SOLUTION OF LARGE-SCALE STRUCTURED OPTIMIZATION PROBLEMS WITH SCHUR-COMPLEMENT AND AUGMENTED LAGRANGIAN DECOMPOSITION METHODS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Jose S. Rodriguez

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Carl D. Laird, Chair

School of Chemical Engineering

Dr. Zoltan Nagy

School of Chemical Engineering

Dr. Joseph F. Pekny

School of Chemical Engineering

Dr. Dengfeng Sun

School of Aeronautics and Astronautics

Approved by:

Dr. Sangtae Kim

Head of the School Graduate Program

To my Family.

ACKNOWLEDGMENTS

I am extremely grateful for all of the opportunities that have arisen during my PhD studies. First I would like to thank my advisor Dr. Carl Laird for his unconditional support throughout all these years of graduate education. I was fortunate to meet Dr. Laird earlier in my career and it was him who encouraged me to pursue a doctorate degree. Since the time I met him he has provided me with numerous opportunities that helped me advance professionally as well as personally. Carl has played a very important part in my life and I will always be grateful with him for all his advice.

Special thanks to Dr. Victor Zavala for inviting me to work with him and his research group at University of Wisconsin-Madison. Even though I visited only for 4 months, it feels like I learned 4 years of optimization theory while I visited. I thank all students at Zavalab for the enlightening conversations. I had a great time in both my visits, always feeling part of the group. Thanks to Victor for his many lessons. I truly admire his diligence and passion for research. I envy his vast knowledge in statistics and optimization, and I hope to work with him again someday in the future.

I cannot thank my research group enough for their support and help. Especially, I would like to thank Jeff, Arpan, Yankai, Michael, Todd and Daniel for their friendship and all the time we spent learning together about optimization. I could not have found better colleagues for this journey. I will be forever thankful for your many lessons and help with my research project. I am also grateful to all my co-authors and collaborators from Sandia National Laboratories (Bethany Nicholson, Katherine Klise, David Hart, and John Siirola) and the U.S. Environmental Protection Agency (Terra Haxton and Jonathan Burkhardt) for their contributions.

I appreciate my committee members, Dr. Zoltan Nagy, Dr. Joseph Pekny, and Dr. Dengfeng Sun, for their constructive comments and suggestions on my research. Thanks are also extended to my supervisors during the internships at United Airlines, Amgen and Eli Lilly.

Finally, I would like to thank every member of my family for all their love and support. Mom and Dad this work would have never been possible without the invaluable lessons and wisdom you gave me throughout my childhood. You are my role models and my discipline and hard work I owe to you. Thanks Juliana and Sergio for always believing in me and for being there to listen to me every time I need you.

Disclaimer: This research was funded in part by the U.S. Environmental Protection Agency (EPA) through its Office of Research and Development funded and managed the research described herein under Interagency Agreement (IA # DW8992403601) with the Department of Energy's Sandia National Laboratories. It has been reviewed by the Agency but does not necessarily reflect the Agencys views. No official endorsement should be inferred. EPA does not endorse the purchase or sale of any commercial products or services.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energys National Nuclear Security Administration under contract DE-NA-0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government. This work was conducted as part of the Institute for the Design of Advanced Energy Systems (IDAES) with funding from the Office of Fossil Energy, Cross-Cutting Research, U.S. Department of Energy.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABBREVIATIONS

MINLP	mixed-integer nonlinear program
MILP	mixed-integer linear program
NLP	nonlinear program
AL	augmented Lagrangian
ALM	augmented Lagrangian method
MM	method of multipliers
ADMM	alternating direction method of multipliers
PH	progressive hedging
QP	quadratic program
SQP	sequential quadratic programming
KKT	KarushKuhnTucker
WDS	water distribution systems

ABSTRACT

Rodriguez, Jose S. Ph.D., Purdue University, August 2019. Solution of Large-scale Structured Optimization Problems with Schur-complement and Augmented Lagrangian Decomposition Methods. Major Professor: Carl D. Laird.

In this dissertation we develop numerical algorithms and software tools to facilitate parallel solutions of nonlinear programming (NLP) problems. In particular, we address large-scale, block-structured problems with an intrinsic decomposable configuration. These problems arise in a great number of engineering applications, including parameter estimation, optimal control, network optimization, and stochastic programming. The structure of these problems can be leveraged by optimization solvers to accelerate solutions and overcome memory limitations, and we propose variants to two classes of optimization algorithms: augmented Lagrangian (AL) schemes and Schur-complement interior-point methods.

The convergence properties of augmented Lagrangian decomposition schemes like the alternating direction method of multipliers (ADMM) and progressive hedging (PH) are well established for convex optimization but convergence guarantees in non-convex settings are still poorly understood. In practice, however, ADMM and PH often perform satisfactorily in complex non-convex NLPs. In this work, we study connections between the method of multipliers (MM), ADMM, and PH to derive benchmarking metrics that explain why PH and ADMM work in practice. We illustrate the concepts using challenging dynamic optimization problems. Our exposition seeks to establish more formalism in benchmarking ADMM, PH, and AL schemes and to motivate algorithmic improvements.

The effectiveness of nonlinear interior-point solvers for solving large-scale problems relies quite heavily on the solution of the underlying linear algebra systems.

The schur-complement decomposition is very effective for parallelizing the solution of linear systems with modest coupling. However, for systems with large number of coupling variables the schur-complement method does not scale favorably. We implement an approach that uses a Krylov solver (GMRES) preconditioned with ADMM to solve block-structured linear systems that arise in the interior-point method. We show that this ADMM-GMRES approach overcomes the well-known scalability issues of Schur decomposition.

One important drawback of using decomposition approaches like ADMM and PH is their convergence rate. Unlike Schur-complement interior-point algorithms that have super-linear convergence, augmented Lagrangian approaches typically exhibit linear and sublinear rates. We exploit connections between ADMM and the Schur-complement decomposition to derive an accelerated version of ADMM. Specifically, we study the effectiveness of performing a Newton-Raphson algorithm to compute multiplier estimates for augmented Lagrangian methods. We demonstrate using two-stage stochastic programming problems that our multiplier update achieves convergence in fewer iterations for MM on general nonlinear problems. In the case of ADMM, the newton update significantly reduces the number of subproblem solves for convex quadratic programs (QPs). Moreover, we show that using newton multiplier updates makes the method robust to the selection of the penalty parameter.

Traditionally, state-of-the-art optimization solvers are implemented in low-level programming languages. In our experience, the development of decomposition algorithms in these frameworks is challenging. They present a steep learning curve and can slow the development and testing of new numerical algorithms. To mitigate these challenges, we developed `PyNumero`, a new open source framework implemented in Python and C++. The package seeks to facilitate development of optimization algorithms for large-scale optimization within a high-level programming environment while at the same time minimizing the computational burden of using Python. The efficiency of `PyNumero` is illustrated by implementing algo-

rithms for problems arising in stochastic programming and optimal control. Timing results are presented for both serial and parallel implementations. Our computational studies demonstrate that with the appropriate balance between compiled code and Python, efficient implementations of optimization algorithms are achievable in these high-level languages.

Numerical Algorithms

1. INTRODUCTION

1.1 Motivation

Optimization of process systems has proven to be an effective method for improving operation and decision making in many industrial applications. The success of numerical optimization along with the rapid evolution of computing technologies has allowed solution of many previously intractable problems. When analyzing complex optimization problems, three features are commonly encountered. The first feature is that these problems are inherently large, often containing thousands or even millions of variables and constraints. Fortunately, a second common feature is that these problems are highly structured. Large-scale problems are often the result of discretization (in time, space, or uncertainty) or the result of large network structure. A third feature present is that these optimization problems are often integrated in real-time frameworks where they are expected to be solved rapidly.

Because of the size and complexity, as well as the need to solve optimization problems quickly, specialized or tailored approaches are often required. This thesis focuses on problem decomposition approaches that exploit the structure of the problem and allow efficient solution by successively solving smaller, more tractable subproblems. High-performance parallel computing architectures can be used for rapid execution of these algorithms, and this work seeks to design decomposition algorithms together with software tools for efficient parallel solution of large-scale optimization.

There are two decomposition strategies considered in this work: problem-level decomposition and internal linear decomposition. Problem-level decomposition includes approaches like Lagrangian decomposition, Bender's decomposition and

progressive hedging. Internal linear decomposition focuses on parallelizing an existing host algorithm through parallel solution of the linear algebra. Of course, these two approaches are related, and this work explores this relationship in the pursuit of efficient algorithms. We pay particular attention to augmented Lagrangian and Schur-complement interior-point methods. We highlight the advantages and disadvantages of these approaches when used separately and propose algorithms that blend ideas from both methods.

1.2 Large-Scale Nonlinear Optimization

Nonlinear optimization considers the following programming problem:

$$\begin{aligned} \min \;\; & f(x) \\ \text{s.t.} \;\; & c(x) = 0 \\ & x_L \leq \; x \leq x_U, \end{aligned} \tag{1.1}$$

where $x \in \mathbb{R}^n$ are the primal variables, $f : \mathbb{R}^n \to \mathbb{R}$ the objective function, and $c : \mathbb{R}^n \to \mathbb{R}^m$ the constraints. The vectors x_L and x_U are the lower and upper bounds on x. We assume both f and c can be non-convex, but twice continuously differentiable. We also assume that at any local minimum x^*, the linear independence constraint qualification (LICQ) and second-order sufficient conditions (SOSC) hold. We refer to this problem as a general *nonlinear programming problem*. Problems with general inequality constraints can be transformed to (1.1) by introducing slack variables. Details on mathematical transformations to write NLPs in the form of (1.1) are discussed in Appendix A.

This general problem formulation can be addressed by a great number of numerical algorithms. Among the most successful methods for solving general nonlinear optimization problems we highlight *primal-dual interior-point methods*. The packages `Ipopt` and `Knitro` are two well-known implementations of interior-point methods. In our work we have implemented a nonlinear interior-point frame-

work for researching and prototyping numerical routines tailored to accelerate solutions of (1.1).

Another category of methods that has recently regained attention for solving (1.1) are the *augmented Lagrangian methods*. These methods have been popular for many years because of their simplicity, but most recently because of their applicability to distributed optimization. Examples of optimization software that run efficient implementations of augmented Lagrangian methods are ALGENCAN and LANCELOT. In our work we used ideas from the augmented Lagrangian method as our work horse to accelerate solution of structured optimization problems of the form (1.1).

In this dissertation we combine ideas from augmented Lagrangian and interior-point methods to design decomposition algorithms for problem (1.1), and in the next two subsections we briefly discuss fundamental concepts of both approaches to familiarize the reader with notation as well as methodology.

1.2.1 Interior-point Method

The interior-point algorithm considers the barrier subproblem where the variable bounds are removed and replaced with barrier terms in the objective,

$$\begin{aligned} \min \quad & f(x) - \mu \sum_{i=1}^{n} \ln(x_U^{(i)} - x^{(i)}) - \mu \sum_{i=1}^{n} \ln(x^{(i)} - x_L^{(i)}) \\ \text{s.t.} \quad & c(x) = 0. \end{aligned} \tag{1.2}$$

Here (i) denotes the i^{th} element of the corresponding vector and μ is the barrier parameter for a single barrier iteration. The Lagrangian of the barrier subproblem (1.2) can then be written as

$$\mathcal{L} = f(x) - \mu \sum_{i=1}^{n} \ln(x_U^{(i)} - x^{(i)}) - \mu \sum_{j=1}^{n} \ln(x^{(j)} - x_L^{(j)}) + \lambda^T c(x), \tag{1.3}$$

where y is the vector of equality constraint multipliers. The general optimality conditions are

$$\begin{aligned} \nabla_x \mathcal{L} = \nabla_x f(x) + \mu(S_U)^{-1}e - \mu(S_L)^{-1}e + \nabla_x c(x)\lambda = 0 \\ c(x) = 0, \end{aligned} \tag{1.4}$$

with S_U=diag$(x_U - x)$ and S_L=diag$(x - x_L)$. The primal-dual formulation is formed by introducing new variables, $z_U = \mu[S_U]^{-1}e$ and $z_L = \mu[S_L]^{-1}e$. Since the algorithm maintains $x_U - x \geq 0$ and $x - x_L \geq 0$, it follows that the new variables $z_U, z_L \geq 0$. The optimality conditions with these new variables are

$$\begin{aligned} \nabla_x \mathcal{L} = \nabla_x f(x) + z_U - z_L + \nabla_x c(x)\lambda = 0 \\ c(x) = 0 \\ S_L z_L - \mu e = 0 \\ S_U z_U - \mu e = 0. \end{aligned} \tag{1.5}$$

Newton's method is used to solve this nonlinear system of equations for a particular value of the barrier parameter μ. The linear system that must be solved for each iteration k of Newton's method is

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}^k & (J^k)^T & -I & I \\ J^k & 0 & 0 & 0 \\ Z_L^k & 0 & S_L^k & 0 \\ -Z_U^k & 0 & 0 & S_U^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \\ \Delta z_L^k \\ \Delta z_U^k \end{bmatrix} = - \begin{bmatrix} \nabla_x f^k + z_U^k - z_L^k + (J^k)^T\lambda \\ c^k \\ S_L^k z_L^k - \mu e \\ S_U^k z_U^k - \mu e \end{bmatrix}, \tag{1.6}$$

where Δx^k, Δy^k, Δz_L^k, and Δz_U^k are the full steps for each of the respective variables, $Z_L^k = \text{diag}(z_L^k)$, $Z_U^k = \text{diag}(z_U^k)$, $c^k = c(x^k)$, J^k=$\nabla_x c(x^k)^T$, $\nabla_x f^k$=$\nabla_x f(x^k)$, and $\nabla^2_{xx}\mathcal{L}^k$=$\nabla^2_{xx}\mathcal{L}(x^k)$. The solution of (1.6) constitutes the core of the interior-point algorithm as it determines the step direction to update both primal and dual variables towards optimality. Several strategies have been proposed in the literature to solve (1.6) [Benzi et al., 2005, Greif et al., 2014]. Here we choose to reduce the 4×4 block system to a symmetric 2×2 block system often called the augmented

form. This is achieved by multiplying the third block row by $(S_L^k)^{-1}$, the fourth block row by $(-S_U^k)^{-1}$, and adding these rows to the first block row, giving

$$\underbrace{\begin{bmatrix} D^k & (J^k)^T \\ J^k & 0 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \end{bmatrix}}_{\Delta v} = - \underbrace{\begin{bmatrix} \tilde{r}_x^k \\ c^k \end{bmatrix}}_{r}, \tag{1.7}$$

where,

$$D^k = \nabla_{xx}^2 \mathcal{L}^k + (S_L^k)^{-1} Z_L^k + (S_U^k)^{-1} Z_U^k \tag{1.8}$$

$$\tilde{r}_x^k = \nabla_x f^k + (J^k)^T \lambda - (S_L^k)^{-1} \mu e + (S_U^k)^{-1} \mu e. \tag{1.9}$$

To ensure descent [Biegler, 2010], inertia correction algorithms are typically used to modify (1.7) and satisfy $\text{In}(K) = (n, m, 0)$ where n, m are the number of positive and negative eigenvalues. The inertia correction modifies (1.7) by adding primal and dual regularization to its block diagonals. The modified system is

$$\begin{bmatrix} D^k + \delta_x I & (J^k)^T \\ J^k & -\delta_\lambda I \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \end{bmatrix} = - \begin{bmatrix} \tilde{r}_x^k \\ c^k \end{bmatrix}, \tag{1.10}$$

where, δ_x and δ_λ are chosen to ensure the inertia condition is satisfied. This system is solved at each iteration to calculate the full step in x and y. The steps Δz_U and Δz_L can be computed as follows:

$$\Delta z_L^k = -(S_L^k)^{-1} Z_L^k \Delta x^k - z_L^k + \mu (S_L^k)^{-1} e \tag{1.11}$$

$$\Delta z_U^k = (S_U^k)^{-1} Z_U^k \Delta x^k - z_U^k + \mu (S_U^k)^{-1} e. \tag{1.12}$$

The variable values for the next iteration are determined by,

$$x^{k+1} = x^k + \alpha^k \Delta x^k \tag{1.13}$$

$$\lambda^{k+1} = \lambda^k + \alpha^k \Delta \lambda^k, \tag{1.14}$$

$$z_L^{k+1} = z_L^k + \alpha_L^k \Delta z_L^k \tag{1.15}$$

$$z_U^{k+1} = z_U^k + \alpha_U^k \Delta z_U^k, \tag{1.16}$$

where α^k is the step size determined by an appropriate line-search and α_L^k and α_U^k are step sizes determined using a fraction to the boundary rule. Once the barrier subproblem has converged, the barrier parameter is updated and the process is repeated. The complete interior-point algorithm is summarized in Algorithm 1. The algorithm is a primal-dual interior-point method with a filter-based line-search based on that described in [Wächter and Biegler, 2006].

1.2.2 The Augmented Lagrangian Method

We review now the augmented Lagrangian method (ALM), alternatively referred to as *method of multipliers*. This method is a fundamental base for many different nonlinear programming algorithms. It combines concepts from penalty functions and duality theory. As opposed to the interior-point that solves for x^* and λ^* simultaneously, the ALM solves a sequence of optimization problems where estimates x^k and λ^k are obtained iteratively. The sequence of problems solved has the following form:

$$\min_{x \in \mathcal{X}} \mathcal{L}_\rho(x, \lambda^k) = f_\rho(x) + c(x)^T \lambda^k = f(x) + \frac{\rho}{2} c(x)^T c(x) + c(x)^T \lambda^k \tag{1.17}$$

where $\rho \in \mathbb{R}^+$ is a penalty parameter and $\mathcal{X} = \{x | x_L \leq x \leq x_U\}$. A typical ALM implementation starts with an initial estimate λ^0. Using the initial estimate,

Algorithm 1: Interior-point Method

1. Initialize the algorithm
Given starting point $(x^0, \lambda^0, z_L^0, z_U^0)$ with $\lambda^0, z_L^0, z_U^0 > 0$; an initial barrier parameter $\mu_0 > 0$; tolerance constants $\epsilon_{tol}, \kappa_\epsilon > 0$; maximum number of iterations k_{max}
Set the iteration index $k \leftarrow 0$

2. Check convergence of the overall NLP
if $\max\{\| \nabla_x \mathcal{L}^k \|_\infty, \ \| c^k \|_\infty, \| S_L^k z_L^k \|_\infty, \ \| S_U^k z_U^k \|_\infty\} \leq \epsilon_{tol}$ **then exit.**

3. Check convergence of barrier subproblem
if $\max\{\| \nabla_x \mathcal{L}^k \|_\infty, \ \| c^k \|_\infty, \| S_L^k z_L^k - \mu^k e \|_\infty, \ \| S_U^k z_U^k - \mu^k e \|_\infty\} \leq \kappa_\epsilon \mu^k$ **then**
Update μ^k according to equation (7) in [Wächter and Biegler, 2006]
Return to step **2**

4. Calculate functions and gradients
Evaluate $f(x^k)$, $c(x^k)$, $\nabla_x f(x^k)$, $\nabla_x c(x^k)$, and $\nabla^2_{xx} \mathcal{L}(x^k, \lambda^k)$

5. Compute the search direction (full-step)
5.1 Solve (1.10) for Δx^k and $\Delta \lambda^k$, correcting the inertia if necessary
5.2 Compute Δz_L^k and Δz_U^k from (1.11) and (1.12)
5.3 Compute values for α^k, α_L^k, and α_U^k using fraction-to-the-boundary rule

6. Update α^k using the line-search filter from [Wächter and Biegler, 2006]

7. Update iteration variables and continue to next iteration
Compute $(x^{k+1}, \lambda^{k+1}, z_L^{k+1}, z_U^{k+1})$ using (1.13–1.16)
Let $\mu^{k+1} \leftarrow \mu^k$ and $k \leftarrow k + 1$
if $k < k_{\max}$ **then exit with error.**
Return to step **3**

$x^{k+1} = x(\lambda^k)$ is found by solving (1.17). Next, the multiplier estimates are updated according to

$$\lambda^{k+1} = \lambda^k + \rho c(x^{k+1}), \tag{1.18}$$

and the process is repeated until $\|\nabla_x \mathcal{L}_\rho(x, \lambda^k)\| \leq \epsilon$ and $c(x) \leq \epsilon$. We highlight that the method relies in an intrinsic relation between primal and dual variables that allows the sequence of updates to converge to (x^*, λ^*). To establish this relation we define the dual function $\phi(\lambda)$ in (1.19) and state three lemmas that explain the reasoning behind the update formulas.

$$\phi(\lambda) = \min_x \quad f_\rho(x) + \lambda^T c(x). \tag{1.19}$$

Lemma 1.2.1 *The gradient of the dual function is given by:*

$$\nabla_\lambda \phi(\lambda) = c(x) \tag{1.20}$$

Lemma 1.2.2 *The Hessian of the dual function is given by* [1]*:*

$$\nabla^2_{\lambda\lambda} \phi(\lambda) = -\nabla_x c(x) \nabla^2_{xx} \mathcal{L}_\rho(x, \lambda)^{-1} \nabla_x c(x)^T \tag{1.21}$$

Lemma 1.2.3 *Let P and Q be two symmetric matrices. Assume that Q is positive semidefinite and P is positive definite on the Null space of Q, that is $x^T P x > 0 \quad \forall \quad \{x | x^T Q x = 0\}$. Then there exits a scalar ρ^* such that*

$$x^T (P + \rho Q) x > 0 \quad \forall \rho \geq \rho^*$$

The update formula for the multiplier estimates presented in (1.18) follows a gradient based update. In fact, using Lemma 1.2.1 and comparing with (1.18) it is clear that ALM updates multiplier estimates by performing a dual-ascent strategy of the form

$$y^{k+1} = y^k + \alpha \nabla_\lambda \phi(\lambda). \tag{1.22}$$

In view of (1.18) as a dual-ascent update, it is natural to consider a Newton update instead. Using Lemma 1.2.2 and assuming that (1.19) is twice continuously differentiable, (1.22) can be modified to follow a second-order update of the following form:

$$y^{k+1} = y^k + \alpha \nabla^2_{\lambda\lambda} \phi(\lambda)^{-1} \nabla_\lambda \phi(\lambda), \tag{1.23}$$

This means that one can carry out first or second-order multiplier updates within the ALM framework. We will see in Chapter 5 that using (1.23) can actually accelerate the convergence rate of ALM. We will also see in Chapter 3 under which conditions the ALM is guaranteed to converge. Here we show using Lemma 1.2.3 that one condition for convergence of (1.17) is that ρ must be sufficiently large. Observe that for (1.17) to converge to a stationary-point, $f_\rho(x)$ must be bounded from below. By applying Lemma 1.2.3 with $P = \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*)$ and $Q = \nabla_x c(x^*) \nabla_x c(x^*)^T$ it

[1] Proofs for the lemmas can be found in Appendix B.

follows that given $\rho \geq \rho^*$ the Hessian of the Augmented Lagrangian $\nabla^2_{xx}\mathcal{L}_\rho(x^*, \lambda^*)$ is positive definite and consequently (1.17) locally convex in the neighborhood of x^*. Hence, provided that $\nabla_x c(x^*)$ is full row-rank because of our LICQ assumption, the solution of (1.17) exist and is unique if and only if $\rho \geq \rho^*$. An excellent discussion regarding the properties of the ALM can be found in [Bertsekas, 1999].

We summarize the standard ALM in Algorithm 2.

Algorithm 2: Augmented Lagrangian Method

1. Initialize the algorithm

Given penalty parameter $\rho > 0$; tolerance $\epsilon > 0$; estimates λ^0; maximum number of iterations k_{max}

Set the iteration index $k \leftarrow 0$

2. Update primal variables

$$(x^{k+1}) = \arg\min_{x \in \mathcal{X}} \mathcal{L}_\rho(x, \lambda^k)$$

2. Update dual variables

$$\lambda^{k+1} = \lambda^k + \rho \nabla_\lambda \phi(\lambda)$$

3. Check convergence

if $\|c^{k+1}\| \leq \epsilon$ **then exit, solution found.**

else Return to step **2**

1.3 Thesis Outline

Our goal is to develop decomposition algorithms and software tools to facilitate parallel solution of nonlinear programming problems with applications that arise in engineering problems in the form of stochastic and dynamic optimization. Therefore, this dissertation is organized into two parts.

The first part describes decomposition algorithms for NLP problems. We concentrate on Schur-complement and Augmented Lagrangian approaches that have proven to be very effective for solution of problems with millions of variables and constraints in a distributed, parallelizable manner. We start the discussion of decomposition methods in Chapter 2 with an overview of the Schur-complement

decomposition for solving structured optimization problems, along with the corresponding formulations for stochastic and dynamic optimization problems. We demonstrate the pros and cons of the Schur-complement method solving a case study of a dynamic parameter estimation problem in water distribution systems. We will see that while significant speedup is possible using this approach, it typically involves specialized parallel implementations of the interior-point algorithm that require considerable coding effort.

Chapter 3 proposes the use of augmented Lagrangian based approaches for general non-convex nonlinear structured optimization problems. These methods are attractive due to their flexibility and ease of implementation. Moreover, recent work in [Zhang and Kwok, 2014, Chang et al., 2016] proposes asynchronous augmented Lagrangian algorithms that can further exploit parallel computing environments. One drawback of these approaches, however, is that convergence is not well understood in the non-convex setting. Chapter 3 focuses on two of these approaches (the alternating direction method of multipliers and progressive hedging), and studies convergence from the perspective of the method of multipliers for which theory is well established in convex and non-convex problems. Benchmark metrics and chemical engineering applications of standard distillation and reactive systems are presented in Chapter 3.

In order to solve structured programs with a large number of coupling variables, Chapter 4 proposes an algorithm for solving structured KKT systems efficiently through iterative linear solvers like GMRES using ADMM as a preconditioner. This approach is unique in that the preconditioning technique is applicable for structured KKT systems in general. While preconditioners are typically designed for a particular application, the ADMM preconditioner is applicable to a large class of problems. Furthermore, the ease of implementation of ADMM facilitates designing an algorithm that calls ADMM as a functional preconditioner in every matrix-vector product of the GMRES algorithm. We assess the effectiveness of this approach using stochastic programming problems with a large number of

first-stage variables. For these problems the Schur-complement decomposition of Chapter 2 becomes intractable. The ADMM-GMRES method on the other hand solves the problem robustly in few iterations almost regardless of the number of coupling variables.

An important drawback of using augmented Lagrangian decomposition algorithms like ADMM and PH is their linear (or even sublinear) convergence rate. To accelerate ADMM we propose a second-order update strategy for the multiplier variables. Chapter 5 presents a derivation of the second-order update and compares the strategy with the Schur-complement decomposition. We perform several numerical experiments on stochastic programming problems to evaluate the effectiveness and robustness of the second-order update. Our results indicate that the second-order update is particularly beneficial for convex QPs. On one hand it significantly reduces the number of ADMM iterations. On the other hand it is remarkably insensitive to the choice of the penalty parameter. Furthermore, unlike the Schur-complement decomposition it does not require forming the Schur-complement matrix to update multiplier variables of the linking constraints. All these together makes the second-order ADMM strategy an excellent candidate for solving structured KKT systems that arise in the interior-point algorithm.

The second part of this dissertation describes `PyNumero`, a new open-source object-oriented programming framework for designing and prototyping general nonlinear optimization algorithms from Python. Typically, efficient optimization solvers are complex code bases written in low-level compiled programming languages. Their high performance computing features makes them very attractive languages for solving large-scale optimization problems. However, these low-level programming languages require significant software engineering expertise and a steep learning curve that slows development. `PyNumero` seeks to mitigate these challenges and aims to facilitate research of decomposition algorithms for nonlinear optimization problems. The package gives access to all high-level features of the Python programming language without making large sacrifices on computa-

tional performance. A description of the package together with examples of implementations of decomposition algorithms are presented in Chapter 6.

2. SCHUR-COMPLEMENT DECOMPOSITION

This chapter describes the Schur-complement method used for solving structured optimization problems with parallel computing machines. We will start with a general overview of algorithms that accelerate the solution of structured KKT systems arising in interior-point and sequential quadratic programming (active set) methods. For continuous nonlinear optimization problems, these two approaches have proven to be the most successful general purpose algorithms. Several interior-point implementations exist, including `Ipopt` [Wächter and Biegler, 2006], `Knitro`[Byrd et al., 2006b], and `LOQO` [Vanderbei, 1999]; SQP implementations include `Filter-SQP` [Fletcher et al., 2002], and `Snopt` [Gill et al., 2002]. Regardless of the choice of implementation, both interior-point and SQP methods have as their most computational expensive step the factorization of the KKT system. For this reason there has been active research on accelerating the solution of nonlinear optimization problems by focusing on speeding up the solution of the KKT.

For structured NLP problems, particularly those arising in stochastic programming and dynamic optimization, interior-point methods are preferable over active-set SQP methods. This is because in the interior-point method the structure of the KKT matrix remains the same at each iteration, making the development of tailored linear solvers convenient. Typically, these tailored approaches rely on the Schur-complement matrix to decompose the solution of the KKT system into smaller linear systems that can be solved in a distributed-paralellizable fashion. Examples of parallel interior-point solvers that exploit structure using the Schur-complement matrix are OOPS [Gondzio and Sarkissian, 2003], `Schur-Ipopt` [Kang et al., 2014], and `Pips-Nlp` [Chiang et al., 2014]. These solvers often exhibit excellent convergence rates and almost perfect scaling efficiency for problems with

small number of coupling variables. Unfortunately, one disadvantage of Schur-complement based approaches is that they require forming and solving the dense Schur system which makes them intractable for problems with large number of coupling variables. For this reason, in this chapter we discuss a Schur-complement decomposition that is appropriate for two-stage stochastic programming and dynamic optimization problems with low degree of coupling. Later in Chapter 4 we present an approach that can overcome the disadvantages of Schur-complement decomposition.

The remainder of this chapter is organized as follows: Section 2.1 provides an overview of different algorithms that parallelize the solution of KKT systems that arise in the interior-point methods. The overview includes algorithms for general unstructured KKT matrices as well as algorithms for block-structured KKTs. Sections 2.1.1 and 2.1.2 review specialized structures in stochastic programming and dynamic optimization that are parallelizable with the schur-complement decomposition. Section 2.2 concludes the chapter with a case study in water distribution systems. Numerical results are presented for a parallel implementation of the interior-point algorithm.

2.1 Overview of Internal Decomposition Algorithms

In order to parallelize interior-point and active-set SQP methods, numerous algorithms have been proposed. These algorithms are often referred to as internal decomposition approaches since they make modifications to a host algorithm (interior-point or SQP) with the aim of parallelizing it. Typically, these modifications target the solution of the KKT system because it is the dominant computation of Newton-based approaches. The modifications can be categorized in two classes: the first class is designed for general unstructured KKT systems and the second class for specialized structures that can benefit from tailored linear solvers.

Parallel algorithms for general unstructured NLPs focus on solving the KKT system with direct and iterative parallel linear solvers. Several parallel linear

solvers exist. Examples of parallel linear solvers used within an optimization context include MUMPS [Amestoy et al., 2001, 2006], PARDISO [Schenk and Gärtner, 2004] and MA86/MA97 [Hogg and Scott, 2010], WSMP [Gupta et al., 1998], and Elemental [Romero et al., 2012]. Amestoy et al. [2001] implemented a distributed memory approach that exploits sparsity and solves large systems through LDL or LU factorization. Schenk and Gärtner [2004] developed a parallel LU factorization method integrated into the PARDISO solver. Hogg and Scott [2010] developed a symmetric indefinite sparse direct solver within the HSL library. In addition to parallel direct factorization techniques, iterative methods for solving sparse matrices with parallel computing have also been developed. The PETSc library supports GPU implementations for Krylov subspace solvers [Balay et al., 1997, Abhyankar et al., 2018]. Among iterative solvers, the preconditioned conjugate gradient (PCG) has proven to be very efficient for solving general unstructured KKT systems [Li and Saad, 2013, Helfenstein and Koko, 2011, Buatois et al., 2009].

Real, large-scale, nonlinear optimization problems have often an inherent block structure in the KKT system that can be exploited to achieve higher parallelization efficiency. The work of Gondzio and Sarkissian [2003], Chiang et al. [2014], [Castro, 2007], and [Kang et al., 2014], has demonstrated that outstanding computational performances can be obtained when exploiting structure by using the Schur-complement matrix. We are interested in particular in structured KKT systems that arise in stochastic programming and dynamic optimization. We review now the block structures for both type of problems and study them within the context of Schur-complement decomposition. It is important to note, however, that despite we are concentration on the structure of these two types of problems, the decomposition algorithms described along this dissertation are applicable to structured optimization problems in general.

2.1.1 Stochastic Optimization and Schur-complement

Consider the two-stage stochastic program of the form

$$\min_{q\in\mathbb{R}^{n_q}} f_q(q) + E[Q(q,\kappa)] \tag{2.1.1}$$

$$\text{s.t. } c_q(q) = 0 \tag{2.1.2}$$

$$q_L \le q \le q_U \tag{2.1.3}$$

Here, $q \in \mathbb{R}^{n_q}$ is the vector of first stage variables, and $\kappa \in \mathbb{R}^{n_\kappa}$ the vector of uncertain parameters. The realization of uncertain parameters reveals in the second stage problem $Q(q,\kappa)$:

$$\min_{x_\kappa} f_\kappa(q, x_\kappa) \tag{2.2.1}$$

$$\text{s.t. } c_\kappa(q, x_\kappa) = 0 \tag{2.2.2}$$

$$x_{L_\kappa} \le x_\kappa \le x_{U_\kappa} \tag{2.2.3}$$

Here, x_κ is the vector of second stage variables. The objective function f_κ and the constraints c_κ depend on the realization of the uncertainty of κ. This realization is usually modeled with a finite number of scenarios with probabilities $\xi_1, ..., \xi_P$ where P is the number of scenarios. We denote the set of scenarios as $\mathcal{P} := 1, \cdots, P$. After discretizing the uncertainty space with a finite number of scenarios, $E[Q(q,\kappa)] = \sum_{i\in\mathcal{P}} \xi_s\, Q(q,\kappa_s)$, and problem (2.1) can be written in the following form:

$$\text{min } f_q(q) + \sum_{i\in\mathcal{P}} f_i(q, x_i) \tag{2.3.1}$$

$$\text{s.t. } c_q(q) = 0 \qquad (\lambda_q) \tag{2.3.2}$$

$$c_i(q, x_i) = 0 \qquad (\lambda_i), \quad i \in \mathcal{P} \tag{2.3.3}$$

$$q_L \le q \le q_U \qquad (\nu_q) \tag{2.3.4}$$

$$x_{L_i} \le x_i \le x_{U_i} \qquad (\nu_i), \quad i \in \mathcal{P}. \tag{2.3.5}$$

Here, x_i is the vector of second stage variables in scenario i, $\lambda_q \in \Re^{m_q}$ and $\nu_q \in \Re^{n_q}$ are the vectors of dual variables for the first stage, and $\lambda_i \in \Re^{m_i}$ and $\nu_i \in \Re^{n_i}$ are the vectors of dual variables in the second stage. We drop ξ_i and redefine the objective functions $f_i = \xi_i f_i \forall i \in \mathcal{P}$. To take advantage of the structure of the problem we introduce duplicates of the first stage variables in every scenario and rewrite the problem as follows:

$$\min\ f_q(A_1 x_1) + \sum_{i \in \mathcal{P}} f_i(x_i) \tag{2.4.1}$$

$$\text{s.t.}\ \ c_q(A_1 x_1) = 0 \qquad (\lambda_q) \tag{2.4.2}$$

$$c_i(x_i) = 0 \qquad (\lambda_i),\ \ i \in \mathcal{P} \tag{2.4.3}$$

$$A_i x_i - q = 0 \qquad (y_i)\quad i \in \mathcal{P} \tag{2.4.4}$$

$$q_L \leq q \leq q_U \qquad (\nu_q) \tag{2.4.5}$$

$$x_{L_i} \leq x_i \leq x_{U_i} \qquad (\nu_i)\quad i \in \mathcal{P}. \tag{2.4.6}$$

where x_i includes the second stage variables in (2.3) plus the duplicates of q, and A_i are linking matrices that extract the duplicates of q from each scenario. The equality and bound constraints previously applied on q only transfer to the first scenario to prevent redundant constraints. The *non-anticipativity* is enforced with constraint (2.4.5). Note that without (2.4.5), (2.4) can be decomposed into P subproblems

$$\begin{aligned} \min_{x_1}\ & f_q(A_1 x_1) + f_i(x_1) \\ \text{s.t.}\ \ & c_q(A_1 x_1) = 0 \\ & c_1(x_1) = 0 \\ & q_L \leq A_1 x_1 \leq q_U \\ & x_{L_1} \leq x_1 \leq x_{U_1} \end{aligned} \qquad \text{and,} \qquad \begin{aligned} \min_{x_i}\ & f_i(x_i) \\ \text{s.t.}\ \ & c_i(x_i) = 0 \\ & x_{L_i} \leq x_i \leq x_{U_i} \\ & \forall i \in \{2, \cdots, P\} \end{aligned}$$

The decomposable structure of (2.4) is inherited by the KKT system that can be permuted in an arrowhead form

$$\begin{bmatrix} K_1 & & & & B_1 \\ & K_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & K_P & B_P \\ B_1^T & B_2^T & \dots & B_P^T & K_q \end{bmatrix} \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \vdots \\ \Delta w_P \\ \Delta q \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_P \\ r_q \end{bmatrix}, \tag{2.7}$$

where,

$$\begin{aligned}
\Delta w_1^T &:= [\Delta x_1^T, \Delta \lambda_1^T, \Delta \lambda_q^T, \Delta y_1^T] \\
\Delta w_i^T &:= [\Delta x_i^T, \Delta \lambda_i^T, \Delta y_i^T] && \forall i \in \{2..P\} \\
r_q^T &:= \sum_{i \in \mathcal{P}} y_i \\
r_1^T &= -\left[\left(\nabla_{x_1} \mathcal{L}_1 + \nu_1 - (S_{L_1})^{-1} \mu e + (S_{U_1})^{-1} \mu e \right)^T, c_1^T, c_q^T, (A_1 x_1 - q)^T \right] \\
r_i^T &= -\left[\left(\nabla_{x_i} \mathcal{L}_i + \nu_i - (S_{L_i})^{-1} \mu e + (S_{U_i})^{-1} \mu e \right)^T, c_i^T, (A_i x_i - q)^T \right] && \forall i \in \{2..P\} \\
K_q &:= \begin{bmatrix} 0_{n_q} \end{bmatrix} \\
K_1 &:= \begin{bmatrix} D_1 & J_1^T & J_q^T & A_1^T \\ J_1 & & & \\ J_q & & & \\ A_1 & & & \end{bmatrix} && (2.8) \\
K_1 &:= \begin{bmatrix} D_i & J_i^T & A_i^T \\ J_i & & \\ A_i & & \end{bmatrix} \\
B_1 &:= \begin{bmatrix} 0 & 0 & 0 & -I \end{bmatrix} \\
B_i &:= \begin{bmatrix} 0 & 0 & -I \end{bmatrix} && \forall s \in \{2..S\} \\
D_i^k &= \nabla^2_{x_i x_i} \mathcal{L}_i^k + (S_{L_i}^k)^{-1} Z_{L_i}^k + (S_{U_i}^k)^{-1} Z_{U_i}^k && \forall i \in \{1..P\}
\end{aligned}$$

Here c_i=$c_i(x_i)$, $c_q = c_q(A_q x_1)$, J_i=$\nabla_{x_i} c_i(x_i)^T$, and J_q=$\nabla_{A_1 x_1} c_q(A_1 x_1)^T$. When all K_i blocks are nonsingular, block-Gaussian elimination brings (2.7) to a block-upper triangular form allowing for decomposition. The approach solves first for the first-stage variables using the Schur-complement matrix and then solve for the second-stage variables as follows:

$$\underbrace{(K_0 - \sum_{i\in\mathcal{P}} B_i^T K_i^{-1} B_i)}_{S}\Delta q = \underbrace{r_0 - \sum_{i\in\mathcal{P}} B_i^T K_i^{-1} r_i}_{r_s} \tag{2.9.1}$$

$$K_s \Delta w_i = r_i - B_i \Delta q, \quad \forall i \in \mathcal{P}. \tag{2.9.2}$$

If any K_i block is singular, inertia correction routines can be used to ensure nonsingularity of all blocks including S [Kang et al., 2014]. Furthermore, the inertia correction will no only ensure that the blocks are invertible, but ensures the requirements of the line-search filter method in Algorithm 1 are satisfied.

2.1.2 Dynamic Optimization and Schur-complement

Dynamic optimization seeks solution of models with time-changing variables and parameters that describe dynamic systems in the form of ordinary differential equations (ODEs), differential-algebraic equations (DAEs), or partial differential equations (PDEs). A dynamic optimization problem is commonly represented on a continuous time horizon $\mathcal{T} := [t_0, t_f] \subseteq \mathbb{R}$

$$\min \int_{t_0}^{t_f} \mathbf{\Phi}(s(t), u(t), w(t), \kappa)\, dt \tag{2.10}$$

$$\text{s.t. } F(\dot{s}(t), s(t), u(t), w(t), \kappa) = 0, \tag{2.11}$$

$$s(t_0) = s_0 \tag{2.12}$$

$$s_L \leq s(t) \leq s_U \tag{2.13}$$

$$u_L \leq u(t) \leq u_U \tag{2.14}$$

$$w_L \leq w(t) \leq w_U. \tag{2.15}$$

where $s \in \mathbb{R}^{n_s}$ are known as the state variables, $u \in \mathbb{R}^{n_u}$ are the control or manipulated variables, $w \in \mathbb{R}^{n_w}$ are the algebraic variables, $\kappa \in \mathbb{R}^{n_\kappa}$ the parameters, and $x(t) = (s(t), u(t), w(t), \kappa)$ is a composition vector containing all variables. The idea is to optimize some performance criterion Φ subject to a set of physical constraints F. In some situations, instead of optimizing performance of the system, one is interested in estimating the system parameters from observations made at time $t_k \in \mathcal{T} \ \forall \ k \in \{1, ..., M\}$ discrete points in time. These are called dynamic estimation problems. In Sections 2.2 we describe the particular problem of demand estimation in water distribution networks.

Direct methods have proven to be very effective for solving (2.10) [Biegler, 2010, Zavala, 2008]. Direct approaches transcribe the dynamic optimization problem (DOP) into a large-scale NLP. They convert the original infinite-dimensional problem into a discrete problem by fully or partially discretizing the dynamic equations. Depending on the sort of discretization, these methods are classified as sequential or simultaneous. When only the control signals are discretized, the methods are said to be sequential methods. The main advantage of using sequential methods lies in the fact that they generate smaller optimization problems than simultaneous methods. However, sequential methods have the disadvantage that the DAE must be integrated multiple times [Biegler, 2017]. Simultaneous methods, on the other hand, discretize both state and control variables and include the set of discretized equations as constraints in the optimization problem. While the simultaneous approach may result in optimization problems significantly larger than the ones obtained from the sequential approach, the DAE system is converged only once, simultaneously with the optimization problem, and this is potential for improved performance since it results in sparse and structured problems suitable for parallel computing.

In this dissertation we follow a simultaneous method and discretize all variables with an orthogonal collocation on finite elements scheme. The discretized optimization problem can now be written in the form

$$\min \sum_{i\in\mathcal{P}} f_i(x_i,\kappa) \tag{2.16.1}$$

$$\text{s.t. } \overline{A}_1 x_1 - s_0 = 0 \qquad (\lambda_{q_0}) \tag{2.16.2}$$

$$c_i(x_i,\kappa) = 0 \qquad (\lambda_i), \quad i\in\mathcal{P} \tag{2.16.3}$$

$$\underline{A}_i x_i - \overline{A}_{i+1} x_{i+1} = 0 \qquad (\underline{y}_i) \quad i\in\{1,\cdots,P-1\} \tag{2.16.4}$$

$$\overline{A}_{i-1} x_{i-1} - \overline{A}_i x_i = 0 \qquad (\overline{y}_i) \quad i\in\{2,\cdots,P\} \tag{2.16.5}$$

$$x_{L_i} \le x_i \le x_{U_i} \qquad (\nu_i) \quad i\in\mathcal{P}. \tag{2.16.6}$$

where $z_i = \left[\dot{s}_{i,1}, s_{i,1}, u_{i,1}, w_{i,1}, \cdots, \dot{s}_{i,n_c}, s_{i,n_c}, u_{i,n_c}, w_{i,n_c} \right] \forall\ i = 1,...,n_e$ is the vector of variables within finite element i; these are the discretized variables at collocation points $1,\cdots,n_c$. $\overline{A}_i$ and $\underline{A}_i$ are matrices that extract the state variables at the beginning and at the end of each finite element. For simplicity of the discussion we assume here each finite element is a partition block. However, we highlight that multiple finite elements can belong to one partition. It is important to note that only the state variables are temporally coupled between partitions, not the algebraic or control variables. As with the stochastic program, we introduce decoupling variables and the problem is rewritten as follows:

$$\min \sum_{i\in\mathcal{P}} f_i(x_i,\kappa) \tag{2.17.1}$$

$$\text{s.t. } \overline{A}_1 x_1 - s_0 = 0 \qquad (\lambda_{q_0}) \tag{2.17.2}$$

$$c_i(x_i,\kappa) = 0 \qquad (\lambda_i), \quad i\in\mathcal{P} \tag{2.17.3}$$

$$\underline{A}_i x_i - q_i = 0 \qquad (\underline{y}_i) \quad i\in\{1,\cdots,P-1\} \tag{2.17.4}$$

$$\overline{A}_i x_i - q_{i-1} = 0 \qquad (\overline{y}_i) \quad i\in\{2,\cdots,P\} \tag{2.17.5}$$

$$x_{L_i} \le x_i \le x_{U_i} \qquad (\nu_i) \quad i\in\mathcal{P}. \tag{2.17.6}$$

The KKT system for (2.17) can be written in an arrowhead form similar to that in (2.7). Like with (2.4), using the Schur-complement decomposition can leverage the structure in (2.17) facilitating the implementation of parallel linear solvers. However, unlike in (2.3), the number of coupling variables in (2.17) increases as the number of block-partitions grows making the approach more restrictive for dynamic problems. The relation for the number of coupling variables and partitions is given by $n_q = n_s * (P-1)$. Differently than (2.4), the linking between partitions do not involve all coupling variables. This is because decomposing the temporal domain only requires linking neighboring partitions which is evidenced in the linking constraints (2.17.5) and (2.17.6). The block matrices to write the KKT system of (2.17) in the block-arrowhead form (2.7) are:

$$
\begin{aligned}
\Delta w_1^T &:= [\Delta x_1^T, \Delta\lambda_1^T, \Delta\lambda_{q_0}^T, \Delta\overline{y}_1^T] \\
\Delta w_i^T &:= [\Delta x_i^T, \Delta\lambda_i^T, \Delta\overline{y}_i^T, \Delta\underline{y}_i^T] && \forall i \in \{2..P-1\} \\
\Delta w_P^T &:= [\Delta x_P^T, \Delta\lambda_P^T, \Delta\overline{y}_P^T] \\
r_q^T &:= \sum_{i=1}^{P-1} \underline{y}_i + \sum_{i=2}^{P} \overline{y}_i \\
r_1^T &= -\left[\left(\nabla_{x_1}\mathcal{L}_1 + \nu_1 - (S_{L_1})^{-1}\mu e + (S_{U_1})^{-1}\mu e\right)^T, c_1^T, (\overline{A}_1 x_1 - s_0)^T, (\underline{A}_1 x_1 - q_1)^T\right] \\
r_i^T &= -\left[\left(\nabla_{x_i}\mathcal{L}_i + \nu_i - (S_{L_i})^{-1}\mu e + (S_{U_i})^{-1}\mu e\right)^T, c_i^T, \left(\overline{A}_i x_i - q_{i-1}\right)^T, (\underline{A}_i x_i - q_i)^T\right] \\
&\quad \forall i \in \{2..P-1\} \\
r_P^T &= -\left[\left(\nabla_{x_P}\mathcal{L}_P + \nu_P - (S_{L_P})^{-1}\mu e + (S_{U_P})^{-1}\mu e\right)^T, c_P^T, \left(\overline{A}_P x_P - q_{P-1}\right)^T\right] \\
K_q &:= \left[0_{n_q}\right] \\
K_i &:= \begin{bmatrix} D_i & J_i^T & \overline{A}_i^T & \underline{A}_i^T \\ J_i & & & \\ \overline{A}_i & & & \\ \underline{A}_i & & & \end{bmatrix} \quad \forall i \in \{2..P-1\}
\end{aligned}
$$

$$K_P := \begin{bmatrix} D_P & J_P^T & \underline{A}_P^T \\ J_P & & \\ \underline{A}_P & & \end{bmatrix}$$

$$B_1 := \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ -I & 0 & \cdots & 0 \end{bmatrix}$$

$$B_i := \begin{bmatrix} 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & -I & \cdots & 0 \\ 0 & \cdots & \cdots & -I & 0 \end{bmatrix} \qquad \forall i \in \{2..P-1\}$$

$$B_P := \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & -I \end{bmatrix}$$

$$D_i^k = \nabla^2_{x_i x_i} \mathcal{L}_i^k + (S_{L_i}^k)^{-1} Z_{L_i}^k + (S_{U_i}^k)^{-1} Z_{U_i}^k \qquad \forall i \in \{1..P\}$$

The Schur-complement decomposition provides then a methodology to leverage block-structure in the KKT system of dynamic and stochastic optimization problems, making it possible to solve the linear system in a distributed-paralellizable manner. In fact, the decomposition approach is applicable to general structures. Note that for both types of problems here the structure of the optimization application is embedded within the A and B matrices, and the steps for the schur-complement decomposition remain the same. The KKT system can be solved with 3 steps. The first step is to form S and r_s by adding the contribution from each block. This step requires the factorizations of each individual K_i. The second step is to solve the Equation (2.9.1) to get Δq. This step requires one factorization and

one backsolve of S. With Δq, the third step is to compute Δw_i from Equation (2.9.2). The approach is summarized in Algorithm 3.

Algorithm 3: Schur-complement decomposition for Structured KKTs

foreach i *in* $1, \cdots, N$ **do**
    factor K_i (Using MA27 subroutines)
Initialize S by letting $S = 0$
Let $r_{sc} = r_s$
foreach i *in* $1, \cdots, N$ **do**
    foreach *nonzero column* j *in* A_i^T **do**
        solve the system $K_i s_i^{<j>} = [B_i^T]^{<j>}$ for $s_i^{<j>}$
        let $S^{<j>} = S^{<j>} - B_i s_i^{<j>}$
    solve the system $K_i p_i = r_i$ for p_i
    let $r_{sc} = r_{sc} - B_i p_i$
solve $S\Delta v_s = r_{sc}$ for Δv_s
foreach i *in* $1, \cdots, N$ **do**
    solve $K_i \Delta v_i = r_i - B_i^T \Delta v_s$ for Δv_i

In addition to parallel linear solvers, scalable parallel interior-point algorithms require the parallelization of all NLP functions and gradient evaluations. Furthermore, all linear algebra operations must be parallelized. These include vector-vector, matrix-vector and matrix-matrix products. To the best of our knowledge, there is currently no modeling framework that facilitates all these features in an intuitive manner. For structured optimization problems, like those presented in Sections 2.1.1 and 2.1.2, Chiang et al. [2014], Kang et al. [2014], and Word [2014] implemented C++ parallel interior-point solvers that determine the structure using AMPL suffixes. In Chapter 6 we discuss a python initiative that seeks to facilitate the implementation of parallel interior-point algorithms. Here, we follow the approach of Word [2014] and solve a case study from water distribution systems with a parallel interior-point solver based on suffixes.

2.2 Case Study: Demand Estimation in Water Distribution Systems

Reliable modeling of dynamic systems frequently requires accurate knowledge of inputs and parameters to be computed from system measurements. In this section, we study a parallel interior-point approach for dynamic optimization applied to inverse problems in water distribution systems. We make use of a Schur-complement decomposition in the linear solution of the KKT system to solve dynamic optimization problems in parallel. This approach is known to be particularly advantageous for problems with few states but many algebraic variables. Water distribution networks typically include thousands of junctions and pipes, but relatively few storage tanks, resulting in dynamic systems with precisely these properties. We explore the convenience of the parallel algorithm for solving a real-time water demand estimation problem. We formulate an optimization model for the estimation of water demand values from flow and pressure data. The formulation considers different robust estimators to mitigate the effect of gross measurement errors, and a regularization term to deal with insufficient data. We tested the formulation and the parallel algorithm on a midsize network with 3358 nodes and 3892 links. To resemble a real water network, which has hundreds of thousands of pipes and nodes, we increased the dimension of the problem by considering a 5 day horizon with 5, 10, and 20 minutes hydraulic time steps, resulting in problems with approximately 12, 6 and 3 million variables. The problem was solved with our parallel interior-point solver on a shared memory-machine with up to 16 cores and the solution time was reduced by a factor of 10.

2.2.1 Optimization Formulation

The problem of demand estimation in water distribution systems can be formulated as a dynamic optimization problem. The transcribed NLP obtained after discretization of the dynamics is presented in (2.18).

$$\min_{\mathrm{Q},\mathcal{H},\mathcal{D}} \quad \sum_{t\in\mathcal{T}_m}\left(\sum_{n\in\mathbb{N}_m} f_m(\mathcal{H}_{n,t}-\mathcal{H}^{\mp}_{n,t})+\sum_{l\in\mathbb{L}_m} f_m(\mathrm{Q}_{l,t}-\mathrm{Q}^{\mp}_{l,t})\right)+\sum_{t\in\mathcal{T}}\sum_{n\in\mathbb{N}} f_r(\mathcal{D}_{n,t}-\mathcal{D}^{*}_{n,t})$$

$$\begin{aligned}
\text{s.t.} \quad & \frac{\mathcal{H}_{n,t}-\mathcal{H}_{n,t-1}}{\Delta t}=\frac{4}{\pi d_n^2}\sum_{l\in\mathbb{L}_n}\mathrm{Q}_{l,t} && \forall n\in\mathbb{T}, t\in\mathcal{T}\\
& \sum_{l\in\mathbb{L}_n}\mathrm{Q}_{l,t}=\mathcal{D}_{n,t} && \forall n\in\mathbb{N}\setminus\mathbb{T}, t\in\mathcal{T}\\
& \mathcal{H}_{n_1,t}-\mathcal{H}_{n_2,t}=r_l\mathrm{Q}^{\nu}_{l,t}+m_l\mathrm{Q}^{2}_{l,t} && \forall l\in\mathbb{L}, t\in\mathcal{T}\\
& \mathcal{D}_{n,t},\mathcal{H}_{n,t}\geq 0 && \forall n\in\mathbb{N}, t\in\mathcal{T}\\
& \mathrm{Q}_{l,t}\geq 0 && \forall p\in\mathbb{P}, t\in\mathcal{T}
\end{aligned} \tag{2.18}$$

Here $\mathrm{Q}_{l,t}$ represents the volumetric flow rate at link l and time t, $\mathcal{D}_{n,t}$ is the water demand out of node n at time t, $\mathcal{H}_n$ is the pressure head at node n (or water level in the case of tanks), $\mathbb{N}$ is the set of all nodes in the network, $\mathbb{T}$ the set of all tanks, $\mathbb{L}$ is the set of all links, $\mathbb{L}_n \subseteq \mathbb{L}$ is the set of all links connected to node n, $\mathbb{P} \subset \mathbb{L}$ is the set of all pumps, $\mathcal{T}$ is the set of times and $\mathcal{T}_m$ is the set of measure times, $\mathbb{N}_m$ is the set of measured nodes, and $\mathbb{L}_m$ the set of measured links. The parameters r_l and m_l are the respective resistant and minor loss coefficients, and ν is the flow exponent. f_m and f_r are estimator functions that will be described later. The demand estimation is constrained with material and momentum balance equations. The first and second constraints ensure mass conservation at each of the nodes $n \in \mathbb{N}$, while the third constraint enforces momentum conservation at each link $l \in \mathbb{L}$. Note that the water level in the tanks $\mathcal{H}_n \ \forall \ n \in \mathbb{T}$ are the state variables. The volumetric flows in the links $\mathrm{Q}_l \ \forall \ l \in \mathbb{L} \setminus \mathbb{P}$ and the pressure heads at each node $\mathcal{H}_n \ \forall \ n \in \mathbb{N} \setminus \mathbb{T}$ are the algebraic variables.

The proposed formulation minimizes the error between flow and pressure measurements $\mathrm{Q}^{\mp}_{n,t}$, $\mathcal{H}^{\mp}_{n,t}$ (taken at discrete times $t \in \mathcal{T}_m$) and the values predicted with the model $\mathrm{Q}_{n,t}, \mathcal{H}_{n,t}$. The measurements provide information to determine the demand values at each node of the system. However, because of the size of water

distribution networks, the number of measurements is likely to be sparse, making the problem ill posed. Therefore, an additional term can be included in the objective function to deal with the ill conditioning of the problem. In our formulation (2.18), we included a simple regularization term in the objective function to take into account the shortage of measurements.

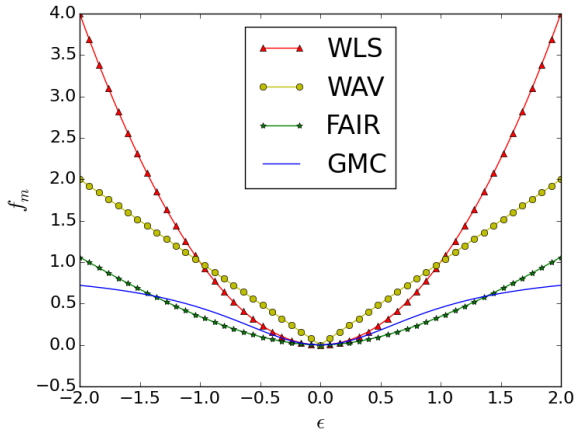
In water distribution networks, gross error can arise from pipe breaks, pump outages, improper use of measuring devices, and random sources introduced by operators of the network. To automatically handle outliers and potential gross errors in the data, we consider different estimators $f_m : \mathbb{R} \to \mathbb{R}$ in the objective function of our formulation. When data is likely to be accurate, we propose using a standard weighted least-squares estimator (WLS). However, in cases where there are possible gross errors in the available data, we propose using robust estimators. Robust estimators make it easy to identify outliers without having to perform any analysis of the data beforehand, but may significantly reduce the effect of outliers and yield less biased estimates [Arora and Biegler, 2001, Llanos et al., 2015]. In our implementation we considered the 4 different estimators presented Table 2.1.

To solve the nonlinear problem (2.18) in serial, we used the NLP solver `Ipopt` [Waechter and Biegler, 2006] and our own implementation of an interior-point algorithm described in Section 1.2.1. In order to solve the problem in multiple processors and exploit the block-structure inherited from the discretization step, we use a Schur-complement decomposition that not only allows us to parallelize the solution of the KKT system, but also compute derivative information and residuals of individual blocks in parallel.

2.2.2 Results and Discussion

The demand estimation formulation was implemented in `Pyomo` [Hart et al., 2012b, 2011a]. `Pyomo` is a Python-based algebraic modeling language that supports the definition and solution of optimization problems using the interpreter language Python. The flexibility provided by Python and `Pyomo` lets us imple-

Table 2.1.: Estimator functions $f_r(\epsilon)$ and $f_m(\epsilon)$. Four estimators are consider in the formulation (2.18). Weighted least squares (WLS), Weighted absolute values (WAV), the fair estimator (FAIR) with $c = 1.389$, and German McClure (GMC). The residual in the estimate is $\epsilon = (p - p^{\mp})$.

Estimator	Function $f_m(\epsilon)$	Estimator Shape
WLS	ϵ^2	
WAV	$\lvert\epsilon\rvert$	
FAIR	$c^2(\frac{\lvert\epsilon\rvert}{c} + \ln(1 + \frac{\lvert\epsilon\rvert}{c}))$	
GMC	$\frac{1}{2}\left(\frac{\epsilon^2}{1+\epsilon^2}\right)$	

ment a framework that can take different water network models. In this study, we present the demand estimation results for a midsize network with 3358 nodes, 3892 links, 61 pumps, 3 valves, and 34 tanks. To investigate the behavior of robust estimators for water demand estimation, we first solved the estimation problem for a set of scenarios with randomly selected outliers in the set of measurements. The error of the estimation is quantified as the sum of squared relative errors:

$$\text{SSRE} = \sum_{t \in \mathcal{T}} \sum_{n \in \mathcal{N} \setminus \mathbb{T}} \frac{(\mathcal{D}_{n,t} - \mathcal{D}_{n,t}^{true})^2}{(\mathcal{D}_{n,t}^{true})^2}$$

Figure 2.1 shows the results as the number of outliers increases. As expected, both FAIR and GMC obtained less biased estimates in comparison to WLS and WAV. Note that among the four estimators, GMC always gave the estimation with smaller SSRE. However, while the number of interior-point iterations remained almost constant for the other 3 estimators, when solving the problem with GMC, an increase in the number of outliers resulted in an increase in the number of itera-

tions. The estimation problem with WLS and FAIR converged in 15 iterations no matter the number of outliers. WAV took about 62 iterations to converge. Because the derivative of WAV is not a continuous function, WAV was implemented as two linear underestimators or inequality constraints. Hence, for each estimate in the objective function, two inequality constraints had to be added to the problem. This caused the interior-point solver to take longer to converge when solving the problem with WAV. GMC, on the other hand, converged in 18 iterations for a small number of outliers. However, as the number of outliers increased, the computation of the search direction in the interior-point iterations became more expensive, as inertia correction of the KKT system was required to ensure a descent direction. Since continuous inertia correction in the interior-point iterations degrades the parallel performance of the Schur-complement parallel algorithm, we decided to work only with the WLS and FAIR estimators.

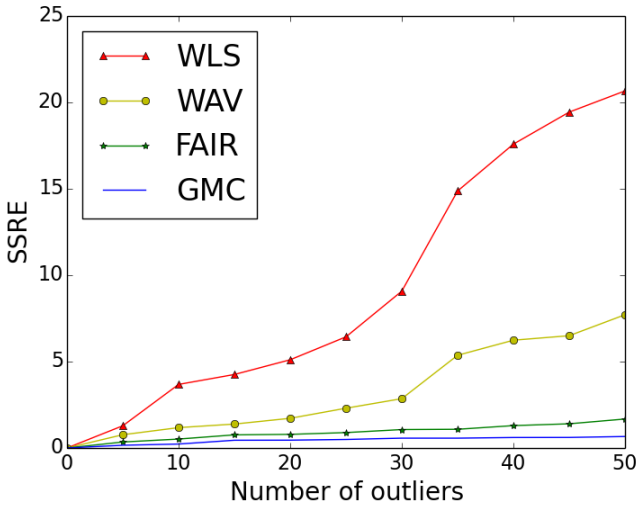
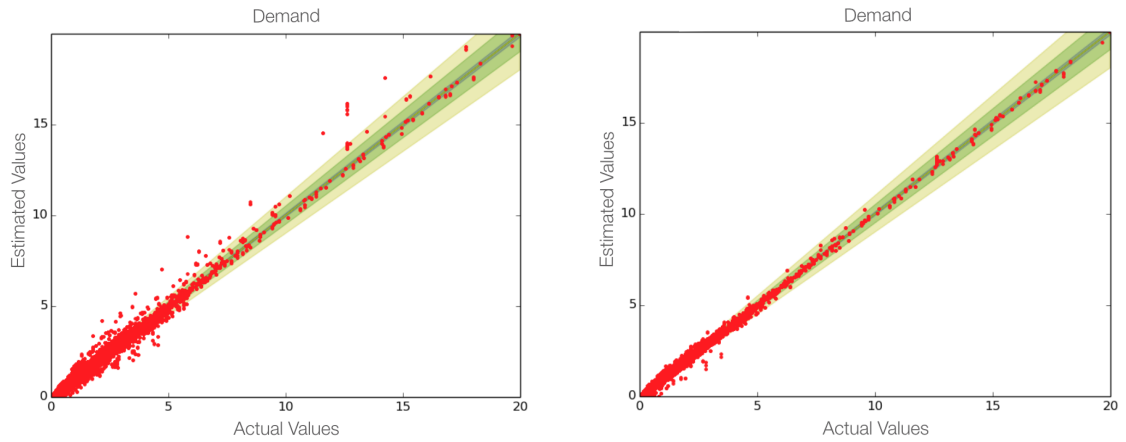


Figure 2.1.: Impact of the number of outliers for different estimators.

For this case study, we mimic the behavior of real measurements by adding normally distributed random noise to simulated data. Flow measurements were taken at 195 randomly selected links and 10% noise was added. Tank levels were also measured but only 0.1% noise was added since tank level measurements are

expected to be highly accurate. Noise was added following a normal distribution with mean value equal to zero and standard deviation equal to some percentage of the average value of the variable over all times. Results for the demand estimation in serial with the WLS and FAIR estimators are presented in Figure 2.2. Figures 2.2(a) and 2.2(b) show the estimated demand values at every node and time against the respective true values (simulated). The color gradient shows the regions of 1%, 5% and 10% error. Notice that most of the values are within a 5% error margin when using the FAIR estimator, while, in the case of the WLS, the range of error is larger because it does not detect outliers.



(a) WLS estimation of demands against true demands

(b) FAIR estimation of demands against true demands

Figure 2.2.: Correspondence between the modeled (predicted) and true (measured) demands

With confidence in the estimation formulation and the results obtained for the serial solution, we now want to test the performance of the parallel algorithm. All benchmark tests of the parallel algorithm were performed on a Linux OS server with 24 cores, 264 GB of DDR3 RAM and a clock speed of 2.6 GHz. We tested the performance of the parallel algorithm on 3 different sizes of problems. To resemble a real water network, which has hundreds of thousands of pipes and nodes, we increased the dimension of the problem by considering a 5 day horizon

with 5, 10, and 20 minutes hydraulic time steps. Figure 2.3 shows the speedup of the parallel implementation when compared with the serial implementation of the Schur-complement decomposition algorithm. Since both FAIR and WLS produced similar timing results, we show here only the results for WLS. While the parallel implementation shows significant speedup factors, as expected, the performance of the parallel implementation seems to deteriorate as the number of blocks increases. Ideally, as we double the number of blocks we should reduce the computation time by half since we provide twice the number of processors. However, in Figure 2.3 we see that this only happens when we have two blocks. To investigate further, we separately timed the relevant steps of the Schur-complement decomposition algorithm. Table 2.2 details the average time per iteration required in the computational steps described in the algorithm presented in Section 2.2.1.

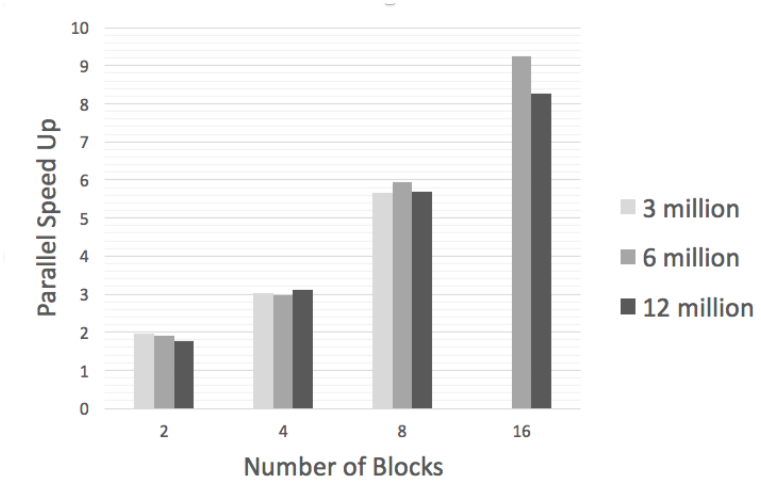


Figure 2.3.: Speedups comparing the parallel Schur-complement linear solver with the serial Schur-complement linear solver for problems of three different sizes. Problem sizes correspond to a time horizon of 5 days with hydraulic time step 20,10 and 5 minutes. The problem with 3 million variables was not solved with 16 cores because the time horizon could not be divided in 16 blocks with integer number of time steps.

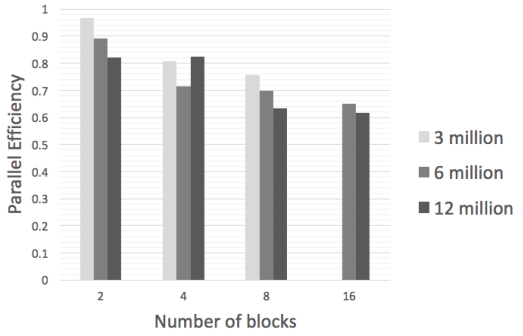
The first two steps of the parallel algorithm are the most computationally expensive. Determining the step in the coupling variables and performing backsolves to obtain the step in uncoupled variables takes less time in comparison with

Table 2.2.: Average time per iteration required in the steps of the Schur-complement algorithm. The four steps of the algorithm are summarized as Block factorization (lines 1-2), form Schur-complement (lines 5-10), solve Schur-complement(line 12-13) and solveDelx (lines 15-17). Three different size of problems were considered.

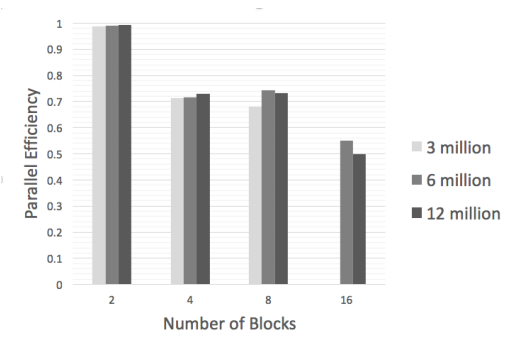
Schur-Complement Times (s)									
Approx N variables	N Blocks	Block factorization		Form SC		Solve SC		Solve Delx	
		Serial SC	Parallel SC	Serial SC	Parallel SC	Serial SC	Parallel SC	Serial SC	Parallel SC
	2	7.4811	3.8706	12.6406	6.3956	0.0002	0.0002	0.3642	0.1844
	4	5.0129	1.5526	17.3731	6.0844	0.0007	0.0008	0.3314	0.0920
3 million	8	3.5388	0.5834	18.0844	3.3199	0.0019	0.0020	0.2875	0.0492
	2	19.2307	10.7759	27.5293	13.8915	0.0002	0.0002	0.7921	0.3991
	4	15.2653	5.3447	37.7005	13.1827	0.0008	0.0008	0.7196	0.1949
	8	10.5001	1.8793	42.2293	7.1067	0.0019	0.0021	0.6631	0.1039
6 million	16	6.9960	0.6713	42.2920	4.8053	0.0046	0.0053	0.5709	0.0609
	2	78.9631	48.0925	54.6044	27.4769	0.0002	0.0002	1.5693	0.7899
	4	55.0606	16.6994	79.8769	27.3931	0.0008	0.0008	1.5119	0.3978
	8	32.7831	6.4781	86.8106	14.8138	0.0019	0.0021	1.3769	0.2069
12 million	16	22.3681	2.2700	91.6838	11.4944	0.0042	0.0050	1.2588	0.1342

factoring the blocks and constructing the Schur-complement. This is consistent for all problem sizes and different numbers of blocks. Note that as the number of blocks increases, the blocks become smaller, and consequently the factorization step becomes faster even in the serial case. However, as the number of blocks increases, more computations are required to form the Schur-complement and thus the serial implementation takes longer. On the other hand, the parallel construction of the Schur-complement is distributed on separate processors, so the time decreases if there is no inter-processor communication overhead, which is required when forming the Schur-complement and factoring the blocks. To improve performance of these two steps the communication operations must be minimized. Figures 2.4(a) and 2.4(b) show the parallel efficiency of the first two steps of the algorithm.

In both of the Figures, we observe higher parallel efficiency for fewer number of blocks. However, as the number of blocks goes from 2 to 16, the efficiency of the block factorization decreases from 90% to 65%. This may be due to two reasons. The inertia of each block K_i is computed in the factorization step. If the inertia of any block is not satisfied, then the entire system needs to be corrected. This



(a) Parallel efficiency of block factorization step



(b) Parallel efficiency of Schur- complement construction

Figure 2.4.: Parallel efficiency of dominant steps in the Schur-complement based parallel algorithm.

requires inter-processor communication and may cause the decrease in parallel efficiency shown in Figure 2.4(a). The second reason can be related to differences in the K_i blocks. When blocks are very different, it may take longer to factorize some blocks than for others. This may result in load imbalance which degrades the parallel speedup. Observe that in general an increase in problem size leads to lower parallel efficiency. This may be related to race-conditions (two processors try to access data at the same time) since the tests were performed in a shared memory machine. Note that in the construction of the Schur-complement this phenomena does not happen, since the sparse vector multiplications performed for constructing the Schur-complement required less memory. However, lower efficiency is also seen as the number of blocks increases. It is worth mentioning that the scalability issues seem to be implementation related. If there are no race-conditions or load imbalances forming the Schur-complement complement in parallel should scale linearly if the ratio between coupled and uncoupled variables is small. In order to improve the scalability of the parallel implementation further optimizations need to be done in the code.

2.3 Summary

In this chapter we have presented an overview of the Schur-complement decomposition. We introduced formulations for stochastic and dynamic optimization and presented a general algorithm that allows for parallelization of the interior-point algorithm presented in Chapter 1. The implementation of the decomposition strategy was illustrated throughout a benchmark problem in water distribution systems. A demand estimation problem was formulated and solved following the parallel algorithm. We proposed a robust optimization formulation that reduces the effect of potential gross errors in measurement values [Llanos et al., 2015, Arora and Biegler, 2001]. To test our formulation, the demand consumption values of a midsize water network with 3358 nodes, 3892 links were estimated. We solved the problem using both serial and parallel implementations of the interior-point algorithm. Significant speedup was obtained. For a problem with over 6 million variables and 5 million constraints, the solution time was reduced by a factor of 10.

Exploitation of structure of optimization problems at the KKT level using the Schur-complement matrix benefits from the convergence properties of the interior-point algorithm. However, implementations of this approach are intrusive and require significant changes to the host algorithm in order to achieve favorable scalability results. These include modifications to the linear solver, function evaluations, derivative computations and all matrix operations. In the next chapter we discuss a different decomposition paradigm that partitions the structure of the optimization problem at the formulation level. We refer to this paradigm as external decomposition. The external approach is less intrusive and easier to implement, but it's convergence rate and robustness are typically less favorable. However, we will see in Chapters 4 and 5 that combining ideas from both approaches overcomes limitations of each separate approach.

3. CONVERGENCE OF AUGMENTED LAGRANGIAN DECOMPOSITION METHODS [1]

This chapter studies convergence properties of external decomposition algorithms. In particular we focus on augmented Lagrangian based approaches. We study connections between the alternating direction method of multipliers, the classical method of multipliers (MM), and progressive hedging schemes. The connections are used to derive benchmarking metrics and strategies to monitor and accelerate convergence and to help explain why ADMM and PH are capable of solving complex (but structured) nonconvex NLPs. Specifically, we observe that ADMM is an inexact version of MM that approaches its performance when multiple coordination steps are performed (thus inheriting its convergence properties). In addition, we note that PH is a specialization of ADMM and use Lyapunov function and primal-dual feasibility metrics used in ADMM to explain why PH is capable of solving nonconvex NLPs and to highlight that PH can be used to tackle a wider range of stochastic programming problems and even other problem classes that arise in machine learning, estimation, and dynamic optimization. We illustrate the concepts using challenging dynamic optimization problems. Our exposition is tutorial in nature and seeks to establish more formalism in benchmarking ADMM, PH, and AL schemes and to motivate algorithmic improvements.

[1]Part of this section is reprinted with permission from
"Benchmarking ADMM in nonconvex NLPs" by Rodriguez, Jose S. and Nicholson, Bethany and Laird, Carl and Zavala, Victor M., 2018. Computers and Chemical Engineering, Copyright 2018 by Elsevier.

3.1 Preliminaries

Since the introduction of the method of multipliers by Hestenes and Powell, extensive research has been done in the field of augmented Lagrangian (AL) methods [Bertsekas, 1976, Mangasarian, 1975, Miele, 1971b,a, O'Doherty and Pierson, 1974, Rockafellar, 1973]. Betsekas and Rockafellar have analyzed AL methods using dual theory to show that the minimization of the AL eliminates the duality gap [Bertsekas, 1999, Rockafellar, 1974]. Extensions by Fletcher, Byrd, Tapia and others have been aimed at accelerating local and global convergence by identifying and updating penalty parameters and performing second order multiplier updates [Betts, 1977, Byrd, 1978, Fletcher, 1975, Glad, 1979, Rupp, 1975, Tapia, 1977]. All these developments have lead to powerful AL optimization codes such as `LANCELOT` and `ALGENCAN` [Birgin and Martinez, 2014, Conn et al., 2013] and extended AL formulations [Di Pillo and Grippo, 1989, Hager, 1987, Huang and Yang, 2003, Li, 1995, 1997, Zavala and Anitescu, 2014].

The convergence properties of AL methods in convex and nonconvex nonlinear programs (NLPs) are well understood. Local convergence of the AL method was analyzed in [Mangasarian, 1975, Rockafellar, 1973]. In [Huang and Yang, 2005] it is shown that first and second-order optimality conditions of the AL converge to those of the original NLP. Global convergence for convex problems is discussed in [Bertsekas, 1982, Polyak, 2004, Rockafellar, 1976] and for nonconvex problems in [Luo et al., 2008, Wang and Li, 2009]. Convergence properties of AL schemes in semidefinite programs was studied in [Sun et al., 2006, 2008].

The AL function and associated duality theory has also motivated the development of specialized decomposition schemes. Specifically, in the context of two-stage stochastic programming, Rockafellar and Wets introduced a powerful and practical decomposition method known as progressive hedging (PH) that uses proximal minimization of the AL function [Rockafellar and Wets, 1991]. Bertsekas and Eckstein combined Douglas-Rachford splitting, AL, and proximal point meth-

ods to derive what is now known as the alternating direction method of multipliers (ADMM) [Eckstein and Bertsekas, 1992b]. ADMM is a highly flexible distributed optimization scheme that targets block-structured problems. In addition to potential improvements in computational time over centralized MM schemes, ADMM decomposition facilitates distribution of memory requirements, enables data privacy, and facilitates the use of a wide range of computing architectures that might exhibit memory and computing speed constraints. This flexibility is beneficial in many applications such as stochastic programming, decentralized control and estimation, embedded and network optimization, and machine learning [Boyd et al., 2011].

The convergence properties of ADMM and PH in convex NLPs is well established but convergence guarantees in nonconvex settings are still poorly understood. In practice, however, ADMM and PH often perform satisfactorily in complex nonconvex NLPs. In this work, we study connections between MM, ADMM, and PH to derive benchmarking metrics that explain why PH and ADMM work in practice. For instance, structural analysis reveals that ADMM is an inexact version of MM and that PH is an specialization of ADMM to stochastic programs. These connections are, surprisingly, not commonly known. Our exposition is tutorial in nature and seeks to motivate the development of algorithmic improvements and to widen the scope of applications for ADMM and PH.

The chapter is structured as follows: Section 3.2 introduces notation and states the problem definition. In Section 3.3.1 we present an overview of the method of multipliers as a basis for understanding ADMM and PH strategies. Sections 3.3.2 and 3.3.3 review ADMM and draws connections between MM, ADMM, and PH. Section 3.4 describes metrics to asses the performance of the different schemes. In Section 3.5 we present case studies to illustrate our developments. Both case studies considered nonconvex dynamic optimization problems that result from typical chemical engineering applications. Section 3.6 closes the chapter with concluding remarks and directions for future work.

3.2 Problem Definition and Setting

This work focuses on structured NLPs of the form:

$$\min_{x_i,q} \quad \sum_{i\in\mathcal{P}} f_i(x_i) \tag{3.1.1}$$

$$\text{s.t.} \;\; c_i(x_i) \geq 0, \; i \in \mathcal{P} \tag{3.1.2}$$

$$A_i x_i + B_i q = 0, \; (y_i) \quad i \in \mathcal{P}. \tag{3.1.3}$$

where the vector $x_i \in \mathbb{R}^{n_{x_i}}$ contains all the variables corresponding to a block partition $i \in \mathcal{P} := \{1, ..., N\}$. We define the entire set of partition variables as $x = (x_1, ..., x_{|\mathcal{P}|})$. The vector $q \in \mathbb{R}^{n_q}$ contains the complicating (coupling) variables. The linking constraints (3.1.3) are defined using the mapping matrices A_i and B_i. The Lagrange multipliers of the coupling constraints are denoted as $y_i \in \mathbb{R}^{m_i}, \; i \in \mathcal{P}$ and we define the entire set of multipliers as $y = (y_1, ..., y_{|\mathcal{P}|})$. We denote the i-th entry of a given vector v as $v(i)$.

For convenience, we write the above NLP in the following compact form:

$$\min_{x\in\mathcal{X},q} \quad f(x) \tag{3.2.1}$$

$$\text{s.t.} \;\; Ax + Bq = 0 \quad (y) \tag{3.2.2}$$

where $\mathcal{X} = \{x \,|\, c(x) \geq 0\}$ and $c(x) = (c_1(x), ..., c_{|\mathcal{P}|}(x))$. We define the sets $\mathcal{X}_i = \{x \,|\, c_i(x_i) \geq 0\}$ and note that $\cap_{i\in\mathcal{P}}\mathcal{X}_i = \mathcal{X}$. Matrices A and B can be easily constructed using the partition matrices A_i and B_i.

The structured problem (3.1) arises in different application domains such as stochastic programming, learning and estimation problems, multi-stage optimal control, and network problems [Biegler, 2017, Kang et al., 2014, Zavala et al., 2008, Word, 2014, Gondzio and Grothey, 2009]. This structure also arises in dynamic optimization problems where the coupling variables correspond to state variables of neighboring partitions.

Two different decomposition paradigms have been proposed in the literature to exploit the structure of (3.1) [Mung Chiang et al., 2007, Kang et al., 2015]. The first paradigm, used in ADMM and PH, decomposes the problem at a formulation level. Here, the overall problem is decomposed into partition subproblems whose solutions are coordinated by a master problem to find the solution of the overall problem. This approach is frequently referred to as external-decomposition. The second paradigm targets a host algorithm and decomposes the problem at the linear algebra level. This paradigm is also known as internal-decomposition. The Schur-complement decomposition described in Chapter 2 is an example of an internal-decomposition scheme. External-decomposition approaches are less intrusive, more flexible, require less communication, and are easier to implement but exhibit slow convergence. Internal-decomposition approaches are more intrusive, require more communication, and are more difficult to implement but have faster convergence. In this chapter we focus on external decomposition algorithms and study in particular the convergence properties of augmented Lagrangian based approaches on general nonconvex structured NLPs.

3.3 Numerical Methods

3.3.1 Method of Multipliers

The method of multipliers (MM), commonly known as the AL method, seeks to solve the NLP (3.1) by finding a minimizer of the AL function:

$$\min_{x\in\mathcal{X},q,y} \mathcal{L}_\rho(x,q,y) = f(x) + y^T(Ax+Bq) + \frac{\rho}{2}\|Ax+Bq\|^2. \tag{3.3}$$

MM is based on the fundamental result that there exists a sufficiently large penalty parameter ρ such that a minimizer of the AL function is a minimizer of the NLP (3.2). The MM scheme performs a minimization of the AL function on the primal variables (x,q) and updates the Lagrange multipliers by using a steepest descent

step in the space of y (where the gradient is given by the primal residual $r(x,q) := Ax + Bq$ at the current iterate). The standard MM scheme is given in Algorithm 4.

Algorithm 4: Method of Multipliers

Given penalty parameter $\rho > 0$, tolerance $\epsilon > 0$, and estimates y^0

for $k = 0, 1, 2, \ldots$ **do**

  update primal variables:

    $(x^{k+1}, q^{k+1}) = \underset{x \in \mathcal{X}, q}{\arg\min}\, \mathcal{L}_\rho(x, q, y^k)$

  compute primal residual:

    $r^{k+1} = Ax^{k+1} + Bq^{k+1}$

  update dual variables:

    $y^{k+1} = y^k + \rho \cdot r^{k+1}$

  if $\|r^{k+1}\| \leq \epsilon$ **then**

    stop

The MM scheme of Algorithm 4 does not require an initial guess for x and q (because these are obtained from the solution of the subproblem in the first iteration $k = 0$). However, an initial guess for x and q can be used to accelerate the solution of the subproblems (which are solved using an NLP solver). The solution of the subproblems at iteration k can then be used to initialize (warm-start) the NLP solver at iteration $k + 1$. We denote a solution of the primal subproblem $\min_{x \in \mathcal{X}, q} \mathcal{L}_\rho(x, q, y^k)$ as $x(y^k, \rho), q(y^k, \rho)$ and we thus have that $(x^{k+1}, q^{k+1}) = (x(y^k, \rho), q(y^k, \rho))$.

Convergence theory of MM is well established for both convex and nonconvex problems. Here, we provide a summary of basic local convergence properties based on the work of Bertsekas [Bertsekas, 1976].

Assumption 1 *There exists a local minimizer (x^*, q^*) of problem* (3.2) *which satisfies the second-order sufficiency conditions for an isolated local minimum [Nocedal and Wright, 2006] (i.e., the Hessian of the Lagrangian function is positive definite in the null space of*

the constraint Jacobian). Moreover, the Hessian is Lipschitz continuous in a neighborhood of x^*.

Proposition 3.3.1 *Let Assumption 1 hold. For any given bounded set* $Y \subseteq \mathbb{R}^{\mathrm{m}}$ *there exists a scalar* $\rho \geq 0$ *such that for every* $\rho > \rho^*$ *and every point* $y \in Y$ *the function* $\mathcal{L}_\rho(x, q, y)$ *has a unique minimizer* $(x(y), q(y))$ *within an open ball centered at* x^*. *Furthermore, for some scalar* $M > 0$ *we have*

$$\|x(y,\rho) - x^*\| \leq \frac{M\|y - y^*\|}{\rho}, \forall \rho > \rho^*, y \in Y$$

$$\|q(y,\rho) - q^*\| \leq \frac{M\|y - y^*\|}{\rho}, \forall \rho > \rho^*, y \in Y$$

$$\|\widetilde{y}(y,\rho) - y^*\| \leq \frac{M\|y - y^*\|}{\rho}, \forall \rho > \rho^*, y \in Y$$

where the vector $\widetilde{y}(y, \rho)$ *has coordinates given by*

$$\widetilde{y}(y,\rho) = y + \rho\left(Ax(y,\rho) + Bq(y,\rho)\right)$$

The above result is one of the strongest convergence results available for nonconvex NLPs. It shows that if y^* is contained in Y, the generated sequence $y^{k+1} = y^k + \rho(Ax^k + Bq^k)$ remains in Y. It also states that, if the penalty parameter is sufficiently large and the minimization of $\mathcal{L}_\rho(x, q, y^k)$ yields the local minimum $(x(y^k, \rho), q(y^k, \rho))$ which is closest to y^*, $(x(y^k, \rho), q(y^k, \rho), y^*)$ converges to (x^*, q^*, y^*). The following result establishes convergence rates for the MM scheme.

Proposition 3.3.2 *Let Assumption 1 hold and let* Y *be an arbitrary sphere centered at* y^* *and the penalty parameter* $\rho \geq max\{M, \rho^*\}$, *then the sequence* $y^{k+1} = y^k + \rho(Ax^k + Bq^k)$ *converges to* (x^*, q^*, y^*). *Furthermore, if* $\rho < \infty$ *and* $y^k \neq y^*$ *for all* k *we have linear convergence*

$$\lim_{k\to\infty} \sup \frac{\|y^{k+1} - y^*\|}{\|y^k - y^*\|} \leq \frac{M}{\rho}.$$

Moreover, if $\rho \to \infty$, we have superlinear convergence

$$\lim_{k\to\infty} \frac{\|y^{k+1} - y^*\|}{\|y^k - y^*\|} = 0.$$

The results from Proposition 3.3.2 have been extended further to consider partial elimination of constraints [Bertsekas, 1977], and to establish quadratic convergence by using a second order update of y^k [Fletcher, 1975, Glad, 1979, Byrd, 1978, Betts, 1977, Rupp, 1975, Tapia, 1977]. In practice, however, MM uses a first-order multiplier update (because second order updates can only be computed in problems with no inequality constraints). Global convergence results for MM are based on the fundamental local convergence results discussed and seek to progressively drive the penalty parameter ρ to infinity [Conn et al., 1991]. We also note that the AL function serves a natural merit function in the MM method because x and q minimize it at every iteration and because the update in y achieves a contraction in the distance to the optimal dual variables.

3.3.2 Alternating Direction Method of Multipliers

ADMM is a variant of MM that enables decomposition of structured problems [Mishra, 2011]. This is based on the key observation that minimizing over the primal variables x and q separately (as opposed to jointly) enables the solution of subproblems over each partition $i \in \mathcal{P}$ independently [Boyd et al., 2011]. A standard ADMM scheme is summarized in Algorithm 5.

Algorithm 5: Alternating Direction Method of Multipliers

Given penalty parameter $\rho > 0$, tolerances $\epsilon_r > 0, \epsilon_s > 0$, and estimates y^0, q^0

for $k = 0, 1, 2, \ldots$ **do**

  update partition variables:

    $x^{k+1} = \arg\min_{x \in \mathcal{X}} \mathcal{L}_\rho(x, q^k, y^k)$

  update coupling variables:

    $q^{k+1} = \arg\min_{q} \mathcal{L}_\rho(x^{k+1}, q, y^k)$

  compute primal residual:

    $r^{k+1} = Ax^{k+1} + Bq^{k+1}$

  compute dual residual:

    $s^{k+1} = \rho A^T B \cdot (q^{k+1} - q^k)$

  update dual variables:

    $y^{k+1} = y^k + \rho \cdot r^{k+1}$

  if $\|r^{k+1}\| \leq \epsilon_r$ *and* $\|s^{k+1}\| \leq \epsilon_s$ **then**

    stop

To gain additional insights into the ADMM scheme, we know that the primal variables x are updated by solving the problem:

$$\min_x \quad f(x) + \left(Ax + Bq^k\right)^T y^k + \frac{\rho}{2}\|Ax + Bq^k\|^2 \tag{3.4.1}$$

$$\text{s.t.} \;\; c\,(x) \geq 0. \tag{3.4.2}$$

In the context of the structured NLP (3.1), one can solve for x by solving the partition subproblems:

$$\min_{x_i} \;\; f_i(x_i) + \left(A_i x_i + B_i q^k\right)^T y_i^k + \frac{\rho}{2}\|A_i x_i + B_i q^k\|^2 \tag{3.5.1}$$

$$\text{s.t.} \;\; c_i(x_i) \geq 0. \tag{3.5.2}$$

To update q one needs to solve the subproblem $\arg\min_q \mathcal{L}_\rho(x^{k+1}, q, y^k)$. This problem can be solved in closed-form from the first-order optimality conditions of the subproblem, to give:

$$q^{k+1} = (B^T B)^{-1} B^T A x^{k+1} \tag{3.6}$$

The sturcture of this update reveals that, in order for the update of the coupling variables q to be well-defined, we require matrix B to have full column rank. In the context of the structured NLP (3.1), one can show that (3.6) is an averaging (consensus) operator of the form:

$$q^{k+1}(j) = \frac{1}{|\mathcal{P}_j|} \sum_{i \in \mathcal{P}_j} x_i^{k+1}(j) \tag{3.7}$$

Here $\mathcal{P}_j \subseteq \mathcal{P}$ is the set of partitions connected to complicating variable $q(j)$.

In the context of the structured NLP (3.1), the dual variables are updated as:

$$y_i^{k+1} = y_i^k + \rho \cdot (A_i x_i^{k+1} + B_i q^{k+1}). \tag{3.8}$$

We note that, if the primal variables (x, q) are updated jointly, ADMM reduces to MM (for which convergence properties are well understood). While both methods have a common algorithmic structure, which is reflected in the updating formulas, ADMM blends ideas from MM and Gauss-Seidel coordination schemes to enable decomposition (3.1). For this reason, the convergence properties of ADMM are of great interest in the literature. For convex problems, ADMM has been proven to be globally convergent with a sublinear rate for general problems and with a linear rate in restricted problem classes [Goldfarb and Ma, 2012, He and Yuan, 2012] . Additional analysis is provided in [Boyd et al., 2011, Eckstein and Bertsekas, 1992a]. The convergence of ADMM on nonconvex problems remains largely open. Recent work by Hong et. al. [Hong et al., 2014] established convergence of ADMM for NLPs with nonconvex objectives but linear constraints. Specifically, they prove that the sequence generated by ADMM converges (glob-

ally) to the set of stationary solutions, provided that the penalty parameter in the AL function is chosen to be sufficiently large. Their analysis is based on using the augmented Lagrangian as a merit function to guide the iterate convergence.

A key observation is that the standard ADMM method uses a single coordinate minimization step to update x and q at every iteration. Recent work on parallelizable augmented Lagrangian approaches [Boland et al., 2018] proposes performing multiple coordinate minimization steps (which we refer to as coordination steps). This is based on the expectation that, by performing multiple steps, ADMM will approach the solution of the primal AL subproblem $\min_{x\in\mathcal{X},q} \mathcal{L}_\rho(x, q, y^k)$ (given by $x(y^k), q(y^k)$) and thus the ADMM scheme will achieve the same performance of the MM scheme. This observation also reveals that ADMM can be interpreted as an inexact version of MM (in which the subproblems are minimized inexactly). Consequently, MM provides the limiting (ideal) performance of ADMM. We also observe that performing multiple coordinating steps is equivalent to applying a block coordinate minimization scheme to the function $\mathcal{L}_\rho(x, q, y^k)$. Consequently, it is possible to borrow convergence theory results for block coordinate descent methods to identify conditions under which the coordination steps will converge to $x(y^k), q(y^k)$ and thus ADMM behaves like MM [Tseng, 2001].

We summarize an ADMM scheme with multiple coordination steps in Algorithm 6. Here, the standard ADMM scheme is obtained when $n_{cs} = 1$ and we expect that the performance of MM is obtained when $n_{cs} = \infty$ (or a large number) is used, provided that the block coordinate descent scheme converges. We also consider a variant of the ADMM method (that we call iADMM) that is initialized using the values of the primal and dual variables obtained in the first iteration of MM. This allows us to evaluate the impact of initialization on ADMM.

Algorithm 6: Alternating Direction Method of Multipliers for Structured NLP

Given penalty parameter $\rho > 0$, tolerances $\epsilon_r > 0, \epsilon_s > 0$, the coupling partition sets $\mathcal{P}_j$, the number of coordination steps n_{cs} and the starting estimates y^0, q^0

for $k = 0, 1, 2, \ldots$ **do**

  Perform coordination steps:

  for $s \in \{0, 1, ..., n_{cs} - 1\}, \ldots$ **do**

    update partition variables :

    foreach $i \in \mathcal{P}$ **do**

      $$x_i^{k+1,s+1} = \underset{x_i \in \mathcal{X}_i}{\arg\min}\, f_i(x_i) + \left(A_i x_i + B_i q^{k,s}\right)^T y_i^k + \frac{\rho}{2}\|A_i x_i + B_i q^{k,s}\|^2$$

    update coupling variables:

    for $j = 0, 1, 2, \ldots, n_q$ **do**

      $$q^{k+1,s+1}(j) = \frac{1}{|\mathcal{P}_j|} \sum_{i \in \mathcal{P}_j} x_i^{k+1,s+1}(j)$$

  compute primal residual:

  foreach $i \in \mathcal{P}$ **do**

    $$r_i^{k+1} = A_i x_i^{k+1,n_{cs}} + B_i q^{k+1,n_{cs}}$$

  compute dual residual:

  for $j = 0, 1, 2, \ldots, n_q$ **do**

    $$s^{k+1}(j) = \rho\left(q^{k+1,n_{cs}}(j) - q^{k,n_{cs}}(j)\right)$$

  update dual variables:

  foreach $i \in \mathcal{P}$ **do**

    $$y_i^{k+1} = y_i^k + \rho \cdot r_i^{k+1}$$

  if $\|r^{k+1}\| \leq \epsilon_r$ *and* $\|s^{k+1}\| \leq \epsilon_s$ **then**

    stop

3.3.3 Progressive Hedging

The PH method is commonly used as a decomposition-based solver for stochastic programming problems [Rockafellar and Wets, 1991]. Here, we note that PH is

equivalent to applying ADMM (Algorithm 6) to two-stage stochastic programs [Boyd et al., 2011]. Correspondingly, the multi-stage stochastic PH is equivalent to a multi-block ADMM scheme. In paticular, two-stage stochastic programs have the structure (3.1) where the first-stage variables correspond to the coupling variables q in NLP (3.1) while the second-stage variables correspond to the partition variable vectors $x_i, i \in \mathcal{P}$. Provided that the first-stage variables couple all scenarios (block partitions), the partition matrices are given by $B_i = -\mathbb{I}, i \in \mathcal{P}$ where $\mathbb{I}$ is the identity matrix.

The PH method follows the same primal and dual variable updates of the standard ADMM scheme. Of particular relevance is the observation that the solution of the subproblem $\arg\min_q \mathcal{L}_\rho(x^{k+1}, q, y^k)$ defines an *averaging* operator over all scenarios (partitions $\mathcal{P}$). For problems with scenarios with equal probabilities, the averaging operator is:

$$q^{k+1}(j) = \frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} x_i^{k+1}(j) \tag{3.9}$$

This averaging update of q is equivalent to the one obtained in Equation 3.7. For details on the derivation of this update from ADMM see Appendix C. Given that PH is a special case of ADMM, we observe that ADMM is in general a more flexible approach to decompose problems. Moreover, one can directly borrow convergence results from ADMM to determine the conditions under which PH is expected to converge. For instance, convergence guarantees for PH in convex stochastic programs with risk constraints has not been established. This is mostly due to the fact that the structure of these problems deviates from the standard two-stage stochastic structure assumed in existing convergence analysis. ADMM convergence results for convex problems indicate that convergence of PH is expected risk constraints as well.

3.4 Benchmarking Metrics

In a convex optimization setting, it has been shown that the Lyapunov function

$$V_\rho(y,q) = \frac{1}{\rho}\|y-y^*\|^2 + \rho\|q-q^*\|^2 \tag{3.10}$$

is monotonically decreased by the ADMM scheme [Boyd et al., 2011]. Since the Lyapunov function depends on the actual primal-dual solution q^*, y^*, which is unknown in general, this metric is not a practical merit function to monitor progress of the algorithm. However, we use this metric to gain insights on the effect of nonconvexity on the performance of the ADMM algorithm. In particular, lack of a monotonic decrease of the Lyapunov function provides an indication that nonconvexity is hindering convergence of ADMM.

The AL function provides a natural merit function to monitor progress of the ADMM in nonconvex problems:

$$\mathcal{L}_\rho(x^k,q^k,y^k) = \sum_{i\in\mathcal{P}} f_i\left(x_i^k\right) + \left(A_i x_i^k + B_i q^k\right)^T y^k + \frac{\rho}{2}\|A_i x_i k + B_i q^k\|^2. \tag{3.11}$$

Specifically, compared to the Lyapunov function, the AL function does not require prior knowledge of the solution. Moreover, the MM method decreases the AL function at every iteration. Consequently, if the performance of ADMM approaches that of MM (e.g., by performing multiple coordination steps), we can expect that the AL function will decrease as well. In other words, reduction of the AL function provides an indication if enough coordination steps have been performed.

Other important metrics include the primal and dual errors (derived from the optimality conditions of (3.1)). These are given by:

$$\nabla_x \mathcal{L}_\rho(x^k,q^k,y^k) = s^k \tag{3.12.1}$$

$$\nabla_q \mathcal{L}_\rho(x^k,q^k,y^k) = 0 \tag{3.12.2}$$

$$Ax^k + Bq^k = r^k \tag{3.12.3}$$

Here, one can show that $s^{k+1} = \rho A^T B(q^{k+1} - q^k)$. The stationarity condition with respect to q is satisfied at each iteration k because the consensus step enforces it explicitly. A stationary point for the NLP is thus attained when $\|r^k\| \to 0$ and $\|s^k\| \to 0$. We also note that the AL function converges $f(x^k)$ when $\|r^k\| \to 0$.

3.5 Numerical Experiments

We study the performance of MM and ADMM on nonconvex optimal control problems of the form:

$$\min \quad \int_{\mathcal{T}} J(w(t), u(t), v(t))dt \tag{3.13.1}$$

$$\text{s.t.} \quad F(\dot{w}(t), w(t), u(t), v(t)) = 0,\ t \in \mathcal{T} \tag{3.13.2}$$

$$w(t_0) = w_0 \tag{3.13.3}$$

$$\underline{w} \leq w(t) \leq \bar{w},\ t \in \mathcal{T} \tag{3.13.4}$$

$$\underline{u} \leq u(t) \leq \bar{u},\ t \in \mathcal{T} \tag{3.13.5}$$

$$\underline{v} \leq v(t) \leq \bar{v},\ t \in \mathcal{T}. \tag{3.13.6}$$

where $\mathcal{T} = [t_0, t_f]$ is the time domain, $w \in \mathbb{R}^{n_w}$ are state variables, $u \in \mathbb{R}^{n_u}$ are control variables, and $v \in \mathbb{R}^{n_v}$ are algebraic states.

The differential algebraic system is discretized by applying a collocation on finite elements scheme as described by Biegler [2010]. In the scheme considered, the time dependent variables are approximated using Lagrange polynomials defined at a set of collocation points $t_j, j \in \mathcal{N}_c := \{0, ..., n_c - 1\}$ on the domain $\tau = (0, 1)$. Applying an affine transformation, the time domain is expressed as $t = h_i\tau$ with h_i defined as the length of finite element $i \in \mathcal{N}_e := \{0, ..., n_e - 1\}$. Then, in each finite element the state variables are approximated by

$$w(\tau) = \sum_{j \in \mathcal{N}_c} w_{i,j} \ell_j(\tau),\ i \in \mathcal{N}_e \tag{3.14}$$

where $\ell(\tau)$ are the Lagrange polynomials. A similar approximation is applied to the algebraic and control variables. The expressions for the derivative variables at the ith finite element and jth collocation point $\dot{w}_{i,j}$ are given by

$$\dot{w}_{i,j} = \frac{1}{h_i} \sum_{j \in \mathcal{N}_c} w_{i,j} \dot{\ell}_j(\tau), i \in \mathcal{N}_e, j \in \mathcal{N}_c \tag{3.15}$$

With this the discretized DAE system can be expressed over the entire domain as

$$F(\dot{w}_{i,j}, w_{i,j}, u_{i,j}, v_{i,j}) = 0, \ i \in \mathcal{N}_e, j \in \mathcal{N}_c \tag{3.16}$$

To enforce continuity of the state variables at the finite element boundaries we impose the constraints:

$$w_{i+1,0} = w_{i,n_c}, \ i = 0, ..., n_e - 1 \tag{3.17}$$

The objective function is discretized using a quadrature formula:

$$\sum_{i=0}^{n_e-1} \sum_{j=0}^{n_c-1} p_j J(w_{i,j}, u_{i,j}, v_{i,j}), \tag{3.18}$$

where p_j are Radau quadrature weights. The discretized problem is an NLP of the form:

$$\min \quad \sum_{i \in \mathcal{N}_e} \sum_{j \in \mathcal{N}_c} p_j J(w_{i,j}, u_{i,j}, v_{i,j}) \tag{3.19.1}$$

$$\text{s.t.} \ \ F(\dot{w}_{i,j}, w_{i,j}, u_{i,j}, v_{i,j}) = 0 \tag{3.19.2}$$

$$w_{0,0} = w_0 \tag{3.19.3}$$

$$w_{i+1,0} = w_{i,n_c}, \ i \in \mathcal{N}_e \tag{3.19.4}$$

$$\underline{w}_{i,j} \leq w_{i,j} \leq \bar{w}_{i,j}, \ i \in \mathcal{N}_e, j \in \mathcal{N}_c \tag{3.19.5}$$

$$\underline{u}_{i,j} \leq u_{i,j} \leq \bar{u}_{i,j}, \ i \in \mathcal{N}_e, j \in \mathcal{N}_c \tag{3.19.6}$$

$$\underline{v}_{i,j} \leq v_{i,j} \leq \bar{v}_{i,j}, \ i \in \mathcal{N}_e, j \in \mathcal{N}_c. \tag{3.19.7}$$

The NLP can be written as the structured problem (3.1) by defining every element as a partition (i.e., $\mathcal{P} = \mathcal{N}_e$) $x_i = \{\dot{w}_{i,j}, w_{i,j}, u_{i,j}, v_{i,j}\}$, $i \in \mathcal{P}$. The coupling variable vector contains the state variables at the boundary of the finite elements

(with the exception of the first and last partition that are only coupled on one end) $q = \{w_{0,n_d}, w_{i,0}, w_{i,n_d}, w_{\mathcal{P},0}\}$. Here, we describe the coupling variable vector q as a composition of the state variables between each finite element. This gives a system where each finite element represents one partition. However, partitions may include multiple finite elements. For example, a problem with 64 finite elements can be separated into two partitions of 32 finite elements each, 4 partitions of 16 finite elements each, and so forth. We highlight that ADMM decomposition can be applied to more general dynamic optimization problems that include path constraints and end-point constraints (because such features do not alter the block structure of the problem).

The optimization problems `Pyomo` Hart et al. [2017, 2011a]. `Pyomo` facilitated the automatic discretization of the differential equations using the `Pyomo.DAE` package [Nicholson et al., 2017], and provided flexibility for the decomposition of the structured NLPs that resulted from discretization. For validation and comparison purposes, the solution of the nonconvex NLPs was obtained by means of three different approaches. In the first approach the original full problem (3.1) was solved with the off-the-shelf interior point solver `Ipopt` Wächter and Biegler [2006]. The results from this approach provided validation for the other approaches. The second approach is the method of multipliers described in Algorithm 4. In the third approach the problem was decomposed and solved using the ADMM scheme described in Algorithm 6. In all three approaches `Ipopt` was used for solving the corresponding subproblems. We thus highlight that all solutions were local minimizers.

We use the following tolerances for the residuals computed in ADMM Boyd et al. [2011]:

$$\epsilon_r = \sqrt{n_x}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\max\{\|Ax^k\|, \|Bq^k\|\}$$
$$\epsilon_s = \sqrt{n_y}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\|A^T y^k\|_2$$

where $\epsilon^{\text{abs}} = \epsilon^{\text{rel}} = 10^{-4}$. For MM we use a tolerance $\epsilon = 10^{-8}$.

The state variables were scaled so that their values lie between zero and one. Besides improving the conditioning of the optimization problems, the scaling facilitated the selection of the AL penalty parameter ρ. This parameter needs to be large enough for the subproblems to converge. However, excessively large values of ρ can result in numerical instabilities. Hence, by scaling the state variables the penalization term $\|Ax - Bq\|$ is expected to be close to one. The penalty parameter can then be selected to balance the objective of the overall problem and the penalization term.

3.5.1 Separation System

We consider an optimal control problem of a distillation column with 30 trays Benallou et al. [1986]. The complete model contains 32 differential equations, one control, and 35 algebraic states. The reflux ratio is the control variable while the purity of the distillate (denoted as $\bar{y}$) provides a measure of product quality. The relative volatility $\alpha_{A,B}$ is assumed constant given a value of 1.6. The mixture is fed to the column in tray 17 and has a composition of x_A= 0.5.

$$\min \quad \int_{t_0}^{t_f} (y_{A,1} - \bar{y})^2 + (u - \bar{u})^2 dt$$

$$\text{s.t.} \quad \frac{dx_{A,0}}{dt} = \frac{1}{A_c} V(y_{A,1} - x_{A,0})$$

$$\frac{dx_{A,i}}{dt} = \frac{1}{A_t} \left(L_1(y_{A,i-1} - x_{A,i}) - V(y_{A,i} - y_{A,i+1})\right) \qquad i \in \{1, ..., FT-1\}$$

$$\frac{dx_{A,FT}}{dt} = \frac{1}{A_t} \left(F x_{A,f} + L_1 x_{A,FT-1} - L_2 x_{A,FT} - V(y_{A,FT} - y_{A,FT+1})\right)$$

$$\frac{dx_{A,i}}{dt} = \frac{1}{A_t} \left(L_2(y_{A,i-1} - x_{A,i}) - V(y_{A,i} - y_{A,i+1})\right) \qquad i \in \{FT+1, ..., NT\}$$

$$\frac{dx_{A,NT+1}}{dt} = \frac{1}{A_r} \left(L_2 x_{A,NT} - (F-D) x_{A,NT+1} - V y_{A,NT+1}\right)$$

$$x_{A,i} = x_{0_{A,i}} \qquad i \in \{0, ..., NT+1\}$$

$$V = L_1 + D, \quad L_2 = L_1 + F, \quad u = \frac{L_1}{D}, \quad \alpha_{A,B} = \frac{y_A(1-x_A)}{x_A(1-y_A)}$$

where $x_{A,i}$ and $y_{A,i}$ are the liquid and vapor mole fractions of component A at the i-th tray. F is the feed flowrate, D is the distillate flowrate, L_1 is the flow rate of the liquid in the rectification section, L_2 is flowrate of the liquid in the stripping section, NT is the number of trays in the column, and FT is the feed tray. The parameter values used in the model are given in Table 3.1. For more details on the model see [Benallou et al., 1986, Kang et al., 2014, Hahn and Edgar, 2002].

Table 3.1.: Parameter values for distillation column example system

Parameter	Value
A_c	0.5 Kmol
A_t	0.25 Kmol
A_r	1.0 kmol
D	0.2 kmol/min
F	0.4 kmol/min
$\bar{y}$	0.8958
$\bar{u}$	2.0

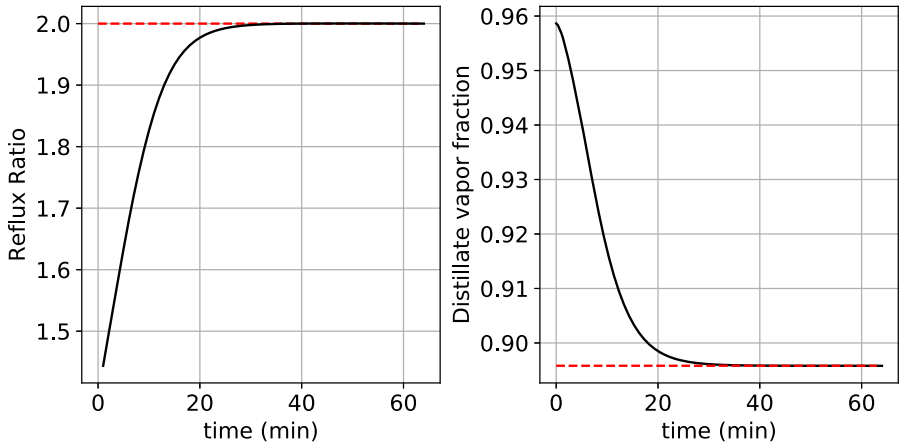


Figure 3.1.: Trajectories of distillate molar fraction (right) and reflux ratio (left) with setpoint (- -)

The differential equations in the dynamic optimization problem (3.21.1) were discretized via an orthogonal collocation method on 64 finite elements with 2 Radau collocation points per finite element. The discretization resulted in a structured

NLP with 12,997 variables and 12,868 constraints. The optimal vapor concentration of the distillate trajectory is presented in Figure 3.1.

The dynamic optimization problem problem was also solved with MM, ADMM, and iADMM schemes described in the previous section. The problem was decomposed into four partitions and solved following Algorithms 4 and 6. In both cases the results were validated against the full problem solutions obtained with `Ipopt`. Figure 3.2 summarizes the performance of the algorithms based on the metrics described in Section 3.4. MM (green) shows the results for the method of multipliers with initial estimates for the dual variables y^0 equal to zero. ADMM (grey) presents the results of Algorithm 6 with dual estimates y^0 initialized at zero and coupling variable estimates q^0 initialized with the initial condition x_{0_A} of the corresponding state variable. For this first set of results, we use a single coordination step to update the primal variables in ADMM ($n_{cs} = 1$). iADMM (red) shows the results of Algorithm 6 initialized using the primal and dual values from the first iteration of MM. From Figure 3.2 it is clear that, despite the strong nonconvexity of the problem, both MM and ADMM converge to the solution. MM requires significantly fewer iterations to converge than ADMM but both methods exhibit linear convergence. The superior performance of MM is due to the simultaneous update of the primal variables (x, q). In contrast, the standard ADMM algorithm uses a sequential update by using a single coordination step. With the ADMM method we observe oscillations in the dual infeasibility metric and in the Lyapunov function. The oscillations indicate that the Lyapunov function is not monotonically decreasing highlighting the effect of the nonconvexity. We also see that initializing ADMM (iADMM) significantly improves performance and achieves a smoother decay in the Lyapunov function. However, we see that the oscillations persist. Figure 3.3 shows the effect of using multiple coordination steps at each iteration of ADMM. We can see that this mimics the joint minimization of x and q used in MM. It is remarkable that just an additional coordination step achieves significant improvements and ameliorates the oscillations of the standard ADMM scheme. We

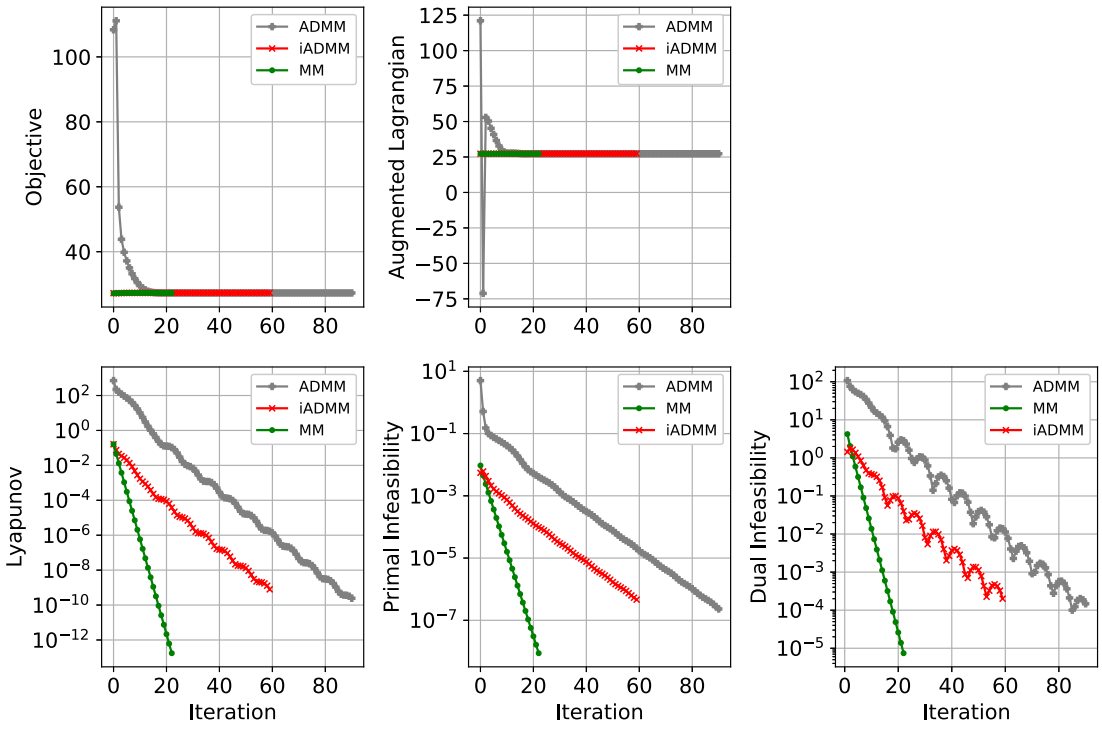


Figure 3.2.: Convergence of MM, iADMM, and ADMM in separation system.

also see that, for this problem, approximately 4 coordination steps are sufficient to approach the performance of MM. This behavior is also observed in the timing results. The solution time for MM was 8.5 seconds while ADMM found solutions in 15.15, 10.93, 13.52, 21.032 y 40.86 seconds with 1, 2, 4, 8, and 16 coordination steps, respectively. Interestingly, the computational times of ADMM reveal that an optimal number of coordination steps exist and that this tends to be small (in this case two). We can see, however, that MM is indeed faster than ADMM because of two reasons. This is because the problem targeted is not large enough to see the benefit of decomposition. Moreover, the results presented do not perform parallelization of the subproblem solutions. If we assume perfect parallelization of the

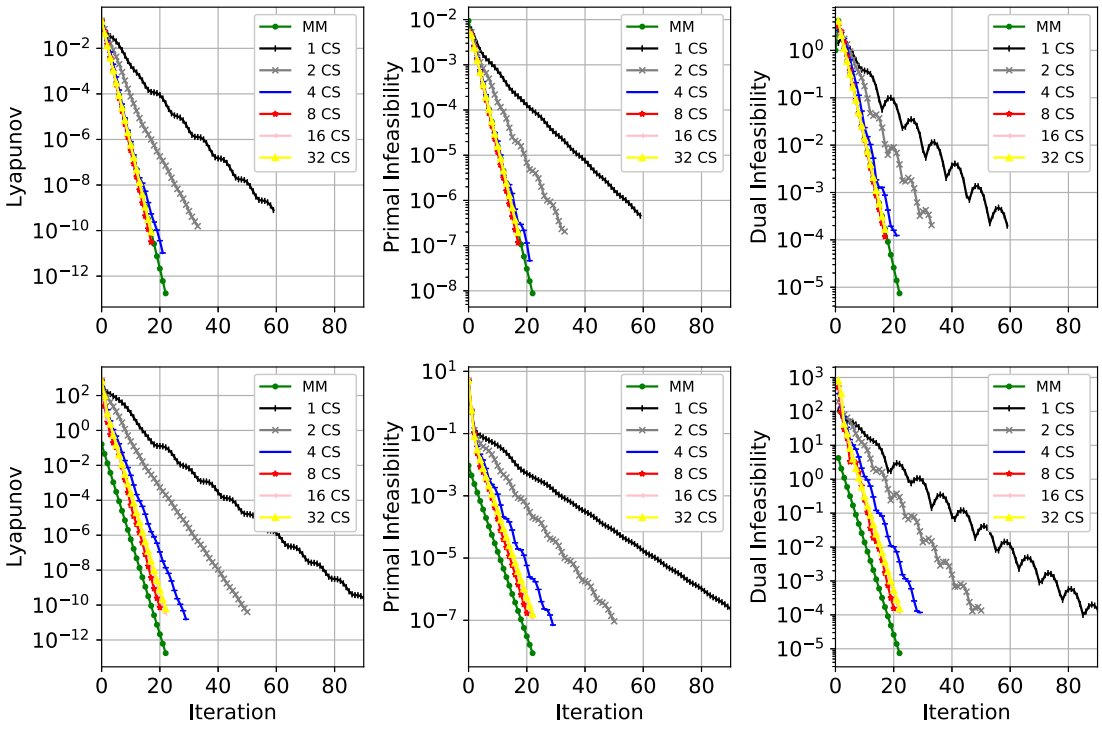


Figure 3.3.: Convergence of iADMM (top row) and ADMM (bottom row) in separation system with multiple coordinate minimization steps.

subproblem solutions, the timing results of ADMM can be reduced by a factor of four (because the problem was decomposed in four partitions).

We highlight that the ADMM scheme provides a flexible approach to decompose dynamic optimization problems in time. This approach share some similarities with other decomposition approaches such as multiple shooting (MS). In MS, the horizon is decomposed into stages and state continuity is enforced progressively by an optimization solver. A key difference between ADMM and MS is that the state continuity constraints in ADMM are enforced implicitly by minimizing the augmented Lagrangian (while in MS these are enforced explicitly by the optimization solver). In principle, however, one can envision implementing using MS by enforcing continuity using an ADMM scheme. Here, the ADMM subprob-

lems would be dynamic optimization problems over a short time horizon and state continuity is enforced by updating the Lagrange multipliers.

3.5.2 Semibatch Reactor

The second study involves the optimal control of a semibatch reactor Abel and Marquardt [2000]. The model considers consecutive highly exothermic reactions $A \longrightarrow B \longrightarrow C$. This model considers a reactor vessel equipped with two heat exchanger systems, a reactor jacket and an internal coil. The control inputs for this model are the cooling medium temperatures in the jacket and in the coil, $T_{w,j}$ and $T_{w,c}$ respectively. The optimization problem seeks to maximize the production of the desired component B and has the form:

$$\text{maximize} \quad \int_{t_0}^{t_f} C_b \, dt \tag{3.21.1}$$

$$\text{subject to} \quad \dot{C}_a = \frac{F_a}{V_r} - k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a \tag{3.21.2}$$

$$\dot{C}_b = k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a - k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \tag{3.21.3}$$

$$\dot{C}_c = k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \tag{3.21.4}$$

$$\dot{V}_r = \frac{F_a M_{W_a}}{\rho_r} \tag{3.21.5}$$

$$(\rho_r c_{p_r})\dot{T}_r = \frac{F_a M_{W_a} c_{p_r}}{V_r}(T_f - T_r) - k_1 \exp\left(-\frac{E_1}{RT_r}\right) C_a \Delta H_1$$

$$- k_2 \exp\left(-\frac{E_2}{RT_r}\right) C_b \Delta H_2 + \alpha_{w,j} \frac{A_j}{V_{r,0}}(T_{w,j} - T_r) \tag{3.21.6}$$

$$+ \alpha_{w,c} \frac{A_c}{V_{r,0}}(T_{w,c} - T_r) \tag{3.21.7}$$

$$V_r(t_0) = V_{r_0}, \quad T_r(t_0) = T_{r_0}, \quad C_i(t_0) = C_{i_0} \quad i \in \mathcal{S} \tag{3.21.8}$$

Here the molar composition of the i-th component C_i , the temperature of the mixture in the reactor T_r, and the volume V_r are the state variables. Values for the model parameters and initial conditions for the differential equations are given in Table 3.2.

Table 3.2.: Parameter values optimal control semibatch reactor

Parameter	Value	Parameter	Value
k_1	15.01 1/s	A_j	5.0 m^2
k_2	85.01 1/s	A_c	3.0 m^2
E_1	30,000.0 kJ/kmol	V_j	0.9 m^3
E_2	40,000.0 kJ/kmol	V_c	0.07 m^3
R	8.314 kJ/kmol/K	ρ_w	700.0 kg/m^3
M_{W_a}	50.0 kg/kmol	c_{p_w}	3.1 kJ/kg/K
ρ_r	1,000.0 kg/m^3	$C_a(t_0)$	0.0 kmol/m^3
c_{p_r}	3.9 kJ/kg/K	$C_b(t_0)$	0.0 kmol/m^3
T_f	300.0 K	$C_c(t_0)$	0.0 kmol/m^3
ΔH_1	-40,000.0 kJ/kmol	$T_r(t_0)$	300.0 K
ΔH_2	-50,000.0 kJ/kmol	$V_r(t_0) = V_{r,0}$	1.0 m^3
$\alpha_{w,j}$	0.8 kJ/s/m^2/K	$\alpha_{w,c}$	0.7 kJ/s/m^2/K

The differential equations in (3.21.6) were discretized with 256 finite elements and 2 Radau collocation points per finite element. The discretization resulted in a structured NLP with 5,637 variables and 5,381 constraints. We use 4 partitions in the problem. The optimal trajectories of the state variables are presented in Figure 3.4

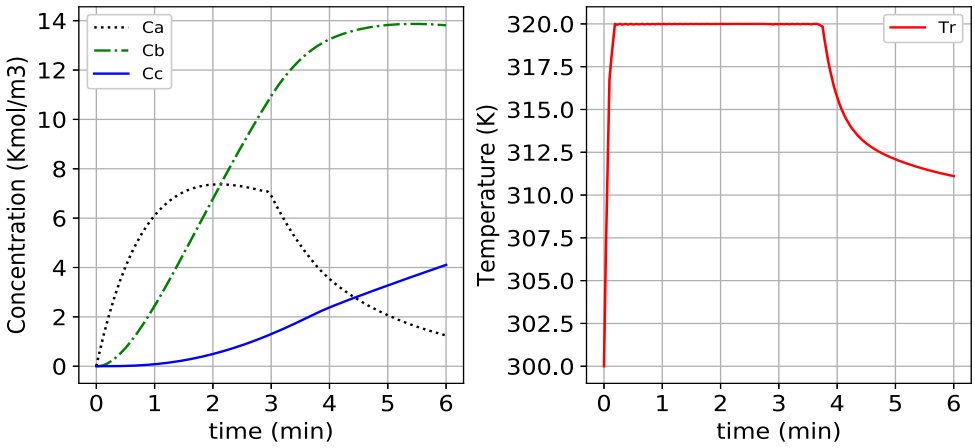


Figure 3.4.: Optimal concentration profiles (left) and optimal temperature profile (right)

The performance of the algorithms in presented in Figure 3.5. We observe similar trends to those seen in the previous case study. In particular, the Lyapunov and the AL functions do not decrease monotonically for the ADMM scheme, highlighting the strong effect of nonconvexity. Notably, MM converges in less than 10 iterations while ADMM and iADMM require over 200 iterations. In this problem we observe stronger oscillations in the dual infeasibility.

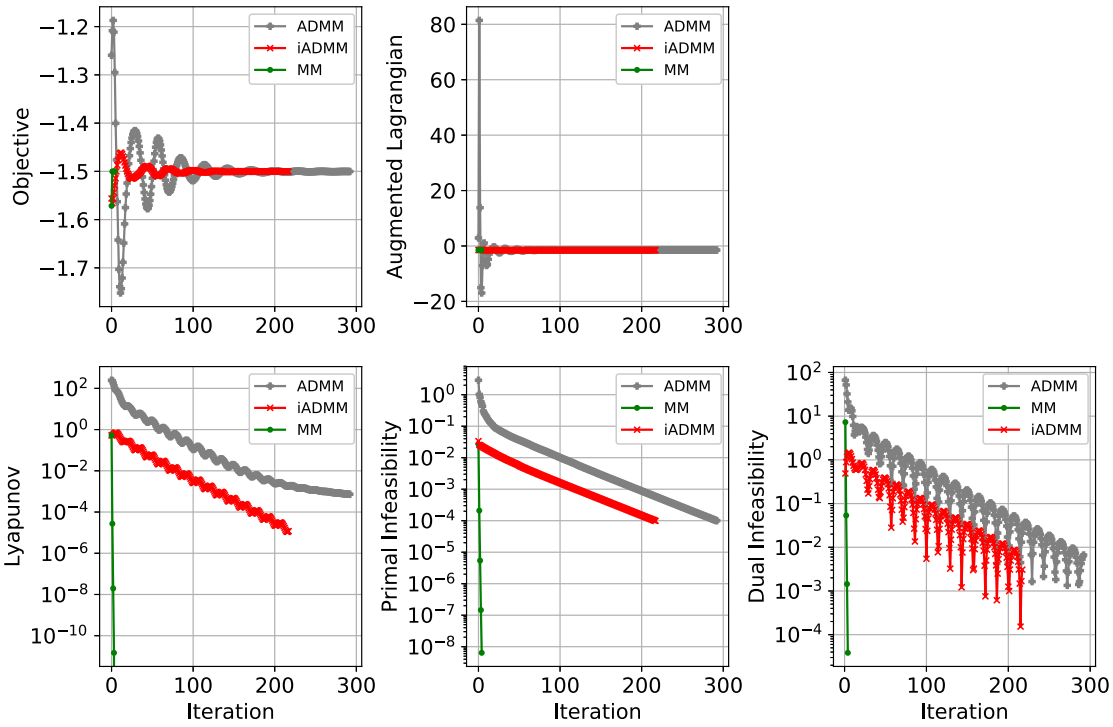


Figure 3.5.: Convergence metrics semibatch reactor problem. From top to bottom and left to right: objective value, augmented Lagrangian value, Lyapunov value, primal infeasibility, dual infeasibility

Motivated by the results obtained in the separation system, we also experimented with the number of coordination steps used in ADMM. As seen in Figure 3.6, performing just a handful of coordination steps is sufficient to eliminate the oscillations in the dual infeasibility. Moreover, we observe that the coordination

steps converge to the performance of MM. We can thus see that, in general, performing multiple coordination schemes drastically improves performance. This observation is relevant because it indicates that performance similar to that of MM can be achieved by solving smaller subproblems in parallel.

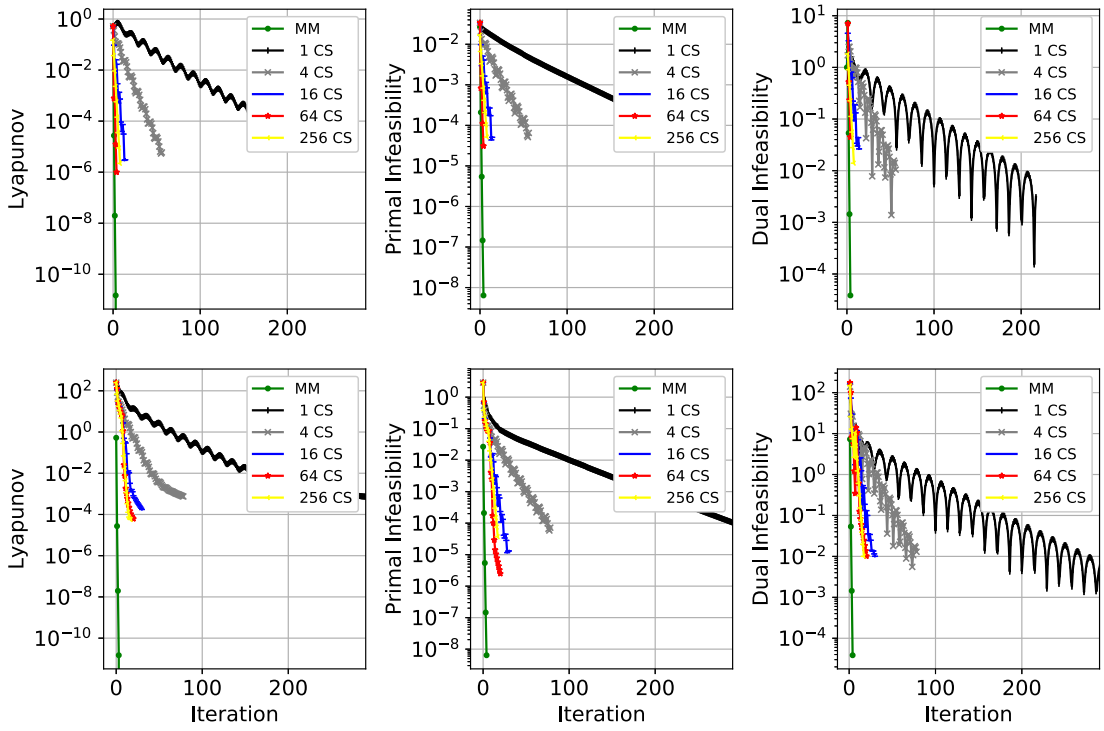


Figure 3.6.: Convergence analysis semibatch reactor problem. From top to bottom: iADMM ADMM. From left to right: Lyapunov value, primal infeasibility, dual infeasibility

From Figure 3.6 we see that MM achieves convergence in a couple of iterations. The timing results for this problem show that MM converges to an optimal solution in 0.36 seconds. ADMM, on the other hand, requires 9.21, 5.61, 5.81, 7.97, 19.52 seconds with 1, 4, 16, 64, and 256 coordination steps, respectively. These results again highlight that the problem is not large enough to see a clear benefit of decomposition. Compared to the first case, parallelization will not help ADMM

overcome the fast convergence of MM. As in the first study, we again see that a small number of coordination steps achieves acceleration of ADMM (in this case four). These results reveal that there is a need to develop schemes that identify a suitable number of coordination steps to be performed at each iterate. For instance, performing more coordination steps will tend to be more helpful in early (rather than in later) iterations. This behavior is typical of inexact optimization schemes. We highlight that the problems studied in this work are used to monitor convergence properties (and not time). Timing analysis would require of more efficient implementations and larger benchmarking instances. We also highlight that ADMM decomposition offers benefits that go beyond computing time. In particular, they enable distribution of memory and they do not require centralization of information (as in MM). This is beneficial in applications that involve low-cost computing devices with limited processing speeds and memory.

3.6 Summary

Application of the simultaneous discretization approach to difficult dynamic optimization problems leads to large-scale nonlinear programming problems that can be challenging for off the shelf solvers. However, these problems can be solved in the context of an ADMM or PH approach that supports structural decomposition of the problem along selected finite element boundaries. ADMM (and PH as a specialization) provides a powerful decomposition mechanism that has been shown to be effective on convex problems in dynamic optimization, machine learning, and stochastic programming. However, while the convergence properties of these methods is well established for convex problems, little is known about convergence on nonconvex problems. In contrast, AL methods have well-established convergence theory for both convex and nonconvex problems, but can become computationally prohibitive on large problems.

In this paper, we show the convergence properties of the AL method, the general ADMM method, and a modified ADMM method on two nonconvex dynamic

optimization problems. As metrics for benchmarking performance, we proposed the use of the AL function, along with the Lyapunov function. While it has been shown that ADMM methods provide monotonic decrease in the Lyapunov function on convex problems, for these nonconvex problems, we observe oscillation in both the dual feasibility and the Lyapunov function. Furthermore, the performance of the general ADMM approach is significantly worse than the AL approach.

To overcome poor convergence performance, we proposed a modified ADMM approach with multiple coordination minimization steps at each ADMM iteration. With this modification, we observe that the oscillations in the dual infeasibility and the Lyapunov function are significantly reduced. Furthermore, the results on both case studies showed that even with only a few additional coordination steps, convergence rates significantly improved, and this modified algorithm approached the performance of the AL method.

Given these observations, we provide connections between the AL and ADMM methods, and show that ADMM can be viewed as an inexact AL method. This link provides an approach to monitor and ensure progress of the ADMM scheme on nonconvex problems. Furthermore, this provides a future context for convergence analysis on nonconvex problems.

4. ADMM PRECONDITIONING FOR BLOCK-STRUCTURED KKT SYSTEMS

In this Chapter we study the solution of block-structured linear algebra systems arising in optimization using iterative solution techniques. These systems are the core computational bottleneck of many problems of interest such as parameter estimation, optimal control, networks, and stochastic programming. Our approach uses a Krylov solver (GMRES) that is preconditioned with an alternating method of multipliers. We show that this ADMM-GMRES approach overcomes well-known scalability issues of Schur complement decomposition in problems that exhibit high degree of coupling. The effectiveness of the approach is demonstrated using linear systems that arise in stochastic optimal power flow problems and that contain up to 2 million variables and 4,000 coupling variables. We find that ADMM-GMRES is nearly an order of magnitude faster than Schur decomposition in problems with high degree of coupling. Moreover, we demonstrate that the approach is robust to the selection of the ADMM penalty parameter.

4.1 Preliminaries

The scalability of optimization solvers relies quite heavily on the solution of the underlying linear algebra systems. Advances in direct sparse linear algebra solvers have been instrumental in the widespread use of quadratic programming and nonlinear programming solvers such as `Ipopt`, `OOQP`, and `Knitro` [HSL, Amestoy et al., 2001, 2006]. Specialized direct solution techniques have also been developed to tackle large-scale and *block-structured* systems (using variants of Schur complement decomposition techniques) [Zavala et al., 2008, Word, 2014, Chiang et al., 2014, Gondzio and Grothey, 2009, Gondzio and Sarkissian, 2003]. Block structures appear in many important applications such as parameter estimation, stochastic

programming, network optimization, and optimal control. Schur decomposition techniques can also leverage *parallel computing architectures* and have enabled the solution of problems with millions to billions of variables and constraints. Unfortunately, many applications of interest still remain inaccessible due to fundamental scalability limitations of Schur complement techniques. Specifically, Schur decomposition does not scale well in problems that exhibit high degrees of *block coupling*. Specifically, assembling and factorizing the Schur complement matrix (which is often highly dense) is a key computational bottleneck in such problems.

Iterative solution techniques [Quarteroni, 2007, Ma and Zheng, 2015, Forsgren et al., 2007, Elman and Golub, 1994, Benzi and Simoncini, 2006, Benzi et al., 2005] and associated preconditioning strategies [Cao et al., 2016, Morales and Nocedal, 2000, Golub and Greif, 2003, Rusten and Winther, 1992, Gill et al., 1992, Zulehner, 2002] have been proposed to address fundamental scalability issues of direct linear algebra strategies. In the context of block-structured problems, attempts have been made to solve the Schur complement system using iterative solution techniques (to avoid assembling and factorizing the Schur complement). Preconditioners for Schur complements arising in special problem classes such as networks and stochastic programs have been developed [Cao et al., 2016]. Unfortunately, preconditioning strategies for general problem classes are still lacking. Another important issue that arises in this context is that the implementation of advanced linear algebra strategies is non-trivial (e.g., it requires intrusive modifications of optimization solvers).

Along a separate line of research, significant advances have been made in the development of problem-level decomposition techniques such as the alternating direction method of multipliers (ADMM) and Lagrangian dual decomposition [Han and Yuan, 2013, Guo et al., 2017, Hong and Luo, 2017, Han and Yuan, 2013, Goldfarb and Ma, 2012, He and Yuan, 2012, Rodriguez et al., 2018]. Such approaches are flexible and rather easy to implement but suffer of slow convergence. Recently, it has been proposed to use ADMM as a preconditioner of Krylov-based iterative

solvers such as GMRES [Zhang and White, 2016, 2018]. In this work, we provide a detailed derivation and test the performance of a ADMM-GMRES method in the context of block-structured linear algebra systems. We demonstrate that this approach overcomes scalability issues of Schur complement decomposition (for problems with high degrees of coupling). We also demonstrate that this approach is significantly more effective than ADMM. Our numerical tests are facilitated by the use of `PyNumero`, a recently-developed Python framework that enables the implementation and benchmarking of optimization algorithms. We use the proposed framework to tackle problems with hundreds of thousands to millions of variables and that arise from standard benchmark sets and power grid applications.

This chapter is structured as follows. In Section 4.2 we define the problem of interest and provide preliminary information on the use of Schur complement decomposition and ADMM approaches. In Section 4.2.3 we provide a detailed derivation of the ADMM-GMRES approach and in Section 4.3.1 we provide benchmark results.

4.2 Problem Definition

We study the solution of block-structured quadratic programs (QP) of the form:

$$\min_{x_i,z} \quad \sum_{i\in\mathcal{P}} \frac{1}{2} x_i^T D_i x_i + c_i^T x_i \tag{4.1.1}$$

$$\text{s.t.} \quad J_i x_i = b_i, \qquad (\lambda_i) \quad i \in \mathcal{P} \tag{4.1.2}$$

$$A_i x_i + B_i q = 0, \ (y_i) \quad i \in \mathcal{P}. \tag{4.1.3}$$

Here, $\mathcal{P} := \{1, \ldots, P\}$ is a set of block variable partitions. Each partition contains a vector of primal variables $x_i \in \mathbb{R}^{n_{x_i}}$ and the vector $q \in \mathbb{R}^{n_q}$ contains the primal variables that the couple partitions. The total number of primal variables is $n := n_q + \sum_{i\in\mathcal{P}} n_{x_i}$. Equation (4.1.2) are the partition constraints with their respective dual variables $\lambda_i \in \mathbb{R}^{m_i}$. Equation (4.1.3) are the constraints that *link* partitions

across set $\mathcal{P}$ and have associated dual variables $y_i \in \mathbb{R}^{l_i}$. We assume that the partition matrices $J_i \in \mathbb{R}^{m_i \times n_i}$ have full rank and that the right-hand-side coefficients $b_i \in \mathbb{R}^{m_i}$ are in the column space of J_i. The total number of partition constraints is $m := \sum_{i \in \mathcal{P}} m_i$. We refer to $A_i \in \mathbb{R}^{n_q \times n_i}$ and $B_i \in \mathbb{R}^{n_q \times n_q}$ as linking matrices and we assume them to have full rank. The total number of linking constraints is $l := \sum_{i \in \mathcal{P}} l_i$. The QP under study is the main computational kernel behind nonlinear programming strategies since it is used for computing primal-dual steps.

We make the blanket assumption that the block-structured QP is strongly convex and that the combined Jacobian matrix (obtained by assembling partition and coupling constraints) has full rank. Strong convexity can be obtained by ensuring that all block Hessian matrices D_i are positive definite (and thus the entire Hessian is positive definite). Strong convexity and full-rank conditions guarantee that the primal-dual solution of the QP exists and is unique. Moreover, these assumptions guarantee that the QP solution is a unique minimizer and that this can be found by solving the first-order stationarity conditions. Additional assumptions will also be needed on the nature of the building blocks of the QP (associated with each partition). Such assumptions are needed to ensure that proposed decomposition schemes are well-defined. Such assumptions will be stated as we proceed (in order to maintain clarity in the presentation).

The Lagrange function of (4.1) can be expressed as:

$$\mathcal{L}(x, q, \lambda, y) = \sum_{i \in \mathcal{P}} \frac{1}{2} x_i^T D_i x_i + c_i^T x_i + y_i^T (A_i x_i + B_i q) + \lambda_i^T (J_i x_i - b_i), \tag{4.2}$$

where $x := (x_1, \cdots, x_P)$, $\lambda := (\lambda_1, \cdots, \lambda_P)$ and $y := (y_1, \cdots, y_P)$. The first-order optimality conditions of (4.1) are given by:

$$\begin{aligned}
\nabla_{x_i} \mathcal{L} &= 0 = \quad D_i x_i + c_i + J_i^T \lambda_i + A_i^T y_i, \ i \in \mathcal{P} \\
\nabla_{\lambda_i} \mathcal{L} &= 0 = \quad J_i x_i - b_i, \ i \in \mathcal{P} \\
\nabla_{y_i} \mathcal{L} &= 0 = \quad A_i x_i - B_i q, \ i \in \mathcal{P} \\
\nabla_q \mathcal{L} &= 0 = \quad \sum_{i \in \mathcal{P}} B_i^T y_i.
\end{aligned} \tag{4.3}$$

These conditions form a *block-structured* linear system of the form shown in (4.4). For the sake of compactness and ease of notation, we rewrite (4.4) as:

$$\begin{bmatrix} D_1 & J_1^T & & & & & A_1^T & & \\ J_1 & & & & & & & & \\ & \ddots & & & & & \ddots & & \\ & & D_P & J_P^T & & & & A_P^T & \\ & & J_P & & & & & & \\ & & & & & B_1^T & \cdots & B_P^T & \\ A_1 & & & & B_1 & & & & \\ & \ddots & & & \vdots & & & & \\ & & A_P & & B_P & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ \lambda_1 \\ \vdots \\ x_P \\ \lambda_P \\ q \\ y_1 \\ \vdots \\ y_P \end{bmatrix} = \begin{bmatrix} -c_1 \\ b_1 \\ \vdots \\ -c_P \\ b_P \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.4}$$

For the sake of compactness and ease of notation, we rewrite (4.4) as:

$$\underbrace{\begin{bmatrix} K & & A^T \\ & & B^T \\ A & B & \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} v \\ z \\ y \end{bmatrix}}_{u} = \underbrace{\begin{bmatrix} \gamma \\ 0 \\ 0 \end{bmatrix}}_{r}. \tag{4.5}$$

where $v = (v_1, \cdots, v_P)$, $v_i = (x_i, \lambda_i)$, $\gamma = (\gamma_1, \cdots, \gamma_P)$, $\gamma_i = (-c_i, b_i)$, $u = (v, q, y)$, $r = (\gamma, 0, 0)$, $K = \text{blkdiag}\{K_1, \cdots, K_P\}$. We also have $A = \text{blkdiag}\{\tilde{A}_1, \cdots, \tilde{A}_P\}$, $B = \text{rowstack}\{B_1 \cdots, B_P\}$ with:

$$K_i = \begin{bmatrix} D_i & J_i^T \\ J_i & \end{bmatrix} \qquad \tilde{A}_i = \begin{bmatrix} A_i & 0 \end{bmatrix}, \; i \in \mathcal{P} \tag{4.6}$$

4.2.1 Solution using Schur Decomposition

One can solve large instances of the block-structured QP by using a Schur-complement decomposition method (we refer to this as Schur decomposition) Zhang

[2005]. This approach decomposes (4.1) by using block Gaussian elimination on a permuted version of the linear system (4.4). The permuted system has the structure:

$$\begin{bmatrix} D_1 & J_1^T & A_1^T & & & & & & \\ J_1 & & & & & & & & \\ A_1 & & & & & & & B_1 & \\ & & & \ddots & & & & \vdots & \\ & & & & D_P & J_P^T & A_P^T & & \\ & & & & J_P & & & & \\ & & & & A_P & & & B_P & \\ & & B_1^T & \cdots & & & B_P^T & & \end{bmatrix} \begin{bmatrix} x_1 \\ \lambda_1 \\ y_1 \\ \vdots \\ x_P \\ \lambda_P \\ y_P \\ q \end{bmatrix} = \begin{bmatrix} -c_1 \\ b_1 \\ 0 \\ \vdots \\ -c_P \\ b_P \\ 0 \\ 0 \end{bmatrix} \tag{4.7}$$

This system can be expressed in compact form as:

$$\begin{bmatrix} K_s & B_s \\ B_s^T & \end{bmatrix} \begin{bmatrix} v_s \\ q \end{bmatrix} = \begin{bmatrix} \gamma_s \\ 0 \end{bmatrix} \tag{4.8}$$

where $v_s = (v_{s_1}, \cdots, v_{s_P})$, $v_{s_i} = (x_i, \lambda_i, y_i)$, $\gamma_s = (\gamma_{s_1}, \cdots, \gamma_{s_P})$, and $\gamma_i = (-c_i, b_i, 0)$. We also have $K_s = \text{blkdiag}\{K_{s_1}, \cdots, K_{s_P}\}$ and $B_s = \text{rowstack}\{B_{s_1}, \cdots, B_{s_P}\}$ with:

$$K_{s_i} = \begin{bmatrix} D_i & J_i^T & A^T \\ J_i & & \\ A & & \end{bmatrix} \qquad B_{s_i} = \begin{bmatrix} 0 & 0 & B_i^T \end{bmatrix}^T, \; i \in \mathcal{P} \tag{4.9}$$

Since we have assumed that D_i is positive definite and the combined constraint Jacobian (J_i, A) is full rank, we have that K_{s_i} is nonsingular. Since K_s is block-diagonal this implies this is non-singular as well. As a result, we can form the Schur complement system:

$$(B_s^T K_s^{-1} B_s) q = B_s^T K_s^{-1} \gamma_s \tag{4.10}$$

We refer to system (4.10) as the Schur complement system and to its coefficient matrix as the Schur complement. Since K_s is block-diagonal, one can form the Schur complement by factorizing the blocks K_{s_i} independently. Using the assembled Schur complement, one factories the Schur complement matrix and solves system (4.10) to find a solution for q. Having q, one then proceeds to find solutions for the partition variables v_s by solving the following system:

$$K_s v_s = \gamma_s - B_s q \tag{4.11}$$

Here, again, one can solve for each element v_{s_i} independently because K_s is block-diagonal. The Schur decomposition method is summarized in Algorithm 7. We refer the reader to [Kang et al., 2014] for details on the implementation of Schur decomposition approaches.

Algorithm 7: Schur Decomposition for Block-Structured QP

Let $S = 0$ and $r_{sc} = 0$
Factorize K_s matrix :
foreach $i \in \mathcal{P}$ **do**
    Factorize K_{s_i}
Form Schur complement system:
foreach $i \in \mathcal{P}$ **do**
    $S = S + B_i^T K_{s_i}^{-1} B_i$
    $r_{sc} = r_{sc} + B_i^T K_{s_i}^{-1} \gamma_{s_i}$
Factorize S and compute coupling variables by solving:
    $Sq = r_{sc}$
Compute partition variables:
foreach $i \in \mathcal{P}$ **do**
    $K_{s_i} v_{s_i} = \gamma_{s_i} - B_i q$

Schur decomposition is a flexible approach that enables the solution of problems with many block partitions. A fundamental limitation of this approach, how-

ever, is that one needs to form and factorize the Schur complement matrix S (which can be highly dense). As a result, Schur decomposition does not scale well with the number of coupling variables. Iterative approaches can in principle be used to solve the Schur system but general preconditioning strategies do not exist for general block-structured systems.

4.2.2 Solution using ADMM

The block-structured QP can also be decomposed and solved using ADMM. This approach seeks to minimize the augmented Lagrangian function:

$$\mathcal{L}_\rho(x, z, \lambda, y) = \sum_{i \in \mathcal{P}} x_i^T D_i x_i + c_i^T x_i + (A_i x_i + B_i z)^T y_i + \frac{\rho}{2} \|A_i x_i + B_i z\|^2 \qquad (4.12)$$

by alternating minimization with respect to the block variables (x_i, y_i) and the coupling variables q.

Algorithm 8: ADMM for Block-Structured QP

Input: Starting point $u^0 = (v^0, y^0, q^0)$, maximum number of iterations N_{ADMM}, penalty parameter $\rho > 0$, and convergence tolerance $\epsilon > 0$

for $k = 0, 1, 2, \ldots, N$ **do**
  Update partition variables:
  foreach $i \in \mathcal{P}$ **do**
    $x_i^{k+1} = \underset{x_i \in \mathcal{X}_i}{\arg\min}\, x_i^T D_i x_i + c_i^T x_i + \left(A_i x_i + B_i q^k\right)^T y_i^k + \frac{\rho}{2} \|A_i x_i + B_i q^k\|^2$
  Update coupling variables:
  $q^{k+1} = \underset{z}{\arg\min} \left(A_i x_i^{k+1} + B_i q\right)^T y_i^k + \frac{\rho}{2} \|A_i x_i^{k+1} + B_i q\|^2$
  Update dual variables:
  foreach $i \in \mathcal{P}$ **do**
    $y_i^{k+1} = y_i^k + \rho \left(A_i x_i^{k+1} + B_i q^{k+1}\right)$
  if $\|y^{k+1} - y^k\| \leq \epsilon$ *and* $\|\rho A^T B \cdot (q^{k+1} - q^k)\| \leq \epsilon$ **then**
    stop

Output: u

In Algorithm 8, $\mathcal{X}_i = \{x \mid J_i x - b_i = 0\}$ is used to denote the feasible set of each partition (the inner block constraints are satisfied exactly). The ADMM algorithm can be implemented by solving the first-order conditions of each subproblem directly. This is because each block subproblem is strongly convex and the block Jacobian has full rank. This approach is sketched in Algorithm 9.

Algorithm 9: ADMM(u^0, N, ρ)

Input: starting point $u^0 = (v^0, y^0, q^0)$, maximum number of iterations N_{ADMM}, penalty parameter $\rho > 0$, and convergence tolerance $\epsilon > 0$

Factorize K_ρ and $B^T B$ matrix:

foreach $i \in \mathcal{P}$ **do**

  Factorize partition matrices K_{ρ_i} and $B_i^T B_i$

for $k = 0, 1, 2, \ldots, N$ **do**

  Update partition variables :

  foreach $i \in \mathcal{P}$ **do**

    $$K_{\rho_i} v^{k+1} = -\left(\gamma_i + \begin{bmatrix} \rho A_i^T B_i q^k \\ 0 \end{bmatrix} + \begin{bmatrix} A_i^T y_i^k \\ 0 \end{bmatrix}\right)$$

  Update coupling variables:

  $$z^{k+1} = -[B^T B]^{-1}\left(B^T A v^{k+1} + \tfrac{1}{\rho} B^T y^k\right)$$

  Update dual variables:

  foreach $i \in \mathcal{P}$ **do**

    $$y_i^{k+1} = y_i^k + \rho\left(\tilde{A}_i v_i^{k+1} + B_i q^{k+1}\right)$$

  if $\|y^{k+1} - y^k\| \leq \epsilon$ *and* $\|\rho A^T B \cdot (q^{k+1} - q^k)\| \leq \epsilon$ **then**

    stop

Output: u

In the above algorithm we have that $K_\rho = \text{blkdiag}\{K_{\rho_1}, \cdots, K_{\rho_P}\}$ with

$$K_{\rho_i} = \begin{bmatrix} D_i + \rho A_i^T A_i & J_i^T \\ J_i & 0 \end{bmatrix}. \tag{4.13}$$

We note that the update of the coupling variables still requires forming and factorizing the matrix $B^T B$ but this can be done in blocks by forming and factorizing $B_i^T B_i$ individually. Moreover, this operation only needs to be performed once. Note also that $B^T B$ is invertible since B has full rank. As a result, the update step for the coupling variables in ADMM is cheaper than that of Schur decomposition and can thus overcome the main computational bottleneck of the later. Unfortunately, it is well-known that ADMM exhibits slow convergence and thus the ability to perform fast operations might be shadowed by the need to perform many iterations.

4.2.3 Solution using ADMM-GMRES

The key observation that motivates our work is that ADMM can be used as a *preconditioner* for iterative linear algebra techniques such as GMRES. To derive the ADMM preconditioning strategy, we consider the *regularized* QP (4.1):

$$\min_{x_i,z} \quad \sum_{i\in\mathcal{P}} \frac{1}{2} x_i^T D_i x_i + c_i^T x_i + \frac{\rho}{2}\|Ax_i + B_i q\|^2 \tag{4.14.1}$$

$$\text{s.t.} \quad J_i x_i = b_i, \qquad (\lambda_i) \quad i \in \mathcal{P} \tag{4.14.2}$$

$$A_i x_i + B_i q = 0, \ (y_i) \quad i \in \mathcal{P}. \tag{4.14.3}$$

The solution of this problem is also a solution of (4.1) (since the penalization term vanishes at the solution). The optimality conditions of the regularized QP are given by:

$$\underbrace{\begin{bmatrix} K_\rho & \rho A^T B & A^T \\ \rho B^T A & \rho B^T B & B^T \\ A & B & \end{bmatrix}}_{H_\rho} \underbrace{\begin{bmatrix} v \\ q \\ y \end{bmatrix}}_{u} = \underbrace{\begin{bmatrix} \gamma \\ 0 \\ 0 \end{bmatrix}}_{r}. \tag{4.15}$$

We refer to (4.15) as the *KKT system* and to H_ρ as the *KKT matrix*. ADMM can be interpreted as a Gauss-Seidel minimization of the block and coupling variables and the dual variables [Boyd et al., 2011, Eckstein and Bertsekas, 1992b, Rodriguez et al., 2018]. This induces a splitting operator $H_\rho = M_\rho - N_\rho$ satisfying:

$$\underbrace{\begin{bmatrix} K_\rho & \rho A^T B & A^T \\ \rho B^T A & \rho B^T B & B^T \\ A & B & \end{bmatrix}}_{H_\rho} = \underbrace{\begin{bmatrix} K_\rho & & \\ \rho B^T A & \rho B^T B & \\ A & B & -\frac{1}{\rho} I \end{bmatrix}}_{M_\rho} - \underbrace{\begin{bmatrix} -\rho A^T B & -A^T \\ & -B^T \\ & -\frac{1}{\rho} I \end{bmatrix}}_{N_\rho} \tag{4.16}$$

Applying splitting (4.16) to (4.15) gives the operator:

$$T_\rho(u) := G_\rho u + f_\rho \tag{4.17}$$

where $G_\rho = M_\rho^{-1} N_\rho$ and $f_\rho = M_\rho^{-1} r$. Note that any u satisfiying the fixed-point $T_\rho(u) = u$ satisfies $(I - G_\rho)u = f_\rho$ and is a solution of the preconditioned KKT system:

$$M_\rho^{-1} H_\rho u = M_\rho^{-1} r. \tag{4.18}$$

This follows from $M_\rho^{-1} H_\rho = M_\rho^{-1} (M_\rho - N_\rho) = I - G_\rho$. This motivates the development of a Richardson recursion of the form $u^{k+1} = G_\rho u^k + f_\rho$, which converges to a u satisfying $(I - G_\rho)u = f_\rho$ and $M_\rho^{-1} H_\rho u = M_\rho^{-1} r$ (provided that the eigenvalues of G_ρ are inside the unit circle). In Appendix D we show that the operator $T_\rho(u)$ can be computed by performing one ADMM iteration (using u as stating point). In other words, we have that $T_\rho(u) =$`ADMM`(u, $N_{ADMM} = 1$, ρ). This also implies that the Richardson recursion can be written as $u^{k+1} =$`ADMM`(u^k, $N_{ADMM} = 1$, ρ). We thus have that the Richardson recursion (and thus ADMM) are consistent preconditioner choices.

The key idea behind ADMM-GMRES is to solve the system $M_\rho^{-1} H_\rho u = M_\rho^{-1} r$ by using the Krylov solver GMRES. This is equivalent to solving $(I - G_\rho)u = f_\rho$. The

right-hand side of this system can be computed as $f_\rho = T_\rho(0)$. GMRES requires the computation of matrix-vector products with the preconditioned coefficient matrix of the form $M_\rho^{-1} H_\rho h = (I - G_\rho)h$. This can be done by using the operator (4.17) as:

$$M_\rho^{-1} H_\rho h = h - [T_\rho(h) - T_\rho(0)]\,, \tag{4.19}$$

This follows from the observation that:

$$\begin{aligned} M_\rho^{-1} H_\rho h &= h - [T_\rho(h) - T_\rho(0)] \\ &= h - [G_\rho h + f_\rho - f_\rho] \\ &= h - G_\rho h \\ &= (I - G_\rho)h. \end{aligned} \tag{4.20}$$

From (4.20) we note that asking the ADMM oracle to iterate until reaching convergence will deliver $T_\rho(h) = h$ satisfying $M_\rho^{-1} H_\rho h = f_\rho$. In such a case, the ADMM preconditioner is perfect (since it solves the actual preconditioned KKT system). Consequently, the quality of the ADMM preconditioner will improve as one increases N_{ADMM}. For details on the properties of the preconditioner we refer the reader to [Zhang and White, 2018, 2016]. The ADMM-GMRES strategy is summarized in Algorithm 10.

Algorithm 10: `ADMM_GMRES`(N_{GMRES}, N_{ADMM}, ρ)

Input: maximum number of GMRES iterations N_{GMRES}, maximum number of ADMM iterations N_{ADMM}, penalty parameter $\rho > 0$, and tolerance $\epsilon > 0$

1 Compute right-hand-side vector:

2 $f_\rho =$ `ADMM`(0, N_{ADMM}, ρ)

3 Call GMRES solver[a]:

4 $u =$ `GMRES`$(I - G_\rho, f_\rho, N_{\text{GMRES}}, \epsilon)$

Output: u

[a] Matrix-vector products are computed as $(I - G_\rho)h = h - (T_\rho(h) - f_\rho)$, where $T_\rho(h) =$`ADMM`(h, $N_{\text{ADMM}} = 1$, ρ).

4.3 Numerical Results

In this section we discuss the implementation of the ADMM-GMRES algorithm and present results for different benchmark problems. All numerical experiments were run using `PyNumero`, which is an open-source framework written in Python and C++ that combines modeling capabilities of the algebraic modeling language `Pyomo` [Hart et al., 2017] with efficient libraries like the `AMPL` solver library [Fourer et al., 1993], the Harwell Subroutine Library (HSL), and NumPy/SciPy [Jones et al., 2001–]. It uses object-oriented principles that facilitate the implementation of algorithms and problem formulations that exploit block-structures via polymorphism and inheritance. All these features facilitate the implementation of ADMM, Schur decomposition, and ADMM-GMRES. The optimization models were implemented in `Pyomo/PyNumero` and all linear algebra operations were performed in compiled-code. Within `PyNumero`, we used an `MA27` interface to perform all direct linear algebra operations. We use the `GMRES` implementation within `Scipy` to perform all iterative linear algebra operations. Iterative linear algebra routines available in `KRYPY` [Gaul and Schlmer, 2015] were also used to validate results. To implement the power grid models we used `EGRET`, a Pyomo-based package that facilitates the formulation if optimization problems that arise in power systems. The stopping criterion for GMRES and ADMM is that the norm of the KKT system residual is smaller than 1×10^{-8}. If the residual convergence criterion was not satisfied after 2,000 iterations, the algorithm was aborted and we report the norm of the residual achieved. All matrix factorizations were obtained using `MA27` with a pivoting tolerance of 1×10^{-8}

4.3.1 Standard Benchmark Problems

We first conducted tests with randomly generated instances to study qualitatively the performance of ADMM-GMRES on block-structured optimization problems. This section focuses on two-stage stochastic programs of the form:

$$\min_{x_i,z} \quad \sum_{i\in\mathcal{P}} \frac{1}{2} x_i^T D x_i + c^T x_i \tag{4.21.1}$$

$$\text{s.t.} \quad Jx_i = b_i, \qquad (\lambda_i) \quad i \in \mathcal{P} \tag{4.21.2}$$

$$A_i x_i - q = 0, \qquad (y_i) \quad i \in \mathcal{P} \tag{4.21.3}$$

where $\mathcal{P}$ is the scenario set, x_i are the second-stage (recourse) variables, and z are first-stage (coupling) variables. We defined a nominal vector b and create scenarios with right-hand-side vector b_i using the nominal vector b as mean and using a standard deviation $\sigma = 0.5b$. We first demonstrate the scalability of Algorithm 10 when solving instances of problem (4.21) with high dimensionality in the coupling variables z. The stochastic problem was constructed in the following manner: D was set to a $4,800 \times 4,800$ block diagonal matrix with 16 dense symmetric blocks. Each block was generated following Algorithm 14 from [Zhang and White, 2018] with log-standard-deviation $s = 0.5$ (see [Zhang and White, 2018] for details). The random matrix J is of size $100 \times 4,800$. The number of scenarios was set to 50 giving an initial problem with $240,000$ variables and $5,000$ constraints. To investigate the scalability of Algorithm 10 to solve (4.21), the number of complicating variables was varied from 100 to $4,000$. Note that, as n_q increases, the number of constraints of (4.21) increases by $50n_q$, while the number of variables by n_q. The largest problem solved contained $244,000$ variables and $205,000$ constraints.

We solved (4.21) using four different strategies. Figure 4.1 depicts the results obtained with Schur decomposition, GMRES (without preconditioner), ADMM, and ADMM-GMRES. These results confirm the observations of Section 4.2. Specifically, Schur decomposition does not scale well as n_q increases. The main reason

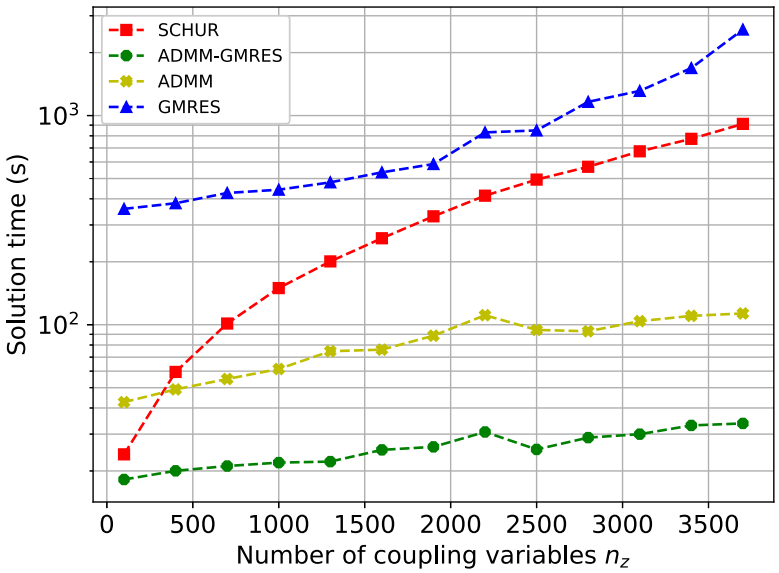


Figure 4.1.: Scalability analysis of Schur decomposition, GMRES (without preconditioner), ADMM, and ADMM-GMRES.

for this behavior is that, as n_q increases, the number of operations required to form the Schur-complement increases significantly. In addition, because the Schur-complement matrix is a dense $n_q \times n_q$ matrix, the factorization time increases cubically as n_q increases. We observe that GMRES takes the longest time to solve the problem. For ADMM and ADMM-GMRES we see more favorable scalability as n_q increases. Specifically, we see that ADMM-GMRES always converges faster than the rest of the methods and the savings increase as the number of coupling variables increases. We also note that ADMM-GMRES mimic the trend in performance of ADMM but is significantly faster.

Figure 4.2 shows the residuals for the iterative approaches for a problem with $1,000$ complicating variables. We can see that all methods exhibit linear convergence but that ADMM-GMRES outperforms the other two methods. Notably, ADMM-GMRES converges in just 35 iterations while ADMM and GMRES require over 300 iterations and $1,000$ iterations, respectively.

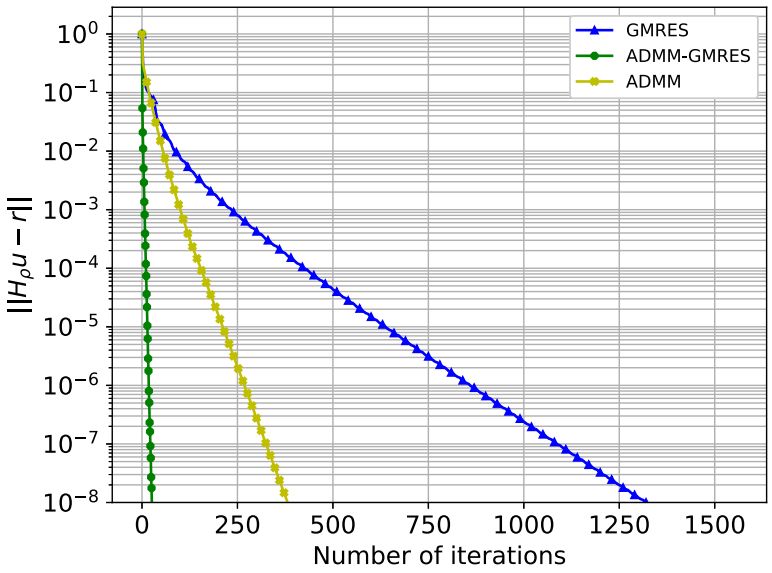
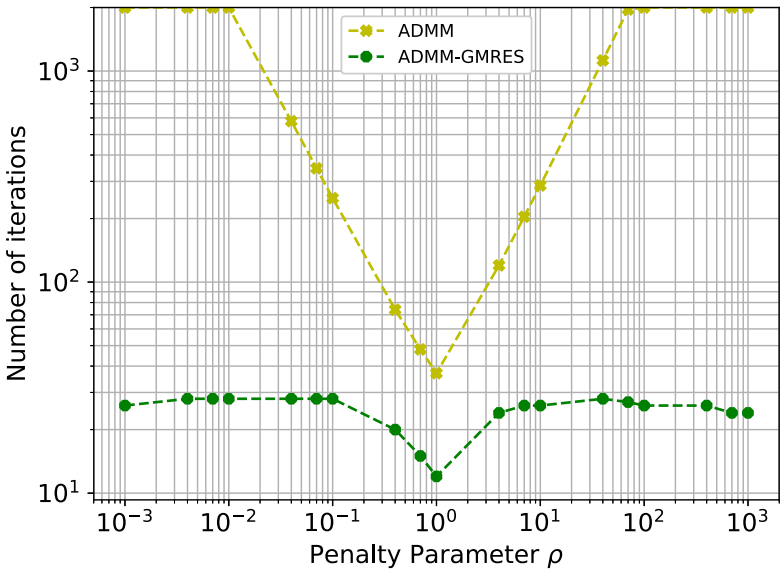


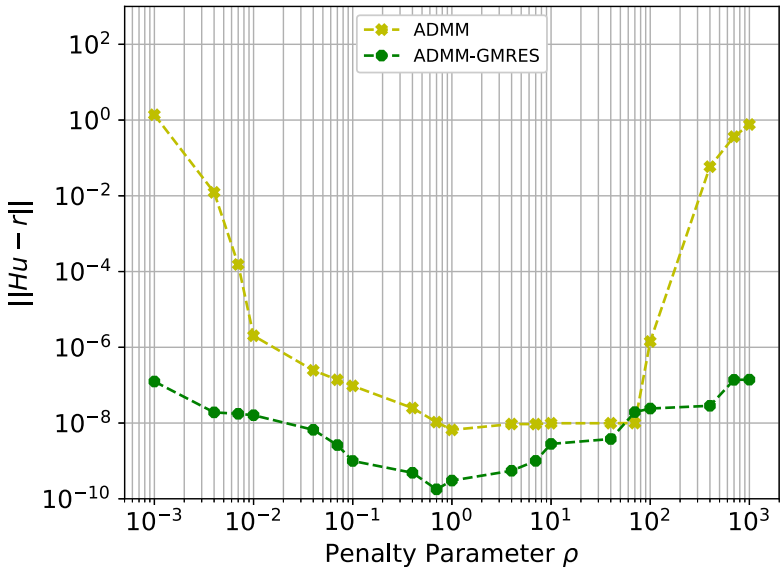
Figure 4.2.: Evolution of residuals for GMRES (without preconditioner), ADMM, and ADMM-GMRES.

An important drawback of ADMM is the need to tune the penalty parameter ρ. The work in [Ghadimi et al., 2015] shows that an optimal value for ρ can be chosen based on the smallest and largest eigenvalues of the matrix $A^T D^{-1} A$. In principle, this selection of ρ is optimal for ADMM-GMRES as well. However, for large-scale structured problems such as the ones considered here, computing the eigenvalues of $A^T D^{-1} A$ is expensive. Heuristic approaches have also been proposed to select ρ at every ADMM iteration with the objective of accelerating convergence [Wohlberg, 2017]. Unfortunately these heuristics do not provide guarantees and might incur additional overhead. In particular, for the QP problems considered here, varying ρ at every ADMM iteration will require forming and factorizing the K_{ρ_i} repetitively. We now demonstrate that, interestingly, ADMM-GMRES is fairly insensitive to the choice of ρ.

Figure 4.3 compares the performance of ADMM against that of ADMM-GMRES for a stochastic program with $n_q = 1,000$ and different values of ρ ranging from 10^{-3} to 10^{3}. Here we measure performance in terms of the number of iterations.



(a) Number of iterations to satisfy convergence criteria



(b) Error with respect to exact solution

Figure 4.3.: Sensitivity of ADMM and ADMM-GMRES to penalty parameter ρ.

Our results on the block-structured problem are in agreement with those in [Zhang and White, 2016] for the single block problem. We see that ADMM-GMRES is remarkably robust at solving (4.21) regardless of the choice of ρ (the number of

iterations always stays below 30). ADMM, on the other hand, fails to converge within 2,000 iterations for small and large values of ρ. The superior performance of ADMM-GMRES is attributed to the fact that the selection of ρ only has an effect on the preconditioner and not the convergence of GMRES itself. Nevertheless, we observe that an optimal selection of ρ improves performance of both ADMM and ADMM-GMRES. The robustness of ADMM-GMRES is a desirable feature when using the solver within more advanced SQP-based solvers. In particular, recent developments of augmented lagrangian interior-point approaches [Chiang et al., 2017, Kuhlmann and Bskens, 2018] provide promising frameworks for ADMM-GMRES since the selection of ρ is made based on information from the outer-iteration of the interior-point [Armand et al., 2014, Armand and Omheni, 2017]. Finally, recent developments of inertia-free methods for nonconvex nonlinear optimization enable the use of iterative linear solvers for the Newton subproblem.

4.3.2 Optimal Power Flow Problems

We demonstrate the computational benefits of using ADMM-GMRES by solving stochastic optimal power flow problems. The optimal power flow problem is frequently used in power networks to determine an efficient dispatch of power generators that satisfy demand and maintains feasible operation conditions. This approach assumes that demand forecasts are accurate and determines a nominal operation point for power generation, power flow in transmission lines and voltage angle at each bus in the power grid. We solve the DC-power flow problem for 35 benchmarks available in the `PGLIB_OPF` library [Zimmerman et al., 2011] and determine nominal operation points for each of them. The solution of each benchmark was obtained using `Ipopt`.

To asses the computational performance of ADMM-GMRES, we formulated a set-point problem that uses the nominal solution of the DC-power flow problem but

seeks to minimize the effect of potential uncertainty in electricity demand values. The optimization solved is the quadratic problem:

$$\min_{x_i,z} \quad \sum_{s\in\mathcal{P}}\sum_{j\in\Omega_G} w_j(P_{G_{j,s}} - P^{\dagger}_{G_j})^2 + \sum_{i,j\in\Omega_L} w_{i,j}(P_{F_{i,j,s}} - P^{\dagger}_{F_{i,j}})^2 + \sum_{j\in\Omega_{\mathcal{B}}} w_j(\theta_{j,s} - \theta^{\dagger}_j)^2 \tag{4.22.1}$$

$$\text{s.t.} \quad \sum_{j\in\Omega_{G_i}} P_{G_{j,s}} - P_{L_{i,s}} = \sum_{j\in\Omega_i} P_{F_{i,j,s}}, \quad i\in\Omega_{\mathcal{B}}, s\in\mathcal{P} \tag{4.22.2}$$

$$P_{F_{i,j,s}} = \frac{\theta_{i,s} - \theta_{j,s}}{X_{i,j}}, \qquad i,j\in\Omega_L, s\in\mathcal{P} \tag{4.22.3}$$

$$P_{G_{j,s}} - z_j = 0, \qquad j\in\Omega_G, s\in\mathcal{P} \tag{4.22.4}$$

$$\theta_{0,s} = 0, \qquad s\in\mathcal{P}. \tag{4.22.5}$$

Here, $\Omega_{\mathcal{B}}$ and Ω_L denote the set of buses and transmission lines in the network, Ω_G the set of generators, Ω_{G_i} the set of generators at bus i, and Ω_i the set of buses connected to bus i. The variables in the model are the generator outputs P_G , the flow in the transmission lines P_F, and the voltage angles θ_j. As parameters we have the reactance of the lines $X_{i,j}$, the loads P_L, and the set-point values $P^{\dagger}_G$, $P^{\dagger}_F$, and $\theta^{\dagger}$ obtained from the DC-power flow solution. We denote the reference bus as θ_0 and define objective weight values w. The goal of formulation (4.22) is to find the closest feasible operation to the optimal DC-power flow solution while accounting for potential uncertainty in the demands. In this problem the first-stage variables are the output of the generators for which we use Equation (4.22.4) to enforce the same power generation across the set of scenarios. The dimensionality of the first-stage of this problem is given by the number of generators in the power network. Hence, this number varies from 3 to $4,092$ in the 35 different benchmark problems considered in our study. For each benchmark we generated 50 random scenarios with normal random distributed noise on the load P_L

Tables 4.1 and 4.2 summarize the results for the 35 benchmarks. The problems were sorted according to their number of coupling variables. Table 4.1 presents the results for benchmarks with a first-stage dimension greater than 100. Results for the smaller benchmarks are shown in Table 4.2. We see that, for the smaller problems, Schur decomposition is the best alternative as it can give exact solutions in about the same time as ADMM. However, for all benchmarks shown in Table 4.1, ADMM-GMRES finds an ϵ-accurate solution in less time than ADMM and Schur decomposition. In particular, for case13659_pegase (with 4,091 first-stage variables), ADMM-GMRES solved the problem almost an order of magnitude faster than Schur decomposition. By looking at the trend for the rest of the problems, these scalability results can be expected to hold as the number of coupling variables increases. We highlight that, for many of the problems shown in Table 4.1, ADMM does not convergence after 2000 iterations; ADMM-GMRES, on the other hand, consistently achieves ϵ-accurate solutions in few iterations and regardless of the choice of ρ. In summary, our results demonstrate that ADMM-GMRES provides a plausible approach to overcome the limitations of Schur complement decomposition.

4.4 Summary

We have demonstrated that ADMM provides an effective mechanism to precondition iterative linear solvers and can overcome scalability limitations of Schur complement decomposition. Our results also demonstrate that the approach is robust to the choice of the penalty parameter. As part of future work, we will investigate the performance of ADMM-GMRES within a nonlinear interior-point framework. Here, it will be necessary to relax our assumptions on strong convexity and on the full rank of the Jacobian. Preliminary results reported in the literature indicate that different types of primal-dual regularized KKT systems can be used to compute search steps within interior-point methods under such relaxed

Table 4.1.: Performance for ADMM-GMRES (ADGM), ADMM, and Schur decomposition (Schur) on problem (4.22) with $n_q \geq 100$.

Pglib-matpower Case	n_x	n_q	Solution Time (s)			Iteration Count		$\|Hu - r\|$		
			Schur	ADMM	ADGM	ADMM	ADGM	Schur	ADMM	ADGM
P1. case13659_pegase	2115450	4091	1670.692	723.158	183.298	2000	325	2.798E-09	3.466E-03	1.602E-08
P2. case9241_pegase	1408950	1444	369.058	507.632	81.278	2000	165	3.841E-09	3.869E-05	2.981E-09
P3. case6470_rte	849800	760	124.416	336.881	51.106	2000	139	2.981E-10	3.932E-03	1.984E-10
P4. case6515_rte	845950	683	112.975	333.421	40.883	2000	109	3.206E-10	7.631E-04	2.627E-10
P5. case6495_rte	843650	679	111.035	320.943	37.302	2000	108	2.427E-10	7.128E-04	4.291E-10
P6. case2868_rte	389850	560	49.152	174.801	23.113	2000	105	3.228E-10	4.195E-04	1.404E-10
P7. case2848_rte	382250	510	43.584	168.000	22.508	2000	100	1.631E-10	2.329E-04	4.032E-11
P8. case2869_pegase	423500	509	48.424	189.774	38.293	2000	154	4.435E-10	1.135E-03	6.830E-10
P9. case6468_rte	813250	398	62.343	326.375	24.449	2000	65	2.774E-10	5.404E-04	4.416E-10
P10. case3012wp_k	367050	378	32.732	174.999	18.422	2000	74	7.032E-11	1.356E-05	1.338E-11
P11. case1951_rte	263900	365	24.864	132.780	19.566	2000	80	2.764E-10	2.836E-05	3.227E-10
P12. case2383wp_k	295900	319	24.458	152.102	17.499	2000	58	6.209E-11	7.444E-07	1.632E-10
P13. case1888_rte	249900	289	19.114	129.940	16.811	2000	66	1.534E-10	2.708E-06	2.935E-10
P14. case3120sp_k	367900	272	23.729	175.397	12.998	2000	56	3.402E-11	1.214E-08	2.220E-10
P15. case1354_pegase	193200	259	15.185	112.178	6.520	2000	47	1.453E-10	3.649E-07	2.728E-10
P16. case240_pserc	48650	142	5.262	64.665	4.737	2000	59	3.942E-10	1.902E-08	7.815E-10
P17. case2746wp_k	311600	103	8.455	100.004	11.080	1287	35	4.171E-11	9.925E-10	4.428E-10

Table 4.2.: Performance for ADMM-GMRES (ADGM), ADMM and Schur decomposition (Schur) on problem (4.22) with $n_q \leq 100$.

Pglib-matpower Case	n_x	n_q	Solution Time (s)			Iteration Count		$\|Hu - r\|$		
			Schur	ADMM	ADGM	ADMM	ADGM	Schur	ADMM	ADGM
P18. case73_ieee_rts	19200	95	3.161	21.751	2.914	772	38	1.001E-11	9.795E-10	6.237E-11
P19. case2746wop_k	311100	84	7.033	78.842	10.125	1022	34	2.763E-11	9.911E-10	1.306E-09
P20. case2736sp_k	308400	81	6.758	81.199	9.796	1054	35	2.538E-11	9.978E-10	4.499E-10
P21. case300_ieee	41200	56	2.212	41.976	6.157	1305	38	6.341E-11	9.891E-10	6.844E-10
P22. case2737sop_k	305650	53	4.668	54.118	9.791	696	34	1.808E-11	9.905E-10	3.925E-10
P23. case200_pserc	26000	37	1.440	13.064	3.150	428	28	3.524E-12	9.645E-10	2.619E-11
P24. case24_ieee_rts	6250	31	1.096	6.219	2.059	231	30	3.218E-12	9.918E-10	5.142E-11
P25. case118_ieee	17050	18	0.799	10.767	2.212	375	32	1.329E-11	9.662E-10	2.874E-10
P26. case162_ieee_dtc	23450	11	0.618	4.629	1.829	148	20	7.489E-12	8.752E-10	1.347E-10
P27. case89_pegase	16100	11	0.588	3.927	1.936	132	23	4.015E-12	9.344E-10	1.228E-10
P28. case39_epri	5200	9	0.476	3.471	1.516	124	18	8.340E-12	8.749E-10	1.543E-10
P29. case30_as	4100	5	0.360	2.386	0.573	81	11	1.632E-13	9.865E-09	5.271E-11
P30. case30_fsr	4100	5	0.362	2.047	0.577	73	11	1.997E-13	8.138E-09	3.175E-11
P31. case5_pjm	1000	4	0.319	2.707	0.474	103	8	2.434E-13	9.123E-09	3.137E-10
P32. case57_ieee	7200	3	0.314	2.382	0.747	82	7	7.034E-13	9.208E-09	2.171E-09
P33. case14_ieee	1850	1	0.237	1.343	0.346	48	3	1.355E-13	9.339E-09	3.045E-11
P34. case30_ieee	3700	1	0.243	1.357	0.354	47	3	5.790E-14	9.653E-09	6.383E-12
P35. case3_lmbd	450	1	0.235	1.233	0.342	44	3	1.515E-13	7.080E-09	1.520E-12

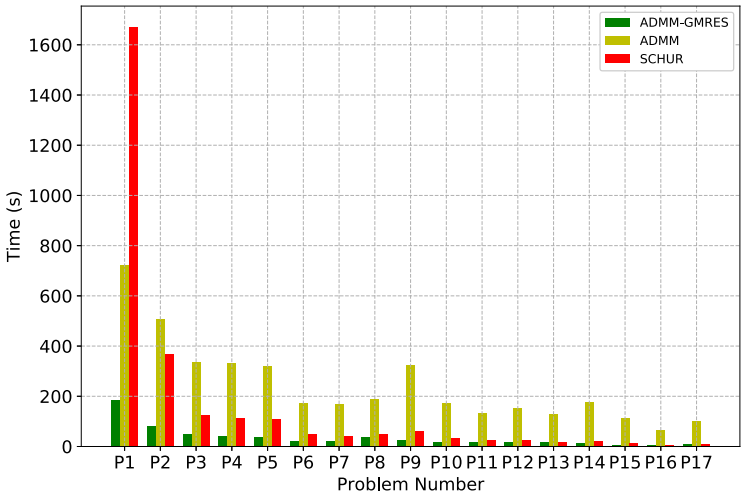


Figure 4.4.: Computational times for Schur decomposition, ADMM, and ADMM-GMRES for problems with $n_q \geq 100$.

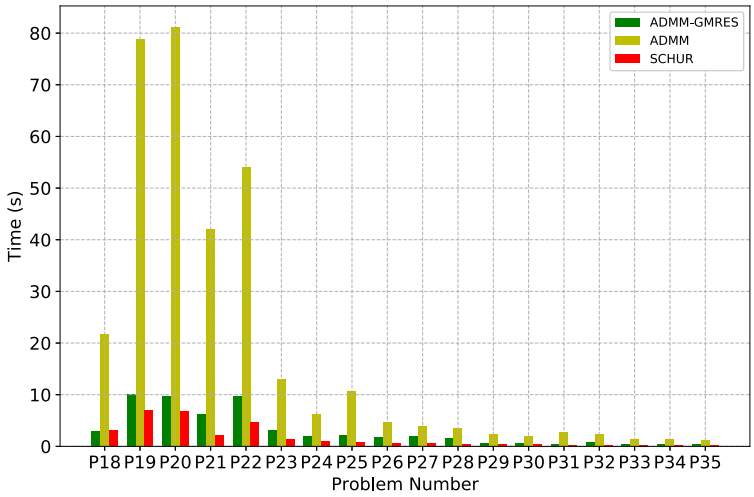


Figure 4.5.: Computational times for Schur decomposition, ADMM, and ADMM-GMRES for problems with $n_q < 100$.

conditions [Chiang et al., 2017]. For instance, the primal-dual regularized system correspond to the optimality conditions of the QP problem:

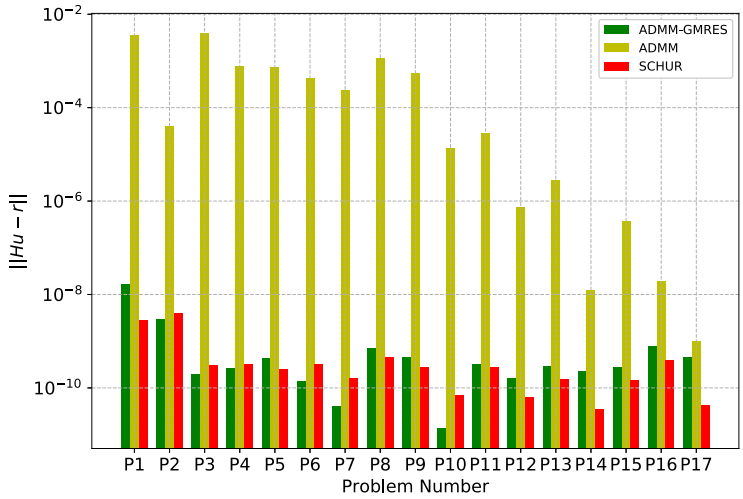


Figure 4.6.: Residuals for Schur decomposition, ADMM, and ADMM-GMRES for problems with $n_q \geq 100$.

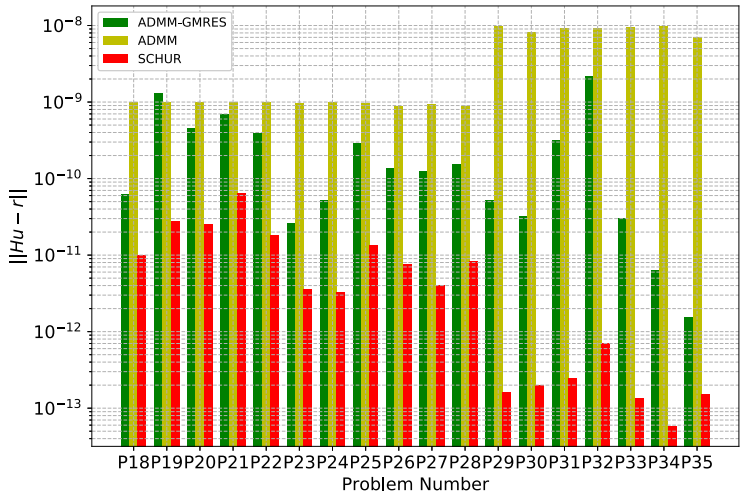


Figure 4.7.: Residuals for Schur decomposition, ADMM, and ADMM-GMRES for problems with $n_q < 100$.

$$\begin{array}{ll} \min\limits_{x,q,r} & \frac{1}{2}x^T(D+\delta I)x + c^T x + \frac{1}{2\rho}\|r\|^2 + \frac{\rho}{2}\|Ax + Bq - \frac{1}{\rho}r\|^2 \\ \text{s.t.} & Ax + Bq - \frac{1}{\rho}r \;=\; 0, \quad (y) \end{array} \tag{4.23}$$

We will investigate ADMM variants to precondition such systems. The effectiveness of using ADMM as a preconditioner makes us wonder whether other approaches can be used for preconditioning as well. For instance, inexact dual Newton strategies can potentially be used to precondition structured KKT systems. This is an interesting direction of future work. We will also investigate advanced ADMM strategies that use second-order multiplier updates to accelerate the preconditioner.

5. ACCELERATING ADMM WITH SECOND-ORDER UPDATES [1]

Many problem classes, including design under uncertainty are inherently structured and can be accelerated with decomposition approaches. This chapter describes a second-order multiplier update for the alternating direction method of multipliers to solve nonlinear stochastic programming problems. We exploit connections between ADMM and the Schur-complement decomposition to derive an accelerated version of ADMM. Specifically, we study the effectiveness of performing a Newton-Raphson algorithm to compute multiplier estimates for the method of multipliers and augmented Lagrangian methods in general. We interpret ADMM as a decomposable version of MM and propose modifications to the multiplier update of the standard ADMM scheme based on improvements observed in MM. The modifications to the ADMM algorithm seek to accelerate solutions of optimization problems for design under uncertainty and the numerical effectiveness of the approaches is demonstrated on a set of ten stochastic programming problems from literature. Practical strategies for improving computational performance are discussed along with comparisons between the algorithms. We observe that the second-order update achieves convergence in fewer unconstrained minimizations for MM on general nonlinear problems. In the case of ADMM, the second-order update reduces significantly the number of subproblem solves for convex quadratic programs.

[1]Part of this chapter is reprinted from
"Second-order Multiplier Updates to Accelerate ADMM Methods in Optimization Under Uncertainty" by Jose S. Rodriguez, Gabriel Hackebeil, John Siirola, Victor M. Zavala, Carl D. Laird, to appear in FOCAPD-19/Proceedings of the 9th International Conference on Foundations of Computer-Aided Process Design, 2019.

5.1 Preliminaries

Large-scale optimization models are used in many fields of science and engineering to provide solutions to problems. In particular, as uncertainty analysis becomes increasingly important in the areas of optimal design and manufacturing, there is a need for reliable and efficient methods to solve large-scale optimization under uncertainty problems. This has motivated extensive research in the field of decomposition algorithms since they allow considerable computational speedup in parallel computing environments while addressing memory requirements. The alternating direction method of multipliers [Boyd et al., 2011] has received particular attention due to its flexibility and ease of implementation for solving large-scale problems in a distributed matter. Being a first-order method, one drawback of ADMM is that it has a slow rate of convergence compared to advanced second-order methods. The Schur-complement decomposition has also received special attention for solving stochastic programming problems that arise in design under uncertainty [Biegler, 2017]. Schur-complement schemes have proven to be very effective for parallelizing the interior-point algorithm by taking advantage of block-structures in the KKT system of structured optimization problems [Gondzio and Grothey, 2009, Word, 2014, Kang et al., 2014]. Although ADMM has a slower convergence rate, it also has reduced communication requirements in parallel computing environments. In this work we derive a second-order multiplier update for ADMM that resembles the Schur-complement algorithm to accelerate its rate of convergence while keeping its ease of implementation and parallel computing benefits. This modification to the ADMM algorithm seeks to make the method more efficient for nonlinear optimization under uncertainty problems.

In this chapter we consider a problem of the form,

$$\min_{x_i,q} \quad \sum_{i\in\mathcal{P}} f_i(x_i) \tag{5.1.1}$$

$$\text{s.t.} \;\; c_i(x_i) \geq 0, \; i \in \mathcal{P} \tag{5.1.2}$$

$$A_i x_i + B_i q = 0, \; (y_i) \quad i \in \mathcal{P}. \tag{5.1.3}$$

where the vector $x_i \in \mathbb{R}^{n_{x_i}}$ contains all the variables corresponding to a block partition $i \in \mathcal{P} := \{1, ..., N\}$. We define the entire set of partition variables as $x = (x_1, ..., x_{|\mathcal{P}|})$. The vector $q \in \mathbb{R}^{n_q}$ contains the complicating (coupling) variables. The linking constraints (5.1.3) are defined using the mapping matrices A_i and B_i. The Lagrange multipliers of the coupling constraints are denoted as $y_i \in \mathbb{R}^{m_i}, \; i \in \mathcal{P}$. We also define the entire set of multipliers as $y = (y_1, ..., y_{|\mathcal{P}|})$.

This structure arises from design problems under uncertainty as multi-stage stochastic programming problems. For the particular case of two-stage stochastic problems, like those presented in Chapter 2, the coupling variables correspond to first-stage variables of the stochastic problem resulting in mapping matrices B_i being the identity matrix. The separable structure of problem (5.1) enables the implementation of decomposition techniques that can be used for distributed and parallel computing to avoid memory limitations and/or accelerate the solution of the optimization problem. These decomposition techniques rely on partitioning the problem to exploit the inherent separable structure. Such partitioning can be done externally at the problem level formulation (Chapter 3) or internally at the linear algebra level of a host algorithm (Chapter 2). Partitioning the problem externally is in general less intrusive, but typically exhibits linear or sublinear convergence rates. Partitioning the problem internally can provide improved convergence rates since the properties of the host algorithm are usually retained, however, these approaches are more intrusive and often require significantly more communication in parallel implementation. In this work we combine ideas from both approaches to overcome their individual limitations.

In Section 5.2 we derive a second-order update formula for the multiplier estimates in MM and ADMM. Connections between the second-order update and the Schur-complement decomposition are highlighted. Section 5.3 reviews the set benchmark stochastic programming problems used in this study. In section 5.4, numerical results for the case studies are provided. Section 5.5 closes the chapter with concluding remarks and directions for future work.

5.2 Second-Order update formula

The original method of multipliers from Hestenes [1969] follows a first-order formula to update multiplier estimates in every iteration of the algorithm. In view of the interpretation of the multiplier update as a dual ascent method, it is natural to consider a Newton approach for MM and it's derivative algorithms like ADMM. The update formula using a Newton-based approach is the following

$$y^{k+1} = y^k + \alpha[\nabla^2\phi(x^{k+1}, q^{k+1}y^k)]^{-1}\nabla\phi(x^{k+1}, q^{k+1}, y^k) \tag{5.2}$$

where $\phi(x, q, y) = \inf_x \mathcal{L}_\rho(x, q, y)$ is the dual function, $\nabla\phi(x, q, y) = c(x, q)$ its gradient, and $\nabla^2\phi(x, q, y)$ it's Hessian. Miele [1971b], Bertsekas [1976], Fletcher [1975], and others [Betts, 1977, Rupp, 1975, Glad, 1979] studied the local and global convergence properties of applying (5.2) to the standard MM. Miele [1971a] was the first to carry numerical experiments using both dual ascent and Newton update formulas for the multipliers on MM. Fletcher [1975] provided numerical evidence for the acceleration obtained with Newton updates together with a comparison with different penalty functions. Rupp [1975] and Bertsekas [1976] prove local convergence for the Newton step using second-order sufficient conditions. Buys [1972] and Tapia [1977] later proposed Quasi-Newton update formulas to carry out the multiplier update and overcome the difficulty of obtaining second-order derivative information.

Provided that the subproblems in ADMM are solved with modern nonlinear solvers (e.g. `Ipopt`, `Knitro`, `Snopt`) second-order derivative information is read-

ily available. Therefore, in this chapter, we explore the application of second-order updates within the ADMM framework.

To derive the second-order update formula consider using Newton method to solve the subproblem

$$\min_{x_i} \quad f_i(x_i) + \rho\|A_i x_i + B_i q^k\|^2 \tag{5.3.1}$$

$$\text{s.t.} \;\; c_i(x_i) = 0, (\lambda_i) \tag{5.3.2}$$

$$A_i x_i + B_i q^k = 0, \; (y_i). \tag{5.3.3}$$

The system of first-order optimality conditions for subproblem (5.3) are as follows:

$$\begin{aligned} &\nabla_{x_i} f(x_i) + A_i^T(y + \rho(A_i x_i - q^k)) + \nabla_{x_i} c(x_i)^T \lambda_i = 0 \\ &c_i(x_i) = 0 \\ &A_i x_i + B_i q^k = 0 \end{aligned} \tag{5.4}$$

The application of Newton's method linearizes the nonlinear system and solves the problem:

$$\begin{bmatrix} \nabla^2_{x_i} \mathcal{L}_i^k + \rho A_i^T A_i & (J_i^k)^T & A_i^T \\ J_i^k & & \\ A_i & & \end{bmatrix} \begin{bmatrix} \Delta q_i^k \\ \Delta \lambda_i^k \\ \Delta y_i^k \end{bmatrix} = - \begin{bmatrix} \nabla_{x_i} \mathcal{L}_{\rho_i}^k \\ c_i(x_i^k) \\ A_i x_i^k + B_i q^k \end{bmatrix} \tag{5.5}$$

Then, with Newton's method given a current iterate $(x_i^k, \lambda_i^k, y_i^k)$, one obtains the next iterate $(x_i^{k+1}, \lambda_i^{k+1}, y_i^{k+1})$ with the solution of the linear system of equations:

$$\begin{bmatrix} K_{\rho_i} & E_i^T \\ E_i^T & \end{bmatrix} \begin{bmatrix} \Delta w_i^k \\ \Delta y_i^k \end{bmatrix} = - \begin{bmatrix} r_i^k \\ A_i x_i^k + B_i q^k \end{bmatrix} \tag{5.6}$$

where $w_i = \begin{bmatrix} x_i^k & \lambda_i^k \end{bmatrix}^T$ is the vector of the subproblem variables,

$$K_{\rho_i} = \begin{bmatrix} \nabla^2_{x_i} \mathcal{L}^k + \rho A_i^T A_i & \nabla_{x_i} c(x_i^k)^T \\ \nabla_{x_i} c(x_i^k) & \end{bmatrix}$$

is the KKT system of the subproblem, $r_i^k = \begin{bmatrix} \nabla_{x_i} \mathcal{L}_\rho^k & c_i(x_i^k) \end{bmatrix}^T$ is the right-hand side for the KKT system, of the subproblem, and $E_i = \begin{bmatrix} A_i & 0 \end{bmatrix}^T$ is a compression matrix that extracts the primal coupling variables from the vector of subproblem variables w_i^k. If K_{ρ_i} is invertible one can then solve system (5.6) explicitly. We first write (5.6) as

$$K_{\rho_i} \Delta w_i^k + E_i^T \Delta y_i^k = -r_i^k \tag{5.7.1}$$

$$E_i \Delta w_i^k = -(A_i x_i^k + B_i q^k) \tag{5.7.2}$$

Solving for Δw_i^k in (5.7.1) and substituting in (5.7.2) gives the second-order step formula:

$$\Delta y_i^k = [E_i^T K_{\rho_i}^{-1} E_i]^{-1} [A_i x_i^k + B_i q^k - E_i^T K_{\rho_i}^{-1} r_i^k] \tag{5.8}$$

Equation (5.8) provides then a second-order step for updating the multiplier estimates in ADMM.

$$y^{k+1} = y^k + \alpha \Delta y^k \tag{5.9}$$

Note from (5.8) that at a stationary point, $r_i = 0$ and $\nabla_{y_i} \phi(x^k, q^k) = A_i x_i^k + B_i q^k = 0$. Consequently at stationary points (5.8) converges to zero. Note also that (5.8) is in the form of (5.2) and that $\nabla^2 \phi(x^{k+1}, q^{k+1} y^k) = E_i^T K_{\rho_i}^{-1} E_i$ which implies that the Schur-complement of the subproblem is the Hessian of the dual function.

It can be shown that (5.8) is equivalent to the Schur-complement update of the linking constraint multipliers of a parallel SQP algorithm. The Schur-complement decomposition in Chapter 2 achieves superlinear convergence by applying second-order updates on the y and q variables at a QP level. The second-order update on q is known to add overhead and increase communication in the decomposition algorithm. We explore using only second-order updates on y externally at a problem level with MM and ADMM. We note that Equation (5.8) is equivalent as well to the Newton update proposed by Bertsekas [1982]. Differently than Bertsekas formula, (5.8) considers a partial elimination of constraints and eliminates only the

linear constraints corresponding to the linking across blocks. In his work, Bertsekas [1982] demonstrated three properties of his second-order update formula: 1) the threshold level $\rho \geq \rho^*$ for the second-order update is lower than that for the first- order update; 2) The second-order update has a faster rate of convergence than the first order update; and 3) convergence of the second-order update is guaranteed only for a limited region of initial multipliers. These three properties hold for (5.8), and hence it can be expected to be less sensitive to the value of ρ, faster to converge, and more dependent on y^0. Section 6 describes how to exploit these properties within an ADMM scheme and demonstrate with numerical experiments the their algorithmic benefits.

5.3 Benchmark Problems

We study the performance of the second-order update on MM and ADMM on a set of ten stochastic programming problems. The first six problems are convex QPs of the form:

$$\min_{x_i,q} \quad \sum_{i\in\mathcal{P}} p_i(x_i^T D_i x_i + g_i^T x_i + d_i) \tag{5.10.1}$$

$$\text{s.t.} \;\; J_i x_i = b_i, \; (\lambda_i) \qquad i \in \mathcal{P} \tag{5.10.2}$$

$$A_i x_i - q = 0, \; (y_i) \quad i \in \mathcal{P}. \tag{5.10.3}$$

Here the data defining the problem is given by the coefficients D_i, g_i, d_i, J_i ; the right-hand side coefficient b_i ; and the probabilities p_i . We consider stochastic variants of problems obtained from [Biegler, 2010] and [Kang et al., 2014], and the CUTEr and QPLIB libraries [Gould et al., 2015, Maros and Mszros, 1999a][2]. We generate a stochastic defining Q_i and b_i as a random vector. The normally distributed random Q_i and b_i were generated using the deterministic Q and b as mean. For all problems the scenario probability p_i was set to $1/|P|$. All problems considered a set of 20 scenarios ($|P| = 20$). Statistics for the set of QPs are summa-

[2]The QPs from QPLIB were convexified as $D + \delta I$

Table 5.1.: Quadratic benchmark problems for numerical experiments with MM and ADMM second-order multiplier updates

Name	Source	n	n_q	m
SimpleQP	Biegler [2010]	61	1	6
KangQP	Kang et al. [2014]	21050	50	21000
CVXQP3	Gould et al. [2015]	20006	6	15120
AUGD2	Gould et al. [2015]	4254	14	2200
QPLIB3775	Maros and Mszros [1999a]	3600	8	1360
QPLIB0018	Maros and Mszros [1999a]	1005	5	120

rized in Table 5.1. These QPs provide benchmark problems to compare against the Schur-complement decomposition.

The remaining set of four problems consisted of stochastic non-convex nonlinear DAE constrained optimization problems of the form:

$$\min_{x_i,q} \quad \sum_{i\in\mathcal{P}} p_i \int_{\mathcal{T}} J_i(s_i(t), \upsilon_i(t), u_i(t), \kappa_i) \tag{5.11.1}$$

$$\text{s.t.} \quad F_i(\dot{s}_i(t), s_i(t), \upsilon_i(t), u_i(t), \kappa_i), \ (\lambda_i) \qquad i \in \mathcal{P}, t \in \mathcal{T} \tag{5.11.2}$$

$$A_i u_i(t) - q(t) = 0, \ (y_i) \qquad i \in \mathcal{P}, t \in \mathcal{T}. \tag{5.11.3}$$

where $\mathcal{T} = [t_0, t_f]$ is the time domain, $s_i(t)$, $\upsilon_i(t)$, and $u_i(t)$ the state, algebraic and control variables of the i-th scenario; κ_i the a vector of time invariant parameters; and F_i the differential algebraic system of equations (DAE).

The differential algebraic system is discretized by applying a collocation on finite element scheme as described by Nicholson et al. [2017]. In the scheme considered the time dependent variables are approximated using Lagrange polynomials defined at a set of collocation points $t_j, j \in \mathcal{N}_c := \{0, ..., n_c - 1\}$ on finite elements

$\mathcal{N}_e := \{0, ..., n_e - 1\}$ resulting on a discretized problem with the following NLP form:

$$\min_{x_i, q} \quad \sum_{i \in \mathcal{P}} p_i \sum_{j \in \mathcal{N}_e} \sum_{k \in \mathcal{N}_c} J_i(s_{i,j,k}, \upsilon_{i,j,k}, u_{i,j,k}, \kappa_i) \tag{5.12.1}$$

$$\text{s.t.} \quad F_i(\dot{s}_{i,j,k}, s_{i,j,k}, \upsilon_{i,j,k}, u_{i,j,k}, \kappa_i), \ (\lambda_i) \qquad i \in \mathcal{P}, t \in \mathcal{T} \tag{5.12.2}$$

$$A_i x_i - q = 0, \ (y_i) \qquad i \in \mathcal{P}, t \in \mathcal{T}. \tag{5.12.3}$$

where q_k are the Radau quadrature weights and $x_i = (s_{i,j,k}, \upsilon_{i,j,k}, u_{i,j,k})$ is the composition vector containing the subproblem variables. The A_i matrix is constructed to map the control variables $u_{i,j,k}$ in each subproblem to the complicating variables q. For this second set of problems we considered variants of DAE problems available as examples in the `Pyomo.dae` package. The problems are labeled according with the name given to the pyomo.dae example. As with the set of QPs, we generate the stochastic version of the problems with random vectors κ_i). For additional description of the optimization models see [Rodriguez et al., 2018, Nicholson et al., 2017]

Table 5.2.: Stochastic optimal control benchmark problems for numerical experiments with MM and ADMM second-order multiplier updates

Name	Source	n	n_q	m
OptimalControl	`Pyomo`	6493	32	6460
RxnKinetics	`Pyomo`	16161	1	16160
Distillation	`Pyomo`	62011	30	61980
Semibatch	`Pyomo`	56524	40	56520

5.4 Results and Discussion

The optimization problems of Section 5.3 were implemented using the open-source modeling framework `Pyomo`. `Pyomo` facilitated the discretization of the DAE problems as well as the derivative information necessary for the second-order update. For the discretization of DAE problems we used `Pyomo.dae` and

for first and second- order derivatives we used the newly introduced package `pyomo.contrib.pynumero`. The solution of the optimization problems was obtained using the state-of the art solver `Ipopt`. For comparison and validation purposes the stochastic NLPs were solved following three different approaches. In the first approach, the original extensive- form was solved directly with `Ipopt`. The results from this approach provided validation for the other approaches. The second approach solved the problems with the method of multipliers. Both, first-order and second-order update formulas were used. In the third approach the problems were decomposed and solved with ADMM. Again, both the first-order and second-order update formulas were used and compared. In all three approaches `Ipopt` was used for solving the corresponding subproblems. We thus highlight that all solutions obtained were local minimizers.

To study the performance of the first and second-order updates the optimization problems were solved with MM and ADMM. The convergence criteria to stop the MM and ADMM iterations are given by

$$\rho \|Ax^k + Bq^k\| \leq \epsilon_1 \tag{5.13}$$

$$\rho A^T B(q^{k+1} - q^k) \leq \epsilon_2 \tag{5.14}$$

as described in Boyd et al. [2011] and Rodriguez et al. [2018]. We used $\epsilon_1, \epsilon_2 = 1.0 \times 10^{-6}$ for all problem instances. In Table 5.3 we compare the performance of the method of multipliers with first and second-order updates. To this end, we compare the number of iterations needed to satisfy the convergence criteria in (5.13) and (5.14). To overcome the sensitivity of the second-order update on y^0 we perform first-order updates initially if necessary. The switching criteria from first to second-order updates was based on (5.14). If $\rho A^T B(q^{k+1} - q^k) \leq 1.0 \times 10^{-4}$ then the second-order was used. As we can see for all problem instances MM with a second-order update required significantly fewer iterations. Moreover, for all QPs the second-order update converged in 2-3 iterations resembling the performance of full-space second-order methods on convex quadratic problems.

Table 5.3.: Number of iterations MM with first and second-order multiplier updates

Name	Penalty Parameter ρ	First-Order	Second-Order
SimpleQP	1×10^1	32	2
KangQP	1×10^2	318	2
CVXQP3	1×10^5	531	2
AUGD2	1×10^1	49	3
QPLIB3775	1×10^3	62	3
QPLIB0018	1×10^3	126	2
OptimalControl	1×10^2	84	6
RxnKinetics	1×10^0	8	3
Distillation	1×10^1	36	5
Semibatch	1×10^1	18	4

Table 5.4 summarizes the results for the two multiplier updates on ADMM. For all problems the complicating variables estimates q^0 were initialized following the aggregation initialization strategy of the Progressive Hedging algorithm [Rockafellar and Wets, 1991]. As we can see, the second-order update again reduces significantly the number of iterations for the QPs. However, for the general nonlinear DAE problems the second-order update seems to have little to no effect. The benefits seen in the quadratic problems are attributed to fast convergence of q^k. When ADMM approaches the solution variables q^*, the second- order update accelerates the convergence of the multiplier estimates y^k and the primal infeasibility $\|y^{k+1} - y^k\|$ goes quickly to zero. Differently than MM, ADMM uses a first-order update for the q variables resulting in the better performance observed in Table 5.3. This suggests potential benefits of modifying the q-update on ADMM to also follow a second-order approach. This also indicates that the formation of the Schur-complement within an SQP framework may be avoided by solving the KKT system with ADMM and the Newton update proposed here.

An important drawback of ADMM of using within an SQP framework is the need to tune properly the value of ρ. Figure (5.1) shows the sensitivity of ADMM with respect to ρ when solving a randomly generated convex QP with the first

Table 5.4.: Number of ADMM iterations with first and second-order multipliers updates

Name	Penalty Parameter ρ	First-Order	Second-Order
SimpleQP	1×10^1	32	2
KangQP	1×10^2	318	2
CVXQP3	1×10^5	531	2
AUGD2	1×10^1	49	32
QPLIB3775	1×10^3	62	28
QPLIB0018	1×10^3	126	24
OptimalControl	1×10^2	86	86
RxnKinetics	1×10^0	66	63
Distillation	1×10^1	49	48
Semibatch	1×10^1	15	13

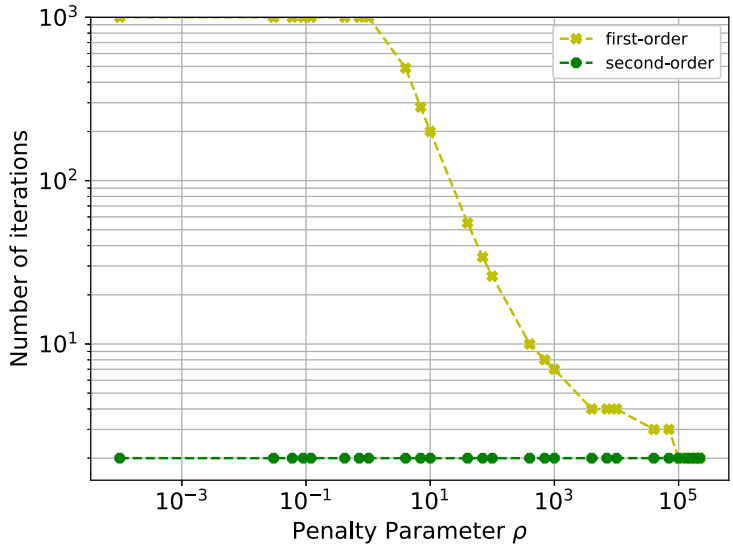


Figure 5.1.: Sensitivity of ADMM to ρ with first and second-order multiplier updates.

and second-order multiplier updates. The stochastic QP was constructed following Algorithm 14 from [Zhang and White, 2018] with n_x=4080, n_λ=3800, n_q=80. To investigate the effect of the penalty parameter we varied ρ from 10^{-3} to 10^5. The problem was initialized following the aggregation initialization strategy of PH and solved with the first and second-order update strategies. We see that differently to

the regular ADMM, our modified ADMM with second-order updates is remarkably robust to solve the stochastic convex QP. While regular ADMM with first-order updates fails to converge after 1000 iterations for small values of ρ, ADMM with the second-order update always converges. This superior behavior of the second-order update is attributed to the fast consensus of q across scenarios of the quadratic problem. For all our numerical experiments we observed that convex stochastic QPs find consensus of the coupling variables in few iterations if a PH initialization strategy is followed. This is important because the second-order update becomes effective once $\|q^k - q^*\| \leq \epsilon$ and is for this reason that we see that almost regarless of the choice of ρ ADMM with second-order updates converges in 2 to 3 iterations.

5.5 Summary

In this chapter we have presented a second-order multiplier update for ADMM. This update strategy follows a Newton extrapolation to accelerate convergence of multiplier estimates. We have demonstrated that the Newton update dramatically reduces the number of subproblem solves for convex QPs. These results are evidence that the Newton update of multipliers resembles that of the Schur-complement decomposition. One interesting characteristic of the second-order update is that makes ADMM more robust to the selection of the penalty parameter. While regular ADMM fails to converge for small values of ρ on convex QPs, the modified ADMM with second-order multiplier updates consistently converged in few iterations.

This chapter concludes the first part of the thesis. In the next chapter we discuss software tools for developing decomposition algorithms. We concentrate in particular on the description of a Python framework that we develop to facilitate research on decomposition algorithms.

Software Tools for Structured Optimization

6. PYTHON NUMERICAL OPTIMIZATION

In this chapter we describe `PyNumero`, an open-source, object-oriented programming framework in Python for prototyping nonlinear optimization algorithms. When combined with `Pyomo`, a Python-based package for optimization modeling, `PyNumero` allows optimization models and solution algorithms to be implemented in the same software platform enabling more rapid development of novel solution algorithms; especially those tailored for specific model structures. `PyNumero` is built on `NumPy` and `Scipy`, well-established Python packages for scientific computing. These packages provide a Python-based API to users while keeping expensive numerical calculations in more efficient, compiled languages like C and Fortran. `PyNumero` also provides Python interfaces to efficient numerical libraries for automatic differentiation and linear algebra including the AMPL Solver Library (ASL), the Harwell Subroutine Library (HSL), and the Parallel Sparse Direct Solver (MUMPS). Finally, `PyNumero` provides infrastructure for parallel computing using the Message Passing Interface (MPI). Solution algorithms prototyped using `PyNumero` can be used to solve optimization models formulated in either `Pyomo` or AMPL. `PyNumero` is designed to avoid marshalling data between the C and Python environments enabling complex algorithm development using high-level Python syntax without a significant sacrifice in performance. This manuscript presents the novel aspects of the implementation and discusses examples of nonlinear optimization algorithms implemented in `PyNumero`. Of special interest are decomposition-based numerical algorithms for large-scale, structured optimization problems. The efficiency of `PyNumero` is illustrated on problems arising in stochastic programming and optimal control, and timing results are presented for both serial and parallel algorithm implementations. Our computational

studies demonstrate that `PyNumero` strikes a balance between the computational efficiency of compiled code and the rapid scripting syntax of Python and enables intuitive yet efficient implementations of optimization algorithms.

6.1 Preliminaries

Software for numerical optimization has historically been broken into two distinct pieces, a platform for modeling optimization problems using high-level syntax, often called an algebraic modeling language (AML), and a platform for solving optimization problems with certain features (linear, nonlinear, mixed-integer, etc.) usually implemented in low-level, compiled programming languages for computational efficiency. This paradigm of separating the modeling language from the solver made sense when the research focus was on creating general-purpose optimization algorithms capable of solving a large variety of problems in some standard form. However, with the increased focus on solving larger and larger problems, there is growing interest in developing more tailored solution algorithms that exploit known model structures often obscured by the current model-solver separation paradigm. While there have been efforts to extend existing tools to address this use-case and pass information on the model structure from the modeling layer to the solver [Chiang et al., 2014, Zavala et al., 2008], these extensions usually rely on infrastructure that was designed for passing simple solver options (e.g. AMPL suffixes) and do not support more complex data structures. Furthermore, general-purpose optimization solvers are typically implemented using low-level programming languages like C++ or Fortran and can be difficult to modify or extend. Significant software engineering expertise is required to prototype even minor modifications to existing solvers. This makes it difficult for optimization practitioners to prototype new solution algorithms and explore new ideas.

To address and mitigate these challenges, we present `PyNumero`, a Python package for numerical optimization that provides a high-level programming framework for rapidly developing nonlinear optimization algorithms. `PyNumero` can be

used alongside Pyomo to provide a unified Python platform for both modeling and solving optimization problems. In addition, PyNumero has been designed with computational performance in mind and utilizes Python-C interfaces internally to ensure that expensive numeric calculations are all performed using compiled code. PyNumero is not intended to replace existing state-of-the-art nonlinear optimization solvers such as Knitro, Ipopt, or Worhp [Byrd et al., 2006a, Wächter and Biegler, 2006, Bskens and Wassel, 2013]. Instead, it supports a variety of tools that allow researchers to explore new optimization methods, try different algorithmic heuristics, and evaluate different linear algebra routines and data structures without requiring substantial rewriting of code. The intent is to enable more practitioners and researchers in nonlinear optimization to write numerical algorithms and rapidly implement new ideas.

Real-world, large-scale optimization problems typically require specialized solution approaches that exploit prior knowledge of the application [Biegler, 2017, Zavala et al., 2008, Gondzio and Grothey, 2009, Nocedal and Wright, 2006]. Decomposition-based algorithms are one such approach for solving these large-scale problems. These algorithms exploit structure in optimization problems and can often be parallelized [Benzi et al., 2005]. Examples of optimization problems with decomposable structures are ubiquitous in stochastic programming, network optimization and dynamic systems [Kang et al., 2014, Word et al., 2014]. By combining PyNumero with Pyomo we seek to facilitate the implementation of structure-exploiting optimization algorithms. This unified modeling and solution platform allows PyNumero to directly interrogate Pyomo model structure and makes it easier to implement parallelizable decomposition algorithms.

The remainder of this chapter is organized as follows. In Section 6.2 we discuss a number of projects related to PyNumero and highlight the main contributions of our work. Then, in Section 6.3 we present an overview of the modeling components in PyNumero for developing nonlinear optimization algorithms from Python. In Section 6.4 we present the design features of PyNumero and present ap-

plication examples in Section 6.5. Timing results together with code examples are presented in Section 6.5 to demonstrate the effectiveness of PyNumero for implementing and solving nonlinear optimization problems. We conclude the chapter in Section 6.7 and describe future directions of the project.

6.2 Related Work

Many software packages have been released to address different aspects of nonlinear optimization. As PyNumero includes and expands upon features available in a broad spectrum of existing software tools, in this section we discuss some of these existing tools and how PyNumero compares to them. We have categorized these existing software tools into three broad categories: solvers and solution algorithms, modeling tools, and application-specific frameworks.

Nonlinear optimization solvers are often implemented in low-level programming languages like C, C++, and Fortran. These software packages typically implement one core optimization algorithm and then devote subsequent development efforts to fine-tuning the performance of that algorithm through additional options, algorithm improvements, heuristics, and code refactoring.
LANCELOT [Conn et al., 2013] is a solver for large-scale nonlinear optimization problems and uses an Augmented Lagrangian algorithm implemented in standard Fortran77. MINOS [Murtagh and Saunders, 1987] uses a projected augmented Lagrangian method for solving general nonlinear problems and is also implemented in Fortran. IPOPT [Wächter and Biegler, 2006] is a line-search filter interior-point (IP) method implemented in C/C++. KNITRO [Byrd et al., 2006a] implements a trust-region interior-point penalty-barrier method in C++. Examples of sequential quadratic programming (SQP) codes are CONOPT [Drud, 1985] (written in Fortran), LINDO [Lin and Schrage, 2009] (written in C/C++), and SNOPT [Gill et al., 2005] (written in Fortran). WORHP [Bskens and Wassel, 2013, Kuhlmann and Bskens, 2017] is a mixed SQP and IP method for solving large-scale nonlinear optimization problems and implemented in Fortran. GALAHAD [Gould et al., 2003], a pack-

age written in fortran95, contains a collection of solution algorithms for large-scale nonlinear optimization.

Most of the optimization solvers described above have excellent computational performance but the implementations can be difficult to navigate, use, or extend. To explore new algorithms or extensions that may improve properties like convergence rate, solve time, or edge-case handling there is a need for easy-to-use and easy-to-extend packages to facilitate research on nonlinear optimization algorithms.

Formulating optimization problems and supplying derivative information used to be done in the same low-level language of the solver but was made more accessible to optimization practitioners with the introduction of AMLs. By taking advantage of and mimicking some of the high-level modeling interfaces offered by several AMLs, `PyNumero` is meant to bring the same ease-of-use to algorithm developers. Some examples of AMLs in Python include `PuLP` [Mitchell et al., 2011] and `Pyomo` [Hart et al., 2011b, 2012a]. `PuLP` focuses on linear optimization while `Pyomo` provides functionality to model linear, nonlinear, mixed-integer linear and mixed-integer nonlinear problems among many other classes of optimization. `Pyomo` also includes functionality to formulate and model large-scale optimization problems that arise in stochastic programming and dynamic optimization [Watson et al., 2012, Nicholson et al., 2017]. Like a standard AML, `Pyomo` constructs an algebraic representation of an optimization problem and hands it to a solver to run a numerical algorithm to find a solution. Hence, construction of the optimization formulation is done in Python while the solution is in a low-level language.

`CasADi` [Andersson et al., 2018, Andersson, 2013] is an open-source software for numerical optimization that also offers functionality for modeling and solving optimization problems. `CasADi` is written in C++ and interfaces with a variety of high-level languages like Python and Matlab. It offers efficient automatic differentiation subroutines and is interfaced with Python via SWIG [Beazley, 1996].

CasADi is an alternative to conventional AMLs. It focuses on optimal control problems and thus offers support for simulation and optimization of dynamic systems. It is self-contained as it creates its own data structures for graph representations.

OSQP [Stellato et al., 2017] is a C++ solver interfaced with Python, Matlab and Julia. OSQP focuses on solving quadratic programs with an algorithm based on the alternating direction method of multipliers. Because it is concerned on solving QPs it does not require specialized algebraic modeling features like those available in Pyomo, PuLP or CasADi. The Python interface of OSQP is similar to PyNumero in that it stores data in sparse matrices from SciPy and vectors in NumPy arrays, making it compatible with features within the NumPy ecosystem.

CVXOPT [Dahl and Vandenberghe, 2007] and NLPy [Arreckx et al., 2016] are complete environments for modeling and solving optimization problems. CVXOPT focuses on solving convex optimization problems while NLPy is for general nonlinear optimization. The goals of NLPy are similar to those of PyNumero in that both packages seek to facilitate research of nonlinear optimization algorithms. However, PyNumero focuses on extending the modeling capabilities in Pyomo to develop structure-exploiting algorithms for stochastic programming and dynamic optimization problems providing tools for both general nonlinear algorithms and decomposition strategies that exploit block-structures in problems. NLPy uses its own interfaces for matrix operations and has its own linear algebra classes. PyNumero is highly integrated with Numpy and Scipy providing broader access to other Python packages (e.g. Petsc4py, PyTrillinos, MPI4py [Dalcin et al., 2011b, Sala et al., 2008, Dalcin et al., 2011a])

Outside of Python there are also software packages that share the same design goals of PyNumero. Recent developments in Julia with the JuMP package [Lubin and Dunning, 2015] aim to provide easy-to-use tools for writing nonlinear optimization algorithms. These tools are available in the Julia Smooth Optimizers package https://juliasmoothoptimizers.github.io/. PyNumero's goal is somewhat different from Julia Smooth Optimizers as it not only offers

tools for general nonlinear optimization, but it provides an object-oriented framework for developing decomposition algorithms for structured optimization. Unlike Julia, Python offers complete support for object-oriented software. `PyNumero` uses object-oriented principles comprehensively, applying them to algorithms and problem formulations that exploit block-structures via polymorphism and inheritance mechanisms. Similar principles have been used in stand-alone, low-level compiled optimization solvers like `PIPS-NLP` and `OOPS` [Chiang et al., 2014]. Our initiative is to extend some of those ideas to Python with `PyNumero`, `Pyomo`, `NumPy`, and `SciPy`.

6.3 Modeling General Nonlinear Programming Problems in PyNumero

This section describes the building blocks in `PyNumero` for modeling optimization problems and writing nonlinear optimization algorithms.

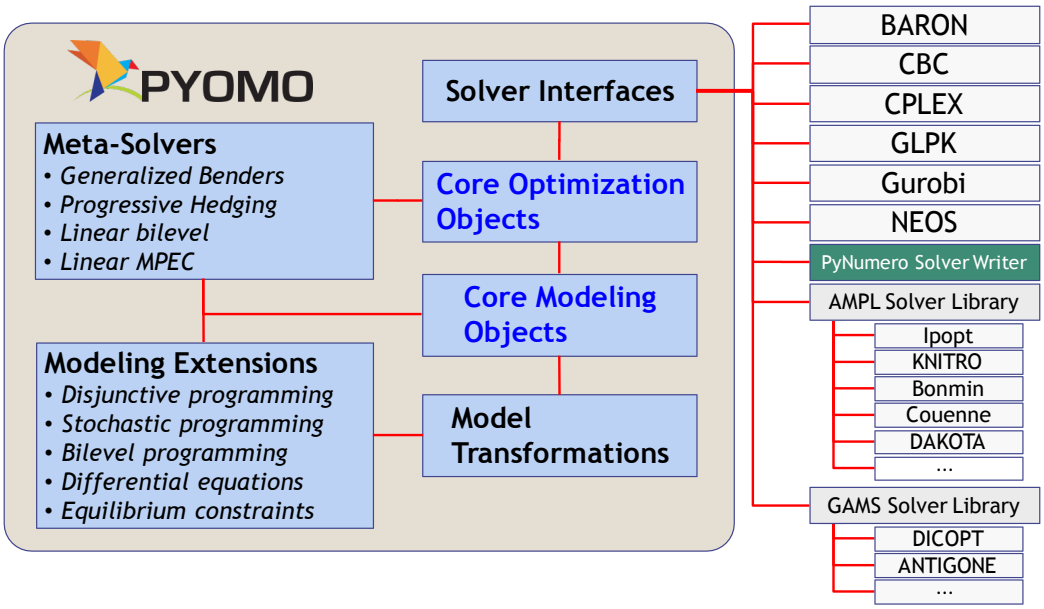


Figure 6.1.: Summary of `Pyomo` features. Modified from [John Siirola, 2017]

`PyNumero` extends the modeling capabilities of the Python package `Pyomo` [Hart et al., 2011b, 2012a]. An overview of the main features of `Pyomo` is shown in

Figure 6.1. This figure also shows how PyNumero fits into the Pyomo eco-system as a new solver interface. Pyomo interfaces to nonlinear solvers using the AMPL Solver Library (ASL). This is a file-based interface that automatically calculates exact first and second derivatives. The optimization model is passed from Pyomo to the ASL using the '.nl' file format. PyNumero also relies on the ASL for calculating derivatives.

Optimization models in Pyomo are represented using Python objects which can be easily analyzed and manipulated. Pyomo also includes a number of modeling extensions for representing high-level model structure. As part of the Pyomo eco-system, PyNumero is fully aware of the structure of Pyomo optimization models. This allows users to write algorithms tailored to a specific problem structure or application.

6.3.1 NLP Interface

PyNumero considers general nonlinear programming problems of the form:

$$\begin{aligned} \min \quad & f(x) \\ & g_L \leq g(x) \leq g_U \\ & x_L \leq x \leq x_U \end{aligned} \tag{6.1}$$

where $x \in \Re^n$ are the primal variables with lower and upper bounds $x_L \in \Re^n$, $x_U \in \Re^n$.The inequality constraints $g : \Re^n \to \Re^m$ are bounded by $g_L \in \Re^m$ and $g_U \in \Re^m$. PyNumero also provides explicit distinction between the equality (defined with $g_L{=}g_U$) and inequality constraints (defined with $g_L{\neq}g_U$) to facilitate the implementation of algorithms that require such distinction,

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & d_L \leq d(x) \leq d_U \\ & x_L \leq x \leq x_U \end{aligned} \tag{6.2}$$

The equality constraints are represented by $c{:}\Re^n \rightarrow \Re^{m_c}$ and $d{:}\Re^n \rightarrow \Re^{m_d}$ denotes the inequality constraints with bounds $d_L \in \Re^{m_d}$ and $d_U \in \Re^{m_d}$ and $m = m_c + m_d$. The `PyomoNLP` class takes a `Pyomo` model and maps it to the general forms defined in (6.1) and (6.2). The listing below shows a small example of how a `PyomoNLP` instance can be used to access different parts of the model. Vectors and matrices are stored in `Numpy` arrays and `Scipy` sparse matrices.

```
from pyomo.contrib.pynumero.interfaces import PyomoNLP
from pyomo.contrib.pynumero.sparse import BlockMatrix
import pyomo.environ as aml

# define optimization model
model = aml.ConcreteModel()
model.x = aml.Var([1, 2, 3], bounds=(0.0, None), initialize=4.0)
model.c = aml.Constraint(expr=model.x[3] ** 2 + model.x[1] == 25)
model.d = aml.Constraint(expr=model.x[2] ** 2 + model.x[1] <= 18.0)
model.o = aml.Objective(expr=model.x[1] ** 4 - \
                         3 * model.x[1]*model.x[2] ** 3 + \
                         model.x[3] ** 2 - 8.0)

# create NLP
nlp = PyomoNLP(model)
# initial guesses for primal and dual variables
x = nlp.x_init()  # array([4., 4., 4.])
y = nlp.y_init()  # array([0., 0.])
# variable bounds
xl = nlp.xl()
```

```
21 xu = nlp.xu()
22 # NLP function evaluations
23 f = nlp.objective(x)
24 print("Objective Function\n", f)  # -504.0
25 g = nlp.evaluate_g(x)
26 print("Constraints\n", g)  # array([-5., 20.])
27 c = nlp.evaluate_c(x)
28 print("Equality Constraints\n", c)  # array([-5.])
29 d = nlp.evaluate_d(x)
30 print("Inequality Constraints\n", d)  # array([20.])
```

6.3.2 Evaluation of First and Second-Order Derivatives

Gradient-based optimization algorithms have been proven to be among the most efficient algorithms for solving nonlinear optimization problems. The recent development of fast automatic differentiation tools [Andersson, 2013, Griewank et al., 1996, Fourer et al., 1993] makes it easy to compute derivative information efficiently. State-of-the-art optimization algorithms typically use first and second-order derivatives along with numerical linear algebra subroutines to find descent directions that lead towards an optimal solution. `PyNumero` uses the Ampl Solver Library (ASL) to compute derivative information for problems (6.1) and (6.2). The ASL is a collection of C subroutines that compute exact derivatives efficiently. The `Ctypes` Python package is used to call the ASL subroutines from Python. `PyNumero` stores derivative values from the ASL in `Numpy` arrays and `Scipy` sparse matrices. This leverages the capabilities within the `Numpy` ecosystem to avoid marshalling of data between the C and Python environments and enables performant Python implementations of gradient-based nonlinear optimization algorithms. It should be noted that even though the current version of `PyNumero` only supports calculation of derivatives from the ASL, interfacing function evaluations of `Pyomo` models with `Numpy` arrays opens a number of possibilities for

interfacing with other AD tools (e.g. PyAdolC). The following listing shows how first and second derivatives can be obtained from a `PyomoNLP` instance.

```
# NLP first and second-order derivatives
df = nlp.grad_objective(x)
print("Gradient of Objective Function:\n", df)  # array([-576., 8., 64.])
jac_g = nlp.jacobian_g(x)
print("Jacobian of Constraints:\n", \
      jac_g.toarray())  # array([[0., 8., 1.],
                        #        [8., 0., 1.]])
jac_c = nlp.jacobian_c(x)
print("Jacobian of Equality Constraints:\n", \
      jac_c.toarray())  # array([[0., 8., 1.]])
jac_d = nlp.jacobian_d(x)
print("Jacobian of Inequality Constraints:\n", \
      jac_d.toarray())  # array([[8., 0., 1.]])
hess_lag = nlp.hessian_lag(x, y)
print("Hessian of Lagrangian\n", \
      hess_lag.toarray())  # array[[-288.,   0., -144.],
                           #        [   0.,   2.,    0.],
                           #        [-144.,   0.,  192.]]))
```

6.3.3 Sparse Linear algebra and Matrix-Vector Storage

Matrix vector operations are fundamental for the development of any numerical algorithm. `Numpy` is a popular Python package that provides functionality to store and manipulate n-dimensional arrays of data, with most of the operations being performed in compiled code. Over the years, the efficiency and flexibility of `Numpy` has made the package an essential library for most of today's scientific/-mathematical Python-based software. `Scipy` is another popular Python package for scientific computing which builds on `Numpy` to provide a collection of common numerical routines (also pre-compiled) and sparse matrix storage schemes. To exploit the capabilities of the `Numpy/Scipy` ecosystem, `PyNumero` stores vectors

and matrices from the NLP interface in `Numpy` arrays and `Scipy` sparse matrices. By doing so, `PyNumero` benefits from the fast pre-compiled operations of `Numpy` (e.g. vectorization and broadcasting), makes all subroutines in `Numpy/Scipy` available for implementing optimization algorithms, and minimizes the burden on users to learn additional syntax besides what is offered in `Numpy/Scipy`.

6.3.4 Block Linear Algebra

General nonlinear optimization algorithms often deal with block algebra operations. In constrained optimization, for instance, the KKT system, also known as a saddle point matrix, consists of a 2×2 block. Certain classes of problems such as stochastic programming and dynamic optimization impose certain structures on the KKT system. An example of the KKT system of an stochastic optimization problem is presented in Figure 6.2 where the plotted points represent non-zero entries in the matrix. One of the main goals of `PyNumero` is to promote research in decomposition algorithms. To this end, `PyNumero` subclasses the `ndarray` object in `Numpy` and sparse matrix objects from `Scipy` to facilitate block-matrix-vector operations. Being able to represent and manipulate the KKT system using its block submatrices greatly simplifies the implementation of both general optimization algorithms and tailored decomposition algorithms. The following listing shows how a KKT matrix can be constructed using block matrices extracted from the `PyomoNLP` instance.

```
# Block matrices
kkt = BlockMatrix(2, 2)
kkt[0, 0] = hess_lag
kkt[0, 1] = jac_g.transpose()
kkt[1, 0] = jac_g
print("KKT system\n", \
      kkt.toarray())  # array([[-288.,    0., -144.,    0.,    8.],
                      #        [   0.,    2.,    0.,    8.,    0.],
                      #        [-144.,    0.,  192.,    1.,    1.],
```

```
10                          #       [   0.,    8.,    1.,    0.,    0.],
11                          #       [   8.,    0.,    1.,    0.,    0.]]))
```

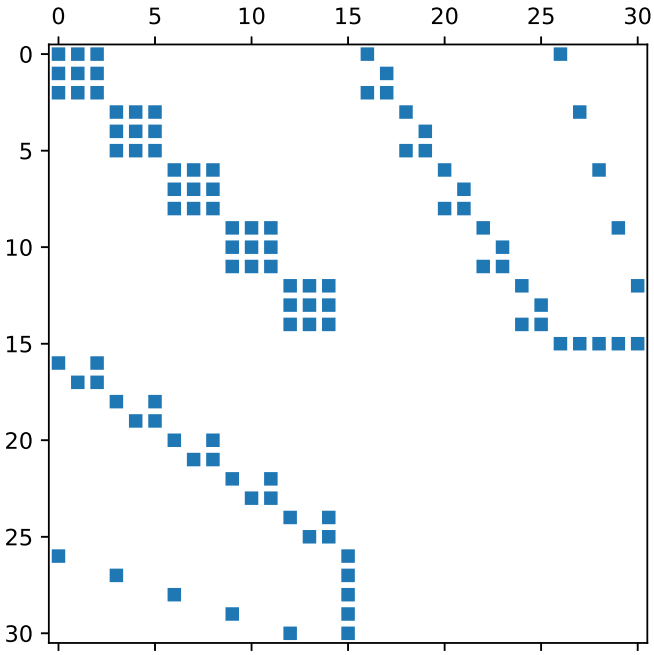


Figure 6.2.: KKT system of an stochastic optimization problem

6.3.5 Numerical Linear Solvers

Efficient implementations of nonlinear optimization algorithms require fast and reliable linear solvers. `PyNumero` provides access to several libraries for the solution of the sparse linear systems that arise in nonlinear programming. Because `PyNumero` stores matrices in `Numpy/Scipy` objects, all subroutines available in the `Numpy` ecosystem can be used when writing algorithms in `PyNumero`. This includes the `Scipy` direct and iterative solvers as well as any other Python package based on `Numpy` such as `PyTrillinos` [Sala et al., 2008], `Petsc4py` [Dalcin et al., 2011b], `Cysparse`, and `Krypy`. `PyNumero` also supports symmetric indefi-

nite linear solvers that provide inertia information. Interfaces to `PyMumps` and the `HSL` linear solvers MA27 and MA57 are available within `PyNumero` to solve sparse linear systems. These latter solvers are particularly important in constrained optimization as they provide critical information like the number of negative eigenvalues.

6.4 Package Implementation and Extensibility

One of the main implementation goals of `PyNumero` is to make it easy for users to write and experiment with nonlinear optimization algorithms. The software design of the package has then focused on interfacing well established scientific computing python subroutines with optimization packages that facilitate modeling of optimization problems. As part of our implementation we have developed a general framework for writing nonlinear optimization algorithms based on the main components presented in Figure 6.3

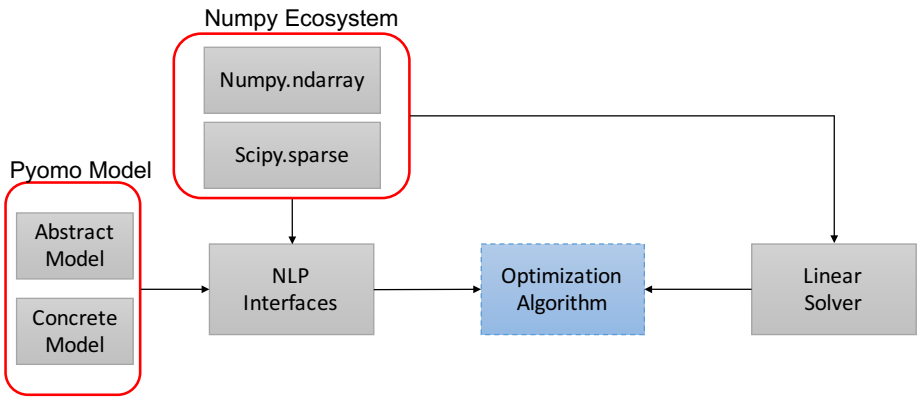


Figure 6.3.: Software design in `PyNumero`

If users would like to create a custom optimization algorithm they will rely on two main components. `PyNumero` defines an abstract class for modeling the general nonlinear programming problem described in Section 6.3.1. This NLP object provides the necessary functionality to query NLP information like bounds on

variables and constraints, or function evaluations and derivatives. The `PyomoNLP` interface described in Section 6.3.1 subclasses from this abstract NLP class and links functionality in `Numpy` and `Scipy` with functionality in ASL allowing users to query NLP information from `Pyomo` models. As well, to write an optimization algorithms users will need a linear solver. Examples of linear solvers in `PyNumero` are interfaces to MUMPS and MA27 and any other linear solver available in the `Numpy` ecosystem. Since both the NLP interfaces and the linear solvers take and return objects from the `Numpy` ecosystem users familiar with `Numpy/Scipy` functionality may easily write their nonlinear optimization algorithms.

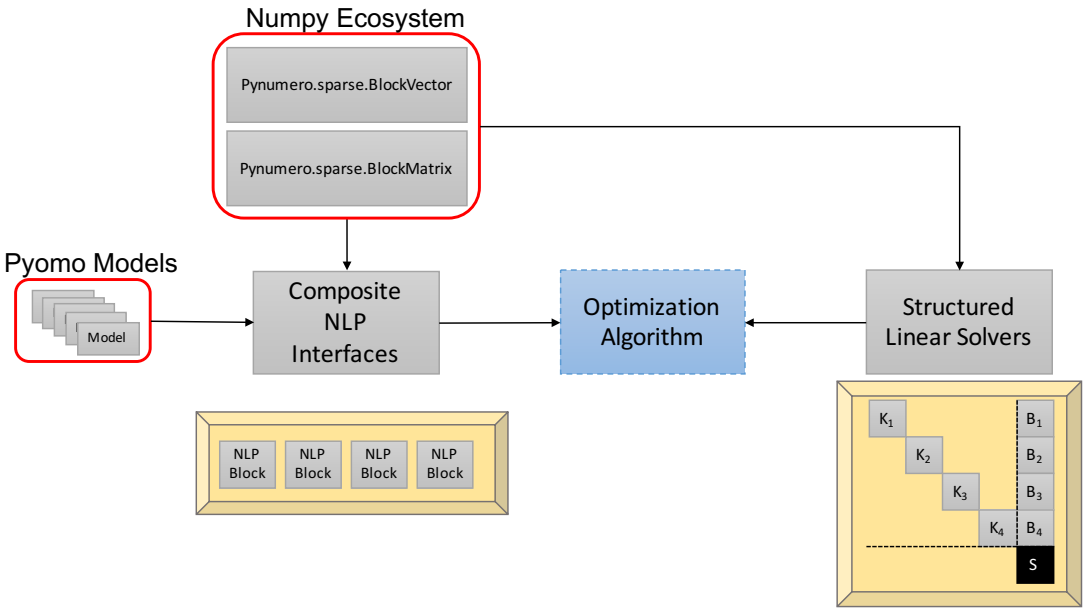


Figure 6.4.: Block structures in `PyNumero`

One other main goal of `PyNumero` is to facilitate research on decomposition algorithms for nonlinear optimization. Block operations are natural candidates for decomposition. We have therefore implemented BlockMatrices and BlockVectors to extend the NLP interfaces in order to exploit decomposable structures at the NLP level and at the linear algebra level. These block algebra classes inherit from `Numpy/Scipy` objects to make them compatible with the design described

in Figure 6.3. However, as we will see in some of the applications presented in the following section, specialized NLP interfaces and linear solvers can exploit the structure and allow for parallel implementations. Figure 6.4 illustrates this idea of interfaces that exploit the structure in optimization problems. The current implementation of `PyNumero` includes interfaces to composite NLP objects for two-stage stochastic NLPs and dynamic optimization NLPs together with Schur-complement-based linear solvers.

6.5 Illustrative Applications

This section provides application examples that illustrate the flexibility and effectiveness of `PyNumero` for writing nonlinear optimization algorithms. We begin with a presentation of a basic implementation of Newton's method. While Newton's method is not necessarily an optimization algorithm, it is certainly the basis for many large-scale optimization algorithms. The example demonstrates the compactness of `PyNumeros` syntax and examines computational performance for solving a sequence of nonlinear system of equations within a Monte-Carlo study. We then discuss the implementation of the interior-point method. The example examines the computational performance of the interior-point implementation on an optimal control problem with thousands of variables and constraints. In the following example we discuss the implementation of the Schur-complement decomposition for solving a quadratic program in parallel. We conclude with a more complex example involving stochastic optimal control of natural gas networks, which is solved with a parallel implementation of the alternating direction method of multipliers. Reported results use `Pyomo` version 5.6.1, Python version 3.6, `Ipopt` version 3.12.3 on a desktop computer running MacOs with processor core i5 and clock speed 2.7GHz. Results for parallel implementations were obtained on a shared-memory machine running Linux OS server with 24 cores, 264 GB of DDR3 RAM and a clock speed of 2.6 GHz. We note that the syntax used in these examples is compatible with more recent `Pyomo` releases including Pyomo 5.6.1. Code for these

examples can be accessed at `https://github.com/Pyomo/pyomo/contrib/pynumero/examples`. Further examples are also found in there.

6.5.1 Newton Solver

Our first example illustrates the compactness and simplicity of PyNumero's syntax for implementing optimization algorithms. Being part of Python and complementing the algebraic modeling features of `Pyomo` for writing numerical algorithms, PyNumero's design has focused on maximizing code readability while minimizing the performance bottlenecks of using Python. In this section we demonstrate these features by presenting the implementation of one of the most popular algorithms in optimization.

Algorithm 11 presents the pseudo-code of Newton's method. Beside the algorithm we present the actual implementation in `PyNumero`. Because `PyNumero` provides fast automatic differentiation capabilities to `Pyomo` models, and since all data is stored in `Numpy` arrays, users can write simple algorithms like Newton's in very few lines of code with standard tools within the `Numpy` ecosystem. With this example we highlight that only a basic understanding of `Pyomo` and `Numpy` is needed to translate pseudo-code into actual implementation.

To evaluate the performance of our Newton's algorithm implementation we solve a sequence of models within a Monte-Carlo study in order to determine the probability design space of a pharmaceutical processes. For the study we considered the kinetic model of the semibatch reactor described in [Laky et al., 2019]. The model was implemented in `Pyomo` and discretized with tools available in the automatic discretization module `Pyomo.DAE`. After discretization, the resulting nonlinear systems with 3609 equations were solved with two approaches. The first approach handed the `Pyomo` model to the state-of-the-art solver `Ipopt` in every Monte-Carlo iteration. The second approach solved the models directly in Python with our implementation of Newton's solver avoiding any interfacing step to an external solver.

In their study, Laky et. al [Laky et al., 2019] concluded that in order for the Monte-Carlo approach to accurately determine the probabilistic design space of pharmaceutical processes a significant number of Monte-Carlo simulations must be run. Figure 6.5 shows the timing results obtained for the Monte-Carlo study. We observe that despite our Newton solver is implemented in Python it can simulate many more Monte-carlo runs than when we utilize the standard `Pyomo` interface. In fact, the Newton implementation with `PyNumero` can achieve a speedup factor of 7 compared to the `pyomo-ipopt` solver. The `PyNumero` implementation executes the numerical algorithm in compiled code and is able to avoid repeated writing of intermediate files every solve like is done with the `pyomo-ipopt` interface.

```python
from pyomo.contrib.pynumero.interfaces import PyomoNLP
from scipy.sparse.linalg import spsolve
import numpy as np

# build nlp
nlp = PyomoNLP(model)
# 1. Initialize algorithm
x = nlp.x_init(); max_iter = 1000; tol = 1e-6
for i in range(max_iter):
    F = nlp.evaluate_c(x)
    # 2. Check convergence
    if np.linalg.norm(F, ord=np.inf) < tol:
        break
    # 3. Compute step direction
    DF = nlp.jacobian_c(x)
    dx = -spsolve(DF, F)
    # 4. Compute step size
    alpha = line_search(x, dx)
    # 5. Update variables
    x += alpha * dx
```

Algorithm 11: Newton's Method

1. Initialize the algorithm
Given system $F(x)$; initial guess x^0; tolerance $\epsilon > 0$
Set the iteration index $k \leftarrow 0$

2. Check convergence
if $F(x^k) \leq \epsilon$ **then exit, solution found.**

3. Compute step direction
$\nabla F(x^k) dx^k = -F(x^k)$

4. Compute step size
$\alpha^k = \text{line\_search}(x^k, dx^k)$

5. Update variables
$x^{k+1} = x^k + \alpha^k dx^k$

6. Update iteration index
Set the iteration index $k \leftarrow k + 1$
Return to step **2**

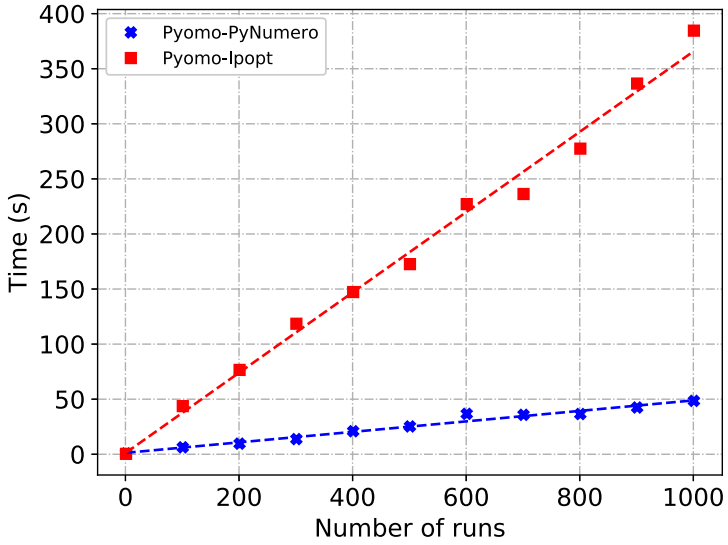


Figure 6.5.: Comparison of computational performance of Monte-Carlo runs.

6.5.2 Interior-point Solver

This section studies the computational performance of a basic implementation of the filter line-search interior-point algorithm. The interior-point method summarized in Algorithm1 was coded in `PyNumero` in approximately two weeks. This time only considers actual implementation time since the authors were familiar with the algorithm. In addition, the authors have extensive expertise in Python programming. The goal of the exercise was to evaluate the flexibility of `PyNumero` for writing an efficient nonlinear optimization solver. The implementation consisted on 1395 lines of Python code, it did not include any clever initialization for multiplier variables, scaling subroutines for primal and dual variables or any feasibility restoration subroutines. Notwithstanding, when tested, the interior-point implementation solved over 250 test problems from the CuterTest suite [Gould et al., 2015]. The subset of problems is available in `https://github.com/Pyomo/pyomo-model-libraries/tree/master/cute`. In addition, instances of the

recently added quadratic problems from [Maros and Mszros, 1999b] available in [Stellato et al., 2017] were also tested.

Table 6.1 presents statistics on some of the problems solved. Statistics on the complete set of problems tested are presented in the supplemental material. We highlight that the timing results are achievable because of the design of `PyNumero` where all expensive operations are executed in compiled code. In the interior-point iteration the most expensive operations are the solution of the augmented system and the calculation of derivatives. Our implementation executes these two steps in compiled code and uses `Ctypes` to interface to efficient linear solvers and AD packages.

To further evaluate the computational performance of our interior-point implementation, we solved a complex nonlinear optimization problem arising from dynamic optimization of a distillation column under uncertainty [Kang et al., 2014]. The problem consists of 32 differential equations, 1 control, 35 algebraic variables, and a set of scenarios that account for uncertainty in the composition of the feed stream. The differential equations were discretized with the orthogonal collocation method [Biegler, 2010] and the optimization formulation is summarized in problem (6.3)

Table 6.1.: Results on a subset of problems from the CuterTest collection.

Name	Iterations	n	m	Objective	Optimality Error	Time(s)	Status
aug3d	2	3873	1000	5.5406773e+02	8.39e-10	0.214	Opt
aug3dcqp	28	3873	4873	9.9336215e+02	3.76e-09	0.554	Opt
blockqp1	31	2005	1001	-9.9650000e+02	2.73e-09	0.367	Opt
cbratu2d	2	882	882	0.0000000e+00	1.31e-12	0.057	Opt
clnlbeam	588	1499	1000	3.4487622e+02	1.00e-09	5.751	Opt
cont-050	38	2597	4998	-4.5638509e+00	3.94e-09	1.097	Opt
cvxqp1m	29	1000	1500	1.0875116e+06	2.14e-09	1.017	Opt
dtoc3	2	14999	10000	2.3526248e+02	3.35e-15	0.111	Opt
dtoc1l	6	14985	9990	1.2533813e+02	3.61e-11	0.297	Opt
dixchlng	10	10	5	2.4718978e+03	7.19e-14	0.067	Opt
expfitc	93	5	502	2.3302576e-02	1.02e-09	0.841	Opt
expquad	52	120	10	-3.6245999e+06	2.51e-09	0.398	Opt
ksip	37	20	1001	5.7579793e-01	5.53e-09	0.488	Opt
gouldqp3	399	699	349	2.0651557e+00	1.25e-09	7.493	Opt
hs099	10	23	18	-8.3107989e+08	2.51e-09	0.089	Opt
hs116	75	13	28	9.7587510e+01	2.61e-09	0.633	Opt
liswet2	32	10002	10000	2.4999983e+01	1.16e-09	0.867	Opt
mosarqp	24	2500	3200	-9.5287544e+02	1.00e-09	0.598	Opt
primal4	222	1489	76	-7.4609069e-01	2.51e-09	3.700	Opt
primalc8	74	520	511	-1.8309430e+04	3.03e-10	0.791	Opt
qsctap3	41	2480	3960	1.4387547e+03	8.41e-09	0.653	Opt
stadat1	382	2001	3999	-2.8526864e+07	3.73e-09	7.231	Opt
stcqp2	119	4097	6149	2.2327313e+04	6.10e-09	15.455	Opt
ubh1	6	18009	18015	1.1160008e+00	2.51e-09	0.259	Opt
oet1	111	3	1002	5.3824313e-01	2.51e-09	0.917	Opt

$$\min \sum_{k=1}^{N}\sum_{i=1}^{n_f}\sum_{j=1}^{n_c} h_{i,k}\Omega_{j,n_c}\left[\alpha\left(y_{1,i,j,k}-y_{1,i,j,k}^{set}\right)^2+\rho\left(u_{i,j,k}-u_{i,j,k}^{set}\right)^2\right] \tag{6.3}$$

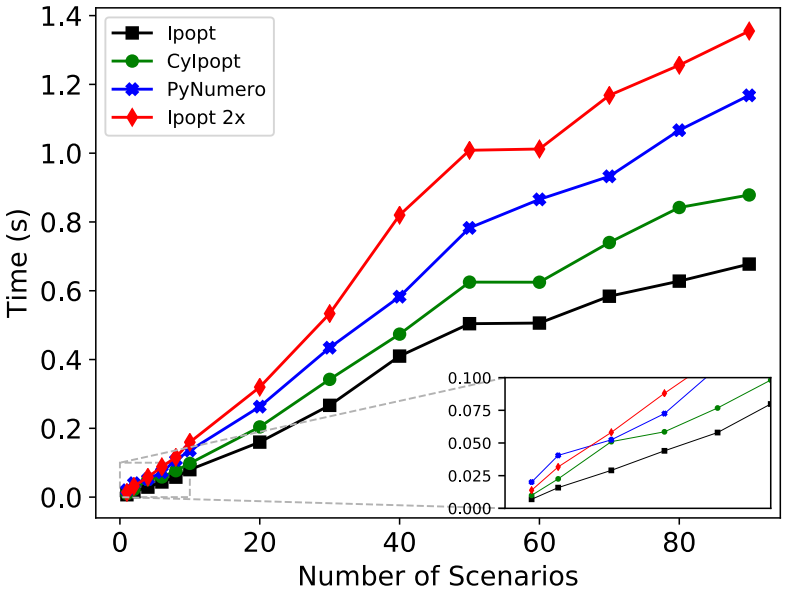
s.t.

$$\left.\begin{aligned}
L1_{i,j,k}-u_{i,j,k}D_{l,k}&=0\\
V_{i,j,k}-L1_{i,j,k}-D_{l,k}&=0\\
L2_{i,j,k}-F-L1_{i,j,k}&=0\\
y_{n,i,j,k}-\frac{x_{n,i,j,k}\alpha_{A,B}}{1+(\alpha_{A,B}-1)x_{n,i,j,k}}&=0,\\
n&=1,...,NT\\
z_{1,i,j,k}-\frac{1}{A_{cond}}V_{i,j,k}(y_{2,i,j,k}-x_{1,i,j,k})&=0\\
z_{n,i,j,k}-\frac{1}{A_{Tray}}[L1_{i,j,k}(x_{n-1,i,j,k}-x_{n,i,j,k})-V_{i,j,k}(y_{n,i,j,k}-y_{n+1,i,j,k})]&=0,\\
n&=2,...,FT-1\\
z_{17,i,j,k}-\frac{1}{A_{Tray}}[Fx_{Feed,k}+L1_{i,j,k}x_{FT-1,i,j,k}&\\
-L2_{i,j,k}x_{FT,i,j,k}-V_{i,j,k}(y_{FT,i,j,k}-y_{FT+1,i,j,k})]&=0\\
z_{n,i,j,k}-\frac{1}{A_{Tray}}[L2_{i,j,k}(x_{n-1_{i,j,k}}-x_{n,i,j,k})-V_{i,j,k}(y_{n,i,j,k}-y_{n+1,i,j,k})]&=0,\\
n&=FT+1,...,NT-1\\
z_{NT,i,j,k}-\frac{1}{A_{Reboiler}}[L2_{i,j,k}x_{NT-1,i,j,k}-(F-D_{l,k})x_{NT,i,j,k}-V_{i,j,k}y_{NT,i,j,k}]&=0\\
x_{n,i,j,k}-x_{n,i-1,n_c,k}-h_{i,k}\sum_{m=1}^{n_c}\Omega_{m,j}z_{n,i,m,k}&=0,\\
n&=1,...,NT\\
x_{n,1,j,k}-x_n^0-h_{i,k}\sum_{m=1}^{n_c}\Omega_{m,j}z_{n,1,m,k}&=0,\\
n&=1,...,NT\\
P_kX_k-u_c&=0\\
x_{n,i,j,k},y_{n,i,j,k},V_{i,j,k},L2_{i,j,k},L1_{i,j,k},D_{l,k}&\geq 0
\end{aligned}\right\}\begin{aligned}&i=1,...,n_f\\&j=1,...,n_c\\&k=1,...,N\end{aligned}$$

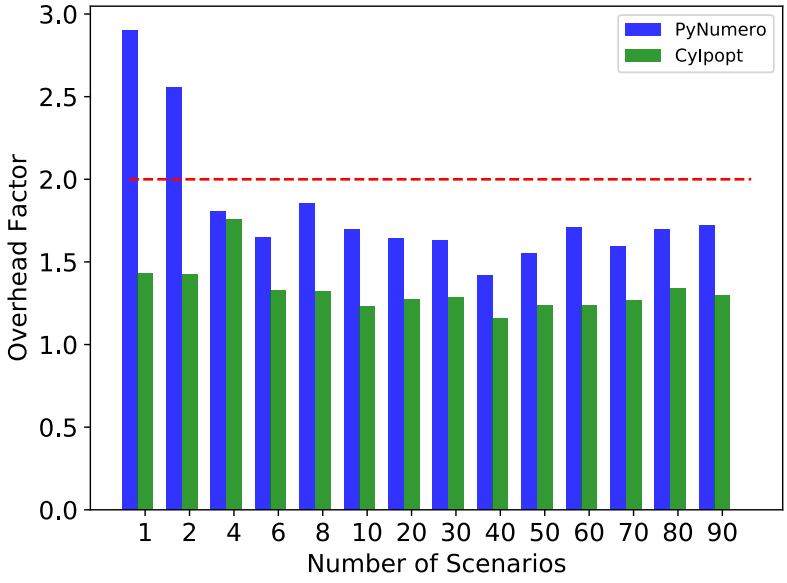
Here, N is the number of scenarios, n_f is the number of finite elements, and n_c is the number of collocation points in each finite element. X_k is the vector of all the variables in each scenario, the matrices P_k are extraction matrices and u_c is the vector of coupling variables across all the scenarios. For further details on the model see [Benallou et al., 1986, Kang et al., 2014, Hahn and Edgar, 2002].

`Pyomo` facilitated the implementation of the optimization problem as well as the discretization of the differential equations. For the discretization we used the orthogonal collocation scheme in `Pyomo.DAE`. To study the computational overhead of running the interior-point method from Python with `PyNumero`, the stochastic problem was solved following three different approaches. In the first approach the problem was solved with `Ipopt`. The timing results of this approach provided a lower bound on the solution time. This is because Ipopt's implementation is 100% in compiled code. The second approach solved the problem with `CyIpopt`. `CyIpopt` [cyipopt Developers, 2017] is a Cython wrapper around `Ipopt` that runs it's interior-point method with Python callbacks for the NLP function evaluations. For this purpose, an interface to `CyIpopt` was implemented in `PyNumero`. Function evaluations and derivative calculations were then directly computed in the NLP interface described in Section 6.3.1. We refer to this second approach as `PyNumero-CyIpopt` as it relies on features from both `PyNumero` and `CyIpopt`. Finally, the third approach solved the optimization problem with the interior-point implementation in `PyNumero`. All three approaches used the linear solver MA27 with a pivot tolerance of 10^{-8} and ASL for computing derivatives.

Figure 6.6 summarizes the timing results obtained. To investigate potential performance overheads caused by the size of the optimization problem the number of scenarios was increased from 1 to 90. The problem size ranged from 2×10^3 to 188×10^3 variables. Figure 6.6(a) shows the average time per iteration for the three approaches. We observe that for all problem sizes `Ipopt` solved the problem the fastest. This is expected because both `PyNumero` and `PyNumero-CyIpopt` incurred in an additional overhead for calling C-functions from Python. We also ob-



(a) Average iteration time. `Ipopt` 2x is the time of `Ipopt` multiplied by a factor of two.



(b) Overhead factors with respect to Ipopt

Figure 6.6.: Timing results for the interior-point implementation.

serve that `PyNumero-CyIpopt` was faster than PyNumero's interior-point imple-

mentation. The reason for this is that `PyNumero-CyIpopt` only has an additional overhead for the NLP function evaluation callbacks. In contrast, PyNumero's interior-point implementation presents additional overhead because it drives the entire algorithm from Python. This includes the outer and inner iteration loops, the line search, and the function evaluations from the ASL interface and the MA27 interface. We highlight that for large problem sizes, PyNumero's iteration was always less than twice that of `Ipopt` (red line). This is more clearly presented in Figure 6.6(b). Figure 6.6(b) shows the iteration time for `PyNumero` and `Cyipopt` divided by the time of `Ipopt`. Note that for the smaller problems with only one and two scenarios the overhead of the Python implementation is greater than a factor of two. The main reason for this behavior is that as the problem size increases more work is performed at compiled code, the solution of the KKT system and the evaluation of derivatives become more relevant while the overhead of the Python callback becomes less significant.

We highlight that the timing results presented in Figure 6.6 were obtained without using any of the just-in-time compilation packages available in CPython. Currently `PyNumero` is only based on CPython, `Ctypes` and `Numpy`. In the future we plan to study potential further performance enhancements by using `Numba` [Lam et al., 2015] and PyPy. Note that because `PyNumero` is part of the `Numpy` ecosystem it can use just-in-time compilation features from both `Numba` and `PyPy`. These just-in-time compiling tools may improve computational performance of python code. For example, the line search subroutines in our basic interior-point implementation were written in python. To improve performance of those subroutines without necessarily using Cython or `Ctypes`, `Numba` and `Pypy` may accelerate the python code significantly.

6.5.3 Schur-complement solver

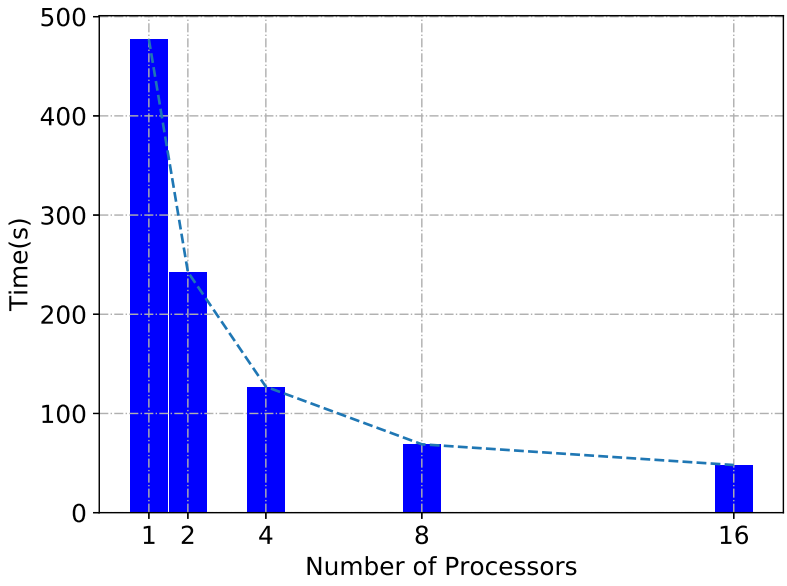
This example studies the parallel implementation of Algorithm 3 to solve the following minimum least-squares parameter estimation problem,

$$
\begin{aligned}
\min_{y,q,d} \quad & \sum_{i \in \mathcal{P}} \frac{1}{2} \| y_i - y_i^\star \|_2^2 \\
\text{s.t.} \quad & y_i - J q_i = 0 \qquad\qquad \forall \ i \in \mathcal{P} \\
& A_i \begin{bmatrix} y_i \\ q_i \end{bmatrix} - d = 0 \qquad \forall \ i \in \mathcal{P}
\end{aligned}
\tag{6.4}
$$

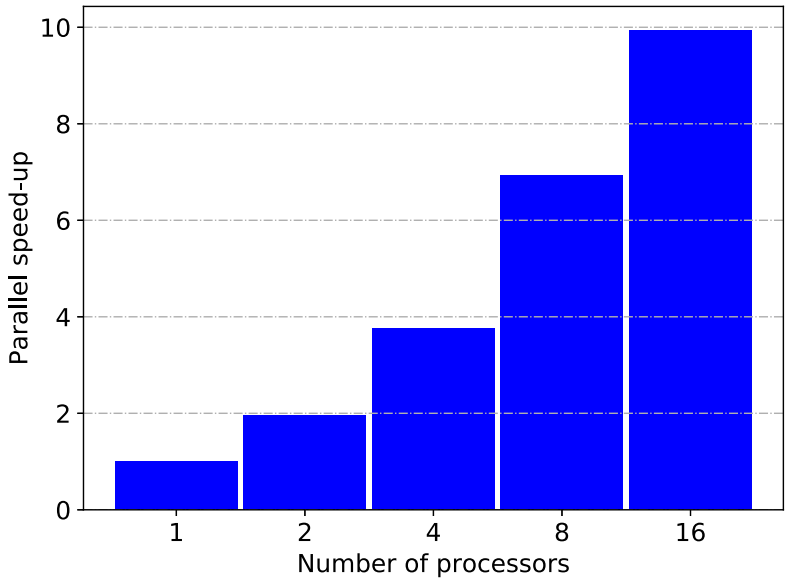
where y_i and q_i are vectors of variables local to each block i. The vector $d \in \mathbb{R}^{n_d}$ is the set of coupling variables that must be the same across all blocks. The vectors $y_i^\star$ correspond to the known measurements for y_i in each block i. J is a randomly generated sparse matrix, and A_i are the linking matrices that map the vector of block variables to the common variables d. For our benchmark problem we used 16 blocks, in each i block the dimension of y_i is $100,000$, the dimension of q_i is $1,000$ and the dimension of d was 100. The random matrix J was generated with `scipy.sparse` with 1% density. Measurement data, $y_l^\star$, was obtained following the procedure proposed in [Kang et al., 2014]. This gives an optimization problem with approximately 1.6 million variables and constraints.

The problem was implemented in `PyNumero` and solved with a parallel implementation of Algorithm 3 that uses `MPI4py`. We used the MA27 interface in `PyNumero` to factorize each block and Numpy's linear solver for dense matrices to factorize the schur-complement. Timing results are summarized in Figure 6.7. We highlight that our python implementation reduces the solution time by one order of magnitude.

Figure 6.8 shows the time required for forming and solving the schur complement matrix as a function of the number of coupling variables. It is clear that for problems with large number of coupling variables it becomes prohibited to form the Schur complement. For this reason researchers have looked for alternatives to step 2 of Algorithm 3 [Kang et al., 2014, Word et al., 2014]. Our intent with `PyNumero` is to facilitate research on Schur-complement-based algorithms. The



(a) Solution times



(b) Speed-up factors

Figure 6.7.: Timing results for parallel Schur-complement Implementation.

next section presents an Augmented Lagrangian solver that combines features of the Schur-complement decomposition and ALM to accelerate it's convergence rate.

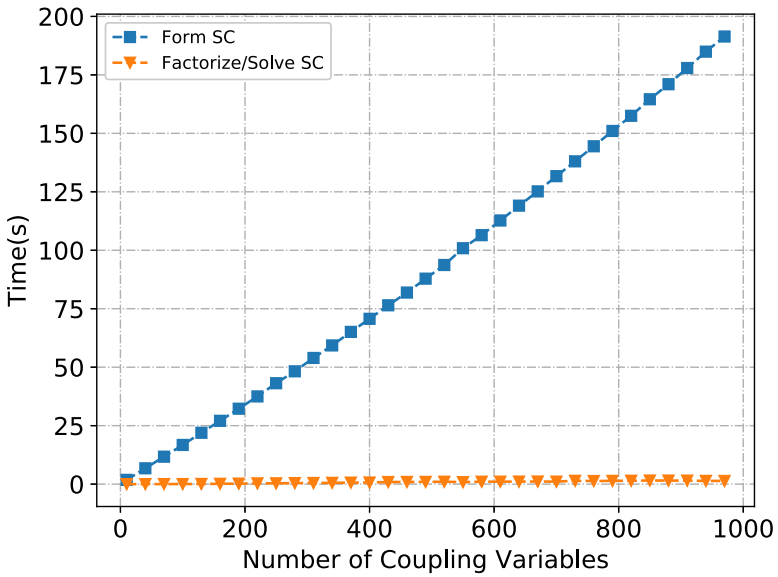


Figure 6.8.: Time required to form and factor the Schur-complement as a function of the number of coupling variables

6.5.4 Stochastic Optimal Control of Natural Gas Networks

In our last application example, we discuss the parallel implementation of the alternating direction method of multipliers for solving a PDE constrained optimization problem under uncertainty. The problem consists on optimizing a natural gas network inventory while accounting for uncertainties in the system demands. Specifically, the objective is to satisfy uncertain gas demands in the network by building up inventory in the pipelines in a way that minimizes the re-

quired compression power. An stochastic formulation for this problem was proposed in [Zavala, 2014], which is as follows:

$$\min \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{S}_n} c_s s_{i,t}\Delta\tau + \sum_{t\in\mathcal{T}}\sum_{\ell\in\mathcal{L}} c_e P_{\ell,t}\Delta\tau + \sum_{t\in\mathcal{T}}\sum_{j\in\mathcal{D}} c_d(d_{j,t}-\bar{d}_{j,t})^2 \tag{6.5.1}$$

$$+ \sum_{k\in\mathcal{X}}\sum_{\ell\in\mathcal{L}} c_T(p_{\ell,T,k}-p_{\ell,0,k})^2 + \sum_{k\in\mathcal{X}}\sum_{\ell\in\mathcal{L}} c_T(f_{\ell,T,k}-f_{\ell,0,k})^2 \tag{6.5.2}$$

$$\text{s.t. } \frac{\partial p_\ell(x,\tau)}{\partial\tau} = -c_{1,\ell}\frac{\partial f_\ell(x,\tau)}{\partial x}, \quad \ell\in\mathcal{L},\ x\in[0,L_\ell],\ \tau\in[0,T] \tag{6.5.3}$$

$$\frac{\partial f_\ell(x,\tau)}{\partial t} = -c_{2,\ell}\frac{\partial p_\ell(x,\tau)}{\partial x} - c_{3,\ell}\frac{f_\ell(x,\tau)|f_\ell(x,\tau)|}{p_\ell(x,\tau)}, \quad \ell\in\mathcal{L},\ x\in[0,L_\ell],\ \tau\in[0,T] \tag{6.5.4}$$

$$p_\ell(L_\ell,\tau) = \theta_{rec(\ell)}(\tau), \ell\in\mathcal{L},\ \tau\in[0,T] \tag{6.5.5}$$

$$p_\ell(0,\tau) = \theta_{snd(\ell)}(\tau),\ \ell\in\mathcal{L}_p,\ \tau\in[0,T] \tag{6.5.6}$$

$$p_\ell(0,\tau) = \theta_{snd(\ell)}(\tau) + \Delta\theta_\ell(\tau),\ \ell\in\mathcal{L}_a,\ \tau\in[0,T] \tag{6.5.7}$$

$$\sum_{\ell\in\mathcal{L}_n^{rec}} f_\ell(L_\ell,\tau) - \sum_{\ell\in\mathcal{L}_n^{snd}} f_\ell(0,\tau) + \sum_{i\in\mathcal{S}_n} s_i(\tau) - \sum_{j\in\mathcal{D}_n} d_j(\tau) = 0,\ n\in\mathcal{N}\ \tau\in[0,T] \tag{6.5.8}$$

$$P_\ell(\tau) = c_p\cdot T\cdot f_\ell(0,\tau)\left(\left(\frac{\theta_{snd(\ell)}(\tau)+\Delta\theta_\ell(\tau)}{\theta_{snd(\ell)}(\tau)}\right)^{\frac{\gamma-1}{\gamma}} - 1\right),\ \ell\in\mathcal{L},\ \tau\in[0,T] \tag{6.5.9}$$

$$\theta_{sup(i),\tau} = \bar{\theta}_i^{sup},\ i\in\mathcal{S},\ \tau\in[0,T] \tag{6.5.10}$$

$$0 = -c_{1,\ell}\frac{\partial f_\ell(x,0)}{\partial x},\ \ell\in\mathcal{L},\ x\in[0,L_\ell] \tag{6.5.11}$$

$$0 = -c_{2,\ell}\frac{\partial p_\ell(x,0)}{\partial x} - c_{3,\ell}\frac{f_\ell(x,0)|f_\ell(x,0)|}{p_\ell(x,0)},\ \ell\in\mathcal{L},\ x\in[0,L_\ell] \tag{6.5.12}$$

$$P_\ell^L \le P_\ell(\tau) \le P_\ell^U,\ \ell\in\mathcal{L}_a,\ \tau\in[0,T] \tag{6.5.13}$$

$$\theta_\ell^{suc,L} \le \theta_{snd(\ell)}(\tau) \le \theta_\ell^{suc,U},\ \ell\in\mathcal{L}_a,\ \tau\in[0,T] \tag{6.5.14}$$

$$\theta_\ell^{dis,L} \le \theta_{snd(\ell)}(\tau) + \Delta\theta_\ell(\tau) \le \theta_\ell^{dis,U},\ \ell\in\mathcal{L}_a,\ \tau\in[0,T] \tag{6.5.15}$$

This model includes detailed network dynamics captured by the PDEs (6.5.3) and (6.5.4). Uncertainty is captured by considering multiple scenarios for natural

gas demand, and the resulting formulation is a two-stage stochastic optimization problem. Because they describe scenario-specific dynamics, model PDEs are replicated for each scenario. The resulting optimization model is very large, scaling as a function of the number of network links, scenarios, and discretization points. All features considered make this problem an excellent example to demonstrate the flexibility of `PyNumero` as an extension of `Pyomo`. A set of sixteen scenarios was considered to account for uncertainty in demand. Each scenario was implemented as a `ConcreteModel` in `Pyomo` and the discretization of the spatial and temporal domain was performed with `Pyomo.DAE` (details available in [Nicholson et al., 2017]). The overall two-stage stochastic program was then constructed using the NLP interfaces available in `PyNumero`. The following listing presents the implementation of the two-stage stochastic program

```
from pyomo.contrib.pynumero.interfaces import (PyomoNLP,
                                               TwoStageStochasticNLP)
from pyomo.contrib.pynumero.algorithms.solvers.admm import AdmmSolver
from pde_model import create_model

n_scenarios = 16
demand_factors = np.random.uniform(0.7, 3.0, n_scenarios)
scenarios = dict()
coupling_vars = dict()
for i in range(n_scenarios):

    # create scenario
    instance = create_model(demand_factor=demand_factors[i])
    nlp = PyomoNLP(instance)

    # define scenarios and first-stage variables
    scenario_name = "s{}".format(i)
    scenarios[scenario_name] = nlp
    coupling_vars[scenario_name] = []

    # list first-stage variables
```

```
22      for k in instance.dp.keys():
23          coupling_vars[scenario_name].append(\
24                                      nlp.variable_idx(instance.dp[k]))
25      for k in instance.dem.keys():
26          coupling_vars[scenario_name].append(\
27                                      nlp.variable_idx(instance.dem[k]))
28
29  # create two-stage stochastic NLP
30  nlp = TwoStageStochasticNLP(scenarios, coupling_vars)
```

The resulting two-stage stochastic program consisted of an optimization problem with approximately 300,000 variables and constraints. Because of the size and block structure of the problem we solve the two-stage stochastic program with the alternating direction method of multipliers. ADMM was implemented in 160 lines of Python code and the package `MPI4Py` was used for parallelization. The ADMM solver called the NLP interface together with the `PyNumero-CyIpopt` interface to solve the corresponding subproblems. Timing results are presented in Figure 6.9.

6.6 Software Distribution

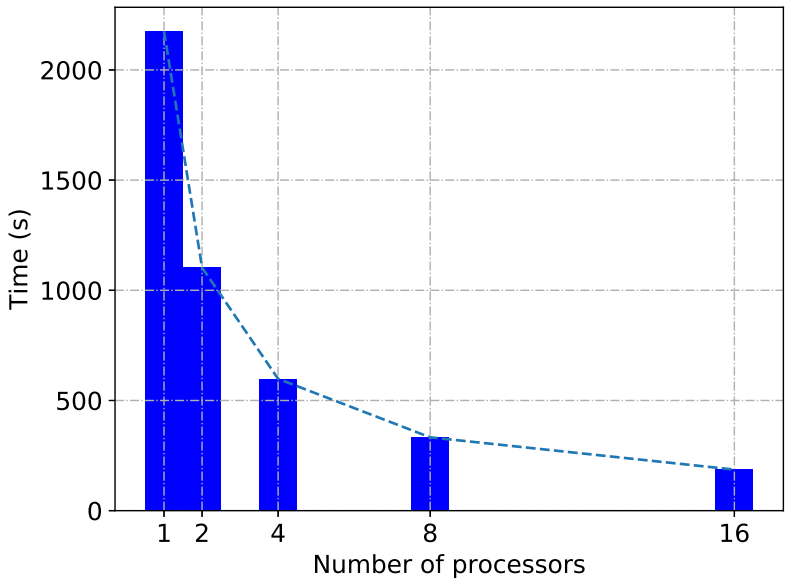
The `PyNumero` package can be obtained with `Pyomo`. All Python files are distributed under the `Pyomo` umbrella available at `https://github.com/Pyomo`. To avoid compilation of the C interfaces the corresponding shared libraries are distributed via conda. Installation of both the Python code and the shared libraries can be done as follows:

- conda install -c conda-forge pyomo

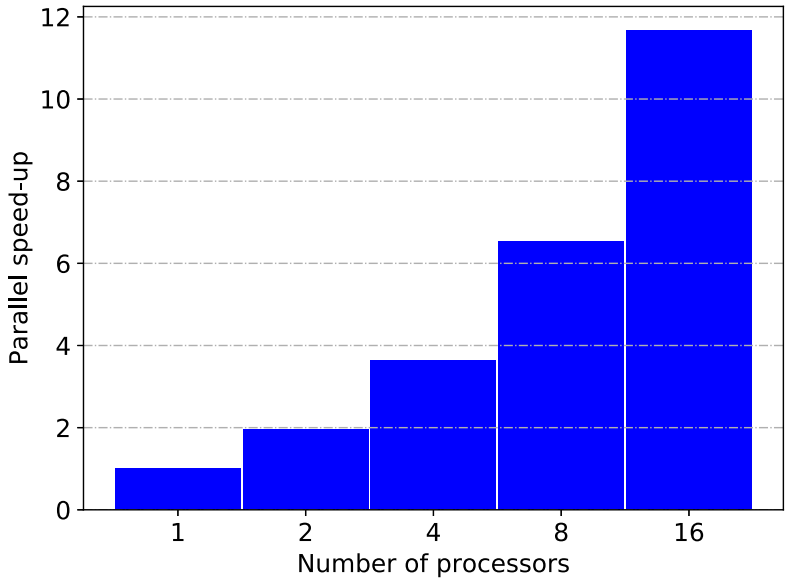
- conda install -c conda-forge pynumero_libraries

To be able to use the `PyNumero-CyIpopt` interface

- conda install -c conda-forge cyipopt



(a) Solution times



(b) Speed-up factors

Figure 6.9.: Timing results for parallel ADMM implementation.

The Interface to MA27 solver is not distributed but is to be made available soon. The project works currently with pyMumps as default linear solver for the Interior-

point code. Currently we offer support for MacOS and Linux operating systems. Windows is not supported yet.

6.7 Summary

We discussed the design and implementation of a Python package called `PyNumero`. The package extends the modeling capabilities of `Pyomo` providing building blocks for writing numerical algorithms for nonlinear optimization from Python. `PyNumero` combines the modeling features of `Pyomo` with efficient libraries like ASL and `Numpy/Scipy`. With this combination, `PyNumero` performs all linear algebra operations in compiled code, and is designed to avoid marshalling of data between the C and Python environments, allowing for high-level development of algorithms without a significant sacrifice in performance. These features are demonstrated with a variety of applications presented in the paper. Timing results together with code code snippets are also included. Among these applications we have shown that an implementation of the interior-point algorithm in `PyNumero` has comparable solution times with the state of the art solver `Ipopt` when solving a dynamic optimization problem with 200K variables and constraints. The overhead from the Python interface to ASL and HSL only increases the solution time by 60% for large-scale instances.

`PyNumero` uses object-oriented principles comprehensively, applying them to algorithms and problem formulations that exploit block-structures via polymorfism and inheritance mechanisms. Since block-structured problems result from real-life optimization problems, we expect the design to promote research of decomposition algorithms. Of special interest are stochastic programming problems and dynamic optimization problems. Current developments in `Pyomo` to model dynamics and uncertainty in optimization problems [Watson et al., 2012, Nicholson et al., 2017] can be combined with features offered in `PyNumero` to prototype and explore new decomposition approaches. Two examples of decomposition al-

gorithms for solving two-stage stochastic programs are presented in the paper and timing results are shown for serial and parallel implementations.

As part of future work we plan to include interfaces to different automatic differentiation packages. Currently, `PyNumero` relies on ASL to compute first and second derivatives. Efficient packages available in Python like `CasADi` and `PyAdolC` seem like excellent extensions to `PyNumero`. Another useful extension of `PyNumero` will involve implementing all block-algebra operations with parallelization capabilities. The current design uses inheritance and polymorfism to represent block vector and block matrices as `Numpy` and `Scipy` objects. We plan to extend these classes to use `MPI4Py` to perform block algebraic operations with parallel computing. Finally, to improve the computational performance of `PyNumero` even further we plan to use just-in-time compiling features from `Numba`.

7. SUMMARY

Large-scale nonlinear optimization problems can arise from a variety of applications and have become remarkably important for design, operation, and planning of engineering systems. We consider particular cases of these optimization problems. The objective of our work is to develop numerical algorithms to solve nonlinear optimization problems that have an inherent decentralized structure. For this reason, in this dissertation we address distributed solution of block-structured NLP problems that arise in stochastic and dynamic optimization. We based our solution strategies on nonlinear interior-point methods and augmented Lagrangian schemes. This chapter first summarizes our contributions and then makes suggestions for directions of future work.

7.1 Thesis Summary and Contributions

Chapter 2 introduces the Schur-complement decomposition for solving structured optimization problems within an interior-point framework. It presents first a description of the optimization problems considered in the dissertation. Examples of general formulations for stochastic programming and dynamic optimization are presented there. Derivation of the Schur-complement matrix for both type of problems are also summarized in Chapter 2. To demonstrate the effectiveness of the Schur decomposition approach a case study in water distribution systems is solved. The problem consisted in a demand estimation with up to 12 million variables and constraints. Speedup factors of up to 10 on a shared memory computer with 16 cores were obtained.

Chapter 3, focuses on augmented Lagrangian based algorithms for decomposing non-convex nonlinear optimization optimization problems. We pay attention

especially to two very popular algorithms; the alternating direction method of multipliers and progressive hedging. Unlike the Schur-complement interior-point algorithm studied in Chapter 2, these approaches are not guaranteed to converge in a non-convex setting. In practice, however, ADMM and PH often perform satisfactorily in complex non-convex NLPs. Moreover, PH has been actively used in the power systems community as a heuristic to solve MILPs and MINLPs. To explain why ADMM and PH are capable of solving such large-scale non-convex NLPs we exploited connections between ADMM and PH with their common ancestor MM and derived a common set of benchmarking metrics to monitor convergence. With these benchmarking metrics we showed in particular that ADMM and PH are inexact versions of MM that approximate its performance when multiple coordination steps are performed. We demonstrated the concepts using challenging non-convex problems arising in dynamic optimization.

We highlighted that the ADMM scheme provides a flexible approach to decompose dynamic optimization problems in time. To the best of our knowledge, this is the first time ADMM has been implemented for decomposing the time domain in optimal control problems. These problems are often decomposed following the multiple shooting method. We discussed similarities between shooting methods and ADMM, and showed the convergence properties of MM and ADMM on two non-convex dynamic optimization problems and proposed modifications to the regular ADMM scheme to accelerate its convergence.

To overcome poor convergence performance, we proposed a modified ADMM approach with multiple coordination minimization steps at each ADMM iteration. With this modification, we observed that the oscillations in the dual infeasibility and the Lyapunov function were significantly reduced. Furthermore, our results showed that even with only a few additional coordination steps, convergence rates significantly improved, approaching the performance of MM.

Chapter 4 describes a decomposition approach that combines features from algorithms presented in Chapters 2 and 3. As discussed in Chapter 2, the Schur-

complement decomposition is very efficient for solving problems with few coupling variables. However, the performance of this approach quickly deteriorates as the number of coupling variable grows. For problems with a significant number of coupling variables it is often convenient to use the augmented Lagrangian algorithms introduced in Chapter 3. However, these approaches are characterized for having linear and sub-linear rates which can take considerable amount of time to converge. Chapter 4 proposes the use of the ADMM-GMRES algorithm to overcome limitations of Schur-complement and augmented Lagrangian approaches when solving large-scale NLPs with high degree of coupling.

We demonstrated that ADMM provides an effective mechanism to precondition iterative linear solvers and can overcome scalability limitations of Schur complement decomposition. The effectiveness of ADMM-GMRES was demonstrated using linear systems that arise in stochastic optimal power flow problems. Our results indicate that ADMM-GMRES is an order of magnitude faster for a problem that contains up to 2 million variables and 4,000 coupling variables. We solved 35 stochastic quadratic problems and the solution times indicate that the computational benefits of using ADMM-GMRES over Schur decomposition are expected to increase as the number of coupling variables grows.

The approach presented in Chapter 4 provides a general framework for structured KKT systems. Unlike traditional preconditioners, the ADMM preconditioner leverages structure of a great variety of KKT systems, and allows for solving a great number of optimization problems. These include examples in dynamic optimization, stochastic programming, networks, and PDE optimization among many others. Our approach provides then a preconditioning strategy for general problem classes which are lacking in the optimization literature.

Given the scale of optimization problems today, an algorithm with similar properties as ADMM but with faster convergence can make a big difference. In the pursuit of accelerating ADMM for solving structured optimization problems, Chapter 5 proposes modifying the first-order multiplier update of the regular ADMM

scheme with a Newton strategy. Unlike related existing literature on trying to accelerate ADMM, our approach is not based on heuristics but combines ideas from Schur-complement decomposition with the aim of retaining its super-linear convergence properties.

The Newton multiplier update formula derived in Chapter 5 significantly reduced the number of MM and ADMM iterations in our numerical experiments with stochastic QPs and DAE constrained problems. In particular, we observed that for convex QPs consensus of the coupling variables is achieved very quickly. ADMM with the Newton update formula takes advantage of this and converges in 2 or 3 iterations resembling the behavior of Schur-complement decomposition. Moreover, our results indicate that using a Newton update formula makes ADMM remarkably robust almost regardless of the selection of the penalty parameter. This is advantageous since proper tuning of the penalty parameter is challenging and is often problem dependent. Our approach however, is almost insensitive to the value of the penalty parameter.

All these improvements indicate that ADMM with second order updates can be an excellent replacement to Schur-complement decomposition for solving KKT systems that arise in the interior-point method. Given that ADMM follows a first-order update of the coupling variables, our ADMM Newton approach does not form the Schur-complement of the coupling variables reducing computational time. The key idea is to remove one of the steps of the Schur-complement decomposition, and exploit the fact that QPs achieve consensus quickly. Moreover, in the interior-point method as the algorithm progresses the step of the primal variables tends to zero and this can be exploited by our proposed ADMM scheme. Further discussion of these advantages are presented in future work.

Our experience with decomposition approaches highlighted the need for flexible coding frameworks to experiment with new ideas and algorithms. Optimization solvers, however, are often complex code bases that few can extend and modify. Currently, optimization practitioners not only need to master the mathematical

concepts, but also familiarize themselves with low-level programming languages like C++ and Fortran before they can try to implement their decomposition algorithms. For this reason we have developed `PyNumero` which is a Python based package that performs computationally expensive operations in C++ but drives algorithms from Python. The goal of `PyNumero` then is to promote development of numerical algorithms for numerical optimization. In particular, we focus on decomposition algorithms and thus `PyNumero` follows an object-oriented design that facilitates the implementation of block-structured algorithms like those described through Chapter 1 to 5.

Chapter 6 discusses the design and implementation of `PyNumero`. The features and capabilities of the package are demonstrated with a variety of applications. These include implementations of an interior-point solver, the alternating direction of multipliers, progressive hedging, and the Schur-complement decomposition for solving engineering problems. We highlight that the interior-point algorithm in `PyNumero` has comparable solution times with the state of the art solver `Ipopt` when solving a dynamic optimization problem with 200K variables and constraints. The implementation required few lines highlighting the flexibility and ease of use of the tool. This was also evidenced in our codes for ADMM, PH and the Schur-complement decomposition. The package has been made available to the optimization community and resides within `Pyomo` `https://github.com/Pyomo/pyomo/tree/master/pyomo/contrib/pynumero`. Distribution is done through standard software package management systems, including `pip` and `conda`. The package is freely available and we hope researchers in decomposition algorithms for nonlinear optimization find it very useful.

7.2 Future Work

The following are some recommendations for future work:

The Schur-complement decomposition described in Chapter 2 solves subproblems by factoring the full-KKT system of each block. Alternatively one can solve

the subproblems using the null-space method. The null-space method is known be very effective when the number of degrees of freedom is small. However, for general problems with large number of variables and few constraints it is not as effective. Its main limitation comes from the need of forming the null-space matrix of the Jacobian of the constraints. As discussed in Chapter 3 for the case of stochastic programming problems the linking matrices B_i are the identity. Consequently the coupling matrix B is a block-stack matrix composed by identity matrices. Computing the null-space of B is actually inexpensive and applying the null-space in this context seems promising. While most approaches presented in literature focus on exploiting the structure of stochastic problems, little is discussed on exploiting the fact that regardless of the application the matrix B is always block-stack identity. This is an interesting direction of future work.

For the modified ADMM approach presented in Chapter 3 we found that an optimal number of coordination steps can accelerate the convergence of augmented Lagrangian decomposition approaches. Future work can consider the derivation of metrics to determine the optimal number of coordination steps. In the particular case of dynamic optimization problems, comparisons with multiple shooting approaches may help determine optimal number of coordination steps for ADMM.

As part of future work for the ADMM-GMRES preconditioning approach of Chapter 4, one may investigate its performance within a nonlinear interior-point framework. Here, it will be necessary to relax our assumptions on strong convexity and on the full rank of the Jacobian. Preliminary results reported in the literature indicate that different types of primal-dual regularized KKT systems can be used to compute search steps within interior-point methods under such relaxed conditions [Chiang et al., 2017]. For instance, the primal-dual regularized system correspond to the optimality conditions of the QP problem:

$$\begin{array}{ll} \min\limits_{x,q,r} & \frac{1}{2}x^T(D+\delta I)x + c^T x + \frac{1}{2\rho}\|r\|^2 + \frac{\rho}{2}\|Ax+Bq-\frac{1}{\rho}r\|^2 \\ \text{s.t.} & Ax+Bq-\frac{1}{\rho}r \;=\; 0, \quad (y) \end{array} \tag{7.1}$$

As future work one may investigate ADMM variants to precondition such systems. The effectiveness of using ADMM as a preconditioner makes us wonder whether other approaches can be used for preconditioning as well. For instance, inexact dual Newton strategies can potentially be used to precondition structured KKT systems. This is an interesting direction of future work which complements the developments of ADMM strategies of Chapter 5 that use second-order multiplier updates to accelerate convergence.

Another direction for future work considers extending the second-order update ADMM framework presented in Chapter 5. This includes testing the approach within the interior-point algorithm and solve the KKT system in each interior-point iteration using our second-order update version of ADMM. The goal would be to compare with the Schur-complement decomposition to proof that ADMM with second-order updates can be faster since it removes communication. In addition, to avoid factorizing the Schur-complement of each subproblem within the second-order update, future research should consider using the preconditioning conjugate gradient. Unlike solving the overall schur-complement with PCG this approach would update the multipliers of linking constraints for each subproblem with PCG.

For algorithm development and application, future work includes further developments in `PyNumero`. The package must provide more flexibility to adjust easily according to various needs of users. For instance `PyNumero` could include interfaces to different automatic differentiation packages. Currently, `PyNumero` relies on ASL to compute first and second derivatives. Efficient packages available in Python like `CasADi` and `PyAdolC` seem like excellent extensions to `PyNumero`. Another useful extension of `PyNumero` involves implementing all block-algebra operations with parallelization capabilities. The current design uses inheritance and polymorfism to represent block vector and block matrices as `Numpy` and `Scipy` objects. An interesting direction for future work should extend these classes to use `MPI4py` to perform block algebraic operations with parallel computing. Finally, to

improve the computational performance of `PyNumero` even further just-in-time compiling features from `Numba` should be included.

REFERENCES

REFERENCES

Hsl. a collection of fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk/. Accessed: 2016-02-12.

O. Abel and W. Marquardt. Scenario-integrated modeling and optimization of dynamic systems. *AIChE Journal*, 46(4):803–823, April 2000.

Shrirang Abhyankar, Jed Brown, Emil M Constantinescu, Debojyoti Ghosh, Barry F Smith, and Hong Zhang. Petsc/ts: A modern scalable ode/dae solver library. *arXiv preprint arXiv:1806.01437*, 2018.

P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

J. Andersson. *A General-Purpose Software Framework for Dynamic Optimization*. PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, Kasteelpark Arenberg 10, 3001-Heverlee, Belgium, October 2013.

Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 2018.

Paul Armand and Riadh Omheni. A globally and quadratically convergent primaldual augmented lagrangian algorithm for equality constrained optimization. *Optimization Methods and Software*, 32(1):1–21, 2017.

Paul Armand, Jol Benoist, Riadh Omheni, and Vincent Pateloup. Study of a primal-dual algorithm for equality constrained minimization. *Computational Optimization and Applications*, 59(3):405–433, 2014.

Nikhil Arora and Lorenz T. Biegler. Redescending estimators for data reconciliation and parameter estimation. *Computers and Chemical Engineering*, 25(11-12): 1585–1599, 2001.

Sylvain Arreckx, Dominique Orban, and Nikolaj van Omme. Nlp.py: An object-oriented environment for large-scale optimization, 06 2016.

Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

Terranna M. Baranowski and Eugene J. LeBoeuf. Consequence management utilizing optimization. *Journal of Water Resources Planning and Management*, 134(4): 386–394, 2008.

David M. Beazley. Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

A. Benallou, D. E. Seborg, and D. A. Mellichamp. Dynamic compartmental models for separation processes. *AIChE Journal*, 32(7):1067–1078, July 1986.

Michele Benzi and Valeria Simoncini. On the eigenvalues of a class of saddle point matrices. *Numerische Mathematik*, 103(2):173–196, 2006.

Michele Benzi, Gene H. Golub, and Jrg Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

Jonathan Berry, William E. Hart, Cynthia A. Phillips, James G. Uber, and Jean-Paul Watson. Sensor placement in municipal water networks with temporal integer programming models. *Journal of Water Resources Planning and Management*, 132 (4), 2006.

D Bertsekas. Approximation procedures based on the method of multipliers. *Journal of Optimization Theory and Applications*, 23:487–510, December 1977.

D.P. Bertsekas. Multiplier Methods : A Survey. *Automatica*, 12(7):133–145, 1976.

D.P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods (Optimization and Neural Computation Series)*. Athena Scientific, 1 edition, 1982.

D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997. ISBN 1886529191.

J. T. Betts. An Accelerated Multiplier Method for Nonlinear Programming . *Journal of Optimization Theory and Applications*, 21(February):137–174, 1977.

L. T. Biegler. *Nonlinear programming concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics SIAM, Philadelphia, Pa., 2010. ISBN 0898719380.

L. T. Biegler. New nonlinear programming paradigms for the future of process optimization. *AIChE Journal*, 63(4):1178–1193, April 2017.

E. G. Birgin and J. M. Martinez. *Practical augmented Lagrangian methods for constrained optimization*, volume 10. SIAM, 2014.

N. Boland, J. Christiansen, B. Dandurand, A. Eberhard, and F. Oliveira. A parallelizable augmented lagrangian method applied to large-scale non-convex-constrained optimization problems. *Mathematical Programming*, pages 1–34, March 2018. URL `http://search.proquest.com/docview/2009419546/`.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Luc Buatois, Guillaume Caumon, and Bruno Levy. Concurrent number cruncher: a gpu implementation of a general sparse linear solver. *International Journal of Parallel, Emergent and Distributed Systems*, 24(3):205–223, 2009.

J. D. Buys. *Dual algorithms for constrained optimization problems*. PhD thesis, University of Leiden, 1972.

R. H. Byrd. Local convergence of the diagonalized method of multipliers. *Journal of Optimization Theory and Applications*, 26(4):485–500, 1978. ISSN 00223239.

Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. Knitro: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization, 3559, 2006*, pages 35–59. Springer Verlag, 2006a.

Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. Knitro: An integrated package for nonlinear optimization. In *Large Scale Nonlinear Optimization, 3559, 2006*, pages 35–59. Springer Verlag, 2006b.

Christof Bskens and Dennis Wassel. The esa nlp solver worhp. In Giorgio Fasano and Jnos D. Pintr, editors, *Modeling and Optimization in Space Engineering*, volume 73, pages 85–110. Springer New York, 2013.

Yankai Cao, Carl Laird, and Victor Zavala. Clustering-based preconditioning for stochastic programs. *Computational Optimization and Applications*, 64(2):379–406, 2016.

Jordi Castro. An interior-point approach for primal block-angular problems. *Computational Optimization and Applications*, 36(2-3):195–219, 2007. doi: 10.1007/s10589-006-9000-1.

Tsung-Hui Chang, Mingyi Hong, Wei-Cheng Liao, and Xiangfeng Wang. Asynchronous distributed admm for large-scale optimization-part i: Algorithm and convergence analysis. *IEEE Transactions on Signal Processing*, 64(12):3118–3130, 2016.

Nai-Yuan Chiang, Rui Huang, and Victor M. Zavala. An augmented lagrangian filter method for real-time embedded optimization. *Automatic Control, IEEE Transactions on*, 62(12):6110–6121, 2017.

Naiyuan Chiang, Cosmin G Petra, and Victor M Zavala. Structured nonconvex optimization of large-scale energy systems using PIPS-NLP. In *Proc. of the 18th Power Systems Computation Conference (PSCC), Wroclaw, Poland*, 2014.

Andrew R Conn, Nicholas IM Gould, and Philippe Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572, 1991.

Andrew R Conn, GIM Gould, and Philippe L Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*, volume 17. Springer Science & Business Media, 2013.

cyipopt Developers. cyipopt: Cython interface for the interior point optimizer IPOPT, 2017.

J. Dahl and L. Vandenberghe. Cvxopt, 2007. `http://mloss.org/software/view/34/`.

Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011a.

Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124–1139, September 2011b.

G. Di Pillo and L. Grippo. Exact penalty functions in constrained optimization. *SIAM Journal on Control and Optimization*, 27(6):1333–1360, November 1989. ISSN 0363-0129.

Arne Drud. Conopt: A grg code for large sparse dynamic nonlinear optimization problems. *Mathematical Programming*, 31(2):153–191, 1985.

J Eckstein and D Bertsekas. On the douglas-rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55a(3):293–318, January 1992a.

Jonathan Eckstein and Dimitri P Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318, 1992b.

Demetrios G. Eliades and Marios M. Polycarpou. Water contamination impact evaluation and source-area isolation using decision trees. *Journal of Water Resources Planning and Management*, 138(5):562–570, 2012.

Howard C. Elman and Gene H. Golub. Inexact and preconditioned uzawa algorithms for saddle point problems. *SIAM Journal on Numerical Analysis*, 31(6): 1645–1661, 1994.

R. Fletcher. An ideal penality function for constrained optimization. *J.Inst.Maths Applics*, 15(January 1974):319–342, 1975.

Roger Fletcher, Sven Leyffer, and Philippe L. Toint. On the global convergence of a filter–sqp algorithm. *SIAM Journal on Optimization*, 13(1):44–59, 2002.

Anders Forsgren, Philip E. Gill, and Joshua D. Griffin. Iterative solution of augmented systems arising in interior methods. *SIAM Journal on Optimization*, 18(2): 666–690, 2007.

R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press, 1993. ISBN 9780894262333. URL `https://books.google.com/books?id=8vJQAAAAMAAJ`.

Andr Gaul and Nico Schlmer. Preconditioned recycling krylov subspace methods for self-adjoint problems. *arXiv.org*, 2015. URL `http://search.proquest.com/docview/2081864267/`.

Euhanna Ghadimi, Andre Teixeira, Iman Shames, and Mikael Johansson. Optimal parameter selection for the alternating direction method of multipliers (admm): Quadratic problems. *IEEE Transactions on Automatic Control*, 60(3):644–658, 2015.

Philip E. Gill, Walter Murray, Dulce B. Poncelen, and Michael A. Saunders. Preconditioners for indefinite systems arising in optimization. *SIAM Journal on Matrix Analysis and Applications*, 13(1):292–311, 1992.

Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM journal on optimization*, 12(4):979–1006, 2002.

Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Rev.*, 47(1):99–131, January 2005. ISSN 0036-1445.

S. T. Glad. Properties of updating methods for the multipliers in augmented Lagrangians. *Journal of Optimization Theory and Applications*, 28(2):135–156, 1979. ISSN 00223239.

Donald Goldfarb and Shiqian Ma. Fast multiple-splitting algorithms for convex optimization. *SIAM Journal on Optimization*, 22(2):533–556, 2012.

Gene H. Golub and Chen Greif. On solving block-structured indefinite linear systems. *SIAM Journal on Scientific Computing*, 24(6):2076–2092, 2003.

J. Gondzio and A. Grothey. Exploiting structure in parallel implementation of interior point methods for optimization. *Computational Management Science*, 6(2): 135–160, May 2009.

J. Gondzio and R. Sarkissian. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, June 2003.

Nicholas Gould, Dominique Orban, and Philippe Toint. Cutest : a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3):545–557, 2015.

Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. Galahad, a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Softw.*, 29(4):353–372, December 2003. ISSN 0098-3500.

Walter M. Grayman, Avi Ostfeld, and Elad Salomons. Locating monitors in water distribution systems: red team-blue team exercise. *Journal of Water Resources Planning and Management*, 132(4), July 2006. ISSN 0733-9496.

Chen Greif, Erin Moulding, and Dominique Orban. Bounds on eigenvalues of matrices arising from interior-point methods. *SIAM Journal on Optimization*, 24 (1):49–83, 2014.

Andreas Griewank, David Juedes, and Jean Utke. Algorithm 755: Adol-c: A package for the automatic differentiation of algorithms written in c/c++. *ACM Trans. Math. Softw.*, 22(2):131–167, June 1996. ISSN 0098-3500.

Ke Guo, Deren Han, David Wang, and Tingting Wu. Convergence of admm for multi-block nonconvex separable optimization models. *Frontiers of Mathematics in China*, 12(5):1139–1162, 2017.

A Gupta, M Joshi, and V Kumar. Wssmp: A high-performance serial and parallel symmetric sparse linear solver. *Applied Parallel Computing*, 1541:182–194, 1998.

Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2016. URL `http://www.gurobi.com`.

W. Hager. Dual techniques for constrained optimization. *Journal of Optimization Theory and Applications*, 55(1):37–71, October 1987.

Juergen Hahn and Thomas F. Edgar. An improved method for nonlinear model reduction using balancing of empirical gramians. 26(10):1379–1397, 2002. ISSN 0098-1354.

Deren Han and Xiaoming Yuan. Local linear convergence of the alternating direction method of multipliers for quadratic programs. *SIAM Journal on Numerical Analysis*, 51(6):3446–3457, 2013. ISSN 0036-1429.

David Hart, J. Santiago Rodriguez, Jonathan Burkhardt, Brian Borchers, Carl Laird, Katherine Klise, and Terranna Haxton. Quantifying hydraulic and water quality uncertainty to inform sampling of drinking water distribution systems. *to appear in Journal of Water Resources Planning and Management*, 2018.

David B. Hart and Sean A. McKenna. Canary users manual, version 4.3.2. Technical report, U.S. Environmental Protection Agency, Washington, DC, EPA/600/R-08/040B, 2012.

W. E. Hart, J. Watson, and D. L. Woodruff. Pyomo: modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011a.

W. E. Hart, C. D. Laird, J. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, and J. D. Siirola. *Pyomo–optimization modeling in python*, volume 67. Springer Science and Business Media, second edition, 2017.

W.E. Hart, J.P. Watson, and D.L. Woodruff. Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260, 2011b.

W.E. Hart, C.D. Laird, J.P. Watson, and D.L. Woodruff. *Pyomo–optimization modeling in Python*, volume 67. Springer Science & Business Media, 2012a.

William E. Hart and Regan Murray. Review of sensor placement strategies for contamination warning systems in drinking water distribution systems. *Journal of Water Resources Planning and Management*, 136(6), November 2010. ISSN 0733-9496.

William E. Hart, Carl Laird, Jean-Paul Watson, and David L. Woodruff. *Pyomo–optimization modeling in Python*, volume 67. Springer Science & Business Media, 2012b. doi: 10.1007/978-1-4614-3226-5.

Bingsheng He and Xiaoming Yuan. On the $o(1/n)$ convergence rate of the douglasrachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012. ISSN 0036-1429.

Rudi Helfenstein and Jonas Koko. Parallel preconditioned conjugate gradient algorithm on gpu. *Journal of Computational and Applied Mathematics*, 2011.

Erika Hernadez, Steven Hoagland, and Lindell Ormsbee. Water distribution database for research applications. In *World Environmental and Water Resources Congress 2016*, pages 465–474, 2016. ISBN 978-0-7844-7986-5.

Magnus R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969. ISSN 00223239.

JD Hogg and JA Scott. An indefinite sparse direct solver for large problems on multicore machines, 2010.

Mingyi Hong and Zhi-Quan Luo. On the linear convergence of the alternating direction method of multipliers. *Mathematical Programming*, 162(1):165–199, March 2017. ISSN 0025-5610.

Mingyi Hong, Zhi Luo, and Meisam Razaviyayn. Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems. *SIAM Journal on Optimization*, October 2014.

X. Huang and X. Yang. A unified augmented lagrangian approach to duality and exact penalization. *Mathematics of Operations Research*, 28(3):533–552, August 2003.

X. Huang and X. Yang. Further study on augmented lagrangian duality theory. *Journal of Global Optimization*, 31(2):193–210, February 2005.

IBM ILOG. Ibm ilog cplex optimization studio v12.7.0 documentation, 2018. URL `https://www.ibm.com/support/knowledgecenter/SSSA5P`.

Shannon L. Isovitsch and Jeanne M. VanBriesen. Sensor placement and optimization criteria dependencies in a water distribution system. *Journal of Water Resources Planning and Management*, 134(2), March 2008. ISSN 0733-9496.

John Siirola. Parallel schur-complement and admm decomposition strategies for dynamic optimization problems. URL: `http://dimacs.rutgers.edu/events/details?eID=612`, 6 2017.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed 11/03/2019].

J. Kang, N. Chiang, C. D. Laird, and V. M. Zavala. Nonlinear programming strategies on high-performance computers. pages 4612–4620. IEEE, December 2015.

Jia Kang, Yankai Cao, Daniel P. Word, and C. D. Laird. An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition. *Computers and Chemical Engineering*, 71:563–573, 2014.

Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134 (6):516–526, 2008.

Renke Kuhlmann and Christof Bskens. A primaldual augmented lagrangian penalty-interior-point filter line search algorithm. *Mathematical Methods of Operations Research*, 2017.

Renke Kuhlmann and Christof Bskens. A primaldual augmented lagrangian penalty-interior-point filter line search algorithm. *Mathematical Methods of Operations Research*, 87(3):451–483, 2018.

Daniel Laky, Shu Xu, Jose S. Rodriguez, Shankar Vaidyaraman, Salvador Garca Muoz, and Carl Laird. An optimization-based framework to define the probabilistic design space of pharmaceutical processes with model uncertainty. *Processes*, 7(2), 2019.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 7:1–7:6, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-4005-2.

D. Li. Zero duality gap for a class of nonconvex optimization problems. *Journal of Optimization Theory and Applications*, 85(2):309–324, May 1995. ISSN 0022-3239.

Duan Li. Saddle point generation in nonlinear nonconvex optimization. *Nonlinear Analysis*, 30(7):4339–4344, 1997.

Ruipeng Li and Yousef Saad. Gpu-accelerated preconditioned iterative linear solvers. *The Journal of Supercomputing*, 63(2):443–466, 2013.

Youdong Lin and Linus Schrage. The global solver in the lindo api. *Optimization Methods Software*, 24(4-5):657–668, August 2009. ISSN 1055-6788.

Claudia E. Llanos, Mabel C. Sanchez, and Ricardo a. Maronna. Robust Estimators for Data Reconciliation. *Industrial and Engineering Chemistry Research*, 54(18):5096–5105, 2015.

Miles Lubin and Iain Dunning. Computing in operations research using julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.

H. Z. Luo, X. L. Sun, and D. Li. On the convergence of augmented lagrangian methods for constrained global optimization. *SIAM Journal on Optimization*, 18 (4):1209–1230, 2008. ISSN 1052-6234.

C. F. Ma and Q. Q. Zheng. The corrected uzawa method for solving saddle point problems. *Numerical Linear Algebra with Applications*, 22(4):717–730, 2015.

O. L. Mangasarian. Unconstrained lagrangians in nonlinear programming. *SIAM Journal on Control*, 13(4):772–791, July 1975.

Angelica V. Mann, Gabriel A. Hackebeil, and Carl D. Laird. Explicit water quality model generation and rapid multiscenario simulation. *Journal of Water Resources Planning and Management*, 140(5), 2014.

Istvn Maros and Csaba Mszros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681, 1999a.

Istvn Maros and Csaba Mszros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11(1-4):671–681, 1999b.

A Miele. Use of the Augmented Penalty Function in Mathematical Programming Problems) Part 1. *Journal of Optimization Theory and Applications*, 8(2), 1971a.

A Miele. Use of the Augmented Penalty Function in Mathematical Programming Problems) Part 2. *Journal of Optimization Theory and Applications*, 8(2), 1971b.

S. K. Mishra. *Topics in Nonconvex Optimization Theory and Applications*. Springer Optimization and Its Applications, 50. Springer Science and Business Media, 2011.

Stuart Mitchell, Stuart Mitchell Consulting, and Iain Dunning. Pulp: A linear programming toolkit for python, 2011.

Jose Luis Morales and Jorge Nocedal. Automatic preconditioning by limited memory quasi-newton updating. *SIAM Journal on Optimization*, 10(4), 2000.

S.H. Mung Chiang, A.R. Low, J.C. Calderbank, and J.C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, January 2007.

Regan Murray, Terra Haxton, Robert Janke, William E. Hart, Jonathan Berry, and Cynthia Phillips. Sensor network design for drinking water contamination warning systems: a compendium of research results and case studies using the teva-spot software. Technical report, U.S. Environmental Protection Agency, Washington, DC, EPA/600/R-09/141, 2010.

B. A. Murtagh and M. A. Saunders. MINOS 5.1 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, Stanford, Calif., 1987.

Bethany Nicholson, John D. Siirola, Jean Watson, Victor M. Zavala, and Lorenz T. Biegler. pyomo.dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations. *Mathematical Programming Computation*, Dec 2017. ISSN 1867-2957. doi: 10.1007/s12532-017-0127-0. URL https://doi.org/10.1007/s12532-017-0127-0.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

R. J. O'Doherty and B. L. Pierson. A numerical study of multiplier methods for constrained parameter optimization. *International Journal of Systems Science*, 5(2): 187–200, 1974.

Avi Ostfeld and Elad Salomons. Optimal layout of early warning detection stations for water distribution systems security. *Journal of Water Resources Planning and Management*, 130(5), September 2004.

Avi Ostfeld and Elad Salomons. Conjunctive optimal scheduling of pumping and booster chlorine injections in water distribution systems. *Engineering Optimization*, 38(3):337–352, 2006.

Shannon L. Isovitsch Parks and Jeanne M. VanBriesen. Booster disinfection for response to contamination in a drinking water distribution system. *Journal of Water Resources Planning and Management*, 135(6):502–511, 2009.

R. Polyak. Primal-dual nonlinear rescaling method for convex optimization. *Journal of Optimization Theory and Applications*, 122(1):111–157, July 2004. ISSN 00223239.

Marco Propato. Contamination warning in water networks: general mixed-integer linear models for sensor location design. *Journal of Water Resources Planning and Management*, 132(4), July 2006. ISSN 0733-9496.

Alfio Quarteroni. *Numerical mathematics*. Springer, Berlin ; New York, 2nd ed.. edition, 2007. ISBN 3540346589.

S. M. Masud Rana and Dominic L. Boccelli. Contaminant spread forecasting and confirmatory sampling location identification in a water-distribution system. *Journal of Water Resources Planning and Management*, 142(12):4016059, 2016. ISSN 07339496.

R. T. Rockafellar. The multiplier method of hestenes and powell applied to convex programming. *Journal of Theory and Applications*, 12(6):555–562, November 1973.

R. T. Rockafellar. Augmented Lagrange Multiplier Functions and Duality in Nonconvex Programming. *SIAM Journal on Control*, 12(2):268–285, 1974.

R. T. Rockafellar. Augmented lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research*, 1(2):97–116, May 1976. ISSN 0364765X.

R Tyrrell Rockafellar and Roger J Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of operations research*, 16(1):119–147, 1991.

Jose S. Rodriguez, Bethany Nicholson, Carl Laird, and Victor M. Zavala. Benchmarking admm in nonconvex nlps. *Computers and Chemical Engineering*, 119:315–325, 2018.

N. Romero, J. Poulson, B. Marker, J. Hammond, and R. Van de Geijn. Elemental: a new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software*, 39(2), 2012.

Lewis a Rossman. EPANET 2: users manual. *Cincinnati US Environmental Protection Agency National Risk Management Research Laboratory*, 38(September):200, 2000. ISSN 03063127. doi: 10.1177/0306312708089715.

R. Rupp. On the combination of the multiplier method of hestenes and powell with Newton's method. *Journal of Optimization Theory and Applications*, 15(2):169–187, February 1975. ISSN 0022-3239.

Torgeir Rusten and Ragnar Winther. A preconditioned iterative method for saddlepoint problems. *SIAM Journal on Matrix Analysis and Applications*, 13(3):887–904, 1992.

M. Sala, W. Spotz, and M. Heroux. Pytrilinos: High-performance distributed-memory solvers for python. *ACM Transactions on Mathematical Software (TOMS)*, 34, March 2008.

Olaf Schenk and Klaus Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Future Generation Computer Systems*, 20(3):475–487, 2004.

Arpan Seth, Katherine A. Klise, John D. Siirola, Terranna Haxton, and Carl D. Laird. Testing contamination source identification methods for water distribution networks. *Journal of Water Resources Planning and Management*, 142(4):1–11, 2016.

B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: An operator splitting solver for quadratic programs. *ArXiv e-prints*, November 2017.

Defeng Sun, Jie Sun, and Liwei Zhang. The rate of convergence of the augmented lagrangian method for nonlinear semidefinite programming. *Mathematical Programming*, 114(2):349–391, August 2008.

J. Sun, L. Zhang, and Y. Wu. Properties of the augmented lagrangian in nonlinear semidefinite optimization. *Journal of Optimization Theory and Applications*, 129(3): 437–456, June 2006.

R. A. Tapia. Diagonalized multiplier methods and quasi-Newton methods for constrained optimization. *Journal of Optimization Theory and Applications*, 22(2): 135–194, 1977.

Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

U. S. EPA. Water security toolkit user manual – version 1.3. Technical report, U.S. Environmental Protection Agency, Washington, DC, EPA/600/R-14/338, 2015.

Robert Vanderbei. Loqo: An interior point code for quadratic programming. *Optimization Methods and Software*, 11(1):451–484, 1999.

Andreas Wächter and Lorenz T Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

Andreas Waechter and Lorenz T. Biegler. On the implementation of an interior point filter line-search algorithm for large scale nonlinear programming. *Mathematical Programming - Springer*, 106:25–57, 2006.

Chang Wang and Duan Li. Unified theory of augmented lagrangian methods for constrained global optimization. *Journal of Global Optimization*, 44(3):433–458, July 2009. ISSN 0925-5001.

Hui Wang and Kenneth W. Harrison. Bayesian approach to contaminant source characterization in water distribution systems: adaptive sampling framework. *Stochastic Environmental Research and Risk Assessment*, 27(8):1921–1928, 2013. ISSN 14363240.

Jean-Paul Watson, Regan Murray, and William E. Hart. Formulation and optimization of robust sensor placement problems for drinking water contamination warning systems. *Journal of Infrastructure Systems*, 15(4):330–339, 2009.

Jean-Paul Watson, David L Woodruff, and William E Hart. Pysp: modeling and solving stochastic programs in python. *Mathematical Programming Computation*, 4 (2):109–149, 2012.

Brendt Wohlberg. Admm penalty parameter selection by residual balancing. *arXiv.org*, 2017.

Angelica Wong, James Young, Carl D. Laird, William E. Hart, and Sean A. McKenna. Optimal determination of grab sample locations and source inversion in large-scale water distribution systems. In *12th Annual Conference on Water Distribution Systems Analysis (WDSA)*, pages 412–425, 2010.

D. P. Word. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Computational Optimization and Applications*, 59(3):667–689, December 2014.

DanielP. Word, Jia Kang, Johan Akesson, and CarlD. Laird. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Computational Optimization and Applications*, 59(3):667–688, 2014.

Xueyao Yang and Dominic L. Boccelli. Bayesian approach for real-time probabilistic contamination source identification. *Journal of Water Resources Planning and Management*, 140(8), 2014.

Victor M. Zavala. *Computational Strategies for the Optimal Operation of Large-Scale Chemical Processes*. PhD thesis, Carnegie Mellon University, 2008.

Victor M. Zavala. Stochastic optimal control model for natural gas networks. *Computers & Chemical Engineering*, 64:103 – 113, 2014. ISSN 0098-1354.

Victor M Zavala and Mihai Anitescu. Scalable nonlinear programming via exact differentiable penalty functions and trust-region Newton methods. *SIAM Journal on Optimization*, 24(1):528–558, 2014.

Victor M Zavala, Carl D Laird, and Lorenz T Biegler. Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems. *Chemical Engineering Science*, 63(19):4834–4845, 2008.

F. Zhang. *The Schur Complement and Its Applications*. Numerical Methods and Algorithms, 4. Springer US, 2005. ISBN 1280337451.

R. Zhang and J.T. Kwok. Asynchronous distributed admm for consensus optimization. In *31st International Conference on Machine Learning, ICML 2014*, volume 5, pages 3689–3697. International Machine Learning Society (IMLS), 2014. ISBN 9781634393973.

Richard Zhang and Jacob White. Parameter insensitivity in admm-preconditioned solution of saddle-point problems. *arXiv.org*, 2016.

Richard Y. Zhang and Jacob K. White. Gmres-accelerated admm for quadratic objectives. *SIAM Journal on Optimization*, 28(4):3025–3056, 2018.

Ray Daniel Zimmerman, Carlos Edmundo Murillo-snchez, Robert John Thomas, and Life Fellow. Matpower steady-state operations, planning and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, pages 12–19, 2011.

Walter Zulehner. Analysis of iterative methods for saddle point problems: a unified approach. *Mathematics of Computation*, 71(238):479–505, 2002. URL `http://www.ams.org/jourcgi/jour-getitem?pii=S0025-5718-01-01324-2`.

APPENDICES

A. NEWTON STEP FOR GENERAL NLPS

Consider the nonlinear programming problem of the form,

$$\begin{aligned} \min \quad & f(x) \\ & g_L \leq g(x) \leq g_U \\ & x_L \leq x \leq x_U \end{aligned} \tag{A.1}$$

where $x \in \Re^n$ are the primal variables with lower and upper bounds $x_L \in \Re^n$, $x_U \in \Re^n$. The inequality constraints $g : \Re^n \to \Re^m$ are bounded by $g_L \in \Re^m$ and $g_U \in \Re^m$. This problem can be written with an explicit distinction between the equality (defined with $g_L = g_U$) and inequality constraints to give,

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & d_L \leq d(x) \leq d_U \\ & x_L \leq x \leq x_U \end{aligned} \tag{A.2}$$

The equality constraints are represented by $c : \Re^n \to \Re^{m_c}$ and $d : \Re^n \to \Re^{m_d}$ denotes the inequality constraints with bounds $d_L \in \Re^{m_d}$ and $d_U \in \Re^{m_d}$ and $m = m_c + m_d$. Introduction of slack variables reformulates the general inequality constraints to

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & d(x) - s = 0 \\ & x - x_L \geq 0, \quad x_U - x \geq 0 \\ & s - d_L \geq 0, \quad d_U - s \geq 0 \end{aligned} \tag{A.3}$$

with $s \in \Re^{n_d}$. If a variable bound does not exist ($x_L, d_L = -\infty$ or $x_U, d_U = \infty$) the problem can be reformulated with compression matrices as follows,

$$\begin{aligned} \text{min} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & d(x) - s = 0 \\ & (P_x^L)^T x - x_L \geq 0, \quad x_U - (P_x^U)^T x \geq 0 \\ & (P_d^L)^T d(x) - d_L \geq 0, \quad d_U - (P_d^U)^T d(x) \geq 0 \end{aligned} \tag{A.4}$$

where $P_x^L \in \Re^{n \times n_{xL}}$, $P_x^U \in \Re^{n \times n_{xU}}$, $P_d^L \in \Re^{m_d \times n_{dL}}$ and $P_d^U \in \Re^{m_d \times n_{dU}}$ are projection or permutation matrices between variables x and the inequalities $d(x)$ and their corresponding bounds. Symbols n_{xL}, n_{xU}, n_{dL} and n_{dU} represent the number of valid bounds.

In order to derive the primal-dual system, we define the Lagrange function of the reformulated NLP (A.4) as,

$$\begin{aligned} \mathcal{L} = f(x) + \lambda_c^T c(x) + \lambda_d^T (d(x) - s) - z_L^T \left((P_x^L)^T x - x_L \right) - z_U^T \left(x_U - (P_x^U)^T x \right) \\ - \nu_L^T \left((P_d^L)^T s - d_L \right) - \nu_U^T \left(d_U - (P_d^U)^T s \right) \end{aligned} \tag{A.5}$$

where $\lambda_c \in \Re^{m_c}$ and $\lambda_d \in \Re^{m_d}$ are the Lagrange multipliers for the equality and inequality constraints, respectively; $z_L \in \Re^{n_{xL}}$ and $z_U \in \Re^{n_{xU}}$ are multipliers for the lower and upper bounds of the x variables; and $\nu_L \in \Re^{n_{dL}}$ and $\nu_U \in \Re^{n_{dU}}$ are the bound multipliers corresponding to the slack variables (multipliers of inequality constraints).

After eliminating the bounds by adding a logarithmic barrier term to the objective function, the primal-dual optimality conditions of problem ($A.4$) are given by:

$$\begin{aligned}
\nabla_x \mathcal{L} = \nabla_x f(x) + J_c(x)^T \lambda_c + J_d(x)^T \lambda_d - P_x^L z_L + P_x^U z_U &= 0 \\
\nabla_s \mathcal{L} = -\lambda_d - P_d^L \nu_L + P_d^U \nu_U &= 0 \\
Sl_x^L Z_L e - \mu e &= 0 \\
Sl_x^U Z_U e - \mu e &= 0 \\
Sl_d^L V_L e - \mu e &= 0 \\
Sl_d^U V_U e - \mu e &= 0 \\
c(x) &= 0 \\
d(x) - s &= 0
\end{aligned} \tag{A.6}$$

where $J_c^T \in \Re^{n \times m_c}$ and $J_d^T \in \Re^{n \times m_d}$ are the Jacobian matrices of the equality and inequality constraints and the diagonal matrices,

$$\begin{aligned}
Z_L &= \text{diag}(z_L) \\
Sl_x^L &= \text{diag}\left((P_x^L)^T x - x_L\right) \\
Z_U &= \text{diag}(z_U) \\
Sl_x^U &= \text{diag}\left(x_U - (P_x^U)^T x\right) \\
V_L &= \text{diag}(\nu_L) \\
Sl_d^L &= \text{diag}\left((P_d^L)^T s - d_L\right) \\
V_U &= \text{diag}(\nu_U) \\
Sl_d^U &= \text{diag}\left(d_U - (P_d^U)^T s\right)
\end{aligned} \tag{A.7}$$

have appropriate dimensions.

The optimality conditions (A.6) can be viewed as a set of nonlinear equations parameterized in the scalar parameter μ. For the solution of this system, we can

derive a sequence of Newton steps obtained from the linearization of the above expressions,

$$\begin{aligned}
D\Delta x + J_c^T\Delta\lambda_c + J_d^T\Delta\lambda_d - P_x^L\Delta z_L + P_x^U\Delta z_U &= -\nabla_x\mathcal{L} \\
-\Delta\lambda_d - P_d^L\Delta\nu_L + P_d^U\Delta\nu_U &= -\nabla_s\mathcal{L} \\
Z_L(P_x^L)^T\Delta x + Sl_x^L\Delta z_L &= -(Sl_x^L Z_L e - \mu e) \\
-Z_U(P_x^U)^T\Delta x + Sl_x^U\Delta z_U &= -(Sl_x^U Z_U e - \mu e) \\
V_L(P_d^L)^T\Delta s + Sl_d^L\Delta\nu_L &= -(Sl_d^L V_L e - \mu e) \\
-V_U(P_d^U)^T\Delta s + Sl_d^U\Delta\nu_U &= -(Sl_d^U V_U e - \mu e) \\
J_c\Delta x &= -c(x) \\
J_d\Delta x - \Delta s &= -(d(x) - s)
\end{aligned} \tag{A.8}$$

where $D \in \Re^{n\times n}$ is the Hessian matrix. The system of linear equations (A.8) has the following structure,

$$\begin{bmatrix}
D & 0 & J_c^T & J_d^T & -P_x^L & P_x^U & 0 & 0 \\
0 & 0 & 0 & -I & 0 & 0 & -P_d^L & P_d^U \\
J_c & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
J_d & -I & 0 & 0 & 0 & 0 & 0 & 0 \\
Z_L(P_x^L)^T & 0 & 0 & 0 & Sl_L^x & 0 & 0 & 0 \\
-Z_U(P_x^U)^T & 0 & 0 & 0 & 0 & Sl_U^x & 0 & 0 \\
0 & V_L(P_d^L)^T & 0 & 0 & 0 & 0 & Sl_L^s & 0 \\
0 & -V_U(P_d^U)^T & 0 & 0 & 0 & 0 & 0 & Sl_U^s
\end{bmatrix}
\begin{pmatrix} \Delta x \\ \Delta s \\ \Delta\lambda_c \\ \Delta\lambda_d \\ \Delta z_L \\ \Delta z_U \\ \Delta v_L \\ \Delta v_U \end{pmatrix}
= -
\begin{pmatrix} \nabla_x\mathcal{L} \\ \nabla_s\mathcal{L} \\ c(x) \\ d(x) - s \\ Sl_x^L Z_L e - \mu e \\ Sl_x^U Z_U e - \mu e \\ Sl_d^L V_L e - \mu e \\ Sl_d^U V_U e - \mu e \end{pmatrix} \tag{A.9}$$

we refer to this set of linear equations as the primal-dual system. The solution of this system is usually the most expensive step in the algorithm. In the current im-

plementation of PyNumero, the primal-dual system is decomposed by eliminating the bound multipliers leading to the augmented linear system,

$$\begin{bmatrix} D + D_x & 0 & J_c^T & J_d^T \\ 0 & D_s & 0 & -I \\ J_c & 0 & 0 & 0 \\ J_d & -I & 0 & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta s \\ \Delta \lambda_c \\ \Delta \lambda_d \end{pmatrix} = - \begin{pmatrix} \nabla_x \bar{\mathcal{L}} \\ \nabla_s \bar{\mathcal{L}} \\ c(x) \\ d(x) - s \end{pmatrix} \tag{A.10}$$

where,

$$\begin{aligned} \nabla_x \bar{\mathcal{L}} &= \nabla_x f(x) + J_c^T \lambda_c + J_d^T \lambda_d + P_x^U (Sl_x^U)^{-1} \mu e - P_x^L (Sl_x^L)^{-1} \mu e \\ \nabla_s \bar{\mathcal{L}} &= -\lambda_d + P_d^U (Sl_d^U)^{-1} \mu e - P_d^L (Sl_d^L)^{-1} \mu e \\ D_x &= P_x^L (Sl_x^L)^{-1} Z_L (P_x^L)^T + P_x^U (Sl_x^U)^{-1} Z_U (P_x^U)^T \\ D_s &= P_d^L (Sl_d^L)^{-1} V_L (P_d^L)^T + P_d^U (Sl_d^U)^{-1} V_U (P_d^U)^T. \end{aligned}$$

Once the augmented linear system is solved, we can obtain step directions for the bound multipliers from,

$$\begin{aligned} \Delta z_L &= -z_L + (Sl_x^L)^{-1} \left(\mu e - Z_L (P_x^L)^T \Delta x \right) \\ \Delta z_U &= -z_U + (Sl_x^U)^{-1} \left(\mu e + Z_U (P_x^U)^T \Delta x \right) \\ \Delta \nu_L &= -\nu_L + (Sl_d^L)^{-1} \left(\mu e - V_L (P_d^L)^T \Delta s \right) \\ \Delta \nu_U &= -\nu_U + (Sl_d^U)^{-1} \left(\mu e + V_U (P_d^U)^T \Delta s \right) \end{aligned} \tag{A.11}$$

B. LOCAL CONVEXITY

We present some basic concepts of local duality theory with the spirit of elucidate some of the more complex concepts necessary to understand the method of multipliers from a dual viewpoint. Consider nonlinear programming problems of the form

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & h(x) \;=\; 0 \end{aligned} \tag{B.1}$$

Where $x \in \mathbb{R}^n$, $f(x) : \mathbb{R}^n \to \mathbb{R}$ and $h(x) : \mathbb{R}^n \to \mathbb{R}^m$ Assuming that x^* is a local solution of (B.1) the first order conditions

$$\nabla_x f(x^*) + \nabla_x h(x^*)^T \lambda^* = 0 \tag{B.2}$$

$$h(x^*) = 0 \tag{B.3}$$

and the second order conditions

$$z^T \nabla^2_{xx} L(x^*, \lambda^*) z \geq 0 \qquad \forall z \in \{z : \nabla_x h(x^*) z = 0\} \tag{B.4}$$

hold. Here $\nabla^2_{xx} L(x^*)$ is the Hessian of the Lagrangian:

$$\nabla^2_{xx} L(x^*) = \nabla^2_{xx} f(x^*) + (\lambda^*)^T \nabla^2_{xx} h(x^*)$$

For local convexity let us assume that $\nabla^2_{xx} L(x^*)$ is positive definite (in $\mathbb{R}^n$ and not only in the null space of the gradient of the constraints) which guarantees that the

Lagrangian function $L(x, \lambda^*)$ is locally convex at x^*. This implies that x^* is a local solution of the unconstrained problem

$$\min_x \quad f(x) + (\lambda^*)^T h(x).$$

Furthermore, for any λ sufficiently close to λ^* the function $f(x) + \lambda^T h(x)$ will have a local minimum point at a point x near x^*. Thus locally there is a unique correspondence between λ and x through solution of the unconstrained problem

$$\min_x \quad f(x) + \lambda^T h(x). \tag{B.5}$$

To establish the relation between x and λ let us define the dual function $\phi(\lambda)$ as

$$\phi(\lambda) = \min_x \quad f(x) + \lambda^T h(x). \tag{B.6}$$

with first and second derivatives presented in Lemmas B.0.1 and B.0.2

Lemma B.0.1 *The gradient of the dual function is given by:*

$$\nabla_\lambda \phi(\lambda) = h(x) \tag{B.7}$$

Proof: Let $x(\lambda)$ be the minimum of (B.5) in the neighborhood of x^*. We have explicitly from (B.6)

$$\phi(\lambda) = \quad f(x(\lambda)) + \lambda^T h(x(\lambda)).$$

thus

$$\begin{aligned}\nabla_\lambda \phi(\lambda) &= \nabla_\lambda f(x(\lambda)) + \nabla_\lambda \left[\lambda^T h(x(\lambda))\right] \\ &= \nabla_x f(x(\lambda))\nabla_\lambda x(\lambda) + \left[h(x(\lambda)) + \lambda^T \nabla_x h(x(\lambda))\nabla_\lambda x(\lambda)\right] \\ &= \left[\nabla_x f(x(\lambda)) + \lambda^T \nabla_x h(x(\lambda))\right] \nabla_\lambda x(\lambda) + h(x(\lambda)) \\ &= h(x(\lambda))\end{aligned}$$

Lemma B.0.2 *The Hessian of the dual function is given by*[1]*:*

$$\nabla^2_{\lambda\lambda} \phi(\lambda) = -\nabla_x h(x) \nabla^2_{xx} L(x, \lambda)^{-1} \nabla_x h(x)^T \tag{B.8}$$

Proof: Starting from the previous lemma

$$\begin{aligned}\nabla^2_{\lambda\lambda} \phi(\lambda) &= \nabla_\lambda \left[\nabla_\lambda \phi(\lambda)\right] \\ &= \nabla_\lambda \left[h(x(\lambda))\right] \\ &= \nabla_x h(x(\lambda)) \nabla_\lambda x(\lambda)\end{aligned}$$

From the first order conditions we have

$$\begin{aligned}&\nabla_x f(x(\lambda)) + \lambda^T \nabla_x h(x(\lambda)) = 0 \\ &\nabla_\lambda \left[\nabla_x f(x(\lambda)) + \lambda^T \nabla_x h(x(\lambda))\right] = 0 \\ &\nabla^2_{xx} f \nabla_\lambda x(\lambda) + \nabla_x h(x(\lambda)) + \lambda^T \nabla^2_{xx} h(x(\lambda)) \nabla_\lambda x(\lambda) = 0 \\ &\nabla^2_{xx} L(x(\lambda), \lambda) \nabla_\lambda x(\lambda) + \nabla_x h(x(\lambda))^T = 0 \\ &\nabla_\lambda x(\lambda) = -\nabla^2_{xx} L(x(\lambda), \lambda)^{-1} \nabla_x h(x(\lambda))^T\end{aligned}$$

[1]Note the similarity of this functional form with the Schur-complement $S = \sum_i B_i K_i^{-1} B_i^T$ formula.

Substituting back in the previous expression the lemma is proved. Note that from the local convexity assumption $\nabla^2_{xx} L(x^*)$ is positive definite and since x is a regular point near x^*, the Hessian of the dual is negative definite.

Theorem B.0.3 *Suppose that the problem* (B.1) *has a local solution at x^* with corresponding value p^* and Lagrange multiplier λ^*. Suppose also that x^* is a regular point and that the corresponding Hessian of the Lagrangian $L(x, \lambda)$ is positive definite. Then the dual problem*

$$\max \phi(\lambda) \tag{B.9}$$

has a local solution at λ^* with corresponding value $p^*$and x^*as the point corresponding to λ^* in the definition of $\phi(\lambda^*)$.

Proof: By Lemma B.0.1 we know that λ^* satisfies the first order conditions

$$\nabla\phi(\lambda^*) = h(x^*) = 0$$

and by Lemma 2 the Hessian of the dual is negative definite. Thus λ^* satisfies the first-order and second-order sufficiency conditions for an unconstrained maximum point of $\phi(\lambda^*)$. The corresponding value $\phi(\lambda^*)$ is found from the definition of $\phi(\lambda)$ to be p^*

Constrained problems satisfying the local convexity assumption can then be solved by solving the associated unconstrained dual problem.

C. DERIVATION OF PH FROM ADMM

Consider the general structured optimization problem

$$\min_{x_i \in \mathcal{X}, z} \quad \sum_{i \in \mathcal{P}} f_i(x_i)$$
$$A_i x_i + B_i z = 0, \ (y_i) \ i \in \mathcal{P}.$$

For the particular case of a two-stage stochastic program, because the z variables create coupling across all scenarios, $B_i{=}I$ where I is the identity matrix. Applying the standard ADMM update to this problem

$$x_i^{k+1} = \arg\min_{x_i \in \mathcal{X}_i} f_i(x_i) + \left(A_i x_i - z^k\right)^T y_i^k + \frac{\rho}{2}\|A_i x_i - z^k\|^2$$
$$z^{k+1} = \arg\min_{z} \sum_{i \in \mathcal{P}} \left(A_i x_i^{k+1} - z\right)^T y_i^k + \frac{\rho}{2}\|A_i x_i^{k+1} - z\|^2$$
$$y_i^{k+1} = y_i^k + \rho\left(A_i x_i^{k+1} - z^{k+1}\right)$$

Note how to update z one needs to solve the subproblem $\nabla_z \mathcal{L}_\rho(x^{k+1}, z^*, y^k) = 0$

$$\nabla_z \mathcal{L}_\rho(x^{k+1}, z, y^k) = |\mathcal{P}|z - \sum_{i \in \mathcal{P}} \frac{y_i^k}{\rho} + A_i x_i^{k+1}$$

and thus,

$$z^{k+1} = z^* = \frac{1}{|\mathcal{P}|} \sum_{i \in \mathcal{P}} \frac{y_i^k}{\rho} + A_i x_i^{k+1}$$

Consider now the update of y. Summing over all scenarios (partitions)

$$\begin{aligned}\sum_{i\in\mathcal{P}} y_i^{k+1} &= \sum_{i\in\mathcal{P}} y_i^k + \sum_{i\in\mathcal{P}} \rho\left(A_i x_i^{k+1} - z^{k+1}\right)\\ &= \sum_{i\in\mathcal{P}} y_i^k + \rho A_i x_i^{k+1} - |\mathcal{P}|\rho z^{k+1}\\ &= \rho|\mathcal{P}|\left(\frac{1}{|\mathcal{P}|}\sum_{i\in\mathcal{P}}\frac{y_i^k}{\rho} + A_i x_i^{k+1} - z^{k+1}\right)\\ &= \rho|\mathcal{P}|\left(z^{k+1} - z^{k+1}\right)\end{aligned}$$

and thus the dual estimates aggregate to zero after the first iteration of ADMM.

$$\sum_{i\in\mathcal{P}} y_i^{k+1} = 0$$

This feature is observed in the PH algorithm in a slightly different notation $\sum_{i\in\mathcal{P}} p_i w_i = 0$ where p_i is the probability of the ith scenario and $w_i = \frac{y_i}{p_i}$. Given that the aggregation of the multipliers is zero, the update of z can be written as follows:

$$z^{k+1} = \frac{1}{|\mathcal{P}|}\sum_{i\in\mathcal{P}} A_i x_i^{k+1}$$

D. COMPUTING OPERATOR $T_\rho(U)$ USING ADMM

Here we prove that the operator $T_\rho(u)$ can be computed by applying one ADMM iteration. Consider, without loss of generality and in order to simplify the presentation, the case for $\rho = 1$ and a single block problem of the form:

$$\min_{x,z} \quad \frac{1}{2}x^T Dx + c^T x + \frac{1}{2}\|Ax + Bz\|^2 \tag{D.1.1}$$

$$\text{s.t.} \quad Ax + Bz = 0, \ (y). \tag{D.1.2}$$

The results that we derive next can be extended to multiple blocks using induction. The KKT system for problem (D.1) is:

$$\begin{bmatrix} K & A^T B & A^T \\ B^T A & B^T B & B^T \\ A & B & \end{bmatrix} \begin{bmatrix} x \\ z \\ y \end{bmatrix} = \begin{bmatrix} -c \\ 0 \\ 0 \end{bmatrix} \tag{D.2}$$

where $K = D + A^T A$. Applying a Gauss-Seidel splitting to this system at a point $u = (x, z, y)$ leads to the update $u^+ = T(u) = Gu + f$, where $G = M^{-1}N$ and $f = M^{-1}r$. The explicit form of M^{-1} is given by:

$$M^{-1} = \begin{bmatrix} K^{-1} & 0 & 0 \\ -\Sigma^{-1}B^T AK^{-1} & \Sigma^{-1} & 0 \\ (I - B\Sigma^{-1}B^T)AK^{-1} & B\Sigma^{-1} & -I \end{bmatrix} \tag{D.3}$$

where $\Sigma := B^T B$. Having M^{-1} we construct:

$$G = M^{-1}N = \begin{bmatrix} K^{-1} & 0 & 0 \\ -\Sigma^{-1}B^TAK^{-1} & \Sigma^{-1} & 0 \\ (I - B\Sigma^{-1}B^T)AK^{-1} & B\Sigma^{-1} & -I \end{bmatrix} \begin{bmatrix} 0 & -A^TB & -A^T \\ 0 & 0 & -B^T \\ 0 & 0 & -I \end{bmatrix}$$
$$= \begin{bmatrix} 0 & -K^{-1}A^TB & -K^{-1}A^T \\ 0 & \Sigma^{-1}B^TAK^{-1}A^TB & \Sigma^{-1}B^T(AK^{-1}A^T - I) \\ 0 & (B\Sigma^{-1}B^T - I)AK^{-1}A^TB & (B\Sigma^{-1}B^T - I)AK^{-1}A^T - B\Sigma^{-1}B^T + I \end{bmatrix} \tag{D.4}$$

and the right-hand-side-vector

$$f = M^{-1}r = \begin{bmatrix} K^{-1} & 0 & 0 \\ -\Sigma^{-1}B^TAK^{-1} & \Sigma^{-1} & 0 \\ (I - B\Sigma^{-1}B^T)AK^{-1} & B\Sigma^{-1} & -I \end{bmatrix} \begin{bmatrix} -c \\ 0 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} -K^{-1}c \\ \Sigma^{-1}B^TAK^{-1}c \\ (B\Sigma^{-1}B^T - I)AK^{-1}c \end{bmatrix} \tag{D.5}$$

By defining $Q := AK^{-1}A^T$ we can write the explicit form of the update u^+ as:

$$\begin{bmatrix} x \\ z \\ y \end{bmatrix}^+ = \begin{bmatrix} 0 & -K^{-1}A^TB & -K^{-1}A^T \\ 0 & \Sigma^{-1}B^TQB & \Sigma^{-1}B^T(Q - I) \\ 0 & (B\Sigma^{-1}B^T - I)QB & (B\Sigma^{-1}B^T - I)Q - B\Sigma^{-1}B^T + I \end{bmatrix} \begin{bmatrix} x \\ z \\ y \end{bmatrix}$$
$$+ \begin{bmatrix} -K^{-1}c \\ \Sigma^{-1}B^TAK^{-1}c \\ (B\Sigma^{-1}B^T - I)AK^{-1}c \end{bmatrix} \tag{D.6}$$

Upon expansion we obtain:

$$x^+ = -K^{-1}A^TBz - K^{-1}A^Ty - K^{-1}c \tag{D.7.1}$$

$$z^+ = \Sigma^{-1}B^TQBz + \Sigma^{-1}B^T(Q-I)y + \Sigma^{-1}B^TAK^{-1}c \tag{D.7.2}$$

$$\begin{aligned} y^+ = {} & (B\Sigma^{-1}B^T - I)QBz + \left[(B\Sigma^{-1}B^T - I)Q - B\Sigma^{-1}B^T + I\right] y \\ & + (B\Sigma^{-1}B^T - I)AK^{-1}c \end{aligned} \tag{D.7.3}$$

We now show that ADMM delivers the same updates after one iteration. We use the augmented Lagrange function:

$$\mathcal{L}(x,z,y) = x^TDx + c^Tx + (Ax+Bz)^T y + \frac{1}{2}\|Ax+Bz\|^2. \tag{D.8}$$

Initializing at $u = (x,z,y)$, the update x^+ is given by:

$$x^+ = \arg\min_x \mathcal{L}(x,z,y) \tag{D.9}$$

For which the optimality conditions are

$$\nabla_x\mathcal{L}(x,z,y) = (D+A^TA)x + A^TBz + A^Ty + c = 0 \tag{D.10}$$

and thus,

$$\begin{aligned} x^+ &= -(D+A^TA)^{-1}\left[A^TBz + A^Ty + c\right] \\ &= -K^{-1}\left[A^TBz + A^Ty + c\right] \\ &= -K^{-1}A^TBz - K^{-1}A^Ty - K^{-1}c \end{aligned} \tag{D.11}$$

We note that (D.11) and (D.7.1) are equivalent. The update for the coupling variables z^+ is given by:

$$z^+ = \arg\min_z \mathcal{L}(x^+,z,y) \tag{D.12}$$

The optimality conditions are given by:

$$\nabla_z \mathcal{L}(x^+, z, y) = B^T y + B^T A x^+ + B^T B z = 0 \tag{D.13}$$

and thus,

$$\begin{aligned}
z^+ &= -\left(B^T B\right)^{-1} \left[B^T y + B^T A x^+\right] \\
&= -\Sigma^{-1} \left[B^T y + B^T A x^+\right] \\
&= -\Sigma^{-1} \left[B^T y - B^T A^{-1} A^T B z - B^T A^{-1} A^T y - B^T A^{-1} c\right] \\
&= -\Sigma^{-1} \left[B^T y - B^T Q B z - B^T Q y - B^T A^{-1} c)\right] \\
&= -\Sigma^{-1} \left[B^T (I - Q) y - B^T Q B z - B^T A^{-1} c\right] \\
&= \Sigma^{-1} B^T Q B z + \Sigma^{-1} B^T (Q - I) y + \Sigma^{-1} B^T A^{-1} c
\end{aligned} \tag{D.14}$$

We thus have that (D.14) and (D.7.2) are equivalent. Finally, the dual variables are updated as $y^+ = y + (Ax^+ + Bz^+)$. Substituting (D.11) and (D.14) in this expression leads to (D.7.3) .

E. EXPLOITING STRUCTURE VIA REFORMULATION

We present here an example of the reformulation approach for solving a large-scale optimization problem in water distribution systems. The application of interest consist on the optimal sampling of a municipal water network.

Drinking water utilities rely on samples collected from the distribution system to provide assurance of water quality. If a water contamination incident is suspected, samples can be used to determine the source and extent of contamination. By determining the extent of contamination, the percentage of the population exposed to contamination, or areas of the system unaffected can be identified. Using water distribution system models for this purpose poses a challenge because significant uncertainty exists in the contamination scenarios (e.g., injection location, amount, duration, customer demands, contaminant characteristics). This article outlines an optimization framework to identify strategic sampling locations in water distribution systems. The framework seeks to identify the best sampling locations to quickly determine the extent of the contamination while considering uncertainty with respect to the contamination scenarios. The optimization formulations presented here solve for multiple optimal sampling locations simultaneously and efficiently, even for large systems with a large uncertainty space. These features are demonstrated in two case studies.

Drinking water distribution systems can be vulnerable to intentional or accidental chemical and biological contamination. Extensive research has focused on the optimal design of online water quality monitoring systems and sensor technology [Berry et al., 2006, Krause et al., 2008, Hart and McKenna, 2012, Ostfeld and Salomons, 2004, Murray et al., 2010, Hart and Murray, 2010, Propato, 2006, Isovitsch and VanBriesen, 2008, Grayman et al., 2006]. To minimize risk after de-

tection, utilities need to make efficient response decisions in order to minimize the impact of a contamination incident. Drinking water utilities have emergency response plans which outline procedures that should be initiated if contamination is suspected. The emergency response procedures could include steps to isolate the contaminated area of the system, disinfect the system, and issue public health advisories [Baranowski and LeBoeuf, 2008]. For these steps to be effective, it is important to identify the source of the contamination as well as the extent of contamination. This work focuses on a robust implementation of optimization formulations to identify sampling locations that provide this information if contamination is suspected.

In general, drinking water utilities rely on grab samples collected from the distribution system to provide assurance of water quality and meet regulatory requirements. If contamination is suspected, additional grab samples can be collected to identify the extent of contamination and the source of the incident [Eliades and Polycarpou, 2012]. Accurate determination of the source and extent of contamination is challenging because of limited available measurements and significant uncertainty in system hydraulics, contaminant reaction dynamics, and incident details.

Wang and Harrison [2013] placed the problem of uncertainty reduction based on cyclic sampling within a stronger statistical framework. Given a set of potential contamination scenarios, they performed Bayesian updates of the scenario probabilities based on sampled measurement information. To identify a preferred candidate node, they chose to select those locations that maximized entropy. However, their approach was based on enumeration, and they limited themselves to a single sampling location per cycle. This approach, while effective for a single location, was computationally intensive and not tractable for optimal identification of many sampling locations within a single cycle. Rana and Boccelli [2016] used a similar Bayesian approach to that proposed in Wang and Harrison [2013]. However, they introduced a demand forecasting component that significantly reduces the uncer-

tainty in system hydraulics and the number of contamination scenarios that need to be considered in the procedure. This approach was also shown to be effective, but remained computationally intensive.

The work of Wang and Harrison [2013] provided an appropriate statistical framework for uncertainty reduction, however, successful application of the approach depends highly on the effectiveness of measurements taken within each cycle. In previous work, Wong et al. [2010] focused on the optimal determination of sampling locations, and proposed a mixed-integer linear programming (MILP) formulation to determine sampling locations. In contrast with stochastic search algorithms or greedy optimization heuristics, solutions obtained with this approach are guaranteed to be globally optimal. In addition, state-of-the-art MILP solvers like CPLEX [IBM ILOG, 2018] or GUROBI [Gurobi Optimization, Inc, 2016] are able to provide information about the quality of the solution. While Wong et al. [2010] showed that this approach provided good sampling locations for source identification, it lacked a strong statistical basis for updating the probability of individual contamination scenarios. Furthermore, the formulation from Wong et al. [2010] becomes intractable for systems with a large uncertainty space since it grows quadratically with the number of contamination scenarios under consideration.

This article presents two MILP formulations for determining optimal sampling locations within the Bayesian updating framework from Wang and Harrison [2013]. The formulations presented here are exact linear transformations that allow efficient solution with off-the-shelf MILP solvers and seek to reduce uncertainty in the source and extent of contamination during a contamination incident. The contributions of this research effort focus on:

- The use of available binary measurement information (i.e., yes/no) from water quality sensors, customer complaints, or public health surveillance systems to update not only contamination scenario probabilities but to also determine the likely extent of contamination.

- The development of a computational-efficient approach to identify optimal grab sample locations. An optimization-based strategy is proposed to provide recommendations on the most effective locations for additional sampling with the goal of further reducing the uncertainty in the source and extent of contamination.

Uncertainty Reduction Framework

During an incident response, knowing the source of the contamination is important in order to stop more contamination from entering the system. Knowledge of the source would also help define the extent of contamination. Water utilities typically rely on water quality samples to get information regarding the contamination incident. However, if an approach to identify sampling locations requires multiple rounds of samples to provide useful information, then the response actions such as disinfection [Ostfeld and Salomons, 2006, Parks and VanBriesen, 2009] or flushing could be delayed. The method proposed here focuses on reducing the uncertainty about the contamination incident as quickly as possible by selecting optimal locations to sample.

A framework for reducing this uncertainty is presented in Figure E.1. The framework is based upon the Bayesian approach from Wang and Harrison [2013], and it shows the general steps that could be taken following a suspected contamination alarm or alert. The main objective is to maximize the knowledge about the state of the system by taking samples, or as shown in Figure E.1 to minimize the uncertainty about the contamination source and extent. Step 1 of Figure E.1 builds a database of simulation results from potential contamination scenarios with different characteristics (e.g., injection location, amount, duration, customer demands, reaction coefficients). Precomputed simulation results from a set of contamination scenarios need to be available for the proposed method to efficiently and effectively identify optimal sampling locations. In this work, the factorial design of experiments proposed in Hart et al. [2018] was used to quantify uncertainty

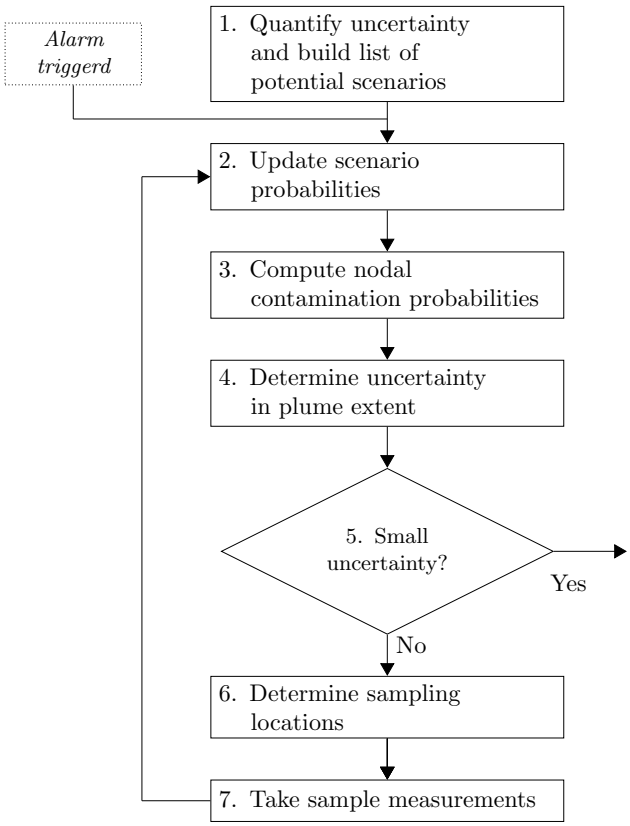


Figure E.1.: Framework used to reduce uncertainty in determining the source and extent of contamination.

and build the list of potential contamination scenarios. If a suspected contamination alarm is triggered, the process to determine sampling locations to help identify the contamination source and extent begins in Step 2. In this step, a Bayesian approach is used to update the probability of the contamination scenarios based on available measurements. Bayesian updates as the ones proposed in the literature[Seth et al., 2016, Wang and Harrison, 2013, Yang and Boccelli, 2014] can be followed in this step. In Step 3, these contamination scenario probabilities and the precomputed simulation results are used to compute, for each node, the probability that the node is contaminated. As measurements are obtained, the probabilities are adjusted and the uncertainty in the contamination source and extent is reduced. Given a particular confidence level (e.g., 95%), nodes can be categorized according to their probability of contamination within Step 4. Any node with a probability higher than a selected threshold are deemed 'likely contaminated' (abbreviated LY for likely yes), any node with a probability less than 1 minus the threshold are deemed 'likely not contaminated' (abbreviated LN for likely no), and the remaining nodes are deemed 'uncertain' (UN). If the number of nodes that remain uncertain is close to zero, then the process is terminated (Step 5), otherwise an optimization-based approach is used to determine the best locations to take additional samples (Step 6). In Step 7, new measurements are taken, and then the approach returns to Step 2.

Optimal sampling (Step 6) must satisfy two primary goals. First, the optimization should identify sampling locations that strengthen the probability update in Step 2. Second, the optimization should be computationally efficient for large water distribution system models and many contamination scenarios. Focusing on the first goal, the ideal measurement locations would maximize the probability of the true contamination scenario. However, two problems with this approach are that: (1) the true scenario is not known a priori, and (2) the Bayesian update would need to be included explicitly within the optimization problem in order to optimize over the probabilities directly. This approach leads to a nonlinear opti-

mization problem with both continuous and discrete variables, which can be very difficult to quickly solve to global optimality. Instead, this work seeks a formulation that remains linear and can be solved with off-the-shelf MILP solvers.

The MILP formulations presented here focus on making significant reductions in the probabilities of unlikely contamination scenarios. To make this point more clear, Figure E.2 shows an example progression of contamination scenario probabilities over several sampling cycles. Initially, in the first sampling cycle no measurements are available, and all contamination scenarios have an equal probability of occurrence. After a round of sampling, the contamination scenario probabilities are updated, with the probabilities increasing for those scenarios that could have led to the measurement result and decreasing for the others. Subsequent sampling rounds further reduce the number of possible likely contamination scenarios. Note that: (1) the best measurement locations are those that rapidly progress from a probability distribution like that in the first sampling cycle to one like that in the final sampling cycle; and (2) out of the initial set of potential contamination scenar-

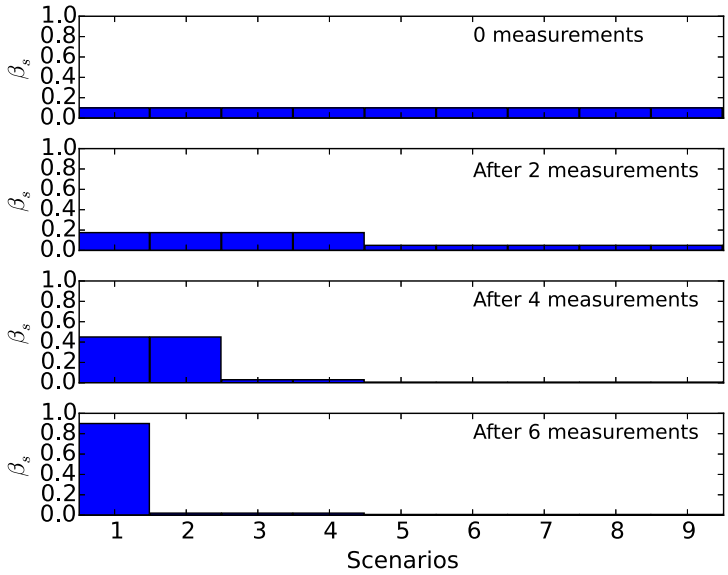


Figure E.2.: Probability of contamination scenarios over multiple sampling rounds.

ios, only a few will agree with all of the measurements, while most will disagree. Because of the second point above, the focus of the optimization formulations is to select locations that maximize the number of disagreements between contamination scenarios and measurements.

Optimal Sampling Formulations

This section presents the derivation of the MILP problem formulations that can be used in Step 6 of Figure E.1 to identify optimal sampling locations. The probability of a location, n, being contaminated is given by Equation (E.1).

$$\gamma_n = \sum_{s \in S} \delta_{s,n} \beta_s \tag{E.1}$$

Here, β_s is the current estimate of the probability of contamination scenario s, and $\delta_{s,n}$ is a binary parameter that is 1 if contamination scenario s contaminates node n, and 0 otherwise. The values of $\delta_{s,n}$ are determined from the simulations precomputed over the full potential contamination scenario set.

Given γ_n (the probability that node n is contaminated), the probability that contamination scenario s is consistent with a perfect measurement (i.e., no false positive or false negative) taken at node n is shown in Equation (E.2).

$$\alpha_{s,n} = \begin{cases} \gamma_n & \text{if } \delta_{s,n} = 1 \\ 1 - \gamma_n & \text{otherwise} \end{cases} \tag{E.2}$$

The probability that contamination scenario s is consistent with the measurements from all selected sampling locations (P_s^{match}) is given by the product of the probabilities $\alpha_{s,n}$ over all selected sampling locations (Equation (E.3)).

$$P_s^{\text{match}} = \prod_{n \in M} \alpha_{s,n} \tag{E.3}$$

Here M is the set of locations (as yet undetermined) selected for sampling in this cycle. The product in Equation (E.3) is over all the *selected* locations, which is determined by the optimization. To reformulate this equation, x_n is a binary variable that will be 1 if location n is selected for sampling, and 0 otherwise. The set M can now be removed, and the product can be written over all candidate sampling locations in N, as in Equation (E.4).

$$P_s^{\text{match}} = \prod_{n \in N} \alpha_{s,n}^{x_n} \tag{E.4}$$

To handle the nonlinearity in Equation (E.4), a set of log transformations (Equations (E.5) and (E.6)) are used to develop two new optimization formulations.

$$P_s^{\text{match}} = \exp(\tilde{P}_s) \tag{E.5}$$

$$\tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) \tag{E.6}$$

The first optimization formulation seeks to maximize the expected number of contamination scenarios that disagree with the expected outcome of the measurements as shown in Equation (E.7).

$$\max \quad \sum_{s \in S} P_s^{\text{miss}} \tag{E.7.1}$$

$$\text{s.t.} \quad P_s^{\text{miss}} = 1 - P_s^{\text{match}} \qquad \forall\, s \in S \tag{E.7.2}$$

$$P_s^{\text{match}} = \exp(\tilde{P}_s) \qquad \forall\, s \in S \tag{E.7.3}$$

$$\tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) \qquad \forall\, s \in S \tag{E.7.4}$$

$$\sum_{n \in N} x_n \leq S_{\text{max}} \tag{E.7.5}$$

$$x_n \in \{0, 1\} \qquad \forall\, n \in N \tag{E.7.6}$$

Here, the parameter S_{max} is the maximum number of samples that can be taken in a sampling cycle. The formulation as written is a mixed-integer nonlinear program (MINLP) because of Equation (E.7.3). However, it can be made linear, since the equality can be replaced with a lower bounding inequality. Since the objective function is maximizing P_s^{miss} (and pushing down on P_s^{match}), this inequality will always be satisfied with equality at the solution. This new inequality is convex and can be replaced with a set of linear under-estimators as graphically presented in the Appendix.

This new MILP formulation, referred to as P1, is shown in Equation (E.8), where v_i are tangent points selected for the linear under-estimators of the exponential term.

$$\max \quad \sum_{s \in S} P_s^{\text{miss}} \tag{E.8.1}$$

$$\text{s.t.} \quad P_s^{\text{miss}} = 1 - P_s^{\text{match}} \qquad \forall\, s \in S \tag{E.8.2}$$

$$P_s^{\text{match}} \geq \exp(v_i) + \exp(v_i)\left(\tilde{P}_s - v_i\right) \qquad \forall\, i \in L,\ s \in S \tag{E.8.3}$$

$$\tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) \qquad \forall\, s \in S \tag{E.8.4}$$

$$\sum_{n \in N} x_n \leq S_{\text{max}} \tag{E.8.5}$$

$$x_n \in \{0, 1\} \qquad \forall\, n \in N \tag{E.8.6}$$

The P1 formulation grows linearly with the number of contamination scenarios, while the distinguishability (D) formulation from Wong et al. (2010) grows quadratically. This is critical for the work presented here, since the uncertainty space considers the source location, system hydraulics, and reaction dynamics which requires large number of scenarios to be characterized.

The second optimization formulation maximizes the worst case number of mismatches (instead of the expected value). This produces the max-min formulation shown in Equation (E.9).

$$\max_{x_n} \quad \min_{s} \quad P_s^{\text{miss}} \tag{E.9.1}$$

$$\text{s.t.} \quad P_s^{\text{miss}} = 1 - P_s^{\text{match}} \qquad \forall\, s \in S \tag{E.9.2}$$

$$P_s^{\text{match}} = \exp(\tilde{P}_s) \qquad \forall\, s \in S \tag{E.9.3}$$

$$\tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) \qquad \forall\, s \in S \tag{E.9.4}$$

$$\sum_{n \in N} x_n \leq S_{\text{max}} \tag{E.9.5}$$

$$x_n \in \{0, 1\} \qquad \forall\, n \in N \tag{E.9.6}$$

Recognizing that $\arg\min_s P_s^{\text{miss}} = \arg\min_s -P_s^{\text{match}}$ and that $\arg\min_x -x = \arg\min_x -\exp(x)$, the entire formulation can be rewritten as Equation (E.10).

$$\max_{x_n} \quad \min_{s} \quad -\tilde{P}_s \tag{E.10.1}$$

$$\text{s.t.} \quad \tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) \qquad \forall\, s \in S \tag{E.10.2}$$

$$\sum_{n \in N} x_n \leq S_{\text{max}} \tag{E.10.3}$$

$$x_n \in \{0, 1\} \qquad \forall\, n \in N \tag{E.10.4}$$

The formulation in Equation (E.10) does not contain the exponential term and is already an MILP without the need for any linear under-estimators, which avoids numerical issues [Bertsimas and Tsitsiklis, 1997] that can occur when too many numerically similar under-estimators are added. Using a standard transformation, this bilevel optimization problem is easily reformulated to a single level optimization problem. The new formulation, referred to as P2, is shown in Equation (E.11), where q is an auxiliary variable that supports the max-min reformulation to a single level optimization problem.

$$\max \quad q \tag{E.11.1}$$

$$\text{s.t.} \quad q \leq -\tilde{P}_s \tag{E.11.2}$$

$$\tilde{P}_s = \sum_{n \in N} x_n \ln(\alpha_{s,n}) \qquad \forall\, s \in S \tag{E.11.3}$$

$$\sum_{n \in N} x_n \leq S_{\max} \tag{E.11.4}$$

$$x_n \in \{0, 1\} \qquad \forall\, n \in N, \tag{E.11.5}$$

Two metrics were used to assess the performance of the P1, P2, and D optimization formulations. The first metric is the number of candidate contamination scenarios within a $\eta\%$ confidence interval (i.e., the n most likely contamination scenarios with a cumulative probability $\eta\%$). The second metric is the fraction of nodes correctly identified. For this calculation, nodes are first labeled in three states. The nodes whose contamination probabilities are above a threshold are labeled as LY for "likely yes," LN for"likely no," and UN for "unknown." The contamination states for each node are identified using Equation (E.12) and used in Step 4 of Figure E.1.

$$\begin{cases} \text{LY} & \text{if} \quad \gamma_n \geq 0.975 \\ \text{UN} & \text{if} \quad 0.025 \leq \gamma_n \leq 0.975 \\ \text{LN} & \text{if} \quad \gamma_n \leq 0.025 \end{cases} \tag{E.12}$$

To compute the fraction of nodes correctly identified, the total number of nodes labeled LY that were also contaminated in the true contamination scenario were added together with the total number of nodes labeled LN that were not contaminated in the true contamination scenario, and then this summation is divided by the total number of nodes in the system. An illustrative example to demonstrate the computations is provided in the following Section.

Illustrative Example

In this section, an example is presented to illustrate the computations described from the optimal sampling formulations Section. The example water distribution system, shown in Figure E.3, has only four nodes and potential injection locations. This example was time-invariant with fixed flow directions as indicated in Figure E.3.

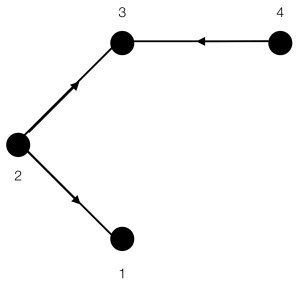


Figure E.3.: Schematic of the simple example four-node water distribution system.

This simple example assumes that the only uncertainty was in the injection location of the contamination scenario, and the set of all candidate contamination scenarios were identified by the location of the contamination scenario ($S = \{1, 2, 3, 4\}$). Additionally, the set of all potential sampling locations includes all nodes, given by $N = \{1, 2, 3, 4\}$. The contamination scenario information can be summarized by the impact parameter $\delta_{s,n}$, in which the rows represented different contamination scenarios and columns represented different sampling locations. An entry $\delta_{s,n}$ in the impact matrix is equal to 1 if location n would be contaminated by scenario s. Initially, when no measurement information was available, the probability distribution of the contamination scenarios β was assumed to be

uniform. The values for δ and β for this simple example are shown in Equation (E.13).

$$\delta = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad \beta = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \tag{E.13}$$

From the impact matrix (δ) and the contamination scenario probabilities (β), the probability that a particular node n is contaminated can be computed using Equation (E.1). For this example, the probabilities of contamination (γ) for each of the four locations are shown in Equation (E.14).

$$\gamma = \left(\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}\right)^T \tag{E.14}$$

Since $\alpha_{s,n}$ represents the probability that contamination scenario s will be consistent with a measurement taken at node n, the matrix of $\alpha_{s,n}$ values for this example is shown in Equation (E.15).

$$\alpha = \begin{pmatrix} \frac{1}{2} & \frac{3}{4} & \frac{1}{4} & \frac{3}{4} \\ \frac{1}{2} & \frac{1}{4} & \frac{3}{4} & \frac{3}{4} \\ \frac{1}{2} & \frac{3}{4} & \frac{3}{4} & \frac{3}{4} \\ \frac{1}{2} & \frac{3}{4} & \frac{3}{4} & \frac{1}{4} \end{pmatrix} \tag{E.15}$$

The P1 and P2 formulations identify locations to take samples that will disagree with many contamination scenarios, or equivalently, to take samples at locations that will agree with only a few contamination scenarios. The expected number of scenarios that will disagree with a measurement taken at location m is given by $E(S) = \sum_{s \in S}(1 - \alpha_{s,m})$. Assuming only a single sampling location is identified per cycle for the simple example system, the expected number of contamination scenarios that disagree with a measurement taken at each of the four locations

would be 2, 1.5, 1.5, and 1.5, respectively. Location 1 would be selected as the optimal sampling location using the P1 formulation. The D formulation would have selected the same location in this case. However, the D formulation does not utilize contamination scenario probabilities within the optimization.

In the previous example, the initial contamination scenario probabilities were assumed to be equal. However, as the uncertainty reduction process continued, the contamination scenario probabilities were updated and the solution with the P1 formulation began to deviate from the D formulation. If the non-uniform probabilities for the contamination scenarios were $\beta = \left(\frac{1}{9}, \frac{6}{9}, \frac{1}{9}, \frac{1}{9}\right)^T$, then the new probabilities of the nodes being contaminated (γ) and the probabilities of agreement (α) would be as shown in Equation (E.16).

$$\gamma = \begin{pmatrix} \frac{7}{9} \\ \frac{6}{9} \\ \frac{8}{9} \\ \frac{1}{9} \end{pmatrix} \qquad \alpha = \begin{pmatrix} \frac{7}{9} & \frac{3}{9} & \frac{1}{9} & \frac{8}{9} \\ \frac{7}{9} & \frac{6}{9} & \frac{8}{9} & \frac{8}{9} \\ \frac{2}{9} & \frac{3}{9} & \frac{8}{9} & \frac{8}{9} \\ \frac{2}{9} & \frac{3}{9} & \frac{8}{9} & \frac{1}{9} \end{pmatrix} \tag{E.16}$$

The D formulation relies on $\delta_{s,n}$ only, so it would have again selected location 1 for sampling, whereas the P1 formulation would have now selected location 2. Given a measurement at location 2, three contamination scenarios were expected to disagree with the measurement. Therefore, only one contamination scenario's probability increases, while the other three scenarios' probabilities decrease. This sampling location resulted in a larger overall reduction in the uncertainty when compared with the location chosen by the D formulation. This simple example highlights that incorporating probability into the selection of the sample location can reduce uncertainty more effectively.

Results and Discussion

The effectiveness of the overall approach and the different formulations for identifying optimal sampling locations were assessed on two different water distribution system models. The first case study used the KL system with 936 nodes from the University of Kentucky's Water Distribution System Research Database (http://www.uky.edu/WDST/database.html) [Hernadez et al., 2016]. The system has a daily production of 11,800 m^3/day (3.12 MGD), with an average residence time of 16 hours. This smaller case study facilitated a comparison between the new formulations (P1 and P2) and the D formulation. Since the D formulation is intractable for large case studies, the comparison with the D formulation is only provided for the smaller test case. The second case study used Network2 from Watson et al. [2009], which has approximately $3{,}000$ nodes, and an initial set of contamination scenarios slightly larger than a million. The system has a daily production of 115,000 m^3/day (30.4 MGD), and an average residence time of 62 hours. For both case studies, the uncertainty reduction approach was demonstrated with a randomly selected "true scenario." The effectiveness of the different formulations was shown in terms of 1) the number of likely contamination scenarios and 2) the fraction of nodes correctly identified.

All of the steps in the uncertainty reduction process were implemented within the Water Security Toolkit (WST) [U. S. EPA, 2015] to simulate the contamination scenarios and identify the optimal sampling locations. The simulations were run in parallel using EPANET and Merlion [Rossman, 2000, Mann et al., 2014] on a Linux server with 24 cores, 264 GB of DDR3 RAM and a clock speed of 2.6 GHz. For all simulated scenarios, nodes with concentration values greater than 0.001 were marked as contaminated nodes. Additionally, in both case studies, 0.05 is used for the probability of measurement error for the Bayesian update of scenario probabilities [Seth et al., 2016]. The optimization formulations were implemented

using Pyomo [Hart et al., 2012b, 2011a] and solved with the state-of-the-art MIP solver ILOGs CPLEX 12.7.0.0 [IBM ILOG, 2018] to an optimality gap of 0.01%.

Case Study 1

Figure E.4 shows the schematic of the system for the first case study. All contamination scenarios considered had a simulation horizon of 96 hours and an injection duration of 90 hours. The parameters used to generate the pool of contamination scenarios are presented in Table E.1. Three different reaction coefficients were considered with a single injection at all 936 nodes starting at hour 6 and finishing at hour 96 in the simulation. This gave a total set of 2,808 contamination scenarios.

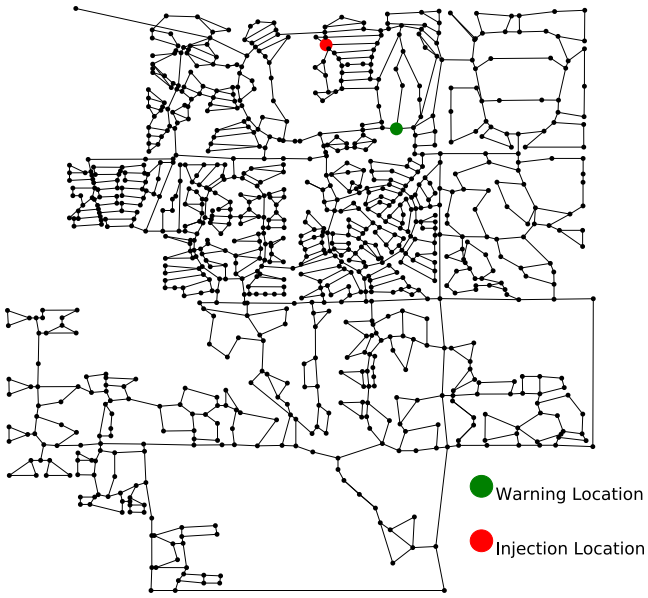


Figure E.4.: Schematic of the first case study system with the the true contamination scenario injection location (red circle) and warning location (green circle) identified.

Table E.1.: Parameters used to create the set of potential contamination scenarios for the first case study system.

Parameter	Values	Units	Values
Reaction coefficient	3	day^{-1}	0,-1,-5
Injection start	1	h	06:00
Injection duration	1	h	90:00
Injection location	936	-	1-936
Injection strength	1	mg/min	1000

For the initial analysis, the 'true scenario' was a contaminant injection at node 344 (marked with a red circle in Figure E.4) with a bulk reaction coefficient of 0.0 day^{-1}. The initial warning was raised at hour 12 from node 258 (shown with a green circle in Figure E.4). Using information from this warning location, the initial set of 2,808 contamination scenarios was reduced to a set of 302 scenarios, which corresponded to the subset of scenarios that could have triggered an alarm at node 258 at hour 12.

Figure E.5 shows the results from the three optimization formulations (P1, P2, and D) in terms of the two metrics described in the formulations Section. On the left axis, the number of candidate contamination scenarios are shown as the dashed lines on a logarithmic scale, and, on the right axis, the fraction of nodes correctly identified is shown as the solid lines. The abscissa includes both the sampling cycle and the progression of time. For this study, four new samples were taken each cy-

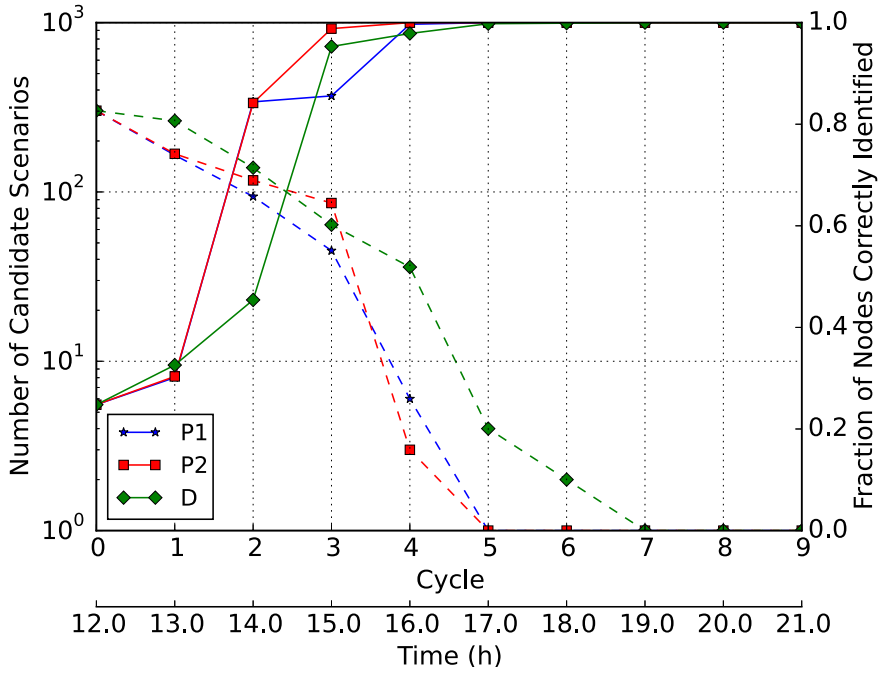


Figure E.5.: The number of candidate contamination scenarios (dashed lines) and the fraction of nodes correctly identified (solid lines) per sampling cycle using the P1, P2, and D formulations.

cle, and the elapsed time between two cycles was 60 minutes. As time progressed, the number of candidate contamination scenarios decreased for all formulations, while the fraction of nodes correctly identified approached 1.0 (Figure E.5). The sampling locations identified with all three formulations quickly reduced the number of candidate contamination scenarios. However, the P1 and P2 formulations both required fewer sampling cycles to determine the true scenario as compared to the D formulation.

Figure E.6 shows the results for all three formulations in terms of the number of nodes labeled as LY (red), LN (blue), and UN (yellow) for each sampling cycle presented in Figure E.5. The solid red line shows the number of nodes that are contaminated by the true contamination scenario at each sampling cycle. The number of uncertain nodes was reduced quickly in the first few sampling cycles. As seen

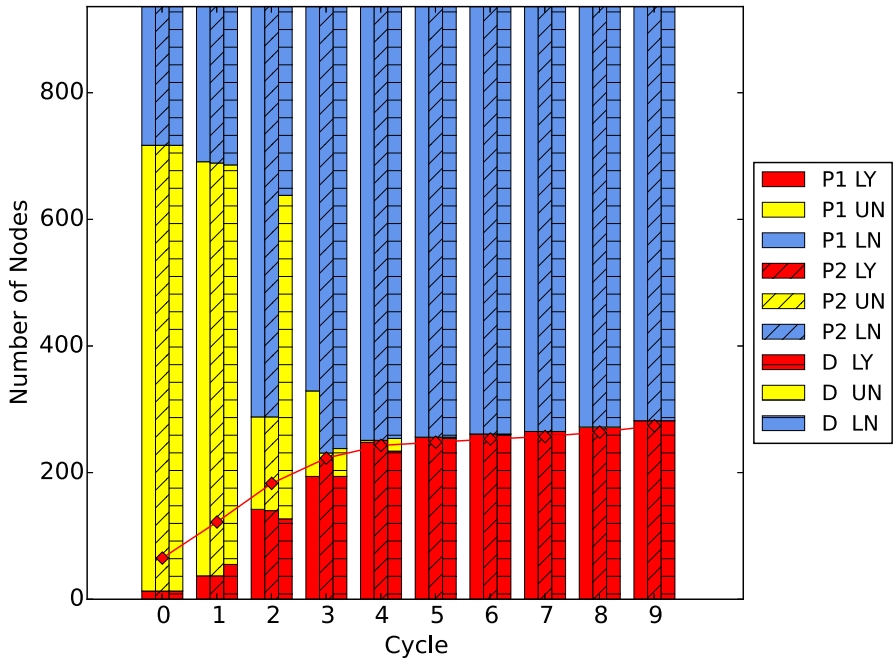


Figure E.6.: Uncertainty reduction comparison showing LY, LN, and UN by sampling at locations identified by the P1, P2, and D formulations. The solid red line is the number of contaminated nodes in the true contamination scenario.

in Figure E.6, when the number of uncertain nodes approached zero, the overall extent of contamination increased, since time was still progressing with each sampling cycle. In these results, the P1 and P2 formulations reduced UN more quickly than the D formulation.

These results only considered a single 'true injection' contamination scenario. To obtain performance statistics, this analysis was repeated for 50 different 'true scenario' cases. In all cases, the start time of the contaminant injection was hour 6, and the initial warning was raised at hour 12. The initial number of candidate contamination scenarios, after the warning was raised, for each case was within the range of 300 to 400.

Figure E.7 shows the number of uncertain nodes per sampling cycle averaged over all 50 true contamination scenarios when four, six, and eight samples were

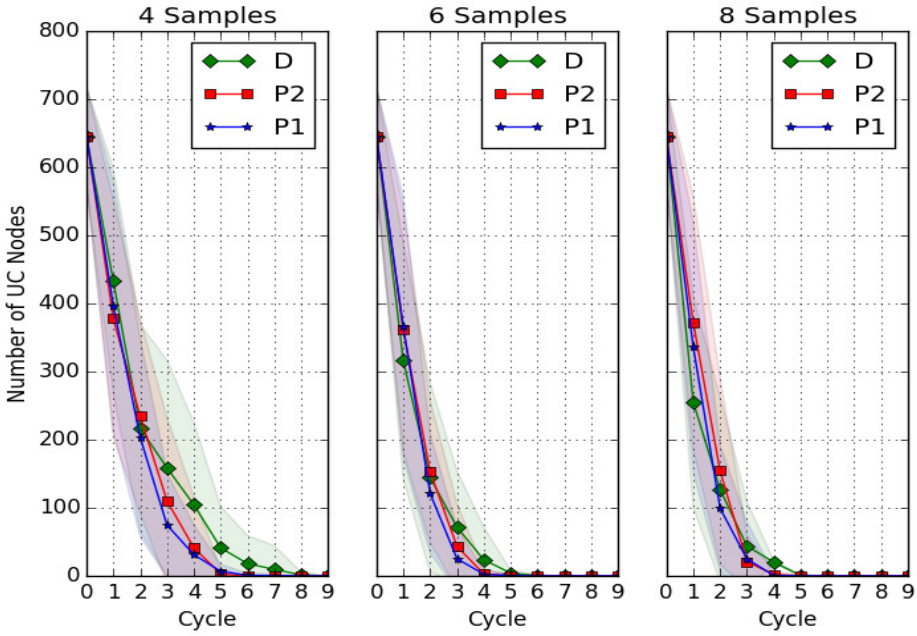


Figure E.7.: Average number of uncertain nodes using four, six, and eight samples per cycle identified by the P1, P2, and D formulations. The elapsed time between sampling cycles was 60 minutes. Each data point is the average for the set of 50 different true contamination scenarios. The shaded area represents the standard deviation.

taken per sampling cycle. The elapsed time between sampling cycles was 60 minutes. The shaded region shows the standard deviation of the number of uncertain nodes for each of the formulations. In almost all instances, the mean and standard deviation were lower for P1 and P2 when compared with the D formulation. However, as the number of measurements per cycle increased the three formulations tended to behave similarly. In addition, as the number of measurements per cycle increased, the standard deviation for all three sampling strategies gets reduced, indicating that taking more measurements reduces uncertainty more quickly over all 50 true scenarios. To investigate the effect of longer durations between sampling cycles, this study was extended for the case of two, four, and eight hours between sampling cycles. The results are presented in the Appendix. In general, the optimization formulations achieved significant reduction in the number of uncertain nodes. However, longer durations between sampling cycles resulted in more uncertainty.

Another reason these new optimization formulations were developed was to improve computational performance. The D formulation scales unfavorably with the number of contamination scenarios, while the size of the new formulations scales linearly. To demonstrate this benefit, the computational times required to solve each of the three formulations is shown in Table E.2. These results were from the first sampling cycle. All timing results represented wall clock time obtained with CPLEX 12.7.0.0. While the P1 formulation took twice as long to solve as the P2 formulation, both were significantly faster than the D formulation and well

Table E.2.: The computional times for the P1, P2, and D formulations for the first case study.

Optimization Formulation	Time to Solution (s)
D	505.42
P1	1.48
P2	0.75

within the required window for use in a real-time context. Note that while these timing results were obtained with a relatively small case study (for comparison purposes), they demonstrate the computational speedup of the P1 and P2 formulations over the D formulation. This is particularly relevant for larger models since the D formulation grows quadratically with the number of scenarios.

The overall results of the first case study indicated that the new optimization formulations (P1 and P2) were more or as effective as the D formulation when used in the proposed uncertainty reduction framework. They generally behaved better in the average case and also produced smaller variation. Also, the P1 and P2 formulations were significantly faster than the D formulation.

Case Study 2

The P1 and P2 formulations were also tested on a larger system model with a larger set of contamination scenarios. Figure E.8 displays a schematic of the system with the true contamination scenario's injection location (red circle) and warning location (green circle) identified.

Table E.3 summarizes the parameters used in this case study. Hydraulic uncertainty was modeled following the procedure described by Hart et al. [2018]. Contaminant injection uncertainty was modeled with 700 different injection locations, each with three different start times (24, 32, and 40 hours in the simulation) and three different durations (1, 12, and 24 hours). In total a set of $1{,}134{,}000$ contamination scenarios were considered. The simulation duration for all scenarios was 96 hours.

As with the first case study, the analysis was first performed with a single 'true contamination scenario', consisting of a contaminant injection at JUNCTION-108 (marked in red in Figure E.8). The contamination scenario was assumed to begin at hour 24 of the simulation and the bulk reaction coefficient was 0.0 day^{-1}. The initial warning was assumed to be at hour 33 at JUNCTION-247 (shown with green in Figure E.8). Based on information from this warning location, the initial set

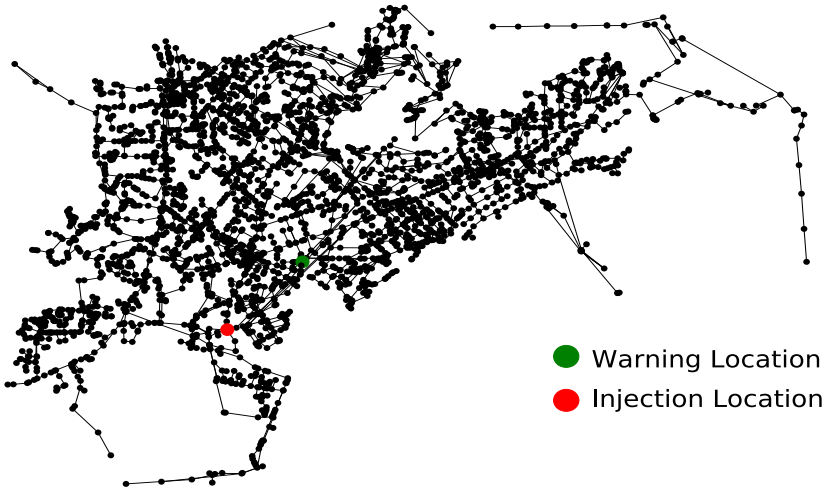


Figure E.8.: A morphed schematic of the second case study system with the contaminant injection (red circle) and the warning (green circle) locations highlighted for the true contamination scenario.

Table E.3.: Parameters used to create the set of potential contamination scenarios for the second case study system (subset of scenarios from [Hart et al., 2018]).

Parameter	Values	Units	Values
Stochastic demand deviation	3	-	0, 0.2, 0.4
Stochastic isolation valve closure	2	-	0, 0.0025
Reaction coefficient	3	day^{-1}	0, -0.1, -5.0
Injection start	3	h	24:00, 32:00, 40:00
Injection duration	3	h	1, 12 , 24
Injection location	700	-	1-700
Injection strength	1	mg/min	1000

of $1{,}134{,}000$ contamination scenarios was reduced to a set of $16{,}347$ scenarios (by selecting only those scenarios that affected JUNCTION-247 at hour 33).

Figure E.9 summarizes the uncertainty reduction results over several sampling cycles for either four, six, or eight samples per cycle using the P1 and P2 formulations. The left axis shows the number of candidate contamination scenarios (scenarios within a 99.9% confidence interval) on a logarithmic scale and the right axis shows the fraction of nodes correctly identified. The x-axis shows the hourly sampling cycles that began at hour 33. Each of the optimizations with P2 solved in less than 20 minutes. As the contamination scenario probabilities were refined, the optimization problem required less time to solve. For the initial sampling cycle, the solution time was around 17 minutes, but from cycle 4 onward, the time was approximately 5 minutes to reach optimality. The optimizations with P1 achieved similar uncertainty reduction results. However, the solution times for all optimizations with P1 were considerably longer. This is because of the higher dimensionality of P1 and the degeneracy introduced by the linear under-estimators which causes ill-conditioning. These degeneracy problems can be avoided with appropriate tuning of the number of under-estimators. However, in this work, no special tuning was done and a fixed number of 20 under-estimators was used for each scenario in all optimizations.

Figure E.10 shows probability maps for the classification of nodes after each sampling cycle for the case of four samples per cycle selected with the P2 formulation. LY nodes are colored in red, UN nodes are colored in yellow, and LN nodes are colored in blue circles. When the initial alarm was triggered (Figure E.10(a)), approximately 30% of the nodes were highly unlikely to be contaminated, while the remaining 70% remained uncertain as to whether they were contaminated. This was a high level of uncertainty in the possible extent of contamination, which would make it difficult for decision makers to identify an effective response action. By cycle 3, the level of uncertainty in the extent of contamination had been

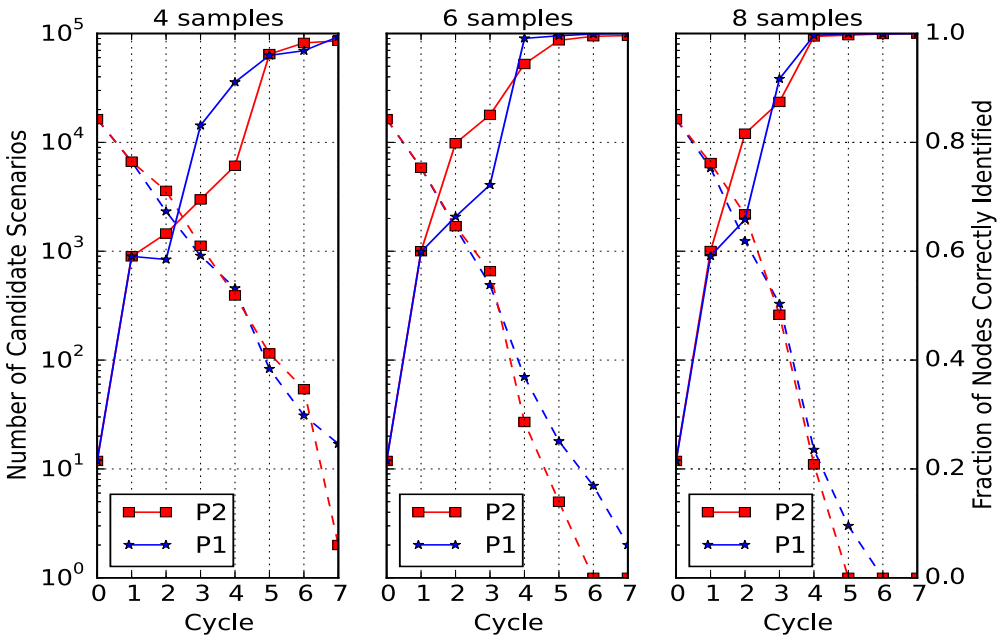
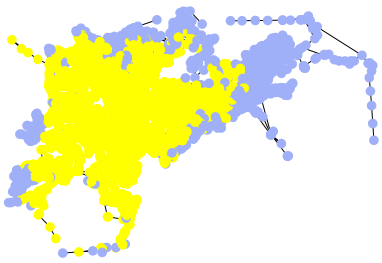
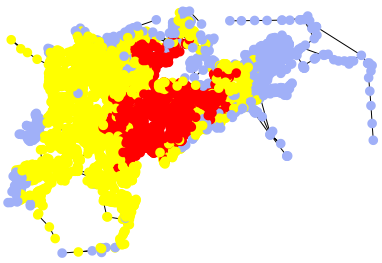


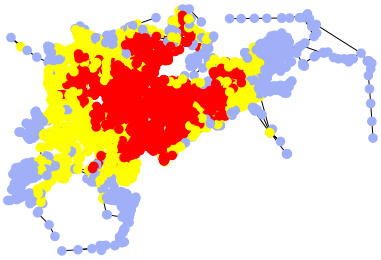
Figure E.9.: The number of candidate contamination scenarios (dashed lines) and the fraction of nodes correctly identified (solid lines) using four, six, and eight samples per cycle for the second case study system.



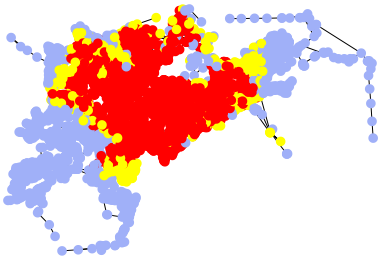
(a) Alarm is triggered – cycle 0.



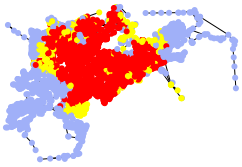
(b) After four measurements – cycle 1.



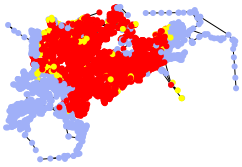
(c) After eight measurements – cycle 2.



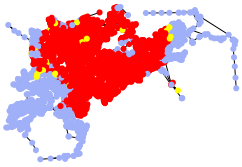
(d) After twelve measurements – cycle 3.



(e) After sixteen measurements – cycle 4.



(f) After twenty measurements – cycle 5.



(g) After twenty-four measurements – cycle 6.

Figure E.10.: Nodal probability maps between sampling cycles.

greatly reduced, and, by cycle 6, the extent of contamination was almost completely characterized.

The uncertainty reduction procedure was repeated for 50 different 'true contamination scenarios', and the mean and standard deviation statistics were calculated for the metrics. For the 50 true scenarios, different contamination source locations were randomly selected from the 700 locations considered in the pool of scenarios. In all 50 cases, the initial warning was raised at hour 33 and the contamination scenario started at hour 24. The initial number of candidate contamination scenarios, after the warning was raised, for each true scenario was within a range of 10,000 to 20,000.

Figure E.11 shows the average fraction of nodes correctly identified for the 50 different contamination scenarios when selecting either four, six, or eight sampling

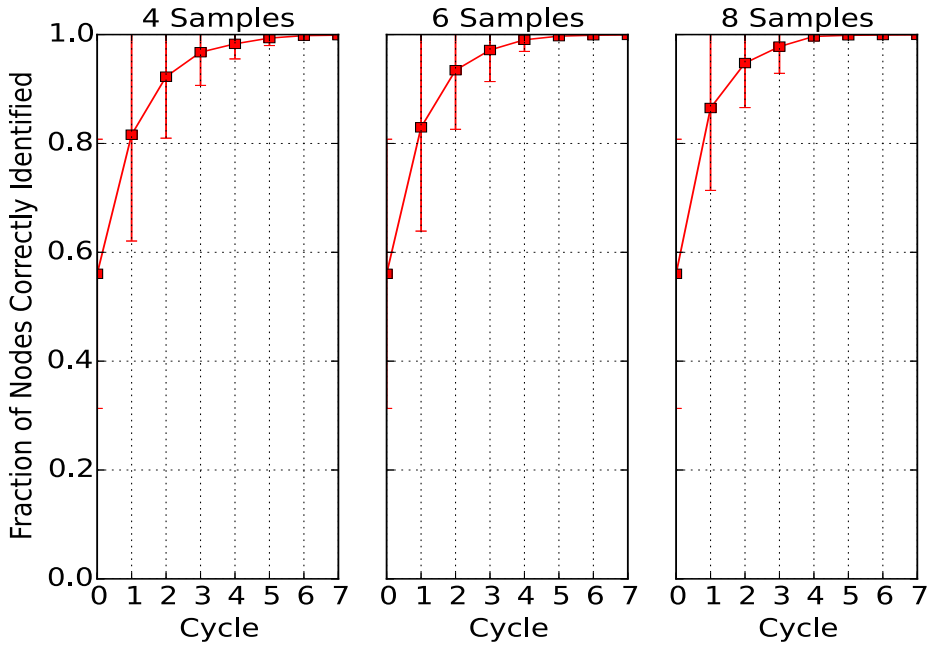


Figure E.11.: Average fraction of nodes correctly identified for the second case study system when selecting either four, six, or eight sampling locations per cycle. Each data point is the average fraction of nodes correctly identified, while the error bars represent one standard deviation above and below per cycle for the 50 different true contamination scenarios.

locations per cycle with formulation P2 (the P1 formulation was not utilized in this study because of the degeneracy problems described previously). The P2 formulation only required a few sampling cycles to significantly reduce the uncertainty in the extent of contamination (a high fraction of the nodes were correctly identified).

The second case study results demonstrated the effectiveness of the P2 formulation to reduce uncertainty in the source and extent of contamination in larger system with an extensive set of potential contamination scenarios.

Summary

Water distribution systems are vulnerable to accidental or intentional contamination incidents, and it is important to develop techniques that can characterize an incident quickly in order to mitigate the effects. This article described an optimization framework to identify sampling locations in a water distribution system in order to efficiently reduce uncertainty in determining the source and extent of contamination. The optimization framework was implemented and made available in the open source software package WST. This work proposed two new optimization formulations to reduce the uncertainty in the contamination source and extent by selecting sampling locations that maximized the expected number of contamination scenarios that disagreed with the measurements. Results for two water distribution system models demonstrated that these formulations efficiently and accurately characterized the source and extent of contamination using the uncertainty reduction framework proposed by Wang and Harrison [2013] . For these formulations to quickly and optimally identify sampling locations, precomputed simulation results from a set of contamination scenarios need to be available. This set of contamination scenarios should be as broad as possible to include estimates of the hydraulic patterns that might be experienced during an incident, such as significant overall demand reduction due to public health notices (e.g., do not drink, do not use). However, the approach does allow the introduction of new contamination scenarios during the sampling process to account for scenar-

ios generated based on real-time data or other system knowledge during an actual incident. While every possible contamination scenario was not included in this study, future work could investigate additional scenarios with varying amounts of contaminant and injection durations, adjustments to the method parameters (e.g., contamination threshold), and applications to larger, realistic water distribution system models. Future work could also develop approaches to reduce the sampling and analysis time to more efficiently identify a contamination incident.