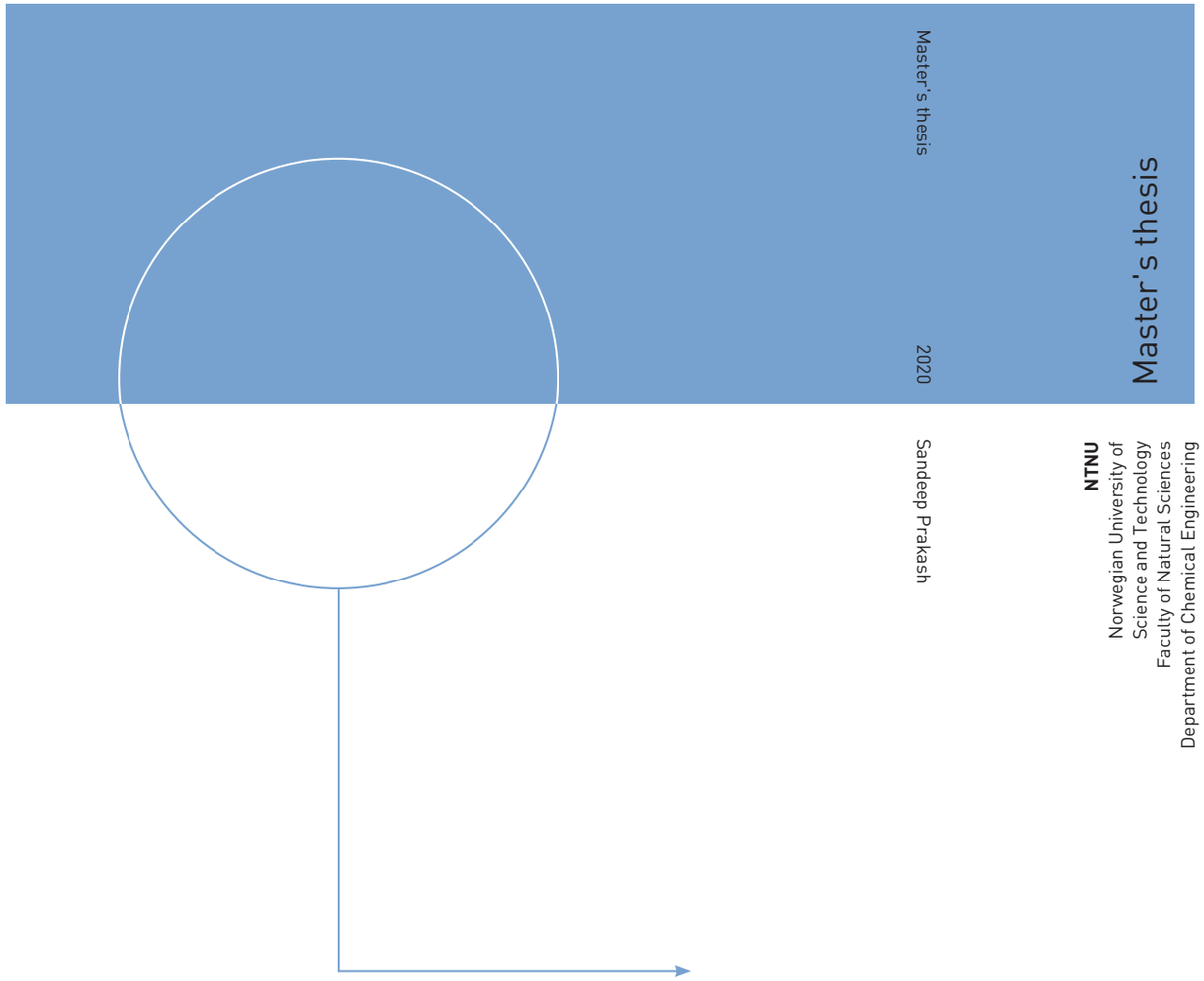

Master's thesis

2020

Sandeep Prakash

NTNU
Norwegian University of
Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Sandeep Prakash

Distributed optimization using ADMM for Optimal Design of Thermal Energy Storage systems

July 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

Distributed optimization using ADMM for Optimal Design of Thermal Energy Storage systems

Sandeep Prakash

Chemical Engineering
Submission date: July 2020
Supervisor: Johannes Jäschke, Associate Professor IKP
Co-supervisor: Mandar Thombre, PhD Candidate IKP

Norwegian University of Science and Technology
Department of Chemical Engineering

Abstract

Our main objective at the beginning of the research period has been on utilizing the alternating direction method of multipliers (ADMM) to get a distributed optimization algorithm to solve structured nonlinear programming problems (NLP). A Thermal Energy Storage (TES) optimal design problem was chosen to motivate the need for distributed optimization and demonstrate the approach using illustrative examples. Due to its cyclic operation and uncertainties in future operating profiles, the optimal sizing of such systems need to consider many scenarios, making the problem size very large.

The traditional approach has been to use a simple linear model to represent the system and then use the multiple scenarios to determine the optimal system capacity. The physical design parameters can then be found from the linear model solution based on some heuristics. The issue with such an approach is that the physical design parameters obtained through this approach is not optimal and would be even infeasible for some of the scenarios considered. This is due to the linear models not accounting the important nonlinear dynamics present in the process. We thus look at ways of using nonlinear dynamic models to make optimal design decisions under uncertainty. The optimal design problem is thus proposed to be framed as a two-stage stochastic nonlinear optimization problem.

An issue that arises with this approach is the large nonlinear optimization problem that this results in. Handling all the variables simultaneously in the memory for solving the problem centrally is expected to require computing hardware specialized for such applications, and something we would want to avoid. Since the speed of arriving at the solution is not a large concern for design problems, we investigate the use of distributed optimization algorithms which solve smaller subproblems iteratively to arrive at the solution to the original large optimization problem. The special structure present in the design problem is exploited to form subproblems in a very general fashion. We make use of ADMM as the distributed optimization algorithm to coordinate between the subproblems. Since the subproblems are able to be solved in parallel each iteration in this approach, it could also be implemented using multiple smaller machines with minimal message passing between them.

Preface

This Master's thesis was written in the spring semester of 2020. It concludes the 2-year master's degree program at the Department of Chemical Engineering at the Norwegian University of Science and Technology (NTNU), leading to an M.Sc. in Chemical Engineering. The final year of my studies was spent at the research group in Process Systems Engineering within the Department of Chemical Engineering. The work performed in this thesis is an extension of the specialization project carried out in the Fall 2019 semester in the same research group.

I would like to thank my supervisor, Associate Professor Johannes Jäschke, for his continued guidance. I appreciate the freedom he has given me to try out various ideas and suggesting relevant research that has been invaluable along the way. My sincere gratitude to PhD candidates Mandar Thombre and Zawadi Ntengua Mdoe, for the long discussions and taking the time in providing invaluable feedback during the writing process. I would also like to extend my gratitude to postdoctoral fellow Dinesh Krishnamoorthy for taking the time to clarify my doubts and helping me with some good coding practices you shared with me.

I cannot thank my dearest friends – Rizwan, Jithin, and Simen enough for being there for me these last two years. Without your constant support, I do not know how I could have made it through the strange period of early 2020. My thoughts go out to my family and friends who have been a source of encouragement throughout.

Declaration of Compliance

I declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU)

Trondheim, Norway

July 13, 2020 Sandeep Prakash

Table of contents

List of figures

List of tables

Chapter 1. Introduction

Reducing our reliance on fossil fuels for meeting our energy demands is one of the critical issues facing the modern world. The transition to a more sustainable energy future is expected to involve an increased proportion of renewable energy sources like wind and solar. These sources however have the disadvantage of being intermittent in their operation and offer fewer degrees of freedom to match the demand. In this regard, energy storage technologies play a crucial role in the integration of renewable energy sources to the energy system, as a way to handle the stochasticity present in the supply and demand of energy [1].

We focus our attention on thermal energy storage (TES) systems which generally refer to the storage of energy as heat or cooling. They are commonly integrated with solar thermal power plants, industrial clusters, hot water systems in buildings, and district heating or cooling networks. A detailed overview of the broad scope of TES systems, the various classifications, and technologies are outlined by Alva et al. [2]. The dynamic nature of the process along with the associated uncertainties with the predictions of future demand and supply makes the problems related to optimal design and in ensuring optimal operations of these systems quite challenging.

The optimal operation for TES systems

The optimal operation of TES system can minimize the reliance on fossil fuels to meet energy demands and improve the profitability of the energy system. Dynamic optimization can be used for this, where the operation is optimized over a time interval, rather than for a single time instance. The process systems community has long made use of detailed dynamic models (based on first principles, observed plant dynamics, or a combination of both) for the control of such processes.

Various authors have focussed on the optimal operation aspect of TES systems in the literature. Powell [3] demonstrates the benefit of integrating TES with various energy systems by solving an open-loop optimization problem. Uncertainties in the forecast of supply and demand profiles were not explicitly accounted in that work. Mdoe [4] later implemented a multistage nonlinear model predictive control (NMPC) to account for uncertainties in the supply and demand side temperatures for a two plant TES system. Data-driven approaches were proposed for scenario selection to represent the uncertainty using fewer number of scenarios for multistage NMPC [5]. Principal

component analysis was used to identify correlations between historical supply and demand profiles, and to build such scenarios for the optimal operation of a TES system [6].

Optimal design for TES systems

During the design stage of an energy storage system, one is primarily interested in modelling the long term profitability of the system. One key issue is addressing the significant uncertainty that exists in the demand and supply profiles during the design phase. Stochastic programming provides a systematic framework to model such problems that require decision making in the presence of uncertainty [7]. The operations research community has long studied location and capacity sizing problems for energy storage systems in electrical networks. Model simplifications are usually used to represent the problem as a two-stage linear stochastic program. This approach was used to solve an electricity network design and capacity expansion problem in the presence of uncertainty in future profiles [8]. A similar approach is used by Lamont [9] to find the optimals size for battery energy storage and pumped hydro-storage systems under similar uncertainties.

For the TES system, a similar approach was followed during the specialization project [10] in the Fall Semester of 2019. The system was modelled in terms of heat duties to get the design problem as a two-stage linear stochastic program. The solution of the linear model was then used to find the physical design parameters (the tank volume and the heat exchanger area) using some heuristics. This approach was extended to include a comparison between the stochastic formulation and a bilevel formulation for the TES design problem by Thombre et al. [11]. However, the issue with such approaches is that they do not use the physical design parameters (like the tank volume or heat exchanger area) directly in the optimization problem, but need to be calculated based on some heuristics from the solution of the approximate linear problem. Such an approach does not give us the optimal physical design parameters for the given uncertainty information. A major issue that arises is due to the linear model approximation that is unable to capture the essential nonlinear dynamics that is present in the system. Hence in some scenarios, the linear problem assume recourse actions that are infeasible to be achieved in reality (for example - transfer of heat duty from a cold stream to a hot stream across a heat exchanger).

To overcome these limitations in the linear model approach, we consider extending the nonlinear dynamic models that are normally used for control applications to make optimal design decisions. Here, we propose to frame the optimal design problem as a two-stage nonlinear stochastic program with the first stage variables as the design parameters and the optimal operation actions as the second stage recourse variables. Since these models represent the real dynamics of the system, no physically infeasible recourse action would be considered in this formulation. However, this approach has the issue of resulting in sizeable nonlinear optimization problems rather quickly. We consider two approaches - a centralized approach and a distributed approach for solving these large optimization problems.

Centralized approach vs Distributed approach for optimization

The two-stage nonlinear design problem can be solved directly using standard nonlinear optimization solvers, which we refer to as the centralized optimization strategy. The problem associated with this is - when the size of the problem grows large, the number of variables that need to be simultaneously held in memory to run this scheme increases, which is expected to be the limiting factor for its application in practice. To address this issue, we explore a distributed optimization approach.

In the distributed optimization strategy, we take advantage of the inherent structure of the problem and use decomposition methods to solve smaller subproblems iteratively. Application of this strategy for scenario decomposition related to multistage NMPC can be found in [12]. Krishnamoorthy et al. presented a primal decomposition algorithm [13] and a dual decomposition algorithm [14] for scenario decomposition in multistage NMPC. The progressive hedging algorithm (PHA) was used for scenario decomposition in a two-stage NMPC by Lucia [15].

In our work, we aim to use the alternating direction method of multipliers (ADMM) algorithm to form parallelizable subproblems in a two-stage NLP, in a more general fashion. The hope is to be able to solve more complex models and multiple scenarios, without the penalty in computational memory required.

1.1. Objective of the Thesis

The main objective of this work has been to explore the ADMM algorithm to solve structured nonlinear programming problems (NLP). Based on the information discussed previously, we use nonlinear dynamic models for optimal design of a TES system to present our appoach. The main focus thus has been on

1. Framing the optimal design problem for a TES system using a nonlinear dynamic model as a two-stage stochastic NLP.

2. Investigating a distributed optimization strategy using ADMM for solving the optimal design problem.

This thesis thus aims to present the methodology and ideas in the context of optimal design of a simple TES case, however it can be applied easily solving structured NLPs found in many different areas.

1.2. Structure of the thesis

The thesis comprises of the following chapters which focus on the following.

In Chapter 2, some technical preliminaries associated with distributed optimization and the ADMM algorithm are presented. A quick background of augmented Lagrangian (AL) methods is provided to solve optimization problems with equality constraints. The basic ADMM algorithm is then presented, which extends the AL method to allow for parallel implementation. Next, we outline how structured NLPs can be presented as a general form consensus problem which is a common form in which many distributed optimization algorithms are presented in literature. The chapter ends by applying the ADMM algorithm to structured NLP and applying any simplifications that occur due to the particular structure present in the problem.

Chapter 3 introduces the TES system under consideration, and the mathematical model used to describe the dynamics of the process. The model is then used to describe the optimal operation problem. An illustrative example is used to analyze the open-loop optimization solution. The chapter ends by motivating the optimal design problem by demonstrating the effect design variables have on optimal operation.

Chapter 4 formally defines the optimal design problem. A simple deterministic case is first shown to explain the basic components involved in the optimization problem. Next, uncertainty is accounted for in the design, and the problem is converted to a two-stage stochastic NLP. An illustrative example is used to compare and contrast the stochastic solution with the solution if we had taken a deterministic approach to design. The chapter next argues how the problem size grows rapidly when considering more scenarios and increasing the prediction horizon. Partitioning the problem is considered for solving the large NLP to reduce memory usage.

In Chapter 5, we demonstrate the approach for scenario decomposition in the context of two-stage NLPs. The problem is shown to be parallelizable by applying the ADMM algorithm. An illustrative example of splitting two scenarios into two partitions is shown using the simplified TES model to comment on the convergence behaviour of the ADMM algorithm.

In Chapter 6, we demonstrate how long prediction horizons that would be required in the design problem could be partitioned and solved using ADMM. A general framework is presented for partitioning a single prediction horizon into many partitions. An illustrative example of splitting a prediction horizon into two partitions using the simplified TES model is presented to comment on the convergence behaviour of the algorithm.

Chapter 7 provides the concluding remarks and presents opportunities for extending this approach in the future.

Chapter 2. Preliminaries on distributed optimization with ADMM

This chapter quickly reviews some basic concepts that we use in later chapters. The standard ADMM algorithm is introduced using a simple equality constrained optimization problem. We then present structured NLPs that are of our interest as a general form consensus problem. We conclude the chapter by applying the ADMM to the structured NLPs and simplifying some of the steps that arise from the structure of the problem itself.

Distributed optimization refers to the general approach of solving a large optimization problem by breaking it into smaller subproblems and solving each of them separately. These subproblems can be solved either sequentially (one subproblem after the other) or in parallel (using separate machines / processors etc.) and put back together to give us the solution to the original large problem. Primal and dual decomposition methods are some of the popular decomposition methods, but they offer feasible iterates and provide convergence under very strong assumptions even in the case of convex optimization problems. A comprehensive review of primal and dual decomposition methods is done by Palomar [16]. Since our aim is to use distributed optimization for a nonlinear optimization problem, we have not considered them in much detail. Instead, we focus on augmented Lagrangian (AL) method which offer more robust convergence properties than primal or dual methods.

2.1. Method of Multipliers

This section we will discuss the method of multipliers, which is an important precursor to understanding the alternating direction method of multipliers (ADMM).

Consider the simple equality constrained optimization problem of the form

$$\min_{x,z} \quad f(x)+g(z) \tag{2.1a}$$

$$Ax+Bz=0 \tag{2.1b}$$

The method of multipliers (MM) or more commonly known as AL method was first introduced by Hestenes and Powell [17]. It was introduced as a way to improve the robustness of standard dual methods which only yield convergence under very strict assumptions like strict convexity and

finiteness of *f* and *g*. The MM scheme solves an optimization problem of the form (2.1) by minimizing the augmented Lagrangian function, given as

$$L_\rho(x,z,\mu) := f(x)+g(z)+\mu^T\left(Ax+Bz\right)+\frac{\rho}{2}\left\|Ax+Bz\right\|_2^2 \tag{2.2}$$

where μ is the associated Lagrange multiplier of the constraint, and ρ is any chosen penalty parameter. The MM scheme is based on the fundamental result that there exists a sufficiently large penalty parameter $\rho>0$ such that the minimizer of the AL function (2.2) is a minimizer of the original problem (2.1). The augmented Lagrangian can thus be seen as the unaugmented Lagrangian of the problem

$$\min_{x,z} \quad f(x)+g(z)+\left\|Ax+Bz\right\|_2^2 \tag{2.3a}$$

$$s.t \quad Ax+Bz=0 \tag{2.3b}$$

This problem is clearly equivalent to the original problem (2.1) since, for any feasible primal variable (*x,z*), the term added to the original objective is zero. The benefit of including the penalty term is that the dual problem of (2.3) can be shown to be differentiable under milder conditions than the dual of the original problem (2.1) [18]. The augmented Lagrangian method is related to the quadratic penalty method [19], but it reduces the possibility of ill-conditioning by explicitly introducing an estimate of the Lagrange multiplier into the function to be minimized [19]. An extensive analysis of the convergence properties of MM and their relation to older ideas of Lagrangian and penalty methods can be found in the monograph by Bertsekas [20].

The MM scheme performs a minimization of the augmented Lagrangian function on the primal variables (x,z) and then updates the Lagrange multipliers using a steepest descent step in the space of μ. The gradient is given by the primal residual $r(x,z) := Ax+Bz$ at the current iteration of the algorithm [21].

The standard MM scheme yields the following algorithm at iteration *k*

$$\left(x^{k+1},z^{k+1}\right) := \arg\min_{x,z} L_\rho(x,z,\mu^k) \tag{2.4a}$$

$$\mu^{k+1} := \mu^k+\rho(Ax^{k+1}+Bz^{k+1}) \tag{2.4b}$$

The superscript *k* is used to indicate the iteration number of the algorithm. The solution of the subproblems at iteration *k* can be used to warm start (initialize) the solver at iteration *k+1* as a way to speed up iterations. It is important to highlight that the MM scheme jointly minimizes on the primal variables, and the quadratic term prevents the algorithm being separable in *x* and *z* even if the original objective function was separable. Consequently, this prevents us from directly implementing the MM in a parallel fashion.

2.2. Alternating Direction Method of Multipliers

ADMM blends the ideas from MM and Gauss-Seidel coordination schemes to enable decomposition and parallel implementation [18]. It is based on the key observation that minimizing over the primal variables *x* and *z* separately (opposed to jointly being done in MM) enables independent subproblems in each partition for structured problems. The standard ADMM scheme has the following steps at iteration *k*

$$x^{k+1} := \arg\min_{x} L_{\rho}(x, z^{k}, \mu^{k}) \tag{2.5a}$$

$$z^{k+1} := \arg\min_{z} L_{\rho}(x^{k+1}, z, \mu^{k}) \tag{2.5b}$$

$$\mu^{k+1} := \mu^{k} + \rho(Ax^{k+1} + Bz^{k+1}) \tag{2.5c}$$

Convergence

We can write the necessary and sufficient condition for optimality for the problem (2.1) at a point (x^{*}, z^{*}, μ^{*}). The primal feasibility condition is

$$Ax^{*} + Bz^{*} = 0 \tag{2.6}$$

and the dual feasibility conditions are (where L_0 represents the unaugmented lagrangian of (2.1))

$$\begin{aligned} 0 &= \nabla_x L_0(x^{*}, z^{*}, \mu^{*}) \\ &= \nabla_x f(x^{*}) + A^{T}\mu^{*} \end{aligned} \tag{2.7}$$

$$\begin{aligned} 0 &= \nabla_z L_0(x^{*}, z^{*}, \mu^{*}) \\ &= \nabla_z g(z^{*}) + B^{T}\mu^{*} \end{aligned} \tag{2.8}$$

At iteration *k+1* of ADMM, the infeasibility of the primal feasibility condition (2.6) can be represented as $r^{k+1} = \left(Ax^{k+1} + Bz^{k+1}\right)$.

Since x^{k+1} minimizes $L_\rho(x\ , z^k, \mu^k)$ by definition (2.5a), the optimality condition for this step is

$$\begin{aligned} 0 &= \nabla_x L_\rho(x^{k+1}, z^k, \mu^k) \\ &= \nabla_x f(x^{k+1}) + A^T\mu^k + \rho A^T(Ax^{k+1} + Bz^k) \\ &= \nabla_x f(x^{k+1}) + A^T(\mu^k + \rho r^{k+1} + \rho B(z^k - z^{k+1})) \\ &= \nabla_x f(x^{k+1}) + A^T\mu^{k+1} + \rho A^T B(z^k - z^{k+1})) \end{aligned}$$

or equivalently,

$$\nabla_x f(x^{k+1}) + A^T\mu^{k+1} = \rho A^T B(z^{k+1} - z^k)$$

The residual in the dual feasibility condition (2.7) at iteration *k+1*, can thus be represented as $s^{k+1} = \rho A^T B(z^{k+1} - z^k)$.

Since z^{k+1} minimizes $L_\rho(x^{k+1}, z, \mu^k)$ by definition (2.5b), the optimality condition for this step is

$$\begin{aligned} 0 &= \nabla_z L_\rho(x^{k+1}, z^{k+1}, \mu^k) \\ &= \nabla_z g(z^{k+1}) + B^T\mu^k + \rho B^T(Ax^{k+1} + Bz^{k+1}) \\ &= \nabla_z g(z^{k+1}) + B^T(\mu^k + \rho r^{k+1}) \\ &= \nabla_z g(z^{k+1}) + B^T\mu^{k+1} \end{aligned}$$

Thus it shows that ADMM iterations always satisfy the dual feasibility condition (2.8).

A good stopping criterion for the algorithm is then the primal and dual residuals to be small and below some specified positive tolerance level $\left\|r^k\right\|_2 \le \varepsilon^{primal}$ and $\left\|s^k\right\|_2 \le \varepsilon^{dual}$.

There are many convergence properties of ADMM discussed in the literature. A fairly general result when applying ADMM to convex problems is presented by Boyd [18] which we quote here

Assumptions 2.1 :

- The functions $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ and $g : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$ are closed, proper and convex
- There exists (x^*, z^*, μ^*), not necessarily unique, for which the unaugmented Lagrangian $L_0(x^*, z^*, \mu) \le L_0(x^*, z^*, \mu^*) \le L_0(x, z, \mu^*)$ holds for all x, z, μ

Under Assumption 2.1, ADMM iterates are shown to satisfy the following properties

- Residual convergence. $r^k \to 0$ as $k \to \infty$, i.e., the iterates approach feasibility
- Objective convergence. $f(x^k) + g(z^k) \to p^*$ as $k \to \infty$, i.e., the objective function of the iterates approach the optimal value
- Dual variable convergence $\mu^k \to \mu^*$ as $k \to \infty$, where μ^* is the dual optimal point

When ADMM is applied to nonconvex problems, it can be considered just as another local optimization method. Even when the minimization steps can be carried exactly, ADMM need not converge, and when it does converge, it need not converge to the optimal point. Although the convergence guarantees do not exist for nonconvex NLPs, in practice however, ADMM has been shown to perform satisfactorily under various case studies, and their convergence comparable to that of MM [21]. It is worth explicitly stating that similar to other local optimization methods, ADMM can converge to a different point depending on the initial values of x^0, z^0, μ^0 and the penalty parameter ρ that are used [18].

There are different extensions and variations to the standard ADMM scheme explored in the literature to improve their convergence properties, some of which are summarized by Boyd [18]. Of particular interest to us have been schemes investigating performing multiple coordination steps between the minimization steps (2.5a) and (2.5b) before performing the dual update step (2.5c). By performing multiple coordination minimization steps, the expectation is that the algorithm approaches the MM scheme (where both *x* and *z* are jointly minimized) convergence properties. In this interpretation, the MM scheme provides the limiting performance of ADMM. Some metrics that would be useful to monitor the progress of ADMM would be the primal and dual residuals every iteration. The AL function decreases monotonously every iteration in the MM scheme but is not necessary for ADMM, which is an indication of insufficient coordination steps being undertaken in the standard ADMM scheme.

2.3. Structured NLPs as general form consensus problems

Our aim is to use distributed optimization for large nonlinear problems. We will focus our attention on structured NLPs which could be partitioned into *P* blocks of the form

$$\min_{x_i, z} \quad \sum_{i=1}^{P} f_i(x_i) \tag{2.9a}$$

$$s.t \quad x_i \in \chi_i \qquad i = 1, 2, ..., P \tag{2.9b}$$

$$A_i x_i + B_i z = 0 \qquad i = 1, 2, ..., P \tag{2.9c}$$

Where the variable $x_i \in \mathbb{R}^{n_{x_i}}$ represents all the variables that correspond to block partition $i \in \mathcal{P} := \{1, \ldots, P\}$. The vector v in the *i*-th partition is represented using the subscript as v_i, and the *i*-th entry of the vector will be represented as $v(i)$ to prevent any ambiguity. The feasible set is built by constraints $\chi_i = \{x_i \mid c_i(x_i) \leq 0\}$. The entire set of partition variables can be collected into a single variable vector $x^T = \left[x_1^T, x_2^T, \ldots, x_P^T\right]$ where $x \in \mathbb{R}^{n_x}$. The vector $z \in \mathbb{R}^{n_z}$ contains all the coupling variables in the problem. Equation (2.9c) is called the linking (complicating) constraint as, without it, the problem would be trivially separable and could have been solved independently. The matrices A_i and B_i help map a subset or a linear combination of the partition variables to the coupling variables. In the simplest case of consensus optimization (local variable component $x_i(j)$ corresponds to global variable component $z(l)$), these would be rather sparse matrices. The Lagrange multiplier associated with the coupling constraint in partition "*i*" is denoted as $\mu_i \in \mathbb{R}^{m_i}$ and the entire set of multipliers associated with the coupling constraints represented as $\mu^T = \left[\mu_1^T, \mu_2^T, \ldots, \mu_P^T\right]$ where $\mu \in \mathbb{R}^m$.

The general form consensus optimization problems can be represented using an undirected graph, and a simple example is shown in Figure 2-1. The left side is the partition optimization problems with their own local variables (the solid dots here represent a scalar element of the vector) and local constraints. The consensus constraint in this case is $x_1(1) = x_2(3) = x_3(2) = z(3)$, which are represented by links connecting each to a global copy $z(3)$ of these private variables.

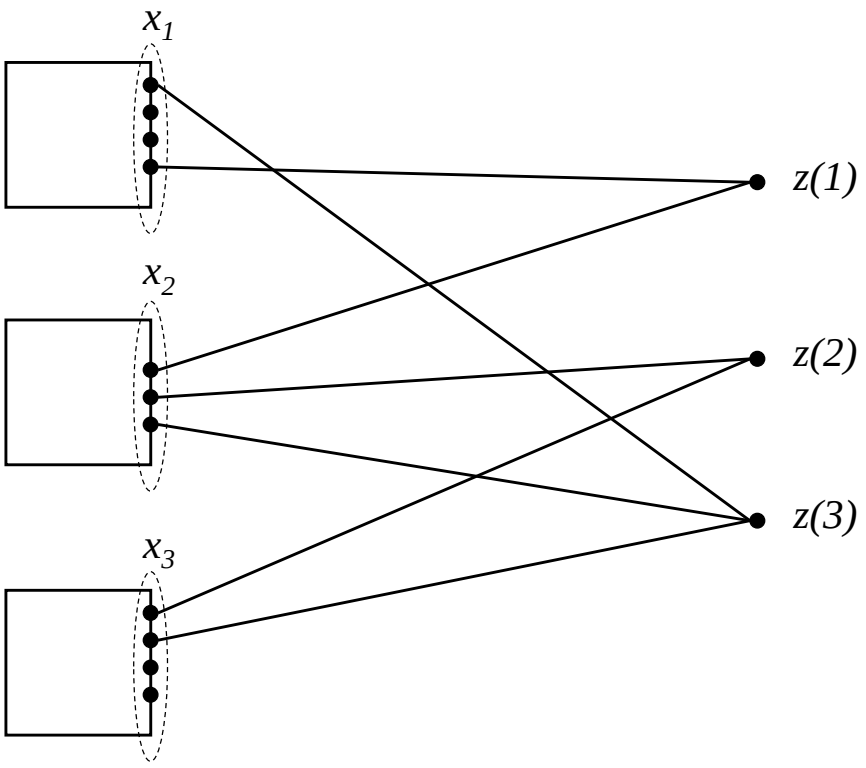

Figure 2-1: Representing a simple general form consensus optimization problem.

The coupling constraint for partition 1 can then be shown to be

$$A_1 x_1 + B_1 z = 0$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(1) \\ x_1(2) \\ x_1(3) \\ x_1(4) \end{bmatrix} + \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} z(1) \\ z(2) \\ z(3) \end{bmatrix} = 0$$

The complete NLP (2.9) could be represented in the compact form as

$$\min_{x,z} \quad f(x) \tag{2.10a}$$

$$s.t \quad x \in \chi \tag{2.10b}$$

$$Ax + Bz = 0 \tag{2.10c}$$

where the feasible set $\chi = \cap_{i=1,..,P} \chi_i$. Matrices A and B can be constructed easily using the partition matrices A_i and B_i given by

$$A = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_P \end{bmatrix} \qquad B = \begin{bmatrix} B_1 \\ \vdots \\ B_P \end{bmatrix} \tag{2.11}$$

A popular distributed optimization paradigm to exploit the structure of the original problem (2.9) is to decompose it into partition subproblems whose solutions are coordinated by some master coordinator scheme. Such approaches are frequently referred to as external decomposition. Some of the well-known approaches like primal decomposition, dual decomposition and ADMM fall into this category.

2.4. ADMM algorithm applied to structured NLPs

This section applies the general ADMM algorithm presented in the previous section to the structured NLPs of the form (2.9). The special structure of the problem is taken advantage of, providing simplifications and analytical solutions to some of the steps in ADMM.

We first form the augmented Lagrangian for the structured NLP (2.9) by taking the coupling constraint to the objective term.

$$L_\rho(x, z, \mu) = \sum_{i=1}^{P} f_i(x_i) + \left(\mu_i\right)^T \left(A_i x_i + B_i z\ \right) + \frac{\rho}{2} \left\| A_i x_i + B_i z\ \right\|_2^2 \tag{2.12a}$$

We can now follow the steps we have outlined for ADMM.

Step (2.5a) in the ADMM algorithm is the update of primal variable *x* which can be written as

$$x^{k+1} = \quad \arg\min_x \quad \sum_{i=1}^{P} f_i(x_i) + \left(\mu_i^k\right)^T \left(A_i x_i + B_i z^k\right) + \frac{\rho}{2} \left\| A_i x_i + B_i z^k \right\|_2^2 \tag{2.13a}$$

$$s.t \quad x_i \in \chi_i \qquad\qquad i = 1, 2, ..., P \tag{2.13b}$$

For the structured NLP, we can parallelize this step in each of the partitions *i* as

$$x_i^{k+1} = \underset{x_i}{\arg\min} \quad f_i(x_i) + \left(\mu_i^k\right)^T \left(A_i x_i + B_i z^k\right) + \frac{\rho}{2}\left\|A_i x_i + B_i z^k\right\|_2^2 \qquad i = 1,2,...,P \qquad (2.14a)$$

$$s.t \quad x_i \in \chi_i \qquad i = 1,2,...,P \qquad (2.14b)$$

Step (2.5b) in the ADMM algorithm is the update of the coupling variable *z* written directly as

$$z^{k+1} = \underset{z}{\arg\min} \quad f(x^{k+1}) + \left(\mu^k\right)^T \left(Ax^{k+1} + Bz\right) + \frac{\rho}{2}\left\|Ax^{k+1} + Bz\right\|_2^2 \qquad (2.15)$$

This step has a closed-form solution which can be derived from the first-order optimality conditions [21] to give

$$z^{k+1} = -\left(B^T B\right)^{-1}\left[\rho B^T A x^{k+1} + B^T \mu\right] \qquad (2.16)$$

This requires the matrix B to have full column rank. In the context of structured NLP, it can be shown [18] that (2.16) is an averaging operator, with the *j-th* element of *z,* has the form

$$z^{k+1}(j) = \frac{1}{\left|\mathcal{P}_j\right|}\sum_{i\in\mathcal{P}_j} x_i^{k+1}(j) \qquad (2.17)$$

where $\mathcal{P}_j \subseteq \mathcal{P} := \{1,\ldots,P\}$ denotes the set of partitions that are connected to the variable $z(j)$. In other words, the *z*-update step is just an averaging of all entries of x_i^{k+1} that correspond to the global copy $z(j)$. A detailed explanation with the derivation of this step is available in [18].

Step (2.5c) is the dual variable update, and can be shown to be solved in the partitions as

$$\mu_i^{k+1} = \mu_i^k + \rho(A_i x_i^{k+1} + B_i z^{k+1}) \qquad (2.18)$$

Chapter 3. Process description and model

In this chapter, we will first describe the TES process and then derive the mathematical model to represent the dynamic behaviour of the process. The optimal operation problem is then framed and the behaviour of the TES system is discussed using an illustrative example. We will then see how design parameters influence optimal operation and motivate the need for optimal design.

3.1. Process description - TES system

The flowsheet in Figure 3-1 is used to represent the heating section of a district heating network. The district heating network uses hot water in a closed-loop as the medium to meet the heating demands of an area. An industrial process represented by the stream q^{wh} is given, which has some cooling demand. This is referred to as the waste heat stream, which could be used as the cheap source of heat for the district heating network. The heat is transferred from the waste heat stream to the district heating circuit using the waste heat boiler (*WHB)*. Additional heat can be added to the water using the peak heat boiler (*PHB)* which uses a more expensive source of energy for heating (fossil fuel or electricity).

The demand for energy from consumers is represented as follows. The total volumetric flow in the district heating network is represented by q^{dh} and is determined by the number of consumers and their demand for hot water at any particular time. Water is returned by the consumers at a temperature $T^{dh,\,Ret}$, which is assumed to be correlated only to the ambient conditions due to the heat losses in the long return pipeline. Water is heated by *PHB* to temperature T^{phb} and must be above a contractually specified temperature $T^{dh,\,minSup}$ before it can be supplied to the consumers. Hence q^{dh}, $T^{dh,\,Ret}$ are time-varying external inputs to our system while $T^{dh,\,minSup}$ provides us with a lower limit for T^{phb} that must always satisfy during operations. The *PHB* adds heat duty Q^{phb} to the water, and the cost associated with this is assumed to be proportional to Q^{phb} with a per-unit cost C^{Qphb}. The total energy that needs to be added by the district heating system is calculated as $Q^{demand} = \rho^{dh} q^{dh} C_p^{\,dh} (T^{dh,\,minSup} - T^{dh,\,Ret})$, where ρ^{dh} and C_p^{dh} represent the density and the specific heat capacity of water in the district heating circuit.

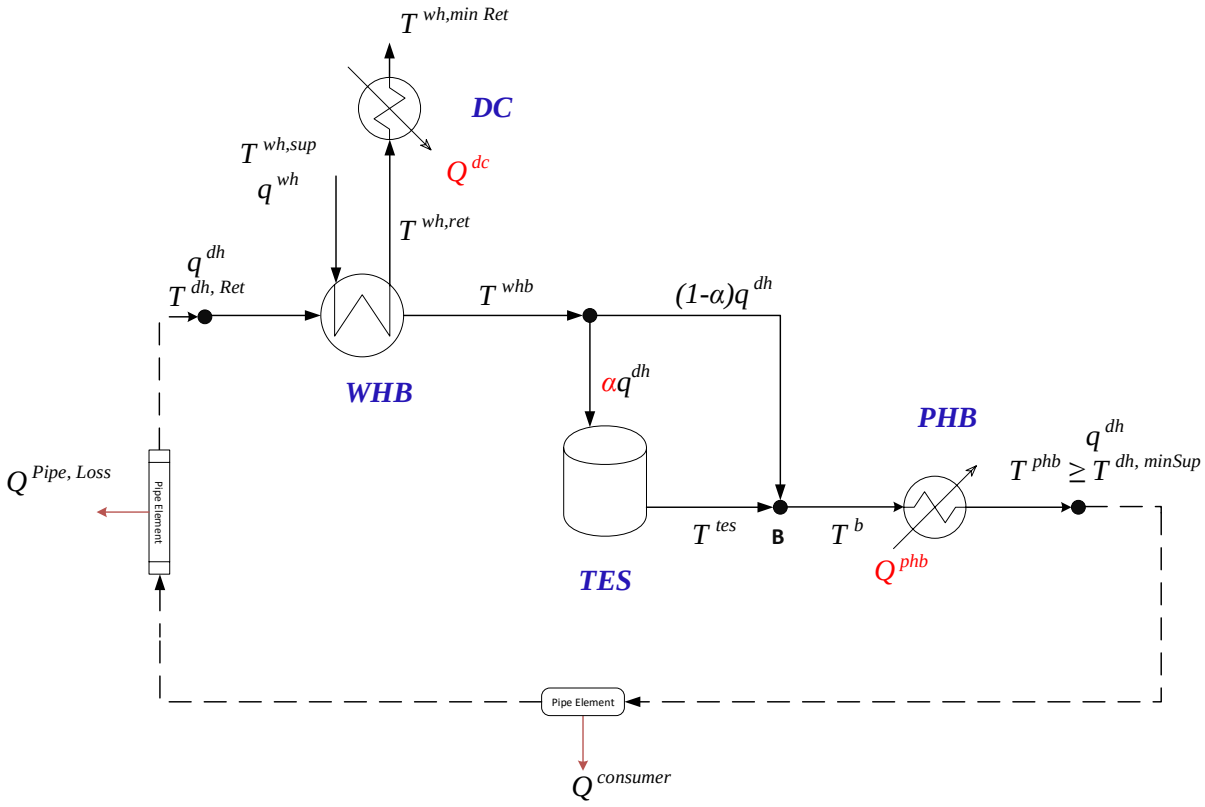

Figure 3-1: Flowsheet for the TES system

The supply of energy from the industrial process is represented as follows. The process has a stream at volumetric flow q^{wh} and temperature $T^{wh,s\ up}$. The process requires that this stream be cooled down to at least $T^{wh,minRet}$. Similar to the consumer side, q^{wh} and $T^{wh,\ sup}$ are determined by the industrial process and are time-varying parameters to the system. Heat is transferred from the process stream to the district heating system using a heat exchanger *WHB*. The exit temperatures for the process stream and water are $T^{\ wh,ret,}$ and T^{whb}, respectively. To meet the process stream return temperature specification of $T^{wh,minRet}$, a dump cooler (*DC*) is used which can remove excess heat duty Q^{dc} by use of some external cooling utility. The use of this cooling utility has a cost associated with it and is assumed proportional to Q^{dc} with a per-unit cost of C^{Qdc}. The total available supply of energy from the process is thus calculated as $Q^{supply} = \rho^{wh} q^{wh} C_p^{\ wh} (T^{wh,\ sup} - T^{wh,\ minRet})$, where ρ^{wh} and C_p^{wh} represent the density and the specific heat capacity of the waste heat stream.

The heating and energy storage functions are represented as follows. The water is first heated using the *WHB* up to T^{whb}. During periods where the demand for energy exceeds the heat duty being extracted from *WHB*, the water at the exit of the *WHB* would not be hot enough to meet the consumer temperature specifications. Hence it will need to be heated further using the *PHB* in order to satisfy the return temperature constrain $T^{phb} >= T^{wh,\,minRet}$. During periods where the demand for energy is lower than is being extracted from the *WHB*, we do not need additional heating from the *PHB* and can directly send the water to the consumers. The temperature being sent would be higher than the minimum return temperature specification. A TES tank is proposed to be constructed that can be used as a buffer between these periods of excess and low demand to reduce this spec giveaway and hence minimize the total reliance on external utilities. The TES assumed here is a simple mixed tank design connected directly to the water stream. By splitting the total flow from the exit of the *WHB*, we can store the excess energy by increasing the temperature in the tank and similarly discharging it during periods of shortfall. The tank is assumed to be completely filled and under perfect level control, and hence the inflow would be equal to outflow. Our aim is to find the optimal size of this tank that needs to be installed, and also assess if any increase to the area in the existing heat exchanger *WHB* would be beneficial.

3.2. Dynamic Process Model

We will first build a simple dynamic model of the process that can be used for the optimization step. The dynamic process model is built according to the process described in the previous section. We will assume the following main assumptions throughout to build the model.

- All fluid streams have constant properties like density (ρ) and specific heat capacity (C_p) which do not change with temperature.
- Heat losses from pipelines or heat exchangers to the surroundings are negligible.
- Liquid holdups in pipes and pressure effects on flow rates are negligible.
- All volume elements are well mixed (without internal temperature gradients), and their exit temperature is equal to the temperature of the volume.
- Perfect level control in TES tank.

The dynamic model can then be built by writing out the mass and energy balances for each of the elements in the process with any additional assumptions being made mentioned therein.

TES Tank

The TES tank is assumed to be well-mixed, and the dynamic mass balance can then be written as

$$\frac{d}{dt}\left(\rho^{dh}V^{tes}\right)=\rho^{dh}(q^{tes,in}-q^{tes,out})$$

Under the main assumptions of constant densities and perfect level control, the mass balance reduces to the inlet volumetric flow rate being equal to the outlet flow rate. This is already accounted for while setting up the variables in the flowsheet and does not need to be added separately. The variable α is used to denote the proportion of the total flow q^{dh} that is being sent to TES.

$$q^{tes,\,in}=\;q^{tes,out}=\;q^{tes}=\alpha q^{dh}$$

The dynamic energy balance for the tank thus can be written as

$$\frac{d}{dt}\left(\rho^{dh}V^{tes}C_p^{dh}T^{tes}\right)=\rho^{dh}\alpha q^{dh}C_p^{dh}(T^{whb}-T^{tes})-Q^{tes,loss}$$

The energy loss from the tank is assumed from the total surface area of the tank to the ambient, and the rate of heat loss being proportional to the temperature difference between the tank and the ambient temperature. Assuming a constant height/ diameter ratio for the construction of tanks for a volume of V^{tes}, the heat loss can then be written as

$$Q^{tes,loss}\propto U^{tes}\left(V^{tes}\right)^{2/3}(T^{tes}-T^{amb})$$

with a proportionality constant *c*. This gives us the dynamics of the temperature in the tank as

$$\frac{dT^{tes}}{dt}=\frac{\alpha q^{dh}(T^{whb}-T^{tes})}{V^{tes}}-\frac{cU^{tes}\left(V^{tes}\right)^{-1/3}\left(T^{tes}-T^{amb}\right)}{\rho^{dh}C_p^{dh}} \qquad (3.1)$$

Peak Heat Boiler (PHB)

The peak heat boiler uses more expensive sources of energy to heat the water. The size of this exchanger is of less interest and is modelled simply as a well-mixed tank of constant volume (V^{phb}) with the additional heat duty being added directly, to capture simple dynamics in the response of this unit. The mass balance, using the earlier assumptions simplifies to inlet and outlet volume flows being equal, and is already accounted for while setting up the variables in the flowsheet. The energy balance equation can be written as

$$\frac{d}{dt}\left(\rho^{dh} V^{phb} C_p^{dh} T^{phb}\right) = \rho^{dh} q^{dh} C_p^{dh} (T^b - T^{phb}) + Q^{phb}$$

Rearranging gives the differential equation for the dynamics of the exit temperature of PHB as

$$\frac{dT^{phb}}{dt} = \frac{q^{dh}(T^b - T^{phb})}{V^{phb}} + \frac{Q^{phb}}{\rho^{dh} V^{phb} C_p^{dh}} \tag{3.2}$$

Node B

Node B is assumed to be an ideal mixer with zero volume. The energy balance gives the temperature at the node as the algebraic equation

$$T^b = \alpha T^{tes} + (1-\alpha) T^{whb} \tag{3.3}$$

Dump Cooler (DC)

We are only interested in the heat duty of this cooler. Hence, it is simply modelled as a point volume with the heat duty removed directly. The energy balance then reduces to an algebraic equation

$$\rho^{wh} q^{wh} C_p^{wh} \left(T^{wh,ret} - T^{dc}\right) = Q^{dc} \tag{3.4}$$

The dump cooler here is used to achieve a specified return temperature to the process. In case the energy is extracted from some flue stream being released to the environment, Q^{dc} could easily just represent the amount of recoverable heat being released to the environment by picking a suitable value for $T^{wh,\,minRet}$, and an economic penalty for wasting this energy.

Waste Heat Boiler (WHB)

We are concerned with finding an optimal area for this heat exchanger and will assume this as a pure countercurrent heat exchanger. The most common way to model the steady-state energy balance for parallel heat exchangers is to use the log mean temperature difference (LMTD) between the temperature differences at the ends of the exchanger (ΔT_1 and ΔT_2) as

$$\Delta T_{LM} = \frac{\Delta T_2 - \Delta T_1}{\ln(\Delta T_2 / \Delta T_1)} = \frac{\Delta T_1 - \Delta T_2}{\ln(\Delta T_1 / \Delta T_2)} \tag{3.5}$$

The heat duty transferred in the heat exchanger is then calculated as,

$$Q = UA\,\Delta T_{LM}$$

Using the LMTD directly in the dynamic energy balance is known to cause issues in optimization due to the highly nonlinear logarithmic term. There are various approximations to the LMTD instead that are popularly for such purposes used in the literature of the form

$$\Delta T_{LM} \approx \Delta T_M = \left[\frac{1}{2}\left(\Delta T_1^n + \Delta T_2^n\right)\right]^{1/n} \tag{3.6}$$

The value of the exponent *n* used in different approximations are as shown in table

Table 3-1: Approximations commonly used for LMTD

n	**Approximation** (ΔT_M)
1	Arithmetic Mean (ΔT_{AM})
$\frac{1}{3}$	Underwood's Mean (ΔT_{UM})
0.3275	Chen's Mean (ΔT_{CM})

This approach was used by Mdoe [22] to model and calculate the optimal control for a different TES system. This approach does not give us an easy way to adjust the model complexity while building and later scaling the design problem and is not considered here. We will instead model the heat exchanger using the popular lumped cell-based approach [23] [24].

In the lumped cell-based heat exchanger model, each stream is modelled as a series of well-mixed tanks referred to as cells. The adjacent cells exchange heat only with each other through the dividing wall. The advantage of this approach is that any flow configuration can be modelled and can better represent the true dynamic behaviour by manipulating the arrangement and using a sufficient number of these cells. The model complexity can be easily controlled by the number of cells used to represent the heat exchanger. The arrangement for a generic countercurrent heat exchanger with the hot stream and cold stream temperatures can then be illustrated as in Figure 3-2.

The convention followed is to number the cells in each stream in the direction of fluid flow (from inlet to outlet) with the total number of cells set as *nCell*. Therefore, cold cell "*i*" is thermally coupled with hot cell numbered "*nCell* + 1 - *i*" and the heat duty being transferred between them denoted as $Q_{(i)}$.

In addition to the previously mentioned assumptions, the following assumptions are used while building the lumped cell-based heat exchanger model.

- The total heat transfer area and volume in each side is uniformly distributed among the cells ($A_{(i)} = \frac{A}{nCell}$ and $V_{(i)} = \frac{V}{nCell}$ for $i=1,2,\ldots,nCell$)
- The overall heat transfer coefficient (U) in each cell is the same and does not change with flowrates
- Wall thermal resistance and capacitance are insignificant.

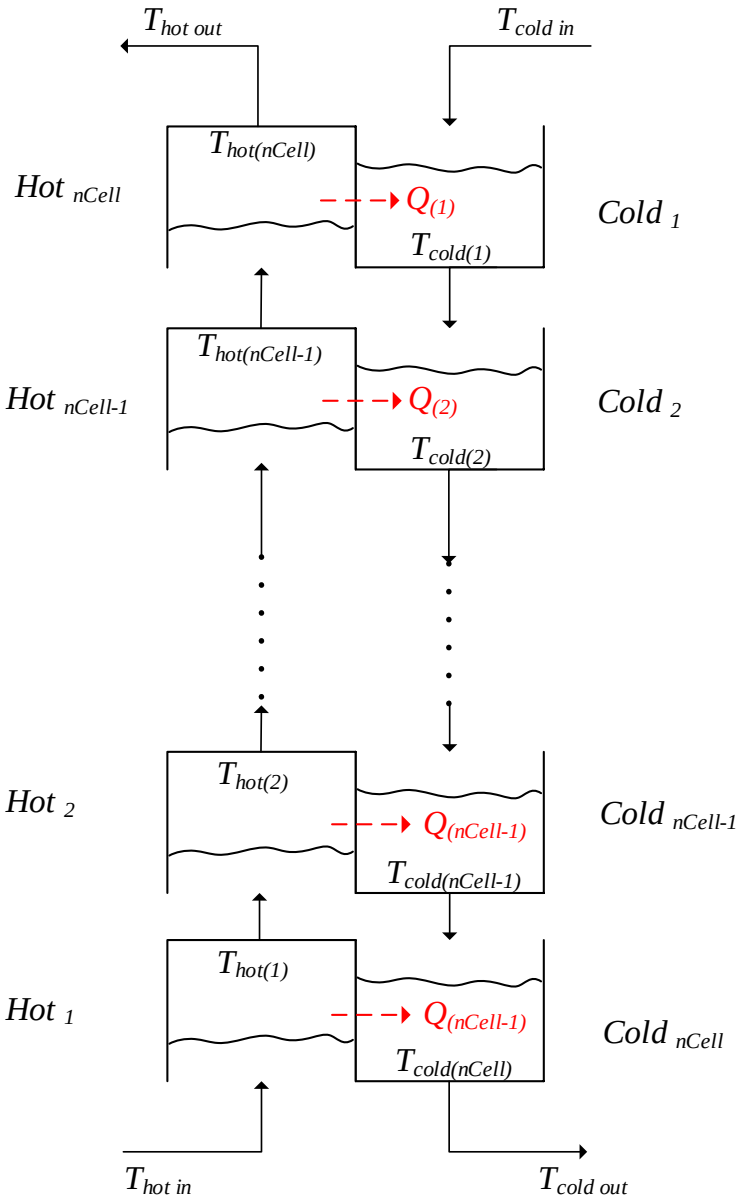

Figure 3-2: Lumped cell-based model of a countercurrent heat exchanger

These assumptions let us write the generic expression for heat transfer into the cold cell "*i*" as

$$Q_{(i)} = (UA)_{(i)}(T_{hot(nCell+1-i)} - T_{cold(i)}) \qquad \forall i \in \{1, 2, ..., nCell\}$$

In modelling the *WHB*, the cold side is the district heating stream, and all the cell temperatures are represented using $T_{(i)}^{whb}$. The hot side is the waste heat stream, and cell temperatures are represented as $T_{(i)}^{wh,ret}$.

The energy balance over the *i-th* cold side cell is

$$\frac{d}{dt}\left(\rho^{dh}V_{(i)}^{whb.dh}C_p^{dh}T_{(i)}^{whb}\right)=\rho^{dh}q^{dh}C_p^{dh}\left(T_{(i-1)}^{whb}-T_{(i)}^{whb}\right)+Q_{(i)}^{whb} \qquad \forall i\in\{1,2,...,nCell\}$$

$$T_0^{whb}=T^{dh,\mathrm{Ret}}$$

The cold cell "*i*" is thermally coupled to the hot cell "*nCell +1- i*" according to the numbering scheme we have followed for the countercurrent heat exchanger. The energy balance in the coupled hot cell can be written as,

$$\frac{d}{dt}\left(\rho^{wh}V_{(nCell+1-i)}^{whb.wh}C_p^{wh}T_{(nCell+1-i)}^{wh,ret}\right)=\rho^{wh}q^{wh}C_p^{wh}\left(T_{(nCell-i)}^{wh,ret}-T_{(nCell+1-i)}^{wh,ret}\right)-Q_{(i)}^{whb} \qquad \forall i\in\{1,2,...,nCell\}$$

$$T_0^{wh,ret}=T^{wh,\mathrm{sup}}$$

The heat duty being transferred between the cells is denoted as

$$Q_{(i)}=(UA)_{(i)}^{whb}\,(T_{(nCell+1-i)}^{whb}-T_{(i)}^{wh,ret}) \qquad \forall i\in\{1,2,...,nCell\}$$

Rearranging and stacking them in standard notation gives us the differential equations for the temperature response in the *WHB* as

$$\frac{d}{dt}\left(T_{(i)}^{whb}\right)=\frac{q^{dh}\left(T_{(i-1)}^{whb}-T_{(i)}^{whb}\right)}{V_{(i)}^{whb.dh}}+\frac{Q_{(i)}^{whb}}{\rho^{dh}C_p^{dh}V_{(i)}^{whb,dh}} \qquad \forall i\in\{1,2,...,nCell\} \qquad (3.7)$$

$$T_0^{whb}=T^{dh,\mathrm{Ret}}$$

$$\frac{d}{dt}\left(T_{(i)}^{wh,ret}\right)=\frac{q^{wh}\left(T_{(i-1)}^{wh,ret}-T_{(i)}^{wh.ret}\right)}{V_{(i)}^{whb.wh}}+\frac{Q_{(nCell+1-i)}^{whb}}{\rho^{wh}C_p^{wh}V_{(i)}^{whb,wh}} \qquad \forall i\in\{1,2,...,nCell\} \qquad (3.8)$$

$$T_0^{wh,ret}=T^{wh,\mathrm{sup}}$$

3.3. Model as DAE

The complete dynamic behaviour of the system can be thus expressed as a set of differential-algebraic equations (DAE) of the form

$$\dot{x}^{diff} = f\left(x^{diff}, x^{alg}, u, p\right) \tag{3.9a}$$

$$0 = g\left(x^{diff}, x^{alg}, u, p\right) \tag{3.9b}$$

The differential equations (3.1), (3.2), (3.7) and (3.8) are collected into the form of the equation (3.9a) and the algebraic equations (3.3) and (3.4) are collected into the form of the equation (3.9b).

The differential state x^{diff} vector (of length *2*nCell + 2*) is then defined as,

$$x^{diff} = \left[T^{tes}, T^{phb}, T^{whb}_{i=1,\ldots,nCell}, T^{wh,ret}_{i=1,\ldots,nCell}\right]$$

the algebraic state vector x^{alg} as,

$$x^{alg} = \left[T^{b}, T^{dc}\right]$$

the manipulated variable vector u as,

$$u = \left[\alpha, Q^{phb}, Q^{dump}\right]$$

the time-varying parameter vector p as,

$$p = \left[q^{dh}, q^{wh}, T^{dh,Ret}, T^{wh,Sup}\right]$$

The various time-invariant parameters that are used in the simulations are specified in Table 3-2.

Table 3-2: Parameter values used in simulations

Parameter	Description	Value	Units
ρ^{dh}	Density of district heating stream	1000	kg/m^3

ρ^{wh}	Density of waste heat stream	800	kg/m^3
C_p^{dh}	Specific Heat Capacity of district heating stream	4.18	$kJ/kg.K$
C_p^{wh}	Specific Heat Capacity of waste heat stream	3.5	$kJ/kg.K$
U^{whb}	Overall heat transfer coefficient for waste heat boiler	5.13	kW/m^2K
U^{tes}	Overall heat loss coefficient for TES tank	0.0001	kW/m^2K
V^{phb}	Volume of Peak Heat Boiler	12	m^3
$V^{whb,\ dh}$	Volume of Waste Heat Boiler (district heating side)	12	m^3
$V^{whb,\ wh}$	Volume of Waste Heat Boiler (waste heat side)	12	m^3
$nCell$	Number of cells used in modelling WHB	5	-
T^{amb}	Ambient Temperature	10	$^{\circ}C$

Some nominal values for the varying elements of the model are specified in Table 3-3. These values are also used as the initial state for the differential states for all the simulations.

Table 3-3: Nominal values for Variables and Inputs

Variable	Description	Nominal Value	Units
T^{tes}	Temperature of TES tank	45	$^{\circ}C$
T^{phb}	Temperature at the exit of Peak Heat Boiler	56	$^{\circ}C$
T^{whb}	WHB exit Temperature of district heating stream	46	$^{\circ}C$
$T^{wh,ret}$	WHB exit temperature for waste heat stream	71	$^{\circ}C$
T^{b}	Temperature of Node B	55	$^{\circ}C$
T^{dc}	Dump cooler exit temperature	30	$^{\circ}C$
α	Split ratio of WHB exit stream sent to TES	0	-
Q^{phb}	Heat duty added to PHB	0	kW
Q^{dc}	Heat duty removed from Dump Cooler	0	kW
V^{tes}	Volume of TES tank	15000	m^3
A^{whb}	Area of WHB heat exchanger	400	m^2
q^{dh}	Volumetric flow rate of district heating stream	900	$m^3/hr.$
q^{wh}	Volumetric flow rate of Waste Heat stream	900	$m^3/hr.$
$T^{dh,\ minSup}$	Minimum supply temperature of water to consumers	55	$^{\circ}C$
$T^{wh,\ minRet}$	Minimum return temperature of waste heat stream	30	$^{\circ}C$
$T^{dh,\ Ret}$	Supply temperature of district heating stream	30	$^{\circ}C$
$T^{wh,\ Sup}$	Supply temperature of waste heat stream	100	$^{\circ}C$
C^{Qphb}	Price per unit of energy added	0.06	USD/kWh
C^{Qdc}	Price per unit of energy removed	0.006	USD/kWh

3.4. The Optimal Operations Problem

In this section, we illustrate the optimal operation of the TES system. The area of the heat exchanger and the volume of the tank are kept fixed as in Table 3-3. A simple step change is in the flowrate q^{wh} to represent a step-change in the supply of energy (while all the other parameters are kept constant for simplicity of discussion) to illustrate the behaviour of the system under optimal operation. We will then show how the design parameters (volume of tank and area of heat exchanger) affect this optimal system behaviour.

During operations, only the costs associated with the purchase of external utilities are considered (in our case Q^{phb} and Q^{dc}) where C^{Qphb} and C^{Qdc} are the unit prices associated with these external utilities. Hence for an operating period from t_0 to t_f, the corresponding operating cost function (C^{oper}) can be defined as

$$C^{oper} := \int_{t_0}^{t_f} \left(C^{Qphb}(t) Q^{phb}(t) + C^{Qdump}(t) Q^{dump}(t) \right) dt \tag{3.10}$$

The optimal operations problem is to find the profile for the input variables that minimizes the operating cost for the period. This can be formulated as the optimization problem in continuous time as

$$\min_{x^{diff}(t), x^{alg}(t), u(t)} \quad \int_{t_0}^{t_f} \left(C^{Qphb}(t) Q^{phb}(t) + C^{Qdump}(t) Q^{dump}(t) \right) dt \tag{3.11a}$$

$$s.t. \quad \dot{x}^{diff}(t) = f\left(x^{diff}(t), x^{alg}(t), u(t), p(t) \right) \qquad t_0 \le t \le t_f \tag{3.11b}$$

$$0 = g\left(x^{diff}(t), x^{alg}(t), u(t), p(t) \right) \qquad t_0 \le t \le t_f \tag{3.11c}$$

$$x^{diff}(0) = \hat{x}_0 \tag{3.11d}$$

$$lbx^{diff} \le x^{diff}(t) \le ubx^{diff} \qquad t_0 \le t \le t_f \tag{3.11e}$$

$$lbx^{alg} \le x^{alg}(t) \le ubx^{alg} \qquad t_0 \le t \le t_f \tag{3.11f}$$

$$lbu \le u(t) \le ubu \qquad t_0 \le t \le t_f \tag{3.11g}$$

The initial condition for the differential states is specified as $\hat{x}_0$. The bounds for the states and inputs are also included in the problem (Equations 3.11e to 3.11g). This problem in infinite-

dimensional and can be solved by first converting it into a finite-dimensional nonlinear programming problem (NLP), dividing it into *N* equally spaced sampling intervals (*n = 0,1,…,N-1*). This discretization can be performed using various approaches such as single shooting, multiple shooting, direct collocation [25]. We will use direct collocation for all our problems in this thesis. After discretization, the optimal operations problem can be posed in the standard NLP form as

$$\min_{x_{n+1}^{diff},x_n^{alg},u_n} \quad \sum_{n=0}^{N-1}\left(C_n^{Qphb}Q_n^{phb}+C_n^{Qdump}Q_n^{dump}\right) \tag{3.12a}$$

$$s.t. \quad x_{n+1}^{diff} = f\left(x_n^{diff},x_n^{alg},u_n,p_n\right) \qquad n=0,1,\ldots,N-1 \tag{3.12b}$$

$$0 = g\left(x_n^{diff},x_n^{alg},u_n,p_n\right) \qquad n=0,1,\ldots,N-1 \tag{3.12c}$$

$$x_0^{diff} = \hat{x}_0 \tag{3.12d}$$

$$lbx^{diff} \le x_n^{diff} \le ubx^{diff} \qquad n=0,1,\ldots,N-1 \tag{3.12e}$$

$$lbx^{alg} \le x_n^{alg} \le ubx^{alg} \qquad n=0,1,\ldots,N-1 \tag{3.12f}$$

$$lbu \le u_n \le ubu \qquad n=0,1,\ldots,N-1 \tag{3.12g}$$

In compact notation, the optimal operations problem can be represented as

$$\min_{x^{oper}} \quad C^{oper}\left(x^{oper}\right) \tag{3.13a}$$

$$s.t. \quad \left(x^{oper}\right) \in \chi^{oper} \tag{3.13b}$$

where all the variables associated with the operation phase are collected into the vector $x^{oper} = \left[x_0^{diff}, x_{n+1}^{diff}, x_n^{alg}, u_n\right]$ for all $n = 0,1,...,N-1$. The operating cost function is defined as $C^{oper}\left(x^{oper}\right) = \sum_{n=0}^{N-1} C_n^{Qphb}Q_n^{phb} + C_n^{Qdump}Q_n^{dump}$ and the feasible set χ^{oper} contains all the feasible points of the constraints (3.12b to 3.12g).

All the illustrative examples in this thesis are implemented in MATLAB R2019b using CasADi (v3.5.1) [26] framework for formulating the mathematical models and the NLP. These problems are then solved using IPOPT [27] to find a local solution.

An Illustrative Example of Optimal Operation

To analyze the solution of (3.12) a simple problem is solved where a step-change in flowrate of q^{wh} is provided. A prediction horizon of 30 hours is considered, and the step change is shown in the top subplot of Figure 3-3. The first 15 hours have a 20% higher flowrate than the nominal flow, and for the next 15 hours, it is 20% lower than nominal. All other time-varying parameters are kept constant and the nominal values are chosen to satisfy the consumer demand without the use of heating in the *PHB*. It is very easy to analyze the solution of the optimal control problem. The exit temperature response of the *WHB* is shown in the bottom subplot of Figure 3-3. When there is an increased flow in the first half, the *WHB* exit temperature increases (in this particular case to above the return specification of $55^{\circ}C$) and vice versa in the latter half.

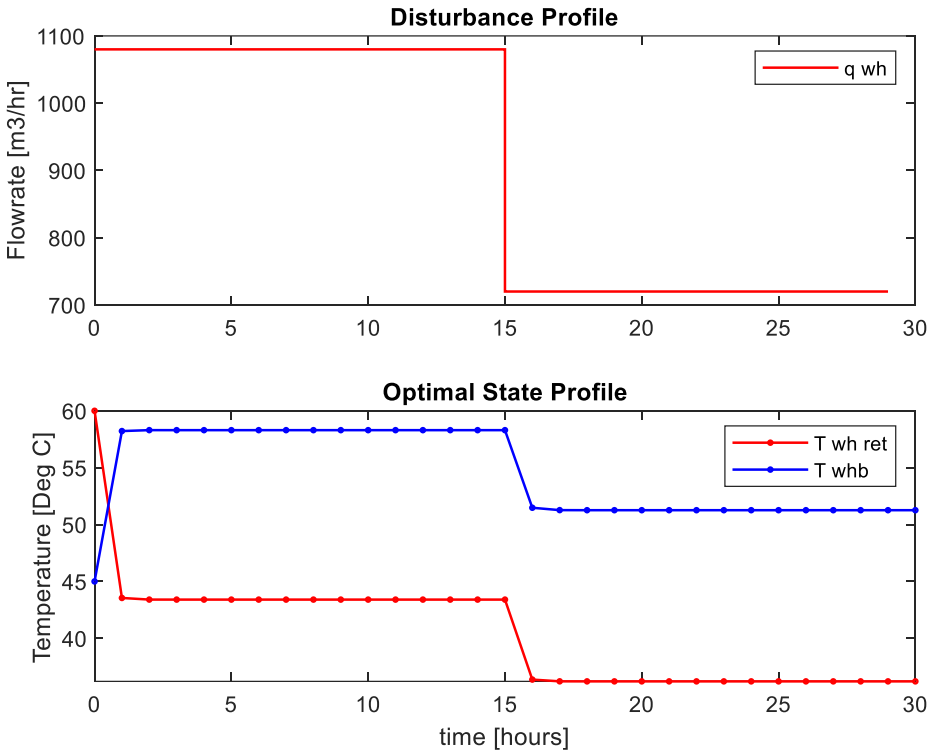

Figure 3-3: Step change in disturbance and response in *WHB* exit temperatures

Optimal operation in this case is rather intuitive - try and store the excess energy in the first half and then release it during the period of shortfall. This is the optimal solution from solving the NLP (3.13). In the bottom subplot of Figure 3-4, the split ratio (α) is being manipulated to charge the TES tank as can be seen by the increase in temperature T^{tes}. If TES temperature is above the minimum return temperature ($55^{\circ}C$), we cannot avoid sending hot water above the minimum

return specification and α is saturated fully in an attempt to maximize the energy stored in the TES. There is no need for Q^{phb} during this time. When the step-change causes a fall in the supply (by hour 15), the *WHB* exit temperature quickly falls below $55^{\circ}C$. Since the TES tank has stored energy, α is manipulated to now discharge the tank. Q^{phb} is required to maintain the temperature T^{phb} at its lower limit when the TES temperature falls below $55^{\circ}C$ around hour 23.

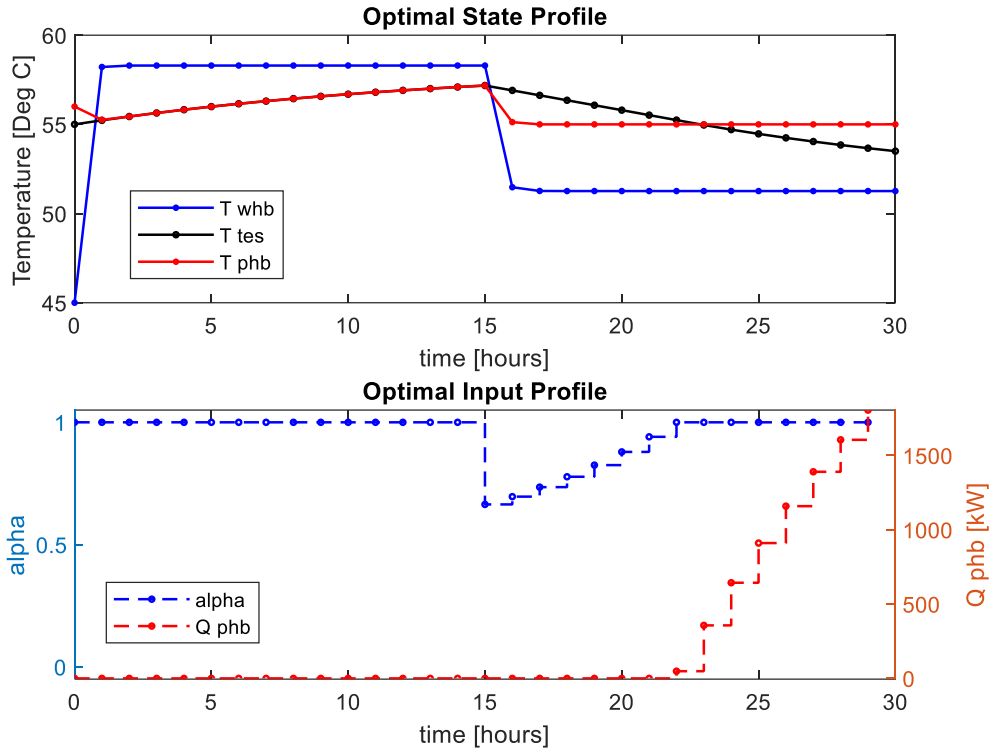

Figure 3-4: Optimal operation profiles

This simple example is thus able to show the crucial nonlinear behaviour associated with the TES system. Even though cumulatively the supply and demand for energy were the same as for the nominal case, we do end up using peak heating to satisfy the demand profile in this simple example. This is because of the importance of temperature (the quality of energy) in real TES systems. Analyzing the system simply as duties would not have considered the second law of thermodynamics which is essential while trying to find real-life equivalent parameters for designing and operating such systems.

The primary factors affecting the quality of energy in this case are the volume of the tank and the area of the heat exchanger. The area of the heat exchanger directly influences the temperature T^{whb}

and hence the maximum temperature that the TES tank can be charged up to. The volume of the tank affects the amount of heat duty that is stored at a particular temperature. Both these parameters have an important effect on the operational cost, which is analyzed in the next section.

3.5. The effect of Design on Optimal Operations

Next, we look at the effect design variables (volume of tank and area of heat exchanger in our case) have on optimal operation under the same step-change in q^{wh} represented in Figure 3-3.

Volume of the Tank

We redo the analysis with different tank sizes (with a constant nominal heat exchanger area) - one lower (10,000 m^3), one larger (20,000 m^3), and compare it to the nominal volume (15,000 m^3) of the tank that was considered earlier. Since the heat exchanger area is the same - the exit temperature of *WHB* will follow the same profile as in the bottom subplot of Figure 3-3 from before. The optimal profile for α is found by solving the OCP to first charge and later discharge the tank. The larger tank holds more charge at the same temperature than the smaller ones. we can see from the top subplot of Figure 3-5 as it takes longer for this temperature to approach T^{whb}. This, in turn, requires less Q^{phb} during the discharging phase as more energy could be dispatched from the larger TES tank.

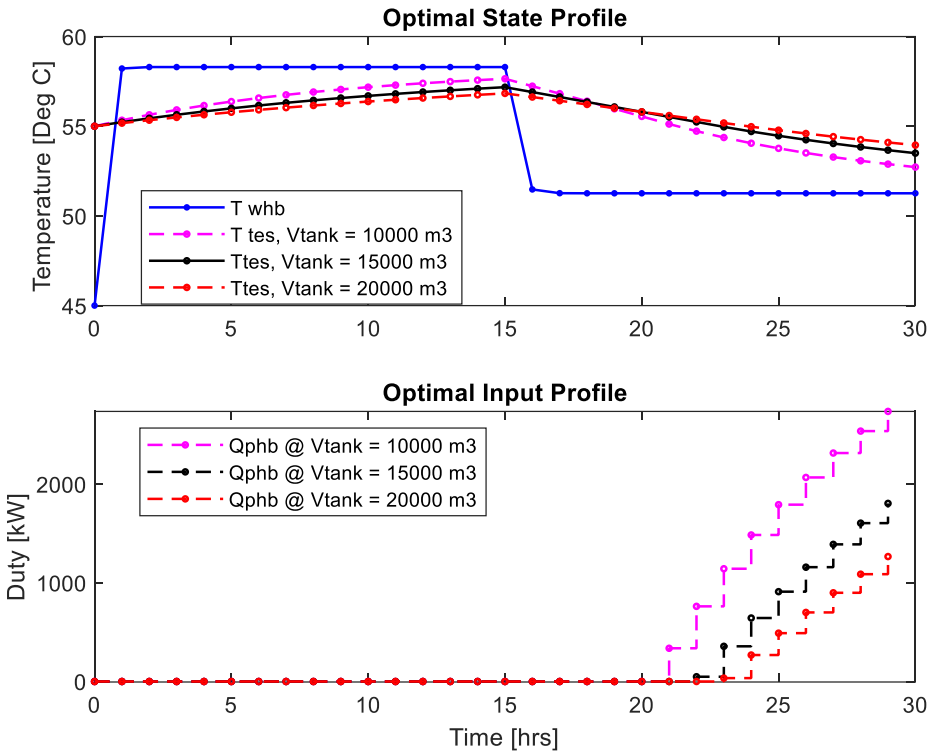

Figure 3-5: Optimal operation at different tank sizes

Area of the Heat Exchanger

Similar to the tank volume, we compare the optimal operation solution under 3 cases of heat exchanger area around the nominal value of 400 m^2 (and the tank volume held constant at the nominal value). From the top subplot in Figure 3-6, we see that increasing the area results in extracting more of the available energy from the supply stream. This means that T^{whb} temperatures will be higher, the TES can be potentially be charged to a higher temperature and thus store more energy for the same tank volume. This effect can be seen in the bottom subplot of Figure 3-6, as less Q^{phb} is required with an increase in the heat exchanger area for the same tank volume.

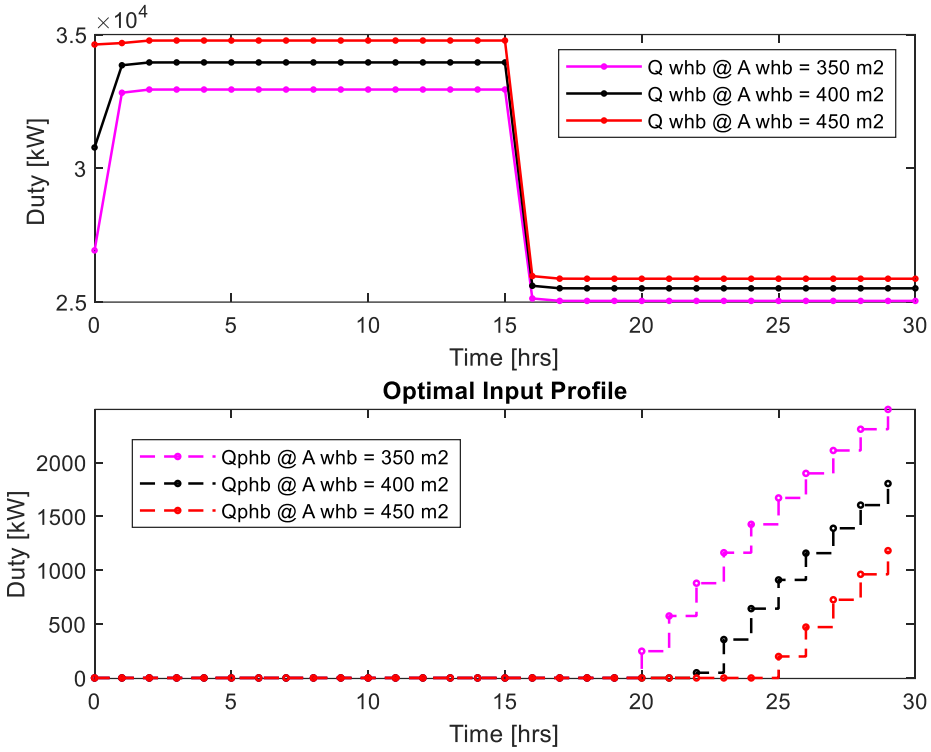

Figure 3-6: Optimal operation at different heat exchanger areas

We can see that these simple sizing parameters chosen during the design phase of the project have an important influence on the operation cost of the dynamic process in the future. We would hence like to find how these sizing parameters, for such a dynamic process with uncertain demand or supply profiles, can be chosen by the designer. This question is explored in the coming chapters of this thesis.

Chapter 4. Optimal Design Problem

In the previous chapter, we introduced the optimal control problem for the TES plant and saw the effect of the original design variables (like heat exchanger area and tank volume) on the optimal operation. This chapter will discuss how these design parameters are chosen.

During process design, one seeks to choose the process and equipment parameters that will maximize the net present value (*NPV*) of the project. For a fixed production capacity of the plant, this reduces to minimizing the total cost of ownership for the plant. We ignore the present value aspects associated with actual project evaluation to make it easier for the analysis of our results and discussions. The costs associated with the plant can be broadly split into the initial capital cost and the operations costs. In our case, the initial capital cost is restricted to consist only the purchased equipment cost for the heat exchanger and the TES tank. These can be estimated using simple cost relationship of the form $a + bS^c$ where S is the characteristic size of the equipment (area in m^2 for heat exchangers and volume in m^3 for tanks). The constants are taken from Sinnott [28] and are shown in Table 4-1.

Table 4-1: Constants used for estimating design cost

Constants	Storage Tank: (cone roof)	Heat Exchanger: (shell and tube)
a	5700	10000
b	700	88
c	0.7	1

The design life of 20 years is considered for the plant. To compare the objective values between the different cases easily, the total capital cost is scaled down to the associated prediction horizon for the optimization problem. It is represented as (ignoring the constants not influencing the optimization).

$$C^{capital} = C^{tank}\left(V^{tes}\right)^{0.7} + C^{hex}\left(A^{whb}\right)^{1} \tag{4.1}$$

The constants C^{tank} and C^{hex} are scaled-down corresponding to the considered prediction horizon for the optimization problem (4.2). When the prediction horizon of 20 years will be considered for the optimization problem, C^{tank} and C^{hex} will be equal to the constants b in the table above.

4.1. Optimal Design – Without Uncertainty

Consider the case where the supply and demand of energy is known perfectly for the prediction horizon *N*. Then the optimal design problem can be simply formulated by extending the optimal control problem from section 3.4 to include the contribution of capital cost in the objective. The design parameters (V^{tes}, A^{whb}) are also now variables for the optimization with associated bounds. The design problem is formulated as the NLP

$$\min_{V^{tes},A^{whb},u_n} \quad C^{\tan k}\left(V^{tes}\right)^{0.7} + C^{HEx}\left(A^{whb}\right) + \sum_{n=0}^{N-1}\left(C_n^{Qphb}Q_n^{phb} + C_n^{Qdump}Q_n^{dump}\right) \tag{4.2a}$$

$$s.t. \quad x_{n+1}^{diff} = f\left(x_n^{diff}, x_n^{a\lg}, u_n, p_n\right) \qquad n=0,1,\ldots,N\text{-}1 \tag{4.2b}$$

$$0 = g\left(x_n^{diff}, x_n^{a\lg}, u_n, p_n\right) \qquad n=0,1,\ldots,N\text{-}1 \tag{4.2c}$$

$$x_0^{diff} = \hat{x}_0 \tag{4.2d}$$

$$lbx^{diff} \le x_n^{diff} \le ubx^{diff} \qquad n=0,1,\ldots,N\text{-}1 \tag{4.2e}$$

$$lbx^{a\lg} \le x_n^{a\lg} \le ubx^{a\lg} \qquad n=0,1,\ldots,N\text{-}1 \tag{4.2f}$$

$$lbu \le u_n \le ubu \qquad n=0,1,\ldots,N\text{-}1 \tag{4.2g}$$

$$lbV^{tes} \le V^{tes} \le ubV^{tes} \tag{4.2h}$$

$$lbA^{whb} \le A^{whb} \le ubA^{whb} \tag{4.2i}$$

For the sake of compact notation, we represent the design problem as

$$\min_{x^{des},x^{oper}} \quad C^{capital}(x^{des}) + C^{oper}\left(x^{des}, x^{oper}\right) \tag{4.3a}$$

$$s.t. \quad \left(x^{des}, x^{oper}\right) \in \chi \tag{4.3b}$$

where the variables associated with design into the vector are collected into $x^{des} = \left[V^{tes}, A^{whb}\right]$. The operation phase variable vector x^{oper} is defined as before. The capital cost depends on the design variables, and the capital cost function is defined as $C^{capital}(x^{des}) = C^{\tan k}\left(V^{tes}\right)^{0.7} + C^{hex}\left(A^{whb}\right)$. The operating cost now depends on both the vectors x^{des} and x^{oper}, and the operating cost function is

defined as $C^{oper}\left(x^{des}, x^{oper}\right) = \sum_{n=0}^{N-1} C_n^{Qphb} Q_n^{phb} + C_n^{Qdump} Q_n^{dump}$. All the feasible points to the constraints (4.2b to 4.2i) are represented using the feasible set χ.

Illustrative example of optimal design – without uncertainty

We illustrate the design problem with a prediction horizon of 30 hours. The supply side profile is given as in the top subplot of Figure 4-1, while all other parameters are held constant at their nominal values. The main results from the solution for the NLP are given in Table 4-2 and the associated optimal input represented in the bottomsubplot of Figure 4-1. An important point to note here is that NLP was solved using IPOPT, which is a local optimizer; thus the solutions we obtain are local minimizers.

Table 4-2: Optimal design for the deterministic case

Design parameters in	Base design	Optimal design	Units
Tank volume (V^{tes})	15000	11420.55	m^3
Heat exchanger area (A^{whb})	300	464.70	m^2
$C^{capital}$	867.9	846.5	*USD*
C^{oper}	170.5	63.6	*USD*
Total Cost	1038.4	910.1	*USD*

We can see that compared to the base design, it is beneficial to increase the heat exchanger area and reduce the tank volume from the nominal values. The trade-off between the capital cost and the operating cost leads to a lower total cost for the period of 1 day being considered.

Comparing the state profiles between the base and optimal design in Figure 4-2, we see that the increased heat exchanger area increases the T^{whb} temperature and thus the TES is able to store more energy even for the lower tank volume as can be seen by the reduced reliance on Q^{phb}. Any further increase in the design parameters to store more energy would just cost more than relying on the external utilities to satisfy the energy demand.

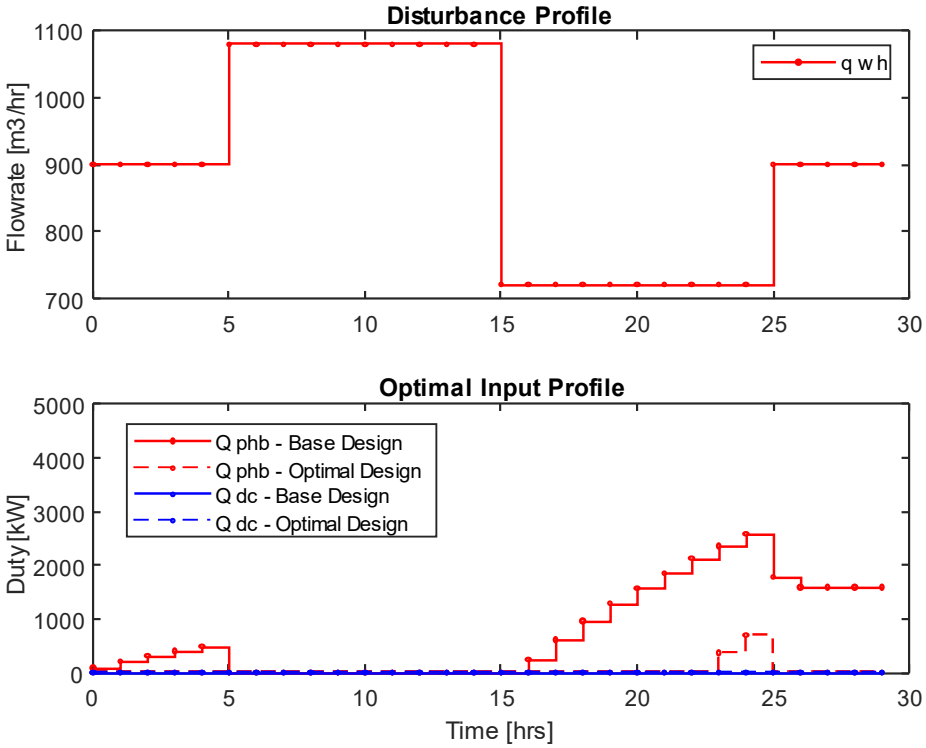

Figure 4-1:Disturbance profile and optimal input profile (base design vs optimal design)

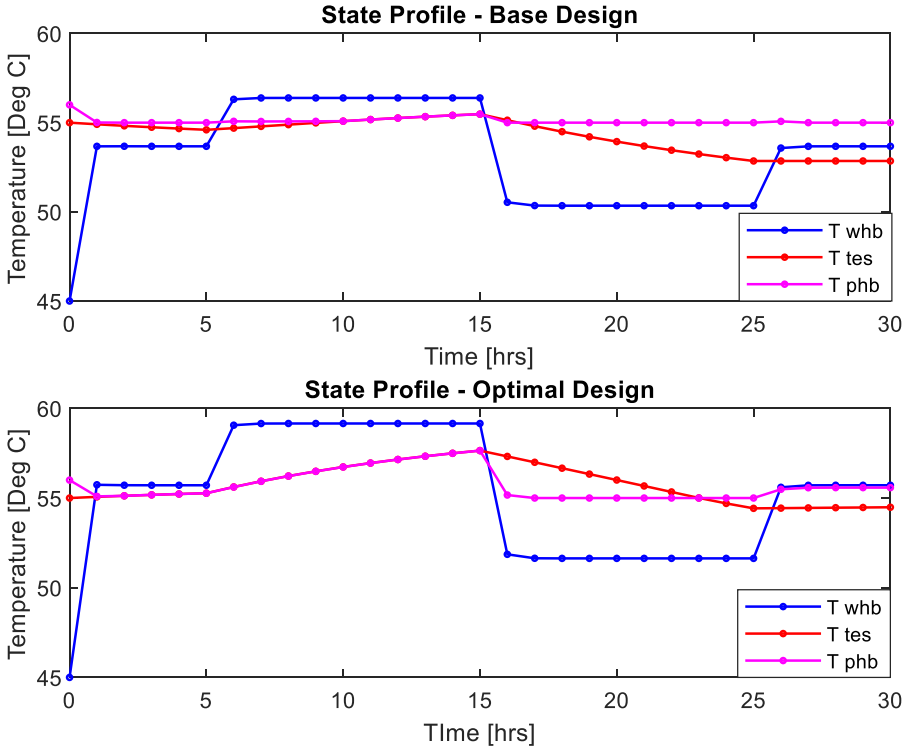

Figure 4-2:State response (base design vs optimal design)

4.2. Optimal Design under Uncertainty

In reality, there is uncertainty associated with the demand and supply profiles of energy and has to be accounted for in the design stage. We can represent each unique combination of supply and demand profiles across the prediction horizon using different representative scenarios. Let the set $\mathcal{S} := \{1, 2, ..., S\}$ denote all such representative scenarios the system is expected to face during operation. Historical operating data from the district heating system could be used to build these scenarios. An example of using historical data for scenario generation in a multistage NMPC is shown by Thombre et al. [6]. Since scenario selection is not our primary focus here, we assume these scenarios are given to us with associated probabilities represented as ω_s.

There are various ways to include this uncertainty information during the design stage itself. One could potentially design the system to be optimal for any of the given scenarios. A simple approach for process design would involve designing the plant for one particular scenario and checking its sensitivity against other scenarios to and make modifications if needed. A more systematic approach would be to optimize the design for a choice of risk measure which accounts all the scenarios. Such approaches could include minimizing the total cost for the worst-case scenario, the expected value of the total cost under all the scenarios, or any other metric we wish to choose.

A simple stochastic optimization framework is used to describe the design problem in our case where the optimal design is defined as the one that minimizes the expected value of the total cost. The optimal design problem can thus be framed as a two-stage nonlinear stochastic problem with full recourse with the design decisions as the first stage variables and the operating decisions as the second stage recourse variables. Formulating it as a two-stage program means that once the design decisions are made, the disturbance is fully revealed, and the optimal control actions are then taken. This is a simplifying assumption and could be relaxed further by assuming multiple stages to represent the sequence of decisions that occur in the operations stage, reacting to uncertainties that might be revealed over time. We do not consider this case and stick with the simple two-stage formulation.

The design problem can be represented in the compact form as

$$\min_{x^{des},x_s^{oper}} \quad C^{capital}(x^{des}) + \sum_{s\in\mathcal{S}} \omega_s C_s^{oper}(x^{des}, x_s^{oper}) \tag{4.4.a}$$

$$s.t. \quad (x^{des}, x_s^{oper}) \in \chi \qquad \forall s \in \mathcal{S} \tag{4.4.b}$$

where each scenario *s* has its own second stage variables x_s^{oper}. The operating cost associated with the scenario is calculated as $C_s^{oper}(x^{des}, x_s^{oper}) = \sum_{n=0}^{N_s-1} C_{n,s}^{Qphb} Q_{n,s}^{phb} + C_{n,s}^{Qdump} Q_{n,s}^{dump}$.

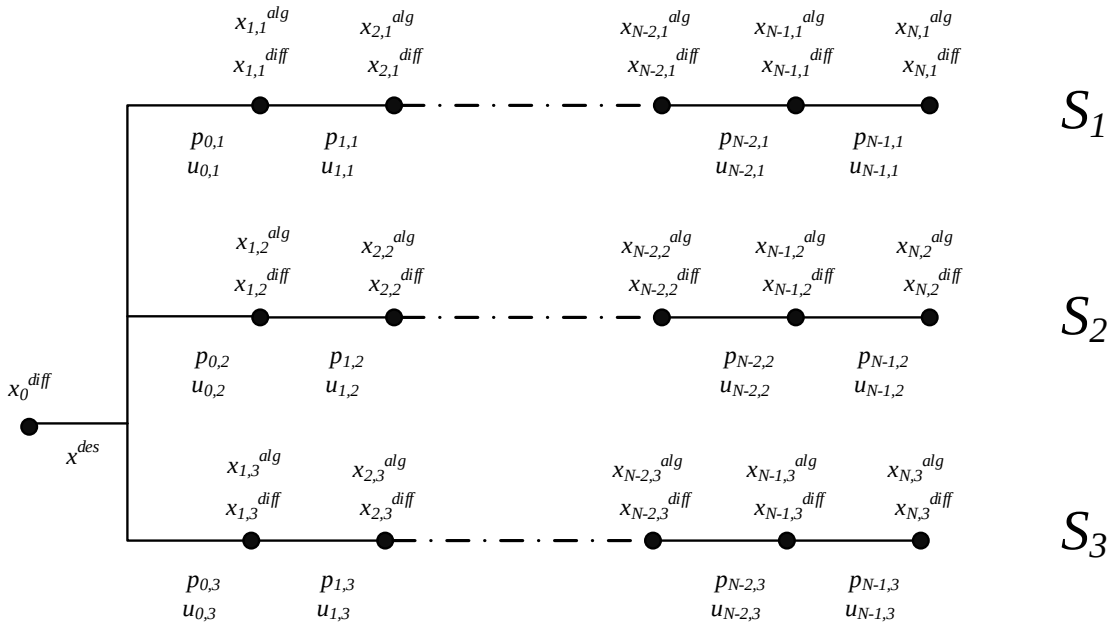

Figure 4-3: Schematic representation of two-stage stochastic design problem with 3 scenarios

A simple schematic of the two-stage design problem with 3 scenarios is represented in Figure 4-3. There are 3 possible scenarios that could be realized, represented using the parameters $p_{n,s}$. The design decision is made in the first stage, after which the uncertainty is completely revealed. Once the uncertainty is revealed, the control actions ($u_{n,s}$) and the corresponding state profiles ($x_{n,s}^{diff}$ and $x_{n,s}^{alg}$) are decided to minimize the operating cost for the realized scenario.

Illustrative example of optimal design – under uncertainty

We illustrate the design problem under uncertainty where two scenarios are considered for the supply side profile given as in Figure 4-4, while all other parameters are held at their nominal values. Both scenarios are considered equally likely, where scenario 2 can be considered an extreme operation of the TES as compared to scenario 1.

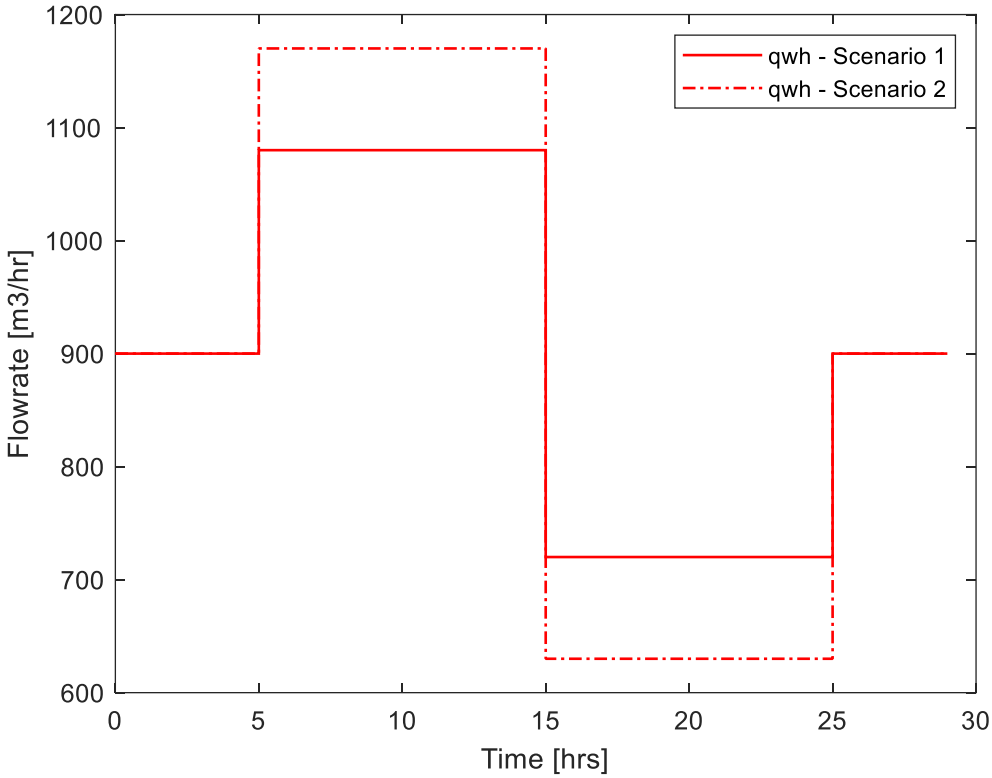

Figure 4-4: Scenarios used in the illustrative example

With perfect information about the profile that would be realized during operation, we would have designed the system to be optimal for that particular scenario as in the previous section. The optimal designs for the individual scenarios are shown in Table 4-3.

Table 4-3: Optimal design with perfect information

Optimal solution based on the individual scenario	Scenario 1	Scenario 2	Units
Tank volume (V^{tes})	11420.56	20633.38	m^3
Heat exchanger area (A^{whb})	464.70	368.79	m^2
$C^{capital}$	846.54	958.39	*USD*
C^{oper}	63.59	441.57	*USD*
Total Cost	910.13	1399.96	*USD*

But in reality, we do not know which of the two scenarios would be realized during operation at the design stage. Hence a stochastic model is used to decide the optimal design for the TES. The total costs associated with this design when either of the scenarios are realized is shown in Table 4-4. The design from the stochastic model would have a higher cost than if we had the optimal design for each scenario. Optimal design from the stochastic model is the one that minimizes the expected total cost, and thus a decision that hedges against all scenarios. This example illustrates that it is impossible, under uncertainty to find a solution that is optimal under all scenarios.

Table 4-4: Optimal design with the stochastic model

The optimal solution for the stochastic model	**Scenario 1**	**Scenario 2**	**Units**
Tank volume (V^{tes})	19013.51		m^3
Heat exchanger Area (A^{whb})	368.79		m^2
$C^{capital}$	924.86		*USD*
C^{oper}	341.55	479.30	*USD*
Total Cost	1266.41	1404.16	*USD*
Expected Total Cost	1335.29		*USD*

4.3. Growth in Problem size and the need for Decomposition

We had considered a simple case in the previous section with a prediction horizon of 1 day and only two scenarios. For a more realistic TES design problem, we would need the prediction horizon to cover multiple charge/ discharge cycles. This would require the prediction horizon to cover much longer periods depending on the type of TES system being considered (for example - months in the case of seasonal energy storage). Since each unique combination of the time-varying parameters represents a scenario, we would also have to consider many scenarios to represent the uncertainty.

This can lead to our two-stage NLP becoming very large, and are thus concerned with our ability to solve the problem with the limited available memory at our disposal. Since the design problem is not used for any real-time application, we are less concerned with the time to solve the problem, but rather the memory required for solving it. We thus look for iterative schemes which can solve this large NLP with limited usage of memory. The aim is to use the structure of the problem to solve it in a distributed matter to arrive at the optimal solution for the original problem.

Chapter 5. Applying ADMM to scenario decomposition

In this chapter, we will see how ADMM can be used for scenario decomposition in a two-stage NLP. We will use a simplified model of the TES system to let us focus more on the decomposition algorithm and its results. The dynamics associated with the heat exchangers are ignored and focus only on the TES tank sizing as the design problem. The disturbance is simply represented as the heat duty transferred from the supplier to the TES system through the waste heat boiler as Q^{whb}. The simplified system is represented in Figure 5-1, and the model equations are then

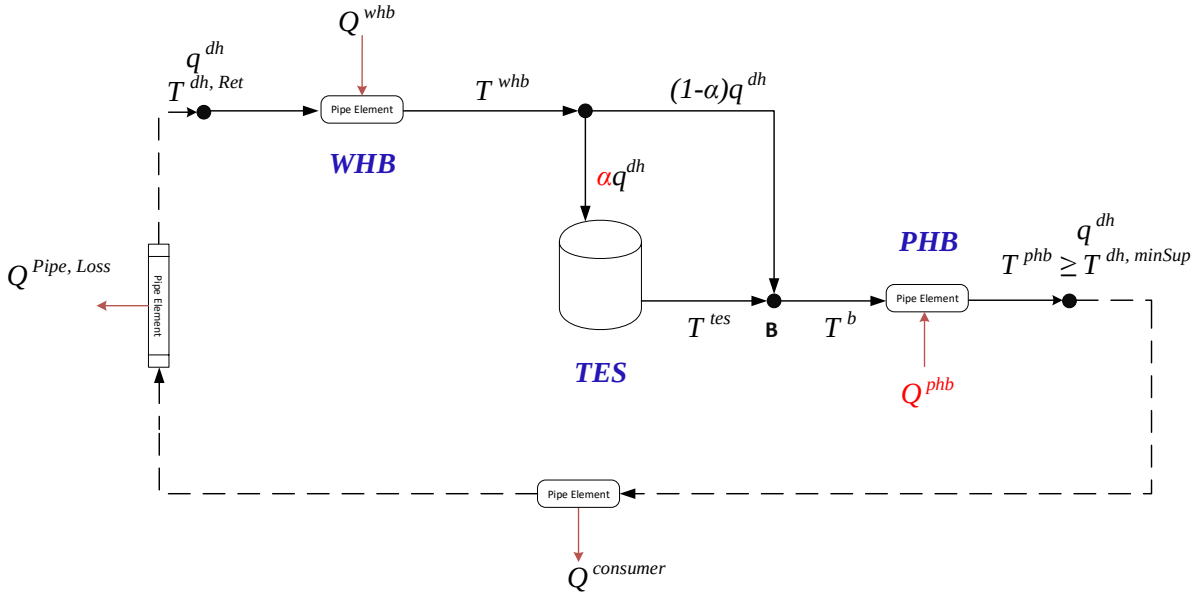

Figure 5-1: Simplified flowsheet of the TES system

the differential equation

$$\frac{d}{dt}\left(T^{tes}\right)=\frac{\alpha q^{dh}\left(T^{whb}-T^{tes}\right)}{V^{tes}}-U^{tes}\left(V^{tes}\right)^{-1/3}\left(T^{tes}-T^{amb}\right) \tag{5.1}$$

and the algebraic equations

$$T^{b}=\alpha T^{tes}+(1-\alpha)T^{whb} \tag{5.2}$$

$$T^{phb}=T^{b}+\frac{Q^{phb}}{q^{dh}\rho^{dh}C_{p}^{dh}} \tag{5.3}$$

$$T^{whb}=T^{dh,\mathrm{Ret}}+\frac{Q^{whb}}{q^{dh}\rho^{dh}C_{p}^{dh}} \tag{5.4}$$

The dynamic behaviour of the system can thus be expressed in the standard DAE form

$$\dot{x}^{diff} = f\left(x^{diff}, x^{a\lg}, u, p\right) \tag{5.5a}$$

$$0 = g\left(x^{diff}, x^{a\lg}, u, p\right) \tag{3.9b}$$

with the differential state vector $x^{diff} = \left[T^{tes}\right]$, the algebraic state vector $x^{a\lg} = \left[T^{b}, T^{phb}, T^{whb}\right]$. The manipulated variable vector is $u = \left[\alpha, Q^{phb}\right]$, and the time-varying parameter vector $p = \left[q^{dh}, Q^{whb}, T^{dh,Ret}\right]$. The variables associated with the design phase is represented as $x^{des} = \left[V^{tes}\right]$. For a prediction horizon of N - the variables associated with the operation phase is then collected as $x^{oper} = \left[x_0^{diff}, x_{n+1}^{diff}, x_n^{a\lg}, u_n\right]$ for $n = 0,1,...,N-1$. The operating cost function can thus be defined as $C^{oper}\left(x^{des}, x^{oper}\right) = \sum_{n=0}^{N-1} C_n^{Qphb} Q_n^{phb}$ and the capital cost function as $C^{capital}(x^{des}) = C^{\tan k}\left(V^{tes}\right)^{0.7}$.

5.1. Decomposing scenarios using ADMM

We will consider the two-stage stochastic formulation for the design problem used in section 4.2. The uncertainty was represented using the scenario set $\mathcal{S} := \{1, 2, ..., S\}$, and the scenario tree was given in Figure 4-3. The design problem was represented in the compact form

$$\min_{x^{des}, x^{oper}} \quad C^{capital}\left(x^{des}\right) + \sum_{s\in\mathcal{S}} \omega_s C_s^{oper}\left(x^{des}, x_s^{oper}\right) \tag{5.6a}$$

$$s.t. \quad \left(x^{des}, x_s^{oper}\right) \in \chi \tag{5.6b}$$

We can consider each scenario as a separate partition, as shown in Figure 5-2, with the number of partitions P equal to the number of scenarios S.

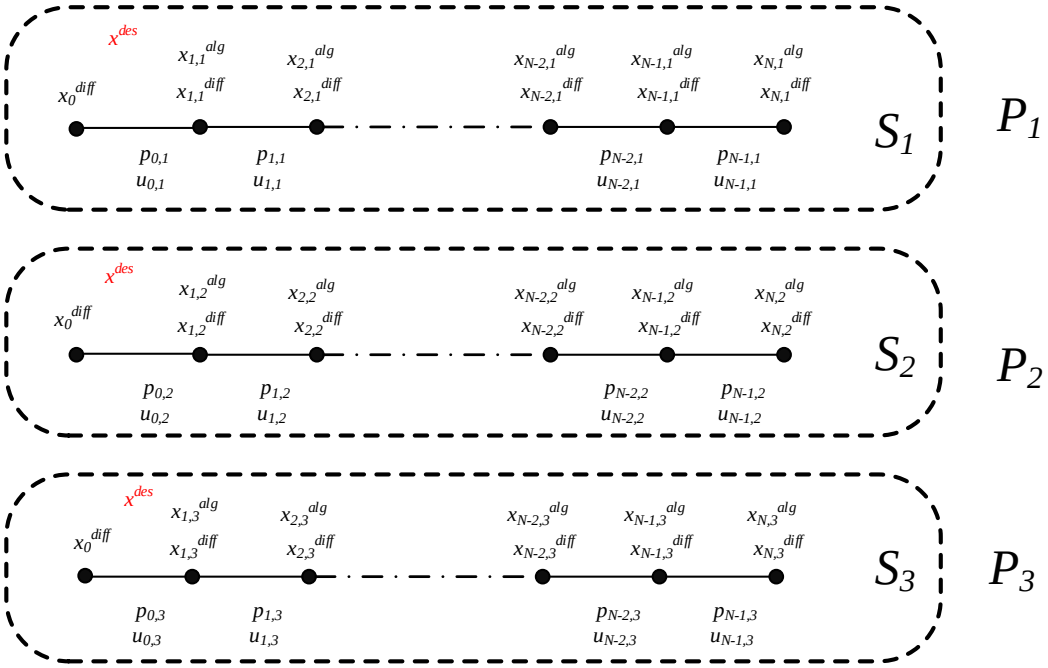

Figure 5-2: Schematic for scenario decomposition with 3 scenarios

The length of the horizon considered in each partition $i \in \mathcal{P}$ we will represent as N_i . Each partition *i* has its own private variable vector which we denote as x_i which is made up of the design variable vector for this partition represented as $\left(x^{des}\right)_i$ and the operation variable vector for this partition as $\left(x^{oper}\right)_i$. These vectors are as defined in the previous section, but now refer to the variables that are local to the partition *i*.

$$\begin{aligned} x_i &:= \left[(x^{des})_i, (x^{oper})_i\right] \\ &:= \left[(x^{des})_i, (x_0^{diff})_i, (x_{n+1,i}^{diff})_i, (x_{n,i}^{a\lg})_i, (u_{n,i})_i\right] \end{aligned} \qquad n = 0,1,\ldots,N_i - 1$$

All the partitions need to reach consensus on their copy of the design variable to be equivalent to the original problem.

$$\left(x^{des}\right)_i = x^{des} \qquad \forall i \in \mathcal{P}$$

We can construct the global variable *z* to contain an estimate of what the design variable in each partition should be (represented in red in the figure above).

$$z = \left[x^{des}\right]$$

The original NLP 5.6 can thus be written in the structured NLP form as

$$\min_{x_i, z} \quad \sum_{i \in \mathcal{P}} \left(\frac{C^{capital}\left((x^{des})_i\right)}{P} + \omega_i C^{oper}\left((x^{des})_i, (x^{oper})_i\right) \right) \tag{5.7a}$$

$$s.t. \quad \left((x^{des})_i, (x^{oper})_i\right) \in \chi_i \qquad i \in \mathcal{P} \tag{5.7b}$$

$$(x^{des})_i - z = 0 \qquad i \in \mathcal{P} \tag{5.7c}$$

ADMM for solving this problem can be derived by forming the augmented Lagrangian by adding the coupling constraint (5.7c) to the objective term

$$\min_{x_i, z} \quad \sum_{i \in \mathcal{P}} \left(\frac{C^{capital}\left((x^{des})_i\right)}{P} + \omega_i C^{oper}\left((x^{des})_i, (x^{oper})_i\right) + \mu_i^T\left((x^{des})_i - z\right) + \frac{\rho}{2}\left\|(x^{des})_i - z\right\|_2^2 \right) \tag{5.8a}$$

$$s.t. \quad \left((x^{des})_i, (x^{oper})_i\right) \in \chi_i \qquad i \in \mathcal{P} \tag{5.8b}$$

We can now directly follow the ADMM algorithm for structured NLPs we had described in section 2.4. In iteration *k*, we can separately solve each individual partition problem as (keeping the global variable fixed).

$$\min_{x_i} \quad \frac{C^{capital}\left((x^{des})_i\right)}{P} + \omega_i C^{oper}\left((x^{des})_i, (x^{oper})_i\right) + \mu_i^{kT}\left((x^{des})_i - z^k\right) + \frac{\rho}{2}\left\|(x^{des})_i - z^k\right\|_2^2 \tag{5.9a}$$

$$s.t. \quad \left((x^{des})_i, (x^{oper})_i\right) \in \chi_i \qquad i \in \mathcal{P} \tag{5.9b}$$

The global variable update step is the averaging operator, and since all the partitions have a copy of the global variable, it can be written as

$$z^{k+1} = \sum_{i \in \mathcal{P}} \omega_i (x^{des})_i$$

If all the scenarios have equal probability, then it can also be written as

$$z^{k+1} = \frac{\sum_{i \in \mathcal{P}} (x^{des})_i}{P}$$

The primal residual is just in the infeasibility of the design variable and can be defined for each partition as

$$r_i^{k+1} = (x^{des})_i^{k+1} - z^{k+1} \tag{5.10}$$

The dual variable update in the partitions is thus

$$\mu_i^{k+1} = \mu_i^k + \rho\left((x^{des})_i^{k+1} - z^{k+1}\right)$$

The dual residual in each partition is thus

$$s_i^{k+1} = \rho(z^{k+1} - z^k) \tag{5.11}$$

An interesting aspect to note here is that by applying ADMM for scenario decomposition in the two-stage stochastic program here, we have ended up with the progressive hedging (PH) algorithm described by Rockafellar and Wets [29]. The PH algorithm is a popular method used to solve two-stage stochastic problems in a distributed manner where the second stage problems are all coupled via the first stage variables. It can thus be shown that PH is a particular case of ADMM, and that ADMM provides a more general framework to derive decomposition strategies for large optimization problems [21].

5.2. Illustrative example of decomposing 2 scenarios as 2 partitions

We will use an illustrative example to demonstrate the approach discussed in the previous section. Let us consider two scenarios with a prediction horizon of 48 hours, where the time-varying disturbance Q^{whb} is only considered and shown in Figure 5-3 and scenarios have equal probability.

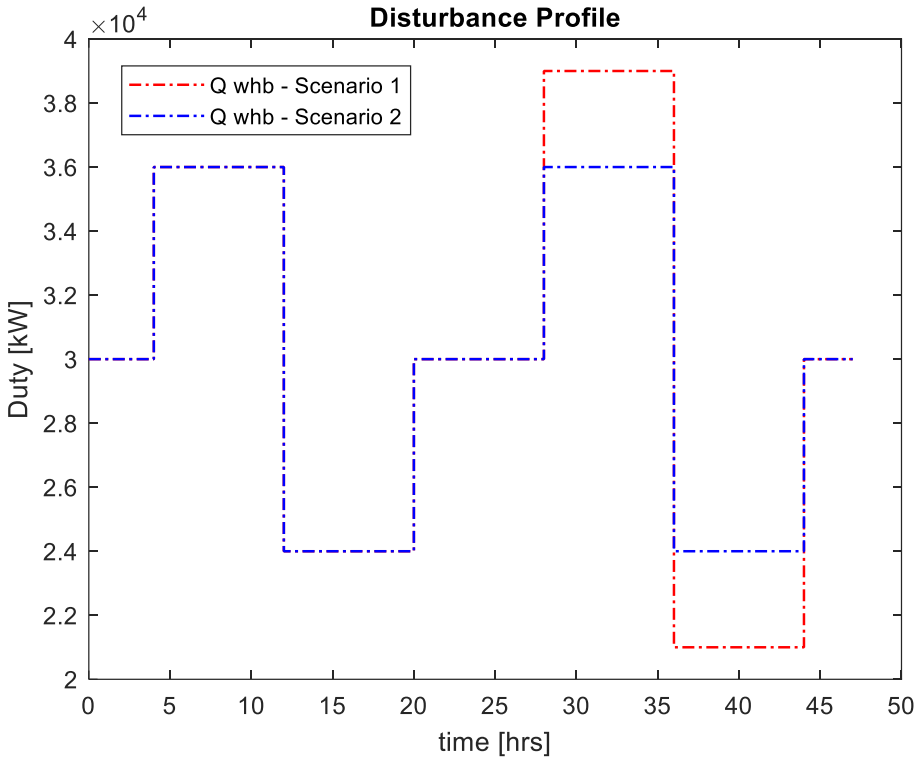

Figure 5-3: Scenarios used in the illustrative example

The design problem (5.6) can be solved as a single NLP which gives us the optimal solution for the tank volume V^{tes} = 17709 m^3 and the objective function value of 730.05 *USD*. In an actual case, one does not expect to solve this problem centrally, but here we use this to compare our results from the distributed algorithm. In the distributed algorithm, we split the NLP into two partitions, each with its own local variables. The global variable z thus includes an estimate of the variable V^{tes}. All the optimization problems are solved using off the shelf interior-point solver IPOPT [27]; thus, individual solutions are not necessarily global minimum. The various metrics during the iterations of the ADMM algorithm using a constant penalty parameter (ρ) are plotted in the following figures.

The primal and dual residuals for partition 1 (Equation 5.10 and 5.11) are shown in semi-log plots in Figure 5-4. Each iteration, the primal and dual residuals correspond to the infeasibility in the tank volume between the partitions converging to zero within a few iterations. This property is not guaranteed by the ADMM algorithm in the case of nonconvex problems and should be checked before analyzing any other results.

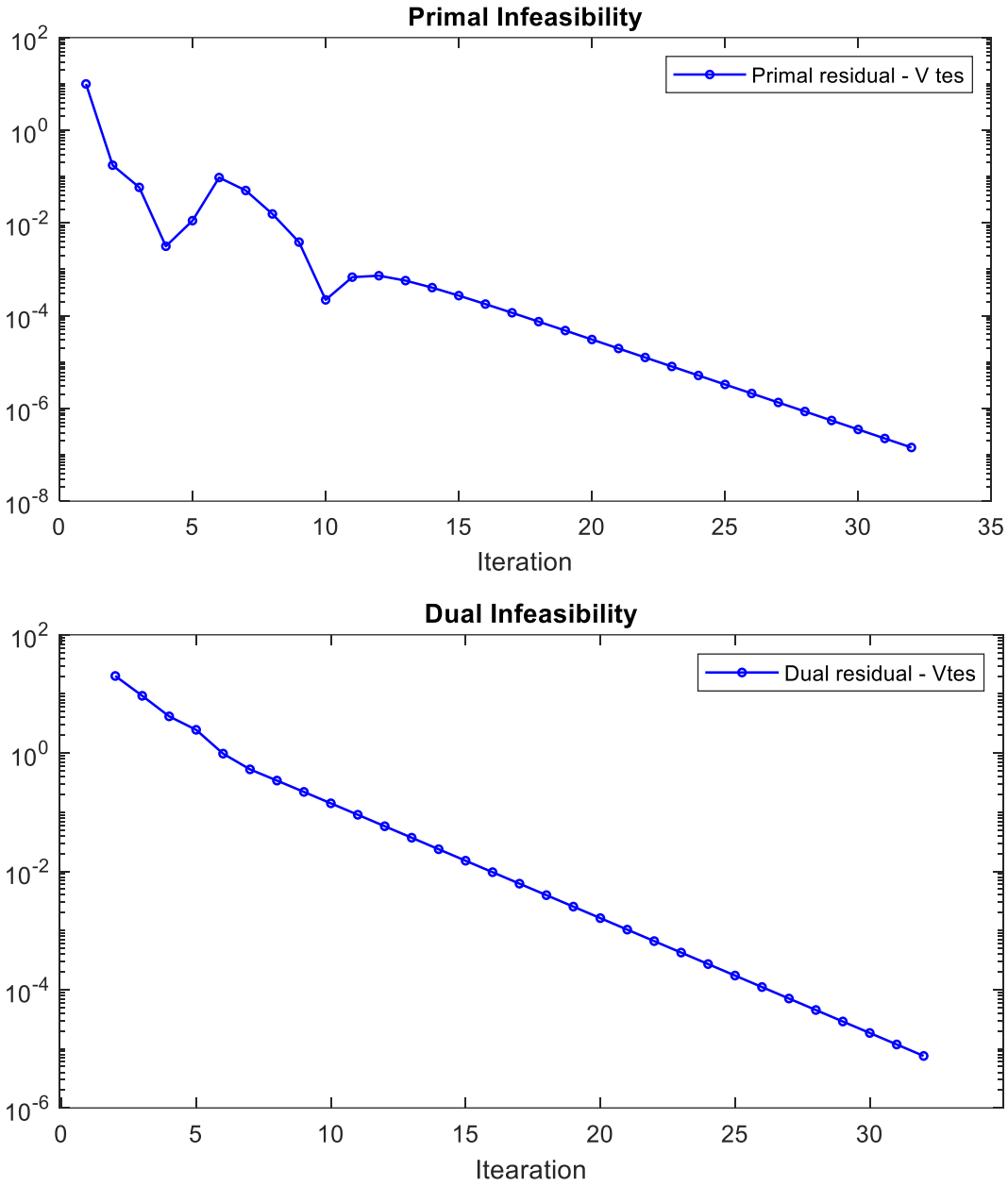

Figure 5-4: Convergence of ADMM (primal and dual residual)

As the primal residual tends to zero, the penalty term in the augmented Lagrangian goes to zero, and its value stabilizes. The objective function value obtained from solving the problem centrally is marked in Figure 5-5 to quickly see if ADMM is converging to the same solution. This is not guaranteed since ADMM must be considered as just another local optimization method as we had discussed in section 2.2. With different initial values of x, z, μ and the penalty parameter ρ, we could expect ADMM to converge to a different local optimum altogether.

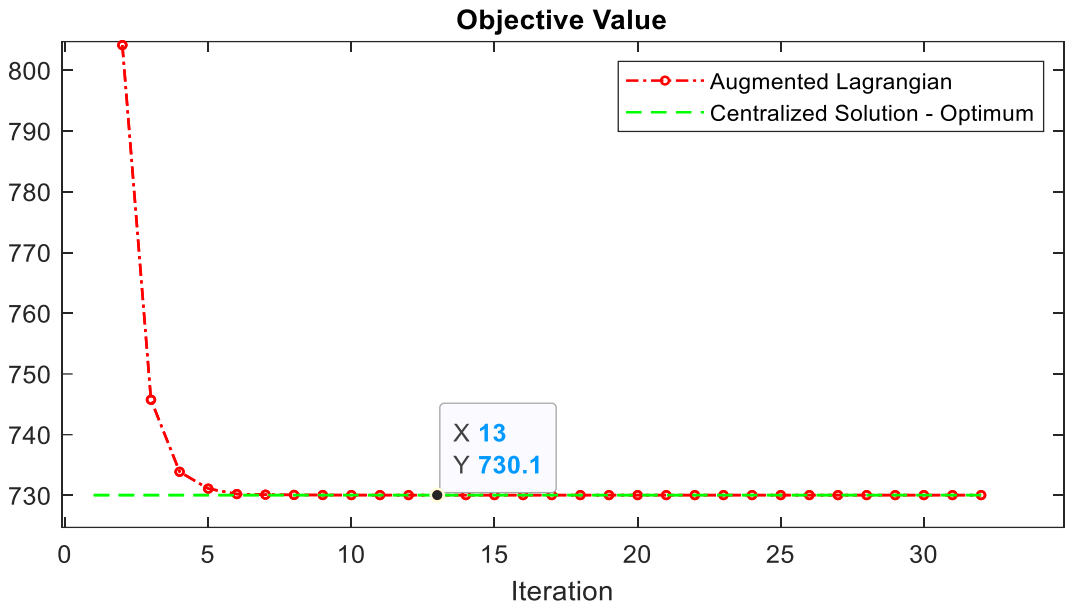

Figure 5-5: Convergence of ADMM (objective function value)

The local copies corresponding to the global variable are shown in Figure 5-6. We can see that both the local variables were initialized with the initial guess for tank volume at 10,000 m^3 and later stabilizes to a constant value (residual convergence). In this particular case, we can see that ADMM has converged to the same optimal value we had found from the central solution.

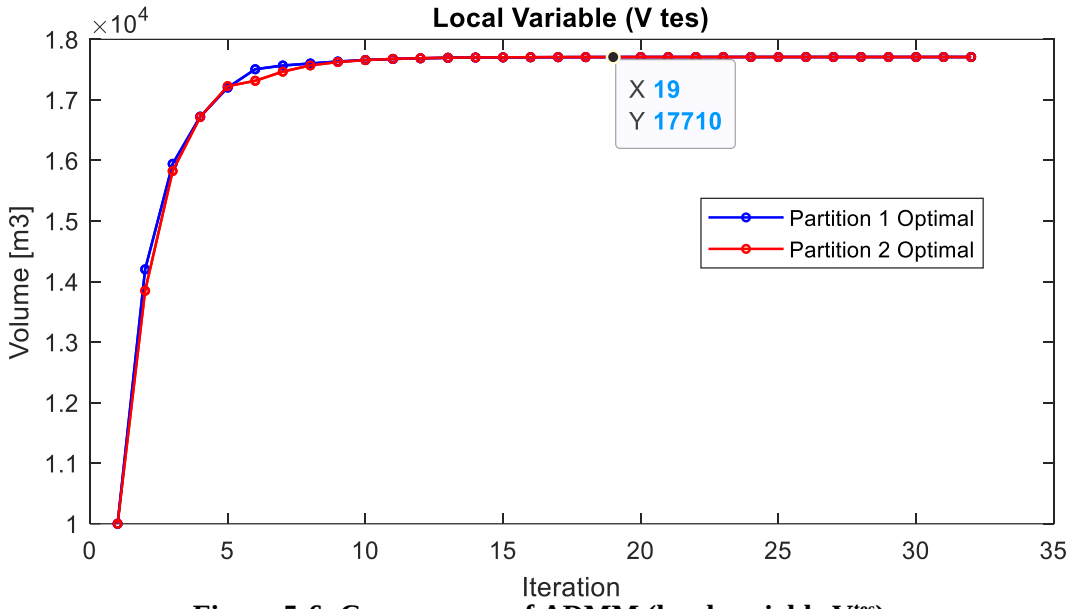

Figure 5-6: Convergence of ADMM (local variable V^{tes})

5.3. Discussions from the illustrative example

An important aspect we had observed while solving the illustrative example is the vital role the penalty term plays on the convergence behaviour of the algorithm. It is crucial to ensure that the penalization term does not skew the objective of the partition problems too much. Using the design variable as they were defined initially, resulted in the penalty term being orders of magnitude larger than the optimal cost itself. This results in the ADMM iterations having a very low rate of convergence from the initial guesses we had given for the design variable. To avoid this, the design variable was scaled down to lie between 10-100 such that the penalization term $\left\| \left(x^{des} \right)_i - z \right\|$ became in the same orders of magnitude as the optimal objective function value.

The choice of the penalty parameter (ρ) for ADMM iterations also influenced the magnitude of the penalization term. More importantly, the penalty parameter has a significant influence on the speed of convergence of the algorithm as it is also the step length for the dual variable update between iterations. This parameter needs to be large enough for the subproblems to converge, but excessively large values did lead to numerical instabilities. A too-large step length caused oscillations in the dual residual, and the solutions of the subproblems cycling between iterations. This behaviour is in some ways similar to a line search algorithm overshooting the optimal point if we use too large of a step length.

The time required to solve each iteration was significantly reduced by using a warm start approach, where the solution from the previous iteration was used as the initial guess for the next iteration.

Chapter 6. Applying ADMM for decomposing a long scenario

When increasing the prediction horizon of the design problem, we expect the need to form partitions within a long scenario in addition to the partitions across scenarios considered earlier. In this chapter, ADMM is applied to partition a single long prediction horizon.

6.1. Decomposing a long scenario

We can partition a single long horizon of length N into P partitions. Each partition "i" can have its own desired length represented as N_i. A schematic for forming 3 partitions is shown in Figure 6-1.

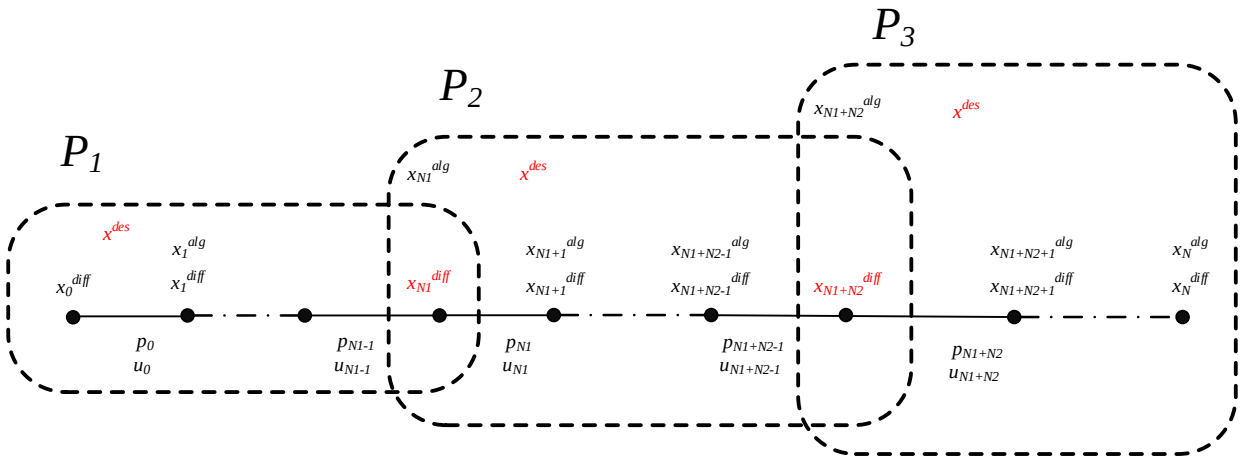

Figure 6-1: Schematic for partitioning a single long scenario into 3 partitions

Each partition has its own local variables as before represented as

$$\begin{aligned} x_i &:= \left[(x^{des})_i, (x^{oper})_i \right] \\ &:= \left[(x^{des})_i, (x_0^{diff})_i, (x_{n+1}^{diff})_i, (x_n^{alg})_i, (u_n)_i \right] \end{aligned} \qquad n = 0,1,\ldots,N_i - 1$$

The partitions must now reach consensus not only on their copy of the design variable, but the adjacent partitions must now also ensure continuity of the differential states at their common boundary (shown in red). Since the first and last partitions are only coupled on one end, this coupling constraint can be written as

$$\left(x^{des}\right)_i = x^{des} \qquad \forall i \in \mathcal{P}$$

$$\left(x_{N_i}^{diff}\right)_i = \left(x_0^{diff}\right)_{i+1} \qquad i = 1,2,\ldots,P-1$$

The global variable now contains the estimate for the design variable along with the differential state at the boundary between partitions.

$$z = \left[x^{des}, x_{N_i}^{diff} \right] \qquad i = 1, 2, \ldots, P-1$$

The design problem can then be written in the structured NLP form as

$$\min_{x_i, z} \quad \sum_{i \in \mathcal{P}} \frac{C^{capital}\left((x^{des})_i\right) + C^{oper}\left((x^{des})_i, (x^{oper})_i\right)}{P} \tag{5.12a}$$

$$\left((x^{des})_i, (x^{oper})_i\right) \in \chi_i \qquad \forall i \in \mathcal{P} \tag{5.12b}$$

$$A_i x_i - B_i z = 0 \qquad \forall i \in \mathcal{P} \tag{5.12c}$$

where the matrices A_i and B_i are sparse matrices used to link the local variables in the partition to the corresponding global copy. Similar to the previous chapter, we form the augmented Lagrangian by adding the coupling constraint (5.12c) to the objective term and applying the ADMM algorithm to solve the partition problems separately as

$$\min_{x_i} \quad \frac{C^{capital}\left((x^{des})_i\right) + C^{oper}\left((x^{des})_i, (x^{oper})_i\right)}{P} + \mu_i^{kT}\left(A_i x_i - B_i z^k\right) + \frac{\rho}{2}\left\| A_i x_i - B_i z^k \right\|_2^2 \tag{5.13a}$$

$$\left((x^{des})_i, (x^{oper})_i\right) \in \chi_i \tag{5.13b}$$

The design variable in the global variable is linked to all the partitions, while each state variable at a boundary is linked to the adjacent partitions only. The elements in the global variable can then be updated as

$$\left(x^{des}\right)^{k+1} = \frac{\sum_{i \in \mathcal{P}} (x^{des})_i}{P} \qquad \forall i \in \mathcal{P}$$

$$\left(x_{N_i}^{diff}\right)^{k+1} = \frac{\left(x_{N_i}^{diff}\right)_i^{k+1} + \left(x_0^{diff}\right)_{i+1}^{k+1}}{2} \qquad i = 1, 2, \ldots, P-1$$

The primal residual is the infeasibility of the coupling constraint and defined for each partition as

$$r_i^{k+1} = A_i x_i^{k+1} - B_i z^{k+1} \tag{5.14}$$

The dual variable is then updated in each partition as

$$\mu_i^{k+1} = \mu_i^k + \rho\left(r_i^{k+1}\right)$$

The dual residual is

$$s^{k+1} = \rho A^T B(z^{k+1} - z^k) \tag{5.15}$$

It is interesting to note that our approach shares similarities to multiple shooting (MS) approach for solving dynamic optimization problems [25]. The main difference is that in MS, the state continuity constraints across the partitions are enforced explicitly by the optimization solver. In contrast, our approach, they are enforced implicitly by minimizing the AL. The optimization solver itself might be using a barrier penalty approach to enforce these equality constraints, which makes the approaches even more similar.

6.2. Illustrative example of decomposing a scenario into 2 partitions

Let us consider a single scenario with a horizon of 48 hours with the time-varying disturbance, as shown in Figure 6-2. The design problem can be solved as a single NLP which gives us the solution for tank volume V^{tes} = 12052 m^3 and the objective function value is 560.35 *USD*. The optimal state profile is shown in the bottom subplot of Figure 6-3. Similar to earlier, we represent the central solution here to compare the convergence behaviour of our distributed algorithm when the initial condition for all the states are chosen to be the same as the centralized solver.

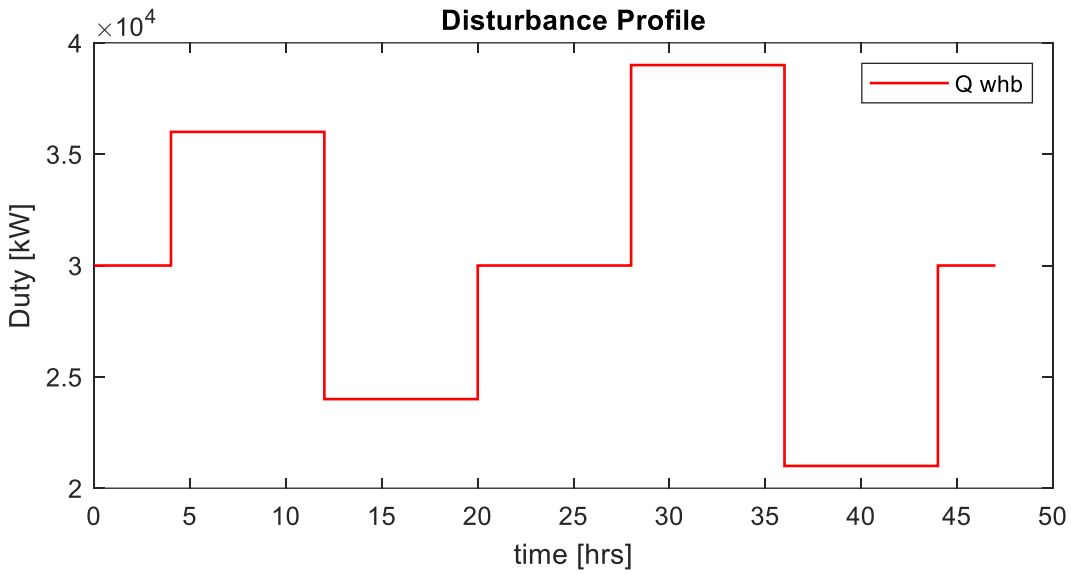

Figure 6-2: Scenario used in the illustrative example

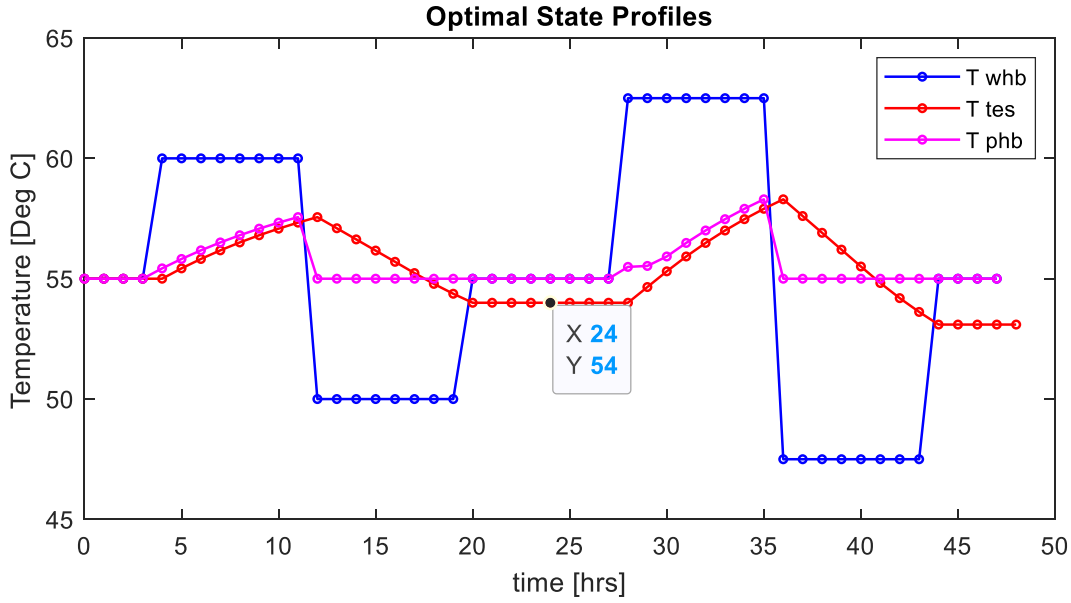

Figure 6-3: Centralized solution (state profile)

In the distributed approach, we can split the NLP into two partitions each with a 24-hour horizon. The global variable z contains the design variable V^{tes} and the differential state at the end of the first partition. From the central solution, we can see that the value of z at the optimum was [12052, 54]. The various metrics during the iterations of the ADMM algorithm is plotted in the following figures using a constant penalty parameter ($\rho = 5$).

The primal and dual residuals (Equations 5.14 and 5.15) are shown in semi-log plots in Figure 6-4. Each iteration, the primal and dual residual vectors contains the infeasibility in V^{tes} and the infeasibility in the differential state T^{tes} continuity between the partitions. Their individual components are represented for observing the trends rather than as a single norm here. We see that the residuals tend to zero with more iterations. We observe oscillations in dual infeasibility, which indicate to us that more coordination steps are necessary before performing the dual update step in ADMM in this case.

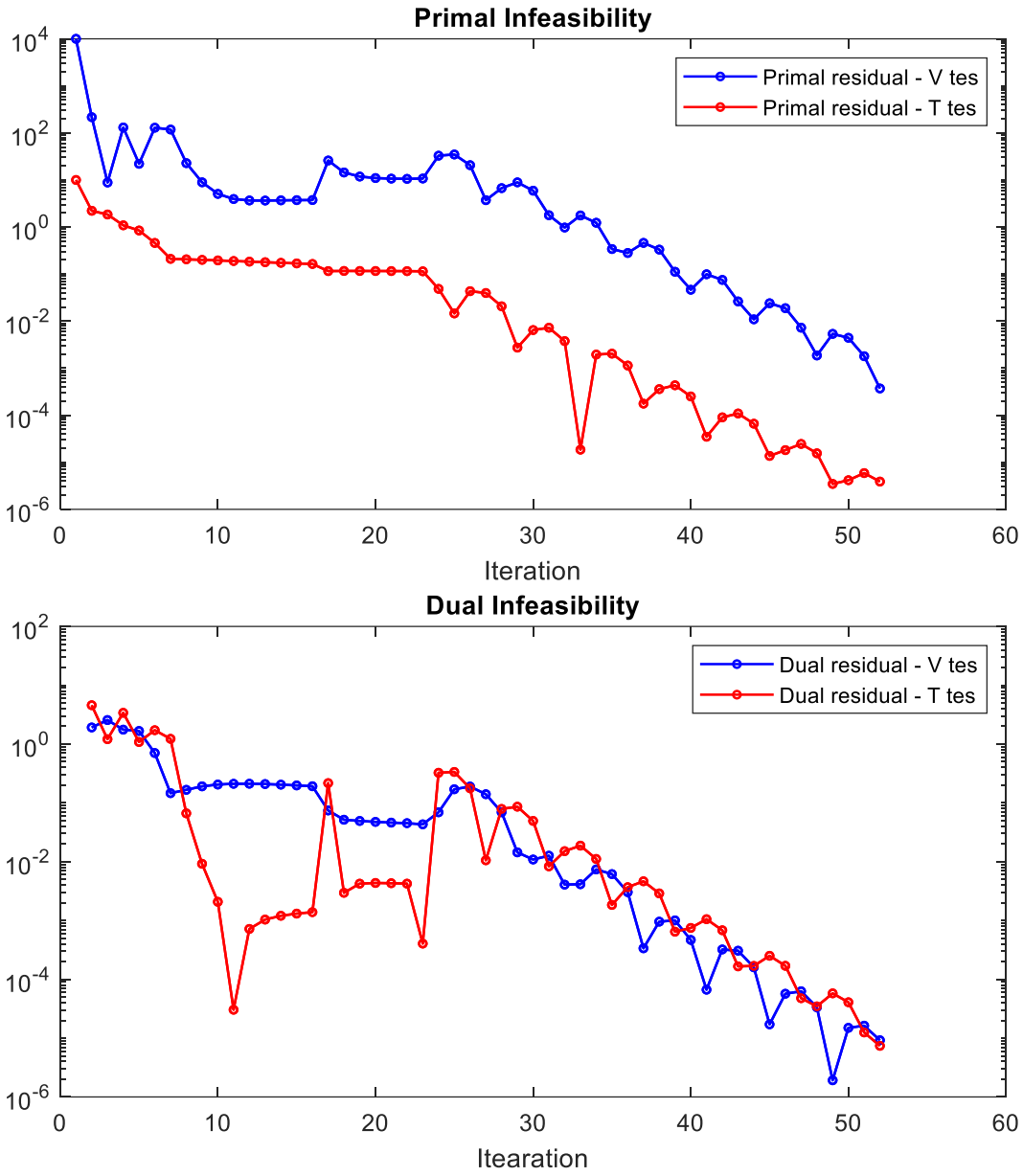

Figure 6-4: Convergence of ADMM (primal and dual residuals)

As the primal residual tends to zero, the penalty term in the augmented Lagrangian goes to zero, and it stabilizes. The objective function value that we got from solving the problem centrally is indicated in Figure 6-5 to see if ADMM is converging to the same solution. The AL in this case, actually increases with iterations highlighting the effect of nonconvexity present in the problem.

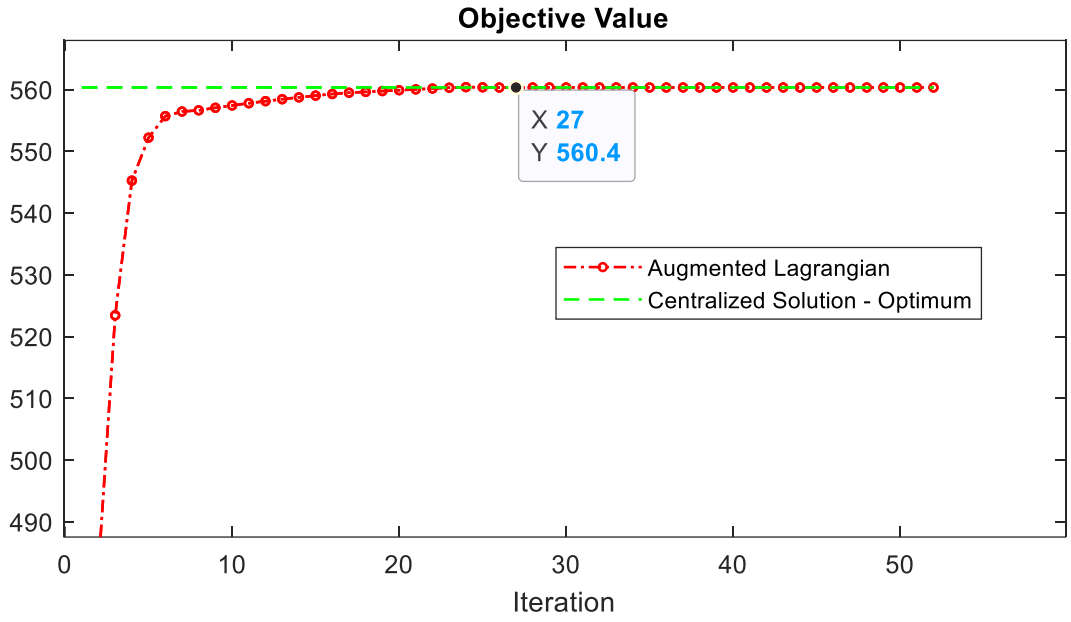

Figure 6-5: Convergence of ADMM (objective function value)

The local copies corresponding to the global variable are shown in Figure 6-6. Volume of the tank is gradually updated from the initial guess towards the optimal solution which happens to match with the solution we had found by solving the problem centrally. The behaviour of the differential state at the boundary is more interesting here. This variable is the end temperature of the TES in partition 1 while it is the initial temperature in partition 2. After the first iteration, the optimal for partition 1 is to have the TES tank discharged, while for partition 2 it is optimal to start the system with a charged TES tank. The penalty term in the augmented Lagrangian prevents partition 2 from setting this variable at its upper limit. The global copy for this temperature is updated based on the local copies, and the associated Lagrange multipliers are updated in each partition. The AL thus is able to implicitly force the partitions to reach consensus on this state variable within a few iterations.

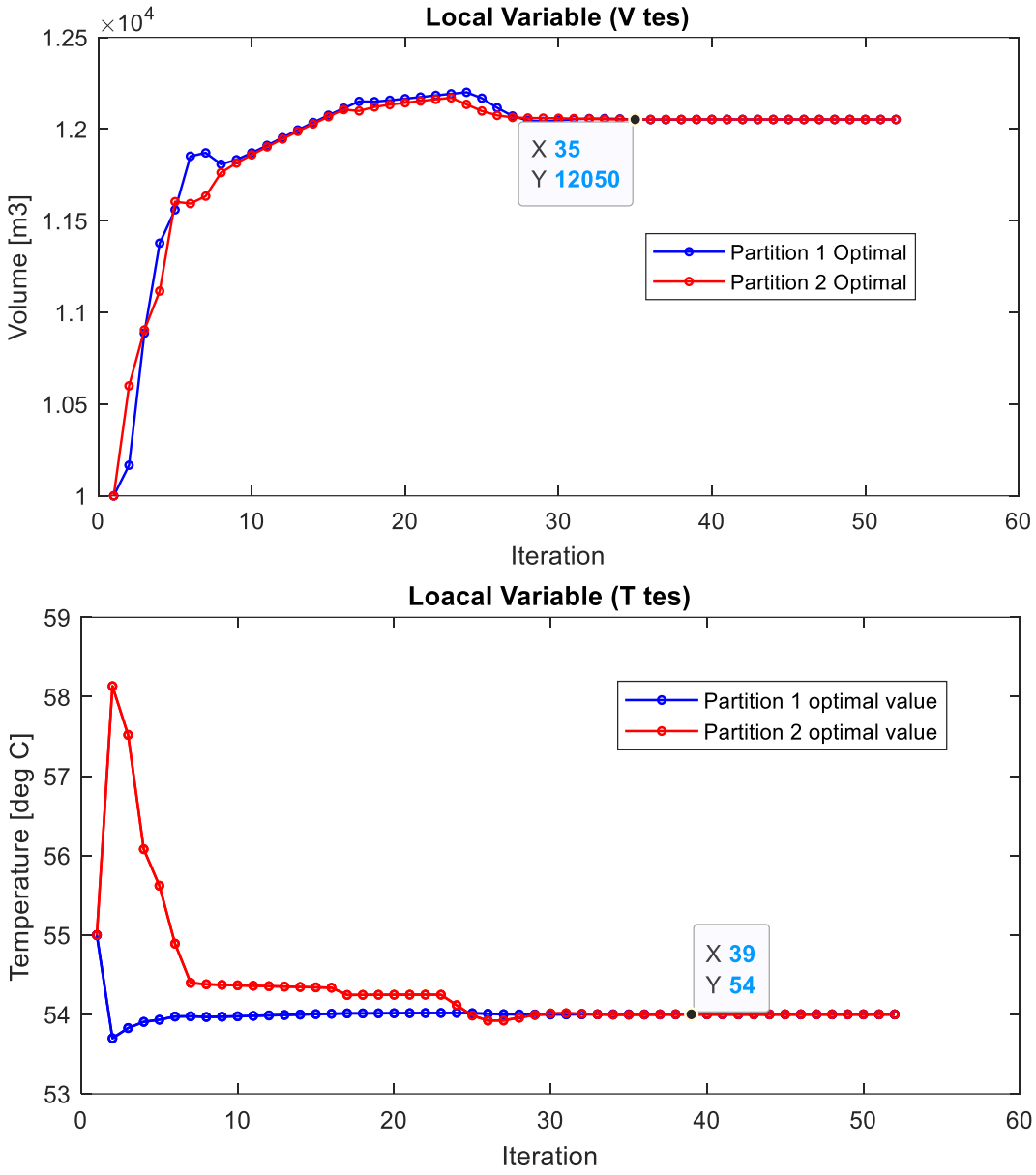

Figure 6-6: Convergence of ADMM (local variables)

6.3. Discussions from the illustrative example

In this example, we had to reach consensus between partitions on two variables which had different magnitudes in their residuals during the iterations. We could observe that with low penalty parameter ρ, the residual convergence in the variable with the lower residual (in our case – the temperature) was much slower. The choice of the penalty parameter ρ played a large role in convergence in this example affecting the speed of convergence but also on the point ADMM converges to. Since ADMM is a local optimization method, this behaviour is to expected and indicates the presence of nonconvexity in the problem.

Chapter 7. Conclusion and future work

Our main objective at the beginning of the research period has been on utilizing the alternating direction method of multipliers (ADMM) to get a distributed optimization algorithm to solve structured nonlinear programming problems (NLP). A Thermal Energy Storage (TES) optimal design problem was chosen to motivate the need for distributed optimization and demonstrate the approach using illustrative examples. The main focus thus has been on

1. Developing a simple model of the TES process to capture the important nonlinear dynamics present in the system.

2. Framing the design problem for a TES system as a two-stage stochastic NLP.

3. Forming partitions in the design problem by exploiting the structure present in the problem.

4. Using ADMM as the distributed optimization algorithm to coordinate between the individual partition problems (which can now be solved in parallel) to get the solution to the original NLP

We demonstrated the approach of forming partitions in the NLP by framing the problem in the general consensus form and solving them using ADMM. While implementing the approach in the illustrative examples, we were able to observe the importance of scaling the problem such that the penalty term does not skew the objective function in the partitions. The coupling constraints and the objective function should be scaled to systematically tune the residual convergence when extending this approach to more complex models.

We briefly highlighted how the choices of the initial guesses for the variables and the penalty parameter affected the ADMM algorithm in the case of nonconvex problems. Investigating the nonconvexity is not straightforward and was not really pursued here, but is another interesting area to explore in the future.

Although the ADMM algorithm provided opportunities to solve the partition problems in parallel each iteration, they were solved sequentially for ease of development in all the case studies performed. When extending this to very large problems, the implementation in a distributed computing environment would be required.

References

[1] IEA, ‘Renewables 2019 – Analysis and forecast to 2024’, 2019.

[2] G. Alva, Y. Lin, and G. Fang, ‘An overview of thermal energy storage systems’, *Energy*, vol. 144, pp. 341–378, 2018.

[3] K. Powell, ‘Dynamic optimization of energy systems with thermal energy storage’, *Prelim. Res. Propos. Austin, Texas, USA*, vol. 6, pp. 1–26, 2011.

[4] Z. N. Mdoe, ‘Optimal control of thermal energy storage under supply and demand uncertainty Optimal control of thermal energy storage’, NTNU, 2019.

[5] M. Thombre, D. Krishnamoorthy, and J. Jäschke, ‘Data-driven online adaptation of the scenario-tree in multistage model predictive control’, *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 461–467, 2019.

[6] M. Thombre, Z. Mdoe, and J. Jäschke, ‘Data-Driven Robust Optimal Operation of Thermal Energy Storage in Industrial Clusters’, 2020.

[7] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*. New York, NY: Springer New York, 2011.

[8] B. Xu *et al.*, ‘Scalable Planning for Energy Storage in Energy and Reserve Markets’, *IEEE Trans. Power Syst.*, vol. 32, no. 6, pp. 4515–4527, 2017.

[9] A. D. Lamont, ‘Assessing the economic value and optimal structure of large-scale electricity storage’, *IEEE Trans. Power Syst.*, vol. 28, no. 2, pp. 911–921, 2013.

[10] S. Prakash, ‘Optimal Operation and Design of Thermal Energy Storage System’, 2019.

[11] M. Thombre, S. Prakash, and R. Knudsen, ‘Optimizing the Capacity of Thermal Energy Storage in Industrial Clusters’, 2020.

[12] R. Martí, S. Lucia, D. Sarabia, R. Paulen, S. Engell, and C. De Prada, ‘Improving scenario decomposition algorithms for robust nonlinear model predictive control’, *Comput. Chem. Eng.*, vol. 79, pp. 30–45, 2015.

[13] D. Krishnamoorthy, B. Foss, and S. Skogestad, ‘A Distributed Algorithm for Scenario-based Model Predictive Control using Primal Decomposition∗’, *IFAC-PapersOnLine*, vol. 51, no. 18, pp. 351–356, 2018.

[14] D. Krishnamoorthy, ‘Novel Approaches to Online Process Optimization Under Uncertainty Addressing the limitations of current industrial practice’, NTNU, 2019.

[15] S. Lucia, S. Subramanian, and S. Engell, ‘Non-conservative robust Nonlinear Model Predictive Control via scenario decomposition’, *Proc. IEEE Int. Conf. Control Appl.*, pp. 586–591, 2013.

[16] D. P. Palomar and M. Chiang, ‘A tutorial on decomposition methods for network utility maximization’, *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, 2006.

[17] M. R. Hestenes, ‘Multiplier and gradient methods’, *J. Optim. Theory Appl.*, vol. 4, no. 5, pp. 303–320, 1969.

[18] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, ‘Distributed optimization and statistical learning via the alternating direction method of multipliers’, *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2010.

[19] J. Nocedal, *Numerical Optimization*. Springer New York, 2006.

[20] D. P. Bertsekas and S. Paulo, *Constrained Optimization and Lagrange Multiplier Methods*. Elsevier, 1982.

[21] J. S. Rodriguez, B. Nicholson, C. Laird, and V. M. Zavala, ‘Benchmarking ADMM in nonconvex NLPs’, *Comput. Chem. Eng.*, vol. 119, pp. 315–325, 2018.

[22] Z. N. Mdoe, ‘Optimal control of thermal energy storage under supply and demand uncertainty Optimal control of thermal energy storage’, 2019.

[23] K. W. Mathisen, M. Morari, and S. Skogestad, ‘Dynamic models for heat exchangers and heat exchanger networks’, *Comput. Chem. Eng.*, vol. 18, pp. S459–S463, 1994.

[24] P. Varbanov, J. Klemeš, and F. Friedler, ‘Cell-based dynamic heat exchanger models - direct determination of the cell number and size’, *Comput. Aided Chem. Eng.*, vol. 28, no. C, pp. 439–444, 2010.

[25] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. 2010.

[26] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, ‘CasADi: a software framework for nonlinear optimization and optimal control’, *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.

[27] A. Wächter and L. T. Biegler, ‘On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming’, *Math. Program.*, vol. 106, no. 1, pp. 25–57, Mar. 2006.

[28] R. Sinnott and G. Towler, *Chemical Engineering Design - Principles, Practice and Economics of Plant and Process Design*. 2003.

[29] R. T. Rockafellar and R. J.-B. Wets, ‘Scenarios and Policy Aggregation in Optimization Under Uncertainty’, *Math. Oper. Res.*, vol. 16, no. 1, pp. 119–147, Feb. 1991.

Appendix

A1. MATLAB Codes

This section contains some important pieces of source code developed in this project. The code is written in MATLAB and used CasADi [26] as the symbolic framework for implementing the nonlinear optimization problems, which were then solved using IPOPT [27].

A1.a. Main File

This file **`main.m`** is used to solve the design problem (in a centralized or distributed fashion). The code in its current version is used to solve the problem using 2 partitions solved sequentially during the ADMM iterations.

```
% Distributed Optimization using ADMM for TES Design
% 2 Nodes
% Author: Sandeep Prakash - sandeepp@stud.ntnu.no
% June 2020

%% Clearing Workspace
close all;
clear
import casadi.*
global par_sim;
global par_model;

%% Simulation Parameters
Tsim = 24*2;        par_sim.Tsim = Tsim;     %Design Life (in hours)
dt = 1;             par_sim.dt = dt;         %hours
Nsim = Tsim/dt;     par_sim.Nsim = Nsim;     %Discrete time steps in sim
par_sim.N_scale = 1;
Num_nodes = 2;      par_sim.Num_nodes = Num_nodes; % = 1 sets centralized

par_sim.Num_colloc_points = 3;
Node_overlap = 0;   par_sim.Node_overlap = Node_overlap;

%% Generating Disturbance profiles
% Initializing Model Parameters
par_model = param_model();
```

```
%Generate Disturbance and COnstraint Profiles
[dist, constr] = Generate_Profiles();

%% Checking integrator at single point
% Creating Integrator
[F,x_var, z_var, p_var, alg, diff, L] = myfunc_Integrator();

% Generating bounds and initial guesses
[x0,z0,u0,Des_var0,lbx,lbz,lbu,lbDes_var,ubx,ubz,ubu,ubDes_var] = ...
Initialization_bounds();
[x0_,z0_,u0_,Des_var0_] = scale_to_one(1, x0,z0,u0,Des_var0);

%Testing integrator at a single point
d0 = [dist.q_dh{1}(1); dist.Q_Supply{1}(1)];
Fk = F('x0',x0,'z0',z0,'p',[u0; Des_var0; d0]);
xf_ = full(Fk.xf);
zf_ = full(Fk.zf);
disp(Fk.qf);

%[xf, zf, u0, Des_var0] = scale_to_one(-1, xf_, zf_, u0_, Des_var0_)
%find_a_Steady_State(d0)

%% Setting up NLP in Nodes

opts = struct('warn_initial_bounds',false, 'print_time',false, ...
'ipopt',struct('max_iter',5000,'print_level',5));

if Num_nodes == 1
%% Setting Centralized Node (unlimited memory)

% Collecting Profiles to Node Data Structure
nodeID = 1;
N0_Data.nodeID = nodeID;
N0_Data.dist = dist;
N0_Data.constr = constr;

% create Node variables
```

```
[Jcentr,w_pubDes, w_pubL, w, w_pubR, g,p_NLP, W0, LBW, UBW, ...
lbg,ubg,p_NLP0] = create_NodeModel(N0_Data);

% w refers to central solver, whereas w0_w0 refers to initial guess
% for the variables (first 0 indicating centralized)
w0_pubDes0 = W0.w_pubDes0;          w0_pubL0 = W0.w_pubL0;
lbw_pubDes = LBW.lbw_pubDes;        lbw_pubL = LBW.lbw_pubL;
ubw_pubDes = UBW.ubw_pubDes;        ubw_pubL = UBW.ubw_pubL;
w0_w0 = W0.w0;                      w0_pubR0 = W0.w_pubR0;
lbw = LBW.lbw;                      lbw_pubR = LBW.lbw_pubR;
ubw = UBW.ubw;                      ubw_pubR = UBW.ubw_pubR;

else
%% Setting Local Nodes (limited memory)

%% Collecting Profiles to Node Data Structure N1
nodeID = 1;
N1_Data.nodeID = nodeID;
N1_Data.dist = dist;
N1_Data.constr = constr;

% create Node 1 variables
[J1,w1_pubDes, w1_pubL, w1, w1_pubR, g1,p1_NLP, W1_0, LBW1, UBW1,...
lbg1,ubg1,p1_NLP0] = create_NodeModel(N1_Data);

w1_pubDes0 = W1_0.w_pubDes0;        w1_pubL0 = W1_0.w_pubL0;
lbw1_pubDes = LBW1.lbw_pubDes;      lbw1_pubL = LBW1.lbw_pubL;
ubw1_pubDes = UBW1.ubw_pubDes;      ubw1_pubL = UBW1.ubw_pubL;

w1_w0 = W1_0.w0;                    w1_pubR0 = W1_0.w_pubR0;
lbw1 = LBW1.lbw;                    lbw1_pubR = LBW1.lbw_pubR;
ubw1 = UBW1.ubw;                    ubw1_pubR = UBW1.ubw_pubR;

%% Collecting Profiles to Node Data Structure N2
nodeID = 2;
N2_Data.nodeID = nodeID;
N2_Data.dist = dist;
N2_Data.constr = constr;
```

```
% create Node 2 variables
[J2,w2_pubDes, w2_publ, w2, w2_pubR, g2,p2_NLP, W2_0, LBW2,UBW2,...
lbg2,ubg2,p2_NLP0] = create_NodeModel(N2_Data);

w2_pubDes0 = W2_0.w_pubDes0;          w2_publ0 = W2_0.w_publ0;
lbw2_pubDes = LBW2.lbw_pubDes;        lbw2_publ = LBW2.lbw_publ;
ubw2_pubDes = UBW2.ubw_pubDes;        ubw2_publ = UBW2.ubw_publ;
w2_w0 = W2_0.w0;                      w2_pubR0 = W2_0.w_pubR0;
lbw2 = LBW2.lbw;                      lbw2_pubR = LBW2.lbw_pubR;
ubw2 = UBW2.ubw;                      ubw2_pubR = UBW2.ubw_pubR;

end

%% Solve NLP
if Num_nodes == 1

%% Solve Centralized NLP
% structure for optimizer
centr_prob = struct('x', vertcat(w_pubDes{:},w_publ{:}, w{:},w_pubR{:}),...
'g', vertcat(g{:}), ...
'p', vertcat(p_NLP{:}), ...
'f', Jcentr );

% creating solver
centr_solver = nlpsol('solver', 'ipopt', centr_prob, opts);

% Solve the NLP
centr_sol = centr_solver('x0', vertcat(w0_pubDes0,w0_publ0,w0_w0,w0_pubR0),...
     'lbx',vertcat(lbw_pubDes,lbw_publ,lbw,lbw_pubR), ...
     'ubx',vertcat(ubw_pubDes,ubw_publ,ubw,ubw_pubR), ...
     'lbg',lbg,'ubg',ubg, ...
     'p', p_NLP0);

flag_centr = centr_solver.stats()
centr_w_opt = full(centr_sol.x);
centr_lam_opt = full(centr_sol.lam_x);
```

```
%Displaying cost, End states and Volume
centr_cost = full(centr_sol.f);
centr_pubDes_star   = centr_w_opt(1);

%for pausing - if any subproblem failed
if  (flag_centr.success) == 1
      disp(['solved succesfully - centrally']);
      design_val = centr_pubDes_star;
%         [~,~,~,design_val] = scale_to_one(-1, 0,0 ,0 ,centr_pubDes_star);
      disp(['Optimal Volume is ' num2str(design_val(1)) ' m3']);
%         disp(['Optimal Area is ' num2str(centr_pubDes_star(2)) ' m2']);
      disp(['Optimal cost is ' num2str(centr_cost) ' USD']);
else
      disp(['Central solver not converged'])
      disp(['Central Solver ' flag_centr.return_status])
end

plot_NodeProfile(centr_w_opt,centr_lam_opt,N0_Data,W0)
else

%% Solve distributed problem

% Iteration Tolerance Parameters
rho = 0.05;
max_iter = 50;
toll_OpVar = 1e-1;       % largest tolerance for operations variables
toll_Des = 1e-1;         % largest tolerance for design variables
         iter = 1;
         nDes = size(w1_pubDes0, 1);
         nOpVar = size(w1_pubR0,1);

%Node 1                          `           %Node 2
% Local copy
N1_pubR_star = w1_pubR0;                     N2_pubL_star = w2_pubL0;
N1_pubDes_star = w1_pubDes0;                 N2_pubDes_star = w2_pubDes0;

% Global copy
z_Des  = (N1_pubDes_star+N2_pubDes_star)./2;
```

```
z_Oper = (N1_pubR_star+N2_pubL_star)./2;

% Primal Residual
N1_eps_primal_Des   = 10.*ones(nDes,1);
N2_eps_primal_Des = N1_eps_primal_Des;
N1_eps_primal_OpVar = 10.*ones(nOpVar,1);
N2_eps_primal_OpVar = N1_eps_primal_OpVar;

%Dual Residual
eps_dual_Des = 10.*zeros(nDes,1);
eps_dual_OpVar = 10.*zeros(nOpVar,1);

%Lagrange Multiplier
N1_lambda_Des   = 1.*zeros(nDes,1);
N2_lambda_Des = N1_lambda_Des;
N1_lambda_OpVar = -10.*ones(nOpVar,1);
N2_lambda_OpVar = N1_lambda_OpVar;

% Collecting for plotting etc
N1_pubR_kPlot(iter,:)   =  N1_pubR_star';
N2_pubL_kPlot(iter,:)     =  N2_pubL_star';
N1_pubDes_kPlot(iter,:) = N1_pubDes_star';
N2_pubDes_kPlot(iter,:)   =  N2_pubDes_star';
z_Des_kPlot(iter,:) =   z_Des';
z_Oper_kPlot(iter,:) = z_Oper';
N1_eps_primal_Plot(1,:) = [N1_eps_primal_Des',  N1_eps_primal_OpVar'];
N2_eps_primal_Plot(1,:) = [N2_eps_primal_Des', N2_eps_primal_OpVar'];
eps_dual_Plot(iter,:) = [eps_dual_Des'; eps_dual_OpVar'];
N1_lambda = [N1_lambda_Des; N1_lambda_OpVar];
N2_lambda =  [N2_lambda_Des; N2_lambda_OpVar];
N1_lambda_plot(1,:) = N1_lambda';
N2_lambda_plot(1,:) = N2_lambda';
N1_f_opt(1) = NaN;
N2_f_opt(1) = NaN;
N1_Aug_Lagrange(1) = NaN;
N2_Aug_Lagrange(1) = NaN;
rho_Plot(1) = rho;
```

```
while ( max(abs(N1_eps_primal_OpVar)) > toll_OpVar || ...
max(abs(N1_eps_primal_Des)) > toll_Des || ...
max(abs(N2_eps_primal_OpVar)) > toll_OpVar || ...
max(abs(N2_eps_primal_Des)) > toll_Des     || ...
max(abs(eps_dual_Des)) > toll_Des || ...
max(abs(eps_dual_OpVar)) > toll_OpVar ) && ...
(iter < max_iter )

%% Solve Node 1

%Building the Augmented Lagrange Function
J = J1;

% Adding Penalty for Design Violation
for i = 1 : nDes
J = J    + N1_lambda_Des(i)* (w1_pubDes{1}(i) - z_Des(i) ) ...
+ rho/2*            (w1_pubDes{1}(i) - z_Des(i) )^2;
end

% Adding Penalty for variable violation
indx = 1;
for i = 1: size(w1_pubR,2)          %Running through all cells
for j = 1: size(w1_pubR{i},1)       %Running through each element inside cell
J = J   + N1_lambda_OpVar(indx,1)*(w1_pubR{i}(j) - z_Oper(indx))...
+ rho/2*                   (w1_pubR{i}(j) - z_Oper(indx))^2;
indx = indx+1;
end
end

% structure for optimizer
N1_prob = struct('x',vertcat(w1_pubDes{:},w1_pubL{:},w1{:},w1_pubR{:}),...
      'g', vertcat(g1{:}), ...
      'p', vertcat(p1_NLP{:}), ...
      'f', J );
% creating solver
N1_solver = nlpsol('solver', 'ipopt', N1_prob, opts);

% Solve the NLP
```

```
N1_sol = N1_solver('x0', vertcat(w1_pubDes0,w1_pubL0, w1_w0, w1_pubR0),...
      'lbx', vertcat(lbw1_pubDes,lbw1_pubL,lbw1,lbw1_pubR),...
      'ubx', vertcat(ubw1_pubDes,ubw1_pubL,ubw1,ubw1_pubR),...
      'lbg', lbg1, 'ubg', ubg1, ...
      'p', p1_NLP0);

flag1 = N1_solver.stats();
N1_w_opt = full(N1_sol.x);
N1_lambda_opt = full(N1_sol.lam_x);
N1_Aug_Lagrange(iter+1,1) = full(N1_sol.f);

%Extracting results
[N1_pubDes_star, N1_pubL_star, N1_w_star, N1_pubR_star, ~,~,~,~,~,~] = ...
      extract_NodeSolution(N1_w_opt,N1_lambda_opt,N1_Data,W1_0,par_sim);

% Calculating the value of penalty used in this iteration
N1_penalty = 0;
% Penalty value for Design Violation
for i = 1 : nDes
N1_penalty = N1_penalty +N1_lambda_Des(i)*(N1_pubDes_star(i)-z_Des(i))...
       +rho/2*           (N1_pubDes_star(i)-z_Des(i))^2;
end
% Penalty for Operations Variable
for i = 1 : size(N1_pubR_star,1)
N1_penalty = N1_penalty+N1_lambda_OpVar(i,1)*(N1_pubR_star(i)-z_Oper(i))...
   +rho/2*               (N1_pubR_star(i)-z_Oper(i))^2;
end

N1_f_opt(iter+1) = N1_Aug_Lagrange(iter+1) - N1_penalty;

%% Solve Node 2
%Building the Augmented Lagrange Function
J = J2;
% Penalty for Design Violation
for i = 1 : nDes
J = J    + N2_lambda_Des(i)*     (z_Des(i) - w2_pubDes{1}(i) ) ...
    + rho/2*                (z_Des(i) - w2_pubDes{1}(i) )^2;
end
```

```
% Penalty for variable violation
indx = 1;
for i = 1: size(w2_pubL,2)          %Running through all cells
for j = 1: size(w2_pubL{i},1)   %Running through each element inside cell
J = J   + N2_lambda_OpVar(indx,1)*  (z_Oper(indx) - w2_pubL{i}(j) )...
        + rho/2*                    (z_Oper(indx) - w2_pubL{i}(j) )^2;
indx = indx + 1;
end
end

% structure for optimizer
N2_prob = struct('x', vertcat(w2_pubDes{:}, w2_pubL{:}, w2{:}, w2_pubR{:}),...
         'g', vertcat(g2{:}), ...
         'p', vertcat(p2_NLP{:}), ...
         'f', J );
% creating solver
N2_solver = nlpsol('solver', 'ipopt', N2_prob, opts);

% Solve the NLP
N2_sol = N2_solver('x0',  vertcat(w2_pubDes0, w2_pubL0, w2_w0, w2_pubR0), ...
       'lbx', vertcat(lbw2_pubDes, lbw2_pubL, lbw2, lbw2_pubR), ...
       'ubx', vertcat(ubw2_pubDes, ubw2_pubL, ubw2, ubw2_pubR), ...
       'lbg', lbg2, 'ubg', ubg2, ...
       'p',   p2_NLP0);

flag2 = N2_solver.stats();
N2_w_opt = full(N2_sol.x);
N2_lambda_opt = full(N2_sol.lam_x);
N2_Aug_Lagrange(iter+1,1) = full(N2_sol.f);   %Augmented Lagrange function

%Extracting Solution Values
[N2_pubDes_star, N2_pubL_star, N2_w_star, N2_pubR_star, ~,~,~,~,~,~] = ...
      extract_NodeSolution(N2_w_opt,N2_lambda_opt,N2_Data,W2_0,par_sim);

% Calculating the value of penalty used in this iteration
N2_penalty = 0;
```

```
% Penalty for Design Violation
for i = 1 : nDes
N2_penalty =N2_penalty+N2_lambda_Des(i)*(z_Des(i)-N2_pubDes_star(i)) ...
                      +rho/2*           (z_Des(i)-N2_pubDes_star(i))^2;
end
% Penalty for Operations Variable
for i = 1   :   size(N2_pubL_star,1)
N2_penalty =N2_penalty+N2_lambda_OpVar(i,1)*(z_Oper(i)-N2_pubL_star(i))...
            + rho/2*                         (z_Oper(i) - N2_pubL_star(i))^2;
end

N2_f_opt(iter+1) = N2_Aug_Lagrange(iter+1) - N2_penalty;

%% Lambda Update Step
% Update global copy
z_Des  = (N1_pubDes_star + N2_pubDes_star)./2;
z_Oper = (N1_pubR_star + N2_pubL_star)./2;

% Update Dual Residual
eps_dual_Des         = rho*(z_Des   - z_Des_kPlot(iter,:)');
eps_dual_OpVar       = rho*(z_Oper  - z_Oper_kPlot(iter,:)');

% Primal Residual
N1_eps_primal_Des   = N1_pubDes_star - z_Des;
N2_eps_primal_Des   =  z_Des -  N2_pubDes_star;
N1_eps_primal_OpVar = N1_pubR_star   - z_Oper;
N2_eps_primal_OpVar =  z_Oper - N2_pubL_star;

% Update Lambda
N1_lambda_Des = N1_lambda_Des     + rho.*(N1_eps_primal_Des);
N2_lambda_Des   = N2_lambda_Des   + rho.*(N2_eps_primal_Des);
N1_lambda_OpVar = N1_lambda_OpVar + rho.*(N1_eps_primal_OpVar);
N2_lambda_OpVar = N2_lambda_OpVar + rho.*(N2_eps_primal_OpVar);


% Warm Starting Solver for next iteration
w1_pubDes0  = N1_pubDes_star;       w2_pubDes0  = N2_pubDes_star;
w1_pubL0    = N1_pubL_star;         w2_pubL0    = N2_pubL_star;
```

```
w1_w0       = N1_w_star;            w2_w0       = N2_w_star;
w1_pubR0    = N1_pubR_star;         w2_pubR0    = N2_pubR_star;

%% Extention - varying rho
if iter <= max_iter
if  N1_eps_primal_Plot(iter,:)*N1_eps_primal_Plot(iter,:)' > ...
10*(eps_dual_Plot(iter,:)*eps_dual_Plot(iter,:)')  ...
N2_eps_primal_Plot(iter,:)*N2_eps_primal_Plot(iter,:)' > ...
10*(eps_dual_Plot(iter,:)*eps_dual_Plot(iter,:)')
rho = 2*rho;
   disp('rho increased');
elseif eps_dual_Plot(iter,:)*eps_dual_Plot(iter,:)' > ...
           10*(N1_eps_primal_Plot(iter,:)*N1_eps_primal_Plot(iter,:)') || ...
       eps_dual_Plot(iter,:)*eps_dual_Plot(iter,:)' > ...
           10*(N2_eps_primal_Plot(iter,:)*N2_eps_primal_Plot(iter,:)')
   rho = rho/2;
   disp('rho decreased');
else
rho;
disp('rho unchanged');
end

% else
% disp('rho should be stabilized');
% end

%% Collect for Plotting etc

N1_pubR_kPlot(iter+1,:)    = N1_pubR_star';
N2_pubL_kPlot(iter+1,:)     = N2_pubL_star';
N1_pubDes_kPlot(iter+1,:)  = N1_pubDes_star';
N2_pubDes_kPlot(iter+1,:)   = N2_pubDes_star';

z_Des_kPlot(iter+1,:)  = z_Des';
z_Oper_kPlot(iter+1,:) = z_Oper;

N1_eps_primal_Plot(iter+1,:) = [N1_eps_primal_Des', N1_eps_primal_OpVar'];
N2_eps_primal_Plot(iter+1,:) = [N2_eps_primal_Des', N2_eps_primal_OpVar'];
```

```
N1_lambda_plot(iter+1,:)     = [N1_lambda_Des', N1_lambda_OpVar'];
N2_lambda_plot(iter+1,:)     = [N2_lambda_Des', N2_lambda_OpVar'];

eps_dual_Plot(iter+1,:)      = [eps_dual_Des', eps_dual_OpVar'];
rho_Plot(iter+1) = rho;

% Update iteration counter for Decomposition
iter = iter + 1;

end

%% ADMM Iterations Summary Data

distr_cost = N1_f_opt(iter-1) + N2_f_opt(iter-1);
%for pausing - if any subproblem failed
if  (flag1.success + flag2.success) == Num_nodes && (iter < max_iter)
disp(['Converged by decomposition in' num2str(iter) 'iterations']);
disp(['Optimal Volume is ' num2str(N1_pubDes_star(1)) '/' ...
                           num2str(N2_pubDes_star(1)) ' m3']);
disp(['Optimal Area is ' num2str(N1_pubDes_star(2)) '/' ...
                         num2str(N2_pubDes_star(2)) ' m2']);
disp(['Optimal Cost is ' num2str(distr_cost) ' USD']);
else
disp(['Decomposition not converged in ' num2str(iter) ' iterations'])
disp(['Optimal Volume is ' num2str(N1_pubDes_star(1)) '/ ' ...
                           num2str(N2_pubDes_star(1)) ' m3']);
disp(['Optimal Area is '    num2str(N1_pubDes_star(2)) '/ ' ...
                            num2str(N2_pubDes_star(2)) ' m2']);
disp(['Optimal Cost is ' num2str(distr_cost) ' USD']);
disp(['N1 ' flag1.return_status])
disp(['N2 ' flag2.return_status])
end

itergrid = linspace(1, iter, iter)';

%% Plotting ADMM

figure(3)
```

```
subplot(2,2,1)  %Residuals - Prmal
semilogy(itergrid, abs(N1_eps_primal_Plot(:,nDes+1:end)),'o-b')%Operational
hold on
xlabel('iteration');
semilogy(itergrid, abs(N1_eps_primal_Plot(:,1:nDes)),'b')       %Design
hold on

%         yyaxis right
%         plot(itergrid, N2_eps_primal_Plot(:,1:nDes),'r')
%         hold on
%         plot(itergrid, N2_eps_primal_Plot(:,nDes+1:end),'-.r')
legend('eps Primal - V tes', 'eps Primal - T tes');
title(['Primal Infeasibility. rho = ', num2str(rho)]);

subplot(2,2,2)                                                  %Dual Residuals
semilogy(itergrid, abs(eps_dual_Plot(:,1:nDes)),'b')
hold on;
semilogy(itergrid, abs(eps_dual_Plot(:,nDes+1:end)),'-ob')
xlabel('itearation');
%                 yyaxis right
%                 plot(itergrid, eps_dual_Plot(:,1:nDes),'r')
%                 hold on;
%                 plot(itergrid, eps_dual_Plot(:,nDes+1:end),'-.r')
legend('eps Dual - Vtes', 'eps Dual - T tes')
title(['Dual Infeasibility. rho = ', num2str(rho)]);

subplot(2,2,3)  %Lambdas
plot(itergrid, N1_lambda_plot(:,1:nDes),'b')
hold on
plot(itergrid, N1_lambda_plot(:,nDes+1:end),'-.b')
hold on
plot(itergrid, N2_lambda_plot(:,1:nDes),'r')
hold on
plot(itergrid, N2_lambda_plot(:,nDes+1:end),'-.r')
%             legend('lambda Vol', 'lambda A whb', 'lambda T tes')
title(['Lambdas. rho = ', num2str(rho)]);

%     subplot(2,2,4)  %Augmented Lagrange Function value
```

```
%         plot(itergrid, N1_Aug_Lagrange,'b-.*')
%         hold on
%         plot(itergrid, N2_Aug_Lagrange,'r-.*')
%         hold on
%         plot(itergrid, N1_f_opt, 'b');
%         hold on
%         plot(itergrid, N2_f_opt, 'r');
%     legend('N1 Aug Lagrange', 'N2 Aug Lagrange', 'N1 Opt f', 'N2 Opt f')
%     title(['Aug Lag vs Objective. rho = ', num2str(rho)]);

subplot(2,2,4)  %Augmented Lagrange Total
plot(itergrid, N1_Aug_Lagrange + N2_Aug_Lagrange, 'r-.o');
hold on
plot(itergrid, 186.7.*ones(iter,1),'g--');
legend('Augmented Lagrangian', 'Original Optimum');
title(['Objective Value. rho = ', num2str(rho)]);

figure(4)   %Design Variables
subplot(2,2,1)  %Primal Value
plot(itergrid, N1_pubDes_kPlot(:,1),'b')       %Vol
hold on
plot(itergrid, N2_pubDes_kPlot(:,1),'r')
title(['Volume. rho = ', num2str(rho)]);

subplot(2,2,4)
plot(itergrid, rho_Plot)
title('rho plot');

figure(5)   %Operational Variables
subplot(2,2,1)
plot(itergrid, N1_pubR_kPlot(:,1), 'bo-')    %Ttes
hold on
plot(itergrid, N2_pubL_kPlot(:,1), 'ro-')
legend('Node 1 optimal value','Node 2 optimal value')
title('Loacal Variable T tes in each iteration');
xlabel('iteration'); ylabel('Temperature [deg C]');
subplot(2,2,2)
%         plot(itergrid, N1_pubR_kPlot(:,2), 'b')    %Tphb
```

```
%         hold on
%         plot(itergrid, N2_pubL_kPlot(:,2), 'r')
title(['T phb. rho = ', num2str(rho)]);
subplot(2,2,3)
%         plot(itergrid, N1_pubR_kPlot(:,3:5), '')    %Twhb
%         hold on
%         plot(itergrid, N2_pubL_kPlot(:,3:5), '')
title(['T whb. rho = ', num2str(rho)]);
subplot(2,2,4)
%         plot(itergrid, N1_pubR_kPlot(:,6:8), '')    %Twhb ret
%         hold on
 %         plot(itergrid, N2_pubL_kPlot(:,6:8), '')
title(['T whb ret. rho = ', num2str(rho)]);

plot_NodeProfile(N1_w_opt,N1_lambda_opt,N1_Data,W1_0)
plot_NodeProfile(N2_w_opt,N2_lambda_opt,N2_Data,W2_0)

end
```

A1.b. Specifying bounds for all the variables

The function **`Initialization_bounds.m`** is used to specify the upper and lower bounds for all the variables in the optimization problem.

```
function [x0,z0,u0,Des_var0,lbx,lbz,lbu,lbDes_var,ubx,ubz,ubu,ubDes_var] =
Initialization_bounds()
global par_model;
N = par_model.Ncell_whb;
lb_Tphb = par_model.T_dh_minSup;
ub_T_dc = par_model.T_wh_minRet;

%Initial Guesses
x0 = [55];                    %T_tes
z0 = [55;56]  ;               %T_b,T_phb
u0 = [0;0];                   %alpha,Q_Phb
Des_var0 = [10000];           %V_tes
```

```
%Bounds
ubx = [100];                    %T_tes,T_phb
lbx = [30];

ubz = [100 ;100   ];            %T_b
lbz = [30 ;lb_Tphb ];

ubu = [1;1500000];
lbu = [0;0];

ubDes_var = [25000];
lbDes_var = [5000];
end
```

A1.c. Specifying the Dynamic model as DAE

The function **`myfunc_Integrator.m`** is used to specify the dynamic model of the TES system.

```
function [F,x_var, z_var, p_var, alg, diff, L] = myfunc_Integrator()

import casadi.*
global par_model;
global par_sim;

% Integration Final Time
 tf = par_sim.dt;   %Hours

%Constants from parameter Structures
 rho_dh = par_model.rho_dh;   %kg/m3
 Cp_dh = par_model.Cp_dh;     %KJ/(kg* K)
 rho_wh = par_model.rho_wh;
 Cp_wh = par_model.Cp_wh;
 T_dh_Ret = par_model.T_dh_Ret;     %Deg C
 T_wh_Sup = par_model.T_wh_Sup;

 V_whb_dh = par_model.V_whb_dh;
 V_whb_wh = par_model.V_whb_wh;
```

```
 Ncell_whb = par_model.Ncell_whb;
 U_whb = par_model.U_whb;
 coef_Qphb = par_model.coef_Qphb;
 coef_Qdc = par_model.coef_Qdc;

 V_phb = par_model.V_phb_dh;
 alpha_loss = par_model.alpha_loss;
 T_amb = par_model.T_amb;

%% Declaring Scaled CasADi variables
%Differential States
T_tes = MX.sym('T_tes',1);

%Algebraic States
T_b = MX.sym('T_b',1);
T_phb = MX.sym('T_phb',1);

% Design Variables
V_tes = MX.sym('V_tes',1);
%     V_tes = 11000;

% control input
alpha = MX.sym('alpha', 1);
Q_phb = MX.sym('Q_phb',1); %KW

%% Unscaled Variables for use in Equations
[~,~,~,~,lbx,lbz,lbu,lbDes_var,ubx,ubz,ubu,ubDes_var] =
Initialization_bounds();

% Differential States
   T_tes       = T_tes*       (ubx(1) - lbx(1)) + lbx(1);
%Algebraic States
   T_b         = T_b_*         (ubz(1) - lbz(1)) + lbz(1) ;
   T_phb       = T_phb_*       (ubz(2) - lbz(2)) + lbz(2);
% Design Variables
   V_tes       = V_tes_*       (ubDes_var(1) - lbDes_var(1)) + lbDes_var(1);
% control input
   alpha       = alpha_*       (ubu(1) - lbu(1)) + lbu(1) ;
```

```
    Q_phb       = Q_phb_*       (ubu(2) - lbu(2)) + lbu(2) ;

% Time Varying Parameters
q_dh = MX.sym('q_dh_',1);    %m3/hr
Q_whb = MX.sym('Q_whb_',Ncell_whb);

% Intermediate Casadi Variables
T_whb = T_dh_Ret + (Q_whb*3600)/(rho_dh*Cp_dh*q_dh);

%% Model Equations Unscaled
%All equations in per hour basis
%Differential Equations
dT_tes = (q_dh*alpha*(T_whb - T_tes))/(V_tes)  - alpha_loss*(T_tes -
T_amb)/(V_tes*rho_dh*Cp_dh)  ;
% dT_phb = (q_dh*(T_b - T_phb)/V_phb) + (Q_phb*3600)/(V_phb*rho_dh*Cp_dh);

%Algebraic Equations
alg1 = alpha*T_tes + (1-alpha)*T_whb  - T_b ;
alg2 = (q_dh*(T_b - T_phb)/V_phb) + (Q_phb*3600)/(V_phb*rho_dh*Cp_dh);

%% Model Equations scaled
%     dT_tes_ = dT_tes/       (ubx(1) - lbx(1));

%% Stacking variables and Parameters

diff = vertcat(dT_tes);
alg = vertcat(alg1,alg2);

x_var = vertcat(T_tes);              %Differential States
z_var = vertcat(T_b,T_phb);          %Algebraic States

    u_var = vertcat(alpha,Q_phb);   %Operation handles (Manipulated Variables)
    Des_var = vertcat(V_tes);                %Design variables
    Dist_var = vertcat(q_dh,Q_whb);          %Disturbance values
p_var = vertcat(u_var,Des_var,Dist_var); %Parameters to be input to integrator

% Operational Objective to be minimized
```

```
L = (Q_phb*coef_Qphb);

%% Creating the Integrator

dae = struct('x',x_var,'z',z_var,'p',p_var,'ode',diff,'alg',alg,'quad',L);
opts = struct('tf',tf);

% create IDAS integrator
F = integrator('F','idas',dae,opts);

end
```

A1.d. Specifying parameters

The function **`param_model.m`** is used to specify the constant parameters used in the problem.

```
function par = param_model()

%% District Heating Parameters
    par.rho_dh = 1000;      %kg/m3
    par.Cp_dh = 4;          %KJ/(kg* K)
    par.T_dh_Ret = 30;      %Deg C
    par.T_dh_minSup = 55;

%% Waste Heat Stream Parameters
    par.rho_wh = 1000;
    par.Cp_wh = 2;
    par.T_wh_Sup = 100;
    par.T_wh_minRet = 50;

%% Waste Heat Boiler
    par.V_whb_wh = 12;
    par.V_whb_dh = 12;
    par.U_whb = 5.13;
    par.Ncell_whb = 1;
    par.A_whb = 33;
```

```
%% Peak Heat Boiler
    par.V_phb_dh = 12;

%% Cost factors
        par.coef_Vtes = (700)/(365*10)*3;               %USD/m3
        par.coef_Awhb = (880)/(365*10)*4;               %USD/m2
        par.coef_Qphb = (0.06)*1;                       %USD/kWh
        par.coef_Qdc = par.coef_Qphb/10;

%% Heat Loss
    par.alpha_loss = 0;
    par.T_amb = 10;
end
```

A1.e. Specifying Disturbance profiles

The function **`Generate_Profiles.m`** is used to specify the individual scenarios.

```
function [disturbance, constraint] = Generate_Profiles()
%Generates profiles for Disturbances and constraints.
%% Initialize some data from parameters
global par_sim;
global par_model;

    Num_nodes = par_sim.Num_nodes;
    Nsim = par_sim.Nsim;
    overlap = par_sim.Node_overlap;
    Tsim = par_sim.Tsim;
    dt = par_sim.dt;
    rho_wh = par_model.rho_wh;
    rho_dh = par_model.rho_dh;
    Cp_wh = par_model.Cp_wh;
    Cp_dh = par_model.Cp_dh;
    T_wh_Sup = par_model.T_wh_Sup;
    T_dh_Ret = par_model.T_dh_Ret;
    T_dh_minSup0 = par_model.T_dh_minSup;
    T_wh_minRet0 = par_model.T_wh_minRet;
```

```
%First Point Values
q_dh0 = 3600*0.3;              %m3/hr
q_wh0 = 3600*0.3;              %m3/hr

% Q_whb_max0 = 17.22488038;       %KW
Q_Supply0 = rho_wh*(q_wh0/3600)*Cp_wh*(T_wh_Sup - T_wh_minRet0);
Q_Demand0 = rho_dh*(q_dh0/3600)*Cp_dh*(T_dh_minSup0 - T_dh_Ret);

%% Build Complete Profiles
n = Nsim;

%% 20%1n/2u/2d/1n  30%1n/2u/2d/1n
    q_dh = q_dh0.*ones(n,1);
    q_wh = [q_wh0.*ones(n/12,1);q_wh0*1.2.*ones(2*n/12,1);
          q_wh0*0.8.*ones(2*n/12,1);q_wh0.*ones(n/12,1);
          q_wh0.*ones(n/12,1);q_wh0*1.3.*ones(2*n/12,1);
          q_wh0*0.7.*ones(2*n/12,1);q_wh0.*ones(n/12,1)];
    T_dh_minSup = T_dh_minSup0.*ones(n,1);
    T_wh_minRet = T_wh_minRet0.*ones(n,1);
        Q_Supply = rho_wh*Cp_wh*(q_wh/3600).*(T_wh_Sup - T_wh_minRet);
        Q_Demand = rho_dh*Cp_dh*(q_dh/3600).*(T_dh_minSup0 - T_dh_Ret);

%% From Mo Data
  q_dh = q_dh0.*ones(n,1);
   q_wh = q_wh0.*Read_MoData();
   T_dh_minSup = T_dh_minSup0.*ones(n,1);
   T_wh_minRet = T_wh_minRet0.*ones(n,1);
       Q_Supply = rho_wh*Cp_wh*(q_wh/3600).*(T_wh_Sup - T_wh_minRet);
       Q_Demand = rho_dh*Cp_dh*(q_dh/3600).*(T_dh_minSup0 - T_dh_Ret);

%% Display Complete Profile
t_plot = linspace(0,Tsim-dt,Nsim)';

figure(1)
stairs(t_plot, Q_Supply, '-r');
    xlabel('time [hrs]'); ylabel('Duty [kW]');
legend('Q whb');  title('Disturbance Profile');
```

```
%% Picking from complete profile
if Num_nodes == 1
% If Centralized
par_sim.Node_Nsim = Nsim;

        % Profile for time in Central node (in Cell structure)
        Tsim_Grid{1} = linspace(0,Tsim-dt,Nsim)';
%% Collecting profiles into Structure
      disturbance.q_dh{1} = q_dh;
        disturbance.Q_Supply{1} = Q_Supply;
        constraint.Q_Demand{1} = Q_Demand;
        constraint.T_dh_minSup{1} = T_dh_minSup;
        constraint.T_wh_minRet{1} = T_wh_minRet;
else

%% If Distributed Optimization
Node_Nsim = (Nsim/Num_nodes+overlap).*ones(1,Num_nodes);
%Initial vector of number of elements in each node
    Node_Nsim(1) = Node_Nsim(1) - overlap;
    par_sim.Node_Nsim = Node_Nsim;

        % Picking from full profile
        nstart = 1;
        for j = 1:Num_nodes
            nend = nstart + Node_Nsim(j)-1;

               Tsim_Grid{j} = t_plot(nstart:nend);

               disturbance.q_dh{j}  = q_dh(nstart:nend);
               disturbance.Q_Supply{j} = Q_Supply(nstart:nend);
               constraint.Q_Demand{j} = Q_Demand(nstart:nend);
               constraint.T_dh_minSup{j} = T_dh_minSup(nstart:nend);
               constraint.T_wh_minRet{j} = T_wh_minRet(nstart:nend);

            nstart = nend - overlap + 1;

            % Plotting each node Profile
            figure(2)
```

```
            plot(Tsim_Grid{j},  disturbance.Q_Supply{j},
            '--o','MarkerSize',4*j);
            hold on
            xlabel('time in hours');  legend('Q Supply');

        end
end

par_sim.Tsim_Grid = Tsim_Grid;
end
```

A1.f. Building the NLP structure

The function **`create_NodeModel.m`** is generalized to create and return the NLP structure depending on the subproblem that is calling it.

```
function [J, w_pubDes, w_pubL, w, w_pubR, g,p_NLP, W0, LBW, UBW,
lbg,ubg,p_NLP0] = ...
      create_NodeModel(Node_Data)
import casadi.*
global par_model;
global par_sim;

dist = Node_Data.dist;      %Time varying Disturbance
constr = Node_Data.constr;  %Time varying constraints
nodeID = Node_Data.nodeID;

coef_Vtes = par_model.coef_Vtes;    %cost coefficient parameter for Tank
volume
coef_Awhb = par_model.coef_Awhb;
Ncells = par_model.Ncell_whb;

dt = par_sim.dt;                    %hour
Num_nodes = par_sim.Num_nodes;
overlap = par_sim.Node_overlap;     % number of overlapping dt's. 0 is no
overlap
Nopt = par_sim.Node_Nsim(nodeID);   % Number of points for optimization
```

```
problem
N_scale = par_sim.N_scale;

% Generating Initial guesses and bounds
[x0,z0,u0,Des_var0,lbx,lbz,lbu,lbDes_var,ubx,ubz,ubu,ubDes_var] =
Initialization_bounds();
        %Finding the size of each Diff state, Alg state, MV
        nx = size(x0,1);
        nz = size(z0,1);
        nu = size(u0,1);                    %Number of Operational MVs
        ndes = size(Des_var0,1);            %Number of Design Variables
        ndist = length(fieldnames(dist)) ;  %Number of Disturbance variables

% Integrator
[~,x_var, z_var, p_var, alg, diff, L] = myfunc_Integrator();

%Function to return values of diff, alg and L for a given collocation point
(x_var, z_var, p_var)
f = Function('f',{x_var,z_var,p_var},{diff,alg,L},{'x','z','p'}, ...
             {'xdot','zeval','qj'});

%% Direct Collocation Polynomials
d = 3;

% Get collocation points
tau_root = [0, collocation_points(d, 'radau')];

% Coefficients of the collocation equation (xdot = C*x)
C = zeros(d+1,d+1);

% Coefficients of the continuity equation
D = zeros(d+1, 1);

% Coefficients of the quadrature function
B = zeros(d+1, 1);

% Construct polynomial basis
for j=1:d+1
```

```
    % Construct Lagrange polynomials to get the polynomial basis at the
collocation point
    coeff = 1;
    for r=1:d+1
        if r ~= j
            coeff = conv(coeff, [1, -tau_root(r)]);
            coeff = coeff / (tau_root(j)-tau_root(r));
        end
    end
    % Evaluate the polynomial at the final time to get the coefficients of the
continuity equation
    D(j) = polyval(coeff, 1.0);

    % Evaluate the time derivative of the polynomial at all collocation points
to get the coefficients of the continuity equation
    pder = polyder(coeff);
    for r=1:d+1
        C(j,r) = polyval(pder, tau_root(r));
    end

    % Evaluate the integral of the polynomial to get the coefficients of the
quadrature function
    pint = polyint(coeff);
    B(j) = polyval(pint, 1.0);
end

%% Build NLP solver
% Empty NLP
w = {};     w0 = [];        lbw = [];         ubw = [];
%Variables for NLP solver - Private (excl public facing ones)

%Variables for NLP solver - Public
w_pubDes = {};  w_pubDes0 = [];     lbw_pubDes = [];    ubw_pubDes = [];    %
Public variable - Design Parameters
w_pubL = {};    w_pubL0 = [];       lbw_pubL = [];      ubw_pubL = [];      %
Public Variables on left edge
w_pubR = {};    w_pubR0 = [];       lbw_pubR = [];      ubw_pubR = [];      %
Public Variables on right edge
```

```
J = 0;                                                  %Objective for NLP
Solver
g = {};     lbg = [];       ubg = [];                   %Constraints for NLP
Solver
p_NLP = {}; p_NLP0 = [];                                %parameters for NLP
solver

    %Declaring Design Variables
    V_TES = MX.sym('V_TES',1);
%         %Collecting the Design Variables
        Des_var     = vertcat(V_TES);
        w_pubDes    = {w_pubDes{:}, Des_var};
        lbw_pubDes  = [lbw_pubDes; lbDes_var];
        ubw_pubDes  = [ubw_pubDes; ubDes_var];
        w_pubDes0   = [w_pubDes0; Des_var0];

%     %Capital Cost (normalized for number of nodes)
    J = J + (coef_Vtes*((V_TES)^0.7))/Num_nodes;

% "Lift" initial conditions
X0 = MX.sym('X0',nx);
w_pubL = {w_pubL{:}, X0};
    % If first node - then initial state is fixed at x0. Else - is a variable
for optimizer.
    if Node_Data.nodeID == 1
        lbw_pubL = [lbw_pubL; x0];
        ubw_pubL = [ubw_pubL; x0];
    else
        lbw_pubL = [lbw_pubL; lbx];
        ubw_pubL = [ubw_pubL; ubx];
    end
w_pubL0 = [w_pubL0; x0];

Xk = X0;    % for linking the initial state to integrator inside collocations
%% Building the NLP
js = 1;
    for k=0:(Nopt-1)
```

```
%Time Varying Disturbances (Dk)
        Dk = MX.sym(['D_' num2str(k) '_' num2str(js)],ndist);
        p_NLP = {p_NLP{:},Dk};
        p_NLP0 = [p_NLP0;dist.q_dh{nodeID}(k+1); dist.Q_Supply{nodeID}(k+1)];

        % Calculatin the steady state point to give as better initial Guess
for NLP
%         [x0, ~, ~] = find_a_Steady_State(
[dist.q_dh{nodeID}(k+1);dist.q_wh{nodeID}(k+1)] );


        % Control Input (Uk)
        Uk = MX.sym(['U_' num2str(k) '_' num2str(js)],nu);

        % If inside overlap region - adding to public variable structure
        if k <= (overlap - 1)                    %Left side Public Variable
            w_pubL   = {w_pubL{:}, Uk};
            lbw_pubL = [lbw_pubL; lbu];
            ubw_pubL = [ubw_pubL; ubu];
            w_pubL0  = [w_pubL0; u0];
        elseif k >= (Nopt - overlap)             %Right side Public Variable
            w_pubR   = {w_pubR{:}, Uk};
            lbw_pubR = [lbw_pubR; lbu];
            ubw_pubR = [ubw_pubR; ubu];
            w_pubR0  = [w_pubR0; u0];
        else                                     %Private Variables
            w   = {w{:},Uk};
            lbw = [lbw;lbu];
            ubw = [ubw;ubu];
            w0  = [w0;u0];
        end

        % Declaring New Collocation variables that are handles for solver
        Xkj = {};
        Zkj = {};
            for j = 1:d
                Xkj{j} = MX.sym(['X_' num2str(k) '_' num2str(j) '_'
num2str(js)],nx);
```

```
                Zkj{j} = MX.sym(['Z_' num2str(k) '_' num2str(j) '_'
num2str(js)],nz);

                if k <= (overlap-1)                     %Left Public
                    w_pubL   = {w_pubL{:}, Xkj{j},Zkj{j}};
                    lbw_pubL = [lbw_pubL; lbx;lbz];
                    ubw_pubL = [ubw_pubL; ubx;ubz];
                    w_pubL0  = [w_pubL0; x0;z0];
                elseif k >= (Nopt - overlap)            %Right Public
                    w_pubR   = {w_pubR{:}, Xkj{j},Zkj{j}};
                    lbw_pubR = [lbw_pubR; lbx;lbz];
                    ubw_pubR = [ubw_pubR; ubx;ubz];
                    w_pubR0  = [w_pubR0; x0;z0];
                else                                    %Private Variable
                    w   = {w{:},Xkj{j},Zkj{j}};
                    lbw = [lbw; lbx;lbz];
                    ubw = [ubw; ubx;ubz];
                    w0  = [w0; x0;z0];
                end
            end

%% Loop over collocation points
      Xk_end = D(1)*Xk;
            for j = 1:d

                % Expression for the state derivative of the collocation point
j
             % (collocation equation RHS i.e xdot = C*x)
                xp = C(1,j+1)*Xk;  % helper state
                for r = 1:d
                    xp = xp + C(r+1,j+1)*Xkj{r};
                end
                %Calculating the diff,alg and quadrature @ collocation point
                [fj,zj,qj] =  f(Xkj{j}, Zkj{j}, vertcat(Uk, Des_var, Dk) );

                % dynamic and algebraic constraints must satisfy
                g = {g{:}, dt*fj-xp, zj};
                lbg = [lbg; zeros(nx,1); zeros(nz,1)];
```

```
                ubg = [ubg; zeros(nx,1); zeros(nz,1)];

%% Constraints to prevent Temp Cross
g = {g{:}, Xkj{j}(8)-Xkj{j}(3), Xkj{j}(7)-Xkj{j}(4),    Xkj{j}(6)-Xkj{j}(5)
};
              lbg = [lbg; zeros(Ncells,1)];
              ubg = [ubg; 80.*ones(Ncells,1)];

%% Quadrature
%Adding the OPEX for this step (quadrature at the last collocation point)
               J = J + (B(d+1)*qj*dt) ;

               % Add contribution to the end states
               Xk_end = Xk_end + D(j+1)*Xkj{j};
           end

%% Reached last collocation point
        % Imp -> Add any other albebraic constraint here :(on last collocation
point)
        %Inequality of T_phb > T_dh_minSup
%         g   = {g{:}, Zkj{d}(2)};
%         lbg = [lbg; constr.T_dh_minSup{nodeID}(k+1)];
%         ubg = [ubg; 100];
%         %Inequality of T_wh_ret > T_wh_minRet
%         g   = {g{:}, Xkj{d}(3)};
%         lbg = [lbg; constr.T_wh_minRet{nodeID}(k+1)];
%         ubg = [ubg; 200];

        % New NLP variable for state at end of interval
        Xk = MX.sym(['X_' num2str(k+1) '_' num2str(js)], nx);
            %If it's not last state -> stored in private variable structure.
Else in public variable structure
            if k <= (overlap-1)             %Left Public
                w_publ   = {w_publ{:},Xk};
                lbw_publ = [lbw_publ;lbx];
                ubw_publ = [ubw_publ;ubx];
                w_publ0  = [w_publ0; x0];
            elseif k >= (Nopt-1 - overlap)  %Right Public
```

```
                w_pubR   = {w_pubR{:},Xk};
                lbw_pubR = [lbw_pubR;lbx];
                ubw_pubR = [ubw_pubR;ubx];
                w_pubR0  = [w_pubR0; x0];
            else                              %Private Variable
                w   = {w{:},Xk};
                lbw = [lbw;lbx];
                ubw = [ubw;ubx];
                w0  = [w0; x0];
            end

        % Shooting Gap constraint
        g   = {g{:},Xk_end-Xk};
        lbg = [lbg;zeros(nx,1)];
        ubg = [ubg;zeros(nx,1)];
    end

    %Collecting numeric limits into a structure to send out
    W0.w_pubDes0 = w_pubDes0;       W0.w_pubL0 = w_pubL0;       W0.w0 = w0;
W0.w_pubR0 = w_pubR0;
    LBW.lbw_pubDes = lbw_pubDes;    LBW.lbw_pubL = lbw_pubL;    LBW.lbw = lbw;
LBW.lbw_pubR = lbw_pubR;
    UBW.ubw_pubDes = ubw_pubDes;    UBW.ubw_pubL = ubw_pubL;    UBW.ubw = ubw;
UBW.ubw_pubR = ubw_pubR;
end
```

A1.g. Plotting optimum profiles

The function **`plot_NodeProfile.m`** is generalized to plot the solution of the subproblem that calls it.

```
function [] = plot_NodeProfile(w,lambda,Node_Data,W0 )
global par_sim;
global par_model;

    dist = Node_Data.dist;      %Time varying Disturbance
    constr = Node_Data.constr;  %Time varying constraints
```

```
    nodeID = Node_Data.nodeID;
    t_plot = par_sim.Tsim_Grid{nodeID};
    dt = par_sim.dt;
    par_model = param_model();
    Nwhb = par_model.Ncell_whb;
    U_whb = par_model.U_whb;
    A_whb = par_model.A_whb;
    T_dh_Ret = par_model.T_dh_Ret;
    rho_dh = par_model.rho_dh;
    Cp_dh = par_model.Cp_dh;

%% Extract Node Solution
[~, w_pubL, ~, w_pubR, Des_opt, u_opt, x_opt, z_opt,lam_Des,lam_x] =
                  extract_NodeSolution(w,lambda,Node_Data,W0,par_sim );

    %Design Variables
    V_tes = Des_opt(1);                          lam_V_tes = lam_Des(1);
    %MVs
    alpha = u_opt(:,1);
    Q_phb = u_opt(:,2);
    %Diff States
    T_tes = x_opt(:, 1);                         lam_T_tes = lam_x(:,1);
    %Alg States
    T_b = z_opt(:, 1);
    T_phb = z_opt(:, 2);

    Q_whb = dist.Q_Supply{nodeID}(:,1);
    q_dh = dist.q_dh{nodeID}(:,1);
    T_whb = T_dh_Ret + (Q_whb*3600)./(rho_dh*Cp_dh.*q_dh);

%% Plotting

%TES Overall
    figure(7)
    subplot(2,2,1)
    stairs(t_plot,alpha);       %alpha
    hold on
    xlabel('t in hours');    ylim([-0.1, 1])
```

```
    legend('alpha')

    % Duties
    subplot(2,2,2)
    stairs(t_plot,Q_phb);                %Qphb
    hold on
    xlabel('t in hours');
    legend('Q phb')

    %Temperatures WHB
    subplot(2,2,3)
    plot(t_plot, T_whb, 'o-')  %Twhb
    hold on
    xlabel('time [hrs]'); ylabel('Temperature [Deg C]');
    legend('T whb'); title('Waste Heat Boiler Exit Temp Profile');

    %Temperatures TES circuit
    subplot(2,2,4)
    plot(t_plot, T_whb,'b-o')  %T whb
    hold on
    plot([t_plot; t_plot(end)+ dt], T_tes,'r-o')  %Ttes
       %since states have both x0 and xN, extending tplot
    hold on
    plot(t_plot,T_phb,'mo-')                    %Tphb
    hold on
    xlabel('time [hrs]'); ylabel('Temperature [Deg C]');
    legend('T whb', 'T tes', 'T phb'); title('Optimal State Profiles');
end
```