\lambda x} \\ \nabla^2_{x\lambda} L & \nabla^2_{\lambda\lambda} L \end{bmatrix}^T \begin{bmatrix} d_x^k \\ d_\lambda^k \end{bmatrix}$$

$$= \begin{bmatrix} \nabla f^k - \nabla c^k \lambda^k \\ c^k \end{bmatrix} + \begin{bmatrix} W^k & \nabla c^k \\ (-\nabla c^k)^T & 0 \end{bmatrix}^T \begin{bmatrix} d_{x^k} \\ d_\lambda^k \end{bmatrix}$$

$$= \begin{bmatrix} \nabla f^k - \nabla c^k \lambda^k \\ c^k \end{bmatrix} + \begin{bmatrix} W^k & -\nabla c^k \\ (\nabla c^k)^T & 0 \end{bmatrix} \begin{bmatrix} d_{x^k} \\ d_\lambda^k \end{bmatrix}.$$

Now, obtaining a solution to (2.46) is equivalent to obtaining a solution to the following system

$$\begin{bmatrix} W^k & -(J^k)^T \\ J^k & 0 \end{bmatrix} \begin{bmatrix} d_x^k \\ d_\lambda^k \end{bmatrix} = \begin{bmatrix} -\nabla f^k + (J^k)^T \lambda^k \\ -c^k \end{bmatrix}, \tag{2.47}$$

where d_x^k and d_λ^k correspond to the step size in the x and λ directions respectively. The Newton step from the current iterate (x^k, λ^k) is given by

$$\begin{bmatrix} x^{k+1} \\ \lambda^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ \lambda^k \end{bmatrix} + \begin{bmatrix} d_x^k \\ d_\lambda^k \end{bmatrix}. \tag{2.48}$$

By subtracting $(J^k)^T \lambda^k$ from both sides of the first equation in (2.47), we obtain

$$\begin{bmatrix} W^k & -(J^k)^T \\ J^k & 0 \end{bmatrix} \begin{bmatrix} d_x^k \\ \lambda^{k+1} \end{bmatrix} = - \begin{bmatrix} \nabla f^k \\ c^k \end{bmatrix}, \tag{2.49}$$

which is equivalent to (2.43), with $d_x^k = d^k$ and $\lambda^{k+1} = \Lambda^k$. This relationship is referred to as the equivalence between SQP and Newton's method and is particularly useful because it enables convergence properties for SQP methods to be easily established. Provided an optimal solution (x^*, λ^*) exists, convergence of the SQP method is guaranteed [130]. Once (1.2) has been approximated by (2.38), quadratic programming algorithms, such as gradient- projection methods [25, 130], range-space methods [25, 130], interior point methods [25, 50, 51, 92, 98, 103, 130], and active set methods [25, 50, 51, 92, 130, 98] can be used to locate an approximate minimizer at each iteration. SQP methods are globalized by incorporating either line search or trust region strategies into the SQP algorithm [25].

Remark 2.2.1. Notice that with the interior point method, at x^k one has μ^k which is then used to get x^{k+1}, where μ^k is updated again to get μ^{k+1}. The process continues until x^* is found. In the primal-dual interior point method as well as SQP however, one calculates $\{x^k, \lambda^k\}$ iteratively until $\{x^*, \lambda^*\}$ are found. The trajectory-based methods developed in this thesis follow a strategy similar to that of the primal-dual interior point method and SQP in that x^k, λ^k and μ^k are updated simultaneously. Furthermore, as with the Primal-dual interior point method, the trajectory-based methods developed in this thesis are primal-dual in the continuous space.

Chapter 3

Review of Some Solution Techniques for MINLPs

In this chapter, we provide a review of various existing methods for mixed integer nonlinear programming problems. In Section 3.1, we present various types of mixed integer problems. Section 3.2 introduces CNLP subproblems pertinent to the discussion in the remainder of the chapter. In Section 3.3 we present an important MILP problem used in the iterative scheme of various MINLP algorithms. A review on some of the existing methods for convex MINLPs and non-convex MINLPs can be found in Sections 3.4 and 3.5 respectively.

3.1 Mixed integer nonlinear programming

When there are both integer as well as continuous variables present in an optimization problem, this is called a mixed integer problem. The problem is classified as either linear, quadratic or nonlinear, depending on the nature of the objective function and constraints. If the objective function and constraints are all linear, then this is referred to as a mixed integer linear programming problem (MILP). The general MILP can be written as

$$\begin{cases} \min\limits_{z} & \underline{q}^T z, \\ \text{st} & c_i(z) = 0, i \in E, \\ & c_i(z) \geq 0, i \in I, \\ & z \in X \times Y, \end{cases} \tag{3.1}$$

where $c : X \times Y \longrightarrow \mathbb{R}^m$ are real valued, linear functions, $\underline{q} \in \mathbb{R}^n$, z is the vector made up of continuous and integer variables, $z = (x,\ y)^T$, and X and Y are defined as in (1.3).

If the objective function is quadratic and the constraints are linear, then this is referred to as a mixed integer quadratic programming problem (MIQP). The general MIQP can be written as

$$\begin{cases} \min\limits_{z} & q(z) = \frac{1}{2} z^T H z + g^T z \\ \text{st} & c_i(z) = 0, i \in E, \\ & c_i(z) \geq 0, i \in I, \\ & z \in X \times Y, \end{cases} \tag{3.2}$$

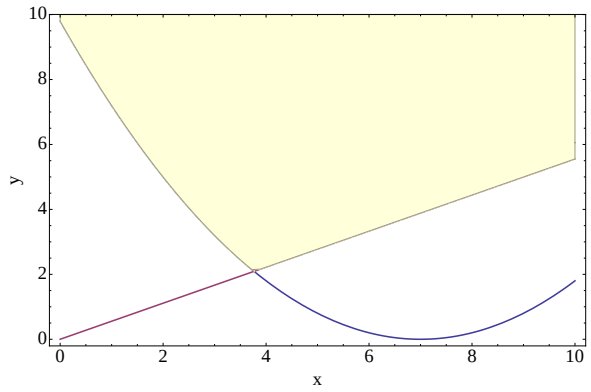
where $H \in \mathbb{R}^{n \times n}$ is a real, symmetric matrix, $g \in \mathbb{R}^n$, and z and c are defined as in (3.1).
When either the objective function or the constraints are nonlinear, this is referred to as the MINLP described by (1.3):

$$\mathscr{M} \begin{cases} \min\limits_{z} & f(z), \\ \text{s.t.} & c_i(z) = 0, i \in E, \\ & c_i(z) \geq 0, i \in I, \\ & z \in X \times Y. \end{cases} \tag{3.3}$$

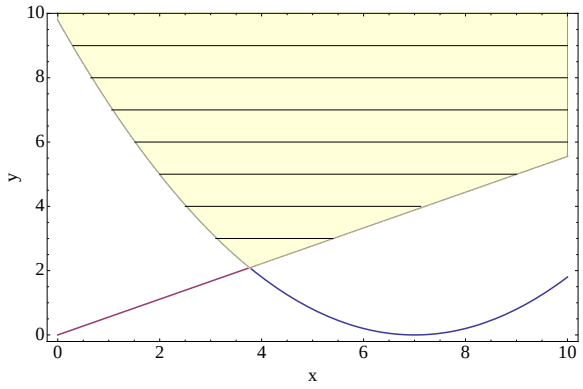
We now demonstrate the effect of the continuous and integer variables on the feasible region of a problem. Let the constraints of a problem be defined as follows:

$$\begin{aligned} c_1(x,y) &= 5y - (x-7)^2 \geq 0, \\ c_2(x,y) &= 1.8y - x \geq 0. \end{aligned} \tag{3.4}$$

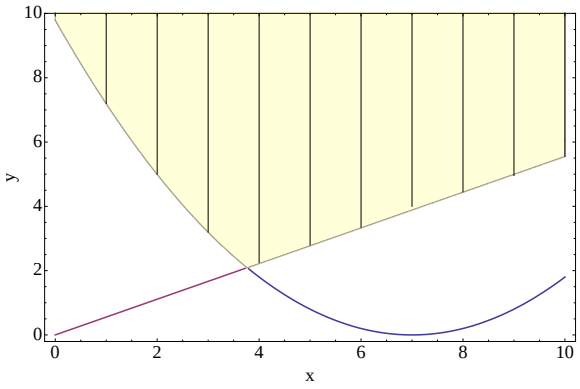
The feasible region formed by the constraints is depicted in Figure 3.1. If both variables are continuous, then the feasible region is represented by the shaded region in Figure 3.1a. When x is continuous and y is discrete, the feasible region of the MINLP is represented by the parallel horizontal lines in Figure 3.1b. If x is the discrete variable and y is continuous, then the feasible region of the MINLP would be represented by the parallel vertical lines in Figure 3.1c. If x and y are both restricted to integer values, probably the most complex problem to solve, then the problem is formally known as an integer nonlinear programming problem (INLP). The feasible region of this problem is represented by the dotted points in Figure 3.1d.
As far as the optimality conditions are concerned, it is not possible to apply the same optimality conditions to MINLPs as those used for CNLPs. For instance, the condition $\nabla L = 0$, usually corresponds to a solution which is continuous and not necessarily integer. Simple optimality conditions are therefore not easily constructed for MINLPs. Figure 3.2 illustrates this. Consider the problem of minimizing $f(x,y) = \frac{(x-3)^2 - 10x}{3x+y+1}$ subject to the constraints in equation (3.4). Figure 3.2 represents the solution of this problem as a CNLP,



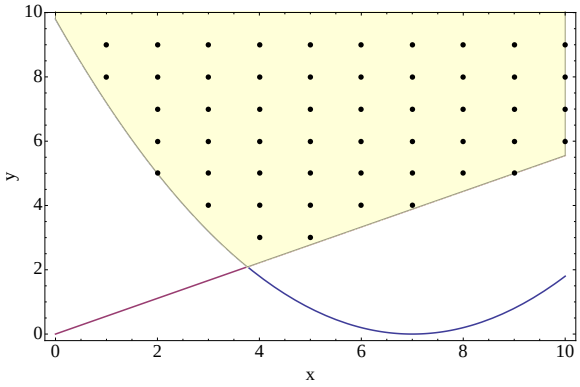
(a) The feasible region for CNLP



(b) The feasible region for MINLP



(c) The feasible region for MINLP



(d) The feasible region for INLP

Figure 3.1 Feasible regions of CNLPs, INLPs and MINLPs

INLP and MINLP. The difference between the CNLP solution and the other solutions is depicted by the enlarged red circular point and the enlarged green triangular point respectively in each figure. We notice that the solutions are very different and that the same optimality conditions can not be applied to the separate problems.

A number of existing algorithms for solving MINLPs with convex and/or pseudo-convex objective and constraint functions can be found in the literature. Most prominent of these methods are the branch and bound method (BB) [36], the generalized benders decomposition method (GBD) [55], the outer-approximation method (OA) [41], [49] the branch and bound integrated with SQP method (sequential quadratic programming) [80], and the extended cutting plane method (ECP) [125]. These methods rely on a number of subproblems which we briefly present now.

3.2 Nonlinear programming subproblems

There are three important CNLP subproblems associated with the solution process for convex MINLPs. These are given below.

The first CNLP subproblem is obtained by continuous relaxation. The CNLP relaxation of $\mathscr{M}$, defined by (3.3), is the problem $\underline{\mathscr{M}}$ defined as:

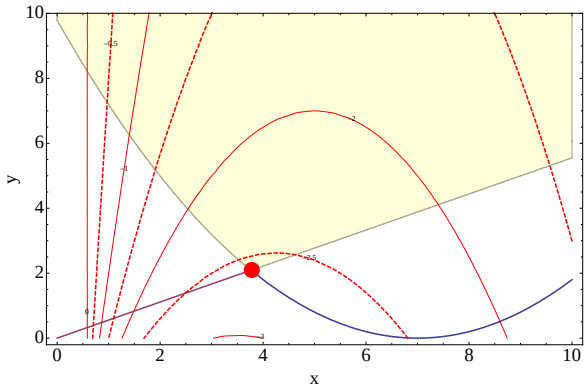
$$\underline{\mathscr{M}} \begin{cases} \min\limits_{x,y} & f(x,y) \\ \text{s.t.} & c_i(x,y) = 0, i \in E, \\ & c_i(x,y) \geq 0, i \in I, \\ & x \in X, y \in \text{ conv}(Y), \end{cases} \tag{3.5}$$

where conv$Y$ is the convex hull of Y. This subproblem is generally solved for each k-th step of a BB search by adding an additional bound constraint to each node of the BB tree [63], [50].

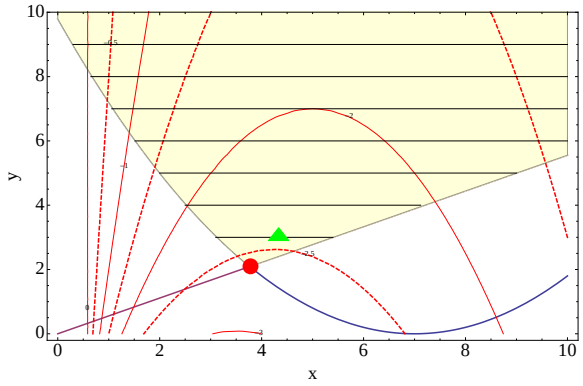
The second CNLP subproblem is obtained by fixing the integer variables in $\mathscr{M}$. For $y^k \in Y$, the following subproblem is obtained by fixing the integer variables y equal to the value y^k:

$$\bar{\mathscr{M}}(y^k) \begin{cases} \min\limits_{x} & f(x,y^k) \\ \text{s.t.} & c_i(x,y^k) = 0, i \in E, \\ & c_i(x,y^k) \geq 0, i \in I, \\ & x \in X. \end{cases} \tag{3.6}$$

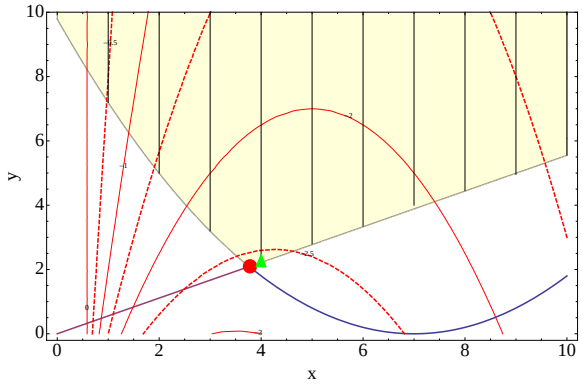
If $f(x,y)$ and $c_i(x,y)$ are convex then $\underline{\mathscr{M}}$ provides an absolute lower bound and $\bar{\mathscr{M}}(y^k)$ an upper bound to $\mathscr{M}$. $\bar{\mathscr{M}}(y^k)$ provides an upper bound provided it has a feasible solution, i.e.



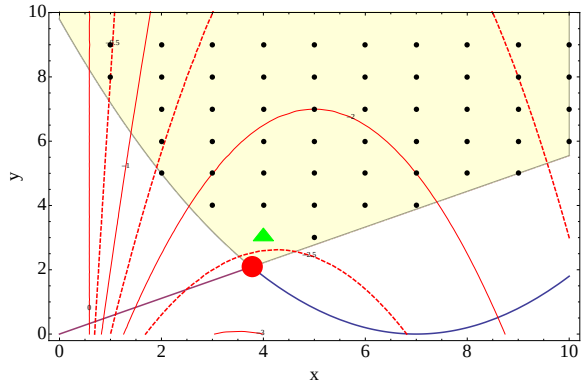
(a) The optimal solution of a CNLP



(b) The different optimal solutions of a CNLP and a MINLP problem



(c) The different optimal solutions of a CNLP and a MINLP



(d) Different optimal solutions of a CNLP and an INLP

Figure 3.2 Optimal solutions of CNLPs, INLPs and MINLPs

a feasible x^k such that the constraints in (3.6) are satisfied [63]. When this is not the case, the third CNLP subproblem, known as the feasibility subproblem, is considered:

$$CNLPF(y^k)\begin{cases} \min\limits_{x\in X} & \sum_{i=1}^{|I|}\alpha_i \\ \text{s.t.} & c_i(x,y^k)=0, \quad i\in E, \\ & c_i(x,y^k)\geq \alpha_i, \quad i\in I, \\ & \alpha_i\geq 0, \end{cases} \tag{3.7}$$

where $|I| = m_I$. Notice that $\sum_{i=1}^{|I|}\alpha_i = 0$ ensures feasibility. The subproblems $CNLPF(y^k)$ and $\bar{\mathscr{M}}(y^k)$ are used by OA [41], [49], [63] and GBD [50], [63]. The above problem can be re-written as:

$$CNLPF1(y^k)\begin{cases} \min\limits_{x\in X} & \sum_{i=1}^{|I|}c_i^+(x,y^k) \\ \text{s.t.} & c_i(x,y^k)=0, \quad i\in E, \end{cases} \tag{3.8}$$

where

$$c_i^+(x,y^k) = \max\{0,-c_i(x,y^k)\},\ i\in I.$$

An l_∞-minimization problem is also defined as:

$$CNLPF2(y^k)\begin{cases} \min\limits_{x\in X}\max\limits_{i\in I} & c_i^+(x,y^k) \\ \text{s.t.} & c_i(x,y^k)=0, \quad i\in E. \end{cases} \tag{3.9}$$

The feasibility problem can be formulated from another point of view where the l_1-minimization is constructed using only the violated inequality constraints, e.g.

$$CNLPF3(y^k)\begin{cases} \min\limits_{x\in X} & \sum\limits_{i\in J}c_i^+(x,y^k) \\ \text{s.t.} & c_i(x,y^k)=0, \quad i\in E, \\ & c_i(x,y^k)\geq 0, \quad i\in I\setminus J, \end{cases} \tag{3.10}$$

where J is the index set which corresponds to infeasible inequality constraints.

A general form of the infeasibility problem suggested by Fletcher and Leyffer [49], [50] is as follows:

$$FP(y^k)\begin{cases} \min\limits_{x\in X} & \sum\limits_{i\in J}\omega_i c_i^+(x,y^k) \\ \text{s.t.} & c_i(x,y^k)=0, \quad i\in E, \\ & c_i(x,y^k)\geq 0, \quad i\in I\setminus J, \\ & \omega_i\geq 0. \end{cases} \tag{3.11}$$

We now consider the MILP used within the solution process of some MINLP algorithms.

3.3 The mixed integer linear programming problem (MILP)

Many optimization models have objective functions and constraints which are linear with respect to the continuous and integer variables [50]. These are referred to as mixed integer linear programming problems (MILPs). Some convex MINLPs or pseudo-convex MINLPs with pseudo-convex objective functions and convex feasible regions are approximated and solved using MILPs [41, 49, 125, 127].
For convex MINLPs, one can exploit the convexity of the nonlinear functions by replacing them with cutting planes. For example, at the k-th iteration of the OA algorithm, a solution $(x^{k+1}, y^{k+1})^T$ is generated by solving an MILP, where cutting planes are used as constraints. We now define this MILP denoted by M-OA^k:

$$\text{M-OA}^k \begin{cases} \min & \alpha \\ \text{s.t.} & f(x^k,y^k) + (\nabla f(x^k,y^k))^T \begin{pmatrix} x-x^k \\ y-y^k \end{pmatrix} \leq \alpha \quad \forall k \in K, \\ & c_i(x^k,y^k) + (\nabla c_i(x^k,y^k))^T \begin{pmatrix} x-x^k \\ y-y^k \end{pmatrix} \geq 0, \quad \forall k \in K, \\ & x \in X, \\ & y \in Y, \\ & \alpha \in \mathbb{R}^+. \end{cases} \tag{3.12}$$

This MILP problem is obtained by adding recent cutting planes to the previous MILP problem. The MILP problem is based on $|K|$ points $(x^k,y^k)^T$ $(k = 1,2,\ldots,|K|)$ generated at $|K|$ previous steps, i.e. $k \in K = \{1,2,\ldots|K|\}$. OA generates the point $(x^{k+1},y^{k+1})^T$ at the k-th iteration, by solving M-OA^k. ECP on the other hand generates the MILP, denoted as M-ECP^k, by adding recent cutting planes corresponding to the most violated constraints, to the previous problem M-ECP^{k-1}. M-ECP^{k-1} is presented later in Section 3.4.

We mentioned earlier on in the chapter, that most prominent MINLP algorithms use either the CNLP subproblems presented in Section 3.2, the MILP we have just presented or a combination of both. We now present these methods.

3.4 Algorithms for convex MINLPs

We briefly discuss the algorithms for convex MINLPs that use the subproblems presented in Sections 3.2. Different algorithms use either one or a combination of these subproblems and/or M-OA^k or M-ECP^k, discussed in Section 3.3.

The Branch and Bound method (BB)

The Branch and Bound method (BB) was first introduced by Dakin [36]. The idea of BB is to successively split the problem into smaller problems which are easier to solve or infeasible. BB uses subproblem $\underline{\mathscr{M}}$ at each node, where $\underline{\mathscr{M}}$ includes integer constraints corresponding to the node. BB starts by solving first the continuous CNLP relaxation, $\underline{\mathscr{M}}$, which means the integrality restriction on the integer variables are removed. If this first solution satisfies the integrality constraint, then the search is stopped. Otherwise, a tree search is performed along an integer variable y_i^k that does not fulfill the integrality constraint. The variable y_i^k can be chosen using a number of methods, see for example [43], [66], [71]. The tree search results in two new subproblems, known as child nodes, where y_i^k is known in the parent node. If we let F^k be the set of points satisfying

$$F^k = \{(x,y)|c_i(x,y) = 0,\ i \in E, c_i(x,y) \geq 0,\ i \in I\},$$

then child nodes are created by the following feasible sets:

$$F_1^{k+1} = \{(x,y)|c_i(x,y) = 0,\ i \in E, c_i(x,y) \geq 0,\ i \in I,\ y_i \leq \lfloor y_i^k \rfloor\},$$

$$F_2^{k+1} = \{(x,y)|c_i(x,y) = 0,\ i \in E, c_i(x,y) \geq 0,\ i \in I,\ y_i \geq \lceil y_i^k \rceil\},$$

where $\lceil . \rceil$ and $\lfloor . \rfloor$ are the respective ceiling and floor functions. The solutions to the subproblems at the child nodes provide upper bounds on the solution to the parent nodes. These upper bounds are successively restricted at giving rise to relaxed CNLP subproblems $\underline{\mathscr{M}}$, which yield lower bounds for the solutions to the subproblems in the descendant nodes.

The best known integer-feasible solution is considered as the current overall upper bound. If the relaxed solution to any nodal subproblem is greater than the current overall upper bound, then that nodal subproblem is fathomed. If the solution of the subproblem, $\underline{\mathscr{M}}$, at any node is integer-feasible, then its solution is compared with the current overall upper bound. Therefore the fathoming takes place when the optimal value of $\underline{\mathscr{M}}$ exceeds the current upper bound, the relaxed problem $\underline{\mathscr{M}}$ is infeasible, or an integer-feasible nodal soution is found. Fathoming at an integer-feasible node takes place since any child node of this node would be

inferior in the sense that it would yield a higher objective function value. The BB algorithm terminates if there are no nodes left to consider (in which case the current upper bound is the optimal solution) [22, 63].

The Outer Approximation method (OA)

The Outer Approximation method (OA) is an alternative to BB. OA was first introduced by Duran and Grossmann [41] for MINLPs which are linear in the integer variables y. It was then generalized to the wide class of convex MINLP problems by Fletcher and Leyfer [49]. In OA, given $(x^k,y^k)^T$ subproblems $\bar{\mathscr{M}}(y^k)$ and M-OA^k are solved successively in a cycle of iterations to generate the points $(x^{k+1},y^{k+1})^T$.
Let the index sets T and S be such that
T^k=$\{j \leq k | \bar{\mathscr{M}}(y^j)$ is feasible and x^j is the optimal solution to $\bar{\mathscr{M}}(y^j)\}$,
S^k=$\{l \leq k | \bar{\mathscr{M}}(y^l)$ is infeasible and x^l is the optimal solution to $CNLPF(y^l)\}$,
where initially i.e. for $k=0$, an integer-feasible solution y^0 is chosen and T^0 and S^0 are empty. At the k-th iteration of OA, let $(\underline{x}^{k+1},y^{k+1})^T$ be the solution obtained by solving M-OA^k:

$$\text{M-OA}^k \left\{ \begin{array}{lll} \min & \alpha & \\ \text{s.t.} & f(x^j,y^j)+(\nabla f(x^j,y^j))^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} & \leq \alpha \quad \forall j \in T^k \\ & c_i(x^j,y^j)+(\nabla c_i(x^j,y^j))^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} & \geq 0, \quad \forall j \in T^k \\ & c_i(x^l,y^l)+(\nabla c_i(x^l,y^l))^T \begin{pmatrix} x-x^l \\ y-y^l \end{pmatrix} & \geq 0, \quad \forall l \in S^k \\ & x \in X, & \\ & y \in Y, & \\ & \alpha \in \mathbb{R}^+. & \end{array} \right. \tag{3.13}$$

The value $\alpha = f(\underline{x}^{k+1},y^{k+1})$ is a lower bound (LB^{k+1}) of the original problem. If y^{k+1} is feasible, then it is chosen to create the subproblem $\bar{\mathscr{M}}(y^{k+1})$. The subproblem $\bar{\mathscr{M}}(y^{k+1})$ is then solved and the corresponding solution x^{k+1} is obtained. The value $f(x^{k+1},y^{k+1})$ is an upper bound (UB^{k+1}). If y^{k+1} is infeasible, then $CNLP(y^{k+1})$ is solved and the corresponding solution x^{k+1} is found.

If $\bar{\mathscr{M}}(y^{k+1})$ is feasible, then $T^{k+1}=T^k \cup \{k\}$ is updated and $S^{k+1}=S^k$. Otherwise $S^{k+1}=S^k \cup \{k\}$ is updated and $T^{k+1}=T^k$. To prevent the discrete variable assignment

$y^j,\ j \in T^{k+1}$ from being the solution to the relaxed master problem, M-OA^{k+1}, it is necessary to define:

$$UB^{k+1} = \min\{f(x^j, y^j) | j \in T^{k+1}\}.$$

This gives rise to the OA master problem, M-OA^{k+1}:

$$\text{M-OA}^{k+1}\left\{\begin{array}{lll} \min & \alpha & \\ \text{s.t.} & \alpha < UB^k & \\ & f(x^j,y^j) + (\nabla f(x^j,y^j))^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} \leq \alpha & \forall j \in T^{k+1} \\ & c_i(x^j,y^j) + (\nabla c_i(x^j,y^j))^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} \geq 0, & \forall j \in T^{k+1} \\ & c_i(x^l,y^l) + (\nabla c_i(x^l,y^l))^T \begin{pmatrix} x-x^l \\ y-y^l \end{pmatrix} \geq 0, & \forall l \in S^{k+1} \\ & x \in X, & \\ & y \in Y, & \\ & \alpha \in \mathbb{R}^+. & \end{array}\right. \tag{3.14}$$

Here new cutting planes have been added to M-OA^{k+1} to obtain the new discrete variable assignment y^{k+2} such that $(\underline{x}^{k+2}, y^{k+2})^T$ is the solution of M-OA^{k+2}. Clearly, lower bounds are such that

$$LB^{k+1} \geq LB^k \geq \ldots LB^0.$$

The new integer y^{k+2} is chosen again to create a subproblem $\bar{\mathscr{M}}(y^{k+2})$ or $CNLPF(y^{k+2})$ and the process continues. Since the functions $f(x,y)$ and $c_i(x,y)$ are convex, the linearizations in M-OA^k correspond to outer-approximations of the feasible region in problem $\mathscr{M}$ and under-estimations of the objective function being approximated [63]. OA terminates when the upper bound generated by the solutions of $\bar{\mathscr{M}}(y^k)$, and the lower bound of M-OA^k, are within a specified tolerance, or when OA^k is infeasible. Naturally the OA algorithm converges in 1 iteration if $f(x,y)$ and $c(x,y)$ are linear [63].

The Generalized Benders Decomposition method (GBD)

The Generalized Benders Decomposition method (GBD) is similar to OA. Here two sequences of updated upper (non-increasing) and lower (non-decreasing) bounds are created that converge within a finite number of iterations. The upper bound, UB, of the problem

comes from the primal problem $\bar{\mathscr{M}}(y^k)$ and the lower bound, LB, comes from the master problem, which is derived from duality theory. Here y^k is the integer solution obtained for the master problem, see below. The solution of $\bar{\mathscr{M}}(y^k)$ also provides information about the Lagrangian multiplier estimate associated with inequality and equality constraints.

Unlike OA, GBD generates the master problem from duality theory. Let y^k be the integer solution obtained from the master problem, M-GBD^{k-1}, at the $(k-1)$-th iteration (Initially y^0 is found such that $\bar{\mathscr{M}}(y^0)$ is feasible). The subproblem $\bar{\mathscr{M}}(y^k)$ is then solved. If $\bar{\mathscr{M}}(y^k)$ is infeasible, then $CNLPF(y^k)$ is solved.

When $\bar{\mathscr{M}}(y^k)$ is feasible, its solution together with the dual variable $\lambda_i^k,\ i \in E \cup I$, corresponding to equality and inequality constraints is found, and the Lagrange function:

$$L(x,y,\lambda^k) = f(x,y) - \sum_{i \in E \cup I} \lambda_i^k c_i(x,y),$$

is constructed. Recall that the solution of $\bar{\mathscr{M}}(y^k)$ provides the upper bound, UB^k. Now, the set T^k is updated as $T^k = T^{k-1} \cup \{k\}$ and the upper bound is found such that:

$$UB^k = \min\{f(x^j,y^j) | j \in T^k\}.$$

On the other hand, when $CNLPF(y^k)$ is solved, its solution x^l together with the dual variable $\bar{\lambda}^l$ are obtained and the Lagrangian function:

$$\bar{L}(x^l,y,\bar{\lambda}^l) = f(x^l,y) - \sum_{i \in E \cup I} \bar{\lambda}_i^l c_i(x^l,y),$$

is found. The following master problem M-GBD^k is then defined

$$\text{M-GBD}^k \begin{cases} \min\limits_{y,\alpha} & \alpha \\ \text{s.t.} & \alpha \geq \inf\limits_x L(x,y,\lambda^k) \quad \forall k \in T^k \\ & 0 \leq \inf\limits_x L(x^l,y,\bar{\lambda}^l) \quad \forall l \in S^k \\ & \alpha \in \mathbb{R}^+. \end{cases} \tag{3.15}$$

The integer solution y^{k+1} of M-GBD^k is then used to create the new subproblem, $\bar{\mathscr{M}}(y^{k+1})$ or $CNLPF(y^{k+1})$, and the process continues. We notice that M-GBD^k has, as constraints, two inner optimization problems (for the case of feasible primal, $\bar{\mathscr{M}}(y^k)$, and infeasible primal, $CNLFP(y^k)$ problems). The solution of M-GBD^k provides a lower bound LB^k of the

original problem, which is non-decreasing, i.e.

$$LB^k \geq LB^{k-1} \geq \cdots \geq LB^1 \geq LB^0.$$

As the iterations proceed, it is shown that the sequence of updated upper bounds is non-decreasing, while the sequence of lower bounds is non-increasing. This results in the sequence converging in a finite number of iterations. GBD stops when $|UB^k - LB^k| \leq \varepsilon$, where ε is a small positive number, or when all integer solutions are considered (in which case the updated upper bound UB^k is treated as the optimal solution) [50].

The Extended Cutting Plane method (ECP)

The extended cutting plane method (ECP) [125] is an extension of the Kelley's cutting plane method for solving convex NLP problems [73]. It was first extended to solve convex MINLPs [125] and later the αECP method was developed to solve convex MINLPs with pseudo-convex constraints [127]. Convergence properties for ECP were then established for MINLPs containing quasi-convex constraints, in Still and Westerlund [119]. The most recent αECP method for MINLP problems containing a pseudo-convex objective function and pseudo-convex constraints, also guarantees convergence to a global minimizer [126].

Some of the advantages of ECP and αECP include the fact that they are first order methods, so no second order derivatives are required. They also require very few objective function evaluations. However, for problems with a fairly simple structure, i.e. containing a small number of integer variables and an objective function which is not very nonlinear, methods such as BB tends to outperform ECP. For more complex or large scale problems however, αECP tends to perform very well [43], [124].

ECP is very similar to OA, but unlike OA and GBD, ECP does not split the problem into CNLP and MILP subproblems, which means the continuous and discrete variables are optimized simultaneously. Naturally this may result in ECP requiring more iterations than OA and GBD. Despite this drawback, ECP efficiently solves large convex MINLPs which are not very nonlinear.

Let $(x^k,\ y^k)^T$ be the solution of the initial bound constrained linearization of problem (3.3), ECP^{k-2}, see below, and define the set

$$\hat{J}^k = \{j \in arg\{\min\ c_j(x^k, y^k)\}\}.$$

This is the set containing the most violated constraint of (3.3). Given $(x^k, y^k)^T$, the (k)-th iteration of ECP solves the M-ECP^k problem:

$$\text{M-ECP}^k \begin{cases} \min & \alpha \\ \text{s.t.} & f(x^j,y^j) + (\nabla f(x^j,y^j))^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} \leq \alpha \quad \forall j \in \hat{J}^k \\ & c_i(x^j,y^j) + (\nabla c_i(x^j,y^j))^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} \geq 0, \quad \forall j \in \hat{J}^k \\ & x \in X, \\ & y \in Y, \\ & \alpha \in \mathbb{R}^+, \end{cases} \tag{3.16}$$

to obtain the point $(x^{k+1}, y^{k+1})^T$. Here, at least one linearization is made, corresponding to the most violating constraint at $(x^{k+1}, y^{k+1})^T$, and the set $\hat{J}^{k+1} = \hat{J}^k \cup \{k\}$ is updated. M-ECP^{k+1} is then solved for a new point $(x^{k+2}, y^{k+2})^T$. New cutting planes are generated, where again only the cutting planes corresponding to most violated constraints are added to M-ECP^{k+2}. The process is repeated until an optimal solution to $\mathscr{M}$, defined by (1.3) is found.

The method can be summarized as follows. ECP successively adds a linearization or cutting plane of the most violated constraint at the predicted point $(x^k, y^k)^T$ and the solution of the increasingly tight M-ECP^k made up of these linearizations. A valid cutting plane does not cut off any part of the feasible region and ensures that the solution for the next iteration is improved. An invalid cutting plane, however may cut off part of the MINLP feasible region [77]. Valid cutting planes at the k-th iteration, together with the cutting planes generated at the previous iteration are added to M-ECP^k and at every iteration M-ECP^k is solved. This is done until all the constraints are satisfied. If a solution to ECP^k, i.e. $(x^{k+1}, y^{k+1})^T$ is infeasible, then the algorithm generates a new point which is different from all points obtained in previous solutions [66, 77].

The solution of M-ECP^k provides a new point, $(x^{k+1}, y^{k+1})^T$, on which to base the choice of the constraint to be linearized for the next iteration of the algorithm. A sequence of decreasing outer approximations to the feasible region is thus formed. The final M-ECP^k solution is a valid underestimation of the MINLP problem. This means that the feasible region of the MINLP is a subset of the feasible region of M-ECP^k:

$$\Omega_m^k \subset \Omega_m^{k-1} \subset \cdots \subset \Omega_m^1 \subset \Omega_m^0,$$

where Ω_m^0 represents the initial feasible region corresponding to M-ECP^0 and Ω_m^k represent the feasible region corresponding to the final solution of the method, M-ECP^k. The optimal solution of M-ECP^k and the MINLP, are therefore identical if the solution is feasible for both problems.

At each iteration the optimal objective value of M-ECP^k yields an objective value which is greater than the solution obtained at the previous iteration, M-ECP^{k-1}. As a result, a sequence of non-decreasing lower bounds to the optimal objective value of the convex MINLP problem is obtained:

$$LB^k \geq LB^{k-1} \geq \cdots \geq LB^1 \geq LB^0.$$

The algorithm has converged when all cutting planes are valid at the optimal solution [66]. More specifically, convergence of ECP is achieved when the maximum constraint violation lies within some prescribed tolerance [22, 63, 125].

3.5 Algorithms for non-convex MINLPs

A non-convex MINLP problem generally contains more than one solution. When $f(x,y)$ and $c(x,y)$ are non-convex, two difficulties arise. First, the NLP subproblems $\underline{\mathscr{M}}$, $\bar{\mathscr{M}}(y^j)$ and $CNLPF(y^j)$ may not have a unique local optimal solution. Second, M-OA and M-ECP do not guarantee a valid lower bound or a valid bounding representation with which the global optimum may be cut off. Under these circumstances, the deterministic methods discussed above cannot be applied to solve general MINLP problems. Unfortunately, very little research can be found in the literature for general non-convex MINLP problems. However, a number of global MINLP algorithms have been developed for non-convex problems with a specific structure [114]. These are based on the convex MINLP algorithms discussed in the previous section. They use convex under-estimators for the original non-convex problems. An overview of the most well known ones are presented below.

Ryoo and Sahinidis [112] extended BB to the Branch-and-Reduce method, for which valid convex underestimating CNLPs can be constructed for the non-convex relaxations. Unfortunately, the method cannot be applied to general nonlinear MINLP problems, except bi-linear and separable problems.

The SMIN-αBB algorithm, proposed by Adjiman et al. [2], was designed to solve mathematical models to global optimality where the binary/integer variables appear linearly and hence are separable from the continuous variable and/or appear in at most bi-linear terms. The nonlinear terms in the continuous variables appear separably from the bi-linear/integer variables. Clearly, SMIN-αBB can be applied for a special class of non-convex problems.

The use of αECP for non-convex MINLP problems was suggested by Porn and Westerlund [102]. The αECP method was designed to solve problems with pseudo-convex objective functions and pseudo-convex inequality constraints. They have suggested a new method which approximates the pseudo-convex objective function with an improving cone of linearizations.
Homogeneous with ECP, all violated constraints are approximated by cutting planes. A line-search technique is also used, where the success of the line-search is due to the convexity of the level sets for pseudo-convex functions. The αECP method solves problems with a pseudo-convex objective function and pseudo-convex constraints and is particularly efficient for solving pseudo-convex INLPs, where convergence to the global optimal solution is guaranteed in a finite number of iterations [22, 102, 127]. Global convergence of this method is also ensured for MINLPs with a linear objective function and pseudo-convex constraints. For pseudo-convex MINLPs, αECP solves a sequence of MILPs. The MILPs approximate the MINLP and the final MILP solution is a valid underestimation of the MINLP [77]. This type of problem is fairly general and many other problems can be modeled in the form of pseudo-convex objective functions with linear constraints. Some problems with a non-linear objective function and pseudo-convex constraints can also be rewritten in the form of a pseudo-convex objective function with linear constraints [66]. The αECP method is therefore applicable to a wide class of MINLPs.

The main differences between ECP and αECP occur in the modification of the parameter α^k and the generation of new constraints for the MILP at each iteration. In particular, the objective function constraint used in regular ECP is replaced with a pseudo-convex reduction constraint in αECP [77, 102]. The reduction constraint is modified whenever an improved solution for the non-convex MINLP is found. At each iteration, the solution of the MILP is used to formulate a new MILP resulting in a sequence of MILPs being solved. If an iteration produces a feasible solution, then the upper bound of the reduction constraint is modified so that no point which gives a worse objective function value, than the one obtained, can be found. When the final pseudo-convex MILP is solved and all the cutting planes are valid cutting planes, and the objective function value evaluation at the MILP solution point differs no more than the tolerance ε_f from the MILP solution:

$$|f(x^k, y^k) - \alpha^k| \leq \varepsilon_f,$$

then the global optimal solution for the pseudo-convex MINLP has been located [77]. Here ε_f is a small positive number.

Chapter 4

A Trajectory-Based Method for UNLPs

In this chapter we present the details of Snyman's trajectory-based method for UNLPs, TAUNLP [115, 116]. Section 4.1 consists of a review of TAUNLP and a detailed description of TAUNLP is presented in Section 4.2. This section is concluded by presenting a pseudo-code for TAUNLP.

4.1 Overview of TAUNLP

Snyman [115, 116] developed a method for obtaining the unconstrained minimizer of (1.1), using a trajectory-based iterative method. The trajectory-based method is designed to simulate the motion of a particle of unit mass in an n-dimensional conservative force field, where at position x the objective function $f(x)$ represents the potential energy of the particle and $T(x)$ denotes the kinetic energy of the particle. By exploiting the conservative properties of the force field, in which the total kinetic and potential energies of the particle remain constant, the particle is guided along a trajectory towards a local minimum by ensuring that its potential energy is strategically reduced [115, 116].

The differential equation (1.7) is derived by applying Hamiltonian's principle to the Lagrangian:

$$\hat{L}(x) = T(x) - f(x),$$

to obtain the equations of motion:

$$\frac{d}{dt}\frac{\partial \hat{L}}{\partial \dot{x}_i} - \frac{\partial \hat{L}}{\partial x_i} = 0, \quad i = 1, \ldots, n. \tag{4.1}$$

By manipulating and vectorizing (4.1), one obtains (1.7) [115, 116]. The trajectory of the particle may be accurately monitored at successive time instances $t^k,\ k = 0,1,2\ldots$ by solving (1.7), using an exact (energy conserving) numerical integration method. The force, $a = -\nabla f(x)$, acting on the particle at x will always tend to reduce its potential energy. However, since no frictional forces are present in (4.1) the particle will move interminably withut settling at its lowest potential energy position x^*. To remedy this situation the velocity of the particle may be monitored along the computed trajectory and manipulated to aid in guiding it toward a local minimizer x^* [115, 116].

If one denotes the velocity of the particle by

$$v(x) = \dot{x},$$

then by convention, the kinetic energy of the particle is defined as

$$T(x) = \frac{1}{2}||\dot{x}||^2,$$

where $||.||$ denotes the l_2-norm. On the other hand, the potential energy of the particle satisfies the following integral [115, 116]

$$f(x) = -\int_{x^*}^{x} a(s)ds + f(x^*), \tag{4.2}$$

since the negative rate of change of the potential energy of the particle is equivalent to the force $a(x)$ acting on it. As a result, the following is obtained

$$a(x) = -\nabla f(x),$$

i.e,

$$\ddot{x} = -\nabla f(x).$$

The significance of x^* in the integral (4.2), is that it is the point at which the particle attains its lowest potential energy i.e. the unconstrained minimizer of (1.1).
Conservation of energy implies the following relation:

$$T(x) + f(x) = T(x^0) + f(x^0) = E^0, \tag{4.3}$$

where x^0 represents the initial position of the particle, E^0 represents the initial total energy of the particle and $T(x^0)$ and $f(x^0)$ represent the initial kinetic and potential energies of the particle respectively. Now, in order to obtain a computed trajectory that will tend to the

local minimum x^* of (1.1), the potential energy should decrease overall along the exaclty computed path. If the initial velocity of any particle is 0, it follows that $T(x^0) = 0$. By (4.3) then, at least from some initial point x^0, the potential energy of the particle is guaranteed to decrease, as desired. Furthermore, an increase in velocity and consequently an increase in the kinetic energy of the particle brings about a decrease in the potential energy of the particle. To demonstrate this, note that by conservation of energy, the following holds at any two consecutive times t^k and t^{k+1}:

$$\Delta f^k = f(x^{k+1}) - f(x^k) = -T(x^{k+1}) + T(x^k) = -\Delta T^k.$$

It follows that $\Delta f^k < 0$ if and only if $\Delta T^k > 0$. A decrease in f is therefore guaranteed by an increase in the velocity of the particle:

$$T(x^k) < T(x^{k+1}). \tag{4.4}$$

The kinetic energy of the particle x^k, i.e., $T(x^k)$, is a function of the velocity v^k:

$$T(x^k) = \frac{1}{2}||v^k||^2. \tag{4.5}$$

It therefore follows from (4.4) and (4.5), that for descent

$$||v^k|| < ||v^{k+1}||. \tag{4.6}$$

Within the iterative procedure of the trajectory-based method the velocity of the particle may be monitored and accordingly be manipulated so that predominantly the function value at two successive times t^k and t^{k+1} satsifies

$$f(x^{k+1}) \leq f(x^k),\ \forall x \in \Omega.$$

Since condition (4.6) brings about a decrease in f, it can be thought of as a descent condition. With the outline of Snyman's [115, 116] trajectory-based algorithm in place, we now present the details of TAUNLP.

4.2 The main features of the TAUNLP algorithm

In this section, the process used by Snyman [115, 116], to solve the unconstrained optimization problem (1.1), will be discussed. It is important to realize at this point that the

computationally economic integration method used by Snyman, namely the Leap-Frog method and implemented below via (4.7) and (4.8), is only approximately energy conserving (see equation (A.8) in the Appendix to [115]). The subtle implication is that one cannot conclude with absolute certainty that if the computed velocities satisfy (4.6) that descent is necessarily achieved in all instances (although for sufficiently small time steps one may expect it to occur for most cases). Therefore the implementation of the descent philosophy described in the previous section required the introduction of some heuristic adjustments to the computed trajectory in order to impose a high probability of the overall descent along the computed path and effective controlled convergence to a local minimum. Throughout this section we make the distinction between "calculated" and "assigned" quantities. During the integration process described below, we "calculate" the position and velocity of the particle at x. Whenever corrective procedures are activated however, we "assign" new values for the position and velocity of the particle. This distinction will become clearer in the discussions that follow.

At the k-th iteration of Snyman's method [115, 116], one has a central system, integrating (1.7) over the interval $[t^k,\ t^{k+1}]$, where there are a number of parameters involved. These include the time step, Δt^k, the parameter used to monitor the progress of the iterates, i.e. $\hat{m}$ and the parameters used to magnify Δt^k, i.e. δ, δ_1 and p^k. The significance of these parameters will be made clear a little later on in this section. Note that Δt^k is just a parameter, i.e. the integration time step used when integrating the system (1.7) from t^k to t^{k+1}. It is not used to mean $\Delta t^k = t^{k+1} - t^k$, unlike $\Delta x^k = x^{k+1} - x^k$.

The Euler formulae used to integrate (1.7) are the respective Euler forward and Euler backward equations below

$$x^{k+1} = x^k + v^k \Delta t^k, \tag{4.7}$$

$$v^{k+1} = v^k - \nabla f(x^{k+1}) \Delta t^k. \tag{4.8}$$

Before integrating (1.7), TAUNLP is initialized as follows. Given $x(0) = x^0$, the velocity of the particle is initialized as

$$v^0 = -\frac{1}{2} \nabla f(x^0) \Delta t^0. \tag{4.9}$$

Now, at any iteration of TAUNLP, as with any algorithm, there is a possibility that $||\nabla f||$ is large. To avoid excessively large steps from occurring whenever $||\nabla f||$ is large, Snyman

[115, 116] proposed that the following inequality be satisfied

$$||\Delta x^k|| < \delta, \tag{4.10}$$

where Δx^k is calculated from equation (4.7) as

$$||\Delta x^k|| = ||v^k||\Delta t^k. \tag{4.11}$$

Here δ denotes the maximum allowable step size. If (4.10) is not satisfied, then v^k is assigned as

$$v^k = \frac{\delta}{\Delta t^k}\frac{v^k}{||v^k||}, \tag{4.12}$$

using the same δ from (4.10), which is typically equal to 1. Since, by (4.7), $||\Delta x^k||$ is dependent on the value of v^k, Snyman [115, 116] proposed that v^k be assigned using (4.12), to reduce the step $||\Delta x^k||$.

At this stage one begins to integrate (1.7) to obtain x^{k+1} and v^{k+1} using (4.7) and (4.8) respectively.

We now consider two cases in which the velocity and position of the particle are adjusted if the iterates are not progressing as desired. The first adjustment takes place whenever (4.6) is not satisfied. If at the k-th iteration the velocity of the particle at any fixed time interval $[t^k,\ t^{k+1}]$ does not increase, i.e. (4.6) is not satisfied, then the new velocity at x^{k+1} is assigned as

$$v^{k+1} = \frac{1}{2}(\frac{1}{2}v^k + \frac{1}{2}v^{k+1}), \tag{4.13}$$

i.e. the half of the average of the known velocities at t^k and t^{k+1}.

This restart at t^{k+1} with v^{k+1} is likely to give a more satisfactory trajectory towards x^*. Should the velocity continue to decrease after the restart and integration, it is likely that the particle is not following the required trajectory and the more severe restart for the velocity,

$$v^{k+1} = 0, \tag{4.14}$$

is used. The next iteration x^{k+1} is then found by integration in $[t^k,\ t^{k+1}]$ either using v^{k+1} in (4.13) or $v^{k+1} = 0$ as the case may be. An adjustment of this new iterate is carried out. In both cases, i.e. when either (4.13) or (4.14) is applied, the position of the particle is also assigned as:

$$x^{k+1} = \frac{1}{2}x^k + \frac{1}{2}x^{k+1}, \tag{4.15}$$

i.e. $\frac{1}{2}x^k + \frac{1}{2}x^{k+1}$ is assigned to x^{k+1} whenever $v^{k+1} = \frac{1}{2}(\frac{1}{2}v^k + \frac{1}{2}v^{k+1})$ or $v^{k+1} = 0$ is activated. The update described by (4.15) was used by Snyman [115, 116], as opposed to $x^{k+1} = x^k$, because it gave improved convergence.

At the $k+1$-th iteration, when v^{k+1} is reduced, using (4.13) or (4.14), x^{k+1} is updated using (4.15). In the next iterate v^{k+2} is then found from (4.8) using v^{k+1}, $\nabla f(x^{k+1})$ and Δt^{k+1}.

The second adjustment of v^k and x^k is based on the angle between two successive gradient approximations. Here Δt^k is also updated since it is a significant parameter for the integration process. To ensure optimal performance Δt^k is monitored and updated based on the following criterion. If the angle between 2 successive gradient estimates $a^k = -\nabla f(x^k)$ and $a^{k+1} = -\nabla f(x^{k+1})$ is greater than $\frac{\pi}{2}$, i.e.

$$(a^{k+1})^T(a^k) \leq 0, \tag{4.16}$$

then Snyman [115, 116] attributes this to the fact that the trajectory is inaccurate. According to Snyman [115, 116], this is likely to be the result of an excessively large time step. Without intervening too much at the first instance when this happens, Snyman [115, 116] proposed that the following correctional procedure be put into place. Whenever (4.16) is satisfied, the parameter $\hat{m}^k$ is updated as follows:

$$\hat{m}^{k+1} = \hat{m}^k + 1, \tag{4.17}$$

where initially $\hat{m}^k = 0$. If $\hat{m}^{k+1} = 3$, i.e. if (4.16) is satisfied for 3 consecutive pairs (a^{k-2}, a^{k-1}), (a^{k-1}, a^k) and (a^k, a^{k+1}) (we refer to this as the angle violation criterion throughout this thesis) then Snyman suggested the following. At the time t^{k+1}, the velocity v^{k+1} is assigned as

$$v^{k+1} = \frac{1}{2}(\frac{1}{2}v^{k+1} + \frac{1}{2}v^k), \tag{4.18}$$

and the position x^{k+1} is assigned using (4.15).

Δt^k is also updated as follows

$$\Delta t^{k+1} = \frac{\Delta t^k}{2}, \tag{4.19}$$

where Δt^{k+1} is used in the next iteration to integrate over $[t^{k+1},\ t^{k+2}]$. The value of $\hat{m}^{k+1}$ at any iteration is 0 if (4.16) is not satisfied.

At each k-th integration in $[t^k,\ t^{k+1}]$, Δt^k is monitored and in the event of a successful step, where

$$(a^{k+1})^T(a^k) > 0, \tag{4.20}$$

Δt^{k+1} is magnified as follows

$$\Delta t^{k+1} = \begin{cases} p^k \Delta t^k & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| < \delta, \\ \Delta t^k & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| \geq \delta, \end{cases} \tag{4.21}$$

where p^k satisfies

$$p^{k+1} = \begin{cases} p^k + \delta_1 & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| < \delta, \\ p^k & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| \geq \delta. \end{cases} \tag{4.22}$$

The parameter p^k is initially set to 1 and whenever (4.20) is satisfied, is gradually increased by the factor δ_1, which is typically equal to 0.001. In the event of an unsuccessful step, i.e. when (4.16) is satisfied, in addition to imposing (4.17), p^{k+1} is reset to 1:

$$p^{k+1} = 1.$$

The implementation of an adaptation of this method is presented later in Chapter 6. The main features of TAUNLP are presented in **Algorithm 1** below.

Algorithm 1 Brief outline of TAUNLP

Given Δt^k, $\hat{m}^k = 0$, p^k, δ, δ_1, initialize $k \leftarrow 0$
Given x^k, compute $\nabla f(x^k)$ and calculate v^k using (4.9).
while $|| - \nabla f(x^k)|| = ||a^k|| > \varepsilon$ **do**
    if $||\Delta x^k|| > \delta$ **then**
        assign (4.12) to v^k,
    else
        update Δt^{k+1} and p^{k+1} using (4.21) and (4.22) respectively.
    end if
    if $\hat{m}^k \geq 3$ **then**
        assign (4.15), (4.18) and (4.19) to x^{k+1}, v^{k+1} and Δt^{k+1} respectively.
    end if
    Integrate the system defined by equation (1.7) over the interval $[t^k,\ t^{k+1}]$ at iteration k, to obtain x^{k+1}, using Euler's method described by (4.7).
    Compute $\nabla f(x^{k+1})$ and integrate the system defined by (1.7) over the interval $[t^k,\ t^{k+1}]$, using Euler's method described by (4.8), to obtain v^{k+1}.
    if (4.16) holds **then**,
        set $p^{k+1} = 1$, and update $\hat{m}^{k+1} = \hat{m}^k + 1$
    else
        set $\hat{m}^{k+1} = 0$
    end if

19: **if** $||\nabla f(x^{k+1})|| = ||-a^{k+1}|| \leq \varepsilon$ **then**
20: **STOP**
21: **else**
22: **go to 24**
23: **end if**
24: **if** condition (4.6) is not satisfied **then**
25: Assign (4.13) and (4.15) to x^{k+1} and v^{k+1}, upon the first instance that (4.6) is not satisfied or (4.14) and (4.15) upon the second consecutive instance that (4.6) is not satisfied, set $k = k+1$ and **return to** 12.
26: **else**
27: Set $k = k+1$ and **go to** 3.
28: **end if**
29: **end while**

Remarks on Algorithm 1

Remark 4.2.1. In lines 4-8 of **Algorithm 1**, one calculates Δx^k as in (4.11) and determines whether the inequality (4.10) is satisfied. If it is satisfied, then Δt^{k+1} and p^{k+1} are magnified. If it is not satisfied, then v^k is updated to reduce Δx^k.

Remark 4.2.2. In lines 9-11, if the angle violation criterion is satisfied, then one reduces the velocity of the particle as in (4.13) and the particle is recalculated using (4.15). One also reduces the step size Δt^k using (4.19), as a corrective measure to improve on the accuracy of the integration method for the next iteration.

Remark 4.2.3. Here we present the integration of x which is activated in line 12 of TAUNLP: Euler's method described in (4.7) is used to integrate (1.7). Given x^k and Δt^k, the method calculates x^{k+1}.

Remark 4.2.4. Here we present the integration of v which is activated in line 13 of TAUNLP: $\nabla f(x^{k+1})$ is updated and Euler's method described in (4.8), is used to integrate (1.7). Given x^{k+1}, v^k and Δt^k, the method calculates v^{k+1}.

Remark 4.2.5. In lines 14-18, the angle between two successive gradient approximations is calculated, and the parameters $\hat{m}^{k+1}$ and p^{k+1} are updated accordingly. If the angle between two successive gradient approximations is less than $\frac{\pi}{2}$, then $\hat{m}^{k+1}$ is set to 0.

Remark 4.2.6. Recall that v^k is the velocity at t^k obtained over the interval $[t^{k-1},\ t^k]$ and v^{k+1} is the velocity at t^{k+1} obtained over the interval $[t^k,\ t^{k+1}]$. In lines 24-26, these are compared. If the descent condition (4.6) is not satisfied, then the assignments described in (4.13) and

(4.15) are used. If there is still no improvement in the velocity of the particle for more than 2 consecutive iterations, then (4.14) is assigned to v^{k+1} and (4.15) is assigned to x^{k+1}.

Remark 4.2.7. Once the descent condition, (4.6), is satisfied, we proceed to the next iteration in line 27 of TAUNLP.

TAUNLP has seen some success since its inception and has proven to be competitive with some methods for unconstrained optimization [115, 116]. One of its most attractive features is that it is a first order method, so no Hessian computation is needed throughout the minimization routine. Furthermore, no function evaluations are needed within the solution process. One of the focal points of this thesis is thus to further extend the trajectory methodology by its application here to Augmented Lagrangian formulations of the for general constrained optimization (1.2). We develop such an algorithm in the next chapter.

Chapter 5

A Trajectory-Based Method for CNLPs

In this chapter, we present the trajectory-based algorithm (TACNLP) for Augmented Lagrangian formulations of the general constrained optimization problem (1.2). The success of the trajectory-based method for unconstrained optimization, TAUNLP, has prompted our interest in the development of TACNLP. The outline of TACNLP, which is an extension of TAUNLP, can be found in Section 5.1. Section 5.2 contains a detailed break down of the main features of TACNLP. Some fundamental changes are made in an attempt to improve on the existing framework of the method proposed by Snyman [115]. This includes incorporating an adaptive step size routine into TACNLP. A new technique for updating the penalty parameter, μ, associated with the augmented Lagrangian (2.30) is also discussed.

5.1 Overview of TACNLP

The trajectory-based algorithm (TACNLP) considered here is a first order method. Much like the gradient-based methods mentioned in Chapter 2, it uses gradient information to progress from one iteration to the next. Within the framework of TACNLP, AL will be used to solve $\mathscr{P}$, defined by (1.2). Since AL contains primal and dual variables, the algorithms presented will minimize (2.30) with respect to the primal variable x and maximize (2.30) with respect to the dual variables λ.

Under the mathematical arguments used to derive (1.7) when minimizing the unconstrained function $f(x)$, we can convert the dual problem of minimizing (2.30) with respect to x and maximizing (2.30) with respect to λ, to

$$\ddot{x} = -\nabla_x \phi_A(x, \lambda; \mu), \quad x(0) = x^0, \quad \dot{x}(0) = 0, \tag{5.1}$$

$$\ddot{\lambda} = \nabla_\lambda \phi_A(x,\lambda;\mu), \quad \lambda(0) = \lambda^{(0)}, \quad \dot{\lambda}(0) = 0, \tag{5.2}$$

where ϕ_A is defined as in (2.30). We therefore look for the trajectories $x(t)$ and $\lambda(t)$ that minimize $\phi_A(x,\lambda;\mu)$ with respect to x and maximize $\phi_A(x,\lambda;\mu)$ with respect to λ, by solving (5.1) - (5.2) simultaneously. This step is homologous with the minimization of the unconstrained problem (1.1). Another step which is used identically in TAUNLP and TACNLP is the manipulation of the velocity component v^k. Here v^k is manipulated in the solution process of (5.1) - (5.2) such that $\phi_A(x,\lambda,\mu)$ is minimized with respect to x. The solution converges when the first order KKT conditions:

$$||\nabla_x \phi_A|| \leq \varepsilon, \tag{5.3}$$

and

$$||\nabla_\lambda \phi_A|| \leq \varepsilon, \tag{5.4}$$

are satisfied, where $\varepsilon > 0$ is a small positive number. By monitoring the trajectories we can ensure that both $x(t)$ and $\lambda(t)$ are guided to their respective optimal values x^* and λ^*.

The penalty parameter, μ in $\phi_A(x,\lambda;\mu)$ is used in various Augmented Lagrangian methods. Within the solution process of these methods, μ is updated to retain feasibility of the equality constraints and improve complementarity of the inequality constraints. This follows from the KKT conditions in **Theorem 2.1.6**, which imply that, for equality constraints the iterates $\{x^k\}$ generated by any CNLP algorithm must satisfy:

$$\lim_{k\to\infty} \underset{i\in E}{c_i}(x^k) = 0.$$

Furthermore, for inequality constraints one of the following must be satisfied

$$\lim_{k\to\infty} \underset{i\in I\cap A_s(x)}{c_i}(x^k) = 0, \tag{5.5}$$

or

$$\lim_{k\to\infty} \underset{i\in I\setminus A_s(x)}{\lambda_i^k} = 0, \tag{5.6}$$

where (5.5) corresponds to active inequality constraints and (5.6) corresponds to inactive inequality constraints. Equations (5.5)-(5.6) can be rewritten as

$$\lim_{k\to\infty} \lambda_i^k c_i(x^k) = 0,$$

which is the complementarity condition. Further details of this are provided in Section 6.2.3.

Conventionally a combination of the strategies employed, for example, in [6, 18] and [130] are used to determine how μ should be updated. Unlike the updates used in [6, 18, 130] though, we update μ by solving a differential equation. Since TACNLP is based on differential equations, in particular involving the trajectories of $x(t)$ and $\lambda(t)$, we incorporate the following differential equation based on μ

$$\ddot{\mu} = -\mu, \quad \mu(0) = \mu^{(0)}, \quad \dot{\mu}(0) = 0. \tag{5.7}$$

The parameter $\mu(t)$ is now coupled with $x(t)$ and $\lambda(t)$ via (5.8) and (5.9). The resultant system of equations to be solved throughout the minimization process in TACNLP becomes:

$$\ddot{x} = -\nabla_x \phi_A(x, \lambda; \mu), \quad x(0) = x^0, \quad \dot{x}(0) = 0, \tag{5.8}$$

$$\ddot{\lambda} = \nabla_\lambda \phi_A(x, \lambda; \mu), \quad \lambda(0) = \lambda^{(0)}, \quad \dot{\lambda}(0) = 0, \tag{5.9}$$

$$\ddot{\mu} = -\mu, \quad \mu(0) = \mu^{(0)}, \quad \dot{\mu}(0) = 0. \tag{5.10}$$

There are some obvious challenges when obtaining a solution to this system of differential equations, as opposed to obtaining a solution to (1.7). These include the fact that ϕ_A is an approximation of, and not the exact representation of the objective function and constraints in (1.2). Moreover, unlike the unconstrained case, here the system (5.8) - (5.10) is coupled with three completely different types of variables. The conflicting role of x and λ and the highly influential nature of μ, make the integration very sensitive. To overcome these and other difficulties, a number of mechanisms have been put into place.

Firstly, whenever the descent condition (4.6) is not satisfied, we manipulate the trajectories $x(t)$ and $\lambda(t)$ via v^k so that working with ϕ_A instead of $f(x)$ is not an issue. We elaborate more on this in Sections 5.2 and 6.1. Secondly, an adaptive step size routine has been

implemented within the solution process of (5.8) - (5.9). The objective here is to control and maintain errors for the stability of the solution of the system of differential equations. An adaptive step size routine is not used within the solution process of (5.10), since this is a simple differential equation which effects, but is not affected by the solutions to (5.8) - (5.9). Lastly, TACNLP is particularly sensitive to badly scaled problems, or problems of relatively high dimension. To overcome this difficulty, a scaling procedure has been put in place to detect and scale these types of problems. We discuss this thoroughly here and in Section 6.1 of Chapter 6.

Unlike Snyman [115], we introduce the adaptive step control to deal with the solution of a system of differential equations. For the adaptive step size routine, we introduce Modified Euler's method which was not implemented by Snyman [115]. Modified Euler's method is an extension of the Euler method presented in Section 4.2. A solution to the system (5.8) - (5.9) is obtained by using a combination of Euler's method and Modified Euler's method. Euler's method and Modified Euler's method are paired together as part of the routine for an adaptive step size method which will be used within the integration strategy. The adaptive step size method adjusts the step size, Δt, to maintain minimal local truncation error of the integration method [29, 30, 65, 69]. This acts as a safe guard for producing a sequence of converging iterates. Within the adaptive step size routine, we need to obtain a solution using both Euler's and Modified Euler's methods. The difference between Euler's and Modified Euler's method will be made clearer in Section 5.2. The approximations $\hat{x}^k$ and $\hat{\lambda}^k$ obtained using Modified Euler's method will be used to acquire an estimate for the local truncation error. No extra computation is needed to calculate the Modified Euler estimate, so this will not hinder the computational time or overall performance of the algorithm. Both Euler's and Modified Euler's methods approximately preserve energy [115], a crucial feature for the success of the minimization procedure. We refer the reader to Snyman's paper [115] for verification of this.

An iterative algorithm, TACNLP, which can be found in Chapter 6, has been developed to obtain a solution to (1.2), by solving (5.8) - (5.10). We now present a detailed break down of the main features of TACNLP.

5.2 The main features of the TACNLP algorithm

In this section, we outline the main features of TACNLP. Within TACNLP, some fundamental adjustments have been made in comparison to the unconstrained case. An adaptive step size routine has been implemented to optimize both primal and dual variables, x^k and λ^k, when solving (5.8) - (5.9). A new method for updating the penalty parameter μ is

also implemented. This has resulted in minimizing the use of parameters. The details of TACNLP are outlined below.

5.2.1 Updates of x^k, λ^k and μ^k

This section is necessary because we use this information in subsequent sections, i.e. the adaptive step size method, the penalty parameter update and the scaling routine. We have mentioned in Section 5.1 that x, λ and μ will be updated using Euler's method and x and λ will be updated using Modified Euler's method as well. Here we denote the time increments used to integrate (5.8) - (5.10) with respect to x, λ and μ by Δt_x^k, Δt_λ^k and Δ_μ^k respectively. Starting with an initial point $(x^0,\ \lambda^0,\ \mu^0)$, the system (5.8) - (5.10) is integrated at the k-th iteration in $[t^k,\ t^{k+1}]$ as follows.

$$x^{k+1} = x^k + v^k \Delta t_x^k \tag{5.11}$$

$$\begin{aligned} \lambda_i^{k+1} &= \lambda_i^k + w_i^k \Delta t_\lambda^k,\ i \in E, \\ \lambda_i^{k+1} &= \max\{\lambda_i^k + w_i^k \Delta t_\lambda^k, 0\},\ i \in I, \end{aligned} \tag{5.12}$$

and

$$\mu^{k+1} = \mu^k + \underline{d}^k \Delta t_\mu^k, \tag{5.13}$$

where

$$v^k = v^{k-1} + a^k \Delta t_x^{k-1}, \tag{5.14}$$

$$w^k = w^{k-1} + b^k \Delta t_\lambda^{k-1}, \tag{5.15}$$

and

$$\underline{d}^k = \underline{d}^{k-1} + c^k \Delta t_\mu^{k-1}. \tag{5.16}$$

In equation (5.12) the update, $\max\{\lambda_i^k + w_i^k \Delta t_\lambda^k, 0\}$, corresponding to inequality constraints is imposed to comply with the KKT conditions in **Theorem 2.1.6**, which state that these must be non-negative. Furthermore a^k, b^k and c^k correspond to the right hand side of the system (5.8) - (5.10). Here $a^k = -\nabla_x \phi_A$ and $b^k = \nabla_\lambda \phi_A$, where ϕ_A is defined as in (2.30), and $c^k = -\mu^k$, since we are not minimizing or maximizing ϕ_A with respect to μ. Using Modified Euler's method, the following updates are obtained

$$\hat{x}^{k+1} = x^k + \hat{v}^k \Delta t_x^k \tag{5.17}$$

and

$$\begin{aligned} \hat{\lambda}_i^{k+1} &= \lambda_i^k + \hat{w}_i^k \Delta t_\lambda^k,\ i \in E, \\ \hat{\lambda}_i^{k+1} &= \max\{\lambda_i^k + \hat{w}_i^k \Delta t_\lambda^k, 0\},\ i \in I, \end{aligned} \tag{5.18}$$

where

$$\begin{aligned} \hat{v}^k &= v^{k-1} + \tfrac{(a^{k-1}+a^k)}{2} \Delta t_x^{k-1}, \\ \hat{w}^k &= w^{k-1} + \tfrac{(b^{k-1}+b^k)}{2} \Delta t_\lambda^{k-1}. \end{aligned} \tag{5.19}$$

The Modified Euler estimates in equations (5.17) - (5.19) will be used within the adaptive step size routine. From (5.14) and (5.15), we notice that the Euler updates used to calculate v^k and w^k depend on a^k and b^k respectively. Notice also that the Modified Euler updates $\hat{v}^k$ and $\hat{w}^k$ calculated in (5.19), require the evaluation of a^{k-1}, b^{k-1}, a^k and b^k, which would have been calculated anyway using Euler's method. This asserts the claim made earlier, that no extra computation is required for computing a solution using Modified Euler's method. We now use these estimates within the adaptive step size routine.

Remark 5.2.1. While integrating (5.8) - (5.10) in $[t^k,\ t^{k+1}]$, we enforce the criteria (4.6) using v^k and v^{k+1} as well as w^k and w^{k+1}. This means that we manipulate the trajectory of $x(t)$ and $\lambda(t)$ to enforce (4.6). This process will be described later in this section and again in Section 6.1 of Chapter 6.

5.2.2 Adaptive step size

We implement the adaptive step size routines for x and λ. Since we are optimizing primal as well as dual variables, we need to ensure that the corresponding step sizes are adapted for both variables. For the adaptive step size implementation, the solutions obtained at the kth

iteration, using Euler's and Modified Euler's method, are denoted as x^{k+1}, λ^{k+1} and $\hat{x}^{k+1}$, $\hat{\lambda}^{k+1}$ respectively. The absolute differences between these are calculated:

$$\begin{aligned} S_x &= ||x^{k+1} - \hat{x}^{k+1}||, \\ S_\lambda &= ||\lambda^{k+1} - \hat{\lambda}^{k+1}||. \end{aligned} \tag{5.20}$$

We then use S_x and S_λ as measurements of the local discretization error of Euler's method [29, 30, 65, 69].

The user prescribed tolerance for the discretization error, $\hat{\varepsilon}$, is compared to this actual error. If the ratio between $\hat{\varepsilon}$ and S_x i.e. $\frac{\hat{\varepsilon}}{S_x}$ is small, then the actual error of the method is greater than $\hat{\varepsilon}$, indicating that the Euler approximate x^k is not very accurate. In this case we need to reduce the step size Δt_x^k, corresponding to (5.8), by the factor $t_{(x,1)}$, which must not be smaller than $\frac{1}{2}$. If, on the other hand $\frac{\hat{\varepsilon}}{S_x}$ is large, then S_x is smaller than the user prescribed tolerance for the local discretization error $\hat{\varepsilon}$, indicating that the integration formula, Euler's method, is relatively accurate. In this case, we would like to increase the step size Δt_x^k by the factor $t_{(x,2)}$, which must be no larger than 2.

We use the same logical argument when comparing the ratio between $\hat{\varepsilon}$ and S_λ. In this case, when $\frac{\hat{\varepsilon}}{S_\lambda}$ is small, we reduce the step size Δt_λ^k, corresponding to (5.9), by the factor $t_{(\lambda,1)} \geq \frac{1}{2}$. Similarly when $\frac{\hat{\varepsilon}}{S_\lambda}$ is large, we increase Δt_λ^k by the factor $t_{(\lambda,2)} \leq 2$. We must be careful not to decrease or increase the step sizes too much in each of the above cases.

To do this, when updating Δt_x^k and Δt_λ^k, we multiply the ratios $\sqrt{(\frac{\hat{\varepsilon}}{S_x})}$ and $\sqrt{(\frac{\hat{\varepsilon}}{S_\lambda})}$ by a positive number less than 1. The common choice for this parameter is 0.9 [29, 30]. The step sizes Δt_x^{k+1} and Δt_λ^{k+1}, corresponding to x and λ respectively, are therefore updated as follows [29, 30]:

$$\begin{aligned} \Delta t_x^{k+1} &= \Delta t_x^k \times \max\{t_{(x,1)}, \min\{t_{(x,2)}, 0.9 \times \sqrt{(\tfrac{\hat{\varepsilon}}{S_x})}\}\}, \\ \Delta t_\lambda^{k+1} &= \Delta t_\lambda^k \times \max\{t_{(\lambda,1)}, \min\{t_{(\lambda,2)}, 0.9 \times \sqrt{(\tfrac{\hat{\varepsilon}}{S_\lambda})}\}\}. \end{aligned} \tag{5.21}$$

By implementing the adaptive step size routine, we are able to update Δt_x^{k+1} and Δt_λ^{k+1} based on the extent to which the maximum prescribed discretization error, $\hat{\varepsilon}$, has been violated at each iteration. Typically $t_{(x,1)} = t_{(\lambda,1)} = \frac{1}{2}$ and $t_{(x,2)} = t_{(\lambda,2)} = 2$, but these values can be varied to increase optimal performance of the algorithm and we take advantage of this within our algorithm. Furthermore, the bounds Δt_{min} and Δt_{max} are set on both Δt_x^k and Δt_λ^k, to prevent them from getting too small or too large respectively. More details of this are presented in Chapter 6.

5.2.3 Scaling

To ensure that TACNLP performs optimally, it is important that there is a mechanism in place for badly scaled problems. Problems where $f(x)$ or $c(x)$ have magnitudes much larger than 1, when evaluated at any point x^k, can impact on the performance of the algorithm. Problems of high dimension may produce similar results. TACNLP is particularly sensitive to these types of problems, specifically within the adaptive step size routine used for x and λ. In particular, badly scaled problems usually produce solutions $\{x^{k+1}, \lambda^{k+1}\}$ and $\{\hat{x}^{k+1}, \hat{\lambda}^{k+1}\}$, described by (5.11), (5.12), (5.17) and (5.18) respectively, which may be large in magnitude when they are far from $\{x^*,\ \lambda^*\}$. The discretization error estimates, S_x and S_λ are consequently large and not accurate representations of the performance of the integration scheme. Because the updates of Δt_x^k and Δt_λ^k rely on these error estimates, they too are compromised. Scaling the objective function and constraints is an effective way of dealing with this. To illustrate this, consider a problem

$$\begin{cases} \min & f(x) = 100(x_2 - x_1^2)^2 - (1-x_1)^2, \\ \text{st} & c_1(x) = x_2 + 1.5 \geq 0, \end{cases}$$

for which

$$\nabla f(x) = [-400x_1(x_2 - x_1^2) + 2(1-x_1),\ 200(x_2 - x_1^2)]^T.$$

This can be written equivalently as

$$\nabla f(x) = 100[-4x_1(x_2 - x_1^2) + 0.02(1-x_1),\ 2(x_2 - x_1^2)]^T.$$

The unconstrained objective function $f(x)$ has the exact same minimizer if it has the gradient

$$\nabla f(x) = [-4x_1(x_2 - x_1^2) + 0.02(1-x_1),\ 2(x_2 - x_1^2)]^T.$$

By scaling this problem we can avoid disproportional error estimates S_x and S_λ, which cause high iteration numbers and eventual divergence of the TACNLP algorithm [38, 58]. Within our algorithmic framework the following scaling implementation is carried out. Before initialization of TACNLP, we generate q random vectors, x^q, within some user defined interval as follows. For each problem, limits on each variable are estimated such that $x_i \in (l_i,\ u_i),\ i = 1, 2, \ldots n$. Typically l_i and u_i are set to -50 and 50 respectively, to cover a suitably large range. The i-th coordinate of a random point x^q is calculated as

$$x_i = l_i + \bar{r} \times (u_i - l_i), \tag{5.22}$$

where $\bar{r} \in (0,1)$ is a random number. Each of the q random points are then generated. We then evaluate the gradient of the objective function, $\nabla f(x)$, the constraints $c_i(x)$, as well as the constraint normals, $\nabla c_i(x)$ at each of the q vectors. We consider only $\nabla f(x)$, $c_i(x)$ and $\nabla c_i(x)$ since these appear explicitly in the gradient of AL defined by (2.30). We then obtain the averages $(\nabla f)_s$, $(c_i)_s$ and $(\nabla c_i)_s$ of $\nabla f(x^q)$, $c_i(x^q)$ and $\nabla c_i(x^q)$ respectively. During the course of the TACNLP algorithm $\nabla f(x^k)$, $c_i(x^k)$ and $\nabla c_i(x^k)$ are scaled with their respective averages. For instance the gradient at x^k, $\nabla f(x^k)$, will be scaled as:

$$\frac{\nabla f(x^k)}{(\nabla f)_s}, \tag{5.23}$$

whenever $||\nabla f(x)|| \geq \tau$. Here τ is some user prescribed tolerance. Similarly $c_i(x^k)$ will be scaled as:

$$\frac{c_i(x^k)}{(c_i)_s}, \tag{5.24}$$

if $||c_i(x)|| \geq \tau$, and $\nabla c_i(x^k)$ will scaled as:

$$\frac{\nabla c_i(x^k)}{(\nabla c_i)_s}, \tag{5.25}$$

if $||\nabla c_i(x)|| \geq \tau$. This scaling procedure ensures that the magnitude of these components are at most close to unity throughout the minimization routine in TACNLP. A problem may arise as the iterates converge closer to the optimal solution because we expect the magnitude of the gradient of the augmented Lagrangian to decrease significantly. If the components $\nabla f(x)$, $c_i(x)$ and $\nabla c_i(x)$, are still scaled when they no longer need to be then this could result in the algorithm terminating prematurely. It is important therefore that scaling only takes place when we are far from the optimal solution. Therefore whenever $||\nabla f(x)|| < \tau$, we no longer scale ∇f. Similar strategies are used for termination of scaling $c_i(x)$ and $\nabla c_i(x)$. This scheme contributes significantly to the convergence of the algorithm, which is outlined below.

5.2.4 Quantities for convergence

The convergence criterion, which will be used in stopping TACNLP, is presented here. Recall that we are using an Augmented Lagrangian which is continuous. Define

$$\zeta(x^k, \lambda^k; \mu^k) = \begin{pmatrix} \nabla_x \phi_A(x^k, \lambda^k; \mu^k) \\ \nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k) \end{pmatrix}$$

where

$$\nabla_x \phi_A(x^k, \lambda^k; \mu^k) = \nabla_x f(x^k) - \sum_{i \in E \cup (I \cap A_s(x))} \lambda_i^k \nabla_x c_i(x^k) + \frac{1}{\mu^k} \sum_{i \in E \cup (I \cap A_s(x))} c_i(x^k) \nabla_x c_i(x^k) + \nabla_x \psi(x^k, \lambda^k; \mu^k)$$

with

$$\nabla_x \psi_A(x^k, \lambda^k; \mu^k) = 0,$$

and

$$\nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k) = - \sum_{i \in E \cup (I \cap A_s(x))} \mathrm{c_i(x^k)} + \nabla_\lambda \psi(x^k, \lambda^k; \mu^k)$$

with

$$\nabla_\lambda \psi(x^k, \lambda^k; \mu^k) = - \sum_{i \in I \setminus A_s(x)} \mu^{\mathrm{k}} \lambda_{\mathrm{i}}^{\mathrm{k}} \ .$$

If we let

$$\theta(x^k, \lambda^k; \mu^k) = ||\zeta(x^k, \lambda^k; \mu^k)||, \tag{5.26}$$

then convergence is said to take place if

$$\theta(x^k, \lambda^k; \mu^k) \leq \varepsilon^k, \tag{5.27}$$

where $\varepsilon^k \to 0$. This is inline with the satisfaction of the KKT conditions (5.3) - (5.4) presented in Section 5.1

An outline of the TACNLP algorithm is now presented, where the fundamental differences between TAUNLP and TACNLP can be observed.

Algorithm 2 Brief outline of TACNLP

1: Given Δt_x^k, Δt_λ^k, $t_{(x,1)} = t_{\lambda,1}$, $t_{(x,2)} = t_{\lambda,2}$, $\hat{\varepsilon}$, initialize $k \leftarrow 0$
2: Given x^k, λ^k and μ^k, compute $a^k = -\nabla_x \phi_A(x^k, \lambda^k; \mu^k)$, $v^k = \frac{1}{2} a^k \Delta t_x^k$, $b^k = \nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k)$, $w^k = \frac{1}{2} b^k \Delta t_\lambda^k$, $c^k = -\mu^k$ and $\underline{d}^k = \frac{1}{2} c^k \Delta t_\mu^k$ (implement with **Scaling** described in **Algorithm 7** if necessary).
3: Let $\varepsilon^k \downarrow 0$ and update $\theta(x^k, \lambda^k; \mu^k)$ as in (5.26)
4: **while** $\theta(x^k, \lambda^k; \mu^k) > \varepsilon^k$ **do**
5: Integrate the system defined by (5.8) - (5.10) over the interval $[t^k,\ t^{k+1}]$ at iteration k to obtain x^{k+1}, λ^{k+1} and μ^{k+1} using Euler's method described by (5.11) - (5.13), and $\hat{x}^{k+1}$ and $\hat{\lambda}^{k+1}$ using Modified Euler's method described in (5.17) - (5.18).

6: Compute $a^{k+1} = -\nabla_x \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$, $b^{k+1} = \nabla_\lambda \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$ and $c^{k+1} = -\mu^{k+1}$, integrate the system defined by (5.8) - (5.10) over the interval $[t^k, t^{k+1}]$ at iteration k to obtain v^{k+1}, w^{k+1} and $\underline{d}^{k+1}$ using Euler's method described by (5.14) - (5.15), and $\hat{v}^{k+1}$ and $\hat{w}^{k+1}$ using Modified Euler's method described by (5.19).

7: Update Δt_x^{k+1} and Δt_λ^{k+1} by implementing an adaptive step size routine to monitor the performance of the integration formula used in step 5.

8: These steps are similar to steps 19 - 23 of **Algorithm 1**, using the optimality conditions stipulated in (5.3) and (5.4).

9: These steps are similar to steps 24 - 28 of **Algorithm 1**, including similar updates for λ^{k+1} and w^{k+1}.

10: Set $k = k+1$ and **go to** 3.

11: **end while**

Remarks on Algorithm 2

Remark 5.2.2. Here we present the integration of x, λ and μ which is activated in line 5 of TACNLP: Euler's method, described in (5.11) - (5.16) is used to integrate the system (5.8) - (5.10). Modified Euler's method described in (5.17) - (5.19), is also used to integrate (5.8) and (5.9). Given (x^k, λ^k, μ^k) and $(\Delta t_x^k, \Delta t_\lambda^k, \Delta_\mu^k)$ the methods calculate $(x^{k+1}, \lambda^{k+1}, \mu^{k+1})$ and $(\hat{x}^{k+1}, \hat{\lambda}^{k+1})$.

Remark 5.2.3. Here we present the integration of v, w and $\underline{d}$ which is activated in line 6 of TACNLP: We update $\nabla_x \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$ and $\nabla_\lambda \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$ and use Euler's method described in (5.11) - (5.16) to integrate the system (5.8) - (5.10). Modified Euler's method described in (5.17) - (5.19), is also used to integrate (5.8) and (5.9). Given $(x^{k+1}, \lambda^{k+1}, \mu^{k+1})$, $(\hat{x}^{k+1}, \hat{\lambda}^{k+1})$, $(v^k, w^k, \underline{d}^k)$ and $(\Delta t_x^k, \Delta t_\lambda^k, \Delta_\mu^k)$ the method calculates $(v^{k+1}, w^{k+1}, \underline{d}^{k+1})$ and $(\hat{v}^{k+1}, \hat{w}^{k+1})$.

Remark 5.2.4. In line 7, an adaptive step size routine is implemented to monitor the error of the integration formula used in line 4 above, and Δt_x^{k+1}, Δt_λ^{k+1} are updated using (5.21).

Remark 5.2.5. In line 8, corrective procedures are implemented pertaining to the descent condition: If at the k-th iteration, the descent condition (4.6) is not satisfied, then the updates described in (4.13) and (4.15) are used to update v^k and x^{k+1}. Similar updates are used for w^k and λ^{k+1}:

$$w^k = \frac{1}{2}(\frac{1}{2}w^k + \frac{1}{2}w^{k+1}), \tag{5.28}$$

$$\lambda^{k+1} = \frac{1}{2}\lambda^{k} + \frac{1}{2}\lambda^{k+1}. \tag{5.29}$$

If the velocity of the particle, v^k, continues to decrease for more than 2 consecutive iterations, then v^k is restarted, as in (4.14). A similar restart is used for w^k:

$$w^k = 0. \tag{5.30}$$

Remark 5.2.6. Since these updates take place at the k-th iteration, they still correspond to the integration within the interval $[t^k,\ t^{k+1}]$. This line is homologous with lines 24-28 of **Algorithm 1**. Notice here that any updates made to w^k and v^k are based on the descent condition (4.6) with respect to v^k. Whenever the descent condition (4.6) is not satisfied we update v^k using (4.13) or (4.14) as well as w^k, using (5.28) or (5.30), since the trajectories $x(t)$ and $\lambda(t)$ are dependent on each other.

Chapter 6

Implementation of Trajectory-Based Algorithms

In this chapter, we present a detailed breakdown of the implementation of TACNLP and compare it with a direct adaptation of TAUNLP [115] for CNLP problems. We denote the adaptation of TAUNLP by ATAUNLP. As per Chapter 5, a few changes were made in TACNLP, in an attempt to improve on the existing framework of TAUNLP. To motivate the new features in TACNLP, we introduce this direct adaptation of TAUNLP for solving CNLPs and compare it with the performance of TACNLP. We present a comparison of the implementations of TACNLP and ATAUNLP, in Section 6.1. The procedures used in TACNLP and ATAUNLP are presented in Section 6.2. Finally the pseudo-codes for the TACNLP and ATAUNLP algorithm are presented in Section 6.3. A numerical comparison of the performance of these two algorithms is presented in Chapter 9.

6.1 Comparison of procedures used in TACNLP and ATAUNLP

In this section, we present the outline of ATAUNLP. ATAUNLP is a direct adaptation of TAUNLP. Recall the synopsis of TAUNLP [115] discussed in Sections 4.1 and 4.2. Recall also the details of TACNLP which was presented in Sections 5.1 and 5.2. TACNLP and ATAUNLP are similar, but there are some crucial differences in the implementation of the two algorithms. We compare and contrast the differences between ATAUNLP and TACNLP to measure the effects of including the adaptive step size routine in the solution process of TACNLP. The main differences in the implementation of the two algorithms is now presented.

6.1.1 Updating x^k and λ^k

Here we present the differences in the x and λ updates for TACNLP and ATAUNLP. We first present the updates used in TACNLP. This is followed by a description of the updates used in ATAUNLP.

TACNLP updates x^k and λ^k at the k-th iteration in $[t^k,\ t^{k+1}]$, using:

- Euler's method, described in (5.11) and (5.12) as well as Modified Euler's method, described in (5.17) and (5.18).
- The assignments described by (4.15) in Section 4.2 and (5.29) in Section 5.2. These assignments are used whenever the descent condition (4.6) is not satisfied

ATAUNLP updates x^k and λ^k at the k-th iteration in $[t^k,\ t^{k+1}]$, using:

- Euler's method, described by (5.11) and (5.12) only.
- The assignments described by (4.15) in Section 4.2 and (5.29) in Section 5.2. These assignments are used whenever the descent condition (4.6) is not satisfied
- The assignments described in (4.15) and (5.29). These assignments are used whenever the angle violation criterion is satisfied.

The differences between TACNLP and ATAUNLP, presented above, are summarized in Table 6.1. The table entry "✓" means the update was used, the table entry "✗" means the update was not used, and the table entry "−" means that update is not applicable in the respective algorithm. Furthermore, the column p_a lists the parameters used in the respective algorithm. An exhaustive list of all the parameters used in TACNLP and ATAUNLP is presented at the end of this chapter, in Table 6.3.

TACNLP and ATAUNLP use the same number of parameters when updating x and λ. As per Table 6.1, the number of parameters used is 7.

Remark 6.1.1. In Table 6.1, the parameters used in Euler's method are identical to those used in Modified Euler's method, so we have not listed them again.

6.1.2 Updating the integration time steps Δt_x^k and Δt_λ^k

The fundamental difference between TACNLP and ATAUNLP is the mechanism for updating the time step. We now present these differences. We first present the updates used in TACNLP. This is followed by a description of the updates used in ATAUNLP.

		TACNLP	p_a	ATAUNLP	p_a	Description
(i)	Euler's method updates: (5.11) - (5.12)	✓	Δt_x Δt_λ Δt_μ	✓	Δt_x Δt_λ Δt_μ	This method is used in both algorithms to integrate the system (5.8) - (5.10)
(ii)	Modified Euler's method updates: (5.17) - (5.18)	✓	–	✗	–	This method is used as part of the adaptive step size routine described in Section 5.2.
(iii)	Descent condition updates: (4.15) and (5.29)	✓	j_x j_λ i_x i_λ	✓	j_x j_λ i_x i_λ	This is used whenever the descent condition (4.6) is not satisfied.
(iv)	Angle violation criterion updates: (4.15), (5.29)	✗	–	✓	–	This update is used whenever the angle violation criterion is satisfied.

Table 6.1 The fundamental differences between the updates x and λ used in TACNLP and ATAUNLP

- TACNLP computes approximate solutions x^k and λ^k using both Euler and modified Euler's method, as described in (5.11), (5.12), (5.17) and (5.18) and uses them within the adaptive step size routine, see (i) and (ii) of Table 6.1. As per the discussion in Sections 5.1 and 5.2, an adaptive step size routine is used in TACNLP to monitor the accuracy of the integration method and to update the time steps accordingly, using (5.21). The accuracy of the integration method is measured by the difference between the Euler's and modified Euler's approximations, as is described in (5.20).

- In ATAUNLP, we compute x^k and λ^k using only Euler's method, described in (5.11) - (5.12), see (i) of Table 6.1. Here, the step sizes Δt_x^k and Δt_λ^k are adjusted based on the angle violation criterion, discussed in Section 4.2. The process for updating Δt_x^k and Δt_λ^k in ATAUNLP is now described.
  Recall that for the unconstrained case, the parameter $\hat{m}$ is used to keep track of the number of times (4.16) is satisfied. For the constrained problem, $\mathscr{P}$, defined by (1.2), we need to monitor the trajectories with respect to x and λ. For this reason, we use $\hat{m}_x$ to keep track of the number of times (6.1) is satisfied and $\hat{m}_\lambda$ to keep track of the number of times (6.2) is satisfied:

$$(a^{k+1})^T(a^k) \leq 0, \tag{6.1}$$

$$(b^{k+1})^T(b^k) \leq 0, \tag{6.2}$$

  where $a^k = -\nabla_x \phi_A(x^k, \lambda^k; \mu)$ and $b^k = \nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k)$. Whenever (6.1) or (6.2) are satisfied, $\hat{m}_x^k$ and $\hat{m}_\lambda^k$ are respectively updated as follows:

$$\hat{m}_x^{k+1} = \hat{m}_x^k + 1, \tag{6.3}$$

$$\hat{m}_\lambda^{k+1} = \hat{m}_\lambda^k + 1, \tag{6.4}$$

  where initially $\hat{m}_x^k = 0$ and $\hat{m}_\lambda^k = 0$. If $\hat{m}_x^k \geq 3$ or $\hat{m}_\lambda^k \geq 3$ then the angle violation criteria is said to be satisfied and we decrease the step sizes respectively, as per (4.19):

$$\begin{aligned} \Delta t_x^{k+1} &= \hat{t}_{(x,1)} \times \Delta t_x^k, \\ \Delta t_\lambda^{k+1} &= \hat{t}_{(\lambda,1)} \times \Delta t_\lambda^k, \end{aligned} \tag{6.5}$$

  where $\hat{t}_{(x,1)} = \hat{t}_{(\lambda,1)} = \frac{1}{2}$. In the event of successful steps, where (6.6) and (6.7) are satisfied:

$$(a^{k+1})^T(a^k) > 0, \tag{6.6}$$

$$(b^{k+1})^T(b^k) > 0, \tag{6.7}$$

we magnify the times steps as per (4.21):

$$\begin{aligned}
\Delta t_x^{k+1} &= \begin{cases} p_x^k \Delta t_x^k & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| < \delta, \\ \Delta t_x^k & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| \geq \delta, \end{cases} \\
\Delta t_\lambda^{k+1} &= \begin{cases} p_\lambda^k \Delta t_\lambda^k & if\ {b^{k+1}}^T b^k > 0, ||\Delta \lambda^k|| < \delta, \\ \Delta t_\lambda^k & if\ {b^{k+1}}^T b^k > 0, ||\Delta \lambda^k|| \geq \delta. \end{cases}
\end{aligned} \tag{6.8}$$

Here $||\Delta x^k||$ is defined as in (4.11), $||\Delta \lambda^k|| = ||w^k|| \Delta t_\lambda^k$ and p_x^k and p_λ^k satisfy:

$$\begin{aligned}
p_x^{k+1} &= \begin{cases} p_x^k + \delta_1 & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| < \delta, \\ p_x^k & if\ {a^{k+1}}^T a^k > 0, ||\Delta x^k|| \geq \delta, \end{cases} \\
p_\lambda^{k+1} &= \begin{cases} p_\lambda^k + \delta_1 & if\ {b^{k+1}}^T b^k > 0, ||\Delta \lambda^k|| < \delta, \\ p_\lambda^k & if\ {b^{k+1}}^T b^k > 0, ||\Delta \lambda^k|| \geq \delta. \end{cases}
\end{aligned} \tag{6.9}$$

A table listing the differences in the mechanisms used to update the time steps Δt_x^k and Δt_λ^k in TACNLP and ATAUNLP, is now presented. From Table 6.2, we see that the total number of parameters used to update the time steps in TACNLP is 5, whereas the total number of parameters used in ATAUNLP is 7.

6.1.3 Scaling

Identical strategies are used for updating the penalty parameter and scaling problems in TACNLP and ATAUNLP. Recall that the details of these were discussed in Section 5.2.

6.1.4 Convergence

The same criteria for establishing the convergence of TACNLP and ATAUNLP is used. This was discussed in Section 5.1 and summarized in (5.27).

		TACNLP	p_a	ATAUNLP	p_a	Description
(i)	Adaptive step size routine described in Section 5.2 using (5.21)	✓	$t_{(x,1)}$ $t_{(x,2)}$ $t_{(\lambda,1)}$ $t_{(\lambda,2)}$ $\hat{\varepsilon}$	✗	–	This is an effective way for controlling the error of the integration method via (5.20).
(ii)	Time step update described in Section 4.1 using (6.5) - (6.8)	✗	–	✓	δ_1 p_x p_λ $\hat{t}_{(x,1)}$ $\hat{t}_{(\lambda,1)}$ $\hat{m}_x$ $\hat{m}_\lambda$	This is used to update the time steps Δt_x^k and Δt_λ^k, but is not as effective as an adaptive step size method.

Table 6.2 The fundamental differences between the time step updates used in TACNLP and ATAUNLP

Remark 6.1.2. In some cases, particularly when we are far from the limit point $\{x^*, \lambda^*\}$, decreasing the step size whenever the angle between two successive gradient estimates is greater than $\frac{\pi}{2}$, hinders the convergence. This has motivated our use of the adaptive step size routine listed in (i) of Table 6.2, instead of using the mechanism for updating Δt, proposed by Snyman [115].

6.2 Implementation of the procedures used in TACNLP and ATAUNLP

We now present each of the items discussed in Section 6.1, in pseudo-code format. First, we present the pseudo-code for the adaptive step size method for x and λ. Second, we present the pseudo-codes using two different penalty parameter *updating schemes*. We then present pseudo-codes for three different penalty parameter *updating criteria*, and finally, we present the pseudo-code for the scaling routine. The pseudo-codes are further explained by remarks on the steps used, below each code. Implementation differences between TACNLP and ATAUNLP in each pseudo-code are also given in the remarks after each code. We begin with the adaptive step size.

6.2.1 Pseudo-code for the adaptive step size routine

The pseudo-code for the adaptive set size routine is given below. Recall the parameters used for the adaptive step size routine were listed in Table 6.2.

Algorithm 3 Adaptive step size scheme used in TACNLP

1: Given $x^{(0)}$ and $\lambda^{(0)}$
2: Compute $\{x^{k+1}, \lambda^{k+1}\}$ and $\{\hat{x}^{k+1}, \hat{\lambda}^{k+1}\}$ as in (5.13), (5.14), (5.17) and (5.18) respectively
3: Compute S_x and S_λ as in (5.20) and compare this with the user supplied tolerance $\hat{\varepsilon}$
4: **if** the estimates (S_x and/or S_λ) $< \hat{\varepsilon}$ **then**
5: accept the step Δt_x^{k+1} and Δt_λ^{k+1} respectively
6: **else** reject the step Δt_x^{k+1} and Δt_λ^{k+1} respectively
7: **end if**

Remarks on Algorithm 3

Remark 6.2.1. In line 2 of **Algorithm 3**, $\{x^{k+1}, \lambda^{k+1}\}$ and $\{\hat{x}^{k+1}, \hat{\lambda}^{k+1}\}$ are computed using Euler's and Modified Euler's method respectively.

Remark 6.2.2. In lines 3-7, the error estimates S_x and S_λ are calculated as in (5.20) and compared to the user prescribed tolerance $\hat{\varepsilon}$ to determine whether to decrease or increase Δt_x^k and Δt_λ^k. The step sizes are updated accordingly, using (5.21).

Remark 6.2.3. The adaptive step size routine described above, is only implemented in TACNLP and not in ATAUNLP.

6.2.2 Pseudo-code for the step size update used in ATAUNLP

We now present the pseudo-code for the time step size update used in ATAUNLP. New parameters have been introduced for the constrained case, as opposed to TAUNLP, see Table 6.2.

Algorithm 4 Step size update used in ATAUNLP

1: Given $\hat{m}_x^k$, $\hat{m}_\lambda^k$, p_x^k, p_λ^k, δ and δ_1
2: **if** $\hat{m}_x^k \geq 3$ or $\hat{m}_\lambda^k \geq 3$ **then**
3: Update Δt_x^{k+1} or Δt_λ^{k+1} respectively using (6.5)
4: **else**
5: Update Δt_x^{k+1} or Δt_λ^{k+1} respectively, using (6.8) where p_x^k and p_λ^k satisfy (6.9).
6: **end if**

Remarks on Algorithm 4

Remark 6.2.4. In lines 2 - 7, Δt_x^k and Δt_λ^k are decreased if (6.1) and (6.2) respectively are satisfied for 3 consecutive iterations. In the event of a good step, when (6.1) and (6.2) are not satisfied, Δt_x^k and Δt_λ^k are increased by the factors p_x^k and p_λ^k respectively.

Remark 6.2.5. The step size update in **Algorithm 4** is only implemented in ATAUNLP.

6.2.3 Pseudo-code for the penalty parameter updating strategy proposed in this thesis

We now consider the pseudo-code for the penalty parameter updating strategy proposed in this thesis. We introduce the parameter r, where $r \in (0,1)$, and the functions $\underline{T}_i(x,\lambda),\ i = 1,\ldots,3$. The significance of the number 3 is that three different penalty parameter *updating criteria* will be presented. For each of these penalty parameter *updating criteria*, $T_i(x,\lambda)$ will be defined differently. This is done to determine which penalty parameter *updating criteria*, based on the different definitions of $T_i(x,\lambda)$, is most effective.

The pseudo-code for the penalty parameter updating strategy, which consists of an *updating scheme* and *updating criteria*, is as follows

Algorithm 5 Penalty parameter updating strategy for ATAUNLP and TACNLP

1: Given x^k, x^{k+1}, λ^k, λ^{k+1}, c^k, $\underline{d}^k$ and r, we define $\Psi_i(x,\lambda) = ||\underline{T}_i(x,\lambda)||, i = 1,\ldots 3$
2: **if**

$$\Psi_i(x^{k+1},\lambda^{k+1}) \leq r\Psi_i(x^k,\lambda^k) \tag{6.10}$$

then
3: set $\mu^{k+1} = \mu^k$
4: **else**
5: Update μ^{k+1} as in (5.13)
6: **end if**

Remarks on Algorithm 5

Remark 6.2.6. The penalty parameter *updating scheme* proposed in this thesis is described in line 5 by (6.10). This describes "*how*" μ^k is updated, i.e., as the solution to the differential equation (5.10). There needs to be a further mechanism in place which tells us "*when*" μ^k should be updated. We consider three such *updating criteria*, which are now described in detail.

Remark 6.2.7. The first penalty parameter *updating criterion*, which we refer to as PC1, considered in **Algorithm 5** is updated based on whether or not feasibility and

complementarity of the respective constraints are improved. We let $\Psi_1(x,\lambda) = ||\underline{T}_1(x,\lambda)||$, where

$$\underline{T}_1(x,\lambda) = \begin{pmatrix} c_i(x), & i \in E \\ \min\{c_i(x),\lambda_i\}, & i \in I \end{pmatrix}.$$

PC1, derived from Andreani et al, [6], ensures that feasibility and complementarity are obtained and maintained throughout the minimization routine in TACNLP and ATAUNLP.

Remark 6.2.8. For the second penalty parameter *updating criterion* PC2, we let $\Psi_2(x,\lambda) = ||\underline{T}_2(x,\lambda)||$, where

$$\underline{T}_2(x,\lambda) = \begin{pmatrix} \nabla_x \phi_A(x,\lambda;\mu) & \\ c_i(x), & i \in E \\ \min\{c_i(x),\lambda_i\}, & i \in I \end{pmatrix},$$

and $\phi_A(x,\lambda;\mu)$ is the augmented Lagrangian defined in (2.30). PC2 does not only require feasibility and complementarity of the respective constraints, but also requires an improvement in $\nabla\phi_A$. This criterion, derived from Birgin et al.[18] is slightly stronger than the first criteria using $\underline{T}_1(x,\lambda)$, in that it requires the resulting sequence of iterates to converge to a KKT point.

Remark 6.2.9. For the third and final penalty parameter *updating criterion* PC3, we let $\Psi_3(x,\lambda) = ||\underline{T}_3(x,\lambda)||$, where

$$\underline{T}_3(x,\lambda) = \begin{pmatrix} c_i(x), & i \in E \\ \lambda_i c_i(x), & i \in I \end{pmatrix}.$$

We have proposed PC3, which is a slightly more relaxed version of PC1 in that it requires that $\lim_{k\to\infty} \lambda_i c_i(x) = 0$, as opposed to $\lim_{k\to\infty} \min\{c_i(x),\lambda_i\} = 0$. PC3 proves to exhibit more favorable results than PC1 and PC2, as we will see in Section 9.3.

Remark 6.2.10. In lines 2-5 of **Algorithm 5**, depending on whether or not the *updating criterion* described is satisfied, μ^{k+1} either remains unchanged or μ^{k+1} is decreased using (5.13).

Remark 6.2.11. The penalty parameter updating strategy described above is implemented identically in TACNLP and ATAUNLP.

6.2.4 Pseudo-code for the conventional penalty parameter updating strategy

In the previous subsection we have implemented the updating of μ^k based on the differential equation solution approach. We now present the pseudo-code for a conventional penalty parameter *updating scheme* used in, for example, Andreani et al, [6, 7] and Birgin et

al.[18, 19]. In Chapter 9, we will compare the performance of our trajectory-based algorithms using this conventional *updating scheme* and the *updating scheme* described in **Algorithm 5**.

Algorithm 6 Conventional penalty parameter updating strategy

1: given x^k, λ^k, x^{k+1}, λ^{k+1} and r
2: **if** $\Psi_i(x^k,\lambda^{k+1}) \leq r\Psi_i(x^{k-1},\lambda^k),\ i = 1,\ldots,3$, where Ψ_i corresponds to either PC1, PC2 or PC3 in **Algorithm 5 then**
3: set $\mu^{k+1} = \mu^k$
4: **else**
5: set $\mu^{k+1} = \gamma\mu^k$, where $0 < \gamma < 1$
6: **end if**

Remark on Algorithm 6

Remark 6.2.12. As with **Algorithm 5**, the conventional updating strategy consist of an *updating scheme* and an *updating criterion*. The *updating citeria* used in **Algorithm 6** are identical to those used in **Algorithm 5**. The *updating scheme* is described below.

Remark 6.2.13. In line 5 of **Algorithm 6**, the penalty parameter is updated by a fixed value, γ, whenever complementarity or feasibility of the constraints is not improved. This is the fundamental difference between the newly introduced penalty parameter *updating scheme* and the penalty parameter *updating scheme* described in **Algorithm 6**.

Throughout the rest of this thesis we refer to the penalty parameter *updating scheme* introduced in this thesis, as the new$-\mu$ scheme, and the conventional *updating scheme* as the conventional$-\mu$ scheme. In Chapter 9, we test the effectiveness of using our penalty parameter *updating scheme* by comparing the performance of our algorithm using the schemes in **Algorithms 5** and **6**.

Remark 6.2.14. When the conventional$-\mu$ scheme presented in **Algorithm 6** is implemented, it is done so identically in TACNLP and ATAUNLP.

6.2.5 Pseudo-code for the scaling routine

The strategy for scaling is now summarized in the form of a pseudo-code below.

Algorithm 7 Scaling used in TACNLP and ATAUNLP

1: Given x^k, λ^k, μ^k, ε, τ and q random vectors x^q
2: Compute $\nabla f(x^q)$, $c_i(x^q)$ and $\nabla c_i(x^q)$ for each x^q, to obtain scaling factors $(\nabla f)_s$, $(c_i)_s$ and $(\nabla c_i)_s$ and compare these with the user supplied tolerance τ
3: Compute $\nabla f(x^k)$, $c_i(x^k)$ and $\nabla c_i(x^k)$
4: **if** $(\nabla f)_s) > \tau$, $(c_i)_s > \tau$, $(\nabla c_i)_s > \tau$ **then**
5: scale: $\nabla f(x^k) = \frac{\nabla f(x^k)}{(\nabla f)_s}$, $c_i(x^k) = \frac{c_(x^k)}{(c_i)_s}$ and $\nabla c_(x^k) = \frac{\nabla c_(x^k)}{(\nabla c_i)_s}$ respectively
6: **if** $\nabla f(x^k) < (\nabla f)_s$, $c_i(x^k) < (c_i)_s$, $\nabla c(x^k) < (\nabla c)_s$ **then**
7: do not scale $\nabla f(x^k)$, $c_i(x^k)$ and $\nabla c(x^k)$ respectively
8: **end if**
9: **end if**

Remarks on Algorithm 7

Remark 6.2.15. Lines 1-3 of **Algorithm 7** entail generating random vectors as per Section 5.2.3, which will be used to obtain a mean value for the functions appearing in the Augmented Lagrangian ϕ_A, i.e. $\nabla f(x)$, $c_i(x)$ and $\nabla c_i(x)$.

Remark 6.2.16. In line 4, the strategy for scaling these functions is described. Recall that the details of this were presented in Section 5.2.

Remark 6.2.17. The scaling mechanism described above is implemented identically in TACNLP and ATAUNLP.

6.3 The TACNLP and ATAUNLP algorithm

In this final section of Chapter 6 we present the pseudo-codes for TACNLP and ATAUNLP, as well as a list of all the parameters used in the implementation of both algorithms, in Table 6.3. We begin by presenting the pseudo-code for TACNLP.

6.3.1 pseudo-code for TACNLP

Having described **Algorithms 3 - 7**, in this we subsection present the pseudo-code for TACNLP in **Algorithm 8** .

Algorithm 8 TACNLP

Given μ_{min}, μ_{max}, Δt_x^k, Δt_λ^k, Δt_μ, Δt_{min}, Δt_{max}, τ, q, $\hat{\varepsilon}$, ε^k, $i_x = 0 = i_\lambda$, $j_x = 2 = j_\lambda$, **counter**$= 0$, initialize $k \leftarrow 0$
Given x^k, λ^k and μ^k, compute: $a^k = -\nabla_x \phi_A(x^k, \lambda^k; \mu^k)$, $v^k = \frac{1}{2} a^k \Delta t_x^k$, $b^k = \nabla_\lambda \phi_A(x^k \lambda^k; \mu^k)$, $w^k = \frac{1}{2} b^k \Delta t_\lambda^k$, $c^k = -\mu^k$ and $\underline{d}^k = \frac{1}{2} c^k \Delta_u^k$ (implement with **Scaling** described in **Algorithm 7** if necessary)
Let $\varepsilon^k \downarrow 0$ and update $\theta(x^k, \lambda^k; \mu^k)$ as in (5.26)
while $\theta(x^k, \lambda^k; \mu^k) > \varepsilon^k$ **do**
    Compute $\Psi_i(x^k, \lambda^k)$ as per **Algorithm 5**.
    Integrate the system defined by (5.8) - (5.10) over the interval $[t^k,\ t^{k+1}]$, at iteration k to obtain $\{x^{k+1}, \lambda^{k+1}, \mu^{k+1}\}$ using Euler's method described in (5.11) - (5.13)
    Compute $\Psi_i(x^{k+1}, \lambda^{k+1})$ as per **Algorithm 5**
    Update μ^{k+1} using **Algorithm 5**
    Compute $a^{k+1} = -\nabla_x \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$, $b^{k+1} = \nabla_\lambda \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$, $c^{k+1} = -\mu^{k+1}$ and integrate the system defined by (5.8) - (5.10) over the interval $[t^k,\ t^{k+1}]$ at iteration k to obtain $\{v^{k+1}, w^{k+1}, \underline{d}^{k+1}\}$, using Euler's method described by (5.14) - (5.16).
    Update $\theta(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$ as in (5.26)
    if $\theta(x^{k+1}, \lambda^{k+1}; \mu^{k+1}) \leq \varepsilon^k$ **then**
        STOP
    else
        if counter$= 0$ **then**
            Compute the Modified Euler estimates $\{\hat{x}^{k+1}, \hat{\lambda}^{k+1}\}$ and $\{\hat{v}^{k+1}, \hat{w}^{k+1}\}$ using (5.17), (5.18) and (5.19) respectively and update Δt_x^{k+1} and Δt_λ^{k+1} using **Algorithm 3**
        elseif counter$= 1$
            $\Delta t_x^{k+1} = \Delta t_x^k$ and $\Delta t_\lambda^{k+1} = \Delta t_\lambda^k$
            if $||v^{k+1}|| > ||v^k||$ **then** set $i_x = 0$, $i_\lambda = 0$ and **go to** 27
            else
                set **counter**$= 1$, $i_x = i_x + 1$ and $i_\lambda = i_\lambda + 1$ and **go to** 23
            end if
        end if
        if $i_x < j_x$ and $i_\lambda < j_\lambda$ **then** update $\{v^{k+1}, x^{k+1}\}$, as in (4.13) and (4.15), and $\{w^{k+1}, \lambda^{k+1}\}$ as in (5.28) and (5.29), and **go to** 10

24: **else**
25: Update $\{v^{k+1}, x^{k+1}\}$ as in (4.14), (4.15), and set $j_x = 1$, and $\{\lambda^{k+1}, w^{k+1}\}$ as in (5.29) and (5.30), set $j_\lambda = 1$, and **go to** 10
26: **end if**
27: Set **counter**$= 0$ and $k = k+1$ and **go to** 3
28: **end if**
29: **end while**

Remarks on Algorithm 8

Remark 6.3.1. Before integrating, the components v^0 and w^0 are initialized similarly to the unconstrained case, using (6.11) and (6.12):

$$v^0 = -\frac{1}{2}\nabla_x \phi_A(x^0, \lambda^0; \mu^0)\Delta t_x^0 \tag{6.11}$$

$$w^0 = \frac{1}{2}\nabla_\lambda \phi_A(x^0, \lambda^0; \mu^0)\Delta t_\lambda^0 \tag{6.12}$$

Remark 6.3.2. The integration of x, λ and μ is activated in line 6 of TACNLP: Euler's method, described in (5.11) - (5.16) is used to integrate the system (5.8) - (5.10) to obtain x^{k+1}, λ^{k+1} and μ^{k+1}. Given $\{x^k, \lambda^k, \mu^k\}$ and $\{\Delta t_x^k, \Delta t_\lambda^k, \Delta_\mu^k\}$ the methods calculate $\{x^{k+1}, \lambda^{k+1}, \mu^{k+1}\}$.

Remark 6.3.3. The penalty parameter updating scheme is described in lines 7-8, where μ^{k+1} is updated based on one of the criteria described in **Algorithm 5**. Even though μ^{k+1} is obtained as a solution to (5.10) at every iteration, this value is only assigned based on **Algorithm 5**.

Remark 6.3.4. In line 9 of TACNLP we present the integration of v, w and $\underline{d}$: We update $a^{k+1} = -\nabla_x \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$, $b^{k+1} = \nabla_\lambda \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$ and $c^{k+1} = \mu^{k+1}$. We then use Euler's method described in (5.11) - (5.16) to integrate the system (5.8) - (5.10) to obtain v^{k+1}, w^{k+1} and $\underline{d}^{k+1}$. Given $\{x^{k+1}, \lambda^{k+1}, \mu^{k+1}\}$, $\{v^k, w^k, \underline{d}^k\}$ and $\{\Delta t_x^k, \Delta t_\lambda^k, \Delta_\mu^k\}$ the method calculates $\{v^{k+1}, w^{k+1}, \underline{d}^{k+1}\}$

Remark 6.3.5. The Modified Euler estimates $\{\hat{x}^{k+1}, \hat{\lambda}^{k+1}\}$ and $\{\hat{v}^{k+1}, \hat{w}^{k+1}\}$ are computed and the adaptive step size routine, described in **Algorithm 3** is implemented in lines 14-18 of TACNLP, provided **counter**$= 0$. This is done so that the adaptive step size routine is only implemented to detect the error of the integration method, not as part of the corrective procedures used when the descent condition, (4.6), is not satisfied.

Remark 6.3.6. In lines 19 -26, adjustments are made based on whether or not the descent condition, (4.6), is met. Once the descent condition (4.6) is satisfied, the iterative process is restarted at line 27.

We now present the pseudo-code for ATAUNLP.

6.3.2 pseudo-code for ATAUNLP

Having described **Algorithms 3 - 7**, in this we subsection present the pseudo-code for ATAUNLP in **Algorithm 9**.

Algorithm 9 ATAUNLP

1: Given μ_{min}, μ_{max}, Δt_x^k, Δt_λ^k, Δt_μ^k, Δt_{min}, Δt_{max}, $\hat{m}_x^k$, $\hat{m}_\lambda^k$, τ, q, $\hat{\varepsilon}$, ε^k, $\delta = 1$, $\delta_1 = 0.001$, set $i_x = 0 = i_\lambda$, $j_x = 2 = j_\lambda$, $s_x = 0 = s_\lambda$, $p_x = 1 = p_\lambda$, initialize $k \leftarrow 0$

2: Given x^k, λ^k and μ^k, compute: $a^k = -\nabla_x \phi_A(x^k, \lambda^k; \mu^k)$, $v^k = \frac{1}{2} a^k \Delta t_x^k$, $b^k = \nabla_\lambda \phi_A(x^k \lambda^k; \mu^k)$, $w^k = \frac{1}{2} b^k \Delta t_\lambda^k$, $c^k = -\mu^k$ and $\underline{d}^k = \frac{1}{2} c^k \Delta_u^k$ (implement with **Scaling** described in **Algorithm 7** if necessary)

3: Let $\varepsilon^k \downarrow 0$ and update θ as in (5.26)

4: **while** $\theta(x^k, \lambda^k; \mu^k) > \varepsilon^k$ **do**

5: Compute $\Psi_i(x^k, \lambda^k)$ as per **Algorithm 5**

6: Compute $||\Delta x^k|| = ||v^k|| \Delta t_x^k$ and $||\Delta \lambda^k|| = ||w^k|| \Delta t_\lambda^k$

7: **if** $||\Delta x^k|| < \delta$ and $||\Delta \lambda^k|| < \delta$ **then go to** 11

8: **else**

9: set $v^k = \frac{\delta}{\Delta t_x^k ||v^k||}$ or $w^k = \frac{\delta}{\Delta t_\lambda^k ||w^k||}$ respectively and **go to** 12

10: **end if**

11: Set $p_x^{k+1} = p_x^k + \delta_1$, $p_\lambda^{k+1} = p_\lambda^k + \delta_1$ and update Δt_x^{k+1} and Δt_λ^{k+1} as in (6.8) and **go to** 12

12: **if** $\hat{m}_x^k < 3$ and $\hat{m}_\lambda^k < 3$ **then go to** 16

13: **else**

14: Update $\{v^{k+1}, x^{k+1}, \Delta t_x^{k+1}\}$ or $\{w^{k+1}, \lambda^{k+1}, \Delta t_\lambda^{k+1}\}$ respectively, as in **Algorithm 4** and **go to** 16.

15: **end if**

16: Integrate the system defined by (5.8) - (5.10) over the interval $[t^k,\ t^{k+1}]$, at iteration k to obtain$\{x^{k+1}, \lambda^{k+1}, \mu^{k+1}\}$ using Euler's method described in (5.11) - (5.13)

17: Compute $\Psi_i(x^{k+1}, \lambda^{k+1})$ as per **Algorithm 5**

18: Update μ^{k+1} using **Algorithm 5**

19: Compute $a^{k+1} = -\nabla_x \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$, $b^{k+1} = \nabla_\lambda \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$, $c^{k+1} = -\mu^{k+1}$ and integrate the system defined by (5.8) - (5.10) over the interval $[t^k,\ t^{k+1}]$ at iteration k to obtain $\{v^{k+1}, w^{k+1}, \underline{d}^{k+1}\}$, using Euler's method described by (5.14) - (5.16).
20: **if** $(a^{k+1})^T a^k > 0$ and $(b^{k+1})^T b^k > 0$ **then**
21: set $\hat{m}_x = 0$ and $\hat{m}_\lambda = 0$ respectively and **go to** 25
22: **else**
23: set $\hat{m}_x = \hat{m}_x + 1$, $p_x = 1$ or $\hat{m}_\lambda = \hat{m}_\lambda + 1, p_\lambda = 1$ respectively and **go to** 25
24: **end if**
25: Update $\theta(x^{k+1}, b^{k+1}; \mu^{k+1})$ as in (5.26)
26: **if** $\theta(x^{k+1}, b^{k+1}; \mu^{k+1}) \leq \varepsilon^k$ **then**
27: **STOP**
28: **else**
29: **if** $||v^{k+1}|| > ||v^k||$ **then** set $i_x = 0$ and $i_\lambda = 0$ and **go to** 37
30: **else**
31: set $i_x = i_x + 1$ and $i_\lambda = i_\lambda + 1$ and **go to** 33
32: **end if**
33: **if** $i_x < j_x$ and $i_\lambda < j_\lambda$ **then** update $\{v^{k+1}, x^{k+1}\}$, as in (4.13) and (4.15), and $\{w^{k+1}, \lambda^{k+1}\}$ as in (5.28) and (5.29), and **go to** 25
34: **else**
35: Update $\{v^{k+1}, x^{k+1}\}$ as in (4.14), (4.15), and set $j_x = 1$, and $\{\lambda^{k+1}, w^{k+1}\}$ as in (5.29) and (5.30), set $j_\lambda = 1$, and **go to** 25
36: **end if**
37: $k = k + 1$ and **go to** 3
38: **end if**
39: **end while**

Remarks on Algorithm 9

Remark 6.3.7. ATAUNLP initializes v^0 and w^0 the same way as TACNLP does.

Remark 6.3.8. The step sizes Δx^k and $\Delta \lambda^k$ are computed in line 6 of ATAUNLP and the parameters used to update Δt_x^{k+1} and $\Delta t_{\lambda^{k+1}}$ are updated accordingly, in lines 7-11.

Remark 6.3.9. Lines 12-15 detect whether the angle violation criterion has been satisfied and updates Δt_x^{k+1}, Δt_λ^{k+1}, x^{k+1}, λ^{k+1}, v^{k+1} and w^{k+1} accordingly.

Remark 6.3.10. The integration of x, λ and μ is activated in line 16 of ATAUNLP: Euler's method, described in (5.11) - (5.16) is used to integrate the system (5.8) - (5.10) to obtain

x^{k+1}, λ^{k+1} and μ^{k+1}. Given $\{x^k, \lambda^k, \mu^k\}$ and $\{\Delta t_x^k, \Delta t_\lambda^k, \Delta_\mu^k\}$ the methods calculate $\{x^{k+1}, \lambda^{k+1}, \mu^{k+1}\}$.

Remark 6.3.11. The penalty parameter updating scheme is described in lines 17-18, where μ is updated based on one of the criteria described in **Algorithms 5**.

Remark 6.3.12. The integration of v, w and $\underline{d}$, is activated in line 18 of ATAUNLP: We update $a^{k+1} = -\nabla_x \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$, $b^{k+1} = \nabla_\lambda \phi_A(x^{k+1}, \lambda^{k+1}; \mu^{k+1})$ and $c^{k+1} = -\mu^{k+1}$. We then use Euler's method described in (5.11) - (5.13) to integrate the system (5.8) - (5.10) to obtain v^{k+1}, w^{k+1} and $\underline{d}^{k+1}$. Given $\{x^{k+1}, \lambda^{k+1}, \mu^{k+1}\}$, $\{v^k, w^k, \underline{d}^k\}$ and $\{\Delta t_x^k, \Delta t_\lambda^k, \Delta_\mu^k\}$ the method calculates $\{v^{k+1}, w^{k+1}, \underline{d}^{k+1}\}$.

Remark 6.3.13. The angle between the two successive estimates a^k and a^{k+1}, as well as b^k and b^{k+1} are checked in line 20 and the parameters contributing to updating Δt_x^{k+1} and Δt_λ^{k+1} are updated accordingly in lines 21-23. This procedure is described in **Algorithm 4**.

Remark 6.3.14. In lines 29 - 36, adjustments are made based on whether or not the descent condition, (4.6), is met. Once the descent condition (4.6) is satisfied, the iterative process is restarted at line 37.

Remark 6.3.15. Note that the fundamental difference between ATAUNLP and TACNLP is described in lines 6-15 and lines 20-24 of ATAUNLP.

We now present a list of all the parameters used in TACNLP and ATAUNLP.

6.3.3 Parameters used in the experiments

Apart from a few exceptions which we will discuss in more detail in Chapter 9, TACNLP and ATAUNLP, were initialized with the following parameters. The table entry "$-$" means the parameter was not used in the respective algorithm.

Description	**Parameter**	TACNLP	ATAUNLP
		Typical value used	Typical value used
Maximum value μ can attain	μ_{max}	1	1
Minimum value μ can attain	μ_{min}	0.1	0.1
Maximum value Δt can attain	Δt_{max}	0.2	0.2
Minimum value Δt can attain	Δt_{min}	0.01	0.01
The time step with respect to x	Δt_x	$\in [0.01,\ 0.2]$	$\in [0.01,\ 0.2]$
The time step with respect to λ	Δt_λ	$\in [0.01,\ 0.2]$	$\in [0.01,\ 0.2]$
The time step with respect to μ	Δt_μ	$\in [0.01,\ 0.2]$	$\in [0.01,\ 0.2]$

Description	**Parameter**	TACNLP	ATAUNLP
		Typical value used	Typical value used
Tolerance for the discretization error of Euler's method	$\hat{\varepsilon}$	10^{-2}	–
The factor Δt_x is decreased by	$t_{(x,1)}$	0.5	–
The factor Δt_x is increased by	$t_{(x,2)}$	2	–
The factor Δt_λ is decreased by	$t_{(\lambda,1)}$	0.5	–
The factor Δt_λ is increased by	$t_{(\lambda,2)}$	2	–
The maximum allowable size for the space steps Δx and $\Delta\lambda$	δ	–	1
Used for the magnification of Δt_x and Δt_λ	δ_1	–	0.001
The factor Δt_x is decreased by	$\hat{t}_{(x,1)}$	–	0.5
The factor Δt_λ is decreased by	$\hat{t}_{(\lambda,1)}$	–	0.5
Used for the magnification of Δt_x	p_x	–	1
Used for the magnification of Δt_λ	p_λ	–	1
Scaling parameter	τ	$\in (100,\ 1000)$	$\in (100,\ 1000)$
Scaling parameter	l_i	-50	-50
Scaling parameter	u_i	50	50
The number of random decision variables generated for scaling purposes	q	100	100
The tolerance for the stopping criteria	ε	10^{-3}	10^{-3}
Used to keep track of the desired progress of the trajectory wrt x	$\hat{m}_x$	–	2
Used to keep track of the desired progress of the trajectory wrt λ	$\hat{m}_\lambda$	–	2
Used to ensure that the descent condition, (4.6), is satisfied	j_x	2	2

Description	**Parameter**	TACNLP	ATAUNLP
		Typical value used	Typical value used
Used to ensure that the descent condition, (4.6), is satisfied	j_λ	2	2
Used to ensure that the descent condition, (4.6), is satisfied	i_x	$\in [0,\ 2]$	$\in [0,\ 2]$
Used to ensure that the descent condition, (4.6), is satisfied	i_λ	$\in [0,\ 2]$	$\in [0,\ 2]$

Table 6.3 A list of the parameters used in the implementation of TACNLP and ATAUNLP

In total, ATAUNLP uses 3 more parameters than TACNLP. These parameters contribute to the time step update in ATAUNLP. As we will see in Chapter 9, these impact significantly on the performance of the methods.

Chapter 7

Convergence analysis

In this chapter, we establish convergence properties of TACNLP. We begin with the global convergence analysis in Section 7.1, and end the chapter with the local convergence discussion of TACNLP in Section 7.2.

7.1 Global convergence analysis

In this section, we discuss the global convergence of TACNLP. We look specifically at the convergence of the algorithm using PC2 since this is the most stringent of the three penalty parameter updating criteria. We begin with some assumptions which are central to our local convergence discussion.

Assumption 1. *The velocity v^* of the stationary point x^* is null, i.e., $||v^*|| = 0$.*

Assumption 2. *The velocity w^* of the stationary point λ^* is null, i.e., $||w^*|| = 0$.*

Assumption 3. *The set Ω_c is compact.*

Assumption 4. *The function $f(x)$ is bounded on the set Ω_c.*

Assumption 5. *MFCQ (presented in Chapter 2), is satisfied at x^* and there is only one vector λ^* of associated Lagrange multipliers.*

Assumption 6. *For AL defined by* (2.30)*, the following relation holds for feasible $\{x^*, \lambda^*\}$:*

$$\phi_A(x^*, \lambda : \mu) \leq \phi_A(x^*, \lambda^* : \mu) \leq \phi_A(x, \lambda^* : \mu).$$

The following lemma is crucial for the convergence discussion.

Lemma 7.1.1. *Assume that $\{x^k\}$ is a sequence generated by TACNLP. Then,*

$$\lim_{k\to\infty} c_i(x^k) = 0,\ i \in E,$$

and

$$\lim_{k\to\infty} \lambda_i^k c_i(x^k) = 0,\ i \in I.$$

Proof. Consider the Lagrange multiplier updates defined by (5.12):

$$\lambda_i^{k+1} = \lambda_i^k + w_i^k \Delta t_\lambda^k,\ i \in E,$$

and

$$\lambda_i^{k+1} = \max\{0, \lambda_i^k + w_i^k \Delta t_\lambda^k\},\ i \in I,$$

where

$$w^k = w^{k-1} + \nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k)\Delta t_\lambda^k \tag{7.1}$$

and the i-th component of $\nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k)$ is:

$$\begin{cases} -c_i(x^k), & i \in E \cup (I \cap A_s(x)), \\ -\mu^k \lambda_i^k, & i \in I \setminus A_s(x). \end{cases} \tag{7.2}$$

We obtain (7.2) by differentiating (2.30) with respect to λ. By **Assumption 2**, we know that

$$||w^*|| = 0.$$

We are required to prove that the sequence $\{w^k\}$ converges to w^*. To do so, we define the function $D(\lambda)$ as in (2.5):

$$\begin{aligned} D(\lambda) &= \inf_{x\in\Omega_c} \phi_A(x^k, \lambda^k; \mu^k) \\ &= \inf_{x\in\Omega_c} \left(f(x) - \begin{cases} \sum_i \lambda_i^k c_i(x) - \sum_i \frac{1}{2\mu^k} c_i^2(x), & i \in E \cup (I \cap A_s(x)); \\ \sum_i \frac{\mu^k}{2} (\lambda_i^k)^2, & i \in I \setminus A_s(x). \end{cases} \right), \end{aligned} \tag{7.3}$$

where the Lagrangian in (2.5) is replaced by the augmented Lagrangian. Provided $D(\lambda)$ is bounded below, for $\lambda_i \in I$, this function yields a lower bound on the optimal value $f(x^*)$. We know that MFCQ is equivalent to the boundedness of the set of Lagrange multiplier vectors λ^*. Therefore, by **Assumption 5**, $D(\lambda)$ is bounded below.

We now show that $D(\lambda)$ yields a lower bound on $f(x^*)$. At some feasible $\bar{x}^k$, we have that

$$\sum_i (\lambda_i^k c_i(\bar{x}^k)) = 0,\ i \in E \cup (I \cap A_s(x)),$$

since $c(\bar{x}^k) = 0,\ i \in E$ and $c(\bar{x}^k) = 0,\ i \in I \cap A_s(x)$. By the same argument

$$\sum_i \frac{1}{2\mu} c_i(\bar{x}^k)^2 = 0,\ i \in E \cup (I \cap A_s(x)).$$

Furthermore, since $\mu > 0$,

$$\sum_i \frac{\mu}{2} (\lambda_i^k)^2 > 0,\ i \in I \setminus A_s(x),$$

by the strict complementary condition. It thus follows from (7.3) that:

$$\begin{aligned} \phi_A(\bar{x}, \lambda; \mu) &= f(\bar{x}) - \begin{cases} \sum_i \lambda_i^k c_i(x) - \sum_i \frac{1}{2\mu^k} c_i^2(x), & i \in E \cup (I \cap A_s(x)); \\ \sum_i \frac{\mu^k}{2} (\lambda_i^k)^2, & i \in I \setminus A_s(x) \end{cases} \\ &\leq f(\bar{x}). \end{aligned} \tag{7.4}$$

Therefore,

$$D(\lambda) = \min_{x \in \Omega_c} \phi_A(x, \lambda; \mu) \leq f(x^*),$$

i.e.,

$$\phi_A(x^*, \lambda; \mu) \leq f(x^*). \tag{7.5}$$

This proves that ϕ_A is bounded above by $f(x^*)$.
Now, recall that at every iteration of **Algorithm 8**, the augmented Lagrangian $\phi_A(x, \lambda; \mu)$, defined in (2.30), is maximized with respect to λ. This follows from the nature of the differential equation (5.2). Consequently, at each k-th iteration we have that

$$\phi_A(x^k, \lambda^{k+1}; \mu^k) \geq \phi_A(x^k, \lambda^k; \mu^k).$$

Therefore, by the boundedness of ϕ_A, for large k and $\varepsilon > 0$ arbitrarily small, we have

$$||\phi_A(x^k, \lambda^{k+1}; \mu^k) - \phi_A(x^k, \lambda^k; \mu^k)|| \leq \varepsilon. \tag{7.6}$$

Since (5.2) implies conservation of energy, the following holds

$$\Delta\phi_A^k = \phi_A(x^k, \lambda^{k+1}; \mu^k) - \phi_A(x^k, \lambda^k; \mu^k) = -\hat{T}(x^k, \lambda^{k+1}; \mu^k) + \hat{T}(x^k, \lambda^k; \mu^k) = -\Delta\hat{T}^k, \tag{7.7}$$

where $\phi_A(x^k, \lambda^k; \mu^k)$ is the potential energy, of the particle λ^k and

$$\hat{T}(x^k, \lambda^k; \mu^k) = \frac{1}{2} ||w^k||^2, \tag{7.8}$$

is the kinetic energy of λ^k. Combining (7.6) and (7.7), we have

$$||\phi_A(x^k, \lambda^{k+1}; \mu^k) - \phi_A(x^k, \lambda^k; \mu^k)|| = || - \hat{T}(x^{k+1}) + \hat{T}(x^k)|| \leq \varepsilon.$$

By (7.8), it follows that for k large enough:

$$||w^{k+1} - w^k|| \leq \varepsilon.$$

Therefore, for large k, (7.1) is approximately satisfied as:

$$w^k \approx \nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k), \tag{7.9}$$

which implies that

$$w^* \approx \nabla_\lambda \phi_A(x^*, \lambda^*; \mu^*). \tag{7.10}$$

But we know that $||w^*|| = 0$. Therefore,

$$0 = ||w^*|| \approx \lim_{k\to\infty} ||w^k|| = \lim_{k\to\infty} ||\nabla_\lambda \phi_A(x^k \lambda^k; \mu^k) \Delta t_\lambda^k||. \tag{7.11}$$

Since Δt_λ^k is finite, based on the typical values in Table 6.3, (7.11) reduces to

$$\lim_{k\to\infty} ||\nabla_\lambda \phi_A(x^k, \lambda^k; \mu^k)|| \approx 0. \tag{7.12}$$

By (7.2), we can expand (7.12) as:

$$0 = \lim_{k\to\infty} \sum_i (-c_i(x^k))^2 = \lim_{k\to\infty} \sum_i (c_i(x^k))^2,\ i \in E \cup (I \cap A_s(x)), \tag{7.13}$$

and

$$0 = \lim_{k\to\infty} \sum_i (-\mu^k \lambda_i^k)^2 = \lim_{k\to\infty} \sum_i (\mu^k \lambda_i^k)^2,\ i \in I \setminus A_s(x). \tag{7.14}$$

Since each term in the summations (7.13) and (7.14) are non-negative, the respective summations can only be null if each of their terms are null:

$$\lim_{k\to\infty} c_i(x^k) = 0,\ i \in E \cup (I \cap A_s(x)), \tag{7.15}$$

and

$$\lim_{k\to\infty} \mu^k \lambda_i^k = 0,\ i \in I \setminus A_s(x). \tag{7.16}$$

By (7.15), the lemma is proved for the equality constrained case.
The inequality constrained case is now proved. By the continuity of $\lambda_i^k,\ i \in E \cup (I \cap A_s(x))$, by (7.15), we have

$$\lim_{k\to\infty} |\lambda_i^k c_i(x^k)| = 0,\ i \in E \cup (I \cap A_s(x)). \tag{7.17}$$

Furthemore, since the penalty parameter μ^k is finite, based on the typical values in Table 6.3, equation (7.16) reduces to

$$\lim_{k\to\infty} \lambda_i^k = 0,\ \ i \in I \setminus A_s(x). \tag{7.18}$$

Thus, by the continuity of $c_i(x^k),\ i \in I \setminus A_s(x)$, by (7.18), we have

$$\lim_{k\to\infty} |\lambda_i^k c_i(x^k)| = 0,\ i \in I \setminus A_s(x). \tag{7.19}$$

Combining (7.17) and (7.19), we conclude that

$$\lim_{k\to\infty} |\lambda_i^k c_i(x^k)| = 0,\ i \in I, \tag{7.20}$$

since $I = I \cap A_s(x) \cup I \setminus A_s(x)$. This completes the proof.

□

We now prove that **Algorithm 8** exhibits convergence of the primal variable x.

Theorem 7.1.2. *Assume that x^* is a limit point of the sequence $\{x^k\}$ generated by* **Algorithm 8**. *Then,*

$$\lim_{k\to\infty} ||x^k|| = x^*.$$

Proof. By equation (5.11), we have

$$x^{k+1} = x^k + v^k \Delta t_x^k. \tag{7.21}$$

It therefore follows that

$$\lim_{k\to\infty} ||x^{k+1} - x^k|| = \lim_{k\to\infty} ||v^k|| |\Delta t_x^k|. \tag{7.22}$$

We need to prove that the right hand side of (7.22) converges to $||v^*||$. To do so we define a function $Q(x)$, such that

$$
\begin{aligned}
Q(x) &= \sup_{\lambda} \phi_A(x^k, \lambda^k; \mu^k) \\
&= \sup_{\lambda} \left(f(x) - \begin{cases} \sum_i \lambda_i^k c_i(x) - \sum_i \frac{1}{2\mu^k} c_i^2(x), & i \in E \cup (I \cap A_s(x)); \\ \sum_i \frac{\mu^k}{2} (\lambda_i^k)^2, & i \in I \setminus A_s(x). \end{cases} \right).
\end{aligned}
$$

We now prove that this function yields an upper bound on $f(x)$. Firstly, by **Assumption 6**, this function is bounded below by $\phi_A(x^*, \lambda^*; \mu)$. Secondly, since $\lambda^* = 0,\ i \in I \setminus A_s(x)$, and $|c_i(x^k)|^2 \geq 0,\ i \in E \cup (I \cap A_s(x))$, we have

$$
\begin{aligned}
\sup_{\lambda} \phi_A(x, \lambda; \mu) &= f(x) - \begin{cases} \sum_i \lambda_i^* c_i(x) - \sum_i \frac{1}{2\mu^*} c_i^2(x), & i \in E \cup (I \cap A_s(x)); \\ \sum_i \frac{\mu^k}{2} (\lambda_i^k)^2, & i \in I \setminus A_s(x) \end{cases} \\
&\geq f(x) - \sum_{i \in E \cup I \cap A_s(x^k)} \lambda_i^* c_i(x).
\end{aligned}
$$

Therefore for k large enough, we have that x^k is feasible and

$$
Q(x) = \max_{\lambda} \phi_A(x, \lambda; \mu) = \phi_A(x, \lambda^*; \mu) \geq f(x),
$$

proving that ϕ_A yields an upper bound on $f(x)$.

Now, at every iteration of **Algorithm 8**, the augmented Lagrangian $\phi_A(x, \lambda; \mu)$, defined in (2.30), is minimized with respect to x. This follows from the nature of the differential equation (5.1). Consequently, at each k-th iteration we have that

$$
\phi_A(x^k, \lambda^k; \mu^k) \geq \phi_A(x^{k+1}, \lambda^k; \mu^k).
$$

Therefore for large k and $\varepsilon > 0$ arbitrarily small, we have

$$
||\phi_A(x^k, \lambda^k; \mu^k) - \phi_A(x^{k+1}, \lambda^k; \mu^k)|| \leq \varepsilon. \tag{7.23}
$$

Since (5.1) implies conservation of energy, we have the following relation

$$
\Delta\phi_A^k = \phi_A(x^{k+1}, \lambda^k; \mu^k) - \phi_A(x^k, \lambda^k; \mu^k) = -\bar{T}(x^{k+1}, \lambda^k; \mu^k) + \bar{T}(x^k, \lambda^k; \mu^k) = -\Delta\bar{T}^k.
$$

By (7.23), it follows that for k large enough:

$$
||v^{k+1} - v^k|| \leq \varepsilon,
$$

where here we take $\bar{T}(x^k, \lambda^k; \mu^k) = \frac{1}{2}||v^k||^2$. Consequently, we have

$$\lim_{k\to\infty} ||v^k|| \approx v^* = 0, \tag{7.24}$$

by **Assumption 1**. Invoking continuity and finiteness of Δt_x^k, (7.22) and (7.24) imply that

$$\lim_{k\to\infty} ||x^{k+1} - x^k|| \approx ||v^*|| = 0. \tag{7.25}$$

By **Assumption 1**, the velocity of any particle is zero when the particle itself is stationary, i.e. at x^*, $v^* = 0$. Therefore, by (7.22), (7.24) and (7.25), we have

$$\lim_{k\to\infty} ||x^k|| \approx ||x^*||.$$

This concludes the proof. □

The next theorem shows that **Algorithm 8** exhibits convergence of the multiplier estimates.

Theorem 7.1.3. *Assume that the sequence* $\{\lambda^k\}$ *is generated by* **Algorithm 8***, and that* λ^* *is a limit point of that sequence. Then*

$$\lim_{k\to\infty} ||\lambda^k|| = \lambda^*. \tag{7.26}$$

Proof. For equality constraints, by (5.12) we have that

$$\lambda^{k+1} = \lambda^k + w^k \Delta t_\lambda^k. \tag{7.27}$$

By the same argument used in **Lemma 7.1.1**, we have that

$$\lim_{k\to\infty} w^k \approx w^* = 0.$$

By (7.27) and the same argument used in **Theorem 7.1.2**, since Δt_λ^k is finite, the results follow:

$$\lim_{k\to\infty} ||\lambda^k|| \approx \lambda^*. \tag{7.28}$$

For inequality constraints, we have by (5.12), that

$$\lambda^{k+1} = \max\{\underline{0},\ \lambda^k + w^k \Delta t_\lambda^k\}, \tag{7.29}$$

where $\underline{0}$ is the zero vector of corresponding dimension. Equation (7.29) gives rise to the following two cases:

(i) $\lambda^k + w^k \Delta t_\lambda^k > 0$, and
(ii) $\lambda^k + w^k \Delta t_\lambda^k \leq 0$.
For Case (i), (7.29) reduces to

$$\lambda^{k+1} = \lambda^k + w^k \Delta t_\lambda^k,$$

which is identical to (7.27), for the equality constrained case. Result thus follow from the first result of this theorem.
For Case (ii), (7.29) reduces to

$$\lambda^k = \underline{0},$$

which implies that

$$\lim_{k\to\infty} ||\lambda^k|| \approx ||\lambda^*|| = 0,$$

and this completes the proof. □

To assert the results in **Theorems 7.1.2** and **7.1.3**, we consider the following theorem.

Theorem 7.1.4. Algorithm 8 *is well defined.*

Proof. The problems that define each x^k are smooth minimization problems with respect to ϕ_A in the compact set Ω_c. Therefore, their solutions converge. □

Lemma 7.1.5. *Assume that* $\{x^k\}$ *is generated by* **Algorithm 8** *and that* $\{x^k\}$ *is a subsequence that converges to* $x^* \in \mathbb{R}^n$.

Then,

$$\lim_{k\to\infty} ||\nabla f(x^k) - \sum_{i\in E\cup I} \lambda_i \nabla c_i(x^k)|| = 0.$$

Furthermore, if the point x^* *is feasible and satisfies MFCQ, then the sequence* $\{\lambda^k\}$ *is bounded, where* λ^k *is the vector of multiplier estimates corresponding to both equality and inequality constraints. In this case* x^* *satisfies the KKT conditions and, if there is one and only one vector* λ^* *of Lagrange multipliers associated with* x^**, we have:*

$$\lim_{k\to\infty} \lambda^k \approx \lambda^*. \tag{7.30}$$

Proof. For the first part of the proof we differentiate (2.30) with respect to x to obtain

$$\nabla_x \phi_A(x^k, \lambda^k; \mu^k) = \nabla f(x) + \Theta(x, \lambda; \mu), \tag{7.31}$$

where Θ is defined as

$$\Theta(x,\lambda;\mu) = \begin{cases} -\sum_i \lambda_i^k \nabla c_i(x^k) + \sum_i \frac{1}{\mu^k} c_i(x^k) \nabla c_i(x^k), & i \in E \cup (I \cap A_s(x)); \\ 0, & i \in I \setminus A_s(x). \end{cases}$$

By (7.15) we have that

$$\lim_{k\to\infty} c_i(x^k) = 0,\ i\ \in E \cup (I \cap A_s(x)),$$

which, by the continuity of ∇c and the finiteness of μ, implies:

$$\lim_{k\to\infty} \left|\left| \frac{1}{\mu^k} c_i(x^k)\nabla c_i(x^k) \right|\right| = 0,\ i\ \in E \cup (I \cap A_s(x)).$$

Therefore,

$$\lim_{k\to\infty} \sum_i \left(\frac{1}{\mu^k} c_i(x^k)\nabla c_i(x^k) \right)^2 = 0,\ i\ \in E \cup (I \cap A_s(x)). \tag{7.32}$$

Since every term in the summation (7.32) is non-negative, each term is satisfied as:

$$\lim_{k\to\infty} \left(\frac{1}{\mu^k} c_i(x^k)\nabla c_i(x^k) = 0 \right),\ i \in E \cup (I \cap A_s(x).$$

Therefore, for all active constraints $c_i(x),\ i \in E \cup (I \cap A_s(x))$, (7.31) reduces to

$$\lim_{k\to\infty} ||\nabla_x \phi_A(x^k, \lambda^k; \mu^k)|| = \lim_{k\to\infty} \left|\left| \nabla f(x^k) - \sum_{i\in E\cup(I\cap A_s(x))} \lambda_i^k \nabla c_i(x^k) \right|\right|. \tag{7.33}$$

By (5.14), we have

$$v^k - v^{k-1} = -\nabla_x \phi_A(x^k, \lambda^k; \mu^k)\Delta t_x^{k-1}, \tag{7.34}$$

and by **Theorem 7.1.2**, we know that

$$\lim_{k\to\infty} ||v^k|| \approx 0. \tag{7.35}$$

We therefore have that

$$\lim_{k\to\infty} ||v^k - v^{k-1}|| \approx 0,$$

and equation (7.34) reduces to:

$$\lim_{k\to\infty} || - \nabla_x \phi_A(x^k, \lambda^k; \mu^k)|| \approx 0. \tag{7.36}$$

It follows from (7.33) and (7.36), that

$$\lim_{k\to\infty} \left|\left| - \left(\nabla f(x^k) - \sum_{i\in E\cup(I\cap A_s(x))} \lambda_i^k \nabla c_i(x^k) \right) \right|\right| = \lim_{k\to\infty} || - \nabla_x \phi_A(x^k, \lambda^k; \mu^k)|| \approx 0.$$

This proves the result for all active constraints $c_i(x),\ i \in E \cup (I \cap A_s(x))$.

For inactive constraints, $c_i(x),\ i \in I \setminus A_s(x)$, we have that

$$0 \approx \lim_{k\to\infty} ||\nabla_x \phi_A(x^k, \lambda^k; \mu^k)|| = \lim_{k\to\infty} ||\nabla f(x^k)|| = \lim_{k\to\infty} ||\nabla f(x^k) - \sum_{i\in I\setminus A_s(x)} \lambda_i^k \nabla c_i(x^k)||,$$

since $\lambda_i^k = 0,\ \forall i \in I \setminus A_s(x)$, and this proves the first part of the lemma.

For the second part of the lemma, we use a proof by contradiction. Assume that the sequence $\{\lambda^k\}$ is unbounded. Therefore

$$\lim_{k\to\infty} M_k = \infty,$$

where

$$M_k = ||\lambda^k||_\infty.$$

The sequence $\{\frac{\lambda^k}{M_k}\}$ is however bounded. It therefore follows that some subsequence is convergent and that its limit is 1, since by the choice of M_k, infinitely many elements in this sequence have modulus equal to 0, except for $\frac{\max|\lambda_i^k|}{M_k}$ which has modulus equal to 1. The KKT Lagrange multiplier estimates $\lambda_i^*,\ i \in I$ corresponding to active constraints however, are not null. Therefore, taking limits for this subsequence and using the first part of the theorem, we obtain that MFCQ cannot hold since the limit of this convergent subsequence is not a KKT multiplier. This contradicts the initial assumption that MFCQ holds.

Now, if MFCQ holds, every limit of a convergent subsequence of $\{\lambda^k\}$ defines a set of KKT multipliers λ_i^* [130]. Therefore, (7.30) holds in the case that the multipliers are unique. □

We use **Lemma 7.1.5** to prove the main convergence result of this section.

Theorem 7.1.6. *Assume that the sequence $\{x^k\}$ is generated by* **Algorithm 8***, and that x^* is a limit point of that sequence. Also assume that strict complementarity holds at x^*. Then the following properties hold:*

(1) The constraints of problem (1.2) *are satisfied as:*

$$c_i(x) = 0,\ i \in E,$$

and

$$c_i(x) \geq 0,\ i \in I,$$

at x^*, where x^* is feasible.

(2) If x^* satisfies MFCQ with respect to all the constraints of (1.2)*, then x^* fulfills the KKT conditions of the problem.*

Proof. For the first part of the proof, by **Lemma 7.1.1** we have that

$$c_i(x^*) = 0,\ i \in E.$$

In addition, by the property of strict complementarity we have that

$$\lambda_i^* > 0,\ c_i(x^*) = 0,\ i \in I \cap A_s(x^*).$$

We also have that

$$c_i(x^*) > 0\ i \in I \setminus A_s(x^*).$$

Thus, $c_i(x^*) \geq 0,\ i \in I$, since $I = (I \cap A_s(x)) \cup I \setminus A_s(x)$. This completes the proof for the first part of the theorem.
For the second part of the theorem, following the results of **Lemma 7.1.5** and the first part of this theorem, by **Theorem 2.1.6** (with MFCQ replacing LICQ) all the requirements of a KKT point are fulfilled. This completes the proof. □

This concludes the global convergence discussion for TACNLP. We now establish local convergence properties of the trajectory-based method.

7.2 Local convergence analysis

In this section, we formalize the basis of our local rate of convergence and proceed with the local convergence analysis of TACNLP. We make use of the following lemma to establish our local convergence properties.

Lemma 7.2.1. *Assume that x^k is generated by* **Algorithm 8**. *Then, there exists $\hat{k} \in \mathbb{N}$ such that $\forall\ 0 \ll \hat{k} \leq k$:*

$$\frac{\lambda^k - \lambda^{k+1}}{(\Delta t_\lambda^{k-1} \Delta t_\lambda^k)} \geq \min\{c(x^k), \mu^k \lambda^k\}, \tag{7.37}$$

where $c(x)$ is the vector of equality and inequality constraints $c_i(x),\ i \in E \cup I$, and λ is the corresponding vector of Lagrange multipliers $\lambda_i,\ i \in E \cup I$.

Proof. Recall by (5.15) and (7.2) that:

$$w_i^k = w_i^{k-1} - \Delta t_\lambda^{k-1} \begin{cases} c_i(x^k), & i \in E \cup (I \cap A_s(x)); \\ \mu^k \lambda_i^k, & i \in I \setminus A_s(x), \end{cases} \tag{7.38}$$

which, by (2.27) and (2.30) can be written equivalently as:

$$w_i^k = w_i^{k-1} - \Delta t_\lambda^{k-1} \begin{cases} c_i(x^k), & c_i(x^k) < \mu^k \lambda_i^k; \\ \mu^k \lambda_i^k, & c_i(x^k) > \mu^k \lambda_i^k. \end{cases} \tag{7.39}$$

Now by the initialization of **Algorithm 8**, using (6.12) and (7.2), we have

$$w_i^0 = -\frac{\Delta t_\lambda^0}{2} \begin{cases} c_i(x^0), & c_i(x^0) < \mu^0 \lambda_i^0; \\ \mu^0 \lambda_i^0, & c_i(x^0) > \mu^0 \lambda_i^0. \end{cases} \tag{7.40}$$

Furthermore, by (7.39), we have

$$w_i^1 = w_i^0 - \Delta t_\lambda^0 \begin{cases} c_i(x^1), & c_i(x^1) < \mu^1 \lambda_i^1; \\ \mu^1 \lambda_i^1, & c_i(x^1) > \mu^1 \lambda_i^1, \end{cases} \tag{7.41}$$

and

$$\begin{aligned} w_i^2 &= w_i^1 - \Delta t_\lambda^1 \begin{cases} c_i(x^2), & c_i(x^2) < \mu^1 \lambda_i^2; \\ \mu^2 \lambda_i^2, & c_i(x^2) > \mu^2 \lambda_i^2, \end{cases} \\ &= w_i^0 - \Delta t_\lambda^0 \begin{cases} c_i(x^1), & c_i(x^1) < \mu^1 \lambda_i^1 \\ \mu^1 \lambda_i^1, & c_i(x^1) > \mu^1 \lambda_i^1 \end{cases} - \Delta t_\lambda^1 \begin{cases} c_i(x^2), & c_i(x^2) < \mu^1 \lambda_i^2; \\ \mu^2 \lambda_i^2, & c_i(x^2) > \mu^2 \lambda_i^2. \end{cases} \end{aligned} \tag{7.42}$$

In general, the i-th component of $w^k,\ k > 0$ can be written as the summation:

$$w_i^k = w_i^0 - \sum_{j=1}^{k} \Delta t_\lambda^{j-1} \min\{c_i(x^j), \mu^j \lambda_i^j\},\ i \in E \cup I. \tag{7.43}$$

Equation (7.43) can be simplified by considering two different cases, which can be written explicitly as follows:

Case (i) $c_i(x^k) < \mu^k \lambda_i^k$ (with x^k feasible)

Recall from Section 2.2, that this case implies $c_i(x^k) = 0$, i.e. $c_i(x^k)$ is (strictly) active.

Case (ii) $c_i(x^k) > \mu^k \lambda^k$

Recall from Section 2.2, that this case implies that $c_i(x^k)$ is inactive.

We have that **Case (i)** is satisfied by all equality constraints as well as all strictly active inequality constraints. We consider these separately.
Firstly, for i $\in I \cap A_s(x)$, and for k large enough, since $\mu^k > 0$, we have that

$$\min\{c_i(x^k), \mu^k \lambda_i^k\} = c_i(x^k). \tag{7.44}$$

Equation (7.43) therefore reduces to

$$w_i^k = -\frac{\Delta t^0}{2} c_i(x^0) - \sum_{j=1}^{k} \Delta t_\lambda^{j-1} c_i(x^j). \tag{7.45}$$

We now show that $\exists\, \hat{k}$ such that all terms in (7.43) with

$$0 \ll j \leq \hat{k},$$

are negligible. We do this because we assume that steady convergence of the iterations only takes place after $\hat{k}$ iterations.

When we are far away from the minimum, it is safe to assume that the particle λ will steer away from the optimal trajectory at least once. This means the decent condition (4.6) will be violated for more than 2 consecutive iterations. In this case the velocity w^j is restarted using (5.30) and all previous directions are ignored in successive calculations. Assuming that this restart take place at the $\hat{k}$-th, iteration, all previous iterations $j = 0, \ldots, \hat{k} - 1$ have no impact on the calculation of w_i^k, and (7.45) reduces to:

$$w_i^k = -\sum_{j=\hat{k}}^{k} \Delta t_\lambda^{j-1} c_i(x^j). \tag{7.46}$$

Now by (5.12), we have

$$\lambda_i^{k+1} = \max\{\lambda_i^k + w_i^k \Delta t_\lambda^k,\, 0\},\ i \in I, \tag{7.47}$$

and since $c_i(x^k)$ is strictly active, we have $\lambda_i^{k+1} > 0$. Equation (7.47) therefore reduces to:

$$\lambda_i^{k+1} = \lambda_i^k + w_i^k \Delta t_\lambda^k. \tag{7.48}$$

By (7.46), we can write (7.48) equivalently as

$$\lambda_i^{k+1} = \lambda_i^k - \Delta t_\lambda^k \sum_{j=\hat{k}}^{k} \Delta t_\lambda^{j-1} c_i(x^j). \tag{7.49}$$

We therefore have

$$\lambda_i^k - \lambda_i^{k+1} = \Delta t_\lambda^k \sum_{j=\hat{k}}^{k} \Delta t_\lambda^{j-1} c_i(x^j), \tag{7.50}$$

which by (7.44), becomes:

$$\lambda_i^k - \lambda_i^{k+1} = \Delta t_\lambda^k \sum_{j=\hat{k}}^{k} \Delta t_\lambda^{j-1} \min\{c_i(x^j), \mu^j \lambda_i^j\}. \tag{7.51}$$

Now since every term in the right hand summation of (7.51) is positive for all inequality constraints and since Δt_λ^k is finite and positive $\forall\, k$, we have

$$\sum_{j=\hat{k}}^{k} \Delta t_\lambda^{j-1} \min\{c_i(x^j), \mu^j \lambda_i^j\} \geq \Delta t_\lambda^{k-1} \min\{c_i(x^k), \mu^k \lambda_i^k\}. \tag{7.52}$$

Combining (7.51) and (7.52), we have

$$\frac{\lambda_i^k - \lambda_i^{k+1}}{(\Delta t_\lambda^{k-1} \Delta t_\lambda^k)} \geq \min\{c_i(x^k), \mu^k \lambda_i^k\}, \tag{7.53}$$

proving that (7.37) holds $\forall i \in I \cap A_s(x)$.

Secondly, for $i \in E$, where $c_i(x)$ is active, we have by strict complementarity that either $\lambda_i^k > 0$ or $\lambda_i^k < 0$. If $\lambda_i^k > 0$, then

$$\min\{c_i(x^k), \mu^k \lambda_i^k\} = c_i(x^k),$$

since $\mu^k > 0$. Here the same result proved above for active inequality constraints applies. We do not need to consider the case where $\lambda_i <^k 0$, as this does not contribute to the rest of the discussion in this chapter.

For **Case (ii)**, $i \in I \setminus A_s(x)$, and $c_i(x^k) > \mu^k \lambda_i^k$. Here $c_i(x)$ is inactive and consequently $\lambda_i = 0$. We therefore have

$$\min\{c_i(x^k), \mu^k \lambda_i^k\} = \mu^k \lambda_i^k.$$

Following the same line of proof used for **Case (i)**, we have

$$\frac{\lambda_i^{k+1} - \lambda_i^k}{(\Delta t_\lambda^{k-1} \Delta t_\lambda^k)} \geq = \min\{c_i(x^k), \mu^k \lambda_i^k\},$$

and this completes the proof, based on the assumption that all previous j iterations, where $j = 0, \ldots (\hat{k} - 1)$, do not impact on the calculations at the current iteration.

When this is not the case, by **Theorem 7.1.6**, the solutions are converging and approaching the feasible region. Therefore, we may assume that:

$$w_i^k = \sum_{j=\hat{k}} \Delta t_\lambda^{j-1} \min\{c_i(x^j), \mu^j \lambda^j\} \geq \Delta t_\lambda^{k-1} \min\{c_i(x), \mu^k \lambda^k\},\ i \in E \cup I,$$

and the results follow. This completes the proof. □

The following assumptions are central to our local convergence discussion.

Assumption 7. *Let a constraint qualification with respect to $c(x)$ be satisfied $\forall x \in \Omega_c$. Then by* **Assumption 3** *and* **Theorem 7.1.4**, *the sequence $\{x^k\}$, generated by* **Algorithm 8**, *is well defined.*

Assumption 8. *The sequence of iterates $\{x^k\}$ generated by* **Algorithm 8** *converges to a feasible point x^*.*

Assumption 9. *The functions f and c are twice continuously differentiable at x^* and the second order sufficient optimality condition is fulfilled at x^*. This means that the KKT conditions hold and there is an associated Lagrange multiplier λ^* such that:*

$$d^T \nabla_{xx}^2 L(x^*, \lambda^*; \mu^*) d > 0,$$

for a non-null vector d.

Assumption 10. *For all $k = 1, 2, 3, \ldots$, we define $\varepsilon_k \to 0$ in such a way that*

$$\varepsilon_k \leq \chi(\Psi_2(x^k, \lambda^k)),$$

where $\chi : \mathbb{R}^+ \to \mathbb{R}^+$ is such that $\lim_{t \to 0} \frac{\chi(t)}{t} = 0$, and Ψ_2 is defined as in **Algorithm 5** *using PC2.*

Assumption 11. *For all $k = 1, 2, 3, \ldots$, we use the penalty parameter updating scheme outlined in* **Algorithm 5** *,with PC2.*

Lemma 7.2.2. *Let* **Assumptions 5, 7** *and* **8** *hold. Then,*

$$\lim_{k\to\infty} \lambda^k = \lambda^*.$$

Proof. The results follow from **Lemma 7.1.5**. □

Lemma 7.2.3. *Let* **Assumptions 7 - 9** *hold. Then there exists* $k_0 \in \{1,2,\dots\}$, $\beta_1, \beta_2 > 0$ *such that, for all* $k \geq k_0$,

$$\beta_1 ||(x^k, \lambda^k) - (x^*, \lambda^*)|| \leq \Psi_2(x^k, \lambda^k) \leq \beta_2 ||(x^k, \lambda^k) - (x^*, \lambda^*)||$$

Proof. The proof follows from **Lemma 7.2.2** and **Assumption 9**, using the local error bound theory [46, 48, 64, 79]. □

Lemma 7.2.4. *Suppose that* **Assumptions 7 - 10** *hold. Then, there exists* $k_1 \in \{1,2,\dots\}$, $q_1, q_4 > 0$ *and a sequence* $\eta \to 0$ *such that, for all* $k \geq k_1$,

$$\left(1 - \frac{q_4}{(\Delta t_\lambda^{k-1} \Delta t_\lambda^k)}\right) \Psi_2(x^{k+1}, \lambda^{k+1}) \leq \left(q_1 \eta_k + \frac{q_4}{(\Delta t_\lambda^{k-1} \Delta t_\lambda^k)}\right) \Psi_2(x^k, \lambda^k), \tag{7.54}$$

where

$$\eta_k = \frac{\chi(\Psi_2(x^k, \lambda^k))}{\Psi_2(x^k, \lambda^k)}, \tag{7.55}$$

and Ψ_2 is defined as in **Algorithm 5** using PC2.

Proof. Throughout this proof, let $\underline{c}(x)$ denote the set of inequality constraints and let $\underline{\lambda}$ denote the corresponding multiplier estimates. Also let $\hat{c}(x)$, denote the set of equality constraints, and let $\lambda\mu = \sum_{i \in E \cup I} \lambda_i \mu$. By (5.12), (6.10), **Assumption 7** and **Lemma 7.1.1**, there exists $q > 0$ such that $\Psi_2(x^{k+1}, \lambda^{k+1}, \underline{\lambda}^{k+1}) \leq q\varepsilon^k$, $\forall\, k = 1,2,\dots$, where $\Psi_2(x^{k+1}, \lambda^{k+1}, \underline{\lambda}^{k+1}) = ||T_2(x^{k+1}, \lambda^{k+1}, \underline{\lambda}^{k+1})||$, and

$$T_2(x^{k+1}, \lambda^{k+1}, \underline{\lambda}^{k+1}) = \begin{pmatrix} \nabla_x \phi_A(x^{k+1}, \lambda^{k+1}, \underline{\lambda}^{k+1}) \\ \hat{c}(x^{k+1}) \\ \min\{\underline{c}(x^{k+1}), \underline{\lambda}^{k+1}\} \end{pmatrix}.$$

Therefore, there exist $q_1, q_2 > 0$ such that, for all $k = 1,2,\dots$,

$$\Psi_2(x^{k+1}, \lambda^{k+1}, \underline{\lambda}^{k+1}) \leq q_1 \varepsilon^k + q_2 \hat{c}(x^{k+1}) + ||\min\{\underline{c}(x^{k+1}), \underline{\lambda}^{k+1}\} \tag{7.56}$$

Since $0.1 \leq \mu^k \leq 1$ $\forall k$, we have that

$$\min\{\underline{c}(x^{k+1}), \mu^{k+1} \underline{\lambda}^{k+1}\} \leq \min\{\underline{c}(x^{k+1}), \underline{\lambda}^{k+1}\}.$$

Therefore (7.56) can be written equivalently as:

$$\Psi_2(x^{k+1},\lambda^{k+1},\underline{\lambda}^{k+1}) \leq q_1\varepsilon^k + q_2\hat{c}(x^{k+1}) + ||\min\{\underline{c}(x^{k+1}),\mu^{k+1}\underline{\lambda}^{k+1}\}||.$$

Thus by the **Lemma 7.2.1** and **Lemma 7.2.2** there exists $q_3 > 0$ such that, for $k \geq k_1$,

$$\Psi_2(x^{k+1},\lambda^{k+1},\underline{\lambda}^{k+1}) \leq q_1\varepsilon^k + \frac{q_3}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)}(||\lambda^{k+1}-\lambda^k|| + ||\underline{\lambda}^{k+1}-\underline{\lambda}^k||).$$

Thus, for all $k \geq k_1$,

$$\Psi_2(x^{k+1},\lambda^{k+1},\underline{\lambda}^{k+1}) \leq q_1\varepsilon^k + \frac{q_3}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)}(||\lambda^{k+1}-\lambda^*|| + ||\lambda^k-\lambda^*|| + ||\underline{\lambda}^{k+1}-\underline{\lambda}^*|| + ||\underline{\lambda}^k-\underline{\lambda}^*||).$$

Therefore, by **Lemma 7.2.3**, there exists $q_4 > 0$ such that, for all $k \geq k_1$,

$$\Psi_2(x^{k+1},\lambda^{k+1},\underline{\lambda}^{k+1}) \leq q_1\varepsilon^k + \frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)}[\Psi_2(x^{k+1},\lambda^{k+1},\underline{\lambda}^{k+1}) + \Psi_2(x^k,\lambda^k,\underline{\lambda}^k)].$$

By **Assumption 10**, for all $k \geq k_1$, we therefore have:

$$(1-\frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)})\Psi_2(x^{k+1},\lambda^{k+1},\underline{\lambda}^{k+1}) \leq q_1\chi(\Psi_2(x^k,\lambda^k,\underline{\lambda}^k)) + \frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)}\Psi_2(x^k,\lambda^k,\underline{\lambda}^k). \tag{7.57}$$

By **Lemma 7.2.2**, **Lemma 7.2.3** and **Assumption 10**, we have that

$$\lim_{k\to\infty}\eta^k = \lim_{k\to\infty}\frac{\chi(\Psi_2(x^k,\lambda^k))}{\Psi_2(x^k,\lambda^k)} = 0,$$

which implies that for k large enough:

$$\eta^k\Psi_2(x^k,\lambda^k) \approx \chi(\Psi_2(x^k,\lambda^k)).$$

Therefore, for $k \geq k_1$ and letting λ once again denote the multiplier estimates for all constraints, (7.57) becomes:

$$(1-\frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)})\Psi_2(x^{k+1},\lambda^{k+1}) \leq (q_1\eta_k + \frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)})\Psi_2(x^k,\lambda^k),$$

as we wanted to prove. □

Theorem 7.2.5. *By the boundedness of Δt_λ, we have that*

$$\Psi_2(x^{k+1},\lambda^{k+1}) \leq r\Psi_2(x^k,\lambda^k).$$

Proof. By **Lemma 7.2.4**, there exists k_1 such that (7.54) holds for $k \geq k_1$. Let $k_2 \geq k_1$ be such that

$$\left(1 - \frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)}\right) > \frac{1}{2},$$

for all $k \geq k_2$. Then, by (7.54), for $k \geq k_2$, we have that

$$\Psi_2(x^{k+1},\lambda^{k+1}) \leq 2\left(q_1\eta + \frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)}\right)\Psi_2(x^k,\lambda^k). \tag{7.58}$$

Let $k_3 \geq k_2$ be such that

$$2\left(q_1\eta + \frac{q_4}{(\Delta t_\lambda^{k-1}\Delta t_\lambda^k)}\right) \leq r,$$

for all $k \geq k_3$. Then, for all $k \geq k_3$,

$$\Psi_2(x^{k+1},\lambda^{k+1}) \leq r\Psi_2(x^k,\lambda^k),$$

and the theorem is proved. □

Corollary 7.2.6. *Under the assumption of* **Theorems 7.1.4, 7.1.6** *and* **7.2.5***, the sequence* $\{x^k,\lambda^k\}$ *converges to* (x^*,λ^*) *with R-linear convergence rate equal to r.*

Proof. By **Theorem 7.2.5** for all k large enough, we have that

$$\Psi_2(x^{k+1},\lambda^{k+1}) \leq r\Psi_2(x^k,\lambda^k).$$

Therefore the result follows from **Lemma 7.2.3**. □

Chapter 8

Trajectory-Based Method for MINLPs

In this chapter, we present a trajectory-based algorithm for MINLPs. We denote this algorithm by TAMINLP. TAMINLP is an adaptation of the trajectory-based algorithm TACNLP for CNLPs. TAMINLP is threefold, and within the framework of TAMINLP we use the same AL used in TACNLP, i.e., (2.30). Here, AL will be based on the subproblems discussed in Section 3.2 i.e., $\underline{\mathscr{M}}$ defined by (3.5) and $\bar{\mathscr{M}}$ defined by (3.6). In Section 8.1, we present some definitions which are pertinent to the rest of the discussion in this chapter. Section 8.2 outlines the details of TAMINLP. The first, second and third phases of the threefold TAMINLP algorithm are presented in Sections 8.3, 8.4 and 8.5 respectively. We summarize the main features of each phase in Section 8.6 and present the pseudo-code outlining these features. Finally, we present the convergence analysis of TAMINLP in Section 8.7.

8.1 Notation and definitions pertaining to the local minimum of MINLPs

Before we discuss the details of TAMINLP, we present some important notations and definitions related to a local solution of MINLPs. Recall the definition of an MINLP:

$$\mathscr{M}\begin{cases} \min\limits_{z} & f(z), \\ \text{s.t.} & c_i(z) = 0, i \in E, \\ & c_i(z) \geq 0, i \in I, \\ & z \in X \times Y. \end{cases} \tag{8.1}$$

Recall also the definition of Ω_m from Chapter 1. If there are no continuous variables present in $\mathscr{M}$ i.e., $n_c = 0$, then the feasible region Ω_m reduces to the discrete part only, and this we denote by Ω_d. Similarly if no discrete variables are present in (8.1) i.e., $n_d = 0$, then Ω_m reduces to the continuous feasible region only. We denote this by Ω_c.

If we fix the integer variables in $\mathscr{M}$ to some feasible values $y = y_d$ in the problem $\mathscr{M}$, then the remaining freedom in the continuous variable x, describes the continuous manifold:

$$f^M(x) = \{f(x, y_d) : (x,\ y_d) \in \Omega_m\}. \tag{8.2}$$

We refer to the feasible region of this manifold as a feasible continuous manifold. More specifically, the constrained region over which $f^M(x)$ is defined is known as the feasible continuous manifold. For illustrative purposes, we consider the problem

$$\begin{cases} \min & f(x,y) = (x-y)^2 - y, \\ \text{s.t.} & -2 \leq x \leq 2, \\ & y \in \{-2,-1,0,1,2\}, \end{cases} \tag{8.3}$$

where x is continuous and y is discrete. Figure 8.1 illustrates $f(x,y) = (x-y)^2 - y$, subject to the simple bound constraints in (8.3), as well as the continuous manifolds of f obtained at the fixed integer-feasible points $y = \{-2,-1,0,1,2\}$.

A manifold minimizer, obtained on each continuous manifold of f, is defined as follows:

Definition 8.1.1. *(Manifold minimizer) A point $x^* \in \Omega_c$ is a manifold minimizer if it is the continuous local minimizer of f on the respective feasible continuous manifold of problem* (8.1)*, such that*

$$f^M(x^*, y_d) \leq f^M(x, y_d), \forall x \in \Omega_c \cap B_\varepsilon(x^*),$$

where $B_\varepsilon(x^*)$ denotes the open ball $\{x \in \mathbb{R}^{n_c} : ||x - x^*|| < \varepsilon\}$, for some $\varepsilon > 0$.

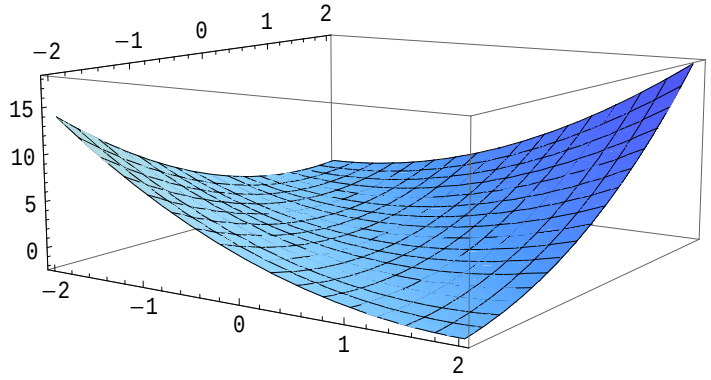
The manifold minima obtained on the continuous manifolds of $f(x,y)$ are depicted in Figure 8.2.

Suppose now that we were to fix the continuous variables in $\mathscr{M}$, to some point x_c. We denote $\mathscr{N}_r(\cdot,\cdot)$, as the neighborhood of some integer point, where the continuous variables are fixed. For example $\mathscr{N}_r(x_c, y_d)$ is the integer neighborhood of y_d when x is fixed at x_c:

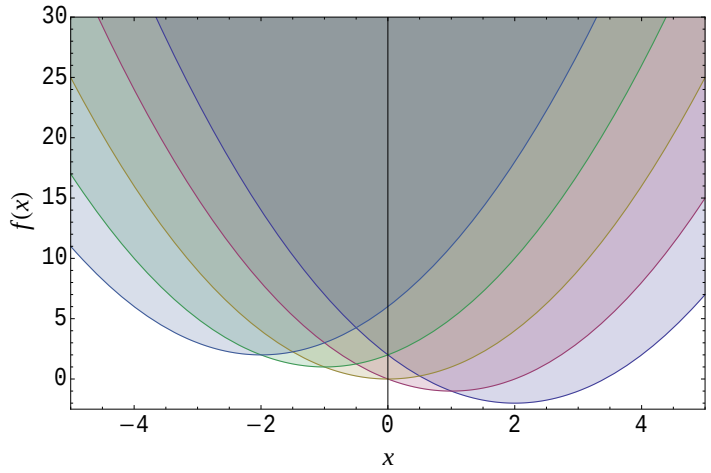
$$\mathscr{N}_r(x_c, y_d) = \{(x,\ y) \in \mathbb{R}^n : x = x_c, y \in \mathscr{N}_d(y_d, \varepsilon_d)\},$$

where

$$\mathscr{N}_d(y_d, \varepsilon_d) = \{y \in \Omega_d : ||y - y_d|| \leq \varepsilon_d\}, \tag{8.4}$$



(a) A 3D plot of $f(x,y) = (x-y)^2 - y$.



(b) A 2D plot of the parabolic manifolds obtained by fixing y in $f(x,y)$ to the integer-feasible values in (8.3). The plot is viewed along the x-axis

Figure 8.1 The plot of $f(x,y) = (x-y)^2 - y$, where the continuous manifolds are obtained by fixing $y = \{-2,-1,0,1,2\}$ in $f(x,y)$.

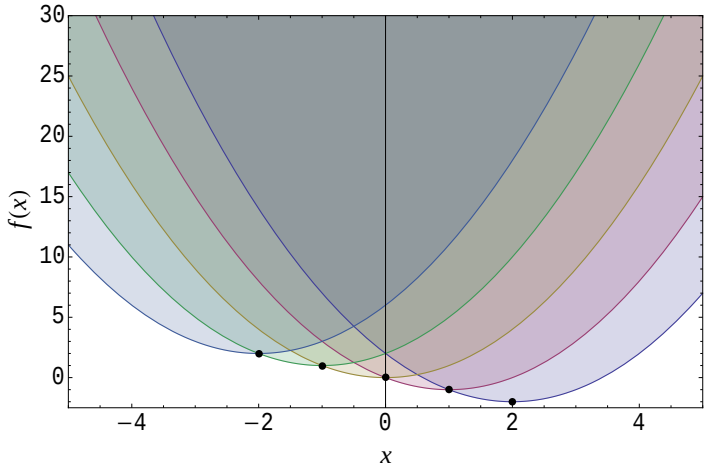


Figure 8.2 The manifold minima of f on the continuous parabolic manifolds, obtained by fixing y in $f(x,y)$ to the integer-feasible values in (8.3). The manifold minima are represented by the black dots on each manifold.

denotes a user defined neighborhood of discrete integer values which fall within the radius of ε_d [90], [91].

The remainder of this discussion would not make much sense without considering the definitions of continuous and discrete local minima separately. We therefore consider the definition of a continuous local minimizer and a discrete local minimizer. We also present three existing definitions for a mixed integer local minimizer. These define a separate local minimizer, an extended local minimizer and a combined local minimizer respectively.

The following definition of a continuous minimizer is considered [92].

Definition 8.1.2. *(Continuous local minimizer) A point $x^* \in \Omega_c$ is a local minimizer if, for some $\varepsilon > 0$,*

$$f(x^*) \leq f(x),\ \forall x \in\ \Omega_c \cap B_\varepsilon(x^*).$$

For an INLP, where no continuous variables are present, we consider the following definition of a discrete local minimizer [129]

Definition 8.1.3. *(Discrete local minimizer) A point $y^* \in \Omega_d$ is a local minimizer if,*

$$f(y^*) \leq f(y),\ \forall\ y \in \Omega_d \cap \mathcal{N}_d(y^*, \varepsilon_d).$$

Definitions for a local minimizer of an MINLP exist in the literature. We consider two such definitions. We then consider the definition of a combined local minimum which was

recently proposed by Newby [90, 91], to overcome some of the short-comings of these existing definitions.

For general MINLPs, the following definition of a local minimizer of $\mathscr{M}$ is used [83, 123]

Definition 8.1.4. *(Separate local minimizer) A point $z^* = (x^*,\ y^*)^T \in \Omega_m$ is a local minimizer of* (8.1) *if, for some $\varepsilon > 0$,*

$$\begin{array}{ll} f(x^*,y^*) \leq f(x,y), & \forall\ (x,y) \in \{(x\ y) : x \in B_\varepsilon(x^*),\ y = y^*\} \cap \Omega_m, \\ f(x^*,y^*) \leq f(x,y), & \forall (x,y) \in \mathscr{N}_r(x^*,y^*) \cap \Omega_m. \end{array} \tag{8.5}$$

The stronger definition of a local minimizer of (8.1) is also considered [1, 12, 84]. We refer to this as an extended local minimizer.

Definition 8.1.5. *(Extended local minimizer) A point $z^* = (x^*,\ y^*)^T \in \Omega_m$ is a local minimizer of* (8.1) *if, for some $\varepsilon > 0$,*

$$f(x^*,y^*) \leq f(x,y), \forall\ (x,y) \in \left(\bigcup_{(x,\ y) \in \mathscr{N}_r(x^*,y^*)} B_\varepsilon(x) \times \{y\} \right) \cap \Omega_m. \tag{8.6}$$

We now consider the definition of a combined local minimizer, recently proposed by Newby [90, 91]

Definition 8.1.6. *(Combined local minimizer) A point $z^* = (x^*,\ y^*)^T \in \Omega_m$ is a local minimizer of* (8.1) *if, for some $\varepsilon > 0$,*

$$\begin{array}{ll} f(x^*,y^*) \leq f(x,y), & \forall\ (x,y) \in \{x \in B_\varepsilon(x^*),\ y = y^*\} \cap \Omega_m, \\ f(x^*,y^*) \leq f(x,y), & \forall (x,y) \in \mathscr{N}_{comb}(x^*,y^*) \cap \Omega_m. \end{array} \tag{8.7}$$

where $\mathscr{N}_{comb}(x^*,y^*)$ will now be defined.

Define $\mathscr{A}(\tilde{x},\tilde{y})$ as

$$\mathscr{A}(\tilde{x},\tilde{y}) = \{(\bar{x},\bar{y}) : \bar{y} = \tilde{y},\ f(\bar{x},\bar{y}) \leq f(x,y)\ \forall (x,y) \in \{(x,y) : x \in B_\varepsilon(\bar{x}), y = \tilde{y}\} \cap \Omega_m\}.$$

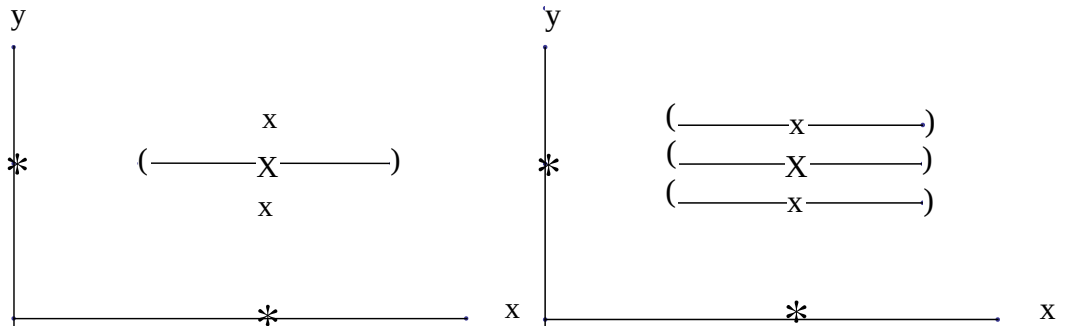
$\mathscr{A}$ is therefore the set of manifold minima obtained at each continuous manifold of f, where each manifold may contain more than one local minimum.

Here $\mathscr{N}_{comb}(x^*, y^*)$ is given by

$$\mathscr{N}_{comb}(x^*, y^*) = \{\underset{z}{\text{argmin}}[f(z)\ s.t.\ z \in A(\tilde{x}, \tilde{y})] : (\tilde{x}, \tilde{y}) \in \mathscr{N}_r(x^*, y^*) \setminus \{(x^*, y^*)\}\}.$$

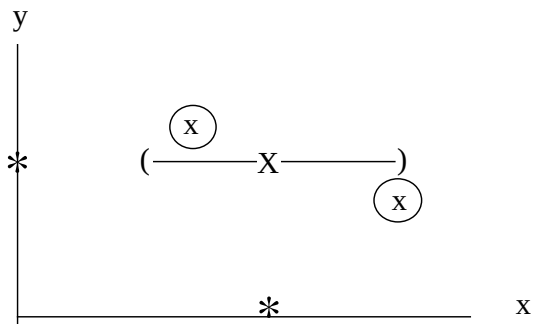
We notice that $N_{comb}(x^*, y^*)$ contains only the set of the smallest local minima on each feasible continuous manifold on which $\mathscr{N}_r(x^*, y^*)$ has a point, excluding $z^* = (x^*, y^*)^T$.

We now demonstrate with figures, an MINLP local minimizer as per each of the above definitions. We use one continuous and one integer variable for the illustrations.



(a) Separate local minimizer, as per **Definition 8.1.4**.

(b) Extended local minimizer, as per **Definition 8.1.5**.



(c) Combined local minimizer, as per **Definition 8.1.6**.

Figure 8.3 Illustration of different local minima corresponding to Definitions **8.1.4, 8.1.5** and **8.1.6** respectively.

In Figure 8.3, "X" denotes the location of the local minimizer in Ω_m with coordinate $(x^*,\ y^*)^T$. Each "x" denotes a point in $\mathscr{N}_r(x^*, y^*)$. The solid lines represent the open balls $B_\varepsilon(x)$ for fixed $y \in \mathscr{N}_r(x, y)$. The points "x" circumscribed by circles, seen in Figure 8.3c, denote the best manifold minimizers. Corresponding to the mixed local minimizer in each figure, both locations of optimal x^* and y^* are marked with ✻.

Figure 8.3a corresponds to the definition of a separate local minimizer, as per **Definition 8.1.4**. Here the point $X = (x^*, y^*)^T$, is identified as the mixed local minimizer since it is the

best minimizer in the open ball $B_\varepsilon(x^*)$ within the neighborhood $\mathcal{N}_r(x^*, y^*)$. In this definition, only $X = (x^*, y^*)^T$, found along the feasible continuous manifold corresponding to y^*, is detected.

Figure 8.3b is an illustration of a local minimizer, as per **Definition 8.1.5**. Here the point $X = (x^*, y^*)^T$ is identified as the mixed local minimizer since it is the best manifold minimizer in the open ball $B_\varepsilon(x^*)$ within the neighborhood $\mathcal{N}_r(x^*, y^*)$. Furthermore, each of the points "x" may or may not be the best manifold minimizer on their respective feasible continuous manifolds, see the illustration in Figure 8.3b. This definition is therefore not ideal.

Figure 8.3c corresponds to the definition of a local minimizer, as per **Definition 8.1.6**. Here the point $X = (x^*, y^*)^T$ is identified as the mixed local minimizer since it is the overall best of the three best manifold minima. This definition makes sure that the points "x" circumscribed by circles are the actual best manifold minima and therefore overcomes the shortcomings of **Definition 8.1.5**.

The definition of a combined local minimizer, i.e., **Definition 8.1.6** was designed to satisfy a list of restrictions proposed by Newby [90, 91]. These restrictions are not satisfied by **Definition 8.1.4** and **Definition 8.1.5**. We refer the reader to Newby [90, 91] for the list of restrictions.

8.2 Overview of TAMINLP

The development and implementation of TACNLP has led to a fairly straightforward adaptation thereof for MINLPs. In this section, we outline the main steps of TAMINLP and briefly present the differences between TACNLP and TAMINLP.

TAMINLP is threefold; it consists of a first, second and third phase. We now provide a brief overview of each phase. This is then followed by a detailed description of the first, second and third phase of TAMINLP, in Sections 8.3, 8.4 and 8.5 respectively.

- In the first phase of TAMINLP, TACNLP solves the continuous relaxation of $\mathscr{M}$ i.e., $\underline{\mathscr{M}}$, to obtain the solution $z_c^* = (x_c^*,\ y_c^*)^T$. If y_c^* is integer, then TAMINLP terminates as the optimal solution has been located. When this is not the case, the solution y_c^* is rounded and fixed to the nearest integer-feasible value $\bar{y}$, which is used in the second phase of TAMINLP. In this phase, the algorithm is initialized at $z(0) = (x(0), y(0))^T = (x^0, y^0)^T$, $\lambda(0) = \lambda^0$ and $\mu(0) = \mu^0$ to initialize the system of differential equations which is solved. The point $z(0) = (x^0,\ y^0)^T$ and λ^0 are user defined and $\mu^0 = \mu_{max}$, see Table 6.3.

- In the second phase of TAMINLP, all the discrete variables $\bar{y}^i$ which lie within some discrete user prescribed neighborhood of $\bar{y}$ are selected. TACNLP is then applied to $\bar{\mathscr{M}}(\bar{y}^i)$ defined by (3.6), where $\bar{\mathscr{M}}(\bar{y}^i)$ is solved for different fixed values of $\bar{y}^i$. The best of these solutions, $z_f^* = (x_f^*, y_f^*)^T$, is stored, where $y_f^* = \bar{y}^i$ for some i. For each $\bar{y}^i$ the second phase of the algorithm is initialized at x_c^*, with $\bar{y}^i$ fixed throughout the minimization routine, e.g $z^0 = (x_c^*, \bar{y}^i)^T$. In this phase TAMINLP uses the initialization $z(0) = (x(0), y(0))^T = (x_c^*, \bar{y}^i)^T$, $\lambda(0) = \lambda^0$ and $\mu(0) = \mu^0$ to initialize the system of differential equations is solved. Here λ^0 is user defined and $\mu^0 = \mu_{max}$ as per Table 6.3.

- The third phase of the minimization consists of performing a final continuous minimization at certain points found during the second phase of the minimization, including $(x_f^*, y_f^*)^T$. These points are typically the ones whose objective function value lie within some user defined neighborhood of the current best known objective function value $f(z_f^*) = f(x_f^*, y_f^*)^T$. Every minimization is initialized at each of these points, by applying TACNLP to $\underline{\mathscr{M}}$. We define $z(0)$ accordingly when initializing the system of differential equations solved in this phase, and $\mu^0 = \mu_{max}$ is defined homogeneously with phases one and two of TAMINLP. Here λ^0 is user defined. The best solution obtained in this phase, i.e., $z^* = (x^*, y^*)^T$ is identified as the overall optimal solution. If y^* is not integer, then an integer solution is strategically found. The details of this are presented later in Section 8.5.

Remark 8.2.1. During the second and third phase of TAMINLP, a few function evaluations are required to determine the best solution. These are however minimal, and never exceed the number of points investigated in the second and third phase.

AL for MINLPs is defined as follows:

$$\phi_A(z, \lambda; \mu) = f(z) - \sum_{i \in E \cup (I \cap A_s(z))} \lambda_i c_i(z) + \frac{1}{2\mu} \sum_{i \in E \cup (I \cap A_s(z))} c_i^2(z) + \psi(z, \lambda; \mu) \tag{8.8}$$

where

$$\psi(z, \lambda; \mu) = - \sum_{i \in I \setminus A_s(z)} \frac{\mu}{2} {\lambda_i}^2.$$

The second and third terms contain index sets corresponding to equality and strongly active inequality constraints and the last term corresponds to inactive inequality constraints, where $z = (x, y)^T$. As with the continuous case, when implementing AL, we include all types of constraints in (8.8). For the theoretical discussion however, we exclude weakly active constraints in (8.8).

Notice that when the integer constituent of $\mathscr{M}$ is relaxed, as in $\underline{\mathscr{M}}$, AL for MINLPs is the same as AL for CNLPs.

The fundamental difference between TACNLP and TAMINLP is that TAMINLP implements TACNLP in a series of minimizations. These minimizations are done in the three separate phases of TAMINLP. Table 8.1 summarizes the differences between TACNLP and TAMINLP.

Algorithm	Description
TACNLP	Implements the continuous minimization described in **Algorithm 8**, to obtain the continuous optimal solution, x^*, of (1.2).
TAMINLP	Implements the continuous minimization described in **Algorithm 8** during all three phases of TAMINLP, to obtain the mixed integer optimal solution, $z^* = (x^*, y^*)^T$, of (8.1): The first phase implements **Algorithm 8** by solving the relaxed problem $\underline{\mathscr{M}}$ to obtain the solution $z_c^* = (x_c^*, y_c^*)^T$. The second phase implements **Algorithm 8** by solving the problem $\bar{\mathscr{M}}(\bar{y}^i)$ for different fixed integer points, $\bar{y}^i$, to obtain the solution $z_f^* = (x_f^*, y_f^*)^T$. The third and final phase of TAMINLP, implements **Algorithm 8**, by solving $\underline{\mathscr{M}}$ for each point $(x_i^*, \bar{y}^i)^T$ whose objective function value lies within some user defined neighborhood of $f(x_f^*, y_f^*)^T$. Here x_i^* is the manifold minimizer corresponding to $\bar{y}^i$. The best feasible solution obtained here is denoted by $z^* = (x^*, y^*)^T$.

Table 8.1 The fundamental difference between TACNLP and TAMINLP

We now present an in-depth discussion of the the three-fold TAMINLP procedure.

8.3 The first phase of TAMINLP

In this section, we present the details of the first phase of TAMINLP. In this phase, TACNLP is applied to $\underline{\mathscr{M}}$ by solving the following set of equations:

$$\ddot{z} = -\nabla_z \phi_A(z, \lambda; \mu), \quad z(0) = z^0, \quad \dot{z}(0) = 0$$

$$\ddot{\lambda} = \nabla_{\lambda}\phi_A(z,\lambda;\mu), \quad \lambda(0) = \lambda^0, \quad \dot{\lambda}(0) = 0,$$

$$\ddot{\mu} = -\mu, \quad \mu(0) = \mu^0, \quad \dot{\mu}(0) = 0, \tag{8.9}$$

to obtain z_c^*, where $z = (x,\, y)^T$, $z_c^* = (x_c^*,\, y_c^*)^T$ and $y \in$ conv$Y$. This minimization is initialized at some user defined values $x(0) = x^0$, $y(0) = y^0$ and $\lambda(0) = \lambda^0$. The penalty parameter is initialized as $\mu(0) = \mu^0$ in accordance with the stipulations given in Table 6.3. The first phase of the algorithm terminates when:

$$||\nabla_x \phi_A(z,\lambda;\mu)|| \leq \varepsilon,$$

and

$$||\nabla_{\lambda}\phi_A(z,\lambda;\mu)|| \leq \varepsilon.$$

Once the solution $z_c^* = (x_c^*,\, y_c^*)^T$ has been obtained we proceed in one of two ways. If y_c^* is integer then the algorithm terminates as the optimal solution has been found. If y_c^* is not integer then we fix y_c^* to the nearest integer value $\bar{y}$. If any bound constraints are present or if any of the discrete variables are binary, and $\bar{y}$ is integer-infeasible with respect to these restrictions, then we apply the following:

$$\bar{y}_j = \begin{cases} u_j, & \text{if } \bar{y}_j > u_j; \\ l_j, & \text{if } \bar{y}_j < l_j, \end{cases} \tag{8.10}$$

where $\bar{y}_j$ corresponds to the jth component of the vector $\bar{y}$ and l_j and u_j are lower and upper bounds on the integer variables respectively. Naturally if y is binary then $l_j = 0,\ u_j = 1, \forall j$. Once the integer-feasible point $\bar{y}$ has been identified, we proceed with the second phase of the minimization in TAMINLP.

8.4 The second phase of TAMINLP

In this section, we present the details of second phase of TAMINLP. For this phase of the minimization, we generate several integer trial points. We use only the integer constituent y, of $z = (x,y)^T$, to generate these trial points. We then perform separate minimizations

with respect to x for each of these trial point. The discussion on how these trial points are generated is now presented.

8.4.1 Generating integer trial points

The trial points used for the second phase of TAMINLP are generated using the rule of the Pattern Search method (PS) [54]. Recall that n_d is the number of discrete variables present in $\mathscr{M}$. Let n_s denote the number of trial points generated, where $n_s = 2n_d$. Given $\bar{y}$, trial points are generated along the n_s coordinate axes as is done in the POLL step of PS [54]. The search pertains only to y, not x, since we are generating integer trial points. The POLL step of PS uses $n_s = 2n_d$ directions given by the set $D = \{d_i\}_{i=1}^{n_s}$:

$$D = \{d_1, d_2, \ldots, d_{2n_d}\} = \{e_1, \ldots, e_{n_d}, -e_1, \ldots, -e_{n_d}\}, \tag{8.11}$$

where e_i is the i-th unit coordinate vector in $\mathbb{R}^{n_d}$. The set D is said to positively span $\mathbb{R}^{n_d}$. For example, the set of positive spanning directions in $\mathbb{R}^2$, is made up of the four column vectors of D:

$$D = \begin{Bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{Bmatrix}. \tag{8.12}$$

In $\mathbb{R}^3$, the set D is defined as follows

$$D = \begin{Bmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \end{Bmatrix}. \tag{8.13}$$

D is defined similarly in $\mathbb{R}^{n_d}$.

We now illustrate, with the use of an example for the case of binary integer variables, the generation of integer (trial) points in $\mathbb{R}^2$, using $\bar{y} = (0,0)^T$. Four trial points are generated by traversing from $\bar{y}$ along each direction in (8.12) [54]. These trial points are $(1,0)^T$, $(0,1)^T$, $(-1,0)^T$ and $(0,-1)^T$ and are each located one unit from $\bar{y}$, see Figure 8.4.

We denote the trial points illustrated in Figure 8.4 as $\bar{y}^i$:

$$\bar{y}^i = \bar{y}^0 + d_i,\ i = 1, \ldots 2n_d, \tag{8.14}$$

where $\bar{y}^0 = \bar{y}$. Often certain restrictions are placed on the integer variables $\bar{y}^i$, calculated using equation (8.14). These include upper and lower bounds or binary restrictions. Unlike the strategy used in (8.10), if any of $\bar{y}^i, i = 1, \ldots, 2n_d$ are integer-infeasible, they are discarded.

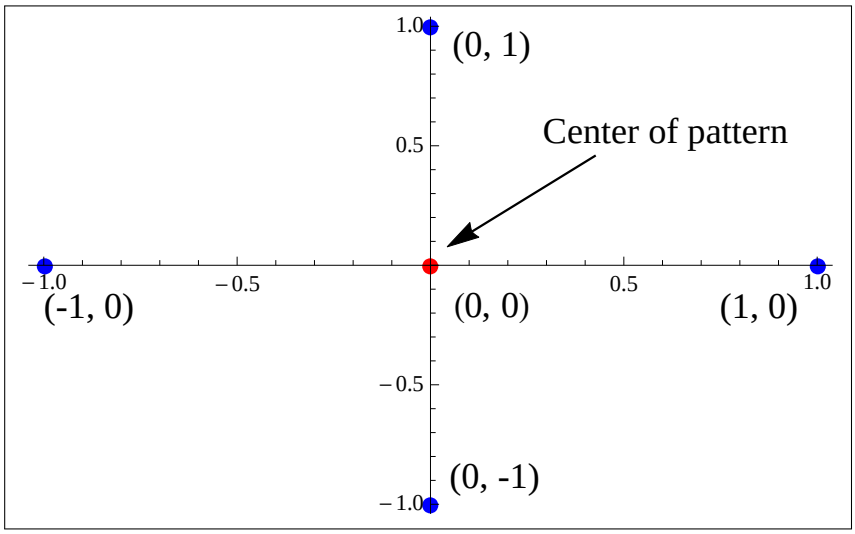


Figure 8.4 Trial points, $(1,0)^T$, $(0,1)^T$, $(-1,0)^T$, $(0,-1)^T$ generated about the pattern center $\bar{y} = (0,0)^T$, using the rule of PS.

For instance, when the integer solutions are restricted to binary variables, then moving in a direction which results in $\bar{y}^i$ taking on a value other than 0 or 1, would render the solution integer-infeasible. For the example above in $\mathbb{R}^2$, if the integer variables are restricted to taking on binary values, then two of the trial points in Figure 8.4 are integer infeasible. These are identified in Figure 8.5.

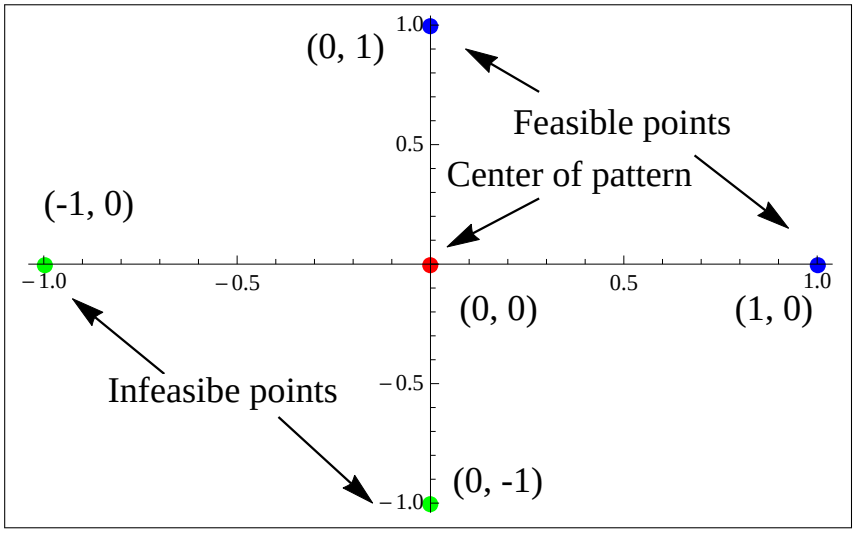


Figure 8.5 Feasible and infeasible trial points, $(1,0)^T$, $(0,1)^T$, $(-1,0)^T$, $(0,-1)^T$, generated using the rule of PS. The blue points are feasible, while the green points are infeasible.

In Figure 8.5, we see that the points $(-1,0)^T$ and $(0,-1)^T$ are integer-infeasible, while $(1,0)^T$ and $(0,1)^T$ are feasible. Traversing along the directions d_1 and d_2 results in

integer-feasible trial points, we therefore refer to these as feasible directions in D.

Our motivation for using this strategy to generate trial points is as follows. Since $\bar{y}$ is the closest integer-feasible point to y_c^*, the integer optimal solution to the problem $\mathscr{M}$ is either $\bar{y}$ or close to $\bar{y}$. The trial points generated in this phase therefore clearly fulfill this premise. Having generated these trial points, we now present the details of the second phase of TAMINLP.

8.4.2 Details of the minimization in the second phase of TAMINLP

The details in this section are based on the assumption that at least a subset of the trial points $\bar{y}^i$ are integer-feasible. Let us assume that $\hat{P}$ of these trial points are integer-feasible. The problem $\mathscr{M}$ is now projected onto the continuous space via the AL function $\phi_A(x,y,\lambda;\mu)$, so that the integer constituent of the problem is constant throughout the minimization. Separate minimizations are therefore performed on $\hat{P}$ continuous manifolds. Feasible continuous manifolds are now defined in terms of the objective and constraint functions $f(x)$ and $c_i(x)$, and adapted to equation (8.2) using $\phi_A(x,y,\lambda;\mu)$. Projecting the augmented Lagrangian $\phi_A(x,y,\lambda;\mu)$ onto the space of the continuous variables x by fixing feasible $y=\bar{y}^i$, gives rise to various continuous manifolds of the form given in equation (8.2):

$$\phi_A^i(x,\bar{y}^i,\lambda;\mu)),\ i=0,\ldots,\hat{P}, \tag{8.15}$$

where $\bar{y}^i=(\bar{y}_1^i,\bar{y}_2^i,\ \bar{y}_3^i,\ \ldots,\ \bar{y}_{n_d}^i)^T$. Having obtained a set of integer-feasible points, which give rise to feasible continuous manifolds, we can begin with the second phase of TAMINLP.

In this phase, we apply TACNLP to $\bar{\mathscr{M}}(\bar{y}^i),\ i=1,\ldots,\ \hat{P}$, where $\bar{\mathscr{M}}(\bar{y}^i)$ is defined as follows

$$\bar{\mathscr{M}}(\bar{y}^i)\begin{cases}\min\limits_{x} & f(x,\bar{y}^i)\\ \text{s.t.} & c_j(x,\bar{y}^i)=0, j\in E,\\ & c_j(x,\bar{y}^i)\geq 0, j\in I,\\ & x\in X.\end{cases} \tag{8.16}$$

The above problem will be solved $\hat{P}$ times, since there are $\hat{P}$ integer-feasible points $\bar{y}^i$. Each $\bar{y}^i$ needs to be explored since any of these may correspond to the optimal solution of the problem.

For each problem $\bar{\mathscr{M}}(\bar{y}^i)$, the second phase of the algorithm terminates when:

$$||\nabla_x\phi_A^i(x,\bar{y}^i,\lambda;\mu)||\leq\varepsilon,$$

and

$$||\nabla_\lambda \phi_A^i(x, \bar{y}^i, \lambda; \mu)|| \leq \varepsilon,$$

Here ϕ_A is projected onto the x space, so it is identical to (2.30), and ε is defined as in the continuous case presented in Chapter 5. Every other aspect of this minimization is identical to that of the continuous case.

Particularly, for each $\bar{y}^i$ we solve the following system, for $i = 1, \ldots, \hat{P}$:

$$\ddot{x} = -\nabla_x \phi_A^i(x, \bar{y}^i, \lambda; \mu), \quad x(0) = x_c^*, \quad \dot{x}(0) = 0, \tag{8.17}$$

$$\ddot{\lambda} = \nabla_\lambda \phi_A^i(x, \bar{y}^{(i)}, \lambda; \mu), \quad \lambda(0) = \lambda^0, \quad \dot{\lambda}(0) = 0,$$

$$\ddot{\mu} = -\mu, \quad \mu(0) = \mu^0, \quad \dot{\mu}(0) = 0,$$

where each minimization is initialized at feasible $x(0) = x_c^*$. The dual variable is set to some initial value λ^0 independent of the solution to the relaxed problem in the first phase. This is because the problems in the first and second phase are separate, and the constraints are changed due to projection onto the continuous space. All the parameters used in TACNLP are set to their initial values used in the first phase of the minimization, see Table 6.3. The updates for x and λ, the adaptive step size implementation, the penalty parameter update, scaling and the convergence conditions (5.27), are identical to that of the continuous algorithm described in Section 5.1. For each $\bar{y}^i$, the solution of (8.17) is denoted by$(x_i^*, \bar{y}^i)^T$. The optimal feasible solution $(x_f^*, y_f^*)^T$, is obtained, such that

$$(x_f^*, y_f^*) = \arg\min_{x_i^*, \bar{y}^i} \{f(x_i^*, \bar{y}^i)\}. \tag{8.18}$$

Since x is fixed when generating the trial points $\bar{y}^i$, there is no guarantee that $\mathscr{M}(\bar{y}^i)$ is feasible in x. If $\mathscr{M}(\bar{y}^i)$ is infeasible in $x\ \forall i$, then $(x_i^*, \bar{y}^i)^T$ are infeasible for $\mathscr{M}$. In this case, new trial points need to be generated[1]. The details of which are now discussed.

[1] Although we initialize the system (8.17) with feasible x_c^* and feasible $\bar{y}^i$, we are unable to prove theoretically that the solution of (8.17) will be feasible. Our numerical experiments however always produced feasible $(x_i^*, \bar{y}^i)^T$.

8.4.3 Increasing the search space when no feasible solution is found

In the second phase of TAMINLP, $\hat{P}$ feasible trial points are generated. Including $\bar{y}$, there are $\hat{P}+1$ continuous manifolds upon which separate minimizations with respect to x are performed, to find a feasible solution $(x_f^*, y_f^*)^T$. Here $(x_f^*, y_f^*)^T$ is the point which yields the current lowest objective function value of $\mathscr{M}$, defined by (8.1). Essentially we minimize $\mathscr{M}(\bar{y}^i)$ by solving $\phi_A^i(x, \bar{y}^i, \lambda; \mu)$ with respect to some initial conditions. However, $\mathscr{M}(\bar{y}^i)$ can be infeasible in x for all $\bar{y}^i$. If this is the case then all $(x_i^*, \bar{y}^i)^T$ are infeasible, and we need to generate trial integer points further away from $\bar{y}$.

A similar consideration is used by Newby [90]. Newby defines the neighborhood $\mathscr{N}_d(\bar{y}, \varepsilon_d)$ given by equation (8.4), where all integer points which fall within the radius of the neighborhood, ε_d, are considered for the search space. This neighborhood can be increased at the user's discretion, by increasing ε_d. As a result, the number of possible integer solutions considered will be increased and the accuracy of the algorithm will improve. We look at this idea in more depth in Section 8.7.

By defining $\mathscr{N}_d(\bar{y}, \varepsilon_d)$, we have control over the number of integer trial points we generate. Using the idea of expanding $\mathscr{N}_d(\bar{y}, \varepsilon_d)$, the discussion on how new trial points are generated when no feasible solution corresponding to the initial trial points $\bar{y}^i$ is found, is now presented.

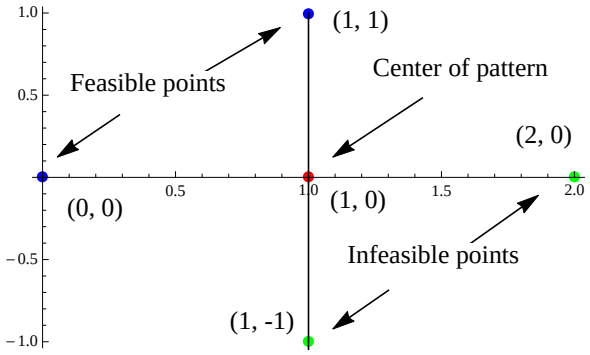
We again use the previous example in $\mathbb{R}^2$, for the case of binary integer variables. Thus far the four points

$$\{(1,0)^T, (0,1)^T, (-1,0)^T, (0,-1)^T\}, \tag{8.19}$$

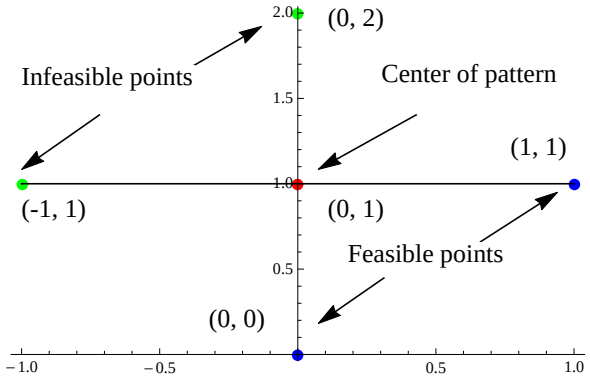
have already been examined, see Figure 8.5. Here, the radius of the neighborhood $\mathscr{N}_d(\bar{y}, \varepsilon_d)$ is 1. This neighborhood is increased by one unit whenever no feasible solution is obtained. Assuming that all solutions produced by (8.17), using the feasible $\bar{y}^i$ in (8.19) are infeasible, we now generate new integer-feasible trial points within the neighborhood of $\bar{y}$, $\mathscr{N}_d(\bar{y}, \varepsilon_d)$, with $\varepsilon_d = 2$. This is done by generating another 4 trial points about each of the feasible trial points $(1,0)^T$ and $(0,1)^T$, using them as $\bar{y}^0$. As a result we obtain the set of 8 new trial points, see Figure 8.6. The feasible and infeasible points generated here are illustrated in Figure 8.6, where the feasible points are depicted by the blue points and the infeasible points are depicted by the green points.

Some patterns occur in the generation of the new integer trial points. To demonstrate this denote the new trial points by

$$\bar{y}^{ij} = d_i + d_j, \tag{8.20}$$



(a) Trial points generated about the center of pattern, $(1,0)^T$.



(b) Trial points generated about the center of pattern, $(0,1)^T$.

Figure 8.6 The set of new trial points generated using feasible trial points as pattern centers

where i and j are indices corresponding to the set of feasible directions $d_i, d_j \in D$, and D is defined by (8.12)[2]. We do not use infeasible directions or infeasible pattern centers for the generation of new trial points. This is motivated in the discussion below.

Notice that the integer trial points generated around the pattern center $\bar{y}^0 = (1,0)^T$, along the feasible directions $(1,0)^T$ and $(0,1)^T$, are:

$$\bar{y}^{11} = \bar{y}^0 + d_1 + d_1 = (2,0)^T,$$

and

$$\bar{y}^{12} = \bar{y}^0 + d_1 + d_2 = (1,1)^T,$$

respectively, see Figures 8.6a. Notice also that the integer trial points generated around the pattern center, $\bar{y}^0 = (0,1)^T$, along feasible directions, are

$$\bar{y}^{21} = \bar{y}^0 + d_2 + d_1 = (1,1)^T,$$

and

$$\bar{y}^{22} = \bar{y}^0 + d_2 + d_2 = (0,2)^T,$$

respectively, see Figure 8.6b. Several observations are made. Firstly, we observe that $\bar{y}^{12} = \bar{y}^{21}$. Secondly, we observe that $\bar{y}^{11}$ and $\bar{y}^{22}$ are not binary, and are therefore infeasible. Lastly, increasing the discrete neighborhood $\mathcal{N}_d(\bar{y}, \varepsilon_d)$ along infeasible directions results in infeasible points, i.e., $(-1,1)^T$ and $(1,-1)^T$, or points which are repeated[3], i.e., $(0,0)^T$, see Figure 8.7:

We rectify these issues as follows:

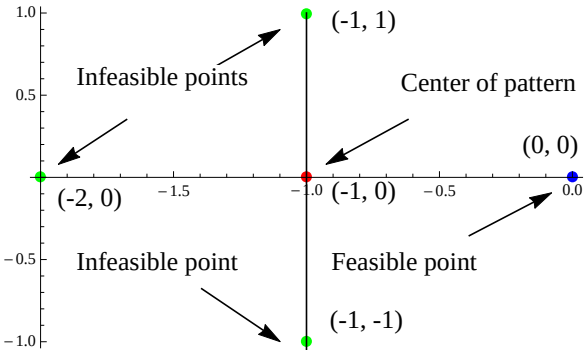
- To avoid the kind of repetitive redundancy displayed by $\bar{y}^{12} = \bar{y}^{21}$, once a solution y_d^{ij} has been obtained, we do not calculate y_d^{ji}, where i and j are indices corresponding to a subset of feasible directions $d_i, d_j \in D$.

- When y is binary, we have seen that generating trial points with repeated indices results in infeasible trial points. For this reason, when y is restricted to binaries, the trial points with repeated indices, $\bar{y}^{ii}$ are not calculated, where i is the index corresponding to the set of feasible directions $d_i \in D$ from the first phase.

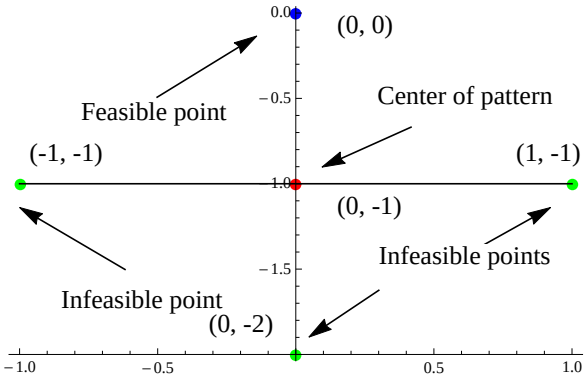
- When $\mathcal{N}_d(\bar{y}, \varepsilon_d)$ is expanded, it is done along feasible directions only. Recall that the first set of integer trial points generated consisted of 4 points, see Figure 8.5.

[2]Recall that feasible directions are those which yield the initial set of integer-feasible trial points in (8.14).

[3]The initial pattern center, $\bar{y}$ is equal to $(0,0)^T$ and would have already been considered, hence $(0,0)^T$ is not a unique integer trial point.



(a) Trial points generated about the center of pattern, $(-1,0)^T$.



(b) Trial points generated about the center of pattern, $(0,-1)^T$.

Figure 8.7 The set of new trial points generated using infeasible trial points as pattern centers

Performing another POLL about each of those points results in the generation of $2{n_d}^2 = 16$ new integer trial points, some of which are infeasible, see Figures 8.6 and 8.7. However, by expanding the discrete neighborhood $\mathcal{N}_d(\bar{y}, \varepsilon_d)$ about integer-feasible points, along feasible directions only, we only calculate 1 new trial point, i.e., $\bar{y}^{12} = (1,1)^T$. This integer-feasible trial point could be generated using either of the feasible pattern centers $(1,\ 0)^T$ or $(0,\ 1))^T$ and traversing along either of the feasible directions $(0,\ 1)^T$ or $(1,\ 0))^T$ respectively, see Figure 8.6.

We summarize the above process as follows. When we generate the first set of integer-feasible trial points, the radius of $\mathcal{N}_d(\bar{y}, \varepsilon_d)$ is 1:

$$\mathcal{N}_d(\bar{y}, 1) = \{y \in \Omega_d : ||y - \bar{y}|| \leq 1\}. \tag{8.21}$$

This yields the integer-feasible trial points $(1,\ 0))^T$ and $(0,\ 1))^T$. We then increase this neighborhood by increasing ε_d by one unit:

$$\mathcal{N}_d(\bar{y}, 2) = \{y \in \Omega_d : ||y - \bar{y}|| \leq 2\}, \tag{8.22}$$

to obtain the integer-feasible trial point $(1,\ 1)^T$. If however no feasible solution x_f^* corresponding to this point is obtained, then we continue increasing our discrete neighborhood (8.22) until a feasible solution is found. For the binary example in $\mathbb{R}^2$ however, we have done an exhaustive enumeration of all 0 - 1 combinations[4].
Once we have identified the set of new integer-feasible trial points, we perform continuous minimizations with respect to the continuous variable x, described in Section 8.4.2, to obtain $(x_f^*, y_f^*)^T$.

The discussion above only pertains to the case where the integer variables are binary. A similar strategy is however used when the integer variables take on general integer values. Firstly, the trial points are generated exactly the same way for general integer and binary integer variables. Feasible trial points are then selected to satisfy any restrictions on the general integer variables, i.e lower and upper bounds, which we denote by l_i and u_i respectively, as given in (8.10). For the binary case these bounds are just 0 and 1 respectively. The rest of the second phase proceeds in the exact same way for binary and general integer variables. It is

[4]For any binary problem, there are 2^{n_d} integer-feasible combinations. For the binary example above these combinations are: $(0,\ 0)^T$, $(1,\ 0)^T$, $(0,\ 1)^T$ and $(1,\ 1)^T$. Thus, for the example illustrated in Figures 8.4 - 8.6, we have done an exhaustive enumeration of 0 -1 alternatives (integer-feasible trial points), within $\mathcal{N}_d(\bar{y}, \varepsilon_d)$. This means that no new integer-feasible point will be found by increasing the radius of the discrete neighborhood $\mathcal{N}_d(\bar{y}, \varepsilon_d)$.

also easy to envisage a strategy for problems where both binary and general integer variables are present.
We summarize the important items discussed during the second phase of TAMINLP, in Table 8.2 below:

	notation	Description
1.	$\bar{y}$	The value obtained by fixing y_c^* to the nearest feasible integer point.
2.	$d_i,\ i = 1,\ldots,2n_d$	The coordinate directions used to generate trial points using the rule of PS.
3.	$\bar{y}^i,\ i = 1,\ldots,2n_d$	The integer trial points, generated using the rule of PS.
4.	$\mathscr{N}_d(\bar{y},\varepsilon_d)$	A discrete user defined neighborhood used to control the number of integer- feasible trial points generated in 3, for the second phase of TAMINLP.
5.	ε_d	The radius of the discrete neighborhood, $\mathscr{N}_d(\bar{y},\varepsilon_d)$ in 4.
7.	$\hat{P}$	The number of integer-feasible trial points generated
8.	$(x_i^*,\bar{y}^i), i = 1,\ldots\hat{P}$	The optimal solution obtained on the manifold corresponding to fixed value $\bar{y}^i$.
9.	$\bar{y}^{ij}$, where $d_i,d_j \in D$	If x_i^* are infeasible $\forall i$ in 8, then ε_d is increased by one unit and new integer trail points y^{ij} are generated.
10.	$z_f^* = (x_f^*,y_f^*)^T$	The best solution obtained, by solving $\bar{\mathscr{M}}(\bar{y}^i)$ for different fixed values of feasible $\bar{y}^i$ (or $\bar{y}^{ij}$).

Table 8.2 Important items used in the discussion of the second phase of TAMINLP minimization.

8.5 The third phase of TAMINLP

In the third phase of TAMINLP, we apply TACNLP to the continuous relaxation of $\mathscr{M}$. Our motivation for this is as follows. Consider again any integer problem which is restricted to binaries. When n_d is large it is impractical to consider all 0 - 1 combinations, for the second phase of TAMINLP. Even when n_d is relatively small, i.e., $n_d = 5$, the number of 0 - 1 combinations which make up integer-feasible points are $2^{n_d} = 2^6$=32, see Appendix (A.1). Exploring all 32 points is computationally expensive. A similar argument applies for general integer problems. As a result, the optimal integer solution y^* may go unexplored. An example is used to illustrate this possible exclusion of y^*.

Consider a binary example in $\mathbb{R}^5$, with $\bar{y} = (0,0,0,0,0)^T$. The $2^5 = 32$ feasible 0 - 1 combinations for this problem are given in Appendix (A.1). Assuming the optimal solution for this problem is $y^* = (1,1,1,1,0)^T$, we now show that its computationally impractical to generate the trial point $y^* = (1,1,1,1,0))^T$, via the procedure discussed in phase two of TAMINLP.

In order to locate $y^* = (1,1,1,1,0))^T$, we would have to generate four sets of different trial points as follows:

For the generation of the first set of trial points, $2n_d = 10$, integer points are generated, as shown in the columns of I:

$$I = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \end{Bmatrix}, \tag{8.23}$$

Only 5 of the points in (8.23) are integer-feasible, and therefore lie within the neighborhood:

$$\mathscr{N}_d(\bar{y},1) = \{y \in \Omega_d : ||y - \bar{y}|| \leq 1\}. \tag{8.24}$$

The integer-feasible points, which are generated along the corresponding n_d feasible directions are given in the columns of I_1.

$$I_1 = \begin{Bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{Bmatrix}. \tag{8.25}$$

Evidently, the optimal point $y^* = (1,1,1,1,0))^T$ is not contained in this set. We then generate the second set of trial points. By employing the strategy discussed in Section 8.4.3, which excludes the generation of repeated and infeasible trial points, we generate 10 integer-feasible

trial points:

$$I_2 = \begin{Bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{Bmatrix}. \tag{8.26}$$

These points satisfy

$$\mathcal{N}_d(\bar{y}, 2) = \{y \in \Omega_d : ||y - \bar{y}|| \leq 2\}. \tag{8.27}$$

Again, the optimal point $y^* = (1,1,1,1,0))^T$ is not contained in this set. We generate a third set of 10 trial points which satisfy

$$\mathcal{N}_d(\bar{y}, 3) = \{y \in \Omega_d : ||y - \bar{y}|| \leq 3\}, \tag{8.28}$$

i.e.,

$$I_3 = \begin{Bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{Bmatrix}, \tag{8.29}$$

and a fourth set of 5 trial points:

$$I_4 = \begin{Bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{Bmatrix}, \tag{8.30}$$

which satisfy

$$\mathcal{N}_d(\bar{y}, 4) = \{y \in \Omega_d : ||y - \bar{y}|| \leq 4\}, \tag{8.31}$$

before we finally locate the optimal point $y^* = (1,1,1,1,0))^T$. Consequently, we would have to perform $\sum_{i=1}^{\mathrm{I_i}^4} = 30$ separate minimizations before y^* is located. This number increases for problems of higher dimensions.

We consider the MINLP example found in Duran et al. [42], with optimal solution $y^* = (0,1,0,1,0,1,0,1))^T$, and $\bar{y} = (1,0,0,1,0,1,0,0))^T$. This optimal solution is only obtained after expanding $\mathscr{N}_d(\bar{y},\varepsilon_d)$ thrice and performing more than 50 separate minimizations. This is computationally impractical.

It is easy to see how the optimum y^* may be excluded in the second phase of TAMINLP. In order to increase the probability of obtaining the optimal solution, without repeating many more integer-feasible solutions, we perform a continuous local search from all points whose objective function values lie within some neighborhood of $f(x_f^*,y_f^*)^T$. This strategy, for the final phase of TAMINLP, is presented below.

8.5.1 Selecting points for the final phase of TAMINLP

For the final phase of TAMINLP, we select the solutions from the second phase, whose objective function values lie within some user defined neighborhood $\mathscr{N}_m(x_f^*,y_f^*)^T$ of $(x_f^*,y_f^*)^T$:

$$\mathscr{N}_m(x_f^*,y_f^*) = \{(x_i^*,\bar{y}^i) \in \Omega_m : ||f(x_f^*,y_f^*) - f(x_i^*,\bar{y}^i)|| \leq \underline{\varepsilon}\}. \tag{8.32}$$

Once we have identified these points, TACNLP is applied to $\underline{\mathscr{M}}$ for each point $(x_i^*,\bar{y}^i)^T$ satisfying (8.32). Since $(x_f^*,y_f^*)^T$ satisfies (8.32) identically, we also perform a continuous minimization about $(x_f^*,y_f^*)^T$. Let $x_1^* = x_f^*$, and $\bar{y}^1 = y_f^*$, then assuming there are $\underline{P}$ points which satisfy (8.32), we perform separate continuous minimization with respect to $\underline{P}$ augmented Lagrangians:

$$\phi_A^i(z,\lambda;\mu)),\ i = 1,\ldots,\underline{P}. \tag{8.33}$$

Particularly, we apply TACNLP to $\underline{\mathscr{M}}$ and solve the system:

$$\ddot{z} = -\nabla_z \phi_A^i(z,\lambda;\mu), \quad z(0) = (x_i^*,\bar{y}^i))^T \quad \dot{z}(0) = 0,$$

$$\ddot{\lambda} = \nabla_\lambda \phi_A^i(z,\lambda;\mu), \quad \lambda(0) = \lambda^0, \quad \dot{\lambda}(0) = 0,$$

$$\ddot{\mu} = -\mu, \quad \mu(0) = \mu^0, \quad \dot{\mu}(0) = 0, \tag{8.34}$$

for each $i = 1, \ldots, \underline{P}$. Each minimization is initialized at the solutions to the second phase which satisfy (8.32). Although we are performing a continuous local search from each of these low lying feasible points, there is no guarantee that any of the optimal solutions will produce integer solutions of the integer part. Depending on whether the integer part of the solution is real or integer, we proceed as follows.

- If the integer part of the best solution to (8.34) is integer, then it is compared with the solution to the second phase $(x_f^*, y_f^*)^T$, to determine which of these is the best. The best point is chosen as the final solution $(x^*, y^*)^T$.
- If the integer part of the best solution to (8.34) is non-integer, then we round it to the nearest integer-feasible point. We also identify all the other solutions obtained during the final phase, rounding all of those which are non-integer, to the nearest integer-feasible point. We then compare all these points with $(x_f^*, y_f^*)^T$. The best of these solutions $z^* = (x^*, y^*)^T$, is chosen such that:

$$f(x^*, y^*) \leq f(\tilde{x}, \tilde{y}). \tag{8.35}$$

Here, $\tilde{z} = (\tilde{x}, \tilde{y})^T \in F$, where $F = F_1 \cup F_2$:

$F_1 = \{(x_f^*, y_f^*)^T\}$,

$F_2 = \{(\hat{x}, \hat{y}))^T \in \Omega_m \mid (\hat{x}, \hat{y})^T\}$,

where:

  - F_2 contains all the solutions obtained during the third phase, with $\hat{y}$ integer-feasible. Here, $\hat{y}$ are either obtained as a solutions to (8.34), or by rounding the non-integer solutions to (8.34) to the nearest integer-feasible points.

A summary of important items introduced in this section is presented in Table 8.3 below.

notation	Description
$\mathcal{N}_m(x_f^*, y_f^*)^T$	Some user defined neighborhood of $(x_f^*, y_f^*)^T$.
$\underline{P}$	The number of points whose objective function value lie within $\mathcal{N}_m(x_f^*, y_f^*)^T$
$z^* = (x^*, y^*)^T$	The overall optimal solution of TAMINLP.

Table 8.3 Important items used in the discussion of the third and final phase of TAMINLP minimization.

8.6 The Pseudo-code of TAMINLP

Here, we present the pseudo-code of TAMINLP. Important steps in the pseudo-code are further explained by remarks.

Algorithm 10 TAMINLP

1: The initialization of TAMINLP is done as in TACNLP.
2: Obtain a solution $(x_c^*, y_c^*))^T$, by solving (8.9) as in **Algorithm 8**.
3: **if** y_c^* is integer **then**
4: Stop
5: **else** Obtain $\bar{y}$ by fixing y_c^*, and calculate $\bar{y}^i,\ i = 0,\ldots,n_s$, using the rule of PS.
6: **end if**
7: Find $\hat{P}$ integer-feasible vectors, $\bar{y}^i,\ i = 1,\ldots\hat{P}$.
8: Solve (8.17), for $\hat{P}$ initial points as in **Algorithm 8**, to find the feasible solution $(x_f^*, y_f^*)^T$ which satisfies (8.18), such that

$$f(x_f^*, y_f^*) \leq f(x_i^*, \bar{y}^i),$$

$i = 1,\ldots,\hat{P}$ and **go to 9**.
9: **if** $(x_i^*, \bar{y}^i)^T \notin \Omega_m, \forall i$, **then**
10: set $\varepsilon_d^{k+1} = \varepsilon_d^k + 1$, calculate new feasible trial points using (8.20) and **go to** 5.
11: **end if**
12: Identify all the points $(x,\ y)^T \in \mathscr{N}_m(x_f^*, y_f^*)$.
13: Obtain the solution $(x^*, y^*)^T$ which satisfies (8.35), by solving $\underline{\mathscr{M}}$ for each point $(x,y)^T \in \mathscr{N}_m(x_f^*, y_f^*)$.

Remarks on Algorithm 10

Remark 8.6.1. In line 2 of **Algorithm 10**, the first phase of TAMINLP is implemented, to obtain $(x_c^*, y_c^*)^T$.

Remark 8.6.2. In line 3, If the solution y_c^* to (8.9) is integer, then the algorithm terminates as the optimal solution has been found. If the solution y_c^* is non-integer then it is rounded to the nearest feasible integer. The rule of PS described in Section 8.4.1, is then implemented in line 5 of **Algorithm 10**.

Remark 8.6.3. In line 7, the integer-feasible points $\bar{y}^i,\ i = 1,\ldots,\hat{P}$ are identified, which are used to initialize the second phase of TAMINLP.

Remark 8.6.4. In line 8, the second phase of TAMINLP is implemented for all $\hat{P}$, to obtain $(x_f^*, y_f^*)^T$.

Remark 8.6.5. If no feasible solution, $(x_i^*, \bar{y}^i)^T$ is obtained in the neighborhood of radius ε_d, then we expand our search space. This is done by increasing the radius of the neighborhood ε_d until a feasible solution is obtained. This process is described in lines 9 - 11 of **Algorithm 10**.

Remark 8.6.6. In lines 12 - 13, the third phase of TAMINLP is implemented, to obtain the local minimizer $(x^*, y^*)^T$.

8.7 Convergence

In this section, we study the convergence properties based on the definition of a local minimum proposed by Newby [90, 91]. We also argue that this definition fails if we do not consider all points in the neighborhood $\mathcal{N}_d(\bar{y}, \varepsilon_d)$, with $\varepsilon_d \geq 1$.

In order to study the convergence properties of TAMINLP, we use the **Definition 8.1.6** of a local minimizer in Ω_m.

Once the solution to the first phase of TAMINLP has been obtained all the feasible initial points for the second phase of the minimization can be defined by the set $\mathcal{N}_r(x_c^*, \bar{y})$ in **Definition 8.1.6**:

$$\mathcal{N}_r(x_c^*, \bar{y}) = \{(x,\ y)^T :\ x = x_c^*, ||y - \bar{y}|| \leq \varepsilon_d\}, \tag{8.36}$$

where ε_d is typically equal to 1. We initialize the second phase of the minimization from each of these points, which is equivalent to finding the (smallest) local minimum on each feasible continuous manifold on which $\mathcal{N}_r(x^*, y^*)$ has a point. Since we are interested in obtaining the smallest of these minima, we identify the feasible point $(x_f^*, y_f^*)^T$ which satisfies:

$$\begin{aligned} &f(x_f^*, y_f^*) \leq f(x,y), \quad \forall\ (x,y) \in \{x \in B_\varepsilon(x_f^*),\ y = y_f^*\} \cap \Omega_m, \\ &f(x_f^*, y_f^*) \leq f(x,y), \quad \forall (x,y) \in \mathcal{N}_{comb}(x_f^*, y_f^*) \cap \Omega_m. \end{aligned} \tag{8.37}$$

According to Newby [90] this point $(x_f^*, y_f^*)^T$ is the optimal solution to (8.1), i.e., $(x^*, y^*)^T$. However, since only a subset of all the points satisfying (8.36) with $\varepsilon_d \geq 1$, is considered, it is possible that y^* may be excluded from this subset (this was illustrated with an example, at the beginning of Section 8.5). This is the shortcoming of **Definition 8.1.6**.

Often, if $\bar{y}$ is feasible and a corresponding feasible x_i^* exists, then the pair $(x_i^*, \bar{y})^T$ is optimal, i.e., $(x^*, y^*)^T = (x_f^*, y_f^*)^T = (x_i^*, \bar{y})^T$. This is because $\bar{y}$ is the closest integer to the continuous optimum y_c^*. In this case, **Definition 8.1.6** suffices. If however $\bar{y}$ is obtained with

rounding error then it may not represent the nearest integer to y_c^*. An example is used to illustrate this.

Suppose a problem has the optimal integer solution $y^* = 1$. Assume that $y_c^* = 0.4999$ is the solution obtained after the first phase of TAMINLP, and that $y = 0$ and $y = 1$ are both integer-feasible. Rounding y_c^* to the nearest integer, we obtain $\bar{y} = 0$, and not $\bar{y} = 1$, which is the optimum. Since $y = 0$ is one unit from $y = 1$ however, by generating feasible trial points in $\mathscr{N}_d(\bar{y}, \varepsilon_d)$, where $\varepsilon_d = 1$, via the second phase of TAMINLP, we are able to recover the optimal solution $y^* = 1$. When n_d is much larger though, say $n_d = 1000$, this becomes difficult. If 20 of the 1000 integer variables are rounded incorrectly as a result of numerical error, then this means we are 20 units away from the optimal solution. In order to recover the optimal solution, we would need to generate feasible trial points in $\mathscr{N}_d(\bar{y}, \varepsilon_d)$, where $\varepsilon_d = 20$. As a result the number of integer trial points which need to be generated is extremely high. In this case **Definition 8.1.6** falls short again.

Thus far we have illustrated two cases which highlight the limitation of **Definition 8.1.6**:

1) The optimal point y^* has been excluded from the subset of candidate trial points generated for the second phase of TAMINLP.

2) The point $(x_i^*, \bar{y})^T \in \Omega_m$, but due to rounding error $\bar{y}$ is not the closest integer-feasible point to y_c^*.

In both cases it is impractical to recover the optimal solution y^*. Therefore, at best we can only guarantee that the point $(x_f^*, y_f^*)^T$ will be found during the second phase of TAMINLP. We therefore denote the feasible point satisfying (8.37) by $(x_f^*, y_f^*)^T$ and not by the actual local minimizer $(x^*, y^*)^T$.

To account for the possible exclusion of y^* during the second phase of TAMINLP, we perform the third phase of TAMINLP.

In the remainder of this chapter, we discuss the convergence of TAMINLP. Recall that we have already established global and local convergence for TACNLP, which is used in the 3 separate phases of TAMINLP. All that is left to prove is that our choice of $\bar{y}_d^{(i)}, i = 1, \ldots \hat{P}$, which determines the number of continuous manifolds upon which we perform minimizations with respect to x, is optimal. Lastly we need to prove that the third phase of the minimization ensures convergence to the optimal solution $(x^*, y^*)^T$.

8.7.1 Convergence of TAMINLP

We reiterate that TAMINLP is designed to locate a local solution of any given problem. This discussion is therefore centered around said local minimizer.

We begin by establishing relationships between y_c^*, $\bar{y}$ and y^*. Firstly, we know that the solution to the relaxed problem $\underline{\mathscr{M}}$, defined by (3.5), i.e., $(x_c^*, y_c^*)^T$ yields a lower bound for the solution to the MINLP $\mathscr{M}$, i.e., $(x^*, y^*)^T$. That is:

$$f(x_c^*, y_c^*) \leq f(x^*, y^*),$$

where

$$f(x^*, y^*) = f(x_c^*, y_c^*),$$

if y_c^* is integer. We also know that

$$f(x_c^*, y_c^*) \leq f(x_c^*, \bar{y}),$$

where

$$f(x_c^*, y_c^*) = f(x_c^*, \bar{y}),$$

if y_c^* is integer. Recall that the feasible trial points generated for the second phase are denoted by $\bar{y}^i, i = 1, \ldots n_s$ and that $\bar{y}^0 = \bar{y}$. For the rest of the discussion, we make the following assumption.

Assumption 12. *The objective function f and constraints c of the relaxed problem $\underline{\mathscr{M}}$ defined by* (3.5)*, are convex near the relaxed solution $(x_c^*, y_c^*)^T$ of $\underline{\mathscr{M}}$.*

It follows from **Assumption 12**, that if $(x_c^*, \bar{y}^0)^T$ is the closest candidate point to $(x_c^*, y_c^*)^T$, then every other candidate point, i.e., every integer-feasible trial point $(x_c^*, \bar{y}^i)^T$, will yield a higher function value than both $(x_c^*, y_c^*)^T$ and $(x_c^*, \bar{y}^0)^T$:

$$f(x_c^*, y_c^*) \leq f(x_c^*, \bar{y}^0) \leq f(x_c^*, \bar{y}^i),\ i = 1, \ldots, \hat{P},$$

Provided the continuous manifold corresponding to $\bar{y}^0$ is feasible, this manifold minimizer will correspond to the solution $(x_f^*, y_f^*)^T$, where $y_f^* = \bar{y}^0$.

If the continuous manifold corresponding to $\bar{y}^0$ is not feasible in x, then no feasible x_i^* corresponding to $\bar{y}^0$ exists. By the convergence of TACNLP, at least one of the remaining feasible trial points $\bar{y}^i\ i = 1, \ldots, n_s$ will converge to $(x_f^*, y_f^*)^T$. If none of these trial points are feasible, then the initial discrete neighborhood $\mathscr{N}_d(\bar{y}, \varepsilon_d)$ is increased until the feasible solution $(x_f^*, y_f^*)^T$ is obtained. Once again convergence to $(x_f^*, y_f^*)^T$ is guaranteed by the convergence of TACNLP.

Since the trial points generated are not exhaustive of all the possible candidate points for the second phase, we execute the third phase of TAMINLP to locate the local optimum

$(x^*, y^*)^T$. Since the third phase of TAMINLP is equivalent to solving TACNLP, we know that convergence to a local solution is guaranteed.

Chapter 9

Numerical Results for CNLPs

In this chapter, we present numerical results for the algorithms presented in Chapters 5 and 6. In order to make any comments on their practical merit, it is necessary to test the performance of each algorithm. In Sections 9.1 and 9.2, numerical results for a MATLAB implementation of TACNLP and ATAUNLP respectively, are presented. Both algorithms are tested on a test set of 71 problems which include problems from the CUTEr test set. A comparison of the performance of TACNLP and ATAUNLP is presented in Section 9.3.

For further comparison, the CUTEr test problems are solved using the benchmarking solver SNOPT [56]. The numerical results of this study are presented and discussed in Section 9.4.

In Section 9.5, a final analysis is done to test the effectiveness of the adaptive step size method as well as the scaling routine introduced in this thesis. To do this a comparison is made between Snymans unconstrained trajectory-based algorithm [115] and our extension of this algorithm for unconstrained problems. We denoted our version of the unconstrained trajectory-based algorithm by ETAUNLP.

All tests in this chapter were performed on a PC with an Intel Core i5 CPU at 2.5 GHz with 4GB of 1333 MHz RAM, running OS X 10.8.5. All Algorithms were coded in MATLAB 2013a 64 bit.

9.1 Results and discussion for TACNLP

In this section, we preset the results for TACNLP. A detailed discussion of the test problems as well as the parameters used for the implementation of TACNLP is presented in Subsection 9.1.1. This is followed by a discussion of the numerical results, in Subsection 9.1.2.
Throughout the rest of this chapter, we make use of two concepts extensively. The first is the penalty parameter *updating criteria*, and the second is the penalty parameter *updating*

scheme. The *updating criteria* determines "when" μ^k will be updated, as per PC1, PC2 or PC3, while the penalty parameter *updating scheme* specifies "how" μ will be updated, as per the new$-\mu$ or conventional$-\mu$ *updating schemes* described in **Algorithms 5** and **6** respectively.
The discussion in Subsection 9.1.2 includes an examination of the effects of using the three different *updating criteria* for μ, i.e., PC1, PC2 and PC3. The comparison is done to determine which of these *updating criteria* is the most efficient. The section is concluded with a comparison of the *updating schemes* for the penalty parameter, i.e., new$-\mu$ and conventional$-\mu$.

9.1.1 Test problems and parameters

The CNLP test problems were collected from Escudero [44], Hock et al. [68] and the CUTEr test collection [23]. A total of 71 nonlinear constrained problems were identified. Of these problems, 12 have been obtained from the CUTEr test set, i.e., problems 60 (BT1) - 71 (BT12). The entire test set can be found in Appendix B. The structure of each problem is further summarized in Appendix A, in Table A.1. Important information such as the optimal value for each problem x^*, and the solution obtained by the TACNLP algorithm, **Algorithm 8** x^k, are also provided in Table A.2.

For the numerical experiments the parameters listed in Table 6.3 were used. For certain problems however, the parameter values specified in Table 6.3 were not optimal. Specifically, certain problems from the test set were either unable to converge or converged with an excessively high number of iterations unless some of the values in Table 6.3 were modified slightly. Test runs were therefore conducted to obtain the optimal values of some parameters. The first set of experiments were conducted with TACNLP using various values of the step size parameters $t_{(x,2)}$ and $t_{(\lambda,2)}$ (in (5.21)). For each problem a number of values for $t_{(x,2)}$ and $t_{(\lambda,2)}$, within the range of $[1,\ 2]$, were used to determine which of these are optimal. In these experiments the other variables were kept constant.

We summarize the problems and the optimal parameter values obtained from the experiments, in Tables 9.1 - 9.2 below.

Problems	$t_{(x,2)}$	$t_{(\lambda,2)}$
1	1.5	2
39, 57	1.1	1.5
54, 56, 68	1.1	2
45, 59, 70	1.1	1.1

Table 9.1 Test problems which did not converge unless the specified parametric values were used, as opposed to the default values stipulated in Table 6.3

With reference to Table 9.1, there are 9 test problems which did not converge unless the assigned parameter values in this table were used.

In the next table, we summarize the problems as well as the time step parameter values used to speed up convergence.

Problems	$t_{(x,2)}$	$t_{(\lambda,2)}$
6, 42, 52	1.5	2
17, 33	2	1.5
16, 25, 44	1.5	1.5
43	1.1	2
58, 61	2	1.1
55	1.1	1.1

Table 9.2 Test problems which converged faster using the specified parametric values as opposed to the default values stipulated in Table 6.3

With reference to Table 9.2, 12 of the problems from the test set converged faster with the parameter values for $t_{(x,2)}$ and $t_{(\lambda,2)}$ specified in this table.

The second experiment conducted, was done to determine problems needing scaling. Within the solution process of TACNLP, the scaling mechanism discussed in Chapter 6 was implemented for 16 problems from the test set. These problems are listed in Table 9.3. Of these, problems 45, 55 - 57, 59, BT10 and BT11 were implemented by scaling ∇f and $c(x)$ only. By scaling ∇c as well, convergence was hindered. Table 9.3 lists the problems and their respective scaled functions. The table entry ✓ indicates that the corresponding vector or function was scaled and the table entry ✗ indicates that it was not.

Problems	$\nabla f(x)$	$c(x)$	$\nabla c(x)$
16, 19, 22, 24, 29,	✓	✓	✓
BT2, BT4, BT6, BT7	✓	✓	✓
45, 55, 56, 57, 59, BT10,BT11	✓	✓	✗

Table 9.3 Scaled problems

Problems	μ_{min}	μ_{max}
1	1	1
45, 50, 58, 59, BT11	0.01	1
40	0.1	10

Table 9.4 Test problems with specifications for the penalty parameter, which differ from those listed in Table 6.3

Whenever scaling is implemented, using **Algorithm 7**, the values $l_i = -50$ and $u_i = 50$ are used for all test problems. Recall, these values are assigned to generate the q random vectors used during scaling, as per (5.22) in Section 5.2.3.

The penalty parameter μ plays a crucial role in locating the optimal solution to TACNLP. The final parameter experiments were conducted to determine the optimal values for μ_{min} and μ_{max}. For each problem experiments were conducted with TACNLP using a number of values for μ_{max} within the range of $(1,\ 10)$, and μ_{min} within the range of $(0.01,\ 1)$. Most problems converged easily without having to drive the penalty parameter μ to 0. A few problems were implemented with parametric value specifications which differed from the default values listed in Table 6.3. These are summarized in Table 9.4 below:

With reference to Table 9.4, seven problems converged more efficiently using values for μ_{min} and μ_{max} which differed from those in Table 6.3. Problem 1 converged with $\mu_{max} = \mu_{min} = 1$, problems 45, 50, 58, 59 and BT11 converged with $\mu_{min} = 0.01$, and problem 40 converged with $\mu_{max} = 10$, see Table 9.4. Every other problem in the test set converged with $\mu_{max} = 1$ and $\mu_{min} = 0.1$.

Remark 9.1.1. For every test problem the initial vector of multiplier estimates was set to the zero vector of corresponding dimension.

We now examine the performance of TACNLP.

9.1.2 Results for TACNLP

Tabulated results based on the performance of TACNLP are presented in Appendix A, in Tables A.1 - A.3. TACNLP was implemented with PC1, PC2 and PC3. This was done to determine which of these criteria yields the best results.
In Table A.1, columns 1 through 3 give the problem P, the number of variables n, and the structure of the objective function $f(x)$. Columns 4 to 7 give the number of linear equality constraints LE, the number of linear inequality constraints LI , the number of nonlinear equality constraints NE, and the number of nonlinear inequality constraints NI. Furthermore, columns 8 through 10 give k_1, k_2 and k_3 which correspond to the iteration number of TACNLP using PC1, PC2 and PC3 respectively. The *updating criteria* PC1, PC2 and PC3 were implemented using the new$-\mu$ *updating scheme* proposed in this thesis. Column 11 in each table gives the iteration number corresponding to an implementation of TACNLP using PC3 with the conventional$-\mu$ *updating scheme*, k_4 [6], [17, 18, 20, 130], described in **Algorithm 6**. We have specifically used PC3 when implementing the conventional$-\mu$ *updating scheme* since it yields better results than PC1 and PC2, see Table A.1. Lastly, columns 12 through 13 give the CPU time corresponding to an implementation of TACNLP using PC3 with the new$-\mu$ *updating scheme* t_{k_3}, and the CPU time corresponding to an implementation of TACNLP using PC3 with the conventional$-\mu$ *updating scheme* t_{k_4}. The CPU time when TACNLP reached the maximum iteration number $\bar{I}$, is also provided for each column t_{k_3} and t_{k_4}. The final row of Table A.1 summarizes the average number of iterations and the average CPU time taken by each implementation of TACNLP, to solve a problem.

In Table A.2, column 1 corresponds to the problem P, and column 2 gives the value of x^*. Columns 3 through 5 give the difference between x^* and the solution obtained by TACNLP, x^k, implemented with PC1, PC2 and PC3 respectively. Each implementation is done with the new$-\mu$ *updating scheme*. Lastly, column 4 gives the difference between x^* and the solution x^k obtained by TACNLP implemented with the conventional$-\mu$ *updating scheme*. Here, x^k corresponds to the final k-th iteration. Information on larger CNLP test problems which could not fit into Table A.2, is provided in Table A.3.

For the convergence of TACNLP, we have used the relatively large tolerance, $\varepsilon = 10^{-3}$, since this is a first order method. The algorithm terminates when condition (5.27) is satisfied. Apart from the algorithm converging to within the above precision, it terminates if the following situation occurs ($\bar{I}$): The number of iterations k in x^k exceeds 500. The solutions obtained under $\bar{I}$ are not far from the optimal. We have added another experiment, where we re-run all

problems to test an additional criterion for divergence. Here, the algorithm terminates if

$$10^3 \times ||x^0 - x^*|| < ||x^k - x^*||.$$

If the iterates diverge, then this has been denoted by ($\bar{D}$).

We now examine the effects of updating the penalty parameter using PC1, PC2 and PC3.

Comparison of *updating criteria* for the penalty parameter

One of the most significant findings from Table A.1 is that the different criteria for updating μ^k impact significantly on the number of iterations of the solution process. Recall that PC1, PC2 and PC3 correspond to an implementation of **Algorithm 5**, using $\underline{T}_1(x,\lambda)$, $\underline{T}_2(x,\lambda)$ and $\underline{T}_3(x,\lambda)$ respectively. These are defined as follows:

$$\underline{T}_1(x,\lambda) = \begin{pmatrix} c_i(x), & i \in E \\ \min\{c_i(x), \lambda_i\}, & i \in I \end{pmatrix}. \tag{9.1}$$

$$\underline{T}_2(x,\lambda) = \begin{pmatrix} \nabla_x \phi_A(x,\lambda;\mu) & \\ c_i(x), & i \in E \\ \min\{c_i(x), \lambda_i\}, & i \in I \end{pmatrix}, \tag{9.2}$$

$$\underline{T}_3(x,\lambda) = \begin{pmatrix} c_i(x), & i \in E \\ \lambda_i c_i(x), & i \in I \end{pmatrix}. \tag{9.3}$$

For each implementation, the penalty parameter is decreased, using (5.13), unless there is an improvement in Ψ :

$$\Psi_i(x^{k+1}, \lambda^{k+1}) \leq r\Psi_i(x^k, \lambda^k),$$

where

$$\Psi_i(x,\lambda) = ||\underline{T}_i(x,\lambda)||, i = 1, \ldots 3,$$

and r is defined as in **Algorithm 5**.

In the experiments conducted, TACNLP implemented with PC3 is seen to yield the overall best results, solving 28 of the test problems with the lowest iteration number, see Table A.1. PC1 yields similar results to PC3, for certain problems, because PC1 and PC3 are very

similar. PC1 however, solves only 21 of the test problems with the lowest iteration number, see Table A.1. TACNLP implemented with PC2 is seen to yield the worst convergence, solving only 15 of the test problems with the lowest iteration number. Every other problem converged within the exact same number of iterations or terminated with $\bar{I}$ or $\bar{D}$ for all three implementations (i.e., PC1, PC2 and PC3).

The superior performance of TACNLP, implemented with PC3 may be because it is a more relaxed criterion for updating μ, in comparison with PC1 and PC2. The inferior performance of TACNLP implemented with PC2 on the other hand may be a result of the more stringent requirement that unless there is some improvement in the augmented Lagrangian at $\{x^k, \lambda^k\}$, μ^k is decreased. Since we are solving the second order differential equation (5.1) in μ, where the solution is rapidly accelerated and μ is decreased rapidly, less frequent updates of μ are needed to maintain feasibility and complementarity of the constraints. The negative performance of TACNLP, implemented with PC2 can thus be attributed to the fact that μ is decreased frequently and consequently too rapidly to yield good progress of the iterates $\{x^k\}$, to a solution.

We assert this claim by plotting the progression of μ^k against k, obtained as a solution to (5.10). We do this for TACNLP implemented with PC1, PC2 and PC3, using problem 2 from the test set. Figure 9.1, illustrates how μ^k, updated as a solution to (5.10), changes with k. The plots corresponding to PC1, PC2 and PC3 are respectively represented in Figure 9.1 by TACNLP with new μ - PC1, TACNLP with new μ - PC2 and TACNLP with new μ - PC3.

The significance of the number 35 in Figure 9.1 is that TACNLP converges in 35 iterations using PC1 and PC2; TACNLP converges in 33 iterations using PC3, see Table A.1. Notice that μ attains a minimum value of 0.9528 using PC3 and attains minimum values of 0.9327 and 0.9303 with PC1 and PC2 respectively. Notice also that μ is decreased more rapidly when updated using PC2. This is inline with the conjecture we made previously stating that PC2 brings about a rapid decrease in μ, which may hamper convergence. This hampering of convergence is true for 30% of the test set, i.e., problems 6, 13, 15, 17, 24, 32, 34, 39, 42, 44, 48 - 50, 52 - 55, 57, 59, 66 and 69, see Table A.1.

Overall PC3 is the most effective criterion since it produces the least decrease in μ and yields better convergence than PC1 and PC2. This is the general trend for most of the problems in the test set, see Tables A.1 - A.2.

We now present a discussion around the different *updating schemes* for the penalty parameter, presented in **Algorithms 5** and **6**.

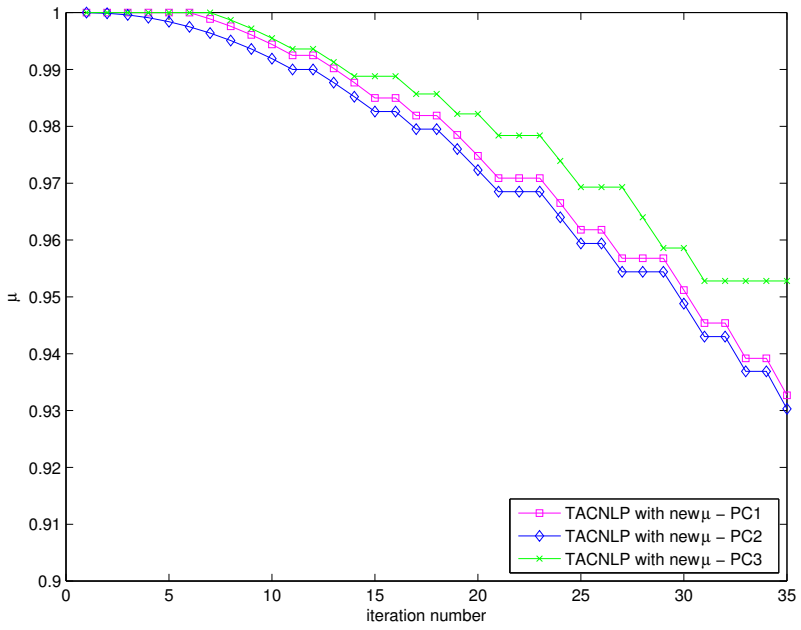


Figure 9.1 Progression of μ^k under PC1, PC2 and PC3 in TACNLP, using Problem 2

Comparison of *updating schemes* for the penalty parameter

We examine the effects of updating μ using the new$-\mu$ *updating scheme* versus using the conventional$-\mu$ *updating scheme* [6, 17, 18, 20, 130]. Recall that, k_4 in Table A.1 represents the iteration number corresponding to an implementation of TACNLP, using the conventional$-\mu$ *updating scheme*.

Data under k_4 are clearly inferior to those under $k_1 - k_3$. A comparison of the data under the headings k_1, k_2 and k_3 shows the superiority of the new$-\mu$ *updating scheme* introduced in this thesis. Judging by the disproportionately high iteration numbers displayed under k_4, TACNLP implemented with the new$-\mu$ *updating scheme* introduced in this thesis, yields far better results than that of the conventional$-\mu$ *updating scheme*. In a few cases, TACNLP implemented with the conventional$-\mu$ *updating scheme* yields better results than TACNLP implemented with the new$-\mu$ *updating scheme*. This is true for six problems from the test set, i.e., 6, 38, 47, 51, 53 and 56. The implementation of the new$-\mu$ *updating scheme* however, resulted in the best overall performance, exhibiting significantly better results in every other problem.

We now illustrate the effect of using the conventional$-\mu$ *updating scheme* implemented with PC3 since this is the most effective *updating criteria*. We plot the progression of μ^k against k, obtained using the conventional$-\mu$ *updating scheme*.

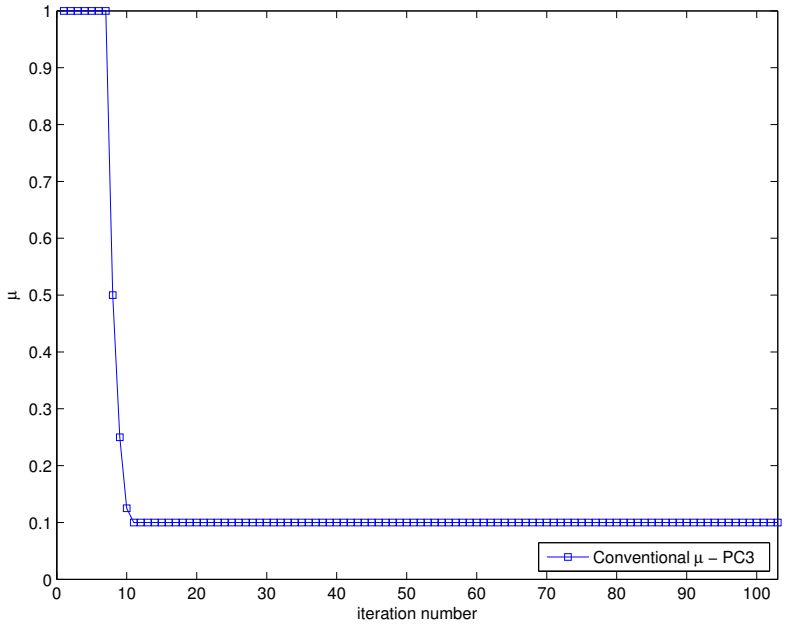


Figure 9.2 The effect of using the conventional updating strategy for μ in TACNLP using Problem 2

Once again we have used problem 2 from the test set for illustrative purposes. This allows us to make a comparison with the plots in Figure 9.1. In Figure 9.2, notice that μ is driven to its lower bound of 0.1, and even so TACNLP only converges in 103 iterations, see Table A.1. This may be the result of placing the bound $\mu_{min} = 0.1$ on μ. However, decreasing μ below μ_{min} resulted in divergence of the algorithm. By comparing the progression of μ in Figure 9.1 to the progressions obtained in Figure 9.2 , we notice huge differences in the amount of iterations taken as well as the range of μ. Here, TACNLP implemented with new$-\mu$, converges using fewer iterations, with μ not much smaller than unity, whereas TACNLP with conventional$-\mu$ converges slower with smaller values of μ.

To further demonstrate the impact of updating μ using the new$-\mu$ *updating scheme*, as opposed to using the conventional$-\mu$ *updating scheme*, we have generated the performance profiles [40] seen in Figure 9.3. Performance profiles are used to illustrate the performance of different algorithms. If one denotes the CPU time taken by algorithm a to solve problem

p by $t_{p,a}$ then the performance ratio is given by

$$r_{p,a} = \frac{t_{p,a}}{\min\{t_{p,a} : a \in A\}},$$

where A is the set of algorithms whose performance we wish to evaluate [40]. A parameter r_M such that $r_M \geq r_{p,a}$ for all $p,\ a$ is chosen, where $r_{p,a} = r_M$ if and only if the algorithm a does not solve problem p [40]. Dolan et al.,[40] defines

$$\rho_a(\tau) = \frac{1}{n_p} size\{p \in P : \log_2(r_{p,a}) \leq \tau\},$$

as the probability for solver $a \in A$ that the log scale of a performance ratio $\log_2(r_{p,a})$ is within a factor τ of the best possible log scaled ratio, where n_p is the number of problems in the set P. The function ρ_a is thus a cumulative distribution function for the performance ratio. We use the $\log_2$ scale to display all the activity that takes place with $\tau \leq r_M$, including the behavior of τ close to 1. The subset of performance ratios which are less than τ, is denoted by $P_{p,a}(\log_2(r_{p,a}) \leq \tau : 1 \leq a \leq n_a)$, where n_a denotes the number of algorithms in the set A. Every performance profile is a plot of $P_{p,a}$ against τ, for each algorithm. The profiles give a clear indication of the relative performance of each algorithm and provide an estimate of the expected performance difference between algorithms.

The profiles generated in this section are based on the CPU times given in Table A.1. In Figure 9.3, new μ update represents the performance of TACNLP implemented with the new$-\mu$ *updating scheme* proposed in this thesis, and conventional μ update represents the performance of TACNLP implemented with the conventional$-\mu$ *updating scheme*. Both implementations are done using PC3. Again, this is done specifically because PC3 yields better results than PC1 and PC2. We have used $\gamma = 0.5$ as the reduction factor for μ^k in the implementation of **Algorithm 6**. Here the number $r_M \approx 2.5$, as indicated on the $x-axis$ in Figure 9.3.

In Figure 9.3, we observe that the implementation of TACNLP with the new$-\mu$ *updating scheme* yields significantly better results than the implementation with the conventional$-\mu$ *updating scheme*. We are interested here in the probability that an algorithm will successfully solve test problems to optimality, faster than any of the algorithms we are testing. This information can be obtained by looking at the height of the performance profile. Judging by the initial height of the performance profiles, we can infer that TACNLP implemented with the new$-\mu$ *updating scheme*, is the fastest algorithm on approximately 77% of the problems. Furthermore, looking at the height at which profile flatlines, we deduce that this implementation of TACNLP solves about 94% of the problems from the test set, to optimality.

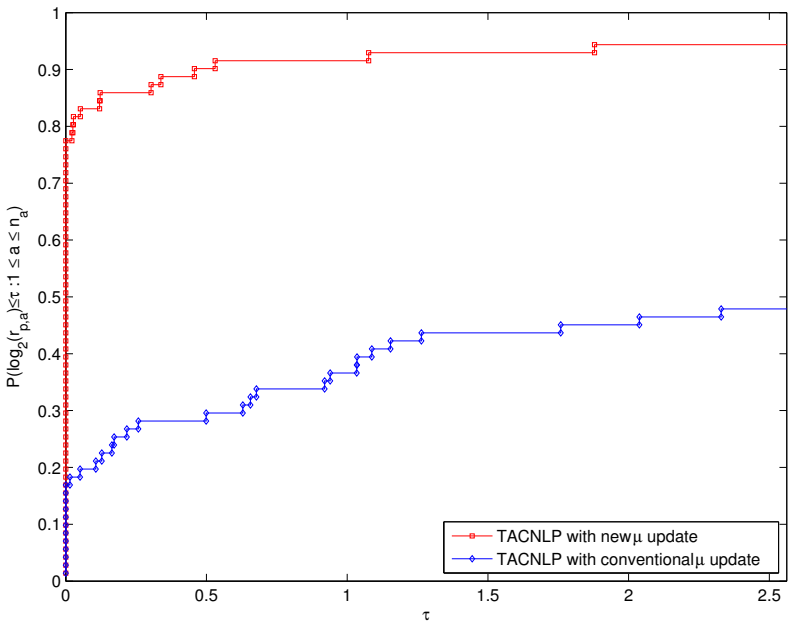


Figure 9.3 Performance profile examining the effectiveness of the new penalty parameter *updating scheme*

On the other hand, TACNLP implemented with the conventional$-\mu$ *updating scheme* is the fastest algorithm on approximately 18% of the problems, solving approximately 48% of the problems from the test set to optimality. This indicates that updating μ using the new$-\mu$ *updating scheme*, i.e., as a solution to (5.10), is far more effective than using the conventional$-\mu$ *updating scheme*.

9.2 Results and discussion for ATAUNLP

In this section, we preset the results for a Matlab implementation of ATAUNLP. As with TACNLP, the numerical results for a MATLAB implementation of ATAUNLP is obtained for a test set of 71 problems. The details of this test set were presented in Subsection 9.1.1. ATAUNLP was implemented with the same parameter values used for TACNLP. The details of these were also presented in Subsection 9.1.1. A discussion of the numerical results for ATAUNLP is now presented in Subsection 9.2.1. Here, we examine the effects using the three different *updating criteria* for μ, i.e., PC1, PC2 and PC3, to determine which of these *updating criteria* is most effective for ATAUNLP. The section is concluded with a comparison of the *updating schemes* for the penalty parameter, i.e., new$-\mu$ and conventional$-\mu$.

9.2.1 Results for ATAUNLP

Tabulated results based on the performance of ATAUNLP are presented in Appendix A, in Tables A.3 - A.5. ATAUNLP was implemented with PC1, PC2 and PC3 to determine which of these criteria yields the best results for ATAUNLP.

In Table A.4, all the columns are defined as in Table A.1. The *updating criteria* PC1, PC2 and PC3 were implemented using the new$-\mu$ *updating scheme* proposed in this thesis. Column 11 in each table gives k_4 which is the iteration number corresponding to an implementation of ATAUNLP using PC3 with the conventional$-\mu$ *updating scheme* [6, 17, 18, 20, 130], described in **Algorithm 6**. We have used PC3 when implementing the conventional$-\mu$ *updating scheme* since its the best *updating criteria* for μ, see Table A.3. Lastly, column 12 in Table A.3 gives the CPU time corresponding to an implementation of ATAUNLP using PC3 with the new$-\mu$ *updating scheme* t_{k_3}. The final row of Table A.4 summarizes the average number of iterations and the average CPU time taken by each implementation of ATAUNLP, to solve a problem.

In Table A.5, the columns are defined as per Table A.2. Each of the implementations, with PC1, PC2 and PC3, is done with the new$-\mu$ *updating scheme*. Finally, column 4 gives the difference between x^* and the solution x^k obtained by the ATAUNLP algorithm, **Algorithm 9**, implemented with the conventional$-\mu$ *updating scheme*. Information on larger CNLP test problems which could not fit into Table A.4, is provided in Table A.3.
ATAUNLP terminates according to the same criteria as TACNLP. This was presented in Section 9.1.2.

We begin by examining the effects of updating the penalty parameter using PC1, PC2 and PC3.

Comparison of *updating criteria* for the penalty parameter

Similar findings were obtained for TACNLP and ATAUNLP, see Tables A.2 and A.4. Here, ATAUNLP implemented with PC3 performed best on 25 of the test problems, while ATAUNLP implemented with PC1 and PC2 performed best on 19 and 21 problems respectively. All other problems either converged within the exact same number of iterations or terminated with $\bar{I}$ or $\bar{D}$ for all three implementations (i.e., PC1, PC2 and PC3).
We plot the progression of μ^k against k, obtained as a solution to (5.10), using PC1, PC2 and PC3. We do so using problem 24 from the test set. The progressions corresponding to PC1,

PC2 and PC3 are respectively represented in Figure 9.1 by ATAUNLP with new μ - PC1, ATAUNLP with new μ - PC2 and ATAUNLP with new μ - PC3.

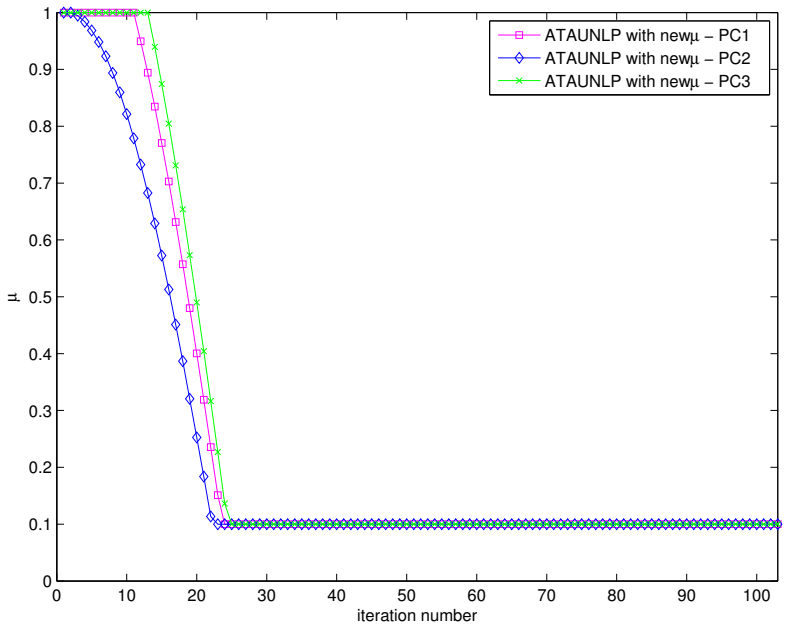


Figure 9.4 Progression of μ^k under PC1, PC2 and PC3 in ATAUNLP

The significance of the number 100 in Figure 9.1 is that ATAUNLP converges in 103 iterations using PC3. TACNLP converges in 125 and 120 iterations using PC1 and PC2 respectively, see Table A.3. Notice that μ attains its lower bound of 0.1 with all three implementations, but yields better convergence with PC3. Notice also that, as with TACNLP, μ is decreased more rapidly when updated using PC2. This is seen to hinder the convergence of some problems. With reference to Tables A.3, we see that this is true for 21% of the test set, i.e., problems 7, 9, 13, 14, 17, 19, 26, 27, 34, 42, 59 and 60 - 62.24. As with the findings for TACNLP, PC3 is the most effective criterion since it yields better convergence than PC1 and PC2. This is the general trend for most of the problems in the test set, see Table A.3.

We now present a discussion around the different *updating schemes* for the penalty parameter, presented in **Algorithms 5** and **6** respectively.

Comparison of *updating schemes* for the penalty parameter

In this section, we examine the effects of updating μ using the two *updating schemes* new$-\mu$ and conventional$-\mu$. We refer to the data in Table A.4. As with TACNLP, the results

corresponding to k_4 are inferior to the results under k_1, k_2 and k_3, corresponding to the new$-\mu$ *updating scheme*. There are a few cases where ATAUNLP implemented with the conventional$-\mu$ *updating scheme* yields better results. This is true for 10 problems from the test set, i.e., 14, 17, 20, 27, 34, 36, 38, 40, 42 and 52. However, the superiority of ATAUNLP implemented using the new$-\mu$ *updating scheme* introduced in this thesis, is established by comparing the entire data set under $k_1 - k_3$ with that of k_4. Here, as with TACNLP, the *new* $-\mu$ *updating scheme* yields the overall best performance.

To assert these findings, we plot the progression of μ^k, updated using the conventional$-\mu$ *updating scheme*, against k. We use PC3 as the *updating criteria* for μ.

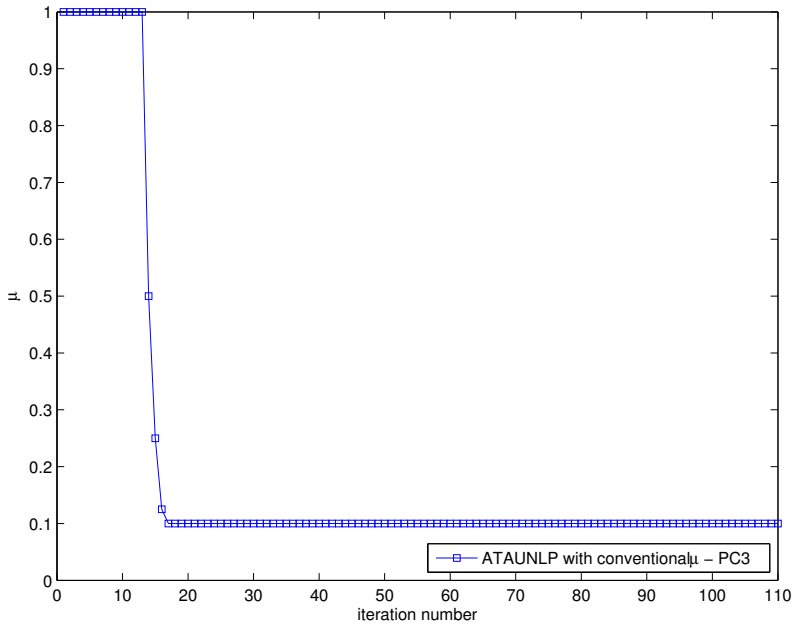


Figure 9.5 Progression of μ^k under PC3, in ATAUNLP using Problem 24

In Figure 9.5, we plot the progression of μ^k using problem 24. This allows us to make a comparison with the plots in Figure 9.4. With reference to Figure 9.5, we see that μ^k is driven to its lower bound within the first 20 iterations. The number 110 in Figure 9.5 represents the iteration number of ATAUNLP using the conventional$-\mu$ *updating scheme*. ATAUNLP implemented using the new$-\mu$ *updating scheme* therefore only performs slightly better, converging in 103 iterations, see Table A.4. By comparing the progression of new$-\mu$ - PC3, in Figure 9.4, with the progression in Figure 9.5, we see that these are almost identical. Even though there are generally no significant discrepancies between performance of ATAUNLP with these two *updating schemes*, ATAUNLP exhibits better overall performance with the scheme proposed in this thesis. Specifically, ATAUNLP implemented with

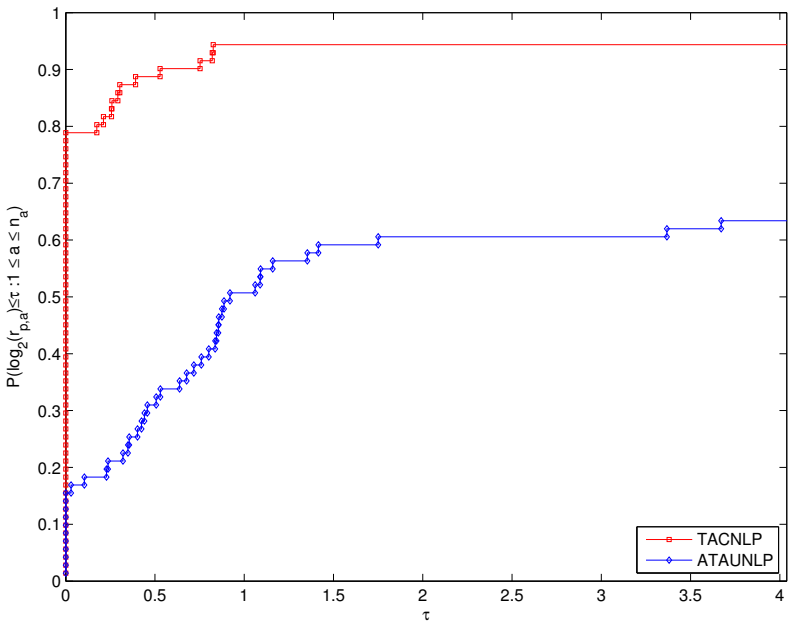


Figure 9.6 Performance profile examining the effectiveness of including the new updates in TACNLP

conventional$-\mu$ performs better on 21 problems, while ATAUNLP implemented with the new$-\mu$ *updating scheme* introduced in this thesis, outperforms on 29 problems, see Table A.4. For every other problem, ATAUNLP either terminated with $\bar{I}$ or $\bar{D}$ for both schemes. At this stage, we may conclude that both TACNLP and ATAUNLP exhibit superior performance implemented with the *updating criterion* PC3, using the new$-\mu$ *updating scheme*.

A comparison of the performance of TACNLP and ATAUNLP, is presented next.

9.3 Comparison of TACNLP and ATAUNLP

We now examine the effectiveness of the adaptive step size routine implemented in TACNLP. To measure the impact of introducing the adaptive step size mechanism in TACNLP, we have generated the performance profiles [40] in Figure 9.6, which compares the performance of TACNLP and ATAUNLP. Both algorithms are implementated with PC3, using the new$-\mu$ updating scheme. The profiles for TACNLP and ATAUNLP are based on the CPU times in Tables A.1 and A.5 respectively. Here the number $r_M \approx 4$, as indicated on the $x-axis$ in Figure 9.6.

The performance profile for TACNLP is significantly higher than that of ATAUNLP, see Figure 9.6. This is inline with the tabulated results in Tables A.1 and A.3, which reveal that TACNLP out-performs ATAUNLP. Judging by the initial height of the performance profiles, we deduce that TACNLP is the fastest algorithm on approximately 80% of the problems. Furthermore, looking at the height at which the profile flatlines, we can conclude that TACNLP solves approximately 94% of the problems from the test set to optimality. ATAUNLP, on the other hand is the fastest algorithm on approximately 17% of the problems, solving approximately 64% of the problems from the test set to optimality. An important observation which can be made from these results is that with the new strategies in place in TACNLP, the iteration numbers have decreased by more than half in most cases when compared to ATAUNLP. In other cases ATAUNLP is seen to diverge or exceed the maximum acceptable iteration number. ATAUNLP is only seen to perform better than TACNLP for problem 29.

Table 9.5 summarizes the overall performance of TACNLP and ATAUNLP. Data under $\bar{I}$, $\bar{D}$ and $\bar{S}$ respectively denote the number of problems which reached the maximum iteration number, the number of problems which diverged, and the number of problems which were solved to optimality.

	TACNLP			*ATAUNLP*		
	$\bar{I}$	$\bar{D}$	$\bar{S}$	$\bar{I}$	$\bar{D}$	$\bar{S}$
PC1 with new$-\mu$ update	7	0	64	24	1	46
PC2 with new$-\mu$ update	13	1	57	25	1	45
PC3 with new$-\mu$ update	5	0	66	24	1	46
PC3 with conventional $-\mu$ update	35	3	33	21	2	48
Total:	60	4	220	94	5	185

Table 9.5 A comparison of the overall performance of TACNLP and ATAUNLP

Table 9.5 also presents the total number of problems which failed to converge within 500 iterations, $\bar{I}$, the total number of problems which diverged, $\bar{D}$, and the total number of problems which were solved, $\bar{S}$, for TACNLP and ATAUNLP respectively. This total is calculated for each penalty parameter updating strategy described in column 1, which consists of the *updating criteria* and *updating scheme* for μ. These separate totals are then tallied up in the last row of Table 9.5, to provide an overview of the general performance of TACNLP and ATAUNLP. TACNLP successfully solves more problems and diverges less frequently than ATAUNLP, when implemented with the new$-\mu$ *updating scheme*. One exception is noticeable between the two when ATAUNLP outperforms TACNLP with

respect to $\bar{S}$ implemented with PC3 and the conventional$-\mu$ *updating scheme*. TACNLP however, is much more superior to ATAUNLP with the new$-\mu$ *updating scheme*. This clearly shows the positive effect of the adaptive set size routine used in TACNLP.

With reference to Tables A.2 and A.3 in Appendix A, we notice that most of the problems which failed to converge within 500 iterations, terminated at solutions x^k close to x^*. Here x^k corresponds to the final k-th iteration. This finding is favorable because it indicates that for a large proportion of the test set TACNLP was in fact able to locate the optimal solution x^*.

We now compare the performance of TACNLP with that of the benchmarking solver SNOPT [56].

9.4 Comparison of TACNLP and SNOPT

To test the performance of TACNLP within the context of existing optimization software, we sum up this section by comparing the performance of TACNLP with that of the sequential quadratic package, SNOPT [56]. We compare the performance of TACNLP implemented with PC3. The performance of SNOPT and TACNLP are tested on the CUTEr test problems, 60 (BT1) - 71 (BT12).

The results in Table 9.6, based on the performance of SNOPT, have been referenced from Gill et al.[59]. The default SNOPT parameters were used throughout and the absolute convergence tolerance for TACNLP has been change to $\varepsilon = 10^{-6}$ to be able to present a fair comparison between TACNLP and SNOPT, since the results for SNOPT were produced using this value. For more information on the implementation of SNOPT, see Gill et al. [57]. In Table 9.6, for each test problem we list the number of variables, n, the number of constraints, m, the number of function evaluations, fe, and the number of iterations, k, used. We also tally up the average amount of iterations taken per solved problem and the average number of function evaluations needed per problem in the last row of Table 9.6. We have not included problems which terminated with $\bar{I}$ in the calculation of these averages, for each algorithm. Following the notational convention used before, a table entry $\bar{I}$ indicates that the algorithm failed to converge within 500 iterations and a table entry $\bar{D}$ indicates that the algorithm diverged. It should be noted that SNOPT is an established package which has been refined over many years, whereas TACNLP is just a straightforward Matlab implementation of **Algorithm 8**.

Of the 12 CUTEr test problems, TACNLP failed to solve four problems within 500 iterations, namely BT4, BT5, BT7 and BT12. In total TACNLP successfully solved 8 problems and

P	(n)	(m)	*SNOPT*		*TACNLP*	
			fe	k	fe	k
60 (BT1)	2	1	21	10	0	93
61 (BT2)	3	1	16	15	0	109
62 (BT3)	5	3	7	6	0	97
63 (BT4)	3	2	10	7	0	$\bar{I}$
64 (BT5)	3	2	11	8	0	$\bar{I}$
65 (BT6)	5	2	16	14	0	143
66 (BT7)	5	3	36	19	0	$\bar{I}$
67 (BT8)	5	2	13	11	0	72
68 (BT9)	4	2	30	18	0	205
69 (BT10)	2	2	23	13	0	200
70 (BT11)	5	3	14	11	0	125
71 (BT12)	5	3	28	19	0	$\bar{I}$
Average:			18	13	0	131

Table 9.6 Results obtained when solving problems from the CUTEr test set using SNOPT and TACNLP

terminated with $\bar{I}$ for 4 problems. SNOPT managed to successfully solve all 12 problems. Furthermore, the average number of iterations required per problem for SNOPT and TACNLP, is $k = 13$ and $k = 131$ respectively. The average number of function evaluations required however, is $f_e = 18$ for SNOPT, and $f_e = 0$ for TACNLP. We hope that the overall results are compensated for by the fact that TACNLP does not use any function evaluations. Another important factor to consider is that SNOPT implements a Quasi-Newton method, using a Broyden (BFGS) class update for the hessian matrix, paired with a line search strategy. TACNLP uses no line search or any sort of second derivative approximations. We therefore expect that TACNLP will spend on average more time on each iteration. Even though the iteration number for TACNLP is on average significantly higher than that of SNOPT, we hope we have convinced the reader that TACNLP redeems itself in terms of overall evaluations (function, first derivative and second derivative) needed to successfully converge to an optimal solution.

9.5 Results for UNLPs

We end off this chapter by comparing the performance of Snymans TAUNLP [115], with our extension of this algorithm for unconstrained problems. We refer to the latter as an extended trajectory-based algorithm for unconstrained nonlinear programming problems (ETAUNLP).

ETAUNLP is an adaptation of TACNLP for the unconstrained case. The major difference between TACNLP and ETAUNLP is that ETAUNLP minimizes the objective function $f(x)$ and not the augmented Lagrangian. ETAUNLP is therefore a much simpler adaptation of TACNLP, with the adaptive step size routine implemented for the integration of x, and the scaling routine implemented wherever necessary, for ∇f. Results based on the performance of TAUNLP and ETAUNLP are presented in Table 9.7. The algorithms were tested on 10 problems, including the 8 test problems used in Snyman [115]. For each algorithm the absolute convergence tolerance was set to 10^{-5}.

P	(n)	*TAUNLP*			*ETAUNLP*		
		k	$\|\|x^*-x\|\|$	time	k	$\|\|x^*-x\|\|$	time
1	2	29	1.44×10^{-6}	0.0164	20	4.65×10^{-6}	0.0137
2	2	23	2.12×10^{-6}	0.0186	2	0	0.0060
3	2	151	7.81×10^{-6}	0.0257	248	6.16×10^{-5}	0.0345
4	2	180	3.7×10^{-3}	0.0332	397	2.11×10^{-2}	0.0591
5	2	371	2.06×10^{-2}	0.0681	$\bar{I}$	1.50×10^{-1}	0.0127
6	2	65	6.32×10^{-6}	0.0243	249	1.73×10^{-6}	0.0267
7	4	463	1.03×10^{-1}	0.0334	111	8.68×10^{-2}	0.0272
8	4	192	3.24×10^{-6}	0.0469	$\bar{I}$	6.03×10^{-6}	0.0746
9	4	180	1.77×10^{-6}	0.0258	66	2.81×10^{-6}	0.0208
10	4	148	1.75×10^{-6}	0.0238	77	1.36×10^{-6}	0.0138
Average:		180		0.0316	146		0.0289

Table 9.7 Comparison of the performance of TAUNLP and ETAUNLP

In Table 9.7, columns 1 through 2 corresponds to the problem P and the dimension of the problem n. Furthermore, for each algorithm the iteration number k, as well as the difference between x^* and the solution x^k obtained by the respective algorithm, are given. Here x^k corresponds to the final k-th iteration. The CPU time for TANULP and ETAUNLP can be found in the column headed "time". The average iteration number and CPU time taken to solve each problem is listed in the final row of Table 9.7. We have not included problems which terminated with $\bar{I}$ in the calculation of these averages, for each algorithm. For the implementation of ETAUNLP, four problems were scaled, i.e., problems 3, 4, 7 and 8. None of the problems in Table 9.7 were scaled in the implementation of TAUNLP. Of the 10 problems, ETAUNLP successfully solved 8 problems and terminated with $\bar{I}$ for 2 problems. TAUNLP successfully solved all 10 problems.

We now illustrate the performance of TAUNLP and ETAUNLP using performance profiles which are based on the CPU time for TAUNLP and ETAUNLP.

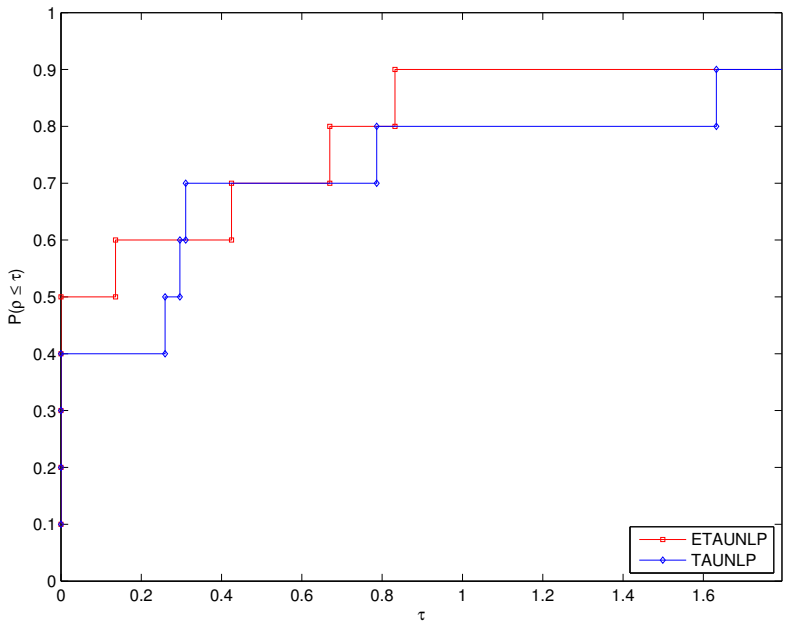


Figure 9.7 Performance profile examining the effectiveness of including the new updates in ETAUNLP

In Figure 9.7, we notice from the initial height of the profile that ETAUNLP is the fastest algorithm on approximately 50% of the problems. The height at which the profile flatlines, also reveals that TACNLP solves approximately 90% of the problems from the test set to optimality. On the other hand TAUNLP is the fastest algorithm on approximately 40% of the problems and solves approximately 90% of the problems from the test set to optimality. Even though both algorithms are competitive, ETAUNLP performs slightly better than Snyman's TAUNLP [115].

We now measure the effect of introducing the scaling mechanism presented in Section 5.2. To do so, we consider problem 7 from Table 9.7, obtained from Snyman [115]:

$$\min \quad f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4. \tag{9.4}$$

By considering the coefficient of 10 in the first and last term, as well as the power of 4 in the last two terms, it is easy to see how this problem may generate iterates which are large. We now implement both TAUNLP and ETAUNLP with scaling, and measure the effects. The results are tabulated in Table 9.8:

Recall that the parameters d_1 used in TAUNLP and $t_{(x,2)}$ used in ETAUNLP are both magnification factors for Δt. Table 9.8 provides data for TAUNLP and ETAUNLP, using different magnification factors for the different step size updates. Each implementation is

	TAUNLP		*ETAUNLP*	
	k	d_1	k	$t_{(x,2)}$
without scaling	463	0.001	$\bar{D}$	1.001
with scaling	236	0.001	354	1.001
without scaling	$\bar{I}$	0.01	$\bar{D}$	1.01
with scaling	123	0.01	111	1.01

Table 9.8 Comparison of the performance of TAUNLP and ETAUNLP with scaling

done firstly without scaling, and then with scaling. In Table 9.8, k corresponds to the iteration number, and d_1 and $(t_{(x,2)})$ are the magnification factors for Δt used in TAUNLP and ETAUNLP respectively.

With reference to Table 9.8, we notice that TAUNLP implemented without scaling solved problem 7 in 463 iterations, while ETAUNLP implemented with scaling solved problem 7 in 111 iterations. Here, the magnification factors for each algorithm were chosen as $d_1 = 0.001$ and $t_{(x,2)} = 1.01$ respectively. Since the parameters d_1 and $t_{(x,2)}$ are both magnification factors for Δt, we can select them to be somewhat comparable. If we choose $d_1 = 0.01$, which is slightly larger than the default value used, i.e., $d_1 = 0.001$ [115], then TAUNLP terminates with $\bar{I}$. If in addition, we implement TAUNLP with scaling, then the algorithm converges to x^* in 123 iterations. With reference to Table 9.8, we notice that this is less than the number of iterations taken using $d_1 = 0.001$, without scaling, i.e., $k = 463$ and with scaling, i.e. $k = 236$. Thus by implementing the scaling mechanism, the choice of the parameter d_1 no longer has to be restricted to very small values, i.e., 0.001 and consequently Δt is magnified more effectively. By implementing the scaling mechanism in ETAUNLP the same problem is solved in 111 iterations, again demonstrating that the scaling mechanism as well as the adaptive step size routine contribute favorably to the performance of the trajectory-based methods.

Chapter 10

Numerical Results of MINLPs

In this chapter, we present numerical results for TAMINLP (presented in Chapter 8). TAMINLP is tested on a set of 24 problems. Details of this test set are presented in Section 10.1. Section 10.2 consists of a description of the numerical study of TAMINLP, and a description of its results.
For comparison, the performance of our TAMINLP algorithm is compared with that of two existing algorithms for convex MINLPs. These are the BB algorithm and the OA algorithm presented in Section 3.4. The numerical results of this study are presented and discussed in Section 10.3.
All tests in this chapter were performed on a PC with an Intel Core i5 CPU at 2.5 GHz with 4GB of 1333 MHz RAM, running OS X 10.8.5. All Algorithms were coded in MATLAB 2013a 64 bit.

10.1 Test problems and parameters

A total of 24 MINLPs and 1 MILP were identified for the numerical experiments in this section. Details of the test set can be found in Table A.6, in Appendix A. The dimensions of the problems range from $n = 2$ to $n = 17$, with the number of constraints ranging from 4 to 57, see Table A.6. Further information on the structure of each problem is summarized below.

With reference to Table 10.1, 16 of the test problems are non-convex, two problems are pseudo convex, five problems are convex and one problem is linear. We have included an MILP to demonstrate that our algorithm is particularly efficient for solving MINLPs.

Problems 18, 23 and 24 are related to the problem of synthesizing a process system [42]. In order to satisfy design specifications, the optimal structural and operating parameters for a

Problems	**Description**
3	Linear
5, 8, 10,13,15	Convex
6, 7	Pseudo-convex
1, 2, 4, 9, 11, 12, 14, 16, 17 - 24	Non-convex

Table 10.1 The structure of the mixed integer problems in the test set

Problems	$t_{(x,2)}$	$t_{(\lambda,2)}$
22	1.5	1.5
16, 17	1.1	2
1, 21, 24	1.1	1.1

Table 10.2 Test problems which did not converge unless the specified parametric values were used as opposed to the default values stipulated in Table 6.3

process must be determined. Here the binary variables are associated with each piece of equipment and the continuous variables represent the process parameters such as the flow rates of material. The constraints are modeled to satisfy design specifications, topological considerations and conservation equations around nodes in the superstructure [42].

TAMINLP was implemented using PC3 with the new$-\mu$ *updating scheme*. For each phase, the absolute convergence tolerance is set to 10^{-3}. For each problem, we set the radius for the initial discrete neighborhood $\mathscr{N}_d(y_d, \varepsilon_d)$, to 1. Whenever the routine for TACNLP was called in TAMINLP, the same parameters were used as with the continuous case. Each phase was initialized using the zero vector of corresponding dimension, for the initial multiplier estimates.

As with the continuous case, certain problems from the test set were unable to converge or converged slowly unless some of the values in Table 6.3 were modified slightly. Test runs were therefore conducted to obtain the optimal values of some parameters. The first test was conducted with TAMINLP, where for each problem a number of values for $t_{(x,2)}$ and $t_{(\lambda,2)}$, within the range of $(1,\ 2)$, were used to determine which of these are optimal. In these experiments the other variables were kept constant. We summarize the problems and the assigned parametric values obtained from the test runs, in Table 10.2 below.

Problems	μ_{min}	μ_{max}
10	0.01	1

Table 10.3 Test problems with specifications for the penalty parameter, which differ from those listed in Table 6.3

With reference to Table 10.2, six test problems exhibited improved convergence using these parameter specifications.

The next test was done to determine optimal values for μ_{max} within the range of $(1,\ 10)$ and μ_{min} within the range of $(0.01,\ 1)$. From the entire test set, only problem 10 was implemented with parametric specifications for μ which differed from the default values listed in Table 6.3. We summarize the details in Table 10.3 below:
Within the solution process of TAMINLP, the scaling mechanism discussed in Subsection 5.2.3 was implemented for problem 20 only, but even so it failed to converge.

10.2 Results and discussion for TAMINLP

Numerical results based on the performance of TAMINLP are presented in Appendix A, in Tables A.6 - A.7. In Table A.6, the first column gives the problem P. Using the same notational convention presented in previous chapters, columns 2 through 3 give the number of continuous variables n_c and the number of discrete variables n_d. Furthermore, column 9 gives $\bar{k}_1$, which correspond to the iteration number of the first phase of TAMINLP. Columns 10 through 11 give $\bar{k}_2$ and $\bar{k}_3$ which give to the iteration number corresponding to the best solution obtained during the second and third phases of TAMINLP respectively. Column 12 gives fe which indicates how many function evaluations were used within the entire algorithm and column 13 gives ε_d, which indicates the radius of the discrete neighborhood $\mathcal{N}_d(y_d, \varepsilon_d)$. Column 14 indicates if $\bar{y}$ is feasible or not and column 15 gives n_p which indicates how many trial points were explored. Lastly, column 16 gives ref which provides the source of each problem. The other symbols are defined as before, while the table entry "$-$" means that information could not be obtained for that particular problem.

The table entry ✓ means $\bar{y}$ is feasible, while the table entry ✗ means $\bar{y}$ is infeasible. Lastly, the table entry ✓* means that $\bar{y}$ is feasible, but no feasible x_i^* is found on the corresponding feasible-continuous manifold.

The tabulated results in Table A.6 indicate that 23 of the 24 test problems converged to feasible solutions. This was true even when the optimality conditions were not satisfied within 500 iterations. For instance, problems 1 converged to the optimal point $z^* = (x^*, y^*)^T$, even though the algorithm terminated with $\bar{I}$, see Table A.6. Furthermore, for problems 1, 5 and 11, the integer constituent of the solution converged to an integer feasible solution after the first phase of TAMINLP. Activation of the second and third phases were therefore not necessary.

Following the discussion from the first example in Section 8.7, we recall that the closest integer-feasible point to the continuous optimum y_c^*, is in general the solution to $\mathscr{M}$, provided y_c^* has been rounded accurately. An example was used to illustrate this at the beginning of Section 8.7. Therefore, provided $\bar{y}$ is feasible (and has not been affected by rounding error) and a corresponding feasible solution x_i^* exists, the point $(x_i^*, \bar{y})^T$ is the optimal solution to the MINLP $\mathscr{M}$. Since $\bar{y}$, which has been accurately rounded, is feasible for problems 2 - 4, 8 - 10, 13 and 22- 24, and the corresponding feasible x_i^* exists, the points $(x_i^*, \bar{y})^T$ are optimal for these problems, i.e., $z^* = (x_i^*, \bar{y})^T$, see Table A.6.
For problem 13, $\bar{y}$ is feasible, and $x_c^* = x^*$. The second phase of TAMINLP therefore terminates upon initialization because x_c^* is the continuous optimal solution corresponding to each $\bar{y}^i$.
For problems 12, 14, 16, 18 and 19, even though $\bar{y}$ is feasible, no corresponding feasible x exists. For these problems $\bar{y}$ is therefore not optimal.
For problems 6, 7, 15, 17 and 21, $\bar{y}$ is infeasible and is therefore not optimal.
For both of the latter instances, the optimal solution corresponds to one of the other feasible trial points generated in the neighborhood $\mathscr{N}_d(\bar{y}, \varepsilon_d)$. Recall that, if all of the trial points generated are infeasible in x, then the neighborhood $\mathscr{N}_d(\bar{y}, \varepsilon_d)$ is increased until at least one feasible point $\bar{y}^i$ and corresponding x_i^* is obtained.

An important observation which was made is that most problems which terminated with the output $\bar{I}$ in the first phase of TAMINLP still converged to an optimal solution. This is true for problems 2, 9, 10, 14 - 17, 19 and 21 - 24. The only problem which did not converge is problem 20, which diverged. Furthermore, TAMINLP was able to locate the solution to problems 6 and 8, which are pseudo-convex and convex respectively, faster than it was able to locate a solution to problem 3, the MILP, see Table A.6. This indicates that TAMINLP is better structured for solving MINLPs.

We now compare the performance of TAMINLP with that of BB and OA.

10.3 Comparison of MINLP algorithms

In this section, we use a small convex MINLP, to compare TAMINLP with some of the existing algorithms for convex MINLPs, discussed in Section 3.4. Specifically, we compare TAMINLP with BB and OA. The following convex MINLP example is taken from [93]:

$$\mathscr{M}\begin{cases}\min\limits_{x,y_1,y_2} & x^2+(y_1-\frac{1}{3})^2+(y_2-\frac{1}{4})^2,\\ \text{st} & 2y_1-2y_2+1-x\geq 0,\\ & x\in[-5,\ 5],\ y\in\{0,\ 1\}^2.\end{cases} \tag{10.1}$$

We begin by illustrating BB using the MINLP in (10.1). This worked solution, was adapted from [93]. BB starts by solving the relaxed CNLP $\underline{\mathscr{M}}$:

$$\underline{\mathscr{M}}\begin{cases}\min\limits_{x,y} & x^2+(y_1-\frac{1}{3})^2+(y_2-\frac{1}{4})^2,\\ \text{st} & 2y_1-2y_2+1-x\geq 0,\\ & x\in[-5,\ 5],\ y\in[0,\ 1]^2,\end{cases}$$

and obtains the solution $(x,y)^T=(0,\frac{1}{3},\frac{1}{4})^T$. This is not an integer feasible solution, so using the branching rule of choosing the most fractional integer variable, the variable $y_2=\frac{1}{4}$ is chosen for branching. This results in two subproblems:

$$\underline{\mathscr{M}}_1\begin{cases}\min\limits_{x,y} & x^2+(y_1-\frac{1}{3})^2+(y_2-\frac{1}{4})^2,\\ \text{st} & 2y_1-2y_2+1-x\geq 0,\\ & y_2-1\geq 0,\\ & x\in[-5,\ 5],\ y\in[0,\ 1]^2.\end{cases}$$

and

$$\underline{\mathscr{M}}_2\begin{cases}\min\limits_{x,y} & x^2+(y_1-\frac{1}{3})^2+(y_2-\frac{1}{4})^2,\\ \text{st} & 2y_1-2y_2+1-x\geq 0,\\ & y_2\leq 0,\\ & x\in[-5,\ 5],\ y\in[0,\ 1]^2.\end{cases}$$

Subproblem $\underline{\mathscr{M}}_1$ has optimal solution $(x,y)^T=(-\frac{1}{15},\frac{7}{15},1)^T$ with $f_1=f(x,y)=\frac{405}{720}$.

The solution to subproblem $\underline{\mathscr{M}}_2$ is $(x,y)^T = (0,\frac{1}{3},0)^T$, with $f_2 = f(x,y) = \frac{1}{16}$. The node corresponding to $\underline{\mathscr{M}}_2$ is the most promising since $f_2 < f_1$. The subproblems $\underline{\mathscr{M}}_{21}$ and $\underline{\mathscr{M}}_{22}$ are therefore solved, by branching on the non-integer variable $\frac{1}{3}$:

$$\underline{\mathscr{M}}_{21} \begin{cases} \min\limits_{x,y} & x^2 + (y_1 - \frac{1}{3})^2 + (y_2 - \frac{1}{4})^2, \\ \text{st} & 2y_1 - 2y_2 + 1 - x \geq 0, \\ & y_2 \leq 0, \\ & y_1 - 1 \geq 0, \\ & x \in [-5,\ 5],\ y \in [0,\ 1]^2. \end{cases}$$

and

$$\underline{\mathscr{M}}_{22} \begin{cases} \min\limits_{x,y} & x^2 + (y_1 - \frac{1}{3})^2 + (y_2 - \frac{1}{4})^2, \\ \text{st} & 2y_1 - 2y_2 + 1 - x \geq 0, \\ & y_2 \leq 0, \\ & y_1 \leq 0, \\ & x \in [-5,\ 5],\ y \in [0,\ 1]^2. \end{cases}$$

Subproblem $\underline{\mathscr{M}}_{21}$ has optimal solution $(x,y)^T = (0,1,0)^T$ with optimal function value $f_{21} = f(x,y) = \frac{73}{144}$. The point $(x,y)^T = (0,1,0)^T$ is integer-feasible and since $f_{21} < f_1$, the node corresponding to $\underline{\mathscr{M}}_1$ is pruned due to dominated upper bound. We store the solution, $(x,y)^T = (0,1,0)^T$ and proceed to solve $\underline{\mathscr{M}}_{22}$. Subproblem $\underline{\mathscr{M}}_{22}$ has the integer-feasible solution $(x,y)^T = (0,0,0)^T$, with optimal function value $f_{22} = f(x,y) = \frac{25}{144}$. Since there are no more nodes to explore in the BB tree and $f_{22} < f_{21}$, the BB optimal solution to the problem (10.1) is $(x^*,y^*)^T = (0,0,0)^T$.

In total, five CNLPs were solved before the BB optimal solution $(x^*,y^*)^T = (0,0,0)^T$ was located.

We now illustrate OA, using problem (10.1)

OA is initialized at the feasible point $y^0 = (0,\ 1)^T$, with $T^{-1} = \emptyset$, $S^{-1} = \emptyset$ and $\text{UBD}^{-1} = \infty$. Since y^0 is feasible, $\bar{\mathscr{M}}(y^0)$ is solved:

$$\bar{\mathscr{M}}(y^0) \begin{cases} \min\limits_{x} & x^2 + \frac{97}{144}, \\ \text{st} & -x - 1 \geq 0, \\ & x \in [-5,\ 5], \end{cases}$$

which has solution $x^0 = -1$. The MILP M-OA^0 is then solved:

$$\text{M-OA}^0 \left\{ \begin{array}{ll} \min\limits_{\alpha} & \alpha, \\ \text{st} & \alpha < UBD^0 = \frac{241}{144}, \\ & -2x - \frac{2}{3}y_1 + \frac{3}{2}y_2 - \frac{263}{144} \leq \alpha, \\ & 2y_1 - 2y_2 + 1 - x \geq 0, \\ & x \in [-5,\ 5],\ y \in \{0,1\}^2, \end{array} \right.$$

to obtain the solution $(x,y)^T = (3,1,0)^T$, and $\alpha = \frac{-1223}{144}$. We then use the feasible point $y^1 = (1,0)^T$ to solve $\bar{\mathscr{M}}(y^1)$ and obtain x^1 =0, and $f^1 = \frac{73}{144} < f^0 = \frac{241}{144}$. M-OA^1 is then solved:

$$\text{M-OA}^1 \left\{ \begin{array}{ll} \min\limits_{\alpha} & \alpha, \\ \text{st} & \alpha < UBD^1 = \frac{73}{144}, \\ & -2x - \frac{2}{3}y_1 + \frac{3}{2}y_2 - \frac{263}{144} \leq \alpha, \\ & \frac{4}{3}y_1 - \frac{1}{2}y_2 - \frac{119}{144} \leq \alpha, \\ & 2y_1 - 2y_2 + 1 - x \geq 0, \\ & x \in [-5,\ 5],\ y \in \{0,1\}^2, \end{array} \right.$$

to obtain the solution $(x,y)^T = (\frac{-1}{2},0,0)^T$ and $\alpha = \frac{-119}{144}$. We choose $y^2 = (0,0)^T$ as the next integer assignment. The point y^2 is feasible, so $\bar{\mathscr{M}}(y^2)$ is solved to obtain $x^2 = 0$, and $f^2 = \frac{25}{144} < f^1$. We then solve M-OA^2:

$$\text{M-OA}^2 \left\{ \begin{array}{ll} \min\limits_{\alpha} & \alpha, \\ \text{st} & \alpha < UBD^2 = \frac{25}{144}, \\ & -2x - \frac{2}{3}y_1 + \frac{3}{2}y_2 - \frac{263}{144} \leq \alpha, \\ & \frac{4}{3}y_1 - \frac{1}{2}y_2 - \frac{119}{144} \leq \alpha, \\ & -\frac{2}{3}y_1 - \frac{1}{2}y_2 - \frac{25}{144} \leq \alpha, \\ & 2y_1 - 2y_2 + 1 - x \geq 0, \\ & x \in [-5,\ 5],\ y \in \{0,1\}^2, \end{array} \right.$$

to obtain the solution $(x,y)^T = (\frac{-1}{2},1,1)^T$, and $\alpha = \frac{1}{144}$. The point $y^3 = (1,1)^T$ is chosen as the next integer assignment, which happens to be feasible. $\bar{\mathscr{M}}(y^3)$ is the solved and the solutions $x^3 = 0$ and $f^3 = \frac{145}{144} > UBD^2$ are obtained. The next MILP, M-OA^3 is then solved:

$$\text{M-OA}^3 \begin{cases} \min\limits_{\alpha} & \alpha, \\ \text{st} & \alpha < UBD^0 = \frac{263}{144}, \\ & -2x - \frac{2}{3}y_1 + \frac{3}{2}y_2 - \frac{263}{144} \leq \alpha, \\ & \frac{4}{3}y_1 - \frac{1}{2}y_2 - \frac{119}{144} \leq \alpha, \\ & -\frac{2}{3}y_1 - \frac{1}{2}y_2 - \frac{25}{144} \leq \alpha, \\ & \frac{4}{3}y_1 + \frac{3}{2}y_2 - \frac{263}{144} \leq \alpha, \\ & 2y_1 - 2y_2 + 1 - x \geq 0, \\ & x \in [-5,\ 5],\ y \in \{0,1\}^2, \end{cases}$$

which is infeasible, so the algorithm stops. The optimal solution is thus found to be $(x^*, y^*)^T = (0,0,0)^T$, corresponding to the solution of $\bar{\mathscr{M}}(y^2)$.
A total of four CNLPs and four MINLPs were solved before the OA optimal solution $(x,y)^T = (0,0,0)^T$ was located.

We now use TAMINLP to solve the problem (10.1).

TAMINLP starts by solving the relaxed problem $\underline{\mathscr{M}}$:

$$\underline{\mathscr{M}} \begin{cases} \min\limits_{x,y} & x^2 + (y_1 - \frac{1}{3})^2 + (y_2 - \frac{1}{4})^2, \\ \text{st} & 2y_1 - 2y_2 + 1 - x \geq 0, \\ & x \in \mathbb{R},\ y \in \mathbb{R}, \end{cases}$$

and obtains the solution $(x_c^*, y_c^*)^T = (0, \frac{1}{3}, \frac{1}{4})^T$. Since y_c^* is non-integer, it is rounded to the nearest integer point $\bar{y}^0 = (0,0)^T$. The set of integer-feasible trial points, $\bar{y}^i,\ i = 1,2.$, which make up the columns of I_1, are then generated:

$$I_1 = \begin{Bmatrix} 1 & 0 \\ 0 & 1 \end{Bmatrix}. \tag{10.2}$$

Including $\bar{y}^0$, the set of integer-feasible points from which the second phase of TAMINLP is initialized, make up the columns of I_2:

$$I_2 = \begin{Bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix}. \tag{10.3}$$

Three separate minimizations are then performed during the second phase of TAMINLP by solving $\bar{\mathscr{M}}(y^i),\ i = 0,\ldots 2$:

$$\bar{\mathscr{M}}(y^0)\begin{cases}\min\limits_x & x^2+\frac{25}{144},\\ \text{st} & x\in[-5,\ 5],\end{cases}$$

$$\bar{\mathscr{M}}(y^1)\begin{cases}\min\limits_x & x^2+\frac{73}{144},\\ \text{st} & x\in[-5,\ 5],\end{cases}$$

$$\bar{\mathscr{M}}(y^2)\begin{cases}\min\limits_x & x^2+\frac{97}{144},\\ \text{st} & x\in[-5,\ 5].\end{cases}$$

The second phase of TAMINLP terminates upon initialization for each $\bar{y}^i$, since the points $(x,y)^T=(0,0,0)^T$, $(x,y)^T=(0,1,0)^T$ and $(x,y)^T=(0,0,1)^T$ are all optimal solutions on their respective manifolds.

The third phase of TAMINLP is initialized from $(x,y)^T=(0,0,0)^T$ and $(x,y)^T=(0,1,0)^T$, as these points lie within the neighborhood (8.32). The point $(x^*,y^*)^T=(0,0,0)^T$ is then confirmed as the TAMINLP optimal solution.

A total of three CNLPs were solved (since the algorithm terminated upon initialization of the second phase) to locate the TAMINLP optimal solution $(x^*,y^*)^T=(0,0,0)^T$. This is less than the amount taken by both BB and OA. Furthermore, no separate feasibility problem needed to be solved since each phase of TAMINLP terminated upon convergence to a feasible solution. Here, it is the roll of the penalty term in AL (8.8), used in TAMINLP:

$$\frac{1}{2\mu}\sum_i c_i^2(x,y),$$

which ensures that infeasible iterates are driven to feasibility.

In conclusion, TAMINLP is seen to perform significantly better than the convex MINLP solvers BB and OA.

In an attempt to compare our algorithm with existing algorithms for general MINLPs [14], [45], [94],[113], [131], we found that they were not comparable. These methods use the number of function calls and CPU times as a criteria for comparison. Our method however, uses a very negligible number of function calls and CPU times are much lower than those reported in recent papers. Hence the comparison will be favorable for our algorithm.

Chapter 11

Conclusion

The trajectory-based methods for CNLPs and MINLPs have been introduced in this thesis. Although the trajectory-based method for UNLPs, TAUNLP [115] exists, the methods for CNLPs and MINLPs developed in this thesis, are new. The method for CNLPs, TACNLP is an extension of TAUNLP, while the method for MINLPs, TAMINLP, is a adaptation of TACNLP. New strategies have been introduced to improve on the existing framework of TAUNLP. These include the integration of an adaptive step size routine into the framework of the trajectory-based algorithms and a new approach for updating the penalty parameter μ. A new technique for scaling badly scaled problems is also introduced. Computational results have been given showing the effectiveness of the contributions made in this thesis.

Some of the strengths of the methods developed in this thesis include the fact that they are first-order methods, with no function evaluations used for TACNLP, and minimal function evaluations used for TAMINLP. Unlike the MINLP methods presented in Chapter 3, no separate infeasibility problem needs to be solved in TAMINLP. Theoretical convergence is also given for both TACNLP and TAMINLP. Furthermore, TAMINLP can solve MILPs, convex MINLPs (both convex MIQPs and general convex MINLPs) and general MINLPs (for which no credible algorithm exists).

11.1 Summary

Chapter 2 contains a review of CNLPs. The review is three-fold. In the first part, we present some theory pertaining to general CNLPs. This review is important since it allows us to establish optimality conditions for the existence of a local solution to the methods developed for CNLPs in this thesis. In the second part, we present a review of constraints handling in constrained nonlinear programming. Various penalty functions are discussed. This review is motivated by the fact that the methods developed in this thesis use the augmented

Lagrangian penalty function. In the third part, we present a review of the existing methods for solving CNLPs. This review provides us a basis for comparison with the methods for CNLPs presented in this thesis.

In Chapter 3, we present a review of MINLPs. The review is divided into three parts. In the first part, we present an overview of general MINLPs. The second part provides details of important CNLP and MILP subproblems used within the solution process of existing methods for MINLPs. This is followed by a review of these existing methods for MINLPs. The methods for MINLPs are divided into two classes in the review; convex MINLPs and non-convex MINLPs with a specific structure, i.e. pseudo-convex MINLPs. These methods are based on one or a combination of CNLP subproblems presented in this second part of this chapter. The detail of the review is motivated by the fact that we have developed an adaptation of TACNLP for general MINLPs. This adaptation for MINLPs, TAMINLP, presented in Chapter 8, is designed to solve general MINLPs and uses a combination of two of the CNLP subproblems presented in this chapter.

In Chapter 4, we present a review of Snyman's trajectory-based method for UNLPs, TAUNLP [115]. This is a first order method, which uses no objective function value evaluations, no line search techniques and no second-order derivative information throughout the solution process. TAUNLP has seen success since its inception and has proven to be competitive with some methods for unconstrained optimization [115]. The detail of the review is motivated by the fact that all the methods developed in this thesis, are extensions of TAUNLP.

Chapter 5 contains the detail of our trajectory-based algorithm for CNLPs, TACNLP. Some fundamental changes are made in an attempt to improve on the existing framework of the method proposed by Snyman [115]. This includes incorporating an adaptive step size routine into TACNLP. A new technique for updating the penalty parameter μ associated with the augmented Lagrangian (2.30) is also introduced, and a new scheme for scaling problems is proposed. As with Snyman's [115] TAUNLP, no function evaluations, no second-order derivative information and no line search techniques are used within the framework of TACNLP.
Constrained optimization is a mature area of research, but very little literature exists on trajectory-based methods for CNLPs. This has motivated the development of TACNLP for general CNLPs.

In Chapter 6, we present a direct adaptation of TAUNLP [115] for CNLPs, ATAUNLP. Details of the implementation of TACNLP and ATAUNLP are presented and compared. This comparison is done to motivate the new features introduced in TACNLP, i.e. the adaptive step size routine. We establish that the number of parameters used in TACNLP is less than that of ATAUNLP. The performance of TACNLP as well as the performance of ATAUNLP are presented in Chapter 9.

In Chapter 7, local and global convergence properties of TACNLP are established. Global convergence as well as local convergence for the method are guaranteed.

Chapter 8 is divided into three parts. In the first part, we review some important definitions pertaining to the local solution of an MINLP. This includes the definition of a local minimizer. We later prove that Newby's definition of a local minimizer [90] is not ideal. The review is important nonetheless because it aids us in developing our own criteria for detecting whether a solution is an optimum.
Following the review, in the second part, we present the detail of our three-fold TAMINLP algorithm. TAMINLP is an adaptation of TACNLP, and relies on two of the CNLP subproblems discussed in Chapter 3. During each phase of TAMINLP solutions to one of these subproblems is sought.
In the final part of this chapter, convergence properties of TAMINLP are established.
Unlike TAUNLP and TACNLP, TAMINLP uses a few function evaluations during the solution process. This is done to determine which of the solutions in the second and third phases are the best. As with TAUNLP and TACNLP, TAMINLP does not require any second-order derivative information or line search techniques, for its convergence.
No trajectory-based methods for MINLPs exist in the literature. This has motivated the development of TAMINLP for general MINLPs..

In Chapter 9 computational results are presented illustrating the effectiveness of the methods developed in Chapters 5 and 6. TACNLP and ATAUNLP are tested on a set of 71 CNLP problems. From the results, we conclude that TACNLP is efficient for solving small scale CNLP problems. Furthermore Snyman's TAUNLP [115] is compared with our extension thereof for UNLPs, ETAUNLP. With the new strategies in place, the methods developed in this thesis are comparable with the existing unconstrained trajectory-based method [115] and out-perform the constrained adaptation thereof.

Computational results are presented in Chapter 10, showing the effectiveness of the MINLP trajectory-based method, TAMINLP, developed in Chapter 8. TAMINLP is tested on 23 MINLP problems. The results obtained for TAMINLP assert that the improvements we proposed in Chapter 8, for the definition of a mixed minimizer, are useful. TAMINLP is compared with BB and OA and proves to exhibits superior performance over these convex MINLP methods.

11.2 Future Work

The main areas of improvement in this thesis are outlined below:

- The research presented in this thesis is new and can therefore be developed to solve optimization problems of other types such as complementarity problems.
- Mathematical theories based on the method can be further developed both for CNLPs and MINLPs.
- The trajectory-based algorithm can be developed further to solve large scale CNLPs and MINLPs.

Our future research will certainly continue in these areas of development.

Bibliography

[1] Abramson, M. A., Audet, C., Chrissis, J. W., and Walston, J. G. (2009). Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters*, 3(1):35–47.

[2] Adjiman, C. S., Androulakis, I. P., and Floudas, C. A. (2000). Global optimization of mixed-integer nonlinear problems. *AIChE Journal*, 46(9):1769–1797.

[3] Aluffi-Pentini, F., Parisi, V., and Zirilli, F. (1984). A differential-equations algorithm for nonlinear equations. *ACM Transactions on Mathematical Software (TOMS)*, 10(3):299–316.

[4] Alvarez, F. (2000). On the minimizing property of a second order dissipative system in hilbert spaces. *SIAM Journal on Control and Optimization*, 38(4):1102–1119.

[5] Alvarez, F. and Cabot, A. (2004). Steepest descent with curvature dynamical system. *Journal of optimization theory and applications*, 120(2):247–273.

[6] Andreani, R., Birgin, E. G., Martínez, J. M., and Schuverdt, M. L. (2007). On augmented lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization*, 18(4):1286–1309.

[7] Andreani, R., Birgin, E. G., Martínez, J. M., and Schuverdt, M. L. (2008). Augmented lagrangian methods under the constant positive linear dependence constraint qualification. *Mathematical Programming*, 111(1-2):5–32.

[8] Antipin, A. S. (1994). Minimization of convex functions on convex sets by means of differential equations. *Differential Equations*, 30(9):1365–1375.

[9] Attouch, H. and Alvarez, F. (2000). *The heavy ball with friction dynamical system for convex constrained minimization problems*. Springer.

[10] Attouch, H. and Cominetti, R. (1996). A dynamical approach to convex minimization coupling approximation with the steepest descent method. *Journal of Differential Equations*, 128(2):519–540.

[11] Attouch, H., Goudou, X., and Redont, P. (2000). The heavy ball with friction method, i. the continuous dynamical system: global exploration of the local minima of a real-valued function by asymptotic analysis of a dissipative dynamical system. *Communications in Contemporary Mathematics*, 2(01):1–34.

[12] Audet, C. and Dennis Jr, J. (2001). Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, 11(3):573–594.

[13] Bartholomew-Biggs, M. (2008). *Nonlinear optimization with engineering applications*, volume 19. Springer.

[14] Belotti, P., Lee, J., Liberti, L., Margot, F., and Wächter, A. (2009). Branching and bounds tightening techniques for non-convex minlp. *Optimization Methods & Software*, 24(4-5):597–634.

[15] Bertsekas, D. P. (1982). Constrained optimization and lagrange multiplier methods. *Computer Science and Applied Mathematics, Boston: Academic Press, 1982*, 1.

[16] Bertsekas, D. P. (1999). Nonlinear programming.

[17] Birgin, E. G., Castillo, R., and Martínez, J. M. (2005). Numerical comparison of augmented lagrangian algorithms for nonconvex problems. *Computational Optimization and Applications*, 31(1):31–55.

[18] Birgin, E. G., Fernández, D., and Martínez, J. M. (2012). The boundedness of penalty parameters in an augmented lagrangian method with constrained subproblems. *Optimization Methods and Software*, 27(6):1001–1024.

[19] Birgin, E. G. and Martínez, J. M. (2008). Structured minimal-memory inexact quasi-newton method and secant preconditioners for augmented lagrangian optimization. *Computational Optimization and Applications*, 39(1):1–16.

[20] Birgin, E. G. and Martínez, J. M. (2012). Augmented lagrangian method with nonmonotone penalty parameters for constrained optimization. *Computational Optimization and Applications*, 51(3):941–965.

[21] Björk, K.-M. (2002). *A global optimization method with some heat exchanger network applications*. PhD thesis, Department of Chemical Engineering, Abo Akademi University.

[22] Björkqvist, J. (2001). *Discrete and disjunctive optimization: Parallel strategies and applications in industrial scheduling*. PhD thesis, Department of Chemical Engineering, Abo Akademi University.

[23] Boggs, P. T. and Tolle, J. W. (1989). A strategy for global convergence in a sequential quadratic programming algorithm. *SIAM Journal on Numerical Analysis*, 26(3):600–623.

[24] Boggs, P. T. and Tolle, J. W. (2000). Sequential quadratic programming for large-scale nonlinear optimization. *Journal of Computational and Applied Mathematics*, 124(1):123–137.

[25] Boyd, S. and Vandenberghe, L. (2009). *Convex optimization*. Cambridge university press.

[26] Brown, A. and Bartholomew-Biggs, M. C. (1989). Some effective methods for unconstrained optimization based on the solution of systems of ordinary differential equations. *Journal of Optimization Theory and Applications*, 62(2):211–224.

[27] Buchheim, C. and Trieu, L. (2013). Quadratic outer approximation for convex integer programming with box constraints. In *SEA*, pages 224–235. Springer.

[28] Burachik, R. S. and Kaya, C. Y. (2012). An augmented penalty function method with penalty parameter updates for nonconvex optimization. *Nonlinear Analysis: Theory, Methods & Applications*, 75(3):1158–1167.

[29] Butcher, J. (1986). Optimal order and stepsize sequences. *IMA journal of numerical analysis*, 6(4):433–438.

[30] Butcher, J. (1990). Order, stepsize and stiffness switching. *Computing*, 44(3):209–220.

[31] Cardoso, M. F., Salcedo, R., de Azevedo, S. F., and Barbosa, D. (1997). A simulated annealing approach to the solution of minlp problems. *Computers & chemical engineering*, 21(12):1349–1364.

[32] Cominetti, R. and Courdurier, M. (2002). Coupling general penalty schemes for convex programming with the steepest descent and the proximal point algorithm. *SIAM Journal on Optimization*, 13(3):745–765.

[33] Conn, A. R., Gould, G., and Toint, P. L. (2010). *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*. Springer Publishing Company, Incorporated.

[34] Cornuejols, G. and Tutuncu, R. (2007). *Optimization methods in Finance*. Cambridge University Press.

[35] Courant, R. et al. (1943). Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Amer. Math. Soc*, 49(1):1–23.

[36] Dakin, R. J. (1965). A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255.

[37] Datta, B. and Harikrishna, V. (2005). Optimization applications in water resources systems engineering. *Indian Institute of Technology, Kanpur*.

[38] Dennis Jr, J. E. and Schnabel, R. B. (1996). *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. Siam.

[39] Diener, I. (1995). Trajectory methods in global optimization. In *Handbook of Global optimization*, pages 649–668. Springer.

[40] Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213.

[41] Duran, M. A. and Grossmann, I. E. (1986a). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical programming*, 36(3):307–339.

[42] Duran, M. A. and Grossmann, I. E. (1986b). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical programming*, 36(3):307–339.

[43] Emet, S. (2004). *A comparative study of solving some nonconvex MINLP problems*. PhD thesis, Department of Chemical Engineering, Abo Akademi University.

[44] Escudero, L. (1988). More test examples for nonlinear programming codes: Klaus schittkowski volume 282 in: Lecture notes in economics and mathematical systems, springer, berlin, 1987, iv+ 261 pages, dm49. 00.

[45] Exler, O. and Schittkowski, K. (2007). A trust region sqp algorithm for mixed-integer nonlinear programming. *Optimization Letters*, 1(3):269–280.

[46] Fabian, M. J., Henrion, R., Kruger, A. Y., and Outrata, J. V. (2010). Error bounds: necessary and sufficient conditions. *Set-Valued and Variational Analysis*, 18(2):121–149.

[47] Fiacco, A. V. and McCormick, G. P. (1990). *Nonlinear programming: sequential unconstrained minimization techniques*, volume 4. Siam.

[48] Fischer, A. (2002). Local behavior of an iterative framework for generalized equations with nonisolated solutions. *Mathematical Programming*, 94(1):91–124.

[49] Fletcher, R. and Leyffer, S. (1994). Solving mixed integer nonlinear programs by outer approximation. *Mathematical programming*, 66(1-3):327–349.

[50] Floudas, C. A. (1995). *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press.

[51] Floudas, C. A., Pardalos, P. M., Adjiman, C., Esposito, W. R., Gümüs, Z. H., Harding, S. T., Klepeis, J. L., Meyer, C. A., and Schweiger, C. A. (2013). *Handbook of test problems in local and global optimization*, volume 33. Springer Science & Business Media.

[52] Forsgren, A., Gill, P. E., and Wright, M. H. (2002). Interior methods for nonlinear optimization. *SIAM review*, 44(4):525–597.

[53] Frisch, K. (1955). The logarithmic potential method of convex programming. *Memorandum, University Institute of Economics, Oslo*.

[54] Gabere, M. N. (2007). Simulated annealing driven pattern search algorithms for global optimization.

[55] Geoffrion, A. M. (1972). Generalized benders decomposition. *Journal of optimization theory and applications*, 10(4):237–260.

[56] Gill, P. E., Murray, W., and Saunders, M. A. (2002). Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM journal on optimization*, 12(4):979–1006.

[57] Gill, P. E., Murray, W., and Saunders, M. A. (2006). Users guide for snopt version 7: Software for large-scale nonlinear programming.

[58] Gill, P. E., Murray, W., and Wright, M. H. (1981). *Practical optimization*. Academic press.

[59] Gill, P. E. and Robinson, D. P. (2012). A primal-dual augmented lagrangian. *Computational Optimization and Applications*, 51(1):1–25.

[60] Gould, N. (2006). An introduction to algorithms for continuous optimization.

[61] Gould, N. I. and Toint, P. L. (2000). *SQP methods for large-scale nonlinear programming*. Springer.

[62] Griewank, A. O. (1981). Generalized descent for global optimization. *Journal of optimization theory and applications*, 34(1):11–39.

[63] Grossmann, I. E. (2002). Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, 3(3):227–252.

[64] Hager, W. W. and Seetharama Gowda, M. (1999). Stability in the presence of degeneracy and error estimation. *Mathematical Programming*, 85(1):181–192.

[65] Hairer, E., Nørsett, S. P., and Wanner, G. (2008). *Solving ordinary differential equations I: nonstiff problems*, volume 1. Springer Science & Business.

[66] Harjunkoski, I. (1997). *Application of MINLP methods to a scheduling problem in the paper-converting industry*. Process design laboratory, Department of chemical engineering, Abo Akademi University.

[67] Hestenes, M. R. (1969). Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320.

[68] Hock, W. and Schittkowski, K. (1980). Test examples for nonlinear programming codes. *Journal of Optimization Theory and Applications*, 30(1):127–129.

[69] Holsapple, R., Iyer, R., and Doman, D. (2007). Variable step-size selection methods for implicit integration schemes for odes. *International journal of numerical analysis and modeling. Computing and Information*, 4(2):210–240.

[70] Housh, M., Ostfeld, A., and Shamir, U. (2012). Box-constrained optimization methodology and its application for a water supply system model. *Journal of Water Resources Planning and Management*, 138(6):651–659.

[71] Karlsson, S. (2001). *Optimization of a sequential-simulated moving-bed separation process with mathematical programming methods*. Process Design Laboratory, Department of Chemical Engineering, Abo Akademi University.

[72] Karuppiah, R. and Grossmann, I. E. (2006). Global optimization for the synthesis of integrated water systems in chemical processes. *Computers & Chemical Engineering*, 30(4):650–673.

[73] Kelley, Jr, J. E. (1960). The cutting-plane method for solving convex programs. *Journal of the Society for Industrial & Applied Mathematics*, 8(4):703–712.

[74] Khalil, E. M., Zhou, H., and Chen, W. (2014). Steepest-ascent revisited: Unconstrained missile trajectory. *International Journal of Aerospace Engineering*, 2014.

[75] Kocis, G. R. and Grossmann, I. E. (1989a). Computational experience with dicopt solving minlp problems in process systems engineering. *Computers & Chemical Engineering*, 13(3):307–315.

[76] Kocis, G. R. and Grossmann, I. E. (1989b). A modelling and decomposition strategy for the minlp optimization of process flowsheets. *Computers & chemical engineering*, 13(7):797–819.

[77] Lastusilta, T. (2007). An implementation of the extended cutting plane method in gams.

[78] Lehmann, T. (2013). *On Efficient Solution Methods for Mixed-Integer Nonlinear and Mixed-Integer Quadratic Optimization Problems*. PhD thesis, PhD thesis, Universität Bayreuth, 2013. urn: nbn: de: bvb: 703-opus4-12758.

[79] Lewis, A. S. and Pang, J.-S. (1998). Error bounds for convex inequality systems. In *Generalized Convexity, Generalized Monotonicity: Recent Results*, pages 75–110. Springer.

[80] Leyffer, S. (2001). Integrating sqp and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, 18(3):295–309.

[81] Lieckens, K. and Vandaele, N. (2007). Reverse logistics network design with stochastic lead times. *Computers & Operations Research*, 34(2):395–416.

[82] Liu, S., Adams, A., and Ibrahim, B. M. (2013). Effects of tax on investment portfolios and financial markets under mixed integer stochastic programming. Technical report, Centre for Finance and Investment, Heriot Watt University.

[83] Liuzzi, G., Lucidi, S., and Rinaldi, F. (2012). Derivative-free methods for bound constrained mixed-integer optimization. *Computational Optimization and Applications*, 53(2):505–526.

[84] Lucidi, S., Piccialli, V., and Sciandrone, M. (2005). An algorithm model for mixed variable programming. *SIAM Journal on Optimization*, 15(4):1057–1084.

[85] Luenberger, D. G. and Ye, Y. (2008). *Linear and nonlinear programming*, volume 116. Springer.

[86] M, G.-P. U. and L, M. O. (1976). Superlinearly convergent quasi-newton methods for nonlinearly constrained optimization problems. *Mathematical Programming*, 11:1–13.

[87] Martinez, J. M. (2000). Box-quacan and the implementation of augmented lagrangian algorithms for minimization with inequality constraints. *Computational and Applied Mathematics*, 19:31–56.

[88] McDonald, C. M. and Karimi, I. A. (1997). Planning and scheduling of parallel semicontinuous processes. 1. production planning. *Industrial & Engineering Chemistry Research*, 36(7):2691–2700.

[89] Murtagh, B. A. and Saunders, M. A. (1998). Minos 5.5 users guide. report sol 83-20r, dept of operations research.

[90] Newby, E. (2013). *General solution methods for mixed integer quadratic programming and derivative free mixed integer non-linear programming problems*. PhD thesis, Computational and Applied Mathematics, The University of the Witwatersrand.

[91] Newby, E. and Ali, M. M. (2014). A trust-region-based derivative free algorithm for mixed integer programming. *Computational Optimization and Applications*, pages 1–31.

[92] Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

[93] Nzengang, F. V. and Ali, M. M. (2010). Introduction to mixed integer nonlinear programming.

[94] O Exler, LT Antelo, J. E. A. A. and Banga, J. (2008). A tabu search-based algorithm for mixed-integer nonlinear problems and its application to integrated process and control system design. *Computers and Chemical Engineering*, 32(8):77–91.

[95] P, H. S. (1976). Superlinearly convergent variable metric algorithms for general nonlinear programming problems. *Mathematical Programming*, 11:263–282.

[96] P, H. S. (1977). A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Applications*, 22:297–309.

[97] Pettersson, F. (1994). *Mixed integer non-linear programming applied on pump configurations*. PhD thesis, Department of Chemical Engineering, Abo Akademi University.

[98] Pillo, G. and Roma, M. (2006). *Large-scale nonlinear optimization*, volume 83. Springer.

[99] Polisetty, P. K. and Gatzke, E. P. (2005). A decomposition-based minlp solution method using piecewise linear relaxations. *International Transactions in Operations Research*.

[100] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[101] Porn, R. (2000). *Mixed integer non-linear programming: convexification techniques and algorithm development*. PhD thesis, Department of Chemical Engineering, Abo Akademi University.

[102] Pörn, R. and Westerlund, T. (2000). A cutting plane method for minimizing pseudo-convex functions in the mixed integer case. *Computers & Chemical Engineering*, 24(12):2655–2665.

[103] Potra, F. A. and Wright, S. J. (2000). Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281–302.

[104] Powell, M. (1977). A fast algorithm for nonlinearly constrained optimization calculations. *Numerical analysis Dundee*, pages 144–157.

[105] Powell, M. (1978a). Algorithms for nonlinear constraints that use lagrangian functions. *Mathematical programming*, pages 224–248.

[106] Powell, M. (1978b). The convergence of variable metric methods for nonlinearly constrained optimization calculations. *Nonlinear programming*, 3:27–63.

[107] Powell, M. J. (1967). *"A method for non-linear constraints in minimization problems"*. UKAEA.

[108] R. Silva, J. S. and Vincente, L. N. (2008). Local analysis of the feasible primal-dual interior point method. *Computational Optimization and Applications*, 41:41 – 57.

[109] Roslöf, J. (2002). *Application of MILP-based Methods to a Class of Industrial Production Scheduling Problems*. PhD thesis, Abo Akademi University.

[110] Rubinov, A., Yang, X., and Bagirov, A. (2002). Penalty functions with a small penalty parameter. *Optimization Methods and software*, 17(5):931–964.

[111] Ryberg, A.-B., Domeij Bäckryd, R., and Nilsson, L. (2012). Metamodel-based multidisciplinary design optimization for automotive applications.

[112] Ryoo, H. S. and Sahinidis, N. V. (1995). Global optimization of nonconvex nlps and minlps with applications in process design. *Computers & Chemical Engineering*, 19(5):551–566.

[113] Schlüter, M., Egea, J. A., and Banga, J. R. (2009). Extended ant colony optimization for non-convex mixed integer nonlinear programming. *Computers & Operations Research*, 36(7):2217–2229.

[114] Smith, E. and Pantelides, C. C. (1997). Global optimisation of nonconvex minlps. *Computers & Chemical Engineering*, 21:S791–S796.

[115] Snyman, J. A. (1982). A new and dynamic method for unconstrained minimization. *Applied Mathematical Modelling*, 6(6):449–462.

[116] Snyman, J. A. (1983). An improved version of the original leap-frog dynamic method for unconstrained minimization lfop1(d). *Applied Mathematical Modelling*, 7:216–218.

[117] Snyman, J. A. (2000). The lfopc leap-frog dynamic method for constrained optimization. *Computers Math. Applic.*, 40:1085–1096.

[118] Snyman, J. A. and Fatti, L. (1987). A multi-start global minimization algorithm with dynamic search trajectories. *Journal of Optimization Theory and Applications*, 54(1):121–141.

[119] Still, C. and Westerlund, T. (2009). *Extended Cutting Plane Algorithm*. Springer.

[120] van den Heever, S. A. and Grossmann, I. E. (2003). A strategy for the integration of production planning and reactive scheduling in the optimization of a hydrogen supply network. *Computers & Chemical Engineering*, 27(12):1813–1839.

[121] Vanderbei, R. J. (1999). Loqo users manual version 3.10. *Optimization methods software*, 12:485–514.

[122] Vanderbei, R. J. (2005). Nonlinear programming and engineering applications. In *Tutorials on Emerging Methodologies and Applications in Operations Research*, pages 7–1. Springer.

[123] Wah, B. W., Chen, Y., and Wang, T. (2007). Simulated annealing with asymptotic convergence for nonlinear constrained optimization. *Journal of Global Optimization*, 39(1):1–37.

[124] Westerlund (2005). *Aspects on N-dimensional allocation*. PhD thesis, Department of Chemical Engineering, Abo Akademi University,.

[125] Westerlund, T. and Pettersson, F. (1995). An extended cutting plane method for solving convex minlp problems. *Computers & Chemical Engineering*, 19:131–136.

[126] Westerlund, T. and Pörn, R. (2002). Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3(3):253–280.

[127] Westerlund, T., Skrifvars, H., Harjunkoski, I., and Pörn, R. (1998). An extended cutting plane method for a class of non-convex minlp problems. *Computers & Chemical Engineering*, 22(3):357–365.

[128] Wilson, R. B. (1963). *A simplicial algorithm for concave programming*. PhD thesis, Graduate School of Business Administration, Harvard University.

[129] Wolsey, L. A. (1998). *Integer programming*, volume 42. Wiley New York.

[130] Wright, S. and Nocedal, J. (1999). *Numerical optimization*, volume 2. Springer New York.

[131] Zhu, W. and Lin, G. (2011). A dynamic convexized method for nonconvex mixed integer nonlinear programming. *Computers & Operations Research*, 38(12):1792–1804.

Appendix A

Results for the trajectory-based algorithms

This appendix contains large data used during the discussion of this thesis, as well as the detailed computational results obtained using TACNLP, ATAUNLP and TAMINLP. Section A.1 contains the results and information on the test problems used in TACNLP and ATAUNLP, in Tables A.1 - A.5. In Tables A.2 and A.5, for each problem P the tables provide the value of x^* as well as the difference between x^* and the solution obtained by each respective algorithm, x^k. Information on larger CNLP test problems is provided in Table A.3. The data used in the convergence discussion of Chapter 8 can be found in Section A.2 in (A.1). Lastly, the results for TAMINLP are given in Section A.3, in Tables A.6 - A.7. An analysis of the results contained in these Appendix is given in chapter 9.

A.1 TACNLP and ATAUNLP Results

P	n	$f(x)$	LE	LI	NE	NI	k_1	k_2	k_3	k_4	t_{k_3}	t_{k_4}
1	2	Constant	0	0	2	0	51	51	51	51	0.0431	0.0666
2	2	Linear	0	0	1	0	35	35	33	103	0.0272	0.0606
3	2	Linear	0	0	1	0	47	49	47	137	0.0292	0.0699
4	2	Linear	0	0	0	1	35	36	36	99	0.0290	0.6184
5	2	Linear	0	0	0	1	37	37	37	$\bar{I}$	0.0299	0.3048
6	2	Linear	0	0	0	1	166	464	82	76	0.0509	0.4689
7	2	Linear	0	0	1	1	74	70	68	373	0.0520	0.2137
8	2	Quadratic	1	0	0	0	82	66	72	81	0.0502	0.0485
9	2	Quadratic	1	0	0	0	33	33	33	73	0.0218	0.0417

P	n	$f(x)$	LE	LI	NE	NI	k_1	k_2	k_3	k_4	t_{k_3}	t_{k_4}
10	2	Quadratic	1	0	0	0	39	39	39	101	0.0272	0.0557
11	2	Quadratic	1	0	0	0	50	46	50	$\bar{I}$	0.0339	0.3567
12	2	Quadratic	0	1	0	0	31	31	31	$\bar{I}$	0.0233	0.3799
13	2	Quadratic	0	1	0	0	62	68	56	$\bar{I}$	0.0391	0.3536
14	2	Quadratic	0	0	1	0	15	15	15	$\bar{I}$	0.0132	0.4300
15	2	Quadratic	0	0	1	0	21	37	21	41	0.0154	0.0291
16	2	Quadratic	0	0	1	0	91	91	91	117	0.0620	0.0678
17	2	Quadratic	0	0	0	2	48	342	48	69	0.0368	0.0440
18	2	Quadratic	0	2	0	0	15	20	15	98	0.0632	0.0623
19	2	Quadratic	0	2	0	0	35	33	35	$\bar{I}$	0.0277	0.3249
20	2	Quadratic	1	0	0	1	38	38	39	$\bar{I}$	0.0310	0.3168
21	2	Quadratic	0	1	2	0	93	62	93	$\bar{I}$	0.0698	0.3141
22	2	Quadratic	0	0	1	0	62	63	62	63	0.0416	0.0382
23	2	Quadratic	0	0	0	1	42	42	42	$\bar{I}$	0.0349	0.3166
24	2	Quadratic	0	0	0	1	64	94	62	219	0.0441	0.1853
25	2	Quadratic	0	2	0	1	31	27	29	27	0.0914	0.0947
26	2	Quadratic	0	1	0	1	35	33	35	$\bar{I}$	0.0310	0.3323
27	2	Nonlinear	0	0	1	0	28	28	28	$\bar{I}$	0.0231	0.1313
28	2	Nonlinear	1	0	0	0	62	50	62	73	0.0425	0.0429
29	2	Nonlinear	0	1	0	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.0634	0.0710
30	2	Nonlinear	0	1	0	0	88	81	83	106	0.0603	0.0164
31	2	Nonlinear	0	2	0	3	78	74	79	103	0.0653	0.0736
32	2	Nonlinear	0	1	0	2	88	102	93	$\bar{I}$	0.0749	0.0527
33	2	Nonlinear	0	0	3	2	108	102	109	$\bar{I}$	0.0408	0.3598
34	2	Nonlinear	0	5	0	0	96	$\bar{I}$	94	$\bar{I}$	0.0526	0.3429
35	2	Nonlinear	0	2	0	0	64	55	66	422	0.0502	0.2522
36	3	Linear	0	1	1	0	60	61	67	$\bar{I}$	0.0559	0.3485
37	3	Linear	0	0	0	2	53	45	46	86	0.0393	0.0628
38	3	Quadratic	0	0	1	0	69	69	69	60	0.0539	0.0427
39	3	Quadratic	2	0	0	0	166	$\bar{I}$	166	$\bar{I}$	0.1347	0.3262
40	3	Quadratic	2	0	0	0	42	42	42	$\bar{I}$	0.0361	0.3625
41	3	Nonlinear	0	0	1	0	52	46	52	$\bar{D}$	0.0431	–
42	3	Nonlinear	0	0	1	0	104	434	104	165	0.0792	0.1118
43	3	Nonlinear	1	0	0	0	74	75	74	$\bar{I}$	0.0622	0.3552

P	n	$f(x)$	LE	LI	NE	NI	k_1	k_2	k_3	k_4	t_{k_3}	t_{k_4}
44	4	Linear	0	0	2	0	157	167	157	186	0.1225	0.1319
45	4	Linear	0	2	0	2	447	447	447	311	0.3360	0.2326
46	4	Quadratic	1	0	1	0	34	34	34	$\bar{I}$	0.0332	0.4334
47	4	Quadratic	0	0	7	0	77	75	79	62	0.0669	0.0488
48	4	Quadratic	3	0	0	0	$\bar{I}$	$\bar{I}$	57	$\bar{I}$	0.0508	0.3746
49	4	Quadratic	0	10	0	0	$\bar{I}$	$\bar{I}$	177	$\bar{I}$	0.1629	0.3987
50	4	Nonlinear	2	0	0	0	66	72	66	$\bar{D}$	0.0490	–
51	4	Nonlinear	0	3	0	0	116	116	116	56	0.1014	0.0481
52	5	Quadratic	2	0	0	0	125	$\bar{I}$	125	$\bar{I}$	0.1030	0.3664
53	5	Nonlinear	0	0	3	0	113	$\bar{I}$	113	97	0.0958	0.0776
54	6	Nonlinear	6	8	0	0	231	$\bar{I}$	231	$\bar{I}$	0.2399	0.4959
55	7	Nonlinear	0	0	0	4	111	164	105	266	0.3538	0.7244
56	9	Quadratic	0	1	0	13	74	74	74	71	0.0836	0.0812
57	10	Quadratic	0	3	0	5	146	171	162	142	0.4242	0.4874
58	20	Nonlinear	0	0	1	0	71	80	71	$\bar{I}$	0.0696	0.4474
59	50	Nonlinear	0	0	1	0	186	$\bar{I}$	186	$\bar{I}$	0.4040	0.4444
60 (BT1)	2	Quadratic	0	1	0	0	60	66	60	$\bar{I}$	0.0551	0.4068
61 (BT2)	2	Quadratic	0	0	1	0	59	59	59	69	0.0507	0.0589
62 (BT3)	5	Nonlinear	2	0	1	0	56	57	56	$\bar{I}$	0.0774	0.4402
63 (BT4)	3	Nonlinear	1	0	0	1	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.5036	0.4873
64 (BT5)	5	Quadratic	3	0	2	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.4405	0.4403
65 (BT6)	5	Quadratic	1	0	1	0	107	93	107	$\bar{I}$	0.0016	0.0150
66 (BT7)	5	Nonlinear	0	0	3	0	$\bar{I}$	$\bar{D}$	$\bar{I}$	$\bar{D}$	0.4282	–
67 (BT8)	5	Quadratic	0	0	2	0	39	44	39	$\bar{I}$	0.0410	0.4535
68 (BT9)	4	Linear	0	0	2	0	188	188	188	$\bar{I}$	0.1515	0.4287
69 (BT10)	2	Linear	0	0	2	0	174	$\bar{I}$	174	$\bar{I}$	0.1437	0.3708
70 (BT11)	5	Nonlinear	1	0	2	0	90	90	90	246	0.1325	0.2088
71 (BT12)	5	Nonlinear	0	0	3	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.4935	0.0638
Average:							83	89	82	129	0.1016	0.2211

Table A.1 Results based on the performance of TACNLP on 71 test problems, with $\mu \in [0.01, 10]$

P		k_1	k_2	k_3	k_4
	x^*	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|2$	$\|\|x^* - x^k\|\|_2$
1	(cf. Table A.3)	1.42×10^{-4}	1.42×10^{-4}	1.42×10^{-4}	1.42×10^{-5}
2	$(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})^T$	9.53×10^{-5}	9.43×10^{-5}	9.53×10^{-5}	4.59×10^{-4}
3	$(3, 2)^T$	7.10×10^{-4}	8.91×10^{-4}	7.10×10^{-4}	8.19×10^{-3}
4	$(0, 0)^T$	2.89×10^{-4}	3.08×10^{-4}	3.20×10^{-4}	9.02×10^{-6}
5	$(-1, -1)^T$	3.11×10^{-4}	2.79×10^{-4}	3.18×10^{-4}	3.64×10^{-3}
6	$(0, 1)^T$	3.03×10^{-4}	1.86×10^{-4}	3.89×10^{-4}	7.64×10^{-4}
7	$(\frac{3}{5}, \frac{4}{5})^T$	5.18×10^{-5}	4.61×10^{-4}	4.06×10^{-4}	5.31×10^{-3}
8	$(1, 0)^T$	8.62×10^{-4}	9.63×10^{-4}	1.00×10^{-3}	1.19×10^{-3}
9	$(\frac{1}{2}, \frac{1}{2})^T$	5.00×10^{-4}	4.60×10^{-4}	5.00×10^{-4}	1.32×10^{-2}
10	$(\frac{5}{6}, \frac{1}{6})^T$	2.35×10^{-4}	2.45×10^{-4}	2.35×10^{-4}	8.28×10^{-5}
11	$(-\frac{1}{19}, -\frac{11}{19})^T$	1.22×10^{-4}	2.11×10^{-4}	1.22×10^{-4}	7.78×10^{-2}
12	$(\frac{7}{5}, \frac{17}{10})^T$	5.04×10^{-4}	2.78×10^{-4}	5.04×10^{-4}	7.37×10^{-2}
13	$(\frac{3}{5}, \frac{7}{10})^T$	1.16×10^{-4}	3.30×10^{-4}	9.86×10^{-5}	8.66×10^{-2}
14	$(2, 2)^T$	4.04×10^{-4}	4.04×10^{-4}	4.04×10^{-4}	8.72×10^{-2}
15	$(-1, -1)^T$	3.49×10^{-4}	3.09×10^{-4}	3.49×10^{-4}	1.20×10^{-1}
16	$(-2, -4)^T$	2.11×10^{-4}	2.11×10^{-4}	2.11×10^{-4}	8.93
17	$(-\frac{13}{40}, \frac{11}{40})^T$	5.22×10^{-5}	1.19×10^{-4}	5.22×10^{-5}	2.28×10^{-5}
18	$(\frac{3}{2}, \frac{3}{2})^T$	6.34×10^{-4}	6.92×10^{-4}	6.34×10^{-4}	2.22×10^{-5}
19	$(1, 1)^T$	3.78×10^{-4}	3.07×10^{-4}	3.07×10^{-4}	4.74×10^{-3}
20	(cf. Table A.3)	4.72×10^{-4}	4.72×10^{-4}	9.09×10^{-4}	2.60×10^{-2}
21	$(-2.372, -1.836)^T$	1.70×10^{-3}	3.63×10^{-4}	1.70×10^{-3}	7.73×10^{-3}
22	$(1, 1)^T$	1.11×10^{-3}	7.48×10^{-4}	1.11×10^{-3}	2.99×10^{-4}
23	$(1.2348, 1.5247)^T$	1.66×10^{-4}	1.66×10^{-4}	1.66×10^{-4}	5.23×10^{-3}
24	$(2, 3)^T$	5.95×10^{-4}	6.62×10^{-5}	1.70×10^{-4}	8.09×10^{-5}
25	$(1, 0)^T$	2.42×10^{-2}	2.29×10^{-2}	2.31×10^{-2}	2.28×10^{-2}
26	$(1, 1)^T$	1.29×10^{-4}	4.52×10^{-4}	2.02×10^{-4}	4.74×10^{-3}
27	$(0, \sqrt{3})^T$	1.85×10^{-4}	1.86×10^{-4}	1.85×10^{-4}	9.57×10^{-2}
28	$(-3, -4)^T$	1.3×10^{-2}	1.4×10^{-2}	1.3×10^{-2}	1.76×10^{-2}
20	$(1, 1)^T$	6.20×10^{-2}	6.20×10^{-2}	6.20×10^{-2}	1.38×10^{-1}
30	$(1.224, \frac{3}{2})^T$	4.20×10^{-5}	7.53×10^{-5}	6.14×10^{-5}	5.74×10^{-5}

P		k_1	k_2	k_3	k_4
	x^*	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$
31	$(\frac{1}{2},\ \frac{1}{2}\sqrt{3})^T$	9.98×10^{-4}	3.52×10^{-4}	3.65×10^{-4}	8.15×10^{-4}
32	$(\frac{1}{2},\ 2)^T$	4.38×10^{-4}	6.73×10^{-4}	2.40×10^{-3}	3.51
33	$(\frac{1}{2},\ \frac{1}{4})^T$	5.34×10^{-2}	4.98×10^{-4}	4.19×10^{-4}	9.78×10^{-2}
34	$(3,\ \sqrt{3})^T$	6.98×10^{-4}	5.44×10^{-3}	3.49×10^{-4}	8.63×10^{-3}
35	$(1,\ 0)^T$	2.84×10^{-3}	4.43×10^{-3}	2.72×10^{-3}	6.17×10^{-3}
36	$(\frac{3}{5},\ \frac{4}{5},\ 0)^T$	6.00×10^{-4}	4.10×10^{-4}	4.29×10^{-4}	6.39×10^{-3}
37	$(\sqrt{\frac{1}{3}},\ \sqrt{\frac{1}{3}},\ -\sqrt{\frac{1}{3}})^T$	2.33×10^{-5}	4.71×10^{-5}	5.36×10^{-5}	7.25×10^{-6}
38	$(0,\ 0,\ 4)^T$	2.23×10^{-4}	2.23×10^{-4}	2.23×10^{-4}	1.51×10^{-3}
39	$(\frac{89}{20},\ -\frac{29}{20},\ -\frac{59}{25})^T$	2.27×10^{-4}	9.55×10^{-3}	2.27×10^{-4}	7.33×10^{-3}
40	$(\frac{32}{15},\ -\frac{13}{15},\ \frac{13}{15})^T$	4.80×10^{-4}	4.80×10^{-4}	4.79×10^{-4}	1.21×10^{-3}
41	$(1,\ 1,\ 1)^T$	2.51×10^{-4}	5.20×10^{-5}	2.51×10^{-4}	NaN
42	$(-1,\ 1,\ 0)^T$	2.23×10^{-3}	6.08×10^{-3}	2.23×10^{-3}	3.06×10^{-4}
43	$(\frac{1}{2},\ -\frac{1}{2},\ \frac{1}{2})^T$	1.55×10^{-3}	4.75×10^{-4}	1.55×10^{-3}	3.53×10^{-2}
44	$(1,\ 1,\ 0,\ 0)^T$	8.87×10^{-4}	8.96×10^{-4}	8.87×10^{-4}	4.45×10^{-4}
45	$(1,\ 1,\ 0,\ 0)^T$	8.68×10^{-2}	8.68×10^{-2}	8.68×10^{-2}	2.19×10^{-2}
46	$(2,\ 2,\ \frac{3}{5}\sqrt{2},\ \frac{4}{5}\sqrt{2})^T$	6.73×10^{-4}	6.69×10^{-4}	6.73×10^{-4}	8.01×10^{-2}
47	(cf. Table A.3)	2.77×10^{-4}	9.37×10^{-5}	6.12×10^{-5}	4.91×10^{-5}
48	$(0,\ 1,\ 2,\ -1)^T$	7.27×10^{-3}	5.81×10^{-3}	2.05×10^{-5}	5.55×10^{-3}
49	$(3,\ 0,\ 4,\ 0)^T$	8.64×10^{-2}	8.64×10^{-2}	5.35×10^{-4}	6.78×10^{-3}
50	(cf. Table A.3)	2.08×10^{-4}	9.99×10^{-5}	2.08×10^{-4}	NaN
51	(cf. Table A.3)	7.45×10^{-5}	7.45×10^{-5}	7.45×10^{-5}	1.38×10^{-4}
52	$(1,\ 1,\ 1,\ 1,\ 1)^T$	1.61×10^{-4}	2.86×10^{-3}	1.61×10^{-4}	1.18×10^{-4}
53	$(1,\ 1,\ 1,\ 1,\ 1)^T$	2.38×10^{-2}	1.09×10^{-1}	2.38×10^{-2}	2.33×10^{-2}
54	$(0,\ \frac{4}{3},\ \frac{5}{3},\ 1,\ \frac{2}{3},\ \frac{1}{3})^T$	2.43×10^{-3}	1.71	2.43×10^{-3}	9.30×10^{-1}
55	(cf. Table A.3)	3.90×10^{-2}	1.96×10^{-2}	2.87×10^{-2}	3.88×10^{-2}
56	(cf. Table A.3)	3.59×10^{-4}	3.59×10^{-4}	3.59×10^{-4}	5.25×10^{-3}
57	(cf. Table A.3)	8.09×10^{-1}	8.09×10^{-1}	8.09×10^{-1}	8.09×10^{-1}
58	(cf. Table A.3)	3.17×10^{-4}	2.53×10^{-4}	3.17×10^{-4}	2.00
59	(cf. Table A.3)	2.69×10^{-4}	1.83	2.69×10^{-4}	3.82×10^{-3}

P		k_1	k_2	k_3	k_4
	x^*	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$
60 (BT1)	$(1,\ 0))^T$	2.56×10^{-4}	2.54×10^{-5}	2.56×10^{-4}	3.49×10^{-3}
61 (BT2)	$(1.10,\ 1.20,\ 1.54)^T$	7.15×10^{-3}	7.15×10^{-3}	7.15×10^{-3}	1.52×10^{-3}
62 (BT3)	(cf. Table A.3)	5.56×10^{-4}	5.73×10^{-4}	5.56×10^{-4}	6.46×10^{-2}
63 (BT4)	(cf. Table A.3)	3.17	3.11	3.18	7.37
64 (BT5)	$(3.51,\ 0.22,\ 3.55)^T$	1.06×10^{-2}	8.41×10^{-3}	1.06×10^{-2}	2.36×10^{-2}
65 (BT6)	(cf. Table A.3)	7.70×10^{-4}	1.70×10^{-4}	7.70×10^{-4}	1.29×10^{-2}
66 (BT7)	(cf. Table A.3)	1.90	NaN	1.89	NaN
67 (BT8)	$(1,\ 0,\ 0,\ 0,\ 0)^T$	4.18×10^{-4}	3.12×10^{-4}	4.18×10^{-4}	2.28×10^{-2}
68 (BT9)	$(1,\ 1,\ 0,\ 0)^T$	1.42×10^{-3}	1.42×10^{-3}	1.42×10^{-3}	2.77×10^{-3}
69 (BT10)	$(1,\ 1)^T$	2.18×10^{-2}	6.56×10^{-3}	2.18×10^{-2}	6.50×10^{-1}
70 (BT11)	(cf. Table A.3)	2.07×10^{-3}	2.07×10^{-3}	2.07×10^{-3}	6.02×10^{-4}
71 (BT12)	(cf. Table A.3)	19.31	19.31	19.31	20.88

Table A.2 The error estimates of the solutions obtained using TACNLP on 71 test problems, with $\mu \in [0.01, 10]$

P	x^*
1	$(\sqrt{\frac{25+\sqrt{301}}{2}},\ \frac{9}{\sqrt{\frac{(25+\sqrt{301}}{2}}})^T$
20	$(\frac{1}{2}(\sqrt{7}-1),\ \frac{1}{4}(\sqrt{7}+1))^T$
47	$(3/11,\ 23/11,\ 0,\ 6/11)^T$
50	$(3.3973,\ 1.2405,\ -2.0731,\ -2.3865)^T$
52	$(2^{(-\frac{1}{3})},\ 2^{(-\frac{1}{2})},\ (-1)^i 2^{(-\frac{11}{12})},\ (-1)^i 2^{(-\frac{1}{4})})^T$
55	$(2.330499,\ 1.951372,\ -0.4775414,\ 4.365726,\ -0.6244870,\ 1.038131,\ 1.594227)^T$
56	$(0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0)^T$
57	$(2.1720,\ 2.3637,\ 8.7739,\ 5.0960,\ 0.9907,\ 1.4306,\ 1.3216,\ 9.8287,\ 8.2801,\ 8.3759)^T$
58	$(0.91287,\ 0.408268,\ -0.000017,\ -0.0000054,\ 0.00002, 0.0000089,\ 0.0000082, -0.000014,\ 0.000022,\ -0.000014,\ 0.0000135,\ -0.000004,\ 0.000011, -0.000013,\ 0.000079,\ 0.000002,\ 0.00000456,\ -0.000009,\ -0.000001,\ -0.0000014)^T$
59	$(0.91285,\ 0.40829,\ -0.0000065,\ -0.0000991,\ 0.000119,\ -0.0000465,\ 0.0000576,\ -0.000048,\ 0.0000257,\ 0.0000117,\ -0.000031,\ 0.0000087,\ 0.00002,\ -0.000012,\ -0.0000164,\ 0.00000734,\ 0.000017,\ 0.0000044,\ -0.0000059,\ -0.0000025,\ 0.0000046,\ 0.00000325,\ -0.0000666,\ -0.000144,\ -0.000012,\ -0.0000039,\ 0.00000099,\ 0.00000015,\ -0.00000068,\ 0.0000024,\ 0.0000054,\ 0.0000027,\ -0.00000293,\ -0.0000038,\ 0.00000061,\ 0.0000044,\ 0.0000041,\ 0.000001455,\ -0.00000126,\ -0.000003,\ -0.00000386,\ -0.00000426,\ -0.00000451,\ -0.0000045,\ -0.0000038,\ -0.00000234,\ -0.00000075,\ -0.000000546,\ -0.0000011,\ -0.0000021)^T$
62 (BT3)	$(-0.76744,\ 0.25581, 0.62791, -0.11628,\ 0.25581)^T$
63 (BT4)	$(4.04,\ -2.95, -0.09)^T$
65 (BT6)	$(1.1662,\ 1.1821,\ 1.3803,\ 1.5060,\ 0.61092)^T$
66 (BT7)	$(-0.79212,\ -1.2624,\ 0,\ -0.89532,\ 1.1367)^T$
70 (BT11)	$(1.1912,\ 1.3626,\ 1.4728,\ 1.6349,\ 1.6790)^T$
71 (BT12)	$(15.811,\ 1.5811,\ 0,\ 15.083,\ 3.7164)^T$

Table A.3 Data for large test problems

P	n	$f(x)$	LE	LI	NE	NI	k_1	k_2	k_3	k_4	t_{k_3}
1	2	Constant	0	0	2	0	133	133	133	133	0.0615
2	2	Linear	0	0	1	0	89	82	89	105	0.0365
3	2	Linear	0	0	1	0	74	73	74	91	0.0341
4	2	Linear	0	0	0	1	98	78	98	142	0.5226
5	2	Linear	0	0	0	1	79	58	79	118	0.0492
6	2	Linear	0	0	0	1	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.2235
7	2	Linear	0	0	1	1	78	171	78	96	0.0397
8	2	Quadratic	1	0	0	0	129	110	131	121	0.0421
9	2	Quadratic	1	0	0	0	93	118	93	111	0.0393
10	2	Quadratic	1	0	0	0	102	86	102	100	0.0367
11	2	Quadratic	1	0	0	0	80	98	80	117	0.0449
12	2	Quadratic	0	1	0	0	104	75	104	103	0.0372
13	2	Quadratic	0	1	0	0	98	100	98	124	0.0488
14	2	Quadratic	0	1	0	0	120	230	120	114	0.0443
15	2	Quadratic	0	0	1	0	77	76	77	102	0.0411
16	2	Quadratic	0	0	1	0	126	126	126	128	0.0519
17	2	Quadratic	0	0	0	2	245	$\bar{I}$	226	132	0.0506
18	2	Quadratic	0	2	0	0	93	100	120	122	0.0438
19	2	Quadratic	0	2	0	0	79	284	79	116	0.0508
20	2	Quadratic	1	0	0	1	138	$\bar{I}$	138	121	0.0562
21	2	Quadratic	0	1	2	0	290	98	290	279	0.1321
22	2	Quadratic	0	0	1	0	66	28	66	115	0.0425
23	2	Quadratic	0	1	0	1	110	103	110	122	0.0592
24	2	Quadratic	0	0	0	1	125	120	103	110	0.0519
25	2	Quadratic	0	2	0	1	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.2960
26	2	Quadratic	0	1	0	1	79	284	79	116	0.0554
27	2	Nonlinear	0	0	1	0	110	170	110	100	0.0403
28	2	Nonlinear	1	0	0	0	286	186	286	233	0.0886
29	2	Nonlinear	0	1	0	0	190	190	190	$\bar{I}$	0.0924
30	2	Nonlinear	0	1	0	0	71	70	71	75	0.6539

P	n	$f(x)$	LE	LI	NE	NI	k_1	k_2	k_3	k_4	t_{k_3}
31	2	Nonlinear	0	2	0	3	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.2811
32	2	Nonlinear	0	1	0	2	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.2787
33	2	Nonlinear	0	0	3	2	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.2647
34	2	Nonlinear	0	5	0	0	177	450	389	105	0.0426
35	2	Nonlinear	0	2	0	0	115	103	93	97	0.0409
36	3	Linear	0	1	1	0	300	210	210	168	0.0793
37	3	Linear	0	0	0	2	124	100	142	124	0.05668
38	3	Quadratic	0	0	1	0	116	128	116	114	0.0478
39	3	Quadratic	2	0	0	0	$\bar{I}$	499	$\bar{I}$	$\bar{I}$	0.3271
40	3	Quadratic	2	0	0	0	$\bar{I}$	262	$\bar{I}$	111	0.3251
41	3	Nonlinear	0	0	1	0	181	98	181	$\bar{I}$	0.5483
42	3	Nonlinear	0	0	1	0	171	333	171	149	0.0684
43	3	Nonlinear	1	0	0	0	93	259	93	163	0.0669
44	4	Linear	0	0	2	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3089
45	4	Linear	0	2	0	2	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3003
46	4	Quadratic	1	0	1	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	269	0.3495
47	4	Quadratic	0	0	7	0	$\bar{I}$	$\bar{I}$	356	280	0.1423
48	4	Quadratic	3	0	0	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3932
49	4	Nonlinear	0	10	0	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3391
50	4	Nonlinear	2	0	0	0	90	99	90	194	0.0906
51	4	Nonlinear	0	3	0	0	102	102	102	108	0.0574
52	5	Quadratic	2	0	0	0	161	106	161	124	0.0581
53	5	Nonlinear	0	0	3	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	203	0.3070
54	6	Nonlinear	6	8	0	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3383
55	7	Nonlinear	0	0	0	4	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{D}$	0.2947
56	9	Quadratic	0	1	0	13	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.4159
57	10	Quadratic	0	3	0	5	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3795
58	20	Nonlinear	0	0	1	0	403	122	403	149	0.0886
59	50	Nonlinear	0	0	1	0	299	$\bar{I}$	299	358	0.2398

P	n	$f(x)$	LE	LI	NE	NI	k_1	k_2	k_3	k_4	t_{k_3}
60 (BT1)	2	Quadratic	0	1	0	0	108	152	108	240	0.1409
61 (BT2)	2	Quadratic	0	0	1	0	119	154	119	128	0.0649
62 (BT3)	5	Nonlinear	2	0	1	0	214	359	214	286	0.1648
63 (BT4)	3	Nonlinear	1	0	0	1	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3630
64 (BT5)	5	Quadratic	3	0	2	0	331	258	$\bar{I}$	$\bar{I}$	0.3167
65 (BT6)	5	Quadratic	1	0	1	0	$\bar{D}$	$\bar{D}$	$\bar{D}$	$\bar{D}$	–
66 (BT7)	5	Nonlinear	0	0	3	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3454
67 (BT8)	5	Quadratic	0	0	2	0	113	85	113	124	0.0638
68 (BT9)	4	Linear	0	0	2	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3177
69 (BT10)	2	Linear	0	0	2	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	299	0.4118
70 (BT11)	5	Nonlinear	1	0	2	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.3401
71 (BT12)	5	Nonlinear	0	0	3	0	$\bar{I}$	$\bar{I}$	$\bar{I}$	$\bar{I}$	0.1678
Average:							143	158	140	124	0.1639

Table A.4 Results based on the performance of ATAUNLP on 71 test problems, with $\mu \in [0.01, 10]$

P		k_1	k_2	k_3	k_4
	x^*	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$
1	(cf. Table A.3)	1.84×10^{-4}	1.84×10^{-4}	1.84×10^{-4}	1.84×10^{-4}
2	$(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})^T$	3.33×10^{-5}	1.23×10^{-3}	3.33×10^{-5}	2.50×10^{-5}
3	$(3, 2)^T$	4.31×10^{-4}	1.96×10^{-4}	4.31×10^{-4}	3.83×10^{-4}
4	$(0, 0)^T$	7.51×10^{-5}	2.03×10^{-4}	7.56×10^{-5}	2.81×10^{-5}
5	$(-1, -1)^T$	1.75×10^{-5}	2.18×10^{-5}	1.75×10^{-5}	2.15×10^{-6}
6	$(0, 1)^T$	9.90×10^{-1}	9.90×10^{-1}	9.90×10^{-1}	9.90×10^{-1}
7	$(\frac{3}{5}, \frac{4}{5})^T$	9.77×10^{-5}	3.97×10^{-4}	5.63×10^{-4}	1.89×10^{-5}
8	$(1, 0)^T$	8.62×10^{-4}	5.07×10^{-4}	1.00×10^{-3}	5.20×10^{-4}
9	$(\frac{1}{2}, \frac{1}{2})^T$	3.86×10^{-4}	4.51×10^{-3}	3.86×10^{-4}	4.86×10^{-6}
10	$(\frac{5}{6}, \frac{1}{6})^T$	2.50×10^{-4}	9.97×10^{-4}	2.50×10^{-4}	1.10×10^{-4}
11	$(-\frac{1}{19}, -\frac{11}{19})^T$	1.79×10^{-4}	9.61×10^{-5}	1.79×10^{-4}	1.10×10^{-4}
12	$(\frac{7}{5}, \frac{17}{10})^T$	1.36×10^{-4}	3.45×10^{-4}	1.36×10^{-4}	3.27×10^{-2}
13	$(\frac{3}{5}, \frac{7}{10})^T$	1.89×10^{-4}	6.51×10^{-4}	1.89×10^{-4}	1.66×10^{-4}
14	$(2, 2)^T$	3.74×10^{-4}	1.82×10^{-3}	3.74×10^{-5}	4.89×10^{-5}
15	$(-1, -1)^T$	1.42×10^{-4}	9.69×10^{-6}	1.42×10^{-4}	6.96×10^{-7}
16	$(-2, -4)^T$	1.05×10^{-4}	1.05×10^{-4}	1.05×10^{-4}	1.23×10^{-4}
17	$(-\frac{13}{40}, \frac{11}{40})^T$	5.79×10^{-5}	7.56×10^{-4}	1.91	1.91
18	$(\frac{3}{2}, \frac{3}{2})^T$	8.10×10^{-5}	2.37×10^{-3}	6.90×10^{-5}	3.01×10^{-5}
19	$(1, 1)^T$	1.56×10^{-3}	1.48×10^{-4}	1.63×10^{-3}	3.56×10^{-5}
20	(cf. Table A.3)	7.50×10^{-5}	4.90×10^{-1}	7.50×10^{-5}	6.35×10^{-5}
21	$(-2.372, -1.836)^T$	4.73×10^{-4}	4.87×10^{-4}	4.73×10^{-4}	4.71×10^{-4}
22	$(1, 1)^T$	4.26×10^{-3}	6.33×10^{-3}	4.26×10^{-3}	1.75×10^{-3}
23	$(1.2348, 1.5247)^T$	2.28×10^{-4}	2.98×10^{-4}	2.28×10^{-4}	1.49×10^{-4}
24	$(2, 3)^T$	1.52×10^{-4}	2.70×10^{-4}	2.00×10^{-4}	4.42×10^{-4}
25	$(1, 0)^T$	6.99	6.88	6.80	5.44
26	$(1, 1)^T$	1.56×10^{-3}	1.48×10^{-4}	1.63×10^{-3}	3.56×10^{-5}
27	$(0, \sqrt{3})^T$	5.46×10^{-5}	8.10×10^{-3}	5.46×10^{-5}	5.67×10^{-5}
28	$(-3, -4)^T$	5.21×10^{-3}	1.83×10^{-3}	5.21×10^{-3}	1.12×10^{-3}
29	$(1, 1)^T$	9.0×10^{-1}	9.0×10^{-1}	9.0×10^{-1}	9.0×10^{-1}
30	$(1.224, \frac{3}{2})^T$	1.60×10^{-1}	1.60×10^{-1}	1.60×10^{-1}	4.30×10^{-1}

P		k_1	k_2	k_3	k_4
	x^*	$\|\|x^*-x^k\|\|_2$	$\|\|x^*-x^k\|\|_2$	$\|\|x^*-x^k\|\|_2$	$\|\|x^*-x^k\|\|_2$
31	$(\frac{1}{2},\ \frac{1}{2}\sqrt{3})^T$	7.95×10^{-2}	1.38	1.86×10^{-2}	9.47×10^{-2}
32	$(\frac{1}{2},\ 2)^T$	5.80×10^{-1}	5.90×10^{-1}	8.10×10^{-1}	3.89
33	$(\frac{1}{2},\ \frac{1}{4})^T$	1.6×10^{-1}	2.40×10^{-1}	5.90×10^{-1}	1.30×10^{-1}
34	$(3,\ \sqrt{3})^T$	1.38×10^{-4}	1.27×10^{-4}	1.93×10^{-4}	1.24×10^{-3}
35	$(1,\ 0)^T$	7.94×10^{-4}	2.68×10^{-3}	2.13×10^{-3}	6.75×10^{-4}
36	$(\frac{3}{5},\ \frac{4}{5},\ 0)^T$	3.00×10^{-4}	3.02×10^{-4}	7.58×10^{-5}	2.82×10^{-5}
37	$(\sqrt{\frac{1}{3}},\ \sqrt{\frac{1}{3}},\ -\sqrt{\frac{1}{3}})^T$	5.99×10^{-5}	9.39×10^{-5}	3.74×10^{-5}	2.45×10^{-6}
38	$(0,\ 0,\ 4)^T$	1.10×10^{-3}	1.11×10^{-3}	1.10×10^{-3}	3.76×10^{-4}
39	$(\frac{89}{20},\ -\frac{29}{20},\ -\frac{59}{25})^T$	4.50×10^{-1}	1.97×10^{-4}	4.50×10^{-1}	1.30×10^{-1}
40	$(\frac{32}{15},\ -\frac{13}{15},\ \frac{13}{15})^T$	4.60×10^{-1}	2.45×10^{-4}	2.40×10^{-1}	2.35×10^{-4}
41	$(1,\ 1,\ 1)^T$	5.29×10^{-4}	7.59×10^{-5}	5.29×10^{-4}	3.38
42	$(-1,\ 1,\ 0)^T$	1.38×10^{-3}	3.51×10^{-3}	1.38×10^{-3}	1.27×10^{-4}
43	$(\frac{1}{2},\ -\frac{1}{2},\ \frac{1}{2})^T$	1.71×10^{-3}	1.55×10^{-3}	1.71×10^{-3}	2.57×10^{-3}
44	$(1,\ 1,\ 0,\ 0)^T$	5.30×10^{-1}	4.70×10^{-1}	5.30×10^{-1}	2.00
45	$(1,\ 1,\ 0,\ 0)^T$	12.20	11.67	12.11	5.20
46	$(2,\ 2,\ \frac{3}{5}\sqrt{2},\ \frac{4}{5}\sqrt{2})^T$	6.60×10^{-1}	1.70×10^{-1}	6.60×10^{-1}	1.43×10^{-3}
47	(cf. Table A.3)	1.41×10^{-3}	1.10×10^{-1}	3.54×10^{-4}	8.83×10^{-4}
48	$(0,\ 1,\ 2,\ -1)^T$	1.022	5.00×10^{-2}	9.10×10^{-1}	1.10×10^{-1}
49	$(3,\ 0,\ 4,\ 0)^T$	6.04	1.07	6.04	5.42
50	(cf. Table A.3)	3.06×10^{-4}	1.41×10^{-4}	3.06×10^{-4}	3.89×10^{-5}
51	(cf. Table A.3)	3.01×10^{-4}	3.01×10^{-4}	3.01×10^{-4}	1.87×10^{-4}
52	$(1,\ 1,\ 1,\ 1,\ 1)^T$	3.65×10^{-4}	2.79×10^{-3}	3.65×10^{-4}	1.27×10^{-4}
53	$(1,\ 1,\ 1,\ 1,\ 1)^T$	4.78×10^{-1}	5.66×10^{-2}	4.78×10^{-1}	7.30×10^{-4}
54	$(0,\ \frac{4}{3},\ \frac{5}{3},\ 1,\ \frac{2}{3},\ \frac{1}{3})^T$	2.53	8.42×10^{-1}	2.53	1.68×10^{-1}
55	(cf. Table A.3)	1.67	4.37×10^{-1}	4.81×10^{-1}	6.86×10^{-1}
56	(cf. Table A.3)	7.04×10^{-1}	7.04×10^{-1}	7.04×10^{-1}	3.43×10^{-1}
57	(cf. Table A.3)	2.26	1.59	2.71	2.76
58	(cf. Table A.3)	8.17×10^{-1}	1.86×10^{-4}	8.17×10^{-1}	1.36×10^{-4}
59	(cf. Table A.3)	1.21×10^{-3}	1.47	1.21×10^{-3}	2.00

P		k_1	k_2	k_3	k_4
	x^*	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$	$\|\|x^* - x^k\|\|_2$
60 (BT1)	$(1,\ 0))^T$	9.91×10^{-5}	5.99×10^{-4}	9.91×10^{-5}	3.57×10^{-5}
61 (BT2)	$(1.10,\ 1.20,\ 1.54)^T$	1.42×10^{-4}	1.10×10^{-3}	1.42×10^{-4}	6.09×10^{-4}
62 (BT3)	(cf. Table A.3)	1.57×10^{-4}	5.78×10^{-4}	1.57×10^{-4}	1.94×10^{-4}
63 (BT4)	(cf. Table A.3)	3.49	3.15	3.18	3.70
64 (BT5)	$(3.51,\ 0.22,\ 3.55)^T$	7.38×10^{-3}	4.65×10^{-4}	7.38×10^{-3}	2.40×10^{-2}
65 (BT6)	(cf. Table A.3)	1.08×10^{8}	2.61×10^{8}	1.08×10^{8}	1.28×10^{12}
66 (BT7)	(cf. Table A.3)	4.02	4.08	4.02	3.44
67 (BT8)	$(1,\ 0,\ 0,\ 0,\ 0)^T$	2.45×10^{-2}	4.65×10^{-4}	2.45×10^{-2}	8.44×10^{-4}
68 (BT9)	$(1,\ 1,\ 0,\ 0)^T$	1.55	4.70×10^{-2}	1.55	2.00
69 (BT10)	$(1,\ 1)^T$	7.27	9.68	7.27	2.56×10^{-2}
70 (BT11)	(cf. Table A.3)	6.31	6.31	6.31	3.73
71 (BT12)	(cf. Table A.3)	18.22	18.22	18.22	18.03

Table A.5 The error estimates of the solutions obtained using ATAUNLP on 71 test problems, with $\mu \in [0.01, 10]$

A.2 MINLP Data

$$\left\{\begin{pmatrix}
0 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 0 & 1\\
1 & 1 & 0 & 0 & 0\\
1 & 0 & 1 & 0 & 0\\
1 & 0 & 0 & 1 & 0\\
1 & 0 & 0 & 0 & 1\\
0 & 1 & 1 & 0 & 0\\
0 & 1 & 0 & 1 & 0\\
0 & 1 & 0 & 0 & 1\\
0 & 0 & 1 & 1 & 0\\
0 & 0 & 1 & 0 & 1\\
0 & 0 & 0 & 1 & 1\\
1 & 1 & 1 & 0 & 0\\
1 & 1 & 0 & 1 & 0\\
1 & 1 & 0 & 0 & 1\\
1 & 0 & 1 & 1 & 0\\
1 & 0 & 1 & 0 & 1\\
1 & 0 & 0 & 1 & 1\\
0 & 1 & 1 & 1 & 0\\
0 & 1 & 1 & 0 & 1\\
0 & 1 & 0 & 1 & 1\\
0 & 0 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 0\\
1 & 1 & 1 & 0 & 1\\
1 & 1 & 0 & 1 & 1\\
1 & 0 & 1 & 1 & 1\\
0 & 1 & 1 & 1 & 1\\
1 & 1 & 1 & 1 & 1
\end{pmatrix}\right\}. \tag{A.1}$$

The rows of (A.1) are made of all the $0-1$ combinations for a binary problem in $\mathbb{R}^5$.

P	n_c	n_d	$f(x)$	LE	LI	NE	NI	$\bar{k}_1$	$\bar{k}_2$	$\bar{k}_3$	fe	ε_d	$\bar{y}$	n_p	ref
1	1	1	Linear	0	4	0	1	$\bar{I}$	0	0	0	–	–	–	[51]
2	1	1	Linear	0	5	0	1	$\bar{I}$	147	69	0	1	✓	2	[31]
3	1	1	Linear	0	4	0	0	56	43	79	3	1	✓	3	[66]
4	1	1	Linear	0	4	0	2	94	98	44	3	1	✓	3	[43]
5	1	1	Nonlinear	0	4	0	0	113	0	0	0	–	–	–	[91]
6	1	1	Nonlinear	0	3	0	1	34	18	$\bar{I}$	3	1	✗	3	[126]
7	1	1	Nonlinear	0	5	0	1	150	41	181	3	1	✗	3	[101]
8	1	1	Nonlinear	0	5	0	1	21	0	18	3	1	✓	3	[43]
9	1	1	Nonlinear	0	6	0	0	$\bar{I}$	218	$\bar{I}$	3	1	✓	3	[101]
10	1	1	Linear	0	4	0	0	$\bar{I}$	39	$\bar{I}$	5	1	✓	5	[50]
11	2	1	Linear	0	1	0	5	103	0	0	0	–	–	–	[31]
12	2	1	Nonlinear	0	8	0	1	129	212	81	1	1	✓*	2	[51]
13	2	2	Nonlinear	0	1	0	0	23	–	19	2	1	✓	2	[93]
14	2	2	Bilinear	10	0	0	0	$\bar{I}$	264	$\bar{I}$	4	1	✓*	5	[101]
15	1	3	Nonlinear	0	12	0	0	$\bar{I}$	488	$\bar{I}$	6	2	✗	4	[50]
16	2	3	Linear	1	12	0	1	$\bar{I}$	212	0	3	1	✓*	4	[99]
17	3	3	Linear	12	0	2	0	$\bar{I}$	192	$\bar{I}$	3	1	✗	4	[21, 51]
18	3	3	Nonlinear	0	16	0	2	401	179	393	3	1	✓*	4	[42]
19	3	4	Nonlinear	0	12	0	5	$\bar{I}$	291	$\bar{I}$	4	1	✓*	5	[51]
20	2	6	Linear	0	21	0	1	$\bar{D}$	$\bar{D}$	$\bar{D}$	–	–	–	–	[99]
21	7	2	Linear	3	14	2	0	$\bar{I}$	285	$\bar{I}$	0	1	✗	3	[97]
22	3	8	Linear	0	26	3	0	$\bar{I}$	190	I	8	1	✓	8	[51]
23	6	5	Nonlinear	1	32	0	3	$\bar{I}$	382	$\bar{I}$	6	1	✓	6	[42]
24	9	8	Nonlinear	2	51	0	4	$\bar{I}$	387	$\bar{I}$	7	1	✓	7	[42]

Table A.6 The performance of TAMINLP on 24 test problems, with $\mu \in [0.01, 10]$

A.3 TAMINLP RESULTS

P	x^*	$\|\|x^k - x^*\|\|$
1	$(4,\ 1)^T$	1.4×10^{-3}
2	$(0.5,\ 1)^T$	3.98×10^{-5}
3	$(3,\ 2)^T$	1.74×10^{-4}
4	$(-2.1381,\ -1)^T$	6.07×10^{-6}
5	$(2,\ -2)^T$	1.43×10^{-3}
6	$(4.3338,\ 3)^T$	9.7×10^{-3}
7	$(3.528,\ 4)^T$	1.69×10^{-2}
8	$(-\sqrt{2}/2,\ 1)^T$	2.36×10^{-9}
9	$(1.9891,\ 0)^T$	4.62×10^{-5}
10	$(0.5,\ 1)^T$	8.60×10^{-4}
11	$(1.375,\ 0.375,\ 1)^T$	2.51×10^{-4}
12	$(0.9419,\ -2.1,\ 1)^T$	1.36×10^{-4}
13	$(0,\ 0,\ 0)^T$	5.23×10^{-6}
14	$(3,\ 0,\ 5,\ 0)^T$	5.00×10^{-3}
15	$(0.2, 0, 0, 1)^T$	1.90×10^{-3}
16	$(4,\ 1,\ 1,\ 0,\ 0)^T$	8.8×10^{-3}
17	$(1.12,\ 1.3,\ 1,\ 0,\ 1,\ 1)^T$	1.040×10^{-2}
18	$(1.30097,\ 0,\ 1,\ 0,\ 1,\ 0)^T$	2.6×10^{-3}
19	$(0.2,\ 0.8,\ 1.908,\ 1,\ 1,\ 0,\ 1)^T$	1
20	$(3,\ 1,\ 1,\ 1,\ 0,\ 1,\ 0,\ 0)^T$	-
21	$(13.428,\ 0,\ 13.428,\ 10,\ 0,\ 3.5140,\ 0,\ 1,\ 0)^T$	1.59×10^{-1}
22	$(0.97,\ 0.9925,\ 0.98,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 1, 0)^T$	1
23	$(0,\ 2,\ 0.65201,\ 0.32601,\ 1.07839,\ 1.07839,\ 0,\ 1,\ 1,\ 1,\ 0)^T$	1.5853
24	$(0,\ 2,\ 0.46784,\ 0.58480,\ 2,\ 0,\ 0,\ 0.26667,\ 0.58480,\ 1,\ 0,\ 0,\ 1,\ 0,\ 1,\ 0,\ 0)^T$	4.2377

Table A.7 The error estimates of the solutions obtained using TAMINLP on 24 test problems, with $\mu \in [0.01, 10]$

Appendix B

Test Problems

This appendix contains details of the continuous test problems used in chapter 9. In the list below we provide the objective function as well as the constraints which make up each CNLP.

- Problem 1

$$\begin{cases} \min & -1 \\ \text{st} & x_1^2 + x_2^2 - 25 = 0, \\ & x_1 x_2 - 9 = 0. \end{cases}$$

- Problem 2

$$\begin{cases} \min & -x_1 - x_2 \\ \text{st} & x_1^2 + x_2^2 - 1 = 0. \end{cases}$$

- Problem 3

$$\begin{cases} \min & 2x_1 + 3x_2 \\ \text{st} & x_1 x_2 - 6 = 0. \end{cases}$$

- Problem 4

$$\begin{cases} \min & x_1 \\ \text{st} & 1 - (x_1 - 1)^2 - x_2^2 \geq 0. \end{cases}$$

- Problem 5

$$\begin{cases} \min & x_1 + x_2 \\ \text{st} & 2 - x_1^2 - x_2^2 \geq 0. \end{cases}$$

- Problem 6

$$\begin{cases} \min & x_1 - x_2 \\ \text{st} & 1 - 3x_1^2 + 2x_1x_2 - x_2^2 \geq 0. \end{cases}$$

- Problem 7

$$\begin{cases} \min & -x_2 \\ \text{st} & 1 + x_1 - 2x_2 \geq 0, \\ & x_1^2 + x_2^2 - 1 = 0. \end{cases}$$

- Problem 8

$$\begin{cases} \min & 10^{-5}(x_2 - x_1)^2 + x_2 \\ \text{st} & x_2 \geq 0. \end{cases}$$

- Problem 9

$$\begin{cases} \min & x_1^2 + x_2^2 \\ \text{st} & x_1 + x_2 - 1 = 0. \end{cases}$$

- Problem 10

$$\begin{cases} \min & x_1^2 + 5x_2^2 \\ \text{st} & x_1 + x_2 - 1 = 0. \end{cases}$$

- Problem 11

$$\begin{cases} \min & -\frac{1}{2}x_1^2 + 6x_1 + 2x_1x_2 + x_2^2 - 2x_2 \\ \text{st} & 3x_1 - 2x_2 - 1 = 0. \end{cases}$$

- Problem 12

$$\begin{cases} \min & (x_1-1)^2+(x_2-2.5)^2 \\ \text{st} & 2+x_1-2x_2 \geq 0. \end{cases}$$

- Problem 13

$$\begin{cases} \min & x_1^2-2x_1-2x_1x_2-6x_2+2x^2 \\ \text{st} & 1-\frac{1}{2}x_1-\frac{1}{2}x_2 \geq 0, \\ & 2-x_1-2x_2 \geq 0. \end{cases}$$

- Problem 14

$$\begin{cases} \min & x_1x_2 \\ \text{st} & x_1+x_2-4 \geq 0. \end{cases}$$

- Problem 15

$$\begin{cases} \min & x_1^2+x_2^2 \\ \text{st} & x_1^2+x_2^2-2=0. \end{cases}$$

- Problem 16

$$\begin{cases} \min & x_1^2+x_2^2 \\ \text{st} & 3x_1^2+4x_1x_2+6x_2^2-140=0. \end{cases}$$

- Problem 17

$$\begin{cases} \min & \frac{1}{6}x_1+\frac{10}{3}x_1x_2 \\ \text{st} & \frac{19}{6}-x_1^2+\frac{5}{2}x_2^2 \geq 0, \\ & x_1-x_2+\frac{3}{5} \geq 0. \end{cases}$$

- Problem 18

$$\begin{cases} \min & (x_1-2)^2-(x_2-2)^2 \\ \text{st} & 3-x_1-x_2 \geq 0, \\ & -2+10x_1-x_2 \geq 0. \end{cases}$$

- Problem 19

$$\begin{cases} \min & (x_1-2)^2-(x_2-1)^2 \\ \text{st} & 2-x_1-x_2 \geq 0, \\ & x_2-x_1^2 \geq 0. \end{cases}$$

- Problem 20

$$\begin{cases} \min & (x_1-2)^2+(x_2-1)^2 \\ \text{st} & 1-0.25x_1^2-x_2^2 \geq 0, \\ & x_1-2x_2+1=0. \end{cases}$$

- Problem 21

$$\begin{cases} \min & x_1^2+x_2 \\ \text{st} & -(x_1+x_2)+1 \geq 0, \\ & -(x_1+x_2^2)+1 \geq 0, \\ & x_1^2+x_2^2-9=0. \end{cases}$$

- Problem 22

$$\begin{cases} \min & (1-x_1)^2 \\ \text{st} & 10(x_2-x_1^2)=0. \end{cases}$$

- Problem 23

$$\begin{cases} \min & (x_1-5)^2+x_2^2-25 \\ \text{st} & -x_1^2+x_2 \geq 0. \end{cases}$$

- Problem 24

$$\begin{cases} \min & 0.5x_1^2+x_2^2-x_1x_2-7x_1-7x_2 \\ \text{st} & 25-4x_1^2-x_2^2 \geq 0. \end{cases}$$

- Problem 25

$$\begin{cases} \min & (x_1-2)^2+x_2^2 \\ \text{st} & (1-x_1)^3-x_2\geq 0, \\ & x_1\geq 0, \\ & x_2\geq 0. \end{cases}$$

- Problem 26

$$\begin{cases} \min & (x_1-2)^2+(x_2-1)^2 \\ \text{st} & 2-x_1-x_2\geq 0, \\ & x_2-x_1^2\geq 0. \end{cases}$$

- Problem 27

$$\begin{cases} \min & \ln(1+x_1^2)-x_2 \\ \text{st} & (1+x_1^2)^2+x_2^2-4=0, \end{cases}$$

- Problem 28

$$\begin{cases} \min & \sin(\frac{\pi}{12}x_1)\cos(\frac{\pi}{16}x_2) \\ \text{st} & 4x_1-3x_2=0. \end{cases}$$

- Problem 29

$$\begin{cases} \min & 100(x_2-x_1^2)^2+(1-x_1)^2 \\ \text{st} & x_2+1.5\geq 0. \end{cases}$$

- Problem 30

$$\begin{cases} \min & 100(x_2-x_1^2)^2+(1-x_1)^2 \\ \text{st} & x_2-1.5\geq 0. \end{cases}$$

- Problem 31

$$\begin{cases} \min & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ \text{st} & x_1 + x_2^2 \geq 0, \\ & x_1^2 + x_2 \geq 0, \\ & x_1^2 + x_2^2 - 1 \geq 0, \\ & -0.5 \leq x_1 \leq 0.5, \end{cases}$$

- Problem 32

$$\begin{cases} \min & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ \text{st} & x_1 x_2 - 1 \geq 0, \\ & x_1 + x_2^2 \geq 0, \\ & 0.5 - x_1 \geq 0. \end{cases}$$

- Problem 33

$$\begin{cases} \min & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ \text{st} & x_1 + x_2^2 \geq 0, \\ & x_1^2 + x_2 \geq 0, \\ & 1 - x_2 \geq 0 \\ & -0.5 \leq x_1 \leq 0.5, . \end{cases}$$

- Problem 34

$$\begin{cases} \min & \frac{1}{27\sqrt{3}}((x_1 - 3)^2 - 9)x_2^3 \\ \text{st} & \frac{x_1}{\sqrt{3}} - x_2 \geq 0, \\ & x_1 + \sqrt{3}x_2 \geq 0, \\ & 6 - x_1 - \sqrt{3}x_2 \geq 0, \\ & x_1 \geq 0, \\ & x_2 \geq 0. \end{cases}$$

- Problem 35

$$\begin{cases} \min & \frac{1}{3}(x_1 + 1)^3 + x_2 \\ \text{st} & x_1 - 1 \geq 0, \\ & x_2 \geq 0. \end{cases}$$

- Problem 36

$$\begin{cases} \min & -x_2 \\ \text{st} & 1-2x_2+x_1 \geq 0, \\ & x_1^2+x_2^2+x_3^2-1=0. \end{cases}$$

- Problem 37

$$\begin{cases} \min & -x_1-x_2+x_3 \\ \text{st} & 1-x_1^2-x_2^2-x_3^2 \geq 0, \\ & 1-x_1^3-x_3 \geq 0. \end{cases}$$

- Problem 38

$$\begin{cases} \min & x_1^2+3x_2^2 \\ \text{st} & x_1^2+x_2^2+x_3^2-16=0. \end{cases}$$

- problem 39

$$\begin{cases} \min & x_1^2+4x_1+2x_1x_2+3x_2^2+5x_2+6x_3 \\ \text{st} & x_1+x_2-3=0, \\ & 4x_1+5x_3-6=0. \end{cases}$$

- Problem 40

$$\begin{cases} \min & 3x_1^2-8x_1+2x_1x_2+\frac{5}{2}x_2^2-3x_2+2x_2x_3+x_1x_3+3x_3^2-3x_3 \\ \text{st} & x_1+x_3-3=0, \\ & x_2+x_3=0. \end{cases}$$

- Problem 41

$$\begin{cases} \min & (x_1-x_2)^2+(x_2-x_3)^4 \\ \text{st} & (1+x_2^2)x_1+x_3^4-3=0. \end{cases}$$

- Problem 42

$$\begin{cases} \min & 0.01(x_1-1)^2+(x_2-x_1^2)^2 \\ \text{st} & x_1+x_3^2+1=0. \end{cases}$$

- Problem 43

$$\begin{cases} \min & (x_1+x_2)^2+(x_2+x_3)^2 \\ \text{st} & x_1+2x_2+3x_3-1=0. \end{cases}$$

- Problem 44

$$\begin{cases} \min & -x_1 \\ \text{st} & x_2-x_1^3-x_3^2=0, \\ & x_1^2-x_2-x_4^2=0. \end{cases}$$

- Problem 45

$$\begin{cases} \min & -x_1 \\ \text{st} & x_2-x_1^3\geq 0, \\ & x_1^2-x_2\geq 0, \\ & x_2-x_1^3-x_3^2=0, \\ & x_1^2-x_2-x_4^2=0. \end{cases}$$

- Problem 46

$$\begin{cases} \min & (x_1-1)^2+(x_2-2)^2+(x_3-3)^2+(x_4-4)^2 \\ \text{st} & x_1-2=0, \\ & x_3^2+x_4^2-2=0. \end{cases}$$

- Problem 47

$$\begin{cases} \min & x_1^2+0.5x_2^2+x_3^2+0.5x_4^2-x_1x_3+x_3x_4-x_1-3x_2+x_3-x_4 \\ \text{st} & -x_1-2x_2-x_3-x_4+5\geq 0, \\ & -3x_1-x_2-2x_3+x_4+4=0, \\ & x_2+4x_3+4\geq 0, \\ & x_i\geq 0,\ i=1,2,3,4. \end{cases}$$

- Problem 48

$$\begin{cases} \min & x_1^2+x_2^2+2x_3^2+x_4^2-5x_1-5x_2-21x_3+7x_4 \\ \text{st} & 8-x_1^2-x_1-x_2^2+x_2-x_3^2-x_3-x_4^2+x_4 \geq 0, \\ & 10-x_1^2+x_1-2x_2^2-x_3^2-2x_4^2+x_4 \geq 0, \\ & 5-2x_1^2-2x_1-x_2^2+x_2-x_3^2+x_4 \geq 0. \end{cases}$$

- Problem 49

$$\begin{cases} \min & x_1-x_2-x_3-x_1x_3+x_1x_4+x_2x_3-x_2x_4 \\ \text{st} & 8-x_1-2x_2 \geq 0, \\ & 12-4x_1-x_2 \geq 0, \\ & 12-3x_1-4x_2 \geq 0, \\ & 8-2x_3-x_4 \geq 0, \\ & 8-x_3-2x_4 \geq 0, \\ & 5-x_3-x_4 \geq 0, \\ & x_i \geq 0,\ i=1\ldots 4. \end{cases}$$

- Problem 50

$$\begin{cases} \min & (x_1-3)^4+(x_2+x_3)^6+(x_4+2)^2+\exp(x_1+x_2+x_3+x_4) \\ \text{st} & 2x_1-3x_2+x_3-1=0, \\ & -x_2+2x_3-x_4+3=0. \end{cases}$$

- Problem 51

$$\begin{cases} \min & -x_1x_2x_3x_4 \\ \text{st} & x_1^3+x_2^2-1=0, \\ & x_1^2x_4-x_3=0, \\ & x_4^2-x_2=0. \end{cases}$$

- Problem 52

$$\begin{cases} \min & (x_1-1)^2+(x_2-x_3)^2+(x_4-x_5)^2 \\ \text{st} & x_1+x_2+x_3+x_4+x_5-5=0, \\ & x_3-2(x_4+x_5)+3=0. \end{cases}$$

- Problem 53

$$\begin{cases} \min & (x_1-x_2)^2+(x_2-x_3)^3+(x_3-x_4)^4+(x_4-x_5)^4 \\ \text{st} & x_1+x_2^2+x_3^3-3=0, \\ & x_2-x_3^2+x_4-1=0, \\ & x_1x_5-1=0. \end{cases}$$

- Problem 54

$$\begin{cases} \min & x_1+2x_2+4x_5+\exp(x_1x_4) \\ \text{st} & x_1+2x_2+5x_5-6=0, \\ & x_1+x_2+x_3-3=0, \\ & x_4+x_5+x_6-2=0 \\ & x_1+x_4-1=0, \\ & x_2+x_5-2=0, \\ & x_3+x_6-2=0, \\ & x_i\geq 0,\ i=1,\dots 6, \\ & 1-x_1\geq 0, \\ & 1-x_4\geq 0. \end{cases}$$

- Problem 55

$$\begin{cases} \min & (x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2+10x_5^6+7x_6^2+x_7^4-4x_6x_7-10x_6-8x_7 \\ \text{st} & 127-2x_1^2-3x_2^4-x_3-4x_4^2-5x_5\geq 0, \\ & 282-7x_1-3x_2-10x_3^2-x_4+x_5\geq 0, \\ & 196-23x_1-x_2^2-6x_6^2+8x_7\geq 0, \\ & -4x_1^2-x_2^2+3x_1x_2-2x_3^2-5x_6+11x_7\geq 0. \end{cases}$$

- Problem 56

$$\begin{cases} \min & -0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7) \\ \text{st} & 1 - x_3^2 - x_4^2 \geq 0, \\ & 1 - x_5^3 - x_6^2 \geq 0, \\ & 1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 \geq 0, \\ & 1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 \geq 0, \\ & 1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \geq 0, \\ & 1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \geq 0, \\ & 1 - x_7^2 - (x_8 - x_9)^2 \geq 0, \\ & 1 - x_1^2 - (x_2 - x_9)^2 \geq 0, \\ & x_5x_8 - x_6x_7 \geq 0, \\ & x_1x_4 - x_2x_3 \geq 0, \\ & 1 - x_9^2 \geq 0, \\ & x_3x_9 \geq 0, \\ & -x_5x_9 \geq 0, \\ & x_9 \geq 0. \end{cases}$$

- Problem 57

$$\begin{cases} \min & x_1^2 + x_2^2 + x_1x_2 - 14x_1 + 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 \\ & +5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_10 - 7)^2 + 45 \\ \text{st} & 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0, \\ & -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0, \\ & 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0, \\ & 120 - 3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 \geq 0, \\ & 40 - 5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 \geq 0, \\ & 30 - 0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 \geq 0. \\ & -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 + 145x_5 + 6x_6 \geq 0. \\ & 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0. \end{cases}$$

- Problem 58

$$\begin{cases} \min & \sum_{i=1}^{20} i(x_i^2 + x_i^4) \\ \text{st} & \sum_{i=1}^{20} x_i^2 - 1 = 0 \end{cases}$$

- Problem 59

$$\begin{cases} \min & \sum_{i=1}^{50} i(x_i^2 + x_i^4) \\ \text{st} & \sum_{i=1}^{50} x_i^2 - 1 = 0 \end{cases}$$

- Problem 60

$$\begin{cases} \min & -x_1 + 10(x_1^2 + x_2^2 - 1) \\ \text{st} & x_1^2 + x_2^2 - 1 = 0 \end{cases}$$

- Problem 61

$$\begin{cases} \min & (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^4 \\ \text{st} & x_1(1 + x_2^2) + x_3^4 - 4 - 3\sqrt{2} = 0 \end{cases}$$

- Problem 62

$$\begin{cases} \min & (x_1 - x_2)^2 + (x_2 + x_3 - 2)^2 + (x_4 - 1)^2 + (x_5 - 1)^2 \\ \text{st} & x_1 + 3x_2 = 0 \\ & x_3 + x_4 - 2x_5 = 0 \\ & x_2 - x_5 = 0 \end{cases}$$

- Problem 63

$$\begin{cases} \min & x_1 - x_2 + x_2^3 \\ \text{st} & x_1^2 + x_2^2 + x_3^2 - 25 = 0 \\ & x_1 + x_2 + x_3 - 1 \geq 0 \end{cases}$$

- Problem 64

$$\begin{cases} \min & 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3 \\ \text{st} & -25 + x_1^2 + x_2^2 + x_3^2 = 0 \\ & -56 + 8x_1 + 14x_2 + 7x_3 = 0 \end{cases}$$

- Problem 65

$$\begin{cases} \min & (x_1-1)^2+(x_1-x_2)^2+(x_3-1)^2+(x_4-1)^4+(x_5-1)^6 \\ \text{st} & x_1^2x_4+\sin(x_4-x_5)-2\sqrt{2} \\ & x_2+x_3^4x_4^2-8-\sqrt{2}=0 \end{cases}$$

- Problem 66

$$\begin{cases} \min & 100(x_2-x_1^2)^2+(1-x_1)^2 \\ \text{st} & x_1x_2-1-x_3^2=0 \\ & x_2^2+x_1-x_4^2=0 \\ & -x_1-x_5^2+0.5=0 \end{cases}$$

- Problem 67

$$\begin{cases} \min & x_1^2+x_2^2+x_3^2 \\ \text{st} & x_1-x_4^2-1=0 \\ & x_1^2+x_2^2-x_5^2-1=0 \end{cases}$$

- Problem 68

$$\begin{cases} \min & -x_1 \\ \text{st} & x_2-x_1^3-x_3^2=0 \\ & x_1^2-x_2-x_4^2=0 \end{cases}$$

- Problem 69

$$\begin{cases} \min & -x_1 \\ \text{st} & x_2-x_1^3=0 \\ & x_1^2-x_2=0 \end{cases}$$

- Problem 70

$$\begin{cases} \min & (x_1-1)^2+(x_1-x_2)^2+(x_2-x_3)^2+(x_3-x_4)^4+(x_4-x_5)^4 \\ \text{st} & x_1+x_2^2+x_3^3-2-\sqrt{18}=0 \\ & x_2-x_3^2+x_4+2-\sqrt{8}=0 \\ & x_1x_5-2=0 \end{cases}$$

- Problem 71

$$\begin{cases} \min & 0.1x_1^2 + x_2 = 0 \\ \text{st} & x_1 + x_2 - x_3^2 - 25 = 0 \\ & x_1^2 + x_2^2 - x_4^2 - 25 = 0 \\ & x_1 - x_5^2 - 2 = 0 \end{cases}$$