LARGE-SCALE DYNAMIC OPTIMIZATION UNDER UNCERTAINTY USING PARALLEL COMPUTING

LARGE-SCALE DYNAMIC OPTIMIZATION UNDER UNCERTAINTY USING PARALLEL COMPUTING

by

IAN D. WASHINGTON, B.A.Sc., M.A.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (2016) McMaster University
(Chemical Engineering) Hamilton, Ontario, Canada

TITLE: Large-Scale Dynamic Optimization Under Uncertainty using Parallel Computing

AUTHOR: Ian D. Washington
B.A.Sc. (University of Waterloo, Waterloo, Ontario, Canada)
M.A.Sc. (University of Waterloo, Waterloo, Ontario, Canada)
SUPERVISOR: Dr. Christopher L.E. Swartz

NUMBER OF PAGES: xiii, 158

Abstract

This research focuses on the development of a solution strategy for the optimization of large-scale dynamic systems under uncertainty. Uncertainty resides naturally within the external forces posed to the system or from within the system itself. For example, in chemical process systems, external inputs include flow rates, temperatures or compositions; while internal sources include kinetic or mass transport parameters; and empirical parameters used within thermodynamic correlations and expressions. The goal in devising a dynamic optimization approach which explicitly accounts for uncertainty is to do so in a manner which is computationally tractable and is general enough to handle various types and sources of uncertainty. The approach developed in this thesis follows a so-called multiperiod technique whereby the infinite dimensional uncertainty space is discretized at numerous points (known as periods or scenarios) which creates different possible realizations of the uncertain parameters. The resulting optimization formulation encompasses an approximated expected value of a chosen objective functional subject to a dynamic model for all the generated realizations of the uncertain parameters. The dynamic model can be solved, using an appropriate numerical method, in an embedded manner for which the solution is used to construct the optimization formulation constraints; or alternatively the model could be completely discretized over the temporal domain and posed directly as part of the optimization formulation.

Our approach in this thesis has mainly focused on the embedded model technique for dynamic optimization which can either follow a single- or multiple-shooting solution method. The first contribution of the thesis investigates a combined multiperiod multiple-shooting dynamic optimization approach for the design of dynamic systems using ordinary differential equation (ODE) or differential-algebraic equation (DAE) process models. A major aspect of this approach is the analysis of the parallel solution of the embedded model within the optimization formulation. As part of this analysis, we further consider the application of the dynamic optimization approach to several design and operation applications. Another

major contribution of the thesis is the development of a nonlinear programming (NLP) solver based on an approach that combines sequential quadratic programming (SQP) with an interior-point method (IPM) for the quadratic programming subproblem. A unique aspect of the approach is that the inherent structure (and parallelism) of the multiperiod formulation is exploited at the linear algebra level within the SQP-IPM nonlinear programming algorithm using an explicit Schur-complement decomposition. Our NLP solution approach is further assessed using several static and dynamic optimization benchmark examples.

An accompanying contribution of the thesis is a proof-of-concept dynamic optimization tool, written in `C/C++`, that automatically discretizes infinite dimensional dynamic optimization formulations to multiperiod nonlinear programming formulations. Additionally, all corresponding objective and constraint derivative information is generated via a combination of automatic differentiation and sensitivity analysis. Furthermore, a separate and stand-alone NLP solver, based on an SQP-IPM algorithm, was written in `C++` which we utilized for discretized multiperiod dynamic formulations, but could also be used for general NLPs with a known user-defined structure.

Acknowledgments

The author wishes to thank his advisor, Christopher L.E. Swartz for his support throughout the course of the project. Further gratitude is owed to the McMaster Advanced Control Consortium (MACC) and its industrial partners, as well the Department of Chemical Engineering at McMaster University for project funding.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter outlines and defines the research problem and further highlights the solution path taken. The research goals and contributions are clearly stated.

1.1 Motivation and Goals

Higher operating costs and shrinking profit margins in the chemical and petro-chemical industries are driving greater applications of advanced control techniques and even further consideration of control at the process design stage. These applications are often model-based and require the solution of optimization formulations comprising very large systems of variables and equations that can be computationally demanding requiring considerable computer memory and solution times. Accordingly, these applications are motivating the development and implementation of solution approaches, numerical techniques and algorithms capable of efficiently exploiting modern computational resources in terms utilizing multiprocessor systems and/or acceleration devices.

The particular direction that we focus on in this work is the development of efficient solution strategies for multiperiod dynamic optimization formulations which incorporate process mod-

els described by differential-algebraic equations (DAEs) with uncertain parameters and/or disturbance inputs. To this end, we are investigating a direct multiperiod discretization approach of the uncertainty space combined with a multiple-shooting discretization of the temporal domain which requires the solution of an embedded dynamic model within an overall nonlinear programming algorithm. The goals of the project include: (1) a proof-of-concept implementation of a multiple-shooting dynamic optimization tool applicable to multiperiod formulations; (2) an analysis of solving large-scale dynamic optimization formulations, in which aspects of the solution algorithm are performed in parallel, where the idea is to provide insight into potential computation improvements; (3) a proof-of-concept parallel computing nonlinear programming implementation applicable to multiperiod dynamic optimization formulations.

1.2 Main Contributions

The main novel contributions of this work include:

1. A proposed solution methodology and framework for solving dynamic optimization formulations that explicitly accounts for uncertainty within the model parameters or inputs, is impartial to the type of uncertainty, and is applicable to and effective at solving large-scale model-based design and control formulations.
2. A demonstration of the solution methodology via an analysis of the parallel implementation of the embedded model solution, and application to large-scale chemical process systems.
3. The development and investigation of a multiperiod nonlinear programming technique that uses sequential quadratic programming (SQP) where each QP subproblem utilizes an interior-point method (IPM) which exploits the formulation structure via a Schur-complement decomposition of the primal-dual equations

with a parallel implementation.

1.3 Dynamic Optimization under Uncertainty

Real processes have uncertain and time varying parameters and disturbances, which must be accounted for in order to develop robust design criteria and control algorithms for flexible process operation. Accounting for uncertainty within a mathematical process design or predictive control formulation, amounts to optimizing a particular objective function while satisfying feasibility conditions over a possible range of uncertain operational parameters or input disturbances. Process design formulations that consider the influence of dynamics allow for the simultaneous assessment of the economic cost of the design and the corresponding control system design and performance. On the other hand, the consideration of uncertainty within model-based predictive control strategies allows one to effectively design robust control strategies that maintain the process within a design feasible region while achieving a desired optimal objective criterion. Particular examples of interest to our project include:

1. the determination of optimal time-invariant design parameters subject to operational parameter uncertainty and open- and closed-loop set-point transitions
2. the establishment of robust optimal control profiles for open-loop set-point transitions with uncertain time-invariant model parameters (e.g., thermodynamic parameters, pressure drop) and/or time-varying disturbances (feed flow rate, compositions)

The class of dynamic optimization formulation that we focus on can be defined as a nonlinear dynamic stochastic program which seeks to optimize, as degrees of freedom, a time-varying control profile and/or design/model parameters. This formulation can be stated according

to,

$$\begin{aligned}
\min_{\mathbf{u}(t),\,\mathbf{p}} \quad & \phi_0(\mathbf{p},t_f) + E_{\boldsymbol{\theta}\in\Gamma}\{\phi(\mathbf{x}(t_f),\mathbf{z}(t_f),\mathbf{p},\boldsymbol{\theta},t_f)\} \\
\text{st}: \quad & \dot{\mathbf{x}}(t) - \mathbf{f}_d(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{v}(t),\mathbf{p},\boldsymbol{\theta},t) = \mathbf{0} \\
& \mathbf{f}_a(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{v}(t),\mathbf{p},\boldsymbol{\theta},t) = \mathbf{0} \\
& \mathbf{x}(t_0) - \mathbf{h}_0(\mathbf{u}(t_0),\mathbf{v}(t_0),\mathbf{p},\boldsymbol{\theta},t_0) = \mathbf{0} \\
& \mathbf{g}(\mathbf{x}(t),\mathbf{z}(t),\mathbf{u}(t),\mathbf{v}(t),\mathbf{p},\boldsymbol{\theta},t) \leq \mathbf{0} \\
& \mathbf{u}(t) \in U,\ \mathbf{p} \in P,\ t \in T
\end{aligned} \tag{P.1.1}$$

where an expectation function is utilized to capture the mean objective value over an infinite set of random parameters $\boldsymbol{\theta}$ and is defined as,

$$E_{\boldsymbol{\theta}\in\Gamma}\{\phi(\cdot)\} = \textstyle\int_{\boldsymbol{\theta}\in\Gamma} P(\boldsymbol{\theta}) \cdot \phi(\mathbf{x}(t_f),\mathbf{z}(t_f),\mathbf{p},\boldsymbol{\theta},t_f)\,\mathrm{d}\boldsymbol{\theta} \tag{1.1}$$

Furthermore, we define the sets $X \subseteq \mathbb{R}^{n_x}$ and $Z \subseteq \mathbb{R}^{n_z}$ for differential and algebraic state variables, $U = [\mathbf{u}^L, \mathbf{u}^U] \subset \mathbb{R}^{n_u}$ for the open-loop control variables, $V \subseteq \mathbb{R}^{n_v}$ for input disturbance variables, $P = [\mathbf{p}^L, \mathbf{p}^U] \subset \mathbb{R}^{n_p}$ for optimization design parameters, $\Gamma \subseteq \mathbb{R}^{n_\theta}$ for uncertain parameters, and $T = [t_0, t_f] \subset \mathbb{R}$ for the independent time domain. Accordingly, $\phi_0(\cdot) : P \times T \mapsto \mathbb{R}$ and $\phi(\cdot) : X \times Z \times P \times \Gamma \times T \mapsto \mathbb{R}$ represent scalar deterministic and stochastic objective function criteria, respectively; $\mathbf{f}_d(\cdot) : X \times Z \times U \times V \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_x}$ and $\mathbf{f}_a(\cdot) : X \times Z \times U \times V \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_z}$ represent the dynamic process model (in semi-explicit DAE form); $\mathbf{h}_0(\cdot) : U \times V \times P \times \Gamma \times \mathbb{R} \mapsto \mathbb{R}^{n_x}$ represents a parameter dependent function governing the initial differential state conditions; $\mathbf{g}(\cdot) : X \times Z \times U \times V \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_g}$ are path inequality constraints (possibly including a subset of point or end-point constraints on closed-loop control performance and/or open-loop inputs); $\mathbf{u}(t) \in U$ are time-variant control input variables; $\mathbf{v}(t) \in V$ are time-variant disturbance input variables (typically defined a priori); and $\mathbf{p} \in P$ are time-invariant model/design parameters; $P(\boldsymbol{\theta}) : \Gamma \mapsto \mathbb{R}$ is a continuous probability density function that represents the uncertainty distribution and $\boldsymbol{\theta} \in \Gamma = \{\boldsymbol{\theta} \in \mathbb{R}^{n_\theta} :\ \boldsymbol{\theta}^L \leq \boldsymbol{\theta} \leq \boldsymbol{\theta}^U\}$ are time-invariant parameters that lie within this

uncertainty region around some nominal/mean value.

1.4 Dynamic Optimization Solution Approach

The approach taken in this thesis to solve large-scale stochastic dynamic optimization problems given by Problem P.1.1 is to discretize both the infinite dimensional uncertainty space and the temporal domain. The resulting formulation produces a nonlinear program which can be solved using a number of possible algorithms. The particular approach adopted to transform the continuous stochastic program of Problem P.1.1 follows a sampling method, which we denote as a multiperiod or multiscenario discretization. In this method, numerous scenario realizations of select model parameters or parameterized disturbance trajectories are generated (via sampling from a particular distribution or uniformly within an interval of the random "uncertain" variables) and the underlying model equations and variables are repeated for each realization and then posed as single large-scale continuous dynamic optimization formulation. Practical solution methods to solve the resulting dynamic optimization formulation typically follow direct discretization approaches whereby the time-varying infinite dimensional control input variable trajectories, and possibly the state variable trajectories, are parameterized over the optimization time horizon and a particular objective function is minimized using a nonlinear programming solver. Popular direct methods include direct sequential (control vector parameterization), direct simultaneous (full transcription), and multiple-shooting, which is a hybrid between sequential and simultaneous methods.

1.5 Intended Applications

The type of applications we are targeting with our dynamic optimization solution approach lean mainly towards off-line problems for optimal process design and operation under transient conditions. These types of optimization formulations are posed in manner that seek

to determine the best possible economic design with good control performance such that the system is able to respond in an agile manner to possibly unforeseen circumstances imposed on the system. In addition, operation problems may include determining a robust set-point or reference trajectory subject to uncertain disturbance inputs to the system, where we use the term robust in referring to the ability of the optimal control trajectory to satisfy the imposed system constraints given the uncertainty in the input disturbance and/or internal model parameters.

1.6 Thesis Outline

This thesis is organized according to the following chapters:

In Chapter 2, we present our first contribution which considers applying a proof-of-concept multiperiod multiple-shooting dynamic optimization algorithm in `Matlab` for the design of dynamic systems. In particular, we focus primarily on potential solution speedup by considering the parallel solution of the embedded ODE models within the nonlinear programming algorithm of the discretized dynamic optimization formulation.

In Chapter 3, we present our second contribution which follows from our first, and considers large-scale multiperiod dynamic optimization formulations with embedded differential-algebraic equation (DAE) models. The key aspects beyond those noted in Chapter 2 include: the specific treatment of semi-explicit index-1 DAEs, the investigation and use of higher-order state sensitivities within the NLP algorithm, a more complete parallel algorithm performance analysis using a proper `C/C++` implementation with shared-memory `OpenMP` constructs.

In Chapter 4, we present our third contribution which focuses on the development of a nonlinear programming algorithm that uses distributed parallel computing concepts via the message-passing interface (MPI) to facilitate the solution of structured large-scale dynamic optimization formulations. Note, that our investigation in this chapter goes beyond those

in previous chapters, which only looked at the parallelization of the embedded dynamic model solution. Furthermore, with this new contribution we seek to alleviate the bottleneck associated with using a serial NLP solver encountered in the previous chapters

In Chapter 5, we summarize the thesis and reiterate what we consider to be the main contributions to this area of study. Furthermore, future directions are discussed and our proposed next steps are laid out.

Chapter 2

A Parallel Implementation for Multiperiod Dynamic Optimization of ODE Systems

This chapter develops a technique for optimizing dynamic systems under uncertainty using a prototype parallel programming implementation. A multiple-shooting discretization scheme is applied, whereby each shooting interval is solved using an error-controlled differential equation solver. In addition, the uncertain parameter space is discretized, resulting in a multiperiod optimization formulation. Each shooting interval and period (scenario) realization is completely independent, thus a major focus of this chapter is on demonstrating potential computational performance improvement when the embedded dynamic model solution of the multiperiod algorithm is implemented in parallel. We assess our parallel multiperiod and multiple-shooting based dynamic optimization algorithm on two case studies involving integrated plant and control system design, where the objective is to simultaneously determine the size of the process equipment and the control system tuning parameters that minimize cost, subject to uncertainty in the disturbance inputs.

Note, portions of this chapter were published according to the journal article:

I.D. Washington and C.L.E. Swartz. “Design under Uncertainty using Parallel Multiperiod Dynamic Optimization”. In: *AIChE Journal*, 2014. 60 (9), 3151–3168.

2.1 Introduction

Dynamic optimization is an important tool for solving practical problems of model predictive control (MPC), moving horizon state estimation (MHE), optimal control, parameter estimation, and design with dynamic performance considerations. Practical dynamic optimization methodologies involve the use of direct techniques whereby partial or full discretization is applied to a dynamic system to yield an algebraic nonlinear program. Within this domain, there are three well established practical classes of algorithms for continuous dynamic optimization applicable to large-scale systems, namely sequential (i.e., single-shooting), simultaneous (i.e., full transcription), and multiple-shooting methods. Sequential methods solve an embedded dynamic model at each optimization iteration; simultaneous methods fully discretize the process model and then solve the optimization and model simultaneously; multiple-shooting is a hybrid of sequential and simultaneous methods that independently solves the embedded model over a number of independent successive intervals, while an outer nonlinear programming (NLP) solver is used to satisfy the particulars of the desired optimization formulation and to ensure that continuity is achieved in the state trajectories between each successive shooting interval. All techniques have been applied successfully to a number of large-scale chemical process systems [1–3], with more detail on the algorithmic aspects given in [4].

In terms of advancing the state of direct dynamic optimization solution techniques such that larger plant-wide models can be used, recent work has focused on parallel implementations for: (1) the parameter sensitivity computation within the sequential approach [5]; (2) the embedded model and sensitivity solution over each shooting interval within the multiple-shooting algorithm [6]; (3) the structured linear algebra of the Karush-Kuhn-Tucker

(KKT) system solution for the interior-point NLP algorithm, used within the simultaneous method [7]. Furthermore, with scenario-based solution strategies for handling uncertainty, many past implementations have been serial with no exploitation of parallelism for the multiple periods/scenarios used. One exception where parallelization is used is a recent study by Zhu, Legg, and Laird [8], who demonstrated a multiperiod nonlinear programming algorithm for the optimal operation of an air separation process under uncertainty. Their approach considers using a multiperiod algorithm to include parametric uncertainty within the disturbances (discretized into several possible realizations) as opposed to simply using nominal model inputs, and then applies large-scale nonlinear programming solution techniques where the underlying linear algebra structure is exploited using a parallelized solution strategy. In another direction, Ricardez-Sandoval [9] has used parallel techniques for Monte Carlo simulation within an integrated design and control framework, whereby independent function evaluations for dynamic model simulation are performed in parallel.

The particular application that we focus on in this chapter is the integration of design and control problem, which offers a systematic approach to address poor control performance, violation of safety and environmental constraints, and degradation in economic performance through dynamic optimization design formulations. The design of chemical process plants is traditionally performed by first sizing the equipment via heuristic based calculations under steady-state conditions, followed by the development of a control system structure and corresponding tuning under transient conditions. It has been well recognized that using such an approach can lead to a process design with unfavorable process dynamics, which can be difficult to adequately control in the presence of unforeseen disturbances or process variability [10]. A more recent strategy has been to design the plant considering dynamic performance by integrating both design and control at the same stage, such that operational variability can be directly addressed at the design stage. The central idea to combining both design and control tasks into a single phase is to permit potential control performance limiting factors to be addressed simultaneously during system design so that designs that

are likely to cause control difficulties can be avoided. Detailed discussions of integration techniques for design and control can be found in the comprehensive reviews by Schijndel and Pistikopoulos [11], Sakizlis, Perkins, and Pistikopoulos [12], and more recently by Yuan et al. [13].

In order to construct a realistic design formulation, uncertainty must be explicitly embedded within the dynamic system that describes possible unknown exogenous disturbance inputs and/or model parameters. Our focus is on a stochastic optimization formulation, for which a straightforward route involves using a scenario-based or multiperiod optimization technique which describes uncertainty through a set of possible parameter scenarios based on a discrete probability distribution or discretized probability functions.

The focus of this chapter is on the underlying solution techniques utilized in systematic optimization approaches to the integration of design and control. More specifically, this chapter is on the development of a parallel computing implementation that utilizes a multiperiod approach to handle disturbance uncertainty, and a multiple-shooting dynamic optimization solution technique. The discretization of the infinite dimensional uncertainty space (multiperiod or multiscenario approach), when applied to large-scale ordinary differential equations (ODEs), often yields large and potentially unwieldy systems of equations, which must be solved in an efficient manner for computational tractability. Fortunately, the resulting uncertainty realizations at the ODE level are independent within the state space, which allows the solution to be implemented in a highly parallel manner. Parallelization can be further exploited by adopting a parallel multiple-shooting algorithm for solving the dynamic optimization problem [6]. As with the sequential (single-shooting) approach, multiple-shooting readily employs error-based step-size control through the integration routine, but it is less prone to failure when optimization iterates yield values of the optimization variables that result in unstable dynamics. Our current algorithm decomposes the differential equations within the optimization formulation by partitioning both the number of periods and shooting intervals and off-loading each of these independent integration tasks to a parallel computing

cluster, thus significantly speeding up each iteration of the nonlinear programming algorithm. The approach is well suited for use as a tool in existing optimization-based design and control frameworks [14], or as a core solver in robust operational problems of MPC or open-loop robust optimal control [15].

The chapter is laid out by first discussing the multiperiod formulation, followed by our dynamic optimization solution approach and implementation. Next, two case studies of different scale are presented to illustrate the computational performance of the algorithm. Finally, some concluding remarks are provided which highlight the benefits of the proposed algorithm.

2.2 Dynamic Multiperiod Optimization

Multiperiod optimization is a commonly used technique to approximate stochastic programs whereby an infinite dimensional (continuous) stochastic program is reformulated as a discrete-time problem such that the probability distribution is discretized at several points in the random variable space [16]. This type of formulation has been widely applied in the context of plant-wide design and scheduling of chemical processes under uncertainty [17, 18]. Steady-state process models are typically used whereby the objective is to optimize an economic criterion while ensuring that the plant can operate under several possible conditions arising in a sequence of possibly different time periods.

The extension of multiperiod formulations to include system dynamics, and hence optimal control formulations, has been addressed by a number of researchers. Ruppen, Benthack, and Bonvin [19] discuss batch reactor trajectory optimization under parametric uncertainty and utilize a simultaneous-based optimization scheme with a successive linear programming (SLP) solution strategy to solve the resulting NLP. Bhatia and Biegler [20] discuss the design and scheduling of batch process plants and utilize sequential quadratic programming (SQP)

for the resulting NLP. The authors' focus was primarily on the optimization formulation to facilitate the integration of design and scheduling. From a design under uncertainty perspective, Mohideen, Perkins, and Pistikopoulos [14] proposed a framework for the integration of design and control which uses an iterative decomposition algorithm involving two optimization stages comprising a mixed-integer multiperiod dynamic optimization design problem and a feasibility analysis problem which is also a mixed-integer dynamic optimization problem.

Many of the past studies that utilize a multiperiod formulation with dynamic systems have generally considered a relatively small number of scenarios to approximate the expected value of the objective function and constraint satisfaction. To accurately capture the effects of uncertainty on systems with many uncertain parameters requires a large number of scenarios. The introduction of numerous scenarios, particularly in large-scale dynamic systems, raises the issue of computational tractability, which consequently requires an efficient solution implementation. In this chapter, we explore the use of parallel computing strategies.

2.2.1 Multiperiod Formulation

For a dynamic system consisting of ordinary differential equations (ODEs) over a fixed time horizon, a general stochastic dynamic program can be stated as,

$$\begin{aligned}
\min_{\mathbf{u}(t),\mathbf{p}} \quad & \mathcal{J} := \phi_0(\mathbf{p}, t_f) + E_{\boldsymbol{\theta}\in\Gamma}\{\phi(\mathbf{x}(t_f), \mathbf{p}, \boldsymbol{\theta}, t_f)\} \\
\text{st}: \quad & \dot{\mathbf{x}}(t) - \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), \mathbf{p}, \boldsymbol{\theta}, t) = \mathbf{0} \\
& \mathbf{x}(t_0) - \mathbf{h}_0(\mathbf{u}(t_0), \mathbf{v}(t_0), \mathbf{p}, \boldsymbol{\theta}, t_0) = \mathbf{0} \\
& \mathbf{g}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), \mathbf{p}, \boldsymbol{\theta}, t) \leq \mathbf{0} \\
& \mathbf{u}(t) \in U,\ \mathbf{p} \in P,\ t \in T
\end{aligned} \tag{P.2.1}$$

where the expectation function is defined as,

$$E_{\boldsymbol{\theta}\in\Gamma}\{\phi(\cdot)\} = \int_{\boldsymbol{\theta}\in\Gamma} P(\boldsymbol{\theta}) \cdot \phi(\mathbf{x}(t_f), \mathbf{p}, \boldsymbol{\theta}, t_f)\, \mathrm{d}\boldsymbol{\theta} \tag{2.1}$$

Furthermore, we define the sets $X \subseteq \mathbb{R}^{n_x}$ for state variables, $U = [\mathbf{u}^L, \mathbf{u}^U] \subset \mathbb{R}^{n_u}$ for the open-loop control variables, $V \subseteq \mathbb{R}^{n_v}$ for input disturbance variables, $P = [\mathbf{p}^L, \mathbf{p}^U] \subset \mathbb{R}^{n_p}$ for optimization design parameters, $\Gamma \subseteq \mathbb{R}^{n_\theta}$ for uncertain parameters, and $T = [t_0, t_f] \subset \mathbb{R}$ for the independent time domain. Accordingly, $\phi(\cdot) : X \times P \times \Gamma \times T \mapsto \mathbb{R}$ represents a scalar objective function criterion, evaluated at the final time, that could possibly represent economic, state/output tracking, or a combination thereof; $\mathbf{f}(\cdot) : X \times U \times V \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_x}$ represents the ordinary differential equation right-hand-side function of the process model; $\mathbf{g}(\cdot) : X \times U \times V \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_g}$ are path inequality constraints (possibly including a subset of interior-point or end-point inequality constraints on closed-loop control performance, open-loop inputs, and/or design parameters); $\mathbf{u}(t) \in U$ are time-variant control input variables; $\mathbf{v}(t) \in V$ are time-variant disturbance input variables (typically defined a priori); and $\mathbf{p} \in P$ are time-invariant model/design parameters; $P(\boldsymbol{\theta}) : \Gamma \mapsto \mathbb{R}$ is a continuous probability density function that represents the uncertainty distribution and $\boldsymbol{\theta} \in \Gamma = \{\boldsymbol{\theta} \in \mathbb{R}^{n_\theta} : \ \boldsymbol{\theta}^L \leq \boldsymbol{\theta} \leq \boldsymbol{\theta}^U\}$ are time-invariant parameters that lie within this uncertainty region around some nominal/mean value.

The infinite dimensional formulation given by Problem P.2.1 can be discretized using a number of scenarios with an associated probability of occurrence (uncertainty space sampling). Accordingly, the objective function can be approximated as a weighted sum of different possible scenarios sampled from a particular distribution. As such, we can formulate Problem P.2.2 by considering $\boldsymbol{\theta} \in \Gamma \equiv \{\boldsymbol{\theta}_i\}_{i=1}^{n_s}$ in a multiperiod (or multiscenario) form defined by a

specified number of uncertain parameter scenarios n_s (realizations),

$$\begin{aligned}
\min_{\mathbf{u}_i(t),\mathbf{d}_i \forall\, i,\mathbf{p}} \quad & \mathcal{J} := \phi_0(\mathbf{p}, t_f) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot \phi_i(\mathbf{x}_i(t_f), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_f) \\
\text{st}: \quad & \dot{\mathbf{x}}_i(t) - \mathbf{f}(\mathbf{x}_i(t), \mathbf{u}_i(t), \mathbf{v}_i(t), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t) = \mathbf{0} \\
& \mathbf{x}_i(t_0) - \mathbf{h}_0(\mathbf{u}_i(t_0), \mathbf{v}_i(t_0), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_0) = \mathbf{0} \\
& \mathbf{g}(\mathbf{x}_i(t), \mathbf{u}_i(t), \mathbf{v}_i(t), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t) \leq \mathbf{0} \\
& \mathbf{u}_i(t) \in U,\ \mathbf{d}_i \in D \ \forall\ i = 1, \ldots, n_s \\
& \mathbf{p} \in P,\ t \in T
\end{aligned} \tag{P.2.2}$$

where the states $\mathbf{x}_i(t)$, open-loop controls $\mathbf{u}_i(t)$, disturbances $\mathbf{v}_i(t)$, local scenario-specific parameters $\mathbf{d}_i$ and uncertain parameter realizations $\boldsymbol{\theta}_i$ are associated with a particular scenario i, and the design or global model parameters $\mathbf{p}$ are uniform over all scenarios. The weight (or probability) associated with each scenario i is represented as $\mathsf{w}_i \in [0, 1]$. This particular formulation, where we associate the control variables with each scenario, is of the form of a two-stage stochastic program. The parameters $\mathbf{p}$ constitute first-stage decisions, and the control inputs $\mathbf{u}_i(t)$ constitute second-stage decisions that can provide compensatory action in response to disturbance and parameter realizations. Due to the discretization approach, the NLP formulation (constraint gradient/incident matrices) has a distinct block structure whereby each scenario realization represents a block [17]. General NLP techniques to exploit block structured formulations and associated software tools are currently the subject of much research [7, 21]. The approach proposed in this chapter does not focus on structure exploitation at the NLP level, but instead takes advantage of the decomposed model structure that is embedded within the multiple-shooting approach. This permits the use of existing NLP solvers, with expensive model function evaluations (i.e., DAE solution) handled in an efficient manner via parallelization.

A popular technique for generating scenarios is Monte Carlo sampling, which entails sampling from probability distributions describing the uncertain parameters. A realization for each pa-

rameter, taken together, constitutes a scenario, and each scenario is assigned an equal weight with the sum of weights equal to one [22]. Several alternative sampling techniques have been proposed, with the goal of maintaining good approximation accuracy with increased computational efficiency; some discussion of such approaches is given in Diwekar [23].

2.2.2 Design and Control Formulation

In the context of design and control, the optimization parameters $\mathbf{p}$ defined in problem P.2.2 refer to the equipment sizes (e.g., reactor, distillation column, or heat exchanger dimensions), controller tuning parameters and process set-points. Furthermore, the objective function would typically be an economic index of the combined annualized capital and operating costs of the process plant. As a result, the particular form of the objective function differs slightly from problem P.2.2, in that one is able to partition the overall cost into a design or capital cost portion $C_{\text{cap}}(\mathbf{p})$, as a function of the design parameters $\mathbf{p}$, and a control or operation cost portion $C_{\text{op}}(\mathbf{x}(t), \mathbf{u}(t), \mathbf{v}(t), \mathbf{p}, \boldsymbol{\theta}, t)$, as a function of the system response $\mathbf{x}(t)$, manipulated inputs $\mathbf{u}(t)$, specified disturbances $\mathbf{v}(t)$, design parameters $\mathbf{p}$ and uncertain parameters $\boldsymbol{\theta}$. Accordingly, the objective function can be stated as follows,

$$\mathcal{J} := C_{\text{cap}}(\mathbf{p}) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot C_{\text{op}}(\mathbf{x}_i(t_f), \mathbf{u}_i(t_f), \mathbf{v}_i(t_f), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_f) \tag{2.2}$$

where the manipulated control inputs $\mathbf{u}(t)$ are determined via an embedded control law and the time-varying disturbance inputs can be specified directly via $\mathbf{v}(t)$, or parameterized through the time-invariant parameters $\boldsymbol{\theta}$, as stepwise profiles according to,

$$\mathbf{v}(t; \boldsymbol{\theta}) := \mathbf{v}_0 + \Delta\mathbf{v}\, \gamma(t, t_{\text{step}}) \tag{2.3}$$

where the uncertain parameters include the initial disturbance values $\mathbf{v}_0$ and the step magnitude values $\Delta\mathbf{v}$, and can be stated accordingly as $\boldsymbol{\theta} := [\mathbf{v}_0^\top, \Delta\mathbf{v}^\top]^\top$. The function $\mathbf{v}(t; \boldsymbol{\theta})$

is defined for single-step disturbance inputs and approximates the disturbance trajectories as continuous profiles in order to avoid discontinuities within the model equations and thus avoid potential integration difficulties or the necessity to stop and restart the integration routine [24]. The approximation is performed using an appropriate smoothing function $\gamma(t, t_{\text{step}})$, which is triggered at t_{step}. One possible smoothing function approximation is via the exponential function defined as,

$$\gamma(t, t_{\text{step}}) := [1 + \exp(-\alpha(t - t_{\text{step}}))]^{-1} \in (0, 1) \tag{2.4}$$

where $\alpha > 0$ is a tuning parameter used to adjust the rate of transition of the step and $t_{\text{step}} > t_0$ is the point in time to initiate the transition. Similar disturbance functions can be derived for multi-step profiles or even sinusoidal profiles with an uncertain amplitude and/or frequency [12].

The system inputs that are defined through the control laws are not independent decision variables. However, they remain scenario-dependent, along with the other system states. Controller parameters such as the controller gain, reset time, set-point and bias in a proportional-integral (PI) control law can be treated as being uniform over all scenarios or scenario dependent (by defining a separate set of parameters for each scenario). The most appropriate set-up would depend on the particular context of the application.

2.3 Proposed Solution Framework

As previously noted, direct methods used to solve dynamic optimization problems fall into three major categories: sequential, simultaneous, and multiple-shooting approaches. In this chapter, we focus on the multiple-shooting approach. It typically results in moderately sized NLP problems, allows for automatic control of integration accuracy through the use of DAE solvers, is less susceptible to failure due to unstable process dynamics than sequential

approaches, and is well suited to parallelization.

In this section we define the multiple-shooting approach, explore the extension to solve multiperiod formulations, and discuss our particular parallel implementation.

2.3.1 Multiple-Shooting Discretization

The multiple-shooting technique was made widely accessible through the MUSCOD software originally developed by Bock and Plitt [25] for optimal control using ordinary differential equation (ODE) models, and subsequently adapted for differential-algebraic equation (DAE) models [26, 27]. The technique combines aspects of both the sequential (or single-shooting) and simultaneous methods in that one is able to expose the embedded model states to the optimization layer. The time horizon is partitioned into a number of shooting intervals. Within each interval, the inputs can be further parameterized and the dynamic system integrated, much like the sequential method. The multiple-shooting technique entails introducing new optimization parameters to represent the state variable initial conditions at the beginning of each interval and new equality constraints to remove the discrepancy or defect between the state values at the final time from the previous interval and the initial time in the current interval. A general dynamic optimization formulation that utilizes the multiple-shooting discretization applicable to ODE models can be written according to Problem P.2.3. Note that to make the presentation of the multiple-shooting formulation less cumbersome, we exclude the functional presence of disturbance variables $\mathbf{v}(t)$, the local parameters $\mathbf{d}_i$, and uncertain parameters $\boldsymbol{\theta}$, as well as the index representing the scenario realizations. Furthermore, the continuous control input vector can be defined using a parameterized function $\mathbf{u}(t) := \boldsymbol{\mathcal{U}}(t, \mathbf{u}_j)$ based on a piecewise approximation within each shooting interval I_j for $j = 0, \ldots, n-1$, where $\mathbf{u}_j \in \mathbb{R}^{(M+1)n_u}$ represent local polynomial coefficients for a polynomial of order M. For example, it is common practice to use Lagrange

interpolation polynomials (see [4], for a detailed discussion).

$$
\begin{aligned}
\min_{\boldsymbol{w},\,\mathbf{p}} \;& \mathcal{J} := \phi(\mathbf{x}_n, \mathbf{p}, t_n) \\
\text{st}: \;& \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathfrak{U}(t, \mathbf{u}_j), \mathbf{p}, t) \\
& \mathbf{0} = \mathbf{h}_0(\mathbf{u}_0, \mathbf{p}, t_0) - \mathbf{x}_0 \\
& \mathbf{x}(t_{j+1}; \mathbf{x}_j, \mathbf{u}_j, \mathbf{p}) - \mathbf{x}_{j+1} = \mathbf{0} \\
& \mathbf{g}(\mathbf{x}_k, \mathfrak{U}(t_k, \mathbf{u}_k), \mathbf{p}, t_k) \leq \mathbf{0} \\
& \forall\, t \in I_j \;,\; j = 0, \ldots, n-1 \\
& \forall\, k = 0, \ldots, n \\
& \boldsymbol{w} \in [\boldsymbol{w}^L, \boldsymbol{w}^U],\; \mathbf{p} \in [\mathbf{p}^L, \mathbf{p}^U]
\end{aligned}
\tag{P.2.3}
$$

The optimization parameters are partitioned into model or design parameters $\mathbf{p}$, and the shooting node parameters for state and input variable parameters, defined collectively as $\boldsymbol{w} := [\mathbf{x}_0^\top, \mathbf{u}_0^\top, \ldots, \mathbf{x}_n^\top]^\top \in \mathbb{R}^{n_x(n+1)+(M+1)n_u n}$. Note that $\mathbf{u}_n$ is used in formulation P.2.3 for notational simplicity, where $\mathbf{u}_n := \mathbf{u}_{n-1}$, and can be removed from the NLP. Additionally, we consider here a fixed end-time formulation where the objective function $\phi(\cdot)$ is represented in Mayer form, which typically only directly depends on the final model states $\mathbf{x}_n$, parameters $\mathbf{p}$, and possibly the final time t_n. The ODE model, $\mathbf{F}(\cdot) := \dot{\mathbf{x}}(t) - \mathbf{f}(\cdot)$ is embedded within the NLP function evaluations and is solved using an appropriate integration solver for $t \in I_j$, $j = 0, \ldots, n-1$ with initial state conditions $\mathbf{x}(t_j) := \mathbf{x}_j$ for all intervals I_j, where the intervals are decoupled using the new parameters and are thus independent from each other. In order to remove the defect between the initial state parameters $\mathbf{x}_j$ and the previous interval's final integrated state values $\mathbf{x}(t_{j-1})$, continuity equality constraints are imposed at each shooting node j. The approach to handling inequality path constraints, $\mathbf{g}(\cdot)$, is to approximate them as interior point constraints at each shooting node, which avoids any otherwise necessary reformulation which is typically used in single-shooting approaches. In some cases this approximation may not suffice to remove inter-node constraint violation and in such cases the modeler can apply an end-point constraint to the integral of the constraint violation, such

as proposed for single-shooting methods [28]. In the final optimization formulation posed to the NLP solver, the state variables $\mathbf{x}(t_{j+1})$ appear only within the continuity constraint evaluations, all other constraint and objective function evaluations are with respect to direct optimization parameters. As a result of the direct state variable presence, sensitivity analysis techniques are necessary when deriving the continuity constraint derivatives.

2.3.2 Objective & Constraint Function Derivative Evaluation

The gradient of the objective function (in Mayer form) can be evaluated directly at the final shooting node with respect to $\boldsymbol{w}_n := \mathbf{x}_n$ and $\mathbf{p}$, and can be stated as,

$$\nabla_{\{\boldsymbol{w},\mathbf{p}\}}\mathcal{J} := \left[\mathbf{0}_{n_r}^{\top}, \mathbf{J}_n^{x_n\top}, \mathbf{J}_n^{p\top}\right]^{\top} \tag{2.5}$$

where the first $n_r := (n_x + (M+1)n_u)n$ vector components are zero, and $\mathbf{J}_n^{x_n}$ and $\mathbf{J}_n^{p}$ represent first derivative vectors making up the NLP objective gradient evaluated at the end-point $(\mathbf{x}_n, \mathbf{p}, t_n)$. These vectors can be either specified analytically or more suitably generated via automatic differentiation at the current optimization parameter iterates.

The multiple-shooting continuity equality constraints, including the initial conditions at t_0, can be defined as,

$$\begin{aligned} \mathbf{c}_0\left(\boldsymbol{w}_0, \mathbf{p}\right) &\equiv \mathbf{h}_0(\mathbf{u}_0, \mathbf{p}, t_0) - \mathbf{x}_0 = \mathbf{0} \\ \mathbf{c}_{j+1}\left(\boldsymbol{w}_j, \mathbf{x}_{j+1}, \mathbf{p}\right) &\equiv \mathbf{x}(t_{j+1}; \mathbf{x}_j, \mathbf{u}_j, \mathbf{p}) - \mathbf{x}_{j+1} = \mathbf{0} \end{aligned} \tag{2.6}$$

where $\boldsymbol{w}_j := [\mathbf{x}_j^{\top}, \mathbf{u}_j^{\top}]^{\top}$ for $j = 0, \ldots, n-1$. The remaining inequality constraints represent NLP point constraints and can be defined at each shooting node according to,

$$\begin{aligned} \mathbf{q}_j\left(\boldsymbol{w}_j, \mathbf{p}\right) &\equiv \mathbf{g}(\boldsymbol{w}_j, \mathbf{p}) \\ \mathbf{q}_n\left(\mathbf{u}_{n-1}, \boldsymbol{w}_n, \mathbf{p}\right) &\equiv \mathbf{g}(\mathbf{u}_{n-1}, \boldsymbol{w}_n, \mathbf{p}) \end{aligned} \tag{2.7}$$

The combined constraint vector for all NLP constraints can now be stated as,

$$\mathbf{c}(\boldsymbol{w},\mathbf{p}) := \begin{bmatrix} \mathbf{c}_0(\boldsymbol{w}_0,\mathbf{p}) \\ \left(\mathbf{q}_0(\boldsymbol{w}_0,\mathbf{p})^\top, \mathbf{c}_1(\boldsymbol{w}_0,\mathbf{x}_1,\mathbf{p})^\top\right)^\top \\ \vdots \\ \left(\mathbf{q}_j(\boldsymbol{w}_j,\mathbf{p})^\top, \mathbf{c}_{j+1}(\boldsymbol{w}_j,\mathbf{x}_{j+1},\mathbf{p})^\top\right)^\top \\ \vdots \\ \left(\mathbf{q}_{n-1}(\boldsymbol{w}_{n-1},\mathbf{p})^\top, \mathbf{c}_n(\boldsymbol{w}_{n-1},\mathbf{x}_n,\mathbf{p})^\top\right)^\top \\ \mathbf{q}_n(\mathbf{u}_{n-1},\boldsymbol{w}_n,\mathbf{p}) \end{bmatrix} \tag{2.8}$$

where $\mathbf{c}(\boldsymbol{w},\mathbf{p}) \in \mathbb{R}^{(n_x+n_q)(n+1)}$. As a result of the stacked structure of both the constraint functions and optimization parameters $\boldsymbol{w} := [\boldsymbol{w}_0^\top,\ldots,\boldsymbol{w}_n^\top]^\top$, we maintain a sparse diagonal matrix structure within the constraint Jacobian matrix with respect to $\boldsymbol{w}$, and a block vector structure for the derivatives with respect to $\mathbf{p}$. This matrix is denoted as $\nabla_{\{\boldsymbol{w},\mathbf{p}\}}\mathbf{c}(\boldsymbol{w},\mathbf{p})$ and can be defined as,

$$\nabla_{\{\boldsymbol{w},\mathbf{p}\}}\mathbf{c}(\boldsymbol{w},\mathbf{p}) := \begin{bmatrix} -\mathbf{I}_{n_x} & \mathbf{H}_0^{u_0} & & & & & \mathbf{H}_0^{p} \\ \mathbf{Q}_0^{x_0} & \mathbf{Q}_0^{u_0} & & & & & \mathbf{Q}_0^{p} \\ \mathbf{X}_1^{x_0} & \mathbf{X}_1^{u_0} & -\mathbf{I}_{n_x} & & & & \mathbf{X}_1^{p} \\ & & \ddots & & & & \vdots \\ & & \mathbf{Q}_j^{x_j} & \mathbf{Q}_j^{u_j} & & & \mathbf{Q}_j^{p} \\ & & \mathbf{X}_{j+1}^{x_j} & \mathbf{X}_{j+1}^{u_j} & -\mathbf{I}_{n_x} & & \mathbf{X}_{j+1}^{p} \\ & & & & \ddots & & \vdots \\ & & & & \mathbf{Q}_{n-1}^{x_{n-1}} & \mathbf{Q}_{n-1}^{u_{n-1}} & \mathbf{Q}_{n-1}^{p} \\ & & & & \mathbf{X}_n^{x_{n-1}} & \mathbf{X}_n^{u_{n-1}} & -\mathbf{I}_{n_x} & \mathbf{X}_n^{p} \\ & & & & & \mathbf{Q}_n^{u_{n-1}} & \mathbf{Q}_n^{x_n} & \mathbf{Q}_n^{p} \end{bmatrix} \tag{2.9}$$

where $\mathbf{H}_0^{u_0}$ and $\mathbf{H}_0^{p}$ are the first derivatives of $\mathbf{h}_0(\cdot)$ with respect to $\mathbf{u}_0$ and $\mathbf{p}$, respectively;

while $\mathbf{Q}_j^{\mathcal{M}}$ represent point constraint first derivatives, at each shooting node, defined as,

$$\mathbf{Q}_j^{x_j} := \frac{\partial \mathbf{q}_j(\boldsymbol{w}_j, \mathbf{p})}{\partial \mathbf{x}_j};\ \mathbf{Q}_j^{u_j} := \frac{\partial \mathbf{q}_j(\boldsymbol{w}_j, \mathbf{p})}{\partial \mathbf{u}_j};\ \mathbf{Q}_j^{p} := \frac{\partial \mathbf{q}_j(\boldsymbol{w}_j, \mathbf{p})}{\partial \mathbf{p}} \tag{2.10}$$

Again, the derivatives $\mathbf{H}_0^{u_0}$, $\mathbf{H}_0^{p}$ and $\mathbf{Q}_j^{\mathcal{M}}$ can be either specified analytically or generated via automatic differentiation. The matrices given by $\mathbf{X}_{j+1}^{\mathcal{M}}$ represent the embedded state parameter sensitivities, for each parameter $\mathcal{M} := \{x_j, u_j, p\}$, evaluated at the end of each shooting node, and are defined according to,

$$\mathbf{X}_{j+1}^{x_j} := \frac{\partial \mathbf{x}(t_{j+1})}{\partial \mathbf{x}_j};\ \mathbf{X}_{j+1}^{u_j} := \frac{\partial \mathbf{x}(t_{j+1})}{\partial \mathbf{u}_j};\ \mathbf{X}_{j+1}^{p} := \frac{\partial \mathbf{x}(t_{j+1})}{\partial \mathbf{p}} \tag{2.11}$$

Many large-scale differential equation solvers provide state sensitivities via forward sensitivity analysis [29] or possibly adjoint (reverse) sensitivity analysis [30], and the particular numerical methods used to compute this information are highly important in terms of both solution accuracy and speed when using large-scale chemical process engineering models [5]. Due to the direct presence of the state variables $\mathbf{x}(t_{j+1})$ in the continuity constraints, a natural approach to generate the optimization function derivatives is forward sensitivity analysis. Accordingly, this involves solving the forward sensitivity equation system alongside the original ODE system, which can be defined in matrix form as,

$$\dot{\mathbf{x}}_y(t) = \mathbf{f}_x(t)\,\mathbf{x}_y(t) + \mathbf{f}_y(t) \quad t \in [t_j, t_{j+1}],\ j = 0, \ldots, n-1 \tag{2.12}$$

$$\mathbf{x}_y(t_j) = [\mathbf{I}_{n_x} | \mathbf{0}_{n_x \times (n_y - n_x)}] \tag{2.13}$$

where $\mathbf{x}_y(t) := \partial \mathbf{x}(t)/\partial \mathbf{y}_j$ represents the desired sensitivity variable solution, $\mathbf{f}_{\{x,y\}}(t) := \partial \mathbf{f}(\mathbf{x}(t), \mathbf{y}_j, t)/\partial\{\mathbf{x}(t), \mathbf{y}_j\}$ are Jacobian matrices of the ODE model, and $\mathbf{y}_j := \{\mathbf{x}_j, \mathbf{u}_j, \mathbf{p}\}$ represents a subset of NLP parameters at each shooting node j. The solution of this system is determined at t_j for $j = 1, \ldots, n$ and subsequently used in Equation 2.11. We will forgo any further details of sensitivity analysis and direct the reader to Biegler [4] for an in depth discussion.

2.3.3 Combined Multiperiod Multiple-Shooting NLP Formulation

The fully discretized combined multiperiod multiple-shooting NLP formulation of problems P.2.2 and P.2.3, can now be stated according to,

$$\begin{aligned} \min_{\bar{\boldsymbol{w}}_i \, \forall \, i, \mathbf{p}} \quad & \mathcal{J} := \phi_0(\mathbf{p}) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot \phi_i(\boldsymbol{w}_{i,n}, \mathbf{d}_i, \mathbf{p}) \\ \text{st}: \quad & \mathbf{c}_i^L \leq \mathbf{c}_i(\bar{\boldsymbol{w}}_i, \mathbf{p}) \leq \mathbf{c}_i^U \\ & \bar{\boldsymbol{w}}_i \in [\boldsymbol{w}^L, \boldsymbol{w}^U] \;\; \forall \; i = 1, \ldots, n_s \\ & \mathbf{p} \in [\mathbf{p}^L, \mathbf{p}^U] \end{aligned} \tag{P.2.4}$$

where depending on the constraint type (equality or inequality), the vectors $\mathbf{c}_i^L$ and $\mathbf{c}_i^U$ are appropriately defined. For each scenario i, the concatenated shooting node parameters are defined as $\bar{\boldsymbol{w}}_i := [\boldsymbol{w}_{i,0}^\top, \ldots, \boldsymbol{w}_{i,n}^\top, \mathbf{d}_i^\top]^\top$, where for each shooting interval we define $\boldsymbol{w}_{i,j} := [\mathbf{x}_{i,j}^\top, \mathbf{u}_{i,j}^\top]^\top$. Again, the NLP as posed produces a highly sparse diagonal structure within the constraint Jacobian matrix, where for each scenario block we have an additional sparse matrix for each shooting interval. Once constructed, the objective and constraint functions and corresponding objective gradient vector and block sparse constraint Jacobian matrices can be passed to an appropriately selected nonlinear programming solver. Typically, the multiple-shooting technique utilizes SQP-based algorithms, as these algorithms are often designed with efficient Lagrangian Hessian approximation schemes, thus allowing the user to provide only first-order derivatives. Interior point algorithms, on the other hand, are often best utilized with the explicit definition of second-order derivatives [31, 32]. Thus, to effectively apply an interior point NLP solver to a multiple-shooting formulation, one would need to further generate second-order sensitivity information at an added computational expense.

2.3.4 Algorithm Parallelization and Implementation

The multiple-shooting algorithm benefits from a naturally parallel structure that does not require any additional decomposition techniques. This natural decoupling of each shooting interval is induced by the introduction of the optimization parameters $\mathbf{x}_j$ for $j = 0, \ldots, n-1$, and allows the ODE in each shooting interval to be independently solved in parallel. Parallel multiple-shooting implementations have been explored to varying degrees by Kiehl [33] for ODE models, Leineweber et al. [6] for DAE models, Jeon [34] for DAE models with parallel adjoint sensitivity computation, and Bachmann et al. [35] within a Modelica modeling environment.

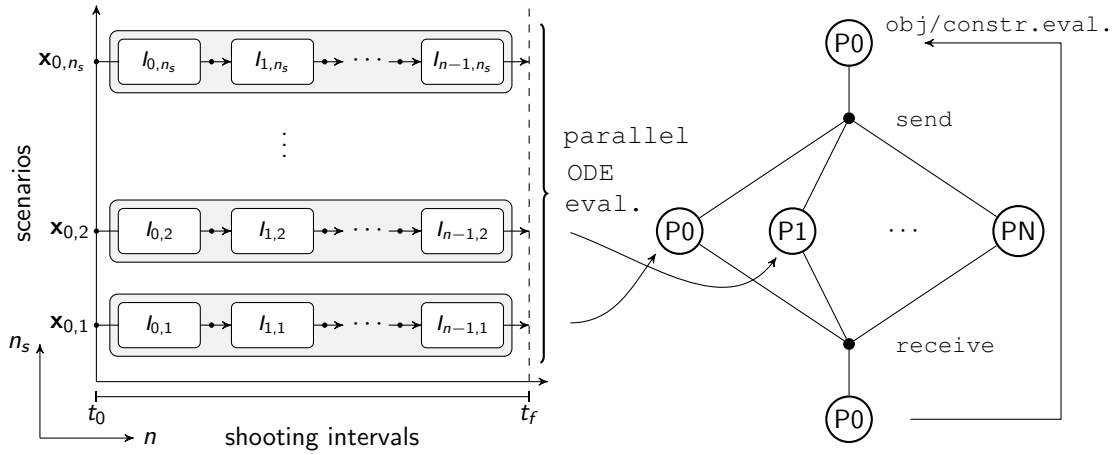

Figure 2.1: Implementation of ODE solution within the parallel multiperiod multiple-shooting dynamic optimization algorithm

The main difference in our proposed optimization formulation and corresponding implementation is that we have incorporated an additional layer of parallelization in terms of the individual scenarios used within the multiperiod approach. This concept can be visualized in Figure 2.1, where for a single scenario realization i each integration task ($I_{j,i}$) for each shooting interval j over the entire time horizon is stacked into a vector and then this is repeated for all scenarios n_s. The result is a large $n \cdot n_s$ dimensional vector of independent

integration tasks which can be broken up and solved in parallel using several processors (P_i for $i = 0, \ldots, N$). However, for numerous integration tasks per ODE evaluation (e.g., $100 - 1000$), it is more likely that the tasks are divided into several evenly distributed blocks (in terms of number, not computation work load) and then off-loaded to a parallel computing server for evaluation. To reap the full potential of the parallel computation, the processors must be loaded with a sufficient amount of computation work. Note that our implementation goes beyond the standard parallel multiple-shooting approach [33] which is confined to parallelizing only the shooting intervals.

A prototype implementation of the multiperiod multiple-shooting dynamic optimization algorithm was created in the `Matlab` scripting language where the NLP and DAE solutions were performed using third-party solvers. The parallelization of the ODE solution was implemented using the `Matlab` Parallel Computing Toolbox (PCT). Additionally, the codes were run on a server within the Shared Hierarchical Academic Research Computing Network (`SHARCNET`), which is a high performance parallel computing network comprising several universities within southern Ontario, Canada. Submission of the codes to the server was performed using the `Matlab` Distributed Computing Server (DCS) whereby the codes were sent to a scheduler and appropriately executed with a specified level of resource allocation (i.e., amount of RAM and number of processors). The NLP solution utilized the sparse reduced SQP solver `SNOPT` (version `7.2-11`) [36], while for the integration of the dynamic model we utilized the tools available from the `SUNDIALS` suite of solvers (version `2.5.0`) [37]. In particular, for explicit ODE models, our implementation uses the provided `Matlab` interface to `CVODES`. The particular focus of parallelization within our implementation is on the ODE and sensitivity solution as opposed to the NLP solution; the former is known to consume up to 90 percent of the total program computation time on serial machines [38]. Accordingly, the NLP solution is performed in serial (on a single processor), while for each function evaluation (objective and constraints) the embedded dynamic model and corresponding sensitivity solution is solved in parallel. Timing measurements were taken

based on the wall clock time of: (1) the full program using `Matlab`'s `tic/toc` technique; (2) the in-solver NLP solution; and (3) the parallel ODE and sensitivity solution (which includes the serial vector formation of the objective/constraints and sparse derivative matrices).

2.4 Example Problems

In this section we apply the parallel multiperiod multiple-shooting approach to two different benchmark design and control problems of different magnitude and complexity. The purpose here is to assess potential performance improvements through a parallel computing implementation. Our first example uses a modified version of the Newell and Lee [39] evaporator benchmark model described by Kookos and Perkins [40], where we consider determining an economically optimal design and controller tuning parameters subject to uncertain input flow rate and composition disturbances. The second example utilizes a larger model of a continuous binary distillation column discussed by Schweiger and Floudas [24]. Each example comprises a set of nonlinear differential-algebraic equations describing the process model, path constraints on the state and closed-loop manipulated variables, and a nonlinear economic objective function that includes design and operating costs. Closed-loop control is achieved via multi-loop PI controllers based on a fixed control structure.

For the particular multiperiod design and control formulations which embed continuous closed-loop PI controllers considered in this chapter, we define the first stage variables as equipment dimensions and controller tuning parameters (i.e., proportional gain K_c, reset time τ_I, and output set-points y_{set}). The second stage decision variables include the closed-loop manipulated variables $\mathbf{u}_i(t)$ over each scenario i, which are implicitly determined by the PI controller equations defined as,

$$\begin{aligned} u_i(t) &= \bar{u}_i + K_c(y_i(t) - y_{\text{set}}) + \tfrac{K_c}{\tau_I} I_i(t) \\ I_i(t) &= \textstyle\int_{t_0}^{t} (y_i(\tau) - y_{\text{set}})\, \mathrm{d}\tau \end{aligned} \tag{2.14}$$

where the controller bias terms $\bar{u}_i$ are the explicit second stage optimization decision variables for each scenario $i = 1, \ldots, n_s$. These control actions allow for recourse once the uncertainty is resolved, and in this chapter we consider uncertainty solely within the disturbance inputs $\mathbf{v}(t)$, which are defined as a function of each uncertain parameter $\boldsymbol{\theta}$ over all scenario realizations. The disturbance inputs are applied as step inputs to the system, which require two uncertain parameters per disturbance trajectory representing the initial value before the step is applied (defined previously in vector form for all disturbance variables as $\mathbf{v}_0$), and a second parameter corresponding to the step magnitude (defined previously as $\Delta\mathbf{v}$). These uncertain parameters are generated from a uniform distribution by sampling between a specified upper and lower bound for $\mathbf{v}_0$ and $\Delta\mathbf{v}$. Throughout the chapter we adopt the terminology of a complete disturbance step realization (or vector of different disturbances) representing a single scenario realization. For example, 10 scenario realizations ($n_s = 10$) of a set of two independent disturbance variables ($n_v = 2$) would correspond to the generation of $2 \cdot n_v \cdot n_s = 40$ random variables.

2.4.1 Example 1: Evaporator Process Design

The evaporation process considered for this first example is depicted in Figure 2.2, where it is desired to remove a volatile liquid from a solution to yield a high concentration of a non-volatile solute. The process is composed of a pressurized evaporation vessel, a separator or settling tank, a recirculation pump and an overhead condenser (heat exchanger). For convenience the chosen model equations used in this study are provided in Section 2.6.

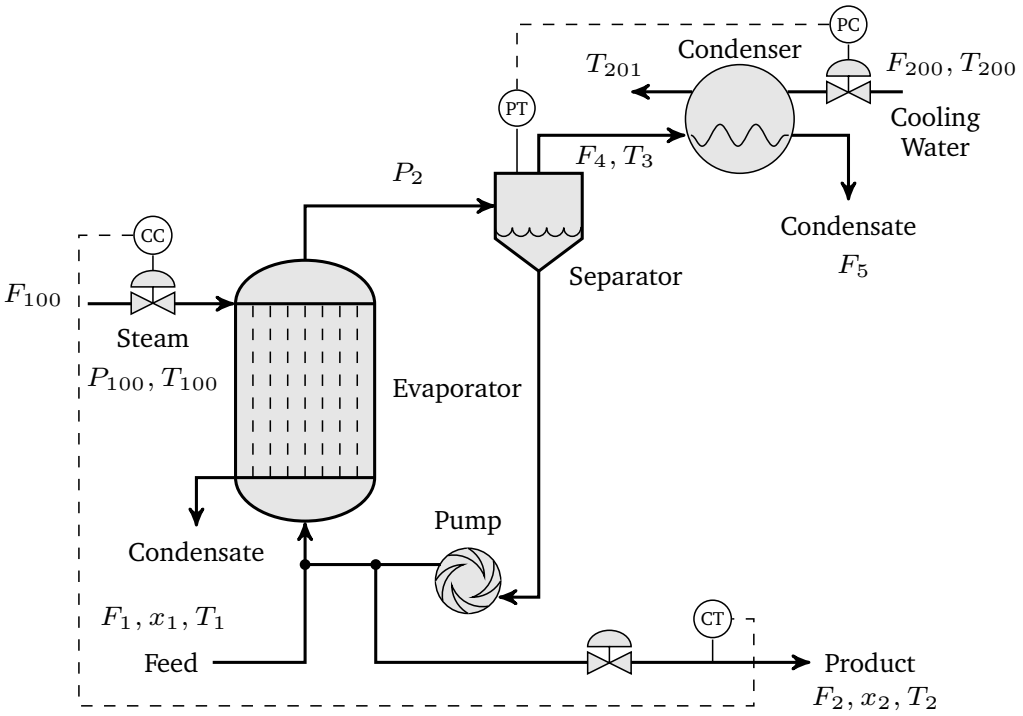

Figure 2.2: Example 1 – evaporator process schematic and control structure

Table 2.1: Example 1 – evaporator model variable definitions

F_1: feed flow rate (kg/min)	P_2: operating pressure (kPa)
F_2: product flow rate (kg/min)	P_{100}: steam pressure (kPa)
F_4: vapor flow rate (kg/min)	T_1: feed temperature (°C)
F_5: condensate flow rate (kg/min)	T_2: product temperature (°C)
F_{100}: steam flow rate (kg/min)	T_3: vapor temperature (°C)
F_{200}: cooling water flow rate (kg/min)	T_{100}: steam temperature (°C)
x_1: feed composition (%)	T_{200}: cooling water inlet temperature (°C)
x_2: product composition (%)	T_{201}: cooling water outlet temperature (°C)

The optimization design parameters $\mathbf{p}$ comprise the evaporator and heat exchanger areas (combined with the heat transfer coefficient, U) (UA_1, UA_2), PI controller tuning parameters (K_{c_i}, τ_{I_i}, $i = 1, 2$) and output variable set-points ($\bar{x}_2, \bar{P}_2$). The differential state variables $\mathbf{x}(t)$ are the evaporator product composition (x_2) and operating pressure (P_2), which are

also considered as output variables to be controlled using the manipulated variables $\mathbf{u}(t)$ of inlet steam pressure (P_{100}) and cooling water flow rate (F_{200}). Additionally, we consider two uncertain external input disturbances $\mathbf{v}(t)$ of feed flow rate (F_1) and composition (x_1). A detailed listing of the variables shown in Figure 2.2 is provided in Table 2.1, while a further listing of variable and parameter specifications is provided in Section 2.6.

The chosen design and control formulation can be described as a continuous multiperiod dynamic optimization problem according to the following equations,

$$
\begin{aligned}
\min_{\mathbf{d}_i \,\forall\, i,\, \mathbf{p}} \quad & \mathcal{J} := C_{\text{cap}}(\mathbf{p}) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot C_{\text{op}}(\mathbf{x}_i(t_f), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_f) \\
\text{st}: \quad & \text{ODE model (5 eqns.)} \\
& \dot{\mathbf{x}}_i(t_0) = \mathbf{0} \\
& P_{100,i}(t) = \bar{P}_{100,i} + K_{c_1}(x_{2,i}(t) - \bar{x}_2) + \tfrac{K_{c_1}}{\tau_{I_1}} I_{1,i}(t) \\
& F_{200,i}(t) = \bar{F}_{200,i} + K_{c_2}(P_{2,i}(t) - \bar{P}_2) + \tfrac{K_{c_2}}{\tau_{I_2}} I_{2,i}(t) \\
& F_{1,i}(t) = \bar{F}_{1,i} + \Delta\bar{F}_{1,i}\,\gamma(t) \\
& x_{1,i}(t) = \bar{x}_{1,i} + \Delta\bar{x}_{1,i}\,\gamma(t) \\
& \varepsilon_{\text{ise},i}(t_f) \leq \epsilon \\
& x_2^L - x_{2,i}(t) \leq 0 \\
& x_{2,i}(t) - x_2^U \leq 0 \\
& P_2^L - P_{2,i}(t) \leq 0 \\
& P_{2,i}(t) - P_2^U \leq 0 \\
& P_{100,i}(t) - P_{100}^U \leq 0 \\
& F_{200,i}(t) - F_{200}^U \leq 0 \quad \forall\, t \in [t_0, t_f],\ i = 1, \ldots, n_s
\end{aligned}
\tag{E.2.1}
$$

We note that the formulation is continuous in the optimization parameters as we consider a fixed plant topology and control structure. The multiperiod algorithm that we consider comprises a single optimization stage (i.e., a single NLP formulation), in which a number

of scenarios is introduced to approximate the uncertainty in the disturbance inputs and hence the stochastic nature of the problem. The investigation to be performed considers the effect on the computational performance of parallelization over uncertainty scenarios and multiple-shooting intervals.

Within the above formulation, the objective is to determine an economically optimal and control feasible design, where it is desired to keep the process at steady-state prior to the input disturbance, and subsequently to drive the process back to the original steady-state using feedback control (i.e., disturbance rejection). The formulation follows the structure of a two-stage stochastic program, where the first stage costs correspond to the design, given here as the capital cost of the plant as a function of the design parameters ($C_{\text{cap}}(\mathbf{p})$), and the second stage costs are dependent on the operating costs of the plant for a given design ($C_{\text{op}}(\cdot)$). The stochastic aspect of the problem is captured using a weighted deterministic formulation where the associated weights are specified equally for each scenario as $\mathsf{w}_i = 1/n_s$. The equality constraints within this formulation are the dynamic process model equations; the initial state variable conditions at time t_0 (i.e., steady-state constraints); the closed-loop PI controller equations; and the single-step disturbance approximation equations. The inequality constraints comprise an end-point constraint on the integrated squared error control performance metric (ε_{ise}) and path constraints on the state (x_2, P_2) and manipulated variables (P_{100}, F_{200}). The experiments performed considered running the optimization algorithm using a high and low number of scenarios for the uncertain disturbance variables $\boldsymbol{\theta} := [\mathbf{v}_0^\top, \Delta\mathbf{v}^\top]^\top$, where $\mathbf{v}_0 := [\bar{F}_1, \bar{x}_1]^\top$ and $\Delta\mathbf{v} := [\Delta\bar{F}_1, \Delta\bar{x}_1]^\top$. The idea is to emulate a high and low work load on the processors in order to examine its effect on the computation speedup and efficiency.

We make several additional remarks to elucidate the optimization formulation and solution procedure:

- The process model can be generally stated as a system of differential-algebraic equations

(DAEs); however, for the examples shown in this chapter, the algebraic equations for the controller and disturbance inputs (as defined in formulation E.2.1) were eliminated via substitution of explicit algebraic variable expressions into the differential equations thus only requiring the solution of an explicit ODE system.

- The steady-state constraints were imposed by setting the explicit right-hand-side of the ODEs to zero at t_0 (i.e., $\dot{\mathbf{x}}(t_0) = \mathbf{0} \mapsto \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}}, \bar{\mathbf{v}}, \mathbf{p}, t_0) = \mathbf{0}$).

- Bound constraints were introduced on the manipulated variables by adding appropriate inequality constraints at each shooting node.

- An end-point constraint on the integrated squared error control performance metric was introduced to influence the control performance. The metric is defined as,

$$\varepsilon_{\text{ise}}(t_f) := \int_{t_0}^{t_f} \left\{ (x_2(t) - \bar{x}_2)^2 + (P_2(t) - \bar{P}_2)^2 \right\} \mathrm{d}t \tag{2.15}$$

  Using such an approach allows one to directly set a tolerance on the control performance via an inequality end-point constraint ($\varepsilon_{\text{ise}}(t_f) \leq \epsilon$).

- The variables and constraints of this formulation can be broken down as follows: multiple-shooting equality constraints for n intervals (including initial constraints at t_0) of dimension $n_c := n_x n_s (n+1)$ (where n_x are the number of continuity constraints at the shooting nodes and n_s are the number of scenarios), similarly we have the same number of shooing node variables ($\mathbf{x}_{j,i}$); closed-loop control signals at t_0 ($P_{100,i}(t_0)$, $F_{200,i}(t_0)$) must be set to their controller bias values ($\bar{P}_{100,i}$, $\bar{F}_{200,i}$) via equality constraints of dimension $n_{h_1} = n_u n_s$, again we have the same number of decision variables as equations; finally, we have equality constraints at t_0 stating that the initial output variables be equal to their set-points of dimension $n_{h_2} = n_y n_s$. For this last set of equality constraints, the outputs are simply the states, which are already specified via an initial steady-state equality constraint; thus, these output constraints may be removed. The remaining constraints are inequalities for path or end-point constraints, and the

main degrees of freedom for the formulation are the design variables UA_1, UA_2, K_{c_j}, τ_{I_j}, $j = 1, 2$, $\bar{x}_2$, and $\bar{P}_2$.

- In order to remove any obscurity in the solution timing results incurred by possible automatic derivative generation techniques, the first-order derivatives of the NLP objective/constraints with respect the optimization parameters were specified analytically and provided directly to the optimization algorithm.

Table 2.2: Example 1 – evaporator optimal design and control parameters

	Initial Guess	Bounds	Optimal Solution		
Parameter ($\mathbf{p}$) †	$\mathbf{p}_g$	$\in [L, U]$	$n_s = 1$	$n_s = 5$	$n_s = 50$
$\mathcal{J}$ (\$/year)	–	–	71690.52	71556.57	71769.11
UA_1 [kW/K]	9.6	$\in [2, 15]$	4.652	4.795	4.932
UA_2 [kW/K]	6.8	$\in [2, 15]$	7.430	7.584	7.734
K_{c_1}	-20	$\in [-50, 50]$	-50	-50	-50
K_{c_2}	20	$\in [-50, 500]$	216.75	500	500
τ_{I_1}	2	$\in [1, 5]$	1.07	1	1
τ_{I_2}	2	$\in [1, 5]$	5.00	1	1
$\bar{x}_2$ [%]	29	$\in [25, 30]$	25.06	25.23	25.25
$\bar{P}_2$ [kPa]	49	$\in [40, 80]$	40.00	40.04	43.07

† Note, the controller bias parameters ($\bar{P}_{100}$, $\bar{F}_{200}$) are scenario dependent optimization parameters as defined by $\mathbf{d}_i$, not shown. Initial guesses defined as $\bar{P}_{100} = 193.45$ kPa, $\bar{F}_{200} = 207.32$ kg/min.

Before considering the potential computation speedup via our parallel solution implementation, we first assess the optimization solution. Table 2.2 lists the design parameters, their initial guesses and lower and upper bounds as posed to the optimization solver. Additionally, stated are the final objective and parameter values for 1, 5 and 50 scenario realizations of the

uncertain disturbance variables.

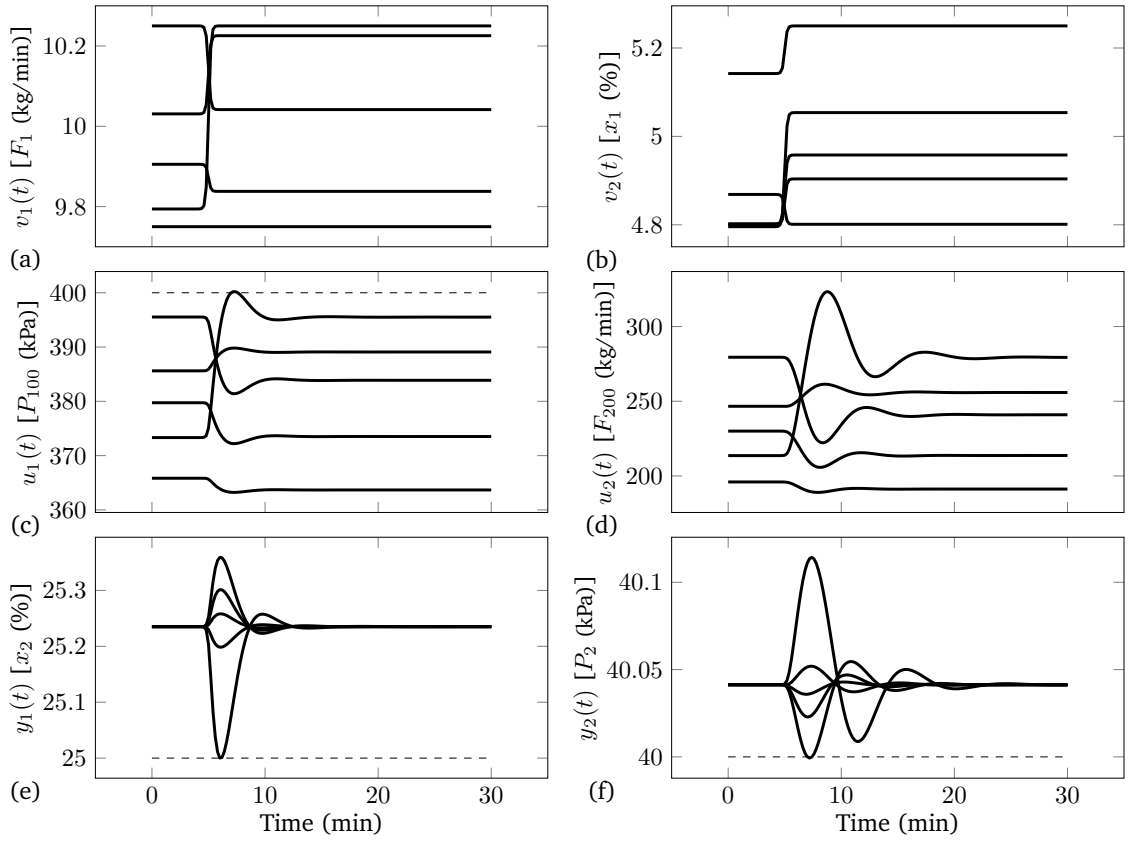

Figure 2.3: Example 1 – evaporator dynamic optimization trajectories for $n_s = 5$: (a)–(b) uncertain disturbance inputs $\mathbf{v}(t)$; (c)–(d) closed-loop inputs $\mathbf{u}(t)$; (e)–(f) controlled outputs $\mathbf{y}(t)$

In Figure 2.3 we illustrate the optimal solution trajectories for 5 scenarios, which corresponds to 5 sets of variable trajectories. For both the closed-loop inputs $\mathbf{u}(t)$ and controlled outputs $\mathbf{y}(t)$, we see that the discretized inequality path constraints are not violated, and for variables P_{100}, x_2 and P_2 the constraint becomes active for a short period of time when the output variables overshoot their set-points (see dashed lines in Figure 2.3). Furthermore, in Table 2.2 we see that for the desired control performance tolerance ϵ (specified a priori) and the chosen parameter bounds, the PI controller parameters K_c and τ_I converged to either an upper or lower bound at the optimal objective value solution. Relaxing the controller

parameter bounds did not significantly change the results as the parameter values remained active at the bounds upon convergence to an optimal solution. This behavior is likely a consequence of the constraint on the integrated squared error where we set a relatively low tolerance value (e.g., $\epsilon = 0.1$) which would induce a rather aggressive control action (i.e., high gain magnitudes).

The parameters obtained using a single scenario realization represent nominal disturbance conditions and are presented for comparison to the more robust solutions which utilize multiple scenarios. There are a few notable aspects that can be observed from the optimal design solution. First, as the number of scenarios n_s is increased, the optimal surface area of both the evaporator (UA_1) and condenser (UA_2) also increases. This behavior reflects an increasing conservativeness of the design, with increasing scenarios, which implicitly supplement the necessary control actions to reject the uncertain disturbance inputs. In other words, in order for the control system to adequately handle all possible disturbance scenarios, the overall equipment surface areas must be increased at an economic penalty to dampen the disturbance effects. Another related observation is that to ensure the path constraints on the controlled variables are not violated, the output set-points (i.e., design solution) must back off from their bounds to adequately account for the anticipated control action used to reject the uncertain disturbance realizations.

For this example, the algorithm was set up using $n = 30$ shooting intervals over a time horizon of 30 minutes and a low and high number of scenarios n_s defined according to Table 2.3. This corresponds to $n \cdot n_s$ independent integration tasks which are solved in parallel. Thus, we are parallelizing over both the shooting intervals and scenario realizations, by viewing each independent integration task as parallelizable and appropriately grouping these tasks into blocks such that a relatively even amount of tasks are distributed among the available processors. The optimization problem is nonlinear and non-convex, and is solved in serial using the `SNOPT` solver. To solve the embedded dynamic model equations and associated sensitivity equations, we used the `CVODES` solver, where the model equations

are posed as an explicit ODE. The design problem being considered has 8 main degrees of freedom for the design and control variables (UA_1, UA_2, $\bar{x}_2$, $\bar{P}_2$, K_{c_j}, τ_{I_j}, $j = 1, 2$). Additionally, the PI controller bias parameters are introduced as degrees of freedom for each specific scenario. Since we required the system to be at steady-state initially, and each initial disturbance realization ($\bar{F}_{1,i}$, $\bar{x}_{1,i}$) is defined a priori, then the controller bias must be introduced for each scenario to appropriately compensate for the initial disturbance values.

The potential computation speedup attainable using the parallelized multiperiod multiple-shooting implementation will now be investigated. As previously discussed, the portion of the algorithm that we are parallelizing is the embedded ODE solution, as opposed to the NLP algorithm. To examine the potential speedup and efficiency of using more processors to evaluate the embedded dynamic model, we have selected the number of processors according to $\mathrm{N} = 2^k$ where $k = 0, 2, 3, 4, 5$. The computation speedup and processor efficiency are computed according to the following equations,

$$\begin{aligned} \text{Speedup } (\mathrm{S}) &= \frac{\text{user function eval. time } (\mathrm{N}=1)}{\text{user function eval. time}(\mathrm{N}=2^k)} = \frac{\mathrm{T}_{\text{serial}}}{\mathrm{T}_{\text{parallel}}} \\ \text{Efficiency } (\mathrm{E}) &= \text{Speedup}/\mathrm{N} = \frac{\mathrm{T}_{\text{serial}}}{\mathrm{N}\,\mathrm{T}_{\text{parallel}}} \end{aligned} \tag{2.16}$$

where we use the NLP constraint and objective (i.e., user) function evaluation time (which includes the parallel ODE and sensitivity solution and a much smaller serial constraint/objective construction component) as a basis of performance measurement, as opposed to the total program wall clock time which includes the serial in-solver NLP solution time. In these definitions, $\mathrm{T}_{\text{serial}}$ represents the constraint/objective evaluation time using a single processor and $\mathrm{T}_{\text{parallel}}$ corresponds to the evaluation time using multiple processors. The computation speedup is one measure of potential performance improvement, and under ideal conditions where the program can be fully parallelized and the processors are sufficiently utilized, the addition of processors should reveal a parallel solution time corresponding to $\mathrm{T}_{\text{parallel}} := \mathrm{T}_{\text{serial}}/\mathrm{N}$ and thus produce a linear speedup of $\mathrm{S} = \mathrm{N}$ (see 45 degree solid line in Figure 2.4 (a)) and an efficiency of 100%. However, in practice these conditions

are not usually observed, as (1) there is an inherent aspect of the program that is serial, and (2) the increase of processors invariably introduces greater communication overhead which impedes both the speedup and efficiency. For example, for a given amount of work governed by the number of integration tasks, the true parallel solution time approximates to $\mathrm{T}_{\mathrm{parallel}} := \mathrm{T}_{\mathrm{serial}}/\mathrm{N} + \mathrm{T}_{\mathrm{overhead}}$, where $\mathrm{T}_{\mathrm{overhead}}$ is the total cumulative amount of time spent communicating data between processors (i.e., parallel overhead). This results in an observed speedup that is lower than the number of processors, with the relative drop off in speedup increasing as the number of processors increases. However, if we increase the amount of computation work, $\mathrm{T}_{\mathrm{overhead}}$ will grow more slowly relative to $\mathrm{T}_{\mathrm{serial}}/\mathrm{N}$, and allow for increased speedup and efficiency, and ultimately better utilization of the available resources. For a more detailed and in depth discussion on computation performance metrics and problem size versus machine size dependencies we refer the readers to Pacheco [41].

In our particular implementation, the timing captures the dynamic model solution whereby the combined initial condition vector $\mathbf{x}_{j,i}$ is communicated in a balanced manner to the available processors, the computation is performed in parallel, and then the state variable solution at the end of each shooting node, $\mathbf{x}_i(t_{j+1})$ for all $j = 0, \ldots, n-1$ and $i = 1, \ldots, n_s$, is communicated back to the master processor. For example, if we have $\mathrm{N} = 4$ processors and $n \cdot n_s$ integration tasks, then the work load for each processor is $n \cdot n_s/4$ where any remaining tasks are distributed among the four processors. The time to communicate the data incurs overhead (denoted as $\mathrm{T}_{\mathrm{overhead}}$), which accumulates for each NLP function evaluation; thus, for the entire NLP solution there will have been performed two communication calls between master and slave processors to solve the entire embedded dynamic model for each objective/constraint evaluation.

Table 2.3: Example 1 – evaporator optimization timings for parallel multiperiod algorithm

NLP size and solution statistics					Wall clock time (sec) for N processors *				
n_s	#vars †	n_h	n_g	#iter ‡	N = 1	N = 4	N = 8	N = 16	N = 32
5	793	795	315	37	318.3	83.5	41.1	22.8	17.7
50	7858	7950	3150	36	3158.2	799.2	387.6	189.3	110.8

† discretized formulation w/ $n = 30$ shooting intervals, n_h equality and n_g inequality constraints, respectively; ‡ no. of major iterations using SNOPT SQP solver w/ optimality, feasibility tolerance of 1×10^{-4} and 1×10^{-6}, respectively; * average wall clock times over 3 experiments.

Table 2.3 lists the optimization setup in terms of the number of scenarios used (n_s); the total number of discretized optimization variables (#vars); the total number of equality (n_h) and inequality (n_g) constraints, respectively. Additionally, we list the total number of major SQP iterations and the average total program wall clock times (combined NLP and ODE solution timings) using an incremental number of computing processors. The optimization algorithm was run using a local parallel computing server within the SHARCNET cluster which uses two 12 core 2.2 GHz AMD Opteron chips and 32 GB of memory per computing node, and solution timings are reported based on an average of several experiments. Given a single node of 24 processors, we required the use of two computing nodes for the 32 processor trials. Note that the wall clock timings should be interpreted as the potential computation speedup from a relative perspective between each of the number of processors used and not as a reflection of the NLP or ODE solver solution speed. We make this statement since the results come from a prototyped implementation within the Matlab scripting language, which by its nature contains significant overhead not present in compiled programming languages. Nevertheless, we observe that when using a single processor with $n_s = 5$ scenarios, the optimization algorithm requires approximately an average of 8.7 seconds per major SQP iteration and when increased to 16 processors the time per iteration decreased to about 0.6 seconds which

indicates an average speedup of 15. When the number of scenarios is increased 10 fold we observe an improved relative speedup of approximately 16, indicating near linear speedup and close to perfect scalability of the embedded model solution.

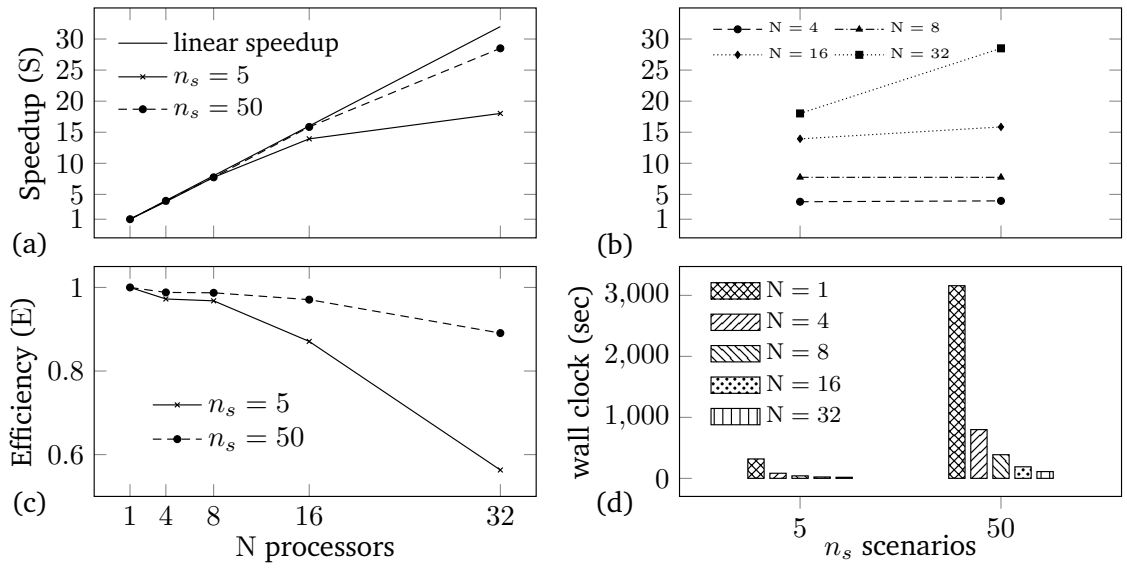

Figure 2.4: Example 1 – parallel multiperiod multiple-shooting algorithm speedup and efficiency with increasing number of processors and scenarios

Figure 2.4 provides plots of the speedup and efficiency and depicts the behavior when the number of scenarios is increased. Note that this figure is not directly based on the wall clock times reported in Table 2.3, and instead uses the cumulative average wall clock time of constraint/objective evaluation, as defined in Equation 2.16, excluding the serial in-solver NLP evaluation time. In Figure 2.4 (a) and (c), when we increase the number of processors N from 16 to 32 at a fixed amount of work ($n_s = 5$), which doubles the overall communication overhead, we see a significant decrease in speedup and efficiency. This behavior occurs because the communication time is relatively large, for the given amount of computation work, compared to the serial solution time. When we increase the problem size from $n_s = 5$ to $n_s = 50$, we see significantly improved speedup and efficiency from $\mathrm{N} = 16$ to $\mathrm{N} = 32$. This is due to a relatively small increase in communication overhead

$\mathrm{T}_{\mathrm{overhead}}$ compared to the serial solution time $\mathrm{T}_{\mathrm{serial}}$. Figure 2.4 (b) further illustrates the behavior of speedup when we increase the computation work at a fixed number of processors. Here we see improved speedup using $\mathrm{N} = 16$ and $\mathrm{N} = 32$ when increasing n_s from 5 to 50, with an unnoticeable effect using lesser processors. Therefore, the effect of communication overhead can be reduced by introducing more computation work (i.e., greater loading of the processors). Ultimately, when utilizing 32 processors on a problem with 50 scenarios, we achieve a speedup in the dynamic model solution of approximately 29 times at 89% processor efficiency.

2.4.2 Example 2: Binary Distillation Process Design

The second example considered is the design and control of a continuous binary distillation column. Our purpose here is to provide further demonstration of our multiperiod multiple-shooting algorithm using a larger model and investigate the algorithm performance in terms of speedup and efficiency, and additionally to look at the timing breakdown between the embedded dynamic model and nonlinear program solution. The design and control problem follows in a similar manner to the previous example, where we fix the control structure and solve an economic optimization formulation for the distillation column diameter and PI controller tuning parameters. For illustrative purposes, we consider a fixed/specified column height, tray dimensions, feed tray location, and reboiler and condenser surface areas. The process is illustrated in Figure 2.5, where F and z correspond to uncertain disturbance inputs for feed flow rate and composition, respectively; R and V represent the manipulated reflux and boil-up flow rates; and x_0 and x_{n_t+1} are controlled output compositions within the bottoms and distillate product.

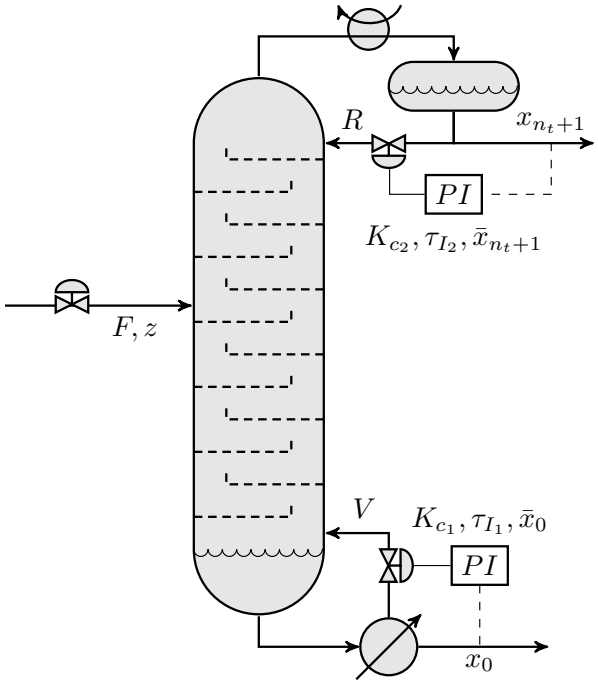

Figure 2.5: Example 2 – distillation process schematic and control structure

The design and control optimization parameters of interest include the column diameter D_{col}, the PI controller tuning parameters K_{c_i}, τ_{I_i} for $i = 1, 2$ and output set-points $\bar{x}_0$, $\bar{x}_{n_t+1}$. The dynamic model of the distillation column contains roughly 40 differential equations and a similar number of algebraic equations (see Section 2.7 for a detailed listing). The model is solved as an ODE by eliminating the algebraic equations via the explicit substitution of the algebraic state variables into the differential equations. The resulting design and control

formulation is posed in a general manner as follows,

$$
\begin{aligned}
\min_{\mathbf{d}_i\,\forall\, i,\,\mathbf{p}} \quad & \mathcal{J} := C_{\text{cap}}(\mathbf{p}) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot C_{\text{op}}(\mathbf{x}_i(t_f), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_f) \\
\text{st}: \quad & \text{ODE model (40 eqns.)} \\
& \dot{\mathbf{x}}_i(t_0) = \mathbf{0} \\
& V_i(t) = \bar{V}_i + K_{c_1}(x_{0,i}(t) - \bar{x}_0) + \tfrac{K_{c_1}}{\tau_{I_1}} I_{1,i}(t) \\
& R_i(t) = \bar{R}_i + K_{c_2}(x_{n_t+1,i}(t) - \bar{x}_{n_t+1}) + \tfrac{K_{c_2}}{\tau_{I_2}} I_{2,i}(t) \\
& F_i(t) = \bar{F}_i + \Delta\bar{F}_i\,\gamma(t) \\
& z_i(t) = \bar{z}_i + \Delta\bar{z}_i\,\gamma(t) \\
& \varepsilon_{\text{ise},i}(t_f) \leq \epsilon \\
& x^L_{n_t+1} - x_{n_t+1,i}(t_f) \leq 0 \\
& x_{0,i}(t_f) - x^U_0 \leq 0 \\
& D^{\min}_{\text{col}}(\bar{V}_i) - D_{\text{col}} \leq 0 \quad \forall\, t \in [t_0, t_f],\ i = 1, \ldots, n_s
\end{aligned}
\tag{E.2.2}
$$

where we use an economic objective function, similar to the previous example, consisting of the capital cost of the column tray/shell construction and the operating costs associated with the cooling water and steam flow rates used in the condenser and reboiler, respectively. Additionally, we pose equality path constraints for V and R; disturbance input equality constraints with uncertain parameters $\boldsymbol{\theta} := [\mathbf{v}_0^\top, \Delta\mathbf{v}^\top]^\top$, where $\mathbf{v}_0 := [\bar{F}, \bar{z}]^\top$ and $\Delta\mathbf{v} := [\Delta\bar{F}, \Delta\bar{z}]^\top$; a control performance inequality end-point constraint to explicitly influence the disturbance rejection; end-point constraints on the controlled output variables; and a column flooding constraint (as defined by the final inequality in formulation E.2.2) which ensures that the column maintains a minimum diameter based on the vapor boil-up rate. A detailed formulation of the objective function is provided in Section 2.7.

Table 2.4: Example 2 – distillation optimal design and control parameters

	Initial Guess	Bounds	Optimal Solution		
Parameter ($\mathbf{p}$) †	$\mathbf{p}_g$	$\in [L, U]$	$n_s = 1$	$n_s = 10$	$n_s = 20$
$\mathcal{J}$ (\$/year)	–	–	20497.49	21077.36	21023.70
D_{col} [m]	0.75	$\in [0.1, 2]$	0.721	0.745	0.746
K_{c_1}	0.74	$\in [-50, 50]$	1.376	2.476	6.747
K_{c_2}	-4.0	$\in [-50, 50]$	-4.185	-5.113	-10.501
τ_{I_1}	3.5	$\in [1, 9]$	1.181	2.206	2.992
τ_{I_2}	8.0	$\in [1, 9]$	2.671	1.998	3.159
$\bar{x}_0$	0.01	$\in [0.005, 0.05]$	0.05	0.05	0.05
$\bar{x}_{n_t+1}$	0.97	$\in [0.85, 1]$	0.98	0.98	0.98

† Note, the controller bias parameters ($\bar{R}$, $\bar{V}$) are scenario dependent optimization parameters as defined by $\mathbf{d}_i$, not shown. Initial guesses defined as $\bar{R} = 1.0024$ kmol/min, $\bar{V} = 1.5426$ kmol/min.

The optimization design parameter initial guesses and bounds are listed in Table 2.4. Additionally stated are the optimal objective and parameter values for nominal disturbance conditions and uncertain disturbance realizations constructed from 10 and 20 scenarios. The objective function solution indicates the expected result that the economics worsen as uncertainty is introduced into the formulation. In other words, in order to account for possible uncertain disturbance realizations, the distillation column diameter must be increased from its nominal value at an economic expense. Also notable, from the listed parameter solution, are the non-unique values of the controller tuning parameters between scenario levels of 10 and 20. These non-unique values reflect the multiple possible tuning settings to adequately control the nonlinear system, which is a known behavior when including both the controller gain and integral reset time as degrees of freedom in the optimization formulation. Figure 2.6 provides the imposed disturbances realizations $\mathbf{v}(t)$ (with uncertain initial and

final step values), the optimal manipulated variables $\mathbf{u}(t)$ and optimal output trajectories $\mathbf{y}(t)$ for a problem size of 10 scenario realizations per disturbance variable. Additionally, the multiple-shooting discretization consisted of $n = 25$ shooting nodes over a 50 minute time horizon, which amounts to total of $n \cdot n_s$ integration tasks $I_{i,j}$ distributed over the available processors.

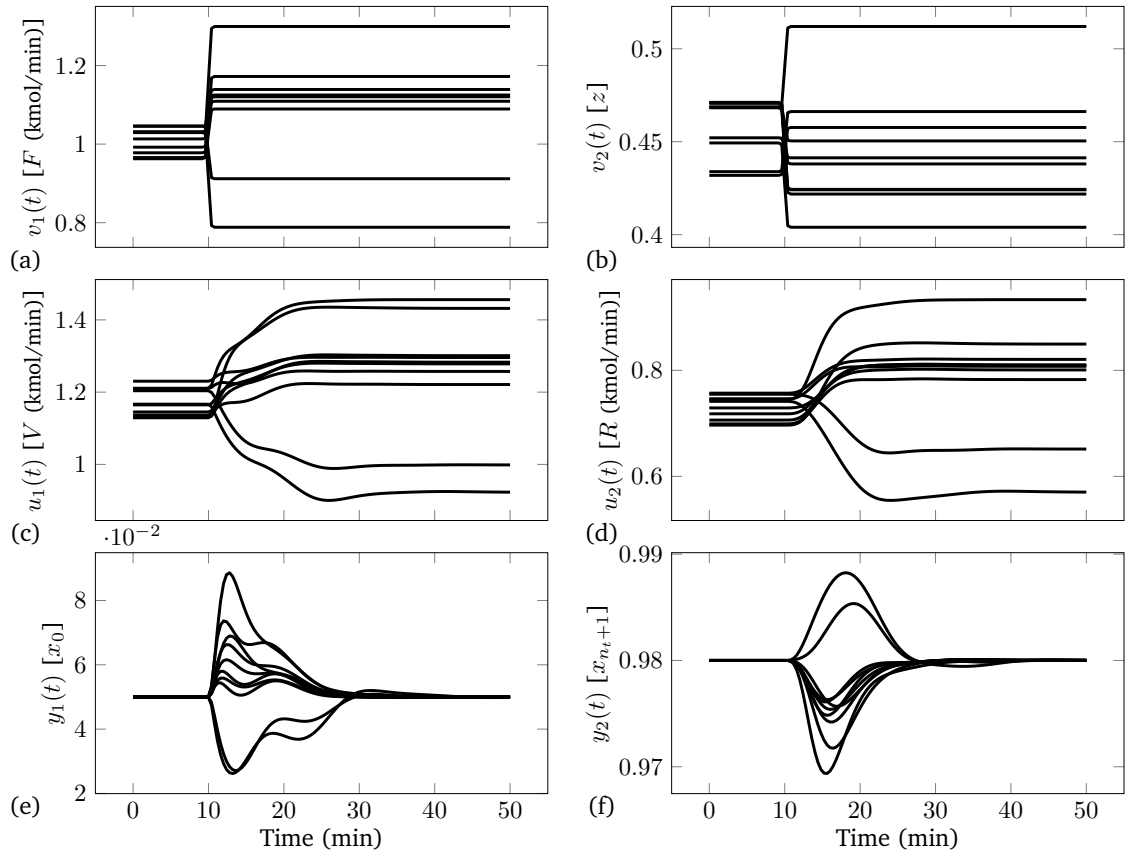

Figure 2.6: Example 2 – distillation dynamic optimization trajectories for $n_s = 10$: (a)–(b) uncertain disturbance inputs $\mathbf{v}(t)$; (c)–(d) closed-loop inputs $\mathbf{u}(t)$; (e)–(f) controlled outputs $\mathbf{y}(t)$

With an optimal solution established, we now turn to assessing the potential performance improvement via parallelization of this much larger distillation model, relative to the previous evaporator model. In a similar manner to the previous example, we provide in Table 2.5 the dimensions of the discretized NLP formulation, the number of major SQP iterations required

by SNOPT to meet the specified feasibility/optimality tolerances, and the total program wall clock times (combined NLP and ODE solution timings). The NLP formulation for $n_s = 20$ is about 2.5 times the size (in terms of variables/constraints) of the largest previous example, and required, using a single processor, about 1.8 hours of total computation time with an average solution time per major SQP iteration of 5.4 minutes. Increasing the number of processors to 16, we observed a reduction in computation time to 0.3 hours or an average of 0.9 minutes per iteration.

Table 2.5: Example 2 – distillation optimization timings for parallel multiperiod algorithm

NLP size and solution statistics					Wall clock time (sec) for N processors				
n_s	#vars †	n_h	n_g	#iter ‡	$\mathrm{N} = 1$	$\mathrm{N} = 4$	$\mathrm{N} = 8$	$\mathrm{N} = 16$	$\mathrm{N} = 32$
10	10167	10190	40	29	4188.25	1760.24	867.35	584.86	508.31
20	20327	20380	80	20	6457.16	2385.95	1406.15	1067.57	880.29

† discretized formulation with $n = 25$ shooting intervals, n_h equality and n_g inequality constraints; ‡ SNOPT SQP solver with optimality, feasibility tolerance of 1×10^{-4} and 1×10^{-5}

An assessment of the cumulative solution time for the dynamic model and sensitivity equations is provided in Figure 2.7. In a similar manner to the previous example, this figure was constructed excluding the in-solver serial NLP solution time, which is present in the wall clock times reported in Table 2.5. The behavior of the speedup and efficiency profiles follows the same trends seen in the previous case study, where for the considered scenario levels of $n_s = 10$ and 20, we see an approximate linear speedup up to $\mathrm{N} = 8$ followed by a slight decline to about $\mathrm{S} \approx 13$ and $\mathrm{E} > 80\%$ at $\mathrm{N} = 16$, ending with a significant drop off at $\mathrm{N} = 32$. In comparison to the largest example in the previous case study, this example involves a much larger NLP formulation which accordingly involves a greater fraction of serial computation. As a result, we see a slightly greater decline in speedup and efficiency at $\mathrm{N} = 16$ and 32, due to a smaller fraction of parallel work load, which indicates that for greater efficiency we

should either increase the work load or decrease the number of processors used. A related aspect when increasing the problem size (via an increase in scenario realizations) is the adverse effect created on the memory requirements within the serial NLP solver.

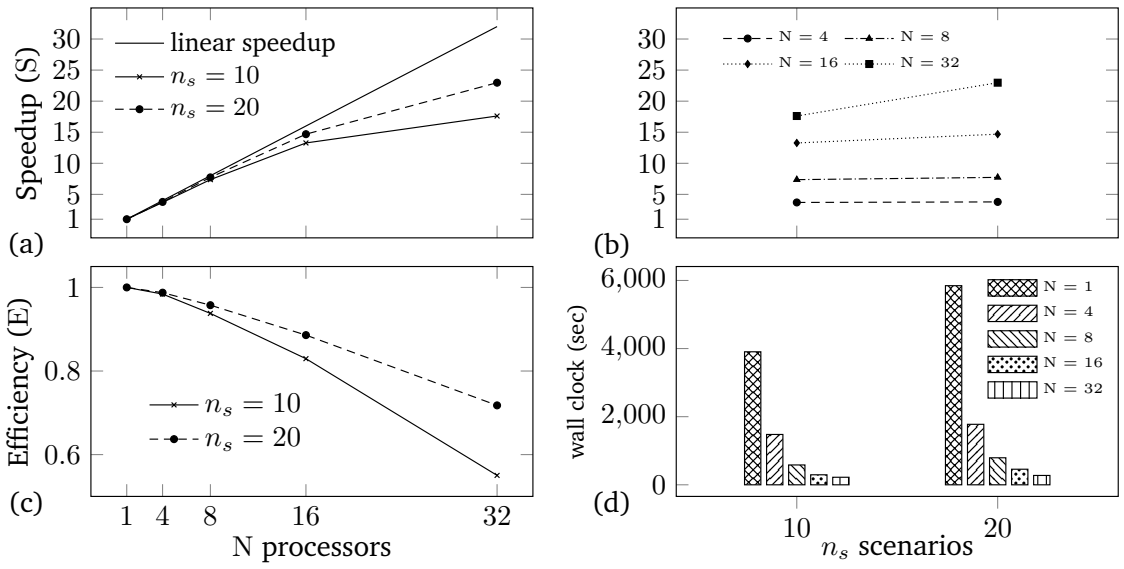

Figure 2.7: Example 2 – parallel multiperiod multiple-shooting algorithm speedup and efficiency with increasing number of processors and scenarios

An important computation aspect to highlight with this example is the relative computation time between NLP and ODE solutions. Accordingly, we break down the timing measurements into the total ODE and sensitivity solution (for all shooting intervals and scenarios) which includes the complete NLP objective/constraint construction, and the remaining in-solver NLP solution time. Note, the ODE solution is the parallelized portion of the algorithm which should reflect a decreased time with increased processors. Figure 2.8 plots a breakdown in time between each of these solutions from which we observe that the serial NLP portion of the code remains relatively constant as processors are added, while the parallel ODE portion enjoys significant speedup. Increasing the number of processors further, beyond that shown, can certainly improve even more the ODE solution time; however, doing so without additionally increasing the problem size is at a cost of reduced efficiency. Introducing

more processors is most beneficial if we increase the problem size and hence provide more computation work to the processors.

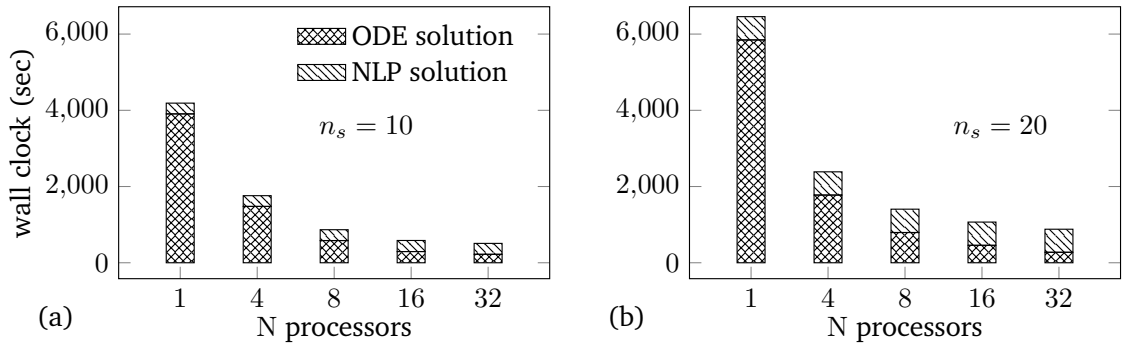

Figure 2.8: Example 2 – ODE and NLP solution wall clock timings with increasing number of processors and scenarios

2.5 Concluding Remarks

In this chapter, we have presented a novel parallel computing approach for large-scale dynamic optimization under uncertainty. A multiple-shooting approach was used for the dynamic optimization, and the uncertain parameter space discretized to yield a multiperiod formulation. The dynamic model and sensitivity equations corresponding to each shooting interval and scenario constitute independent integration tasks, well-suited for parallel processing. The formulation was applied to two integrated design and control case studies, where the objective was to determine design and controller tuning parameters that minimize the combined annualized capital and operating cost of the plant subject to uncertain disturbance inputs and the dynamic process model. The solution of the dynamic model was parallelized by evenly distributing the independent integration tasks from each shooting interval and scenario realization over several processors. Through parallelization, the embedded dynamic model function evaluations were able to be solved with near linear speedup as the number of processors were increased, before observing an expected drop off in efficiency. Ultimately, the cumulative model solution time could be reduced to less than half of the

total optimization program run time. Additionally, speedup and efficiency results indicate an appropriate number of processors to use for a given problem size and how to most efficiently scale up the computation resources with problem size.

2.6 Evaporator Model Equations

The evaporator model is based on the adaptation of the Newell and Lee [39] model given in Kookos and Perkins [40]. The dynamic closed-loop evaporator model is defined in terms of the differential $\mathbf{x}(t)$ and algebraic $\mathbf{z}(t)$ states; the closed-loop manipulated input variables $\mathbf{u}(t)$; the uncertain disturbance input variables $\mathbf{v}(t)$; the design parameters $\mathbf{p}$; and the controller bias parameters $\mathbf{d}$. The variables and parameters are defined as,

$$\mathbf{x}(t) := [x_2, P_2, I_1, I_2, \varepsilon_{ise}]^\top \tag{2.17}$$

$$\mathbf{z}(t) := [T_2, T_3, T_{100}, Q_{100}, F_{100}, F_4, F_2, Q_{200}, T_{201}, F_5]^\top \tag{2.18}$$

$$\mathbf{u}(t) := [P_{100}, F_{200}]^\top \tag{2.19}$$

$$\mathbf{v}(t) := [F_1, x_1]^\top \tag{2.20}$$

$$\mathbf{p} := [UA_1, UA_2, K_{c_1}, \tau_{I_1}, \bar{x}_2, K_{c_2}, \tau_{I_2}, \bar{P}_2]^\top \tag{2.21}$$

$$\mathbf{d} := [\bar{P}_{100}, \bar{F}_{200}]^\top \tag{2.22}$$

The system of nonlinear first-order ordinary differential equations making up the model include: a balance on composition x_2; a relation balance for pressure P_2; integrated output error equations for I_1 and I_2; and a cumulative integrated squared error equation to determine $\varepsilon_{\text{ise}}(t_f)$.

$$M\dot{x}_2 - (F_1 x_1 - F_2 x_2) = 0 \tag{2.23}$$

$$C\dot{P}_2 - (F_4 - F_5) = 0 \tag{2.24}$$

$$\dot{I}_1 - (x_2 - \bar{x}_2) = 0 \tag{2.25}$$

$$\dot{I}_2 - \left(P_2 - \bar{P}_2\right) = 0 \tag{2.26}$$

$$\dot{\varepsilon}_{\text{ise}} - \left((x_2 - \bar{x}_2)^2 + (P_2 - \bar{P}_2)^2\right) = 0 \tag{2.27}$$

The algebraic equations used to explicitly define the algebraic variables, and subsequently eliminate them from the model, are defined as follows,

$$T_2 := 0.5616\, P_2 + 0.3126\, x_2 + 48.43 \tag{2.28}$$

$$T_3 := 0.507\, P_2 + 55 \tag{2.29}$$

$$T_{100} := 0.1538\, P_{100} + 90 \tag{2.30}$$

$$Q_{100} := UA_1(T_{100} - T_2) \tag{2.31}$$

$$F_{100} := Q_{100}/\lambda_s \tag{2.32}$$

$$F_4 := \frac{Q_{100} + F_1 C_p (T_1 - T_2)}{\lambda + C_p(T_3 - T_2)} \tag{2.33}$$

$$F_2 := F_1 - F_4 \tag{2.34}$$

$$Q_{200} := \frac{2C_p F_{200} U A_2 (T_3 - T_{200})}{2C_p F_{200} + U A_2} \tag{2.35}$$

$$T_{201} := T_{200} + \frac{Q_{200}}{F_{200} C_p} \tag{2.36}$$

$$F_5 := Q_{200}/\lambda \tag{2.37}$$

Additionally, the algebraic controller and disturbance expressions are given as,

$$P_{100} := \bar{P}_{100} + K_{c_1}\left(x_2 - \bar{x}_2\right) + \tfrac{K_{c1}}{\tau_{I_1}} I_1 \tag{2.38}$$

$$F_{200} := \bar{F}_{200} + K_{c_2}\left(P_2 - \bar{P}_2\right) + \tfrac{K_{c2}}{\tau_{I_2}} I_2 \tag{2.39}$$

$$F_1 := \bar{F}_1 + \Delta\bar{F}_1\, \gamma \tag{2.40}$$

$$x_1 := \bar{x}_1 + \Delta\bar{x}_1\, \gamma \tag{2.41}$$

Expressions for the evaporator capital and operating costs were obtained from Douglas [42]. In our calculations, we consider the annualized process capital cost defined based on the

evaporator unit and the overhead condenser as a function of the overall surface areas for each unit, UA_1 and UA_2, respectively. The operating cost is based on the average steam and cooling water flow rates over a one year period.

$$C_{\text{evap}} := 5463\,(UA_1)^{0.65} \tag{2.42}$$

$$C_{\text{cond}} := 2820\,(UA_2)^{0.65} \tag{2.43}$$

$$C_{\text{steam}} := c_s\,\Delta H_{\text{vap}}\,F_{100}(t_f)\,T_A = 4890\,F_{100}(t_f) \tag{2.44}$$

$$C_{\text{cw}} := c_{\text{cw}}\,\Delta H_{\text{cond}}\,F_{200}(t_f)\,T_A = 4.9\,F_{200}(t_f) \tag{2.45}$$

The final capital and operating cost expressions (prior to discretization and using open-loop inputs) are,

$$C_{\text{cap}}(\mathbf{p}) := \tfrac{1}{\beta_p}\,(C_{\text{evap}} + C_{\text{cond}}) \tag{2.46}$$

$$C_{\text{op}}(\mathbf{x}(t_f), \mathbf{u}(t_f), \mathbf{v}(t_f), \mathbf{d}, \mathbf{p}, t_f) := \beta_t\,(C_{\text{steam}} + C_{\text{cw}}) \tag{2.47}$$

where β_p and β_t are the payback period and tax fraction, respectively. We remark that the operating cost expression evaluated at t_f assumes a sustained period of steady-state operation at the final time conditions. The following table defines the model parameters.

Table 2.6: Example 1 – evaporator model parameter values

Description	**Symbol**	**Value**	**Units**
cooling water heat capacity	C_p	0.07	kW min/(K kg)
latent heat of evaporation (water)	λ	38.5	kW min/kg
latent heat of steam (saturated)	λ_s	36.6	kW min/kg
feed temperature	T_1	40	°C
cooling water inlet temperature	T_{200}	25	°C
evaporator liquid holdup	M	20	kg
pressure equation parameter	C	4	kg min/kPa
steam cost parameter	$c_s\,\Delta H_{\text{vap}}$	1.0×10^{-2}	\$/kg
cooling water cost parameter	$c_{\text{cw}}\,\Delta H_{\text{cond}}$	1.0×10^{-5}	\$/kg
payback period	β_p	3	yr
tax rate fraction	β_t	1	–
annual operating time	T_A	8150	hr/yr
initial feed flow rate interval	$\bar{F}_1$	$[9.75, 10.25]$	kg/min
feed flow rate step interval	$\Delta\bar{F}_1$	$[-0.25, 0.25]$	kg/min
initial feed composition interval	$\bar{x}_1$	$[4.75, 5.25]$	%
feed composition step interval	$\Delta\bar{x}_1$	$[-0.25, 0.25]$	%
product composition bounds	x_2	$[25, 30]$	%
operating pressure bounds	P_2	$[40, 80]$	kPa
steam pressure upper bound	P_{100}^U	400	kPa
cooling water flow rate upper bound	F_{200}^U	600	kg/min

2.7 Distillation Model Equations

The binary distillation model was adapted from Schweiger and Floudas [24]. The variables for the distillation model are defined on a stage-wise basis where we label each stage starting from the reboiler at $n = 0$, proceeding with each tray from $n = 1, \ldots, n_t$, and ending with the condenser at $n = n_t + 1$. The variables and parameters are defined as,

$$\mathbf{x}_0(t) := [x_0, L_0]^\top \tag{2.48}$$

$$\mathbf{x}_n(t) := [x_n, L_n]^\top \ \ \forall \ \ n \in \mathcal{N}\backslash\{0, n_t+1\} \tag{2.49}$$

$$\mathbf{x}_{n_t+1}(t) := [x_{n_t+1}, D]^\top \tag{2.50}$$

$$\mathbf{x}(t) := [\mathbf{x}_0(t)^\top, \ldots, \mathbf{x}_{n_t+1}(t)^\top, I_1, I_2, \varepsilon_{\text{ise}}]^\top \tag{2.51}$$

$$\mathbf{z}_n(t) := [M_n^l, y_n]^\top \ \ \forall \ \ n \in \mathcal{N}\backslash\{n_t+1\} \tag{2.52}$$

$$\mathbf{z}_{n_t+1}(t) := M_{n_t+1}^l \tag{2.53}$$

$$\mathbf{z}(t) := [\mathbf{z}_0(t)^\top, \ldots, \mathbf{z}_{n_t+1}(t)^\top]^\top \tag{2.54}$$

$$\mathbf{y}(t) := [x_0, x_{n_t+1}]^\top \tag{2.55}$$

$$\mathbf{u}(t) := [R, V]^\top \tag{2.56}$$

$$\mathbf{v}(t) := [F, z]^\top \tag{2.57}$$

$$\mathbf{p} := [D_{\text{col}}, K_{c_1}, \tau_{I_1}, \bar{x}_0, K_{c_2}, \tau_{I_2}, \bar{x}_{n_t+1}]^\top \tag{2.58}$$

$$\mathbf{d} := [\bar{R}, \bar{V}]^\top \tag{2.59}$$

where we use the index set $\mathcal{N} := \{0, \ldots, n_t+1\}$ to represent all stages in the column. A detailed explanation of the system variables is given in Table 2.7. The system of nonlinear first-order ordinary differential equations making up the model include: a relation balance for the liquid flow down the column for each stage; a composition balance for the light key component for each stage; integrated output error equations for I_1 and I_2; and a cumulative integrated squared error equation to determine $\varepsilon_{\text{ise}}(t_f)$.

$$\tau_0 \, \dot{L}_0 - (L_1 - (L_0 + V)) = 0 \tag{2.60}$$

$$\tau \, \dot{L}_n - (L_{n+1} - L_n + \mathcal{F}_n) = 0 \ \ \forall \ \ n \in \mathcal{N}\backslash\{0, n_t, n_t+1\} \tag{2.61}$$

$$\tau \, \dot{L}_{n_t} - (R - L_{n_t}) = 0 \tag{2.62}$$

$$\tau_{n_t+1} \dot{D} - (V - (R + D)) = 0 \tag{2.63}$$

$$M_0^l \dot{x}_0 - (L_1(x_1 - x_0) + V(x_0 - y_0)) = 0 \tag{2.64}$$

$$M_n^l \dot{x}_n - (L_{n+1}\,(x_{n+1} - x_n) + V\,(y_{n-1} - y_n) + \mathcal{F}\mathcal{Z}\mathcal{X}_n) = 0 \ \ \forall \ \ n \in \mathcal{N}\backslash\{0, n_t, n_t+1\} \tag{2.65}$$

$$M_{n_t}^l \dot{x}_{n_t} - (R\,(x_{n_t+1} - x_{n_t}) + V\,(y_{n_t-1} - y_{n_t})) = 0 \tag{2.66}$$

$$M^l_{n_t+1}\dot{x}_{n_t+1} - V(y_{n_t} - x_{n_t+1}) = 0 \tag{2.67}$$

$$\dot{I}_1 - (x_0 - \bar{x}_0) = 0 \tag{2.68}$$

$$\dot{I}_2 - (x_{n_t+1} - \bar{x}_{n_t+1}) = 0 \tag{2.69}$$

$$\dot{\varepsilon}_{\text{ise}} - \left((x_0 - \bar{x}_0)^2 + (x_{n_t+1} - \bar{x}_{n_t+1})^2\right) = 0 \tag{2.70}$$

The algebraic equations used to explicitly define the algebraic variables are given as follows,

$$y_n := \alpha\, x_n/(1 + x_n(\alpha - 1)) \;\; \forall \;\; n \in \mathcal{N}\backslash\{n_t + 1\} \tag{2.71}$$

$$M^l_n \approx M := \gamma_1\, D^2_{\text{col}} \left(h_{\text{weir}} + \gamma_2/D^{2/3}_{\text{col}}\right) \tag{2.72}$$

$$\tau := \gamma_3\, D^{4/3}_{\text{col}} \tag{2.73}$$

$$\tau_0 = \tau_{n_t+1} := 100\,\tau \tag{2.74}$$

$$M^l_0 = M^l_{n_t+1} := 10\,M \tag{2.75}$$

$$\mathcal{F}_n := \begin{cases} F & \text{if } \; n = n_f \\ 0 & \text{otherwise} \end{cases} \tag{2.76}$$

$$\mathcal{FZX}_n := \begin{cases} F\,(z - x_n) & \text{if } \; n = n_f \\ 0 & \text{otherwise} \end{cases} \tag{2.77}$$

$$D^{\min}_{\text{col}} := \gamma_4\, \bar{V}^{0.5} \tag{2.78}$$

Additionally, the algebraic controller and disturbance expressions are given as,

$$V := \bar{V} + K_{c_1}\,(x_0 - \bar{x}_0) + \tfrac{K_{c_1}}{\tau_{I_1}} I_1 \tag{2.79}$$

$$R := \bar{R} + K_{c_2}\,(x_{n_t+1} - \bar{x}_{n_t+1}) + \tfrac{K_{c_2}}{\tau_{I_2}} I_2 \tag{2.80}$$

$$F := \bar{F} + \Delta\bar{F}\,\gamma \tag{2.81}$$

$$z := \bar{z} + \Delta\bar{z}\,\gamma \tag{2.82}$$

Table 2.7: Example 2 – distillation model variable definitions

x_n: liquid composition	F: liquid feed flow rate (kmol/min)
y_n: vapor composition	R: reflux flow rate (kmol/min)
z: feed composition	D: distillate flow rate (kmol/min)
L_n: liquid flow rate (kmol/min)	V: vapor boilup flow rate (kmol/min)
M_n^l: liquid molar holdup (kmol)	$I_{\{1,2\}}$: integrated controller error
	ε_{ise}: cumulative integrated squared error

Expressions for the distillation column capital and operating costs were again obtained from Douglas [42] and are defined in a simplified form as,

$$C_{\text{tray}} := 95.5\, D_{\text{col}}^{1.55}\, H_{\text{col}} \tag{2.83}$$

$$C_{\text{shell}} := 2928\, D_{\text{col}}^{1.066}\, H_{\text{col}}^{0.802} \tag{2.84}$$

$$C_{\text{utility}} := (c_s\, \Delta H_{\text{vap}} + c_{\text{cw}}\, \Delta H_{\text{cond}})\, \bar{V}\, T_A = 7756\, \bar{V} \tag{2.85}$$

where the column height is defined as $H_{\text{col}} = S_{\text{tray}}\, n_t$. The final capital and operating cost expressions (in \$/year) are,

$$C_{\text{cap}}(\mathbf{p}) := \tfrac{1}{\beta_p}\, (C_{\text{tray}} + C_{\text{shell}}) \tag{2.86}$$

$$C_{\text{op}}(\mathbf{x}(t_f), \mathbf{u}(t_f), \mathbf{v}(t_f), \mathbf{d}, \mathbf{p}, t_f) := \beta_t\, C_{\text{utility}} \tag{2.87}$$

The model parameters used in defining the distillation model are listed in the following table.

Table 2.8: Example 2 – distillation model parameter values

Description	**Symbol**	**Value**	**Units**
number of trays	n_t	16	–
feed tray location	n_f	8	–
relative volatility	α	2.5	–
weir height	h_{weir}	0.0254	m
tray spacing	S_{tray}	0.5080	m
tray holdup parameter	γ_1	6.0305	kmol/m^3
tray holdup parameter	γ_2	0.008929	$\text{m}^{5/3}$
time constant parameter	γ_3	0.05271	$\text{min m}^{-4/3}$
flood constraint parameter	γ_4	0.6719	$\text{m min}^{1/2}\,\text{kmol}^{-1/2}$
steam cost	c_s	4.99×10^{-7}	\$/kJ
cooling water cost	c_{cw}	1.23×10^{-8}	\$/kJ
heat of vaporization	ΔH_{vap}	3.1×10^4	kJ/kmol
heat of condensation	ΔH_{cond}	3.2×10^4	kJ/kmol
initial feed flow rate interval	$\bar{F}$	$[0.5, 1.5]$	kmol/min
feed flow rate step interval	$\Delta\bar{F}$	$[-0.5, 0.5]$	kmol/min
initial feed composition interval	$\bar{z}$	$[0.3, 0.6]$	–
feed composition step interval	$\Delta\bar{z}$	$[-0.1, 0.1]$	–
condenser composition lower bound	$x^L_{n_t+1}$	0.98	–
reboiler composition upper bound	x^U_0	0.05	–

List of References

[1] A. Cervantes and L. T. Biegler. “Large-Scale DAE Optimization using a Simultaneous NLP Formulation”. In: *AIChE Journal* 44.5 (1998), pp. 1038–1050 (cit. on p. 10).

[2] M. C. Colantonio and B. Pytlak. “Dynamic optimization of large scale systems: case study”. In: *International Journal of Control* 72.9 (1999), pp. 833–841 (cit. on p. 10).

[3] M. Fikar, M. A. Latifi, and Y. Creff. “Optimal changeover profiles for an industrial depropanizer”. In: *Chemical Engineering Science* 54.13-14 (1999), pp. 2715–2720 (cit. on p. 10).

[4] L. T. Biegler. *Nonlinear Programming: Concepts, algorithms, and applications to chemical processes*. SIAM, 2010 (cit. on pp. 10, 20, 23).

[5] A. Hartwich et al. “Parallel sensitivity analysis for efficient large-scale dynamic optimization”. In: *Optimization and Engineering* 12.4 (2011), pp. 489–508 (cit. on pp. 10, 23).

[6] D. B. Leineweber et al. “An efficient mulitple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part II: Software aspects and applications”. In: *Computers & Chemical Engineering* 27.2 (2003), pp. 167–174 (cit. on pp. 10, 12, 25).

[7] C. D. Laird, A. V. Wong, and J. Akesson. “Parallel Solution of Large-Scale Dynamic Optimization Problems”. In: *21st European Symposium on Computer Aided Process Engineering ESCAPE 21*. 2011, pp. 2–6 (cit. on pp. 11, 16).

[8] Y. Zhu, S. Legg, and C. D. Laird. “Optimal operation of cyrogenic air separation systems with demand uncertainty and contractual obligations”. In: *Chemical Engineering Science* 66.5 (2011), pp. 953–963 (cit. on p. 11).

[9] L. A. Ricardez-Sandoval. “Optimal design and control of dynamic systems under uncertainty: A probabilistic approach”. In: *Computers & Chemical Engineering* 43.10 (2012), pp. 91–107 (cit. on p. 11).

[10] M. Morari and J. D. Perkins. “Design for operations”. In: *Foundations of Computer-Aided Process Design*. Ed. by L. T. Biegler and M. F. Doherty. Vol. 91. AIChE Symposium Series, No. 304. New York: CACHE AIChE, 1995, pp. 105–114 (cit. on p. 11).

[11] J. M. G. van Schijndel and E. N. Pistikopoulos. “Towards the integration of process design, process control & process operability - Current status and future trends”. In: *Foundations of Computer-Aided Process Design*. Ed. by M. F. Malone, J. A. Trainham, and

B. Carnahan. New York: American Institute of Chemical Engineers, 2000, pp. 99–112 (cit. on p. 12).

[12] V. Sakizlis, J. D. Perkins, and E. N. Pistikopoulos. “Recent advances in optimization-based simultaneous process and control design”. In: *Computers & Chemical Engineering* 28.10 (2004), pp. 2069–2086 (cit. on pp. 12, 18).

[13] Z. Yuan et al. “State-of-the-art and progress in the optimization-based simultaneous design and control for chemical processes”. In: *AIChE Journal* 58.6 (2012), pp. 1640–1659 (cit. on p. 12).

[14] M. J. Mohideen, J. D. Perkins, and E. N. Pistikopoulos. “Optimal design of dynamic systems under uncertainty”. In: *AIChE Journal* 42.8 (1996), pp. 2251–2272 (cit. on pp. 13, 14).

[15] R. Huang and L. T. Biegler. “Robust nonlinear model predictive controller design based on multi-scenario formulation”. In: *Proceedings of the 2009 Conference on American Control Conference*. ACC’09. 2009, pp. 2341–2342 (cit. on p. 13).

[16] A. Shapiro. “Stochastic programming approach to optimization under uncertainty”. In: *Mathematical Programming* 112.1 (2008), pp. 183–220 (cit. on p. 13).

[17] M. G. Ierapetritou, J. Acevedo, and E. N. Pistikopoulos. “An optimization approach for process engineering problems under uncertainty”. In: *Computers & Chemical Engineering* 20.6-7 (1996), pp. 703–709 (cit. on pp. 13, 16).

[18] N. V. Sahinidis and I. E. Grossmann. “Reformulation of multiperiod MILP models for planning and scheduling of chemical processes”. In: *Computers & Chemical Engineering* 15.4 (1991), pp. 255–272 (cit. on p. 13).

[19] D. Ruppen, C. Benthack, and D. Bonvin. “Optimization of batch reactor operation under parametric uncertainty – computational aspects”. In: *Journal of Process Control* 5.4 (1995), pp. 235–240 (cit. on p. 13).

[20] T. K. Bhatia and L. T. Biegler. “Dynamic Optimization for Batch Design and Scheduling with Process Model Uncertainty”. In: *Industrial & Engineering Chemistry Research* 36.9 (1997), pp. 3708–3717 (cit. on p. 13).

[21] J. Gondzio and A. Grothey. “Exploiting structure in parallel implementation of interior point methods for optimization”. In: *Computational Management Science* 6.2 (2009), pp. 135–160 (cit. on p. 16).

[22] A. Shapiro. “Monte Carlo Sampling Methods”. In: *Stochastic Programming*. Ed. by A. Ruszczynski and A. Shapiro. Vol. 10. Handbooks in Operations Research and Management Science. Oxford, UK: Elsevier, 2003, pp. 353–425 (cit. on p. 17).

[23] U. Diwekar. *Introduction to Applied Optimization*. Springer, 2008 (cit. on p. 17).

[24] C. A. Schweiger and C. A. Floudas. “Interaction of design and control: Optimization with dynamic models”. In: *Optimal Control: Theory, Algorithms, and Applications*. Ed. by W. W. Hager and P. M. Pardalos. New York: Kluwer Academic Publishers, 1997, pp. 388–435 (cit. on pp. 18, 27, 51).

[25] H. G. Bock and K. J. Plitt. “A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems”. In: *Ninth IFAC World Congress*. Budapest, 1984 (cit. on p. 19).

[26] H. G. Bock et al. “A direct multiple shooting method for real-time optimization of nonlinear DAE processes”. In: *Nonlinear Model Predictive Control*. Ed. by F. Allgower and A. Zheng. Vol. 26. Progress in Systems and Control Theory. Basel, Switzerland: Birkhauser Verlag, 2000, pp. 245–267 (cit. on p. 19).

[27] D. B. Leineweber et al. “An efficient mulitple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part I: Theoretical aspects”. In: *Computers & Chemical Engineering* 27.2 (2003), pp. 157–166 (cit. on p. 19).

[28] V. S. Vassiliadis, R. W. H. Sargent, and C. C. Pantelides. “Solution of a class of multistage dynamic optimization problems. 2. Problems with path constraints”. In: *Industrial & Engineering Chemical Research* 33.9 (1994), pp. 2123–2133 (cit. on p. 21).

[29] M. Caracotsios and W. E. Stewart. “Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations”. In: *Computers & Chemical Engineering* 9.4 (1985), pp. 359–365 (cit. on p. 23).

[30] Y. Cao et al. “Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution”. In: *SIAM Journal on Scientific Computing* 24.3 (2003), pp. 1076–1089 (cit. on p. 23).

[31] A. Wachter and L. T. Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical Programming* 106.1 (2006), pp. 25–57 (cit. on p. 24).

[32] R. H. Byrd, J. Nocedal, and R. A. Waltz. “Large-Scale Nonlinear Optimization”. In: ed. by G. Pillo and M. Roma. Springer, 2006. Chap. Knitro: An Integrated Package for Nonlinear Optimization, pp. 35–59 (cit. on p. 24).

[33] M. Kiehl. “Parallel multiple shooting for the solution of initial value problems”. In: *Parallel Computing* 20.3 (1994), pp. 275–295 (cit. on pp. 25, 26).

[34] M. Jeon. “Parallel optimal control with multiple shooting, constraints aggregation and adjoint methods”. In: *Journal of Applied Mathematics and Computing* 19.1 (2005), pp. 215–229 (cit. on p. 25).

[35] B. Bachmann et al. “Parallel Multiple-Shooting and Collocation Optimization with OpenModelica”. In: *9th International Modelica Conference*. Munich, Germany, Sept. 2012, pp. 659–668 (cit. on p. 25).

[36] P. E. Gill, W. Murray, and M. A. Saunders. “SNOPT: An SQP algorithm for large-scale constrained optimization”. In: *SIAM Review* 47.1 (2005), pp. 99–131 (cit. on p. 26).

[37] A. C. Hindmarsh et al. “SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers”. In: *ACM Transactions on Mathematical Software* 31.3 (2005), pp. 363–396 (cit. on p. 26).

[38] B. R. Keeping and C. C. Pantelides. “A distributed memory parallel algorithm for the efficient computation of sensitivities of differential-algebraic systems”. In: *Mathematics and Computers in Simulation* 44.6 (1998), pp. 545–558 (cit. on p. 26).

[39] R. B. Newell and P. L. Lee. *Applied Process Control – A Case Study*. New Jersey: Prentice Hall, 1989 (cit. on pp. 27, 48).

[40] I. K. Kookos and J. D. Perkins. “An algorithm for simultaneous process design and control”. In: *Industrial & Engineering Chemistry Research* 40.19 (2001), pp. 4079–4088 (cit. on pp. 27, 48).

[41] P. S. Pacheco. *An Introduction to Parallel Programming*. New York, NY, USA: Morgan Kaufmann, 2011 (cit. on p. 37).

[42] J. M. Douglas. *Conceptual Design of Chemical Processes*. New York: McGraw-Hill, 1988 (cit. on pp. 49, 54).

Chapter 3

A Parallel Implementation for Multiperiod Dynamic Optimization of Large-Scale DAE Systems

This chapter develops a technique for optimizing large-scale differential-algebraic equation process models under uncertainty using a parallel embedded model approach. A combined multiperiod multiple-shooting discretization scheme is used, which creates a significant number of independent numerical integration tasks for each shooting interval over all scenario/period realizations. Each independent integration task is able to be solved in parallel as part of the function evaluations within a gradient-based nonlinear programming solver. This chapter seeks to extend the concepts laid out in the previous chapter and focuses more on demonstrating potential parallel computation performance improvement when applied to large-scale embedded differential-algebraic equations (DAEs) using a more rigorous software framework. We assess our parallel dynamic optimization approach on two examples; the first is a benchmark literature problem, while the second is a large-scale air separation problem that considers a robust set-point transition under parametric uncertainty. Results indicate that focusing on the speedup of the embedded model evaluation can significantly decrease the overall computation time; however, as the multiperiod formulation grows with

increased realizations the computational burden quickly shifts to the internal computation performed within the nonlinear programming algorithm. This highlights the need for further decomposition, structure exploitation and parallelization within the nonlinear programming algorithm and is the subject for further investigation in the next chapter.

Note, portions of this chapter were published according to the journal article:

I.D. Washington and C.L.E. Swartz. "Multiperiod Dynamic Optimization for Large-Scale Differential-Algebraic Process Models under Uncertainty". In: *Processes*, 2015. 3, 541–567.

3.1 Introduction

The optimization of process systems under uncertainty is important and in many cases necessary for capturing realistic solutions to the optimal operation and design of physical systems. Both external system disturbances (e.g., environmental conditions, feed stream availability, product demands) and inherent internal unknowns (e.g., kinetic parameters, physical and transport properties) within the process necessitate considering uncertainty within the optimization model formulation and solution. This has long been realized and many computational approaches specifically from the process systems community have been proposed (see, Geletu and Li [1] for a recent review). In this chapter we are concerned with the solution of dynamic optimization under uncertainty for which, from a design perspective, a number of applications include [2–4]. In general, optimization under uncertainty can be classified depending on how uncertainty is modeled and fall under two categories: stochastic optimization and robust optimization. In the stochastic optimization approach, uncertain parameters are modeled as random variables with a known or imposed probability distribution. Stochastic objective functionals and possibly constraint functionals are often expressed using a probabilistic representation, and the practical solution implementation requires the conversion from an infinite to finite dimensional formulation. This is often

achieved through the use of sampling techniques (to approximate the various probability distributions) followed by a deterministic optimization procedure. Furthermore, to properly quantify the influence of parametric uncertainty on the optimization solution, multiple repeated sampling and optimization solutions are often performed followed by a statistical analysis [5]. A popular stochastic optimization approach that has emerged over the last several decades is chance constraint programming (CCP), in which constraints are relaxed according to a particular probability distribution. A key aspect when using chance constraint optimization is efficiently and accurately approximating multivariate integrals associated with the probabilistic terms, and recent work covering dynamic systems is discussed by Arellano-Garcia and Wozny [6] and Kloppel et al. [7]. The robust optimization approach, on the other hand, requires no a priori knowledge of the uncertain parameters and instead these parameters are assumed to take values from a bounded interval or set. The central idea of this approach is to ensure no constraint violation under all possible realizations within the imposed uncertainty interval. Robust optimization formulations are conveniently posed as min-max problems, where the idea is to minimize the maximum impact of uncertainty on the performance index subject to the largest possible constraint violation (i.e., worst-case analysis). Recent work in this direction is discussed by Diehl et al. [8] and Houska et al. [9] who provide a framework for robust optimal control of dynamic systems. Regardless of the particular uncertainty classification, the conversion of an infinite dimension problem to an implementable finite deterministic nonlinear programming formulation is necessary and one approach to do so is a multiperiod discretization. The approach can serve as a complete solution such as in a robust model predictive control framework [10, 11] or as a component of a more elaborate iterative solution process [2].

In this chapter we are primarily concerned with the use of dynamic process models described by a system of differential-algebraic equations (DAEs) and the efficient incorporation of uncertainty using multiperiod optimization. This approach can be used to fully or partially address stochastic or robust optimization formulations, depending on how one characterizes

uncertainty. Given the widespread use of multiperiod formulations, our approach in this work is to focus on the computational aspects and in particular the use of dynamically constrained formulations and their efficient implementation. We further remark that this chapter is an extension of our previous work described in Chapter 2, and key contributions of this work include: (1) the application of the ideas of Chapter 2 to DAE systems; (2) the implementation of a C/C++ software framework that links several reputable numerical packages; (3) the assessment of second-order sensitivity generation when using higher-order gradient-based nonlinear programming methods; (4) the application to a large-scale air separation system; (5) the exploration of parallel performance with respect to discretization refinement versus embedded model size that was not previously considered. The chapter is laid out by first discussing the particular optimization formulation addressed and relevant literature pertaining to solution algorithms of such formulations. Next, we provide our solution approach to multiperiod problems with embedded DAE functionals, followed by a discussion on our particular implementation. Following this, we give two examples of different scale to illustrate the computational performance of our approach. Finally, some concluding remarks are provided and future work noted.

3.2 Problem Statement

The multiperiod optimization formulation considered in this chapter seeks to extend Problem P.2.2 from Chapter 2 where we specifically tailor our new formulation to problems with embedded semi-explicit index-1 differential-algebraic equation models. Accordingly, we

consider the general multiperiod nonlinear dynamic optimization formulation:

$$
\begin{aligned}
\min_{\mathbf{u}_i(t),\,\mathbf{d}_i\,\forall\, i,\,\mathbf{p}} \quad & \mathcal{J} := \phi_0(\mathbf{p}, t_f) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot \phi_i(\mathbf{x}_i(t_f), \mathbf{z}_i(t_f), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_f) \\
\text{st}: \quad & \dot{\mathbf{x}}_i(t) - \mathbf{f}_d(\mathbf{x}_i(t), \mathbf{z}_i(t), \mathbf{u}_i(t), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t) = \mathbf{0} \\
& \mathbf{f}_a(\mathbf{x}_i(t), \mathbf{z}_i(t), \mathbf{u}_i(t), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t) = \mathbf{0} \\
& \mathbf{x}_i(t_0) - \mathbf{h}_0(\mathbf{u}_i(t_0), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_0) = \mathbf{0} \\
& \mathbf{g}(\mathbf{x}_i(t), \mathbf{z}_i(t), \mathbf{u}_i(t), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t) \leq \mathbf{0} \\
& \mathbf{u}_i(t) \in U = \{\mathbf{u}_i(t) \in \mathbb{R}^{n_u} \,|\, \mathbf{u}^L \leq \mathbf{u}_i(t) \leq \mathbf{u}^U\} \\
& \mathbf{d}_i \in D = \{\mathbf{d}_i \in \mathbb{R}^{n_d} \,|\, \mathbf{d}^L \leq \mathbf{d}_i \leq \mathbf{d}^U\} \\
& \mathbf{p} \in P = \{\mathbf{p} \in \mathbb{R}^{n_p} \,|\, \mathbf{p}^L \leq \mathbf{p} \leq \mathbf{p}^U\} \\
& t \in T = [t_0, t_f] \;\; \forall\, i = 1, \ldots, n_s
\end{aligned}
\tag{P.3.1}
$$

In the above formulation, the differential and algebraic states are represented by $\mathbf{x}_i(t) \in X \subseteq \mathbb{R}^{n_x}$ and $\mathbf{z}_i(t) \in Z \subseteq \mathbb{R}^{n_z}$, respectively; the open-loop continuous control variables are $\mathbf{u}_i(t) \in U \subseteq \mathbb{R}^{n_u}$; the scenario dependent model parameters are $\mathbf{d}_i \in D \subseteq \mathbb{R}^{n_d}$; the uncertain parameters are $\boldsymbol{\theta}_i \in \Gamma \subseteq \mathbb{R}^{n_\theta}$. All of these variables are associated with a particular period/scenario i. The model parameters $\mathbf{p} \in P \subseteq \mathbb{R}^{n_p}$ are defined uniformly over all scenarios, and are often referred to as first stage, scenario independent, or complicating variables in the literature. The objective function comprises two terms: $\phi_0(\mathbf{p}, t_f) : P \times T \mapsto \mathbb{R}$ which represents a scalar scenario independent portion, and $\phi_i(\cdot) : X \times Z \times D \times P \times \Gamma \times T \mapsto \mathbb{R}$ which represents a scalar scenario dependent portion. The embedded dynamic model is comprised of two separate functionals: $\mathbf{f}_d(\cdot) : X \times Z \times U \times D \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_x}$ and $\mathbf{f}_a(\cdot) : X \times Z \times U \times D \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_z}$, which represent the differential and algebraic functions, respectively, of the DAE model in semi-explicit form, assumed to be index-1 such that the Jacobian of $\mathbf{f}_a(\cdot)$ with respect to $\mathbf{z}_i(t)$ is nonsingular. Furthermore, the DAE functionals are assumed to be sufficiently smooth to ensure existence and uniqueness of the solution [12]. Additionally, $\mathbf{g}(\cdot) : X \times Z \times U \times D \times P \times \Gamma \times T \mapsto \mathbb{R}^{n_g}$ are path inequality

constraints. The weight (or probability) associated with each scenario i is represented as $\mathsf{w}_i := 1/n_s$ or more generally as $\mathsf{w}_i \in [0,1]$, where n_s is the total number of scenarios considered. This particular formulation, where the control variables $\mathbf{u}_i(t)$ are associated with each scenario i, allows for recourse to uncertainty and is in the form of a two-stage stochastic program. The parameters $\mathbf{p}$ constitute first-stage decisions, and parameters $\mathbf{d}_i$ and the control inputs $\mathbf{u}_i(t)$ constitute second-stage decisions that can provide compensatory action in response to disturbance and (uncertain) parameter realizations. Alternatively, the control inputs could be assumed scenario independent and if we neglect design parameters the resulting formulation would resemble a robust optimal control problem.

Nonlinear programming solution techniques tailored to multiperiod formulations have received some attention in the literature. Varvarezos, Biegler, and Grossmann [13] proposed a reduced sequential quadratic programming (rSQP) approach (based on an active-set QP subproblem) which decomposes the multiperiod nature through introducing additional linear constraints and scenario dependent parameters which effectively removes the potentially nonlinear complicating scenario independent parameters and forms a new NLP structure which is easier to solve at the QP level. This decomposed rSQP approach was further explored by Bhatia and Biegler [14] who introduced an interior-point solution technique for each QP subproblem. Ultimately, the resulting interior-point rSQP technique showed superior scalability (with respect to scenario realizations) compared to the active-set rSQP approach. A similar interior-point SQP approach has been used on discretized dynamic optimization formulations which result in highly structured NLP formulations (see, [15, 16]). More recently, Zavala, Laird, and Biegler [17] have demonstrated a parallel primal-dual interior-point approach to tackle discretized multiperiod dynamic optimization formulations. This has ultimately led to general interior-point approaches to handle discretized nominal dynamic optimization formulations [18] and structured NLP formulations [19]. All of these previously noted studies have been on structured NLP techniques involving explicit objective and constraint functionals; however, our particular interest in the present chapter is on

solution techniques involving implicit or embedded functionals which require a secondary solution algorithm for evaluation within the NLP constraints. These types of formulations arise in shooting approaches to dynamic optimization and require the solution of an embedded differential-algebraic system in order to fully evaluate the NLP functions. In conjunction with the multiperiod approach to uncertainty, very little has be demonstrated in the literature on shooting-based multiperiod dynamic optimization.

3.3 Proposed Solution Approach

Our proposed solution approach to Problem P.3.1 uses a combined multiperiod multiple-shooting discretization whereby the embedded DAE model integration tasks are solved in parallel. The main contribution of this chapter is the assessment of such an approach when applied to reasonably large differential-algebraic process models for design under uncertainty with recourse and alternatively robust optimal control. The main difference in our proposed approach and corresponding implementation, from other multiple-shooting approaches presented in the literature [20, 21], is that we have incorporated an additional layer of parallelization in terms of the individual scenarios used within the multiperiod approach.

3.3.1 Multiperiod Multiple-Shooting Discretization

The multiperiod multiple-shooting approach discretizes a continuous nonlinear uncertain dynamic optimization formulation to an algebraic nonlinear program (NLP) with an embedded DAE model within the constraints. The technique entails introducing new optimization parameters ($\mathbf{x}_{j,i}$, $\mathbf{z}_{j,i}$) for all periods/scenarios i to represent the differential and algebraic state variable initial conditions at the beginning of each shooting interval j, and new equality constraints to remove the discrepancy or defect between the differential state variable values at the final time from the previous interval and the initial time in the current interval.

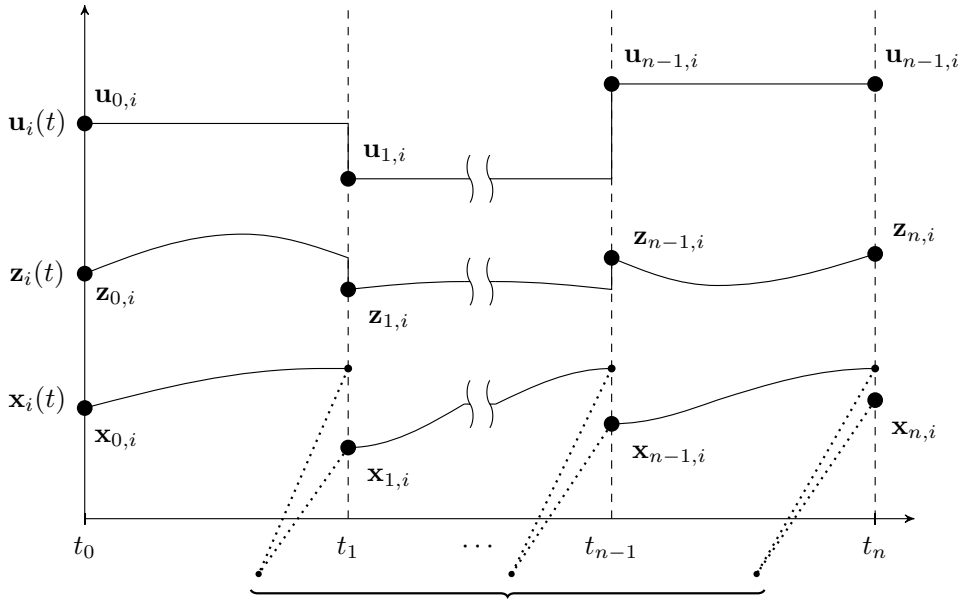

Figure 3.1: Multiperiod multiple-shooting discretization for DAEs

This idea is sketched in Figure 3.1 for scenario realization i, where we assume that the time intervals used in the input control trajectory parameterization (i.e., parameterized by the $\mathbf{u}_{j,i}$) corresponds directly to the defined shooting grid. Our current approach has been to use a rather straight forward implementation of the multiple-shooting technique whereby we provide the discretized formulation in full space to an existing sparse NLP solver, as opposed to exploiting the structure of the formulation by implementing a custom (possibly reduced-space) nonlinear programming technique (cf. the `SCPGEN` code generation solver within `CASADI` [22] or the nonlinear optimal control solver `MUSCOD-II` which uses a tailored reduced SQP algorithm [23]).

A general multiperiod dynamic optimization formulation that utilizes the multiple-shooting discretization applicable to semi-explicit differential-algebraic equation models can be written

as,

$$
\begin{aligned}
\min_{\boldsymbol{w},\mathbf{p}} \;& \mathcal{J} := \phi_0(\mathbf{p}, t_n) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot \phi_i(\mathbf{x}_{n,i}, \mathbf{z}_{n,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_n) \\
\text{st}: \;& \dot{\mathbf{x}}_i(t) = \mathbf{f}_d(\mathbf{x}_i(t), \mathbf{z}_i(t), \boldsymbol{\mathcal{U}}(t, \mathbf{u}_{j,i}), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t) \\
& \mathbf{0} = \mathbf{f}_a(\mathbf{x}_i(t), \mathbf{z}_i(t), \boldsymbol{\mathcal{U}}(t, \mathbf{u}_{j,i}), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t) - \boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t) \\
& \mathbf{0} = \mathbf{h}_0(\boldsymbol{\mathcal{U}}(t_0, \mathbf{u}_{0,i}), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_0) - \mathbf{x}_{0,i} \\
& \mathbf{x}_i(t_{j+1}; \mathbf{x}_{j,i}, \mathbf{z}_{j,i}, \mathbf{u}_{j,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i) - \mathbf{x}_{j+1,i} = \mathbf{0} \\
& \mathbf{f}_a(\mathbf{x}_{k,i}, \mathbf{z}_{k,i}, \boldsymbol{\mathcal{U}}(t_k, \mathbf{u}_{k,i}), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_k) = \mathbf{0} \\
& \mathbf{g}(\mathbf{x}_{k,i}, \mathbf{z}_{k,i}, \boldsymbol{\mathcal{U}}(t_k, \mathbf{u}_{k,i}), \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_k) \leq \mathbf{0} \\
& \forall\, t \in I_{j,i}\ ,\ j = 0, \ldots, n-1 \\
& \forall\, k = 0, \ldots, n\ ,\ \forall\, i = 1, \ldots, n_s \\
& \boldsymbol{w} \in [\boldsymbol{w}^L, \boldsymbol{w}^U],\ \mathbf{p} \in [\mathbf{p}^L, \mathbf{p}^U]
\end{aligned}
\tag{P.3.2}
$$

In the above formulation, $j = 0, \ldots, n-1$ represents n shooting intervals. The optimization parameters are partitioned into scenario-independent global parameters $\mathbf{p}$, and scenario-dependent parameters defined collectively for all shooting intervals and scenario realizations as,

$$
\boldsymbol{w} := [\mathbf{x}_{0,1}^\top, \mathbf{z}_{0,1}^\top, \mathbf{u}_{0,1}^\top, \ldots, \mathbf{x}_{n,n_s}^\top, \mathbf{z}_{n,n_s}^\top, \mathbf{d}_{n_s}^\top]^\top \in \mathbb{R}^{((n_x+n_z)(n+1)+(M+1)n_u n+n_d)n_s} \tag{3.1}
$$

The continuous control input vector can be defined using a parameterized function $\mathbf{u}_i(t) = \boldsymbol{\mathcal{U}}(t, \mathbf{u}_{j,i})$ based on a piecewise approximation within each shooting interval $I_{j,i}$, where $\mathbf{u}_{j,i} \in \mathbb{R}^{(M+1)n_u}$ represent local polynomial coefficients. Note that $\mathbf{u}_{n,i}$ is used in defining the final end point constraint in Problem P.3.2 for notational simplicity, where $\mathbf{u}_{n,i} = \mathbf{u}_{n-1,i}$, and can be removed from the NLP (see Figure 3.1 for a sketch of the parameterization). Additionally, we consider here a fixed end-time formulation where the objective function is represented in Mayer form, which typically only directly depends on the final model states $\mathbf{x}_{n,i}$, $\mathbf{z}_{n,i}$, parameters $\mathbf{d}_i$ and $\mathbf{p}$ and, possibly, the final time t_n. The relaxed DAE model, $\mathbf{F}(\cdot) = \{[\dot{\mathbf{x}}_i(t) - \mathbf{f}_d(\cdot)]^\top, [\mathbf{f}_a(\cdot) - \boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t)]^\top\}^\top$, is embedded within the NLP function

evaluations and is solved using an appropriate DAE solver for $t \in I_{j,i}$, $j = 0, \ldots, n-1$ with initial differential state conditions $\mathbf{x}_i(t_j) = \mathbf{x}_{j,i}$ and algebraic state conditions $\mathbf{z}_i(t_j) = \mathbf{z}_{j,i}$ for all intervals $I_{j,i}$, where the intervals are effectively decoupled using the new parameters and are thus independent of each other. The particular formulation given here relies on the use of a relaxed form of the DAEs using a so-called relaxation function represented here through the function $\boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t)$, where $\boldsymbol{\gamma}_{j,i} := \mathbf{f}_a(\mathbf{x}_{j,i}, \mathbf{z}_{j,i}, \mathbf{u}_{j,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_j)$ is functionally dependent on the shooting parameters at node j. One particular representation of this relaxation function can be given as,

$$\boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t) := \boldsymbol{\gamma}_{j,i} \cdot \alpha(t) \tag{3.2}$$

$$\alpha(t) := \exp\left[-\beta\,(t - t_j)/(t_{j+1} - t_j)\right] \tag{3.3}$$

where $\alpha(t)$ represents a scalar damping factor which is non-increasing and non-negative over the shooting interval $t \in I_{j,i}$ and at each shooting node j takes the value $\alpha(t_j) = 1$. This in turn causes the relaxation function $\boldsymbol{\vartheta}(\cdot)$ to induce consistent algebraic equations at each shooting node ($\boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t_j) = \boldsymbol{\gamma}_{j,i}$), where upon convergence of the NLP, $\boldsymbol{\gamma}_{j,i} = \mathbf{0}$ and the relaxation function vanishes ($\boldsymbol{\vartheta}(\mathbf{0}, t_j) = \mathbf{0}$) such that the original DAE model holds. The symbol β is a scalar tuning parameter set, according to Leineweber et al. [23], as $\beta = 5$. This relaxed DAE form also requires the addition of point equality constraints (for the algebraic model equations) in the NLP at each shooting node to ensure that the original model is obtained upon NLP convergence (see [23, 24] for a more complete treatment of DAE relaxation). It is worth noting that using the relaxed DAE approach avoids the otherwise necessary DAE (re-)initialization problem.

To simplify the formulation and to assist in our presentation, the multiple-shooting continuity

equality constraints, including the initial conditions at t_0, can be defined as,

$$\begin{aligned} \mathbf{c}_{0,i}(\boldsymbol{w}_{0,i}, \mathbf{d}_i, \mathbf{p}) &\equiv \mathbf{h}_0(\mathbf{u}_{0,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i, t_0) - \mathbf{x}_{0,i} = \mathbf{0} \\ \mathbf{c}_{j+1,i}(\boldsymbol{w}_{j,i}, \mathbf{x}_{j+1,i}, \mathbf{d}_i, \mathbf{p}) &\equiv \mathbf{x}_i(t_{j+1}; \mathbf{x}_{j,i}, \mathbf{z}_{j,i}, \mathbf{u}_{j,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i) - \mathbf{x}_{j+1,i} = \mathbf{0} \\ & \forall\, j = 0, \ldots, n-1\ ,\ \forall\, i = 1, \ldots, n_s \end{aligned} \tag{3.4}$$

where $\boldsymbol{w}_{j,i} = [\mathbf{x}_{j,i}^\top, \mathbf{z}_{j,i}^\top, \mathbf{u}_{j,i}^\top]^\top \in \mathbb{R}^{n_x+n_z+(M+1)n_u}$ for $j = 0, \ldots, n-1$, and at the final shooting node, $\boldsymbol{w}_{n,i} = [\mathbf{x}_{n,i}^\top, \mathbf{z}_{n,i}^\top]^\top$. The algebraic consistency equality constraints and remaining inequality constraints represent NLP point constraints and can be defined at each shooting node according to,

$$\begin{aligned} \mathbf{q}_{j,i}(\boldsymbol{w}_{j,i}, \mathbf{d}_i, \mathbf{p}) &\equiv \left[\mathbf{f}_a(\boldsymbol{w}_{j,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i)^\top, \mathbf{g}(\boldsymbol{w}_{j,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i)^\top\right]^\top \\ \mathbf{q}_{n,i}(\mathbf{u}_{n-1,i}, \boldsymbol{w}_{n,i}, \mathbf{d}_i, \mathbf{p}) &\equiv \left[\mathbf{f}_a(\mathbf{u}_{n-1,i}, \boldsymbol{w}_{n,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i)^\top, \mathbf{g}(\mathbf{u}_{n-1,i}, \boldsymbol{w}_{n,i}, \mathbf{d}_i, \mathbf{p}, \boldsymbol{\theta}_i)^\top\right]^\top \\ & \forall\, j = 0, \ldots, n-1\ ,\ \forall\, i = 1, \ldots, n_s \end{aligned} \tag{3.5}$$

The combined constraint vector for each period/scenario can now be stated as,

$$\mathbf{c}_i(\boldsymbol{w}_i, \mathbf{p}) := \begin{bmatrix} \mathbf{c}_{0,i}(\boldsymbol{w}_{0,i}, \mathbf{d}_i, \mathbf{p}) \\ \left(\mathbf{q}_{0,i}(\boldsymbol{w}_{0,i}, \mathbf{d}_i, \mathbf{p})^\top, \mathbf{c}_{1,i}(\boldsymbol{w}_{0,i}, \mathbf{x}_{1,i}, \mathbf{d}_i, \mathbf{p})^\top\right)^\top \\ \vdots \\ \left(\mathbf{q}_{j,i}(\boldsymbol{w}_{j,i}, \mathbf{d}_i, \mathbf{p})^\top, \mathbf{c}_{j+1,i}(\boldsymbol{w}_{j,i}, \mathbf{x}_{j+1,i}, \mathbf{d}_i, \mathbf{p})^\top\right)^\top \\ \vdots \\ \left(\mathbf{q}_{n-1,i}(\boldsymbol{w}_{n-1,i}, \mathbf{d}_i, \mathbf{p})^\top, \mathbf{c}_{n,i}(\boldsymbol{w}_{n-1,i}, \mathbf{x}_{n,i}, \mathbf{d}_i, \mathbf{p})^\top\right)^\top \\ \mathbf{q}_{n,i}(\mathbf{u}_{n-1,i}, \boldsymbol{w}_{n,i}, \mathbf{d}_i, \mathbf{p}) \end{bmatrix},\ \forall\, i = 1, \ldots, n_s \tag{3.6}$$

The fully discretized combined multiperiod multiple-shooting NLP formulation of Problem

P.3.2, can now be stated in parameterized form according to Problem P.3.3.

$$\begin{aligned} \min_{\boldsymbol{w}_i \,\forall\, i,\, \mathbf{p}} \quad & \mathcal{J} := \phi_0(\mathbf{p}) + \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \cdot \phi_i(\boldsymbol{w}_{n,i}, \mathbf{d}_i, \mathbf{p}) \\ \text{st}: \quad & \mathbf{c}_i^L \leq \mathbf{c}_i(\boldsymbol{w}_i, \mathbf{p}) \leq \mathbf{c}_i^U \\ & \boldsymbol{w}_i \in [\boldsymbol{w}^L, \boldsymbol{w}^U] \;\; \forall\, i = 1, \ldots, n_s \\ & \mathbf{p} \in [\mathbf{p}^L, \mathbf{p}^U] \end{aligned} \tag{P.3.3}$$

Depending on the constraint type (equality or inequality), the vectors $\mathbf{c}_i^L$ and $\mathbf{c}_i^U$ are appropriately defined. For each scenario i, the concatenated shooting node parameters are defined as $\boldsymbol{w}_i = [\boldsymbol{w}_{0,i}^\top, \ldots, \boldsymbol{w}_{n,i}^\top, \mathbf{d}_i^\top]^\top \in \mathbb{R}^{(n_x+n_z)(n+1)+(M+1)n_u n+n_d}$. Note that the embedded DAE model $\mathbf{F}(\cdot)$ is removed from the NLP formulation, as it is solved using an embedded DAE solver in order to construct the shooting node continuity constraints.

The multiple-shooting approach benefits from a naturally decoupled temporal domain structure that does not require any additional decomposition techniques. This decoupling of each shooting interval is induced by the introduction of the optimization parameters $\mathbf{x}_{j,i}$, $\mathbf{z}_{j,i}$ for $j = 0, \ldots, n-1$ and $i = 1, \ldots, n_s$ and allows the embedded DAE in each shooting interval and scenario realization to be independently solved in parallel. Accordingly, all scenario realizations i and shooting intervals j over the entire time horizon result in $m = n \cdot n_s$ independent integration tasks, which can be broken up and solved in parallel using several processors.

3.3.2 First-Order Derivative Generation

Shooting-based dynamic optimization approaches necessitate the use of DAE parameter sensitivity in order to generate derivatives of constraints involving implicit functionals. For

example, a relaxed semi-explicit index-1 parameterized DAE system can be stated as,

$$\begin{aligned} \dot{\mathbf{x}}_i(t) - \mathbf{f}_d(\mathbf{x}_i(t), \mathbf{z}_i(t), \mathbf{y}_{j,i}, t) &= \mathbf{0}_{n_x} \\ \mathbf{f}_a(\mathbf{x}_i(t), \mathbf{z}_i(t), \mathbf{y}_{j,i}, t) - \boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t) &= \mathbf{0}_{n_z} \end{aligned} \quad t \in [t_j, t_{j+1}] \tag{3.7}$$

$$\mathbf{x}_i(t_j) = \mathbf{x}_{j,i} \tag{3.8}$$

where all time-invariant parameters within each interval j and scenario i are denoted by $\mathbf{y}_{j,i} = \{\mathbf{x}_{j,i}, \mathbf{z}_{j,i}, \mathbf{u}_{j,i}, \mathbf{d}_i, \mathbf{p}\}$. From this system, the linear first-order forward sensitivity equations can be derived (in matrix form) as,

$$\begin{aligned} \dot{\mathbf{x}}_i^y(t) - \left[\mathbf{f}_d^x(t)\,\mathbf{x}_i^y(t) + \mathbf{f}_d^z(t)\,\mathbf{z}_i^y(t) + \mathbf{f}_d^y(t)\right] &= \mathbf{0}_{n_x \times n_y} \\ \mathbf{f}_a^x(t)\,\mathbf{x}_i^y(t) + \mathbf{f}_a^z(t)\,\mathbf{z}_i^y(t) + \mathbf{f}_a^y(t) - \nabla_y \boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t) &= \mathbf{0}_{n_z \times n_y} \end{aligned} \quad t \in [t_j, t_{j+1}] \tag{3.9}$$

$$\mathbf{x}_i^y(t_j) = [\mathbf{I}_{n_x} | \mathbf{0}_{n_x \times (n_y - n_x)}] \tag{3.10}$$

where $\{\mathbf{x}_i^y(t), \mathbf{z}_i^y(t)\} := \partial\{\mathbf{x}_i(t), \mathbf{z}_i(t)\}/\partial \mathbf{y}_{j,i}$ represents differential and algebraic sensitivity variables, respectively, and $\mathbf{f}_{\{d,a\}}^{\{x,z,y\}}(t) := \partial \mathbf{f}_{\{d,a\}}(\mathbf{x}_i(t), \mathbf{z}_i(t), \mathbf{y}_{j,i}, t)/\partial\{\mathbf{x}_i(t), \mathbf{z}_i(t), \mathbf{y}_{j,i}\}$ are Jacobian matrices of the DAE model with respect to the differential variables, algebraic variables and parameters. This extended linear DAE system is solved forward in time alongside the original system to generate $\mathbf{x}_i^y(t_{j+1})$, which are used to construct the block structured continuity constraint Jacobian (see, Equation 2.9 in Chapter 2). Particularly efficient techniques and software tools to solve the combined systems of Equations 3.7 and 3.9 are discussed by Maly and Petzold [25], Feehery, Tolsma, and Barton [26], Schlegel et al. [27], and Kristensen et al. [28]. In the context of large-scale chemical process engineering models, Hartwich et al. [29] discuss a parallel implementation of the combined DAE and sensitivity system and demonstrate reasonable orders of speedup.

3.3.3 Second-Order Derivative Generation

Sequential quadratic programming (SQP) algorithms (e.g., `filterSQP`) or primal-dual nonlinear interior-point methods (IPM) (e.g., `IPOPT`, `KNITRO`) can often utilize second-order derivatives of the objective/constraint functions, which are used to construct the Lagrangian Hessian (used in the QP subproblem or primal-dual search direction linear solve). To provide such information when using embedded DAE models (i.e., implicit functionals) requires that a second-order sensitivity analysis be performed to construct an approximate representation of the continuity constraint Hessian. All other contributing portions of the Lagrangian Hessian (i.e., explicit objective and point constraint functionals) can be computed exactly using automatic differentiation. A question that arises, and that we seek to address, is whether supplying the second-order derivatives via sensitivity analysis can reduce the number of nonlinear programming algorithm iterations sufficiently to justify the additional computation work of second-order sensitivity analysis.

The complete Lagrangian Hessian for our particular multiperiod formulation can be stated as,

$$\mathbf{H}(\boldsymbol{x},\boldsymbol{\nu}) = \nabla^2_{xx}\mathcal{J}(\boldsymbol{x}) + \sum_{i=1}^{n_s}\sum_{j=0}^{n}\left[\sum_{s=1}^{n_x}\nu^c_{s,j,i}\nabla^2_{xx}\mathrm{c}_{s,j,i}(\boldsymbol{x}) + \sum_{l=1}^{n_q}\nu^q_{l,j,i}\nabla^2_{xx}\mathrm{q}_{l,j,i}(\boldsymbol{x})\right] \quad (3.11)$$

where the italicized symbol $\boldsymbol{x}$ represents a composite vector of all primal NLP variables (as distinct from the model states given by $\mathbf{x}_i(t)$) and $\boldsymbol{\nu} = \{\nu^c_{s,j,i}, \nu^q_{l,j,i}\}$ is a similar concatenation of all dual variables. More specifically, $\nu^c_{s,j,i}$ are equality constraint multipliers related to the continuity constraints in Equation 3.4, represented individually here by $c_{s,j,i}(\boldsymbol{x})$, and $\nu^q_{l,j,i}$ are either equality or inequality constraint multipliers related to the point constraints in Equation 3.5, which are again defined individually as $q_{l,j,i}(\boldsymbol{x})$. In order to compute the individual Hessian portions related to the continuity constraints, $\nu^c_{s,j,i}\nabla^2_{xx}\mathrm{c}_{s,j,i}(\boldsymbol{x})$, a direct second-order sensitivity analysis can be performed on the portion of the Lagrangian involving the embedded functionals. For example, consider the continuity constraint Lagrangian

portion as,

$$\mathcal{L}_c(\boldsymbol{x}, \boldsymbol{\nu}^c) = \sum_{i=1}^{n_s} \boldsymbol{\nu}_{0,i}^{c\top} \mathbf{c}_{0,i}(\mathbf{y}_{0,i}) + \sum_{i=1}^{n_s} \sum_{j=0}^{n-1} \boldsymbol{\nu}_{j+1,i}^{c\top}\, \mathbf{c}_{j+1,i}(\mathbf{x}_i(t_{j+1}; \mathbf{y}_{j,i}), \mathbf{y}_{j+1,i}) \tag{3.12}$$

where the second portion of this term can be taken as a scalar point-wise implicit functional for each scenario and shooting interval and defined accordingly as,

$$g_{j+1,i}(\mathbf{y}_{j,i}) = \boldsymbol{\nu}_{j+1,i}^{c\top}\, \mathbf{c}_{j+1,i}(\mathbf{x}_i(t_{j+1}; \mathbf{y}_{j,i}), \mathbf{y}_{j+1,i}) \tag{3.13}$$

Using the sensitivity generation approach described by Ozyurt and Barton [30], the directional Hessian of this point-wise functional can be determined using a forward-over-adjoint direct second-order sensitivity analysis. The particular purpose here is to investigate the application of this technique in the context of a multiperiod multiple-shooting algorithm. Furthermore, the application considered for our analysis is in the form an ODE; thus, we restrict our presentation of second-order sensitivity analysis to the purely ODE case. Accordingly, we consider the ODE system given by,

$$\dot{\mathbf{x}}_i(t) = \mathbf{f}(\mathbf{x}_i(t), \mathbf{y}_{j,i}, t) \quad t \in [t_j, t_{j+1}] \tag{3.14}$$

$$\mathbf{x}_i(t_j) = \mathbf{x}_{j,i} \tag{3.15}$$

For the more general semi-explicit index-1 DAE case, we refer readers to the work by Cao et al. [31] for first-order methods, Hannemann-Tamas [32] for higher order methods and Albersmeyer and Diehl [33] for higher order relaxed DAE methods. The final form of the directional Hessian of a point-wise functional, in the context of our multiperiod approach, can be stated as,

$$\begin{aligned} \frac{\partial^2 g_{j+1,i}}{\partial \mathbf{y}_{j,i}^2}\mathbf{u} = (\boldsymbol{\lambda}_i(t_j)^\top \otimes \mathbf{I}_{n_y})\, \mathbf{x}_i^{yy}(t_j)\mathbf{u} + \mathbf{x}_i^{y}(t_j)^\top\, \boldsymbol{\mu}_i(t_j) + \\ g_{yy}(t_{j+1})\mathbf{u} + g_{yx}(t_{j+1})\, \mathbf{s}_i(t_{j+1}) - \mathbf{q}_i(t_j) \end{aligned} \tag{3.16}$$

where $\otimes$ represents the Kronecker product; $\boldsymbol{\lambda}_i(t_j) \in \mathbb{R}^{n_x}$ is a vector of first-order adjoint variables at t_j for scenario i; $\boldsymbol{\mu}_i(t_j) \equiv \boldsymbol{\lambda}_i^y(t_j)\mathbf{u} \in \mathbb{R}^{n_x}$ is a vector of directional second-order adjoint variables at t_j; $\mathbf{s}_i(t_{j+1}) \equiv \mathbf{x}_i^y(t_{j+1})\mathbf{u} \in \mathbb{R}^{n_x}$ is a vector of directional first-order forward sensitivity variables at t_{j+1} (i.e., the solution of directional first-order forward sensitivity equations); $\mathbf{q}_i(t_j) \in \mathbb{R}^{n_y}$ is a vector of directional second-order adjoint quadrature variables; $\mathbf{x}_i^{yy}(t_j)\mathbf{u} \equiv \mathbf{0}_{n_x n_y}$ is a vector of directional second-order forward sensitivity variables initially known at t_j, while $\mathbf{x}_i^y(t_j) \equiv [\mathbf{I}_{n_x}|\mathbf{0}_{n_x\times(n_y-n_x)}]$ is a matrix of first-order forward sensitivity variables initially known at t_j; $g_{yy}(t_{j+1}) \equiv \mathbf{0}_{n_y\times n_y}$ and $g_{yx}(t_{j+1}) \equiv \mathbf{0}_{n_y\times n_x}$ are second-order derivatives of the scalar functional $g_{j+1,i}$ evaluated directly at t_{j+1}, which for the multiple-shooting continuity constraints, are simply matrices of zeros. In order to determine the first-order and directional second-order adjoint variables, one needs to first solve forward from t_j to t_{j+1} the first-order forward sensitivity equations to compute the directional sensitivities $\mathbf{s}_i(t_{j+1})$ and then solve backward from t_{j+1} to t_j for each direction $\mathbf{u} \equiv \mathbf{e}_l$, $l = 1, \ldots, n_y$, the combined first- and second-order directional adjoint system given by,

$$\begin{aligned}
\dot{\boldsymbol{\lambda}}_i(t) &= -\mathbf{f}_x(t)^\top \boldsymbol{\lambda}_i(t) \\
\dot{\boldsymbol{\mu}}_i(t) &= -\mathbf{f}_x(t)^\top \boldsymbol{\mu}_i(t) - \left(\boldsymbol{\lambda}_i(t)^\top \otimes \mathbf{I}_{n_x}\right)\left(\mathbf{f}_{xy}(t)\mathbf{u} + \mathbf{f}_{xx}(t)\,\mathbf{s}_i(t)\right) \\
\boldsymbol{\lambda}_i(t_{j+1}) &= g_x(t_{j+1}) \equiv \boldsymbol{\nu}_{j+1,i}^c \\
\boldsymbol{\mu}_i(t_{j+1}) &= g_{xy}(t_{j+1})\mathbf{u} + g_{xx}(t_{j+1})\,\mathbf{s}_i(t_{j+1}) \equiv \mathbf{0}_{n_x}
\end{aligned} \tag{3.17}$$

Additionally, alongside the adjoint system, the quadrature variable vector $\mathbf{q}_i(t_j)$ can be determined from the system,

$$\dot{\mathbf{q}}_i(t) = \mathbf{f}_y(t)^\top \boldsymbol{\mu}_i(t) + \left(\boldsymbol{\lambda}_i(t)^\top \otimes \mathbf{I}_{n_y}\right)\left(\mathbf{f}_{yy}(t)\mathbf{u} + \mathbf{f}_{yx}(t)\,\mathbf{s}_i(t)\right), \quad \mathbf{q}_i(t_{j+1}) = \mathbf{0}_{n_y} \tag{3.18}$$

where $\mathbf{f}_x(t) := \partial\mathbf{f}(\mathbf{x}_i, \mathbf{y}_{j,i}, t)/\partial\mathbf{x}_i(t) \in \mathbb{R}^{n_x\times n_x}$ and $\mathbf{f}_{xy}(t) := \partial^2\mathbf{f}(\mathbf{x}_i, \mathbf{y}_{j,i}, t)/\partial\mathbf{x}_i(t)\partial\mathbf{y}_{j,i} \in \mathbb{R}^{n_x n_x\times n_y}$ in which this latter term is in the form of a stacked Hessian to avoid the otherwise tensor form. Similarly, $\mathbf{f}_{xx}(t) := \partial^2\mathbf{f}(\mathbf{x}_i, \mathbf{y}_{j,i}, t)/\partial\mathbf{x}_i(t)^2 \in \mathbb{R}^{n_x n_x\times n_x}$, with all other derivatives defined in an analogous manner. The evaluation of both adjoint and quadrature systems

requires the efficient evaluation of several matrix-vector products, comprised of first and second derivative terms, using an appropriate automatic differentiation (AD) tool. Once the adjoint and quadrature variables are determined at t_j, the directional Hessian given by Equation 3.16 can be formed. This process is repeated for all $j = 0, \ldots, n-1$ and $i = 1, \ldots, n_s$, and the Lagrangian Hessian $\nabla^2_{xx}\mathcal{L}_c(\boldsymbol{x}, \boldsymbol{\nu}^c)$ is assembled (based on each direction that corresponds to a particular parameter) and further combined with the objective and point constraint Hessian.

3.3.4 Implementation Details

The results in this chapter were generated using a `C/C++` implementation which acts to coordinate the user model input, multiperiod multiple-shooting discretization, and interaction between several available DAE integration and NLP optimization routines. The implementation utilizes several `CSPARSE` routines [34] and interfaces the integration routines `CVODES` and `IDAS` from the SUNDIALS suite of solvers [35], the NLP solvers `SNOPT` [36] and `IPOPT` [37], and the AD tool `ADOLC` [38]. The particular implementation aspect we investigate in this chapter is an `OpenMP` loop parallelization of the high-level DAE integration tasks and we sketch the approximate solution steps according to Algorithms 1 and 2. Note that the sensitivity generation approach employed by the SUNDIALS integrators follows a so-called "first-differentiate-then-discretize" methodology (i.e., the first- and second-order sensitivity equations are formed prior to applying the discretized numerical integration routine); an alternative approach is a so-called "first-discretize-then-differentiate" technique, whereby certain aspects of the internally discretized integration algorithm are differentiated either via AD or through numerical differences during the integration procedure. This latter approach is generally known as internal numerical differentiation (IND) and has shown greater solution accuracies and speeds when used in conjunction with embedded model shooting-based dynamic optimization schemes [39–41]. Despite the merits of this newer approach, for ease of availability through existing solvers we have followed the first approach in this study.

Algorithm 1 sketches a high-level SQP-type solution procedure for the NLP given by Problem P.3.3. The purpose of outlining the NLP solution approach is to provide some insight to where specifically within the algorithm an embedded DAE solver (and possibly a first- and second-order sensitivity solution) is required. Alternatively, one could utilize a nonlinear interior-point approach where the major differences from Algorithm 1 are an adaptive barrier update strategy that nests a procedure similar to Steps 11 to 18 where the Newton search direction is determined from a single solution of the primal-dual equations as opposed to Step 13 shown here, which solves the QP to optimality (see [42] for the details). For the particular algorithm shown, initially, a complete specification of the scenario realization set, initial primal (and possibly dual) variables and termination tolerances is provided. Note that an initial simulation of the embedded DAE can be used to determine initial feasible guesses for the primal shooting variables $\mathbf{x}_{j,i}$ and $\mathbf{z}_{j,i}$ for all j and i, given $\mathbf{u}_{j,i}$, $\mathbf{d}_i$ and $\mathbf{p}$. The dual variables can be initialized at zero (cold start) or warm started if a previous similar NLP solution is available. Following this, an iterative quadratic programming procedure is performed whereby: (1) objective, constraint and derivative functions are evaluated; (2) termination criteria are checked (and possibly termination signaled); (3) a search direction is determined from a quadratic program (using either an active-set or primal-dual interior-point method) (this step is often preceded by a dimensionality reduction of the original QP; additionally, infeasible QP's are handled in a so-called feasibility restoration phase); (4) a globalization procedure is performed to determine the step size (we note a line search, but a trust-region approach within the QP itself is possible); and (5) primal and dual variables are updated and objective, constraint and derivative functions are re-evaluated. The time-dominant aspect of the algorithm occurs with the embedded implicit function evaluations denoted by DAE_SOLVE, which we handle specifically within our implementation by Algorithm 2. Additionally, note that during the step size globalization procedure, re-evaluation of objective and constraint functions is required, and for the multiple-shooting continuity constraints, sensitivity generation is deactivated within DAE_SOLVE. We remark that an algorithm for the procedure DSOA_SOLVE follows in a similar manner to Algorithm

2, whereby a second-order forward-over-adjoint sensitivity analysis is performed for each shooting interval j and scenario i, which is specifically handled by the SUNDIALS integration solvers.

Algorithm 1 Multiperiod gradient-based nonlinear program (NLP) solution approach with embedded differential-algebraic equations (DAE). QP, quadratic programming.

Input: initial primal and dual variable guesses and tolerances

generate scenario realizations: $\boldsymbol{\theta} := \boldsymbol{\theta}_i \in [\boldsymbol{\theta}^L, \boldsymbol{\theta}^U] \; \forall \; i = 1, \ldots, n_s$

define initial guesses for primal, $\boldsymbol{x}^{[0]} := \{\{\mathbf{x}_{j,i}^{[0]}, \mathbf{z}_{j,i}^{[0]}, \mathbf{u}_{j,i}^{[0]}\}_{j,i}, \; \mathbf{d}_i^{[0]}, \; \mathbf{p}^{[0]}\}$, and dual variables $\boldsymbol{\nu}^{[0]}$

provide optimality ($\texttt{tol}_{\text{kkt}}$) and feasibility ($\texttt{tol}_{\text{feas}}$) tolerances

Output: primal/dual solution $\boldsymbol{x}^*$, $\boldsymbol{\nu}^*$ to a local minimum of the NLP satisfying tolerances

procedure $\{\boldsymbol{x}^*, \boldsymbol{\nu}^*\} \leftarrow$ NLP_SOLVE($\boldsymbol{x}^{[0]}$, $\boldsymbol{\nu}^{[0]}$, $\texttt{tol}_{\{\text{kkt,feas}\}}$)

    $k \leftarrow 0$

    initial `eval` of objective/constraints and 1st derivatives (gradient, Jacobian)

    $\mathcal{J}(\boldsymbol{x}^{[0]})$, $\nabla_{\boldsymbol{x}} \mathcal{J}(\boldsymbol{x}^{[0]})$ ▷ explicit function `eval` $\forall \; j, i$

    $\{\mathbf{c}_{j,i}(\boldsymbol{x}^{[0]}), \; \nabla_{\boldsymbol{x}} \mathbf{c}_{j,i}(\boldsymbol{x}^{[0]})\}_{j,i} \leftarrow$ DAE_SOLVE($\boldsymbol{x}^{[0]}, \boldsymbol{\theta}$) ▷ implicit function `eval` $\forall \; j, i$

    $\{\mathbf{q}_{j,i}(\boldsymbol{x}^{[0]}), \; \nabla_{\boldsymbol{x}} \mathbf{q}_{j,i}(\boldsymbol{x}^{[0]})\}_{j,i}$ ▷ explicit function `eval` $\forall \; j, i$

    initial Lagrangian Hessian approximation (or `eval` exactly via DSOA_SOLVE($\boldsymbol{x}^{[0]}, \boldsymbol{\nu}_c^{[0]}, \boldsymbol{\theta}$))

    repeat until termination criteria satisfied

        check KKT conditions (and other termination criteria)

        compute search direction of primal/dual variables ($\mathbf{d}_{\boldsymbol{x}}^{[k]}$, $\mathbf{d}_{\boldsymbol{\nu}}^{[k]}$) via QP solver

        compute step size $\alpha^{[k]}$ via a line search (requires objective/constraint `eval`)

        perform step:

$$\boldsymbol{x}^{[k]} \leftarrow \boldsymbol{x}^{[k]} + \alpha^{[k]} \mathbf{d}_{\boldsymbol{x}}^{[k]}$$

$$\boldsymbol{\nu}^{[k]} \leftarrow \boldsymbol{\nu}^{[k]} + \alpha^{[k]} \mathbf{d}_{\boldsymbol{\nu}}^{[k]}$$

        $k \leftarrow k + 1$

        re-evaluate function derivatives $\forall \; j, i$ (used to construct the next QP)

        $\{\nabla_{\boldsymbol{x}} \mathbf{c}_{j,i}(\boldsymbol{x}^{[k]})\}_{j,i} \leftarrow$ DAE_SOLVE($\boldsymbol{x}^{[k]}, \boldsymbol{\theta}$)

        $\{\nabla_{\boldsymbol{x}} \mathbf{q}_{j,i}(\boldsymbol{x}^{[k]})\}_{j,i}$

        update Hessian approximately (or `eval` exactly via DSOA_SOLVE($\boldsymbol{x}^{[k]}, \boldsymbol{\nu}_c^{[k]}, \boldsymbol{\theta}$))

    end

end procedure

Algorithm 2 Parallel multiperiod DAE and first-order sensitivity function evaluation.

Input: state initial conditions, control parameters and invariant model parameters

1: specified scenario realizations $\boldsymbol{\theta} := \{\boldsymbol{\theta}_i\}_{i=1}^{n_s}$

2: NLP variables $\boldsymbol{x} := \{\{\mathbf{x}_{j,i}, \mathbf{z}_{j,i}, \mathbf{u}_{j,i}\}_{j,i}, \{\mathbf{d}_i\}_i, \mathbf{p}\}$

3: provide relative ($\texttt{tol}_{\text{rel}}$) and absolute ($\texttt{tol}_{\text{abs}}$) integration tolerances for DAE solution

Output: differential state solution $\mathbf{x}_i(t_{j+1})$, $\mathbf{x}_i^y(t_{j+1}) := \partial \mathbf{x}_i(t_{j+1}) / \partial \{\boldsymbol{w}_i, \mathbf{d}_i, \mathbf{p}\}\ \forall\, i, j$

4: **procedure** $\{\mathbf{x}_i(t_{j+1}), \mathbf{x}_i^y(t_{j+1})\} \leftarrow$ DAE_SOLVE($\boldsymbol{x}$, $\boldsymbol{\theta}$, $\texttt{tol}_{\{\text{rel,abs}\}}$) $\quad \forall\, i, j$

5: **for** $i := 1$ **to** n_s **do** ▷ in parallel using `OpenMP` for $k = 1, \ldots, n \cdot n_s$ tasks

6: **for** $j := 0$ **to** $n - 1$ **do**

7: set initial differential and algebraic DAE variables:

$$\mathbf{x}_i(t_j) \leftarrow \mathbf{x}_{j,i}, \quad \dot{\mathbf{x}}_i(t_j) \leftarrow \text{ODERHS\_EVAL}$$

$$\mathbf{z}_i(t_j) \leftarrow \mathbf{z}_{j,i}, \quad \dot{\mathbf{z}}_i(t_j) \leftarrow \mathbf{0}_{n_z}$$

8: set initial differential DAE sensitivity variables:

$$\mathbf{x}_i^y(t_j) \leftarrow [\mathbf{I}_{n_x} | \mathbf{0}_{n_x \times (n_y - n_x)}]$$

$$\dot{\mathbf{z}}_i^y(t_j) \leftarrow \mathbf{0}_{n_z \times n_y}$$

9: compute initial differential and algebraic DAE sensitivity variables:

$$\{\dot{\mathbf{x}}_i^y(t_j),\, \mathbf{z}_i^y(t_j)\} \leftarrow \text{LINEAR\_SYSTEM\_SOLVE}$$

10: solve DAE and 1st order sensitivity system

$$\{\mathbf{x}_i(t_{j+1}),\, \mathbf{x}_i^y(t_{j+1})\} \leftarrow \text{SUNDIALS\_DAE\_SOLVER}$$

11: **end for**

12: **end for**

13: **end procedure**

Algorithm 2 computes the solution of the discretized relaxed embedded DAE (both state and first-order sensitivity variables over each interval and scenario), which is used to evaluate the continuity constraints and associated Jacobian at each major iteration of the NLP algorithm. Note that using the relaxed DAE approach, the initial conditions of the differential and algebraic states are by formulation always consistent at t_j. For each shooting node and scenario realization, the embedded DAE is initialized in Step 7 of Algorithm 2 by the NLP parameters; next, in Step 8, the initial values of the differential sensitivity variables (in matrix form) are set to an augmented identity matrix. The next step is to solve the DAE and first-

order sensitivity system using an appropriate DAE solver with efficient methods for handling the sensitivity equation solution. The last step (not shown) is to construct the continuity equations and Jacobian, the latter of which is a matrix with appropriately positioned blocks of sensitivity variables at t_{j+1}. Several remarks are warranted with respect to Algorithm 2: (1) when using an implicit integration routine, such as `IDAS` from SUNDIALS, one needs to additionally provide initial values for the time-derivatives of the differential states $\dot{\mathbf{x}}_i(t_j)$, the initial time-derivatives of the differential sensitivity variables $\dot{\mathbf{x}}_i^y(t_j)$ and the initial algebraic sensitivity variables $\mathbf{z}_i^y(t_j)$; (2) the differential time-derivatives are determined from a single evaluation of the ODE portion of the DAE; and (3) the differential time-derivative sensitivity variables and algebraic sensitivity variables are computed from a linear solve of the initial sensitivity equations (note that this requires a single initial factorization and several back solves using multiple right-hand-side vectors for a linear system given by $\mathbf{A}\,\mathbf{X} = \mathbf{B}$). We elaborate on this last point. For example, given that variables $\mathbf{x}_i(t_j)$, $\mathbf{z}_i(t_j)$, and $\mathbf{x}_i^y(t_j)$ are known at t_j, we can rearrange the initial first-order sensitivity equation system as,

$$\begin{bmatrix} \mathbf{I}_{n_x} & -\mathbf{f}_d^z(t_j) \\ \mathbf{0}_{n_z \times n_x} & \mathbf{f}_a^z(t_j) \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}_i^y(t_j) \\ \mathbf{z}_i^y(t_j) \end{bmatrix} = \begin{bmatrix} \mathbf{f}_d^x(t_j)\,\mathbf{x}_i^y(t_j) + \mathbf{f}_d^y(t_j) \\ -\mathbf{f}_a^x(t_j)\,\mathbf{x}_i^y(t_j) - \mathbf{f}_a^y(t_j) + \nabla_y \boldsymbol{\vartheta}(\boldsymbol{\gamma}_{j,i}, t_j) \end{bmatrix} \tag{3.19}$$

and solve for the initial values of $\dot{\mathbf{x}}_i^y(t_j)$ and $\mathbf{z}_i^y(t_j)$ (in matrix form).

The particular details of the underlying ODE/DAE or NLP solution used in this chapter can be found in the previously noted references. However, we remark on the following: both first- and second-order sensitivity analysis is handled directly by the SUNDIALS integrators, and all the user needs is an appropriate AD tool to form Equations 3.9 and 3.16 to 3.18; for our implementation, we used the tape-based features of `ADOLC` for all serial computation and the tapeless forward differentiation features when evaluation is needed within parallel `OpenMP` regions of the code (we refer readers to the `ADOLC` manual for the particular details of tapeless and tape-based operator overloaded AD). We further note that all code and third party libraries used for our example problems were compiled using `gcc-4.7` (with

`openmp-3.1`) and run using 64-bit Linux. The hardware was an HP Proliant computing server configured with four sockets using AMD Opteron 6386SE series chips (16 cores per chip) at 2.8 GHz, which provide a total of 64 available cores (processors/threads). Furthermore, an appropriate amount of memory was allocated/utilized to suit the requirements of the program.

3.4 Example Problems

We demonstrate our proposed parallel multiperiod dynamic optimization approach using a batch reactor problem and a large-scale air separation problem. The objective is to assess the computational performance (resource utilization efficiency and scalability) of the proposed method when the number of scenarios and shooting intervals (embedded integration tasks), model size and available computing processors are increased.

3.4.1 Example 1: Batch Reactor Problem

The initial portion of this example is performed with a parallel implementation using the ODE integration solver `CVODES` and NLP solver `SNOPT`. Subsequently, further comparison is made using second-order sensitivity analysis for generating the Lagrangian Hessian when using the NLP solver `IPOPT` with the exact Hessian versus an approximate limited memory quasi-Newton update, which only requires first-order sensitivity information. For this last portion, we currently only report serial solution times of the implementation.

The example considered is adapted from [14] and involves a batch reactor problem in a purely ODE form that follows a first order competing reaction scheme $\mathsf{A} \rightarrow \mathsf{B}$ and $\mathsf{A} \rightarrow \mathsf{C}$, where B is the desired product and the kinetic parameters are assumed to be uncertain. Similar batch reactor models can be found in [43] and in Example 8.4 from [44]. The objective is to operate the reactor for an indeterminate duration (i.e., design variable),

such that a maximum profit is achieved. The objective function comprises a revenue term proportional to the product conversion x_{B} and an operating cost dependent on the duration of operation t_f. The optimization problem is defined according to formulation E.3.1,

$$\begin{aligned}
\min_{t_f, u_i(\tau)\,\forall\, i} \quad & \mathcal{J} := c_1\, {t_f}^{c_2} - \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i\, c_0\, x_{\mathsf{B},i}(1) \\
\text{st}: \quad & \dot{x}_{\mathsf{A},i}(\tau) = -[\theta_{1,i}\, u_i(\tau)^{\theta_{2,i}} + u_i(\tau)]\, x_{\mathsf{A},i}(\tau)\, t_f \\
& \dot{x}_{\mathsf{B},i}(\tau) = \theta_{1,i}\, u_i(\tau)\, x_{\mathsf{A},i}(\tau)\, t_f \\
& x_{\{\mathsf{A},\,\mathsf{B}\},i}(0) = x_{\{\mathsf{A0},\,\mathsf{B0}\}} \\
& x_{\{\mathsf{A},\,\mathsf{B}\},i}(\tau) \in [0,1] \\
& u_i(\tau) \in [0,5]\,, t_f \in [0.5, 1.25]\,, \tau \in [0,1],\ \forall\, i = 1, \ldots, n_s \\
& \theta_1 \in (0.45, 0.55),\ \theta_2 \in (2.15, 2.25)
\end{aligned} \tag{E.3.1}$$

The initial state conditions are taken as $x_{\mathsf{A0}} = 1$, $x_{\mathsf{B0}} = 0\ \forall\, i$, where we use a normalized time horizon such that the end-time t_f is taken as a design parameter. The cost/objective coefficients are set as $c_0 = 700$, $c_1 = 50$, $c_2 = 2$; and the weights are set as $\mathsf{w}_i = 1/n_s$. The parameterized control profile is taken to be piecewise constant and initial guesses for the polynomial coefficients are set to 1.0 for all scenarios and shooting intervals. For this example we kept the number of shooting intervals constant at $n = 25$ and with a uniform size over the time horizon. The uncertain model parameters (θ_1, θ_2) were determined by sampling uniformly between the defined bounds.

Figure 3.2 depicts a base line solution to formulation E.3.1 using a single processor with increasing scenario realizations. For the input and state solution trajectories in Figure 3.2 (a), the solid input and state trajectory lines represent the nominal solution, while the shaded bands represent an envelope of possible solutions generated via discrete realizations of the uncertain parameter values. Interesting aspects to note include: (1) as the number of scenarios is increased, both the optimal objective value (defined here as the ratio of the multiperiod objective value to the nominal objective value, $\mathcal{J}/\bar{\mathcal{J}}$) and parametric degree of

freedom, t_f, converge to a point (or rather confidence interval) which can be considered close to the true solution of the original infinitely dimensional stochastic program; (2) considering $n_s = 40$ as the base line, we see a $\times 2.26$, $\times 4.20$, $\times 8.81$ increase in total computation time per major SQP iteration for $n_s = \{80, 160, 320\}$, respectively (i.e., almost a linear increase in computation time as scenario realizations are added). Based on Figure 3.2 (a) an appropriate number of scenarios to use would exceed 80, where the profiles for $\mathcal{J}/\bar{\mathcal{J}}$ and t_f level off.

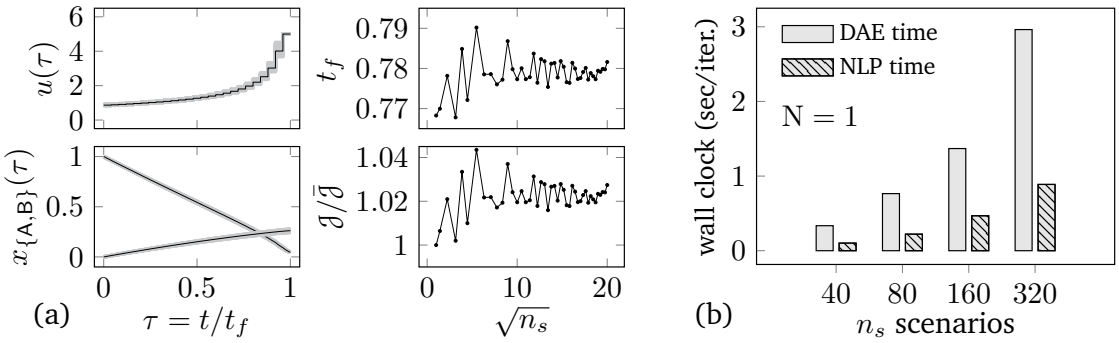

Figure 3.2: Example 1 – (a) control input & state trajectories (nominal solution represented by the solid line) and (b) base line DAE & NLP solution times for increasing n_s

Table 3.1: Example 1 – parallel computation results comparing increasing n_s

							Total program solution time (sec)				
n_s	$m^\star$	#vars	#cons	#iter†	$\mathcal{J}/100^\ddagger$	t_f	N = 1	N = 4	N = 8	N = 16	N = 32
1	25	78	53	24/60	−1.4911	0.7683	0.389	0.251	0.235	0.201	–
40	1000	3081	2081	40/1239	−1.5235	0.7785	17.48	8.87	7.01	6.26	6.24
80	2000	6161	4161	46/2244	−1.5463	0.7868	45.41	21.61	16.88	15.20	14.87
160	4000	12321	8321	49/4395	−1.5340	0.7823	90.00	42.37	34.74	30.87	30.35
320	8000	24641	16641	49/8718	−1.5200	0.7772	188.69	82.60	65.10	56.74	55.67

* $m = n \cdot n_s$, total no. integration tasks, where $n = 25$; † SNOPT solver w/ major/minor iterations;

‡ NLP optimality/feasibility tol. $= \{1 \times 10^{-6}, 1 \times 10^{-8}\}$, DAE relative/absolute tol. $= \{1 \times 10^{-6}, 1 \times 10^{-8}\}$

Parallel solution times for the total program are reported in Table 3.1 for $n_s = \{40, 80, 160, 320\}$

(number of scenario realizations) and $\mathrm{N} = \{4, 8, 16, 32\}$ (number of processors/threads). Additionally, the serial solution time is reported for each scenario realization level and the time required for the nominal dynamic optimization solution (i.e., $n_s = 1$). We further remark that the parallel solution times are an average of several independent experiments; the NLP problem dimension is represented by the total number of variables `#vars` and equality/inequality constraints `#cons`; and the number of NLP iterations until termination is given by `#iter`. Considering $n_s = 80$ as the ideal number of scenarios to use, we see a $\times 116$ total computation increase from the nominal solution and if 16 processors are used (i.e., the maximum advisable N for the given problem size – see discussion below) this number drops by 66% indicating a $\times 39$ increase from the nominal serial solution. Given that we are only parallelizing the discretized implicit DAE integration tasks, a 66% improvement is a promising result. A breakdown of the specific computation performance using speedup $\mathrm{S} = \mathrm{T}_{\mathrm{serial}}/\mathrm{T}_{\mathrm{parallel}}$, where $\mathrm{T}_{\mathrm{serial}}$ and $\mathrm{T}_{\mathrm{parallel}}$ represent serial and parallel program run times, respectively, and efficiency $\mathrm{E} = \mathrm{S}/\mathrm{N}$ is sketched in Figure 3.3. Note, for our particular case we consider each metric to be based on the time to evaluate objective/constraint functionals and derivatives (denoted as DAE time) and exclude the serial in-solver time related to the matrix computations within the NLP solver (denoted as NLP time). From Figure 3.3 (a), the parallel performance in terms of speedup is quite good up to about 8 processors/threads, after which a significant deviation from ideal speedup is observed. This undesirable behavior using $\mathrm{N} \geq 16$, for our chosen problem size of $m = n \cdot n_s$, can be explained using the laws of Amdahl and Gustafson [45]. Amdahl's law gives us an indication of the possible scalability or maximum speedup for a fixed problem size, while Gustafson's law can be used to understand the influence of problem size on scalability. Considering first Amdahl's law, the parallel time can be approximated as $\mathrm{T}_{\mathrm{parallel}} := f\,\mathrm{T}_{\mathrm{serial}} + (1-f)\mathrm{T}_{\mathrm{serial}}/\mathrm{N}$, where f represents an inherent serial fraction of the overall computation, which results in the speedup expression $\mathrm{S}(\mathrm{N}) = (f + (1-f)/\mathrm{N})^{-1}$ and as $\mathrm{N} \to \infty$ the maximum possible speedup is $\mathrm{S}(\infty) = f^{-1}$. So for our particular example, if the time to evaluate the NLP objective/constraint functionals has an inherent serial portion of 10% then we would achieve

a maximum possible speedup of 10. Fortunately, if we further consider the influence of problem size m, whereby the serial fraction of the program is now considered a function of problem size $f(m)$, it can be shown using Gustafson's law that speedup can be given by $\mathrm{S}(m,\mathrm{N}) = f(m) + \mathrm{N}(1 - f(m))$, where $f(m) := a(m)/(a(m) + b(m))$, and $a(m)$ and $b(m)$ represent the inherent serial and parallel portions, respectively. Thus, if we are able better load the processors with more work such that the inherent serial portion diminishes relative to each parallel portion ($b(m) \gg a(m)$) then the fraction $f(m)$ decreases with increasing m and as $m \to \infty$ the speedup will approach N. This concept can be better seen using the log-p model where $\mathrm{T}_{\mathrm{parallel}} := \mathrm{T}_{\mathrm{serial}}/\mathrm{N} + \log_2(\mathrm{N})$ and $\mathrm{T}_{\mathrm{serial}} \propto m$ (see, pg. 79 of [45]). The speedup expression can be derived as $\mathrm{S}(m,\mathrm{N}) = \mathrm{N}/(1 + (\mathrm{N}/m)\log_2(\mathrm{N}))$ and if $m = M\,\mathrm{N}$ where M is the work per processor, then the speedup (and efficiency) can be controlled by limiting the influence of the $\log_2(\mathrm{N})$ term by increasing M. Additionally, to ensure a uniform M on each processor one needs to properly balance and schedule the distribution of work. For example, in our case study we found that if the computation time on each processor for a chunk size of M is relatively constant between processors then a so-called `OpenMP` static scheduling policy is adequate, while if the computation time differs a dynamic (round-robin) policy is preferred which is able to better balance the computation load between processors. To achieve good scalability, one often tries keep the efficiency fixed by increasing the problem size (or rather work per process, M) at the same rate as the number of processors/threads N. If this is possible then the algorithm can be considered weakly scalable; on the other hand, if one is able to keep the efficiency constant for a fixed problem size as N increases then the algorithm is considered strongly scalable. Based on these definitions of scalability, our particular parallel implementation is not strongly scalable; however there is enough evidence to suggest weak scalability. For example, from Figure 3.3 (d) the "DAE time" (i.e., out-of-solver NLP function evaluation time of which the majority represents the parallelized DAE solution) remains relatively constant for a work load of $M = 250$ integration tasks per processor up to about $\mathrm{N} = 16$ after which a slight increase in wall clock time is observed (i.e., decrease in efficiency) which can be attributed to a greater influence of parallel computation

overhead (i.e., the previously noted $\log_2(\mathrm{N})$ term) relative to the chosen computation load M.

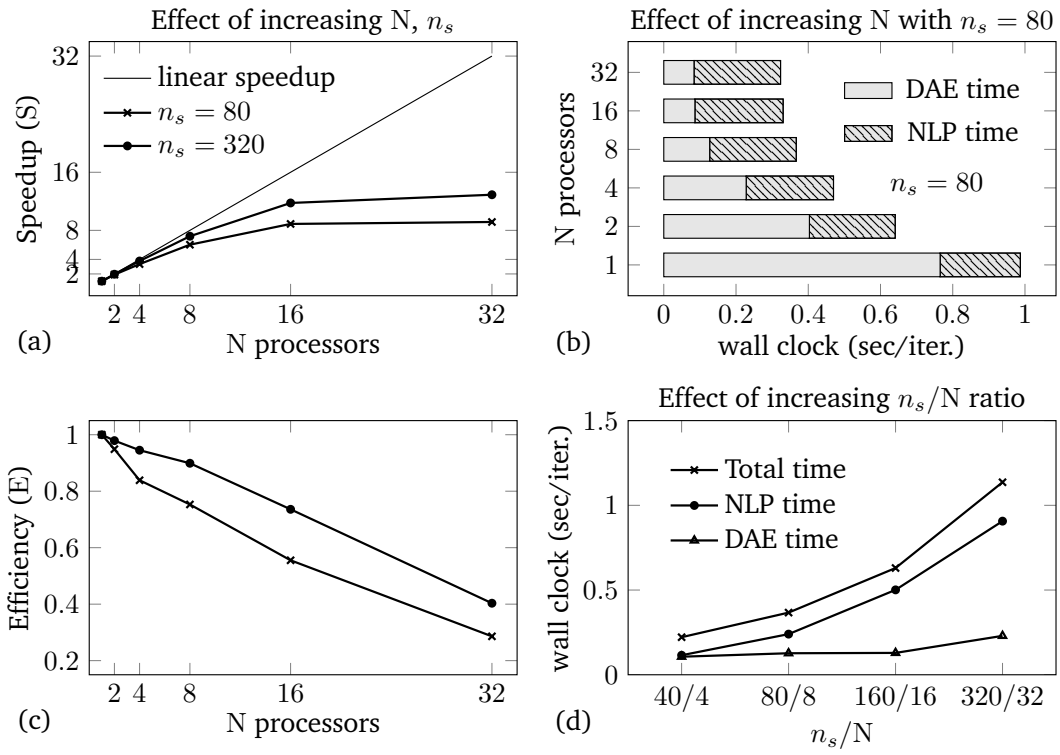

Figure 3.3: Example 1 – speedup, efficiency and wall clock times for increasing n_s

The next aspect of the study considers assessing the use of forward-over-adjoint second-order sensitivity analysis in order to form a representation of the Lagrangian Hessian. Note, that such a procedure is quite expensive given the numerous forward and reverse sweeps of the integrator for all shooting intervals and scenarios, and the objective here is to provide some insight on the additional cost when compared to a quasi-Newton approximation scheme. For demonstration purposes we use the interior-point nonlinear programming solver `IPOPT-3.11.9` with default options and `MA27`, `MC19` for the linear solver and scaling, respectively. Results comparing the limited memory BFGS approximation to the sensitivity approach are reported in Table 3.2, where we highlight the total number of primal-dual IPM iterations, total computation time, time spent in the NLP solver, total time to compute

the continuity constraint Jacobian using forward sensitivity analysis and additional point constraint 1st derivatives using AD which we denote overall as FSA, and total time to compute the lower triangular portion of the Lagrangian Hessian (Equation 3.11) via second-order sensitivity analysis (including all AD computations) which we denote as DSOA. From Table 3.2, comparing columns `qn` and `ex` for quasi-Newton and exact Hessian, respectively, we make the following observations: the DSOA approach reduces the overall number of primal-dual iterations (as one would expect); the total computation time increases on average by about $\times 25$ over the quasi-Newton approach where about 98% of the total computation is spent generating the Lagrangian Hessian. From these results it is quite clear that providing the Lagrangian Hessian of our multiperiod NLP formulation by means of second-order sensitivity analysis is very expensive. From an implementation perspective, the computation in each shooting interval could be parallelized; however, this is unlikely to lead to a significant enough decrease in time to justify the use of second-order sensitivities as implemented in our study. An alternative approach proposed by Hannemann and Marquardt [46] is to use a so-called composite or aggregated approach which only requires a single second-order sensitivity computation encompassing all shooting intervals. Such a technique has shown to reduce the Hessian computation time considerably when used in the context of implicit Runge-Kutta integration techniques. Given our adherence to the SUNDIALS solvers in this work, we have not explored this new technique, but it would be the next logical step.

Table 3.2: Example 1 – serial computation results comparing Hessian generation approach

		#iter		Total (sec)		NLP (sec)		FSA (sec)		DSOA (sec)
n_s	$\mathcal{J}/100^{\dagger}$	qn	ex	qn	ex	qn	ex	qn	ex	ex
1	-1.4911	52	48	0.793	22.96	0.311	0.111	0.482	0.297	22.56
40	-1.5235	70	66	27.81	632.42	2.277	0.591	25.54	12.27	619.56
80	-1.5463	65	64	48.61	1218.46	3.697	1.060	44.91	23.57	1193.82
160	-1.5340	71	65	107.3	2204.61	8.023	1.920	99.32	47.12	2155.57
320	-1.5200	49	44	151.2	4397.64	11.70	3.962	139.5	92.78	4300.90

† IPOPT optimality tolerance $= 1 \times 10^{-6}$

3.4.2 Example 2: Air Separation Problem

This next example explores further the influence processor loading on algorithm scalability and additionally the influence of DAE model size. A large-scale DAE air separation model is used which considers the separation of nitrogen from air. The model used here was adapted from Cao [47], and a simplified process schematic is shown in Figure 3.4.

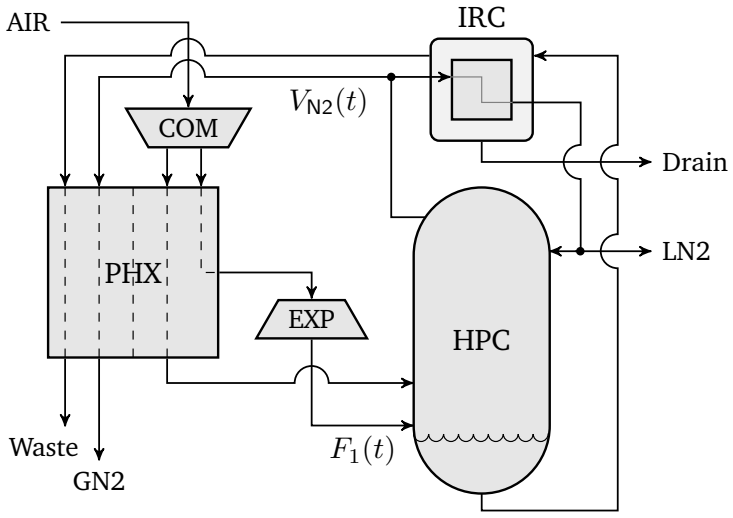

Figure 3.4: Example 2 – air separation process schematic

As a first step, air enters from the atmosphere and is compressed using a multi-staged compressor (COM); impurities are then removed using several adsorption units; high pressure purified air is then cooled in a multi-path heat exchanger (PHX) using the returning gas product (GN2) and waste streams from a high pressure distillation column (HPC); the cooled air stream is then split where a portion goes through a turbine (EXP) to promote further cooling before entering the bottom of the distillation column, while the other stream goes directly to the distillation column. The air entering the column is converted into high purity gaseous nitrogen which exits at the top and crude liquid oxygen which accumulates at the bottom. A portion of the high purity nitrogen gas is drawn off as product (V_{N2}) while the remainder is fed to an integrated reboiler/condenser (IRC) to exchange heat with a crude oxygen stream which is drawn from the bottom of the column. The heat exchange converts gaseous nitrogen to liquid which is then refluxed back to the top of the column and optionally drawn off as liquid nitrogen product (LN2).

The portion of the process we focus on in this study is the distillation column (HPC) and integrated reboiler/condenser (IRC) units, with the air feed to the bottom of the distillation column (F_1) as the input stream. A detailed listing of the variables and equations used in our particular model can be found in Section 3.6 with further details in [47]. The optimization formulation considered seeks to determine a robust control profile to transition from one steady-state to another while satisfying path inequality constraints on product composition (defined in the form of product impurity y_{O2}) and tray flooding (defined implicitly by ensuring the vapor velocity ν_{n_t} is below the flooding velocity $\bar{\nu}_{n_t}$ on the critical tray of n_t which represents the top tray of the column) all while under the influence of uncertainty within key model parameters. The formulation can be defined as a continuous multiperiod

dynamic optimization problem according to the following equations,

$$
\begin{aligned}
\min_{\mathbf{u}(t)} \quad & \mathcal{J} := \textstyle\sum_{i=1}^{n_s} \mathsf{w}_i \int_{t_0}^{t_f} \|\mathbf{y}_i(t) - \mathbf{y}_{\mathsf{sp}}\|^2 + \|\Delta\mathbf{u}(t)\|^2 \, \mathrm{d}t \\
\mathrm{st}: \quad & \text{DAE model (125 ODEs, 329 AEs)} \\
& \mathbf{x}_i(t_0) - \mathbf{x}_{\mathsf{ss}} = \mathbf{0} \;\; \text{(initial steady-state)} \\
& V_{\mathsf{N2}}^{\mathsf{sp}} - V_{\mathsf{N2},i}(t_f) \leq 0 \;\; \text{(min. production rate at } t_f\text{)} \\
& y_{\mathsf{O2},i}(t) - y_{\mathsf{O2}}^{\max} \leq 0 \;\; \text{(max. product impurity)} \\
& \nu_{n_t,i}(t) - \bar{\nu}_{n_t,i}(t) \leq 0 \;\; \text{(avoid tray flooding)} \\
& \mathbf{u}(t) \in [\mathbf{u}^L, \mathbf{u}^U] \,, \; \forall \, t \in [t_0, t_f] \,, \; i = 1, \ldots, n_s \\
& \boldsymbol{\theta} \in [\boldsymbol{\theta}^L, \boldsymbol{\theta}^U]
\end{aligned}
\tag{E.3.2}
$$

where $\mathbf{y}(t) := V_{\mathsf{N2}}(t)$ and $\mathbf{y}_{\mathsf{sp}} := V_{\mathsf{N2}}^{\mathsf{sp}}$ are the measured output and corresponding final desired set-point for the nitrogen production rate; $\mathbf{u}(t) := F_1(t)$ is the manipulated input feed rate of air to the first tray (i.e., column bottom), which we penalize in the objective function according to the rate of change $\Delta\mathbf{u}(t) := \mathrm{d}\mathbf{u}(t)/\mathrm{d}t$; $\boldsymbol{\theta} := [\eta, \, \Delta P]^\top$ is a vector of uncertain model parameters, which we select a priori as the tray efficiency $\eta \in [0.4, 0.6]$ and pressure drop between trays $\Delta P \in [0.45, 0.55]$ kPa. Note the stated DAE model dimension of $n_x = 125$ differential and $n_z = 329$ algebraic variables/equations excludes all subexpression algebraic variables/equations; thus, the algebraic portion of the model is in fact quite larger than it might appear. Select output and state solution trajectories of formulation E.3.2 are plotted in Figure 3.5 for variables: $\bar{F}_1(t) \equiv F_1(t)/F_1(t_0)$ and $\bar{V}_{\mathsf{N2}}(t) \equiv V_{\mathsf{N2}}(t)/V_{\mathsf{N2}}(t_0)$, and path constrained variables: $y_{\mathsf{O2}}(t)$ and $\alpha(t)$, where $\alpha(t) := \nu_{n_t}(t)/\bar{\nu}_{n_t}(t) \leq 1$.

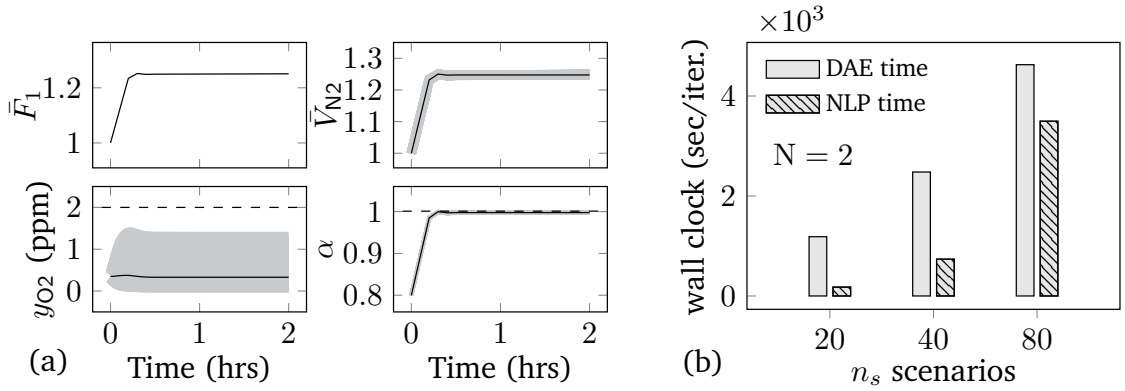

Figure 3.5: Example 2 – (a) robust control & select output trajectories (nominal solution represented by the solid line) and (b) base line DAE & NLP solution times for increasing n_s

A piecewise linear input control profile was selected and the rate of change of this profile was penalized in the optimization objective function. Note, that in order to prevent unnecessary chattering of the control input in the latter portion of the time horizon, the profile uses an evenly distributed parameterization of $n-1$ shooting intervals within the first 0.5 hrs of the time horizon and a final single interval within the remaining 1.5 hrs. From Figure 3.5, we see that the production rate of nitrogen vapor $\bar{V}_{\mathsf{N2}}(t)$ and the vapor velocity ratio $\alpha(t)$ both increase in proportion to the feed air input $\bar{F}_1(t)$ (as one would logically expect); however, due to the flooding constraint there is a clear limitation on the rate of production increase and the influence of parametric uncertainty within the model directly affects the onset of constraint activation. Ultimately we are able to establish an optimal control profile that is robust to the prescribed uncertainties within the model and adherent to the constraints within the formulation. In order to establish a performance base line, we consider a fixed number of shooting intervals, three increasing scenario sizes and two processors. For shooting intervals $n = 6$ and scenarios $n_s = \{20, 40, 80\}$, the average total solution time per major SQP iteration was approximately 1.37×10^3 sec (22.7 min), 3.22×10^3 sec (53.6 min) and 8.12×10^3 sec (135.4 min), respectively; which are about $\times 15.53$, $\times 36.68$, and $\times 92.51$ greater than the nominal solution time of 1.46 min. The corresponding ratio of time spent in the NLP solver versus the DAE solver was 0.14, 0.30 and 0.88, respectively, which indicates that as the

problem size increases the computational burden shifts dramatically from the DAE solver to the in-solver aspects of the NLP solver (e.g., active-set determination, matrix factorizations, matrix-matrix/matrix-vector multiplications, etc.)

Table 3.3: Example 2 – parallel computation results comparing increasing n and n_s

							Total solution time (sec)/1.0×10^5				
n	n_s	m	#vars	#cons	#iter	$\mathcal{J}/10^{\dagger}$	N = 2	N = 16	N = 32	N = 48	N = 64
6	1	6	3184	3194	9/1308	1.0159	0.0079	–	–	–	–
	20	120	63680	63974	14/42704	1.0134	0.1908	0.0535	0.0420	0.0426	0.0501
	40	240	127360	127955	16/85672	1.0181	0.5146	0.1785	0.1589	0.1596	0.1688
	80	480	254720	255915	17/170391	1.0156	1.3808	0.7166	0.6669	0.6720	0.6882
12	1	12	5914	5930	12/3221	1.0141	0.0135	–	–	–	–
	20	240	118280	118808	18/81273	1.0114	0.7163	0.1955	0.1558	0.1491	0.1474
	40	480	236560	237628	11/162037	1.0159	0.9167	0.5179	0.4891	0.4732	0.4714
	80	960	473120	475268	17/322952	1.0129	7.7468	3.2047	2.8156	2.7239	2.6748

† NLP optimality/feasibility tol. $= \{1 \times 10^{-4}, 1 \times 10^{-6}\}$, DAE relative/absolute tol. $= \{1 \times 10^{-3}, 1 \times 10^{-4}\}$

With the base line solution properties established, we now turn to assessing the potential computation speedup via our parallel multiperiod approach. Table 3.3 lists the optimization problem size and solution results, in terms of the number of SQP iterations and total wall clock times, for an incremental number of integration tasks $m = n \cdot n_s$ (where $n = \{6, 12\}$ and $n_s = \{20, 40, 80\}$) using an increasing number of computing processors. Considering first a problem with $n = 6$ and increasing scenario realizations ($n_s = \{20, 40, 80\}$), we observed good scaling properties using at most $\mathrm{N} \leq 32$. For example, using $\mathrm{N} = 16$ we observe an overall average computation speedup of $\times 3.57$, $\times 2.88$, $\times 1.93$ for each n_s, respectively; and for $\mathrm{N} = 32$, $\times 4.54$, $\times 3.24$, $\times 2.07$; where the decrease in rate of speedup from 16 to 32 processors is due to the increasing serial portion of the NLP solver. If we remove this large serial NLP portion, it can be confirmed that the parallel implementation of the embedded

DAE shooting intervals can be done fairly efficiently. For example, Figure 3.6 (a) provides speedup trends for N from 2 to 64 at a fixed amount of work (m) based on the so-called "DAE time" as defined in the previous case study. It is evident that good strong scaling properties are observed up to about 32 processors, after which a more sharply decreasing rate is observed which can be attributed in part to an insufficient work load per processor M. Figure 3.6 (b) compares the amount of time, per major SQP iteration, spent in the DAE solver versus the NLP solver. Considering a problem size of $m = 480$ and imposing an increase in processors from 2 to 64, we see a significant reduction in DAE solution time relative the serial NLP solution time. Furthermore, from Figure 3.6 (c), if we increase the number of processors in proportion to the problem size m the DAE solution time remains relatively constant which indicates reasonably good weak-scaling properties. For the case of $n = 12$ (see, Figure 3.6 (d)–(e)), where we effectively double the NLP size, better speedup with increasing number of processors is observed, which indicates the expected result that as we increase the work per processor (M) we also reduce the relative parallel overhead and ultimately see better strong scaling properties. However, for the particular active-set SQP solver used in this study, we quickly run into significant serial computation overhead within the QP subproblems (i.e., a sharp increase in the number of QP iterates).

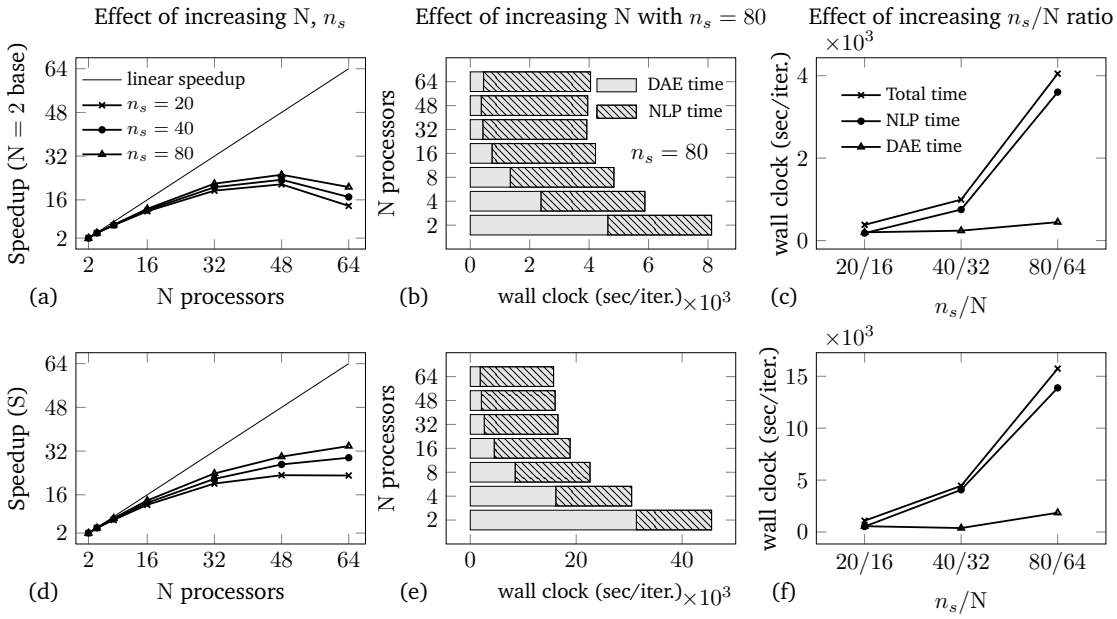

Figure 3.6: Example 2 – speedup and wall clock times for increasing N and n_s, where $n = 6$ fixed for (a)–(c) and $n = 12$ fixed for (d)–(f)

Table 3.4: Example 2 – parallel computation results comparing different DAE dimensions

						Total solution time (sec)/1.0×10^5				
$n_x/n_z{}^*$	$m^\dagger$	#vars	#cons	#iter	$\mathcal{J}/10$	N = 2	N = 16	N = 32	N = 48	N = 64
23/57	6	566	568	19/184	0.8559	0.0006	–	–	–	–
	480	45280	45915	22/16326	0.8560	0.0824	0.0230	0.0247	0.0368	0.0416
59/153	6	1490	1492	12/616	1.0111	0.0011	–	–	–	–
	480	119200	119835	18/38928	1.0110	0.2185	0.0896	0.0870	0.0881	0.1052
125/329	6	3184	3194	10/1294	1.0159	0.0079	–	–	–	–
	480	254720	255915	17/170391	1.0156	1.3808	0.7166	0.6669	0.6720	0.6882

* dimension based on number of distillation trays $n_t = \{5, 17, 39\}$;

† problem size based on $n = 6$ and $n_s = \{1, 80\}$

Finally, we consider the aspect of increasing the embedded DAE size and provide a relative comparison on the influence of DAE size on the overall parallel scalability of the implementa-

tion. Table 3.4 compares model sizes of $n_x/n_z = \{23/57,\ 59/153,\ 125/329\}$ where n_x and n_z represent the number of differential and algebraic state variables, respectively, that are a result of increasing the number of distillation column trays according to $n_t = \{5,\ 17,\ 39\}$. The base line total solution time per SQP iteration for each model size using $n_s = 80$ scenarios was 6.24 min, 20.23 min and 135.37 min, respectively, which are over 100 times the nominal solution time. Through parallelization these base line times can be reduced by about $\times 3.6$, $\times 2.4$, and $\times 1.9$, respectively; where again we observe that as the serial NLP portion grows the potential speedup diminishes. Figure 3.7 reveals more closely the speedup and efficiency excluding the influence of the serial in-solver NLP contribution. As the model size is increased (i.e., more expensive integration tasks) a more pronounced improvement is observed when compared to increasing the number of integration tasks per processor via discretization alone (i.e., increasing M via increasing n_s or n). In other words, if one compares the delta in speedup from $n_x/n_z = 59/153$ and $n_x/n_z = 125/329$ in Figure 3.7 (a) to the delta in speedup from $n_s = 40$ and $n_s = 80$ in Figure 3.6 (a) then a larger deviation results from increasing the model size versus the discretization refinement. This result is particularly positive and highlights that one is able to more easily achieve better parallel scalability using larger embedded DAE models versus creating more independent integration tasks.

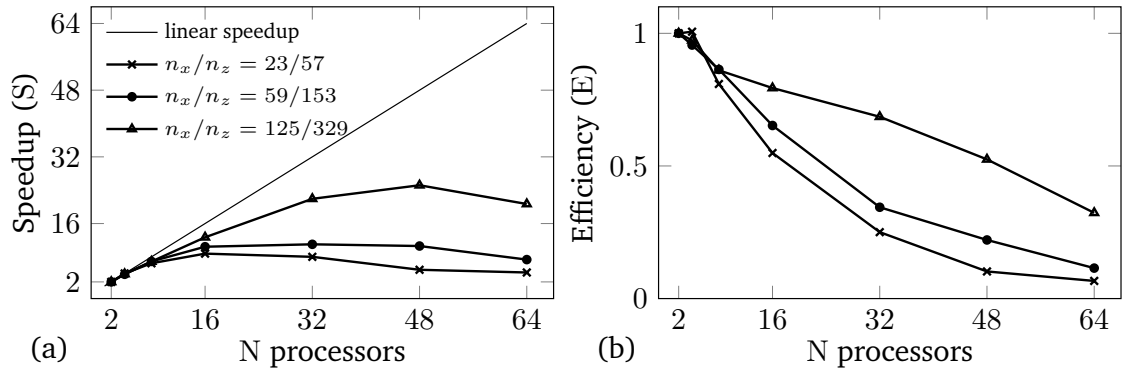

Figure 3.7: Example 2 – speedup and efficiency for increasing DAE size n_x/n_z based on $n_t = \{5,\ 17,\ 39\}$, with $n = 6$, $n_s = 80$ fixed

3.5 Concluding Remarks

In this chapter we have presented a parallel computing approach for large-scale dynamic optimization under uncertainty that targets the decomposition of the embedded differential-algebraic equation model. A combined multiperiod multiple-shooting approach was used to discretize the DAE optimization formulation to yield a multiperiod NLP formulation with embedded implicit DAE functionals within the constraints. The DAE model and sensitivity equations corresponding to each shooting interval and scenario constitute independent integration tasks, well-suited for parallel processing. Our multiperiod approach was applied to a large-scale DAE air separation model comprising up to 125 ODEs and 329 algebraic equations for the purpose of obtaining a robust optimal control profile subject to uncertainty in the model parameters. Results indicated fairly good parallel scalability using a parallel `OpenMP` implementation of the DAE solution; however, the extent of scalability depends largely on the amount of work per processor and on the ability to effectively balance the work load between processors. In this chapter we were able to demonstrate the benefits of parallelizing the DAE solution portion of the multiple-shooting algorithm; however, as the NLP size grows with scenario realizations, the computation bottleneck shifts to the NLP solver. While it is possible to alleviate some of the computation burden through the use of a sparse interior-point NLP solver (as opposed to an active-set solver, primarily used in this study), a better approach, and the subject for future work, would be to take advantage of the multiperiod block structure of the NLP by exploiting the partial separability of the system and tailoring the linear algebra within the algorithm (possibly through an interior-point QP solution strategy within an overall SQP approach) to suit a given structure which would speed up the algebraic computations and reduce the overall memory consumption.

3.6 Air Separation Model Equations

The air separation system considered in this study is limited to just the distillation column model. The column is modeled using several component sub-models; (1) column sump, (2) tray and (3) condenser and reboiler sides of a combined integrated reboiler-condenser. The combination of these components represents a DAE system comprising material and energy balances, composition summation equations, a tray efficiency equation, a tray hydraulic equation, and an equation of state to determine the vapor phase molar volume. The model represents a continuous tray-by-tray equilibrium-based distillation system and the formulation utilized here is an index-1 DAE, which we manually reduced from an initially index-2 formulation. The notation used in the model presentation is similar to that used in many standard separation/thermodynamic textbooks; however, for a description of the specific variables and equations one can refer to [47].

Sump model ($n = 0$):

$$\dot{M}_0 - (L_1 - L_0) = 0 \tag{3.20}$$

$$M_0\,\dot{x}_{i,0} - L_1(x_{i,1} - x_{i,0}) = 0 \quad \forall\ i = 2,3 \tag{3.21}$$

$$\textstyle\sum_{i=1}^{3} x_{i,0} - 1 = 0 \tag{3.22}$$

$$L_0 - M_0/\tau_0 = 0 \tag{3.23}$$

Tray model ($n = 1, \ldots, n_t$):

$$\dot{M}_n - (L_{n+1} + V_{n-1} - (L_n + V_n) + F_n) = 0 \quad \forall\ n = 1, \ldots, n_t \tag{3.24}$$

$$\begin{aligned} M_n\,\dot{x}_{i,n} - (L_{n+1}(x_{i,n+1} - x_{i,n}) + V_{n-1}(y_{i,n-1} - x_{i,n}) \\ -V_n(y_{i,n} - x_{i,n}) + F_n(z_{i,n} - x_{i,n})) = 0 \end{aligned} \quad \forall\ i = 2,3\ ,\ n = 1, \ldots, n_t \tag{3.25}$$

$$\begin{aligned} M_n\,\widehat{H}^l_n - (L_{n+1}(H^l_{n+1} - H^l_n) + V_{n-1}(H^v_{n-1} - H^l_n) \\ -V_n(H^v_n - H^l_n) + F_n(H^z_n - H^l_n)) = 0 \end{aligned} \quad \forall\ n = 1, \ldots, n_t \tag{3.26}$$

$$\sum_{i=1}^{3} x_{i,n} - 1 = 0 \quad \forall \; n = 1, \ldots, n_t \tag{3.27}$$

$$\sum_{i=1}^{3} y_{i,n} - 1 = 0 \quad \forall \; n = 1, \ldots, n_t \tag{3.28}$$

$$y_{i,1} - K_{i,1}\, x_{i,1} = 0 \quad \forall \; i = 1, 2, 3 \tag{3.29}$$

$$y_{i,n} - (y_{i,n-1} + \eta(K_{i,n} x_{i,n} - y_{i,n})) = 0 \quad \forall \; i = 1, 2, 3 \;,\; n = 2, \ldots, n_t \tag{3.30}$$

$$M_n/(\rho_n\, A_{\text{tray}}) - (h_{\text{weir}} + 1.41(L_n/(\sqrt{g}\, \rho_n\, l_{\text{weir}}))^{2/3}) = 0 \quad \forall \; n = 1, \ldots, n_t \tag{3.31}$$

$$P_n - \left(\frac{R\, T_n}{v_n - b_n} - \frac{a_n}{(v_n - 0.414\, b_n)(v_n + 2.414\, b_n)} \right) = 0 \quad \forall \; n = 1, \ldots, n_t \tag{3.32}$$

IRC condenser-side model ($n = n_t + 1$):

$$V_{n_t}(1 - r_{\text{draw}}) - L_{n_t+1} = 0 \tag{3.33}$$

$$L_{n_t+1} - L_{\text{reflux}} - L_{N2} = 0 \tag{3.34}$$

$$V_{n_t}(1 - r_{\text{draw}}) H^v_{n_t} - L_{n_t+1} H^l_{n_t+1} - Q = 0 \tag{3.35}$$

$$x_{i,n_t+1} - y_{i,n_t} = 0 \quad \forall \; i = 1, 2, 3 \tag{3.36}$$

$$P_{n_t+1} - \sum_{i=1}^{3} x_{i,n_t+1}\, P^*_{i,n_t+1} = 0 \tag{3.37}$$

IRC reboiler-side model ($n = n_t + 2$):

$$\dot{M}_{n_t+2} - (L_0 - (L_{n_t+2} + V_{n_t+2})) = 0 \tag{3.38}$$

$$M_{n_t+2}\, \dot{x}_{i,n_t+2} - (L_0(x_{i,0} - x_{i,n_t+2}) - V_{n_t+2}(y_{i,n_t+2} - x_{i,n_t+2})) = 0 \quad \forall \; i = 2, 3 \tag{3.39}$$

$$L_0(H^l_0 - H^l_{n_t+2}) - V_{n_t+2}(H^v_{n_t+2} - H^l_{n_t+2}) + Q = 0 \tag{3.40}$$

$$\sum_{i=1}^{3} x_{i,n_t+2} - 1 = 0 \tag{3.41}$$

$$\sum_{i=1}^{3} y_{i,n_t+2} - 1 = 0 \tag{3.42}$$

$$y_{i,n_t+2} - K_{i,n_t+2}\, x_{i,n_t+2} = 0 \quad \forall \; i = 1, 2, 3 \tag{3.43}$$

$$\Delta T_{\text{irc}} - T_{n_t+1} + T_{n_t+2} = 0 \tag{3.44}$$

$$L_{n_t+2} - M_{n_t+2}/\tau_{n_t+2} = 0 \tag{3.45}$$

Subexpressions:

$$Q := (UA)_{\text{irc}}\,\Delta T_{\text{irc}} \tag{3.46}$$

$$P_n := P_{n_t+1} + (n_t - n + 1)\,\Delta P \tag{3.47}$$

$$K_{i,n} := \gamma_{i,n} P^*_{i,n}/P_n \tag{3.48}$$

$$\gamma_{i,n} := \exp\left[\textstyle\sum_{j=1}^{3}\sum_{k=1}^{3}(A_{ji} - 0.5A_{jk})x_{j,n}x_{k,n}/(R\,T_n)\right] \tag{3.49}$$

$$P^*_{i,n} := \exp(A_i + B_i/(T_n + C_i)) \tag{3.50}$$

$$H^l_n := \textstyle\sum_{i=1}^{3} x_{i,n}\,h^l_{i,n} \tag{3.51}$$

$$H^v_n := \textstyle\sum_{i=1}^{3} y_{i,n}\,h^v_{i,n} \tag{3.52}$$

$$h^l_{i,n} := c_{i,1} + c_{i,2}\,T_n \tag{3.53}$$

$$h^v_{i,n} := d_{i,1} + d_{i,2}\,T_n \tag{3.54}$$

$$\rho_n := \textstyle\sum_{i=1}^{3}\, x_{i,n}\,\rho_{i,n} \tag{3.55}$$

$$\rho_{i,n} := \frac{P^c_i}{R\,T^c_i}\,r_i^{-(1+(1-T_n/T^c_i)^{2/7})} \tag{3.56}$$

$$a_n := \textstyle\sum_{i=1}^{3}\sum_{j=1}^{3}\, y_{i,n}y_{j,n}(a_{i,n}a_{j,n})^{0.5}(1 - k_{ij}) \tag{3.57}$$

$$a_{i,n} := 0.45724(R\,T^c_i)^2/P^c_i\big(1 + f_i\,\left(1 - \sqrt{T_n/T^c_i}\right)\big)^2 \tag{3.58}$$

$$b_n := \textstyle\sum_{i=1}^{3}\, y_{i,n}\,b_{i,n} \tag{3.59}$$

$$b_{i,n} := 0.0778\,R\,T^c_i/P^c_i \tag{3.60}$$

$$f_i := 0.37464 + 1.54226\,\omega_i - 0.26992\,\omega_i^2 \tag{3.61}$$

$$l_{\text{weir}} := f_{\text{weir}} D_{\text{col}} \tag{3.62}$$

$$A_{\text{col}} := (\pi/4) D^2_{\text{col}} \tag{3.63}$$

$$A_{\text{tray}} := f_{\text{active}} A_{\text{col}} \tag{3.64}$$

$$\widehat{H}^l_n := \textstyle\sum_{i=1}^{3}\frac{\partial H^l_n}{\partial x_{i,n}}\,\widehat{x}_{i,n} + \frac{\partial H^l_n}{\partial T_n}\,\widehat{T}_n \tag{3.65}$$

$$\widehat{T}_n := \frac{-\sum_{i=1}^{3}\left(K_{i,n}\,\widehat{x}_{i,n} + x_{i,n}\,\sum_{k=1}^{3}\frac{\partial K_{i,n}}{\partial x_{k,n}}\,\widehat{x}_{k,n}\right)}{\sum_{i=1}^{3} x_{i,n}\frac{\partial K_{i,n}}{\partial T_n}} \tag{3.66}$$

$$\begin{aligned}\widehat{x}_{i,n} := (&L_{n+1}(x_{i,n+1} - x_{i,n}) + V_{n-1}(y_{i,n-1} - x_{i,n}) \\ &- V_n(y_{i,n} - x_{i,n}) + F_n(z_{i,n} - x_{i,n}))/M_n\end{aligned} \tag{3.67}$$

Objective/constraint function subexpressions:

$$V_{N2} := V_{n_t}\, r_{\text{draw}} \tag{3.68}$$

$$y_{O2} := y_{n_t,2} \times 10^6 \tag{3.69}$$

$$\nu_n := V_n\, v_n / A_{\text{tray}} \tag{3.70}$$

$$\bar{\nu}_n := C_n \left(\frac{\sigma_n}{0.02}\right)^{0.2} \left(\frac{\rho_n\, mw_n^l - mw_n^v/v_n}{mw_n^v/v_n}\right)^{0.5} \tag{3.71}$$

$$C_n := (0.0744\, S + 0.0117) \log_{10}(FP_n^{-1}) + 0.0304\, S + 0.0153 \tag{3.72}$$

$$FP_n := \frac{L_n\, mw_n^l}{V_n\, mw_n^v} \left(\frac{mw_n^v/v_n}{\rho_n\, mw_n^l}\right)^{0.5} \tag{3.73}$$

$$\sigma_n := \left(\sum_{i=1}^{3} p_i\, (\rho_n\, x_{i,n} - y_{i,n}/v_n)\right)^4 \tag{3.74}$$

$$mw_n^l := \sum_{i=1}^{3} x_{i,n}\, mw_i \tag{3.75}$$

$$mw_n^v := \sum_{i=1}^{3} y_{i,n}\, mw_i \tag{3.76}$$

Many of the subexpression equations represent thermodynamic properties which require parameter values. These parameters are readily available from standard reference books or thermodynamic property databases. For our study we primarily obtained all parameter values from the Multiflash thermodynamic database. However, for the liquid and vapor phase enthalpy expressions we used Aspen Properties data to fit a linear expression as a function of temperature. Additional parameters used are listed in Table 3.5.

Table 3.5: Example 2 – air separation model parameter values

Description	**Symbol**	**Value**	**Units**
feed tray	n_f	1	–
gravitational constant	g	9.81	m/s^2
gas constant	R	8.31451	m^{3}kPa/(kmol K)
column diameter	D_{col}	0.8	m
weir height	h_{weir}	0.05	m
tray spacing	S	0.1778	m
fraction of active tray area	f_{active}	0.67	–
fraction of column diameter	f_{weir}	0.75	–
IRC heat transfer area/coefficient	$(UA)_{\text{irc}}$	0.113488	MJ/(K s)
liquid N2 distillate product	L_{N2}	1.26×10^{-7}	kmol/s
product gas draw fraction	r_{draw}	0.2	–
condenser-side pressure	P_{n_t+1}	319.0673	kPa
sump holdup time constant	τ_0	120	s
reboiler-side holdup time constant	τ_{n_t+2}	120	s
feed composition $i = \{N2, O2, Ar\}$	z_i	$\{0.7811, 0.2096, 0.0093\}$	–

List of References

[1] A. Geletu and P. Li. “Recent Developments in Computational Approaches to Optimization under Uncertainty and Application in Process Systems Engineering”. In: *ChemBioEng Reviews* 1.4 (2014), pp. 170–190 (cit. on p. 62).

[2] M. J. Mohideen, J. D. Perkins, and E. N. Pistikopoulos. “Optimal design of dynamic systems under uncertainty”. In: *AIChE Journal* 42.8 (1996), pp. 2251–2272 (cit. on pp. 62, 63).

[3] V. Sakizlis, J. D. Perkins, and E. N. Pistikopoulos. “Recent advances in optimization-based simultaneous process and control design”. In: *Computers & Chemical Engineering* 28.10 (2004), pp. 2069–2086 (cit. on p. 62).

[4] S. Wang and M. Baldea. “Identification-based optimization of dynamical systems under uncertainty”. In: *Computers & Chemical Engineering* 64 (2014), pp. 138–152 (cit. on p. 62).

[5] U. Diwekar. *Introduction to Applied Optimization*. Springer, 2008 (cit. on p. 63).

[6] H. Arellano-Garcia and G. Wozny. “Chance constrained optimization of process systems under uncertainty: I. Strict monotonicity”. In: *Computers & Chemical Engineering* 33.10 (2009), pp. 1568–1583 (cit. on p. 63).

[7] M. Kloppel et al. “Using Sparse-Grid Methods To Improve Computation Efficiency in Solving Dynamic Nonlinear Chance-Constrained Optimization Problems”. In: *Industrial & Engineering Chemistry Research* 50.9 (2011), pp. 5693–5704 (cit. on p. 63).

[8] M. Diehl et al. “Numerical solution approaches for robust nonlinear optimal control problems”. In: *Computers & Chemical Engineering* 32.6 (2008), pp. 1279–1292 (cit. on p. 63).

[9] B. Houska et al. “Robust optimization of nonlinear dynamic systems with application to a jacketed tubular reactor”. In: *Journal of Process Control* 22.6 (2012), pp. 1152–1160 (cit. on p. 63).

[10] R. Huang and L. T. Biegler. “Robust nonlinear model predictive controller design based on multi-scenario formulation”. In: *Proceedings of the 2009 Conference on American Control Conference*. ACC’09. 2009, pp. 2341–2342 (cit. on p. 63).

[11] S. Lucia et al. “Handling uncertainty in economic nonlinear model predictive control: A comparative case study”. In: *Journal of Process Control* 24.8 (2014), pp. 1247–1259 (cit. on p. 63).

[12] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical solution of initial-value problems in differential-algebraic equations*. Philadelphia: SIAM, 1996 (cit. on p. 65).

[13] D. K. Varvarezos, L. T. Biegler, and I. E. Grossmann. “Multiperiod design optimization with SQP decomposition”. In: *Computers & Chemical Engineering* 18.7 (1994), pp. 579–595 (cit. on p. 66).

[14] T. K. Bhatia and L. T. Biegler. “Multiperiod design and planning with interior point methods”. In: *Computers & Chemical Engineering* 23.7 (1999), pp. 919–932 (cit. on pp. 66, 82).

[15] J. Albuquerque et al. “Interior point SQP strategies for large-scale, structured process optimization problems”. In: *Computers & Chemical Engineering* 23.4-5 (1999), pp. 543–554 (cit. on p. 66).

[16] A. M. Cervantes et al. “A reduced space interior point strategy for optimization of differential algebraic systems”. In: *Computers & Chemical Engineering* 24.1 (2000), pp. 39–51 (cit. on p. 66).

[17] V. M. Zavala, C. D. Laird, and L. T. Biegler. “Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems”. In: *Chemical Engineering Science* 63.19 (2008), pp. 4834–4845 (cit. on p. 66).

[18] D. P. Word et al. “Efficient parallel solution of large-scale nonlinear dynamic optimization problems”. In: *Computational Optimization and Applications* 59.3 (2014), pp. 667–688 (cit. on p. 66).

[19] J. Kang et al. “An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition”. In: *Computers & Chemical Engineering* 71 (2014), pp. 563–573 (cit. on p. 66).

[20] D. B. Leineweber et al. “An efficient mulitple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part II: Software aspects and applications”. In: *Computers & Chemical Engineering* 27.2 (2003), pp. 167–174 (cit. on p. 67).

[21] B. Bachmann et al. “Parallel Multiple-Shooting and Collocation Optimization with OpenModelica”. In: *9th International Modelica Conference*. Munich, Germany, Sept. 2012, pp. 659–668 (cit. on p. 67).

[22] J. Andersson. “A General-Purpose Software Framework for Dynamic Optimization”. PhD thesis. Arenberg Doctoral School, KU Leuven, Department of Electrical Engineering (ESAT/SCD) and Optimization in Engineering Center, 2013 (cit. on p. 68).

[23] D. B. Leineweber et al. “An efficient mulitple shooting based reduced SQP strategy for large-scale dynamic process optimization. Part I: Theoretical aspects”. In: *Computers & Chemical Engineering* 27.2 (2003), pp. 157–166 (cit. on pp. 68, 70).

[24] B. Houska and M. Diehl. “A quadratically convergent inexact SQP method for optimal control of differential algebraic equations”. In: *Optimal Control Applications and Methods* 34.4 (2013), pp. 396–414 (cit. on p. 70).

[25] T. Maly and L. R. Petzold. “Numerical methods and software for sensitivity analysis of differential-algebraic systems”. In: *Applied Numerical Mathematics* 20.1-2 (1996), pp. 57–79 (cit. on p. 73).

[26] W. F. Feehery, J. E. Tolsma, and P. I. Barton. “Efficient sensitivity analysis of large-scale differential-algebraic systems”. In: *Applied Numerical Mathematics* 25.1 (1997), pp. 41–54 (cit. on p. 73).

[27] M. Schlegel et al. “Sensitivity analysis of linearly-implicit differential-algebraic systems by one-step extrapolation”. In: *Applied Numerical Mathematics* 48.1 (2004), pp. 83–102 (cit. on p. 73).

[28] M. R. Kristensen et al. “Sensitivity analysis in index-1 differential algebraic equations by ESDIRK methods”. In: *IFAC World Congress*. Vol. 16. 1. 2005, pp. 895–895 (cit. on p. 73).

[29] A. Hartwich et al. “Parallel sensitivity analysis for efficient large-scale dynamic optimization”. In: *Optimization and Engineering* 12.4 (2011), pp. 489–508 (cit. on p. 73).

[30] D. B. Ozyurt and P. I. Barton. “Cheap Second Order Directional Derivatives of Stiff ODE Embedded Functionals”. In: *SIAM Journal on Scientific Computing* 26.5 (2005), pp. 1725–1743 (cit. on p. 75).

[31] Y. Cao et al. “Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution”. In: *SIAM Journal on Scientific Computing* 24.3 (2003), pp. 1076–1089 (cit. on p. 75).

[32] R. Hannemann-Tamas. “Adjoint Sensitivity Analysis for Optimal Control of Non-Smooth Differential-Algebraic Equations”. PhD thesis. RWTH-Aachen University, 2012 (cit. on p. 75).

[33] J. Albersmeyer and M. Diehl. “The Lifted Newton Method and Its Application in Optimization”. In: *SIAM Journal on Optimization* 20.3 (2010), pp. 1655–1684 (cit. on p. 75).

[34] T. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006 (cit. on p. 77).

[35] A. C. Hindmarsh et al. “SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers”. In: *ACM Transactions on Mathematical Software* 31.3 (2005), pp. 363–396 (cit. on p. 77).

[36] P. E. Gill, W. Murray, and M. A. Saunders. “SNOPT: An SQP algorithm for large-scale constrained optimization”. In: *SIAM Review* 47.1 (2005), pp. 99–131 (cit. on p. 77).

[37] A. Wachter and L. T. Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical Programming* 106.1 (2006), pp. 25–57 (cit. on p. 77).

[38] A. Walther and A. Griewank. “Getting started with ADOL-C”. In: *Combinatorial Scientific Computing*. Ed. by U. Naumann and O. Schenk. Chapman-Hall CRC Computational Science, 2012. Chap. 7, pp. 181–202 (cit. on p. 77).

[39] J. Albersmeyer and H. G. Bock. “Sensitivity Generation in an Adaptive BDF-Method”. In: *Modeling, Simulation and Optimization of Complex Processes*. Ed. by H. G. Bock et al. Springer, 2008, pp. 15–24 (cit. on p. 77).

[40] R. Quirynen et al. “Autogenerating microsecond solvers for nonlinear MPC: A tutorial using ACADO integrators”. In: *Optimal Control Applications and Methods* 36.5 (2014), pp. 685–704 (cit. on p. 77).

[41] R. Hannemann-Tamas and L. S. Imsland. “Full algorithmic differentiation of a Rosenbrock-type method for direct single shooting”. In: *Control Conference (ECC), 2014 European*. June 2014, pp. 1242–1248 (cit. on p. 77).

[42] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd. Springer, 2006 (cit. on p. 78).

[43] T. K. Bhatia and L. T. Biegler. “Dynamic Optimization in the Design and Scheduling of Multiproduct Batch Plants”. In: *Industrial & Engineering Chemistry Research* 35.7 (1996), pp. 2234–2246 (cit. on p. 82).

[44] L. T. Biegler. *Nonlinear Programming: Concepts, algorithms, and applications to chemical processes*. SIAM, 2010 (cit. on p. 82).

[45] P. S. Pacheco. *An Introduction to Parallel Programming*. New York, NY, USA: Morgan Kaufmann, 2011 (cit. on pp. 85, 86).

[46] R. Hannemann and W. Marquardt. “Continuous and discrete composite adjoints for the hessian of the lagrangian in shooting algorithms for dynamic optimization”. In: *SIAM Journal on Scientific Computing* 31.6 (2010), pp. 4675–4695 (cit. on p. 88).

[47] Y. Cao. “Design for Dynamic Performance: Application to an Air Separation Unit”. MA thesis. Hamilton, On., Canada: Department of Chemical Engineering, 2011 (cit. on pp. 89, 90, 98).

Chapter 4

Towards a Structure Exploiting Parallel NLP Algorithm for Multiperiod Dynamic Optimization

This chapter develops a sequential quadratic programming (SQP) algorithm that utilizes a parallel interior-point method (IPM) for the QP subproblems. Our approach is able to efficiently decompose and solve large-scale multiperiod nonlinear programming (NLP) formulations with embedded dynamic model representations, through the use of an explicit Schur-complement decomposition within the IPM. The algorithm implementation makes use of an appropriate computing environment that uses the parallel distributed computing message passing interface (MPI) and specialized vector-matrix class representations, as implemented in the third-party software package OOPS. The proposed approach is assessed, with a focus on computational speedup, using several benchmark examples involving applications of parameter estimation and design under uncertainty which utilize static and dynamic process models. Results indicate significant improvements in the NLP solution speedup when moving from a serial full-space direct factorization approach, to a serial Schur-complement decomposition, to a parallelized Schur-complement decomposition for the primal-dual linear system solution within the IPM. Consequently, we have demonstrated progress towards solving multiperiod formulations, particularly those with expensive embedded ODE/DAE

model evaluations, which can tractably accommodate a significant increase in the number scenario realizations. This highlights the possibility for solving even larger plant-wide model representations with finer discretization levels which previously required impractical solution times.

Note, portions of this chapter were submitted for review to the journal Computers & Chemical Engineering.

4.1 Introduction

Higher operating costs and shrinking profit margins in the chemical and petro-chemical industries are driving greater applications of advanced control techniques and even further consideration of control at the process design stage. These applications are often model-based and require the solution of dynamic optimization formulations comprising very large systems of variables and equations that can be computationally demanding, requiring considerable computer memory and solution times. Accordingly, these applications are motivating the development and implementation of solution approaches, numerical techniques and algorithms capable of efficiently exploiting modern computational resources in terms utilizing multiple computers, multiprocessor systems and/or acceleration devices. Applications of industrial relevance include: model predictive control (MPC) and moving horizon state estimation (MHE) formulations comprised of large block structured matrices due to their respective discretized time horizons [1]; and multiperiod approximations of stochastic optimization formulations, which present a block-bordered diagonal matrix structure as a result of the repetitive scenario/period realizations [2]. Industrial applications of this nature typically produce optimization formulations involving upwards of a few hundred thousand to a million variables and constraints, which makes the use of decomposition techniques combined with some form of parallel computing a necessity. Early chemical engineering applications involving multiperiod formulations include the simultaneous design and operation of batch

distillation [3], and the integration of design, planning and scheduling for multi-product batch plants [4].

Past work on algorithm development has primarily focused on improving the efficiency of the linear algebra computations through exploiting structure via decomposition. Varvarezos, Biegler, and Grossmann [5] proposed a reduced sequential quadratic programming (rSQP) approach (based on an active-set QP subproblem) which decomposes the multiperiod structure through introducing additional linear constraints and scenario dependent parameters which effectively removes the potentially nonlinear complicating scenario independent parameters and forms a new NLP structure which is easier to solve at the QP level. Bartlett and Biegler [6] present an active-set QP algorithm that uses a Schur-complement decomposition and demonstrate its superiority when compared to several other active-set QP solvers on structured linear MPC and reduced SQP problems. This decomposed rSQP approach was further explored by Bhatia and Biegler [7] who use an interior-point method (IPM) for each QP subproblem. Ultimately, the resulting interior-point rSQP technique showed superior scalability (with respect to scenario realizations) compared to the active-set rSQP approach [8]. Following this work, Albuquerque et al. [9] developed a similar interior-point SQP approach for discretized dynamic optimization formulations which resulted in significant performance improvements. More recently, Zavala, Laird, and Biegler [10] have demonstrated a parallel primal-dual nonlinear interior-point approach to tackle discretized multiperiod dynamic optimization formulations. This has ultimately led to general nonlinear interior-point approaches to handle discretized nominal dynamic optimization formulations [11] and structured NLP formulations [12]. In a similar direction, Gondzio and Grothey [13, 14] introduced a primal-dual interior-point QP algorithm and linear algebra library that uses a block-wise Cholesky decomposition within a Schur-complement decomposition. Their algorithm uses a distributed-memory parallel implementation and they demonstrate near perfect speedups on several large scale stochastic QP formulations. Following this, further extensions were made in [15] to demonstrate a parallel SQP-IPM algorithm for stochastic nonlinear

optimization. From an on-line control perspective, Domahidi et al. [16] discuss a primal-dual interior-point QP algorithm tailored to structured linear MPC formulations, which again uses a block-wise Cholesky factorization. They demonstrate superior performance when using the block-wise factorization approach compared to the full-space approach. Following from this, Frasch, Sager, and Diehl [17] proposed a novel QP solution approach, algorithm and software package that targets optimal control formulations and demonstrate superior solution times when compared to a few other established solvers.

The particular direction that we follow in this work is a solution algorithm for nonlinear multiperiod optimization formulations which targets dynamic process models with uncertain parameters and/or disturbance inputs. To this end, we are investigating the efficiency of an SQP-IPM algorithm for multiperiod optimization formulations, as well as discretization approaches of the uncertainty space combined with a multiple-shooting discretization of the temporal domain, which requires the solution of an embedded dynamic model within an overall nonlinear programming algorithm. It is hypothesized that the proposed SQP-IPM approach is more efficient than nonlinear IPM algorithms for handling NLP formulations with embedded dynamic models which are very expensive to evaluate at each iterate of the algorithm. In this chapter, we are primarily concerned with building on the work performed in [18] by addressing the computational bottleneck encountered when using a serial active-set rSQP algorithm. Accordingly, we seek to demonstrate a sequential quadratic programming algorithm where the QP subproblems are solved with an interior-point method via the OOPS solver which employs a direct Schur-complement decomposition [14]. Additionally, a partially separable quasi-Newton strategy is developed and used to iteratively approximate and update second-order information required to formulate the QP subproblems. A novel aspect of the work is the development and assessment of a parallel nonlinear multiperiod optimization approach for design under uncertainty, where we specifically target the application of large-scale dynamic optimization using embedded ODE/DAE model formulations via the multiple-shooting solution approach.

The chapter is laid out by first discussing the nonlinear programming solution algorithm and relevant supporting literature. Next, we discuss how the underlying problem formulation is decomposed to facilitate structure exploitation within the linear algebra of the interior-point solution algorithm. Following this, we provide several illustrative numerical examples for both steady-state and dynamic optimization design problems. Finally, some concluding remarks are provided with some recommendations for further work.

4.2 Problem Formulation

In this section we consider multiperiod approximations to continuous nonlinear stochastic programs which are formulated as structured nonlinear programs [2, 19]. Our particular interest is the efficient solution of two-stage stochastic programming approximations which classify decision variables into first and second stage variables such that the first stage variables represent system design decisions while second stage variables represent scenario dependent system operation variables which are used for recourse to disturbance or uncertain parameter realizations. These particular formulations have seen significant use for chemical process design and operations applications under uncertainty [20, 21] and more recently in model-based robust control applications [22, 23]. The particular multiperiod nonlinear programming formulation we consider can be stated as,

$$\begin{aligned} \min_{\mathbf{w}_i \,\forall\, i,\, \mathbf{p}} \quad & \phi_0(\mathbf{p}) + \textstyle\sum_{i=1}^{n_s} w_i\, \phi_i(\mathbf{w}_i, \mathbf{p}) \\ \text{st}: \quad & \mathbf{c}_0(\mathbf{p}) = \mathbf{0} \\ & \mathbf{c}_i(\mathbf{w}_i, \mathbf{p}) = \mathbf{0} \quad \forall\, i = 1, \ldots, n_s \\ & \mathbf{w}_i \in W = \{\mathbf{w}_i \in \mathbb{R}^{n_w} \,|\, \mathbf{w}_L \leq \mathbf{w}_i \leq \mathbf{w}_U\} \\ & \mathbf{p} \in P = \{\mathbf{p} \in \mathbb{R}^{n_p} \,|\, \mathbf{p}_L \leq \mathbf{p} \leq \mathbf{p}_U\} \end{aligned} \tag{P.4.1}$$

where the vector $\mathbf{w}_i \in \mathbb{R}^{n_w}$ represents all second stage primal variables from a particular scenario/period realization with lower and upper bounds given by $\mathbf{w}_L$ and $\mathbf{w}_U$, respec-

tively; $\mathbf{p} \in \mathbb{R}^{n_p}$ represent all first stage primal design variables that are common over all scenario realizations, again with bounds given by $\mathbf{p}_L$ and $\mathbf{p}_U$; n_s represents the number of periods/scenarios used in the approximation. The objective function involves two nonlinear functional components, the first is $\phi_0 : \mathbb{R}^{n_p} \mapsto \mathbb{R}$ which represents a design cost that is only dependent on the design variables, while the second is $\phi_i : \mathbb{R}^{n_w} \times \mathbb{R}^{n_p} \mapsto \mathbb{R}$ which represents a weighted operational cost that is summed over all scenario realizations. The associated weight or probability for each scenario occurrence is given by $w_i \in [0, 1]$. Similarly, $\mathbf{c}_0 : \mathbb{R}^{n_p} \mapsto \mathbb{R}^{m_1}$ and $\mathbf{c}_i : \mathbb{R}^{n_w} \times \mathbb{R}^{n_p} \mapsto \mathbb{R}^{m_2}$ represent nonlinear constraints of a scenario-independent and scenario-dependent nature, respectively. Problem P.4.1 may include steady-state design problems, discretized dynamic optimization or optimal control formulations, and dynamic optimization formulations which include embedded dynamic models. This latter application class is of particular interest, as we seek to improve on current solution algorithms for dynamic optimization formulations which incorporate embedded dynamic models.

The first step to facilitating the application of a solution algorithm capable of exploiting the formulation structure is to re-formulate Problem P.4.1. The idea here is that we need to introduce a decomposable structure within the formulation that can be exploited in the NLP solution algorithm. An established approach is to decouple the common parameters $\mathbf{p} \in \mathbb{R}^{n_p}$ from the scenario-dependent objective and constraint functionals by introducing artificial decoupling parameters and linking constraints [5]. Accordingly, such a reformulation involves appending $\mathbf{w}_i$ with $\mathbf{q}_i \in \mathbb{R}^{n_q}$, which represent local design parameters within each scenario, followed by the mapping of these local variables back to the original global parameters $\mathbf{p}$. Note, the dimension n_q may be greater or equal to n_p, depending on the particular

application. Thus, a revised form of Problem P.4.1 can be stated as,

$$\begin{aligned} \min_{\mathbf{z}_i \,\forall\, i,\, \mathbf{p}} \quad & \phi_0(\mathbf{p}) + \textstyle\sum_{i=1}^{n_s} w_i\, \phi_i(\mathbf{z}_i) \\ \text{st}: \quad & \mathbf{c}_0(\mathbf{p}) = \mathbf{0} \\ & \mathbf{c}_i(\mathbf{z}_i) = \mathbf{0} \quad \forall\, i = 1, \ldots, n_s \\ & \mathbf{M}_i\, \mathbf{z}_i - \mathbf{p} = \mathbf{0} \quad \forall\, i = 1, \ldots, n_s \\ & \mathbf{F}_i\, \mathbf{z}_i - \bar{\mathbf{z}} \leq \mathbf{0} \quad \forall\, i = 1, \ldots, n_s \end{aligned} \tag{P.4.2}$$

where $\mathbf{z}_i := [\mathbf{w}_i^\top, \mathbf{q}_i^\top]^\top \in \mathbb{R}^{n_z}$, $n_z = n_w + n_q$, and $\mathbf{M}_i := [\mathbf{0}_{n_p \times n_w} | \mathbf{I}_{n_p} | \mathbf{0}_{n_p \times (n_q - n_p)}]$ represents a mapping matrix that extracts the local parameters $\mathbf{q}_i$ from $\mathbf{z}_i$. The variable bounds are written as inequalities by introducing the definitions $\mathbf{F}_i^\top := [-\mathbf{I}_{n_z}, \mathbf{I}_{n_z}]$ and $\bar{\mathbf{z}}^\top := [-\mathbf{z}_L^\top, \mathbf{z}_U^\top]$. For the case of dynamic optimization formulations, we consider a multiple-shooting discretization approach which involves the solution of an embedded system of ordinary differential equations (ODEs) for each shooting node. The embedded model takes the form,

$$\begin{aligned} \dot{\mathbf{x}}_i(t) &= \mathbf{f}(\mathbf{x}_i(t), \mathbf{u}_i(t), \mathbf{q}_i, t) \quad t \in [t_j, t_{j+1}] \\ \mathbf{x}_i(t_j) &= \mathbf{x}_{j,i} \end{aligned} \tag{4.1}$$

where $\mathbf{u}_i(t)$ represents a control profile parameterized by $\mathbf{u}_{j,i}$ and $\mathbf{f}(\cdot) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_q} \times \mathbb{R} \mapsto \mathbb{R}^{n_x}$ represents the explicit ODE model right-hand-side. Accordingly, the constraints $\mathbf{c}_i(\mathbf{z}_i)$ take the form,

$$\mathbf{c}_i(\mathbf{z}_i) := \begin{cases} \mathbf{x}_i(t_0) - \mathbf{x}_{0,i} \\ \mathbf{x}_i(\mathbf{x}_{j-1,i}, \mathbf{u}_{j-1,i}, \mathbf{q}_i; t_j) - \mathbf{x}_{j,i} \quad j = 1, \ldots, n \end{cases} \tag{4.2}$$

which represent continuity constraints at each shooting node, where $\mathbf{x}_i(t_0)$ are specified initial conditions of the ODE, $\mathbf{x}_i(t_j)$ represent the ODE solution at nodes $j = 1, \ldots, n$, $\mathbf{x}_{j,i}$ represent state variable parameters, $\mathbf{u}_{j,i}$ represent control variable parameters or degrees of freedom. These parameters relate back to the NLP by the structured definition, $\mathbf{w}_i^\top :=$

$[\mathbf{x}_{0,i}^\top, \mathbf{u}_{0,i}^\top, \ldots, \mathbf{x}_{n-1,i}^\top, \mathbf{u}_{n-1,i}^\top, \mathbf{x}_{n,i}^\top]$, which are further repeated for each scenario $i = 1, \ldots, n_s$. Additionally, $\mathbf{c}_i$ may also include other forms of point or path equality and inequality constraints. Introducing an embedded ODE within the constraints requires the application of appropriate parameter sensitivity and derivative generation techniques in order to supply the NLP algorithm with the necessary first-order and possibly second-order objective and constraint derivatives. Such techniques are widely available in modern ODE/DAE integration packages [24] and for the particular multiple-shooting discretization used in this chapter we refer readers to our previous work in this area which discusses the constraint derivative formulation in Chapter 3.

4.3 Proposed Solution Algorithm

In this section we present our proposed SQP algorithm that uses an interior-point method (IPM) for the QP subproblems capable of exploiting the formulation structure as defined by Problem P.4.2. Interior-point methods were originally designed to solve large linear programs due to the polynomial growth of their worst-case solution time with problem size, as opposed to the worst-case exponential growth characteristic of the more traditional active-set simplex algorithm [25]. Since their original inception, IPMs have been extended for quadratic (convex/nonconvex) and general nonlinear/nonconvex programs [26]. Our use here focuses on IPMs for QPs, as encountered within the SQP algorithm. The proposed algorithm is particularly amenable to large-scale NLP formulations with expensive function evaluations such as those commonly encountered when evaluating embedded differential equations within the formulation constraints. Many of the steps in our proposed approach follow those in existing SQP-IPM algorithms [9, 27, 28]; however, of particular distinction here is the use of an explicit Schur-complement decomposition within the IPM step direction solution which exploits the multiperiod formulation structure [7, 29]. It is this last aspect that we seek to investigate in the context of embedded model dynamic optimization formulations.

4.3.1 SQP Approach

To start, we outline the basic aspects associated with an SQP solution approach to NLP formulations, focusing on implementation choices that we have adopted. For generality, we consider an NLP written according to,

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{st}: \quad & \mathbf{h}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{g}(\mathbf{x}) \leq \mathbf{0} \end{aligned} \tag{P.4.3}$$

where $\mathbf{x} \in \mathbb{R}^n$ represent primal variables and $f : \mathbb{R}^n \mapsto \mathbb{R}$, $\mathbf{h} : \mathbb{R}^n \mapsto \mathbb{R}^{m_1}$, and $\mathbf{g} : \mathbb{R}^n \mapsto \mathbb{R}^{m_2}$ represent nonlinear, possibly nonconvex, twice continuously differentiable functions. Note that bound constraints on $\mathbf{x}$ can be concatenated into $\mathbf{g}$. An optimal solution to Problem P.4.3 can be characterized using the Lagrangian function defined as,

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) := f(\mathbf{x}) + \boldsymbol{\lambda}^\top \mathbf{h}(\mathbf{x}) + \boldsymbol{\mu}^\top \mathbf{g}(\mathbf{x}) \tag{4.3}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{m_1}$ and $\boldsymbol{\mu} \in \mathbb{R}^{m_2}$ represent Lagrange multipliers or dual variables which provide a degree of optimal cost sensitivity with respect to perturbations of constraints within the active-set. Candidate primal-dual solutions $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ must satisfy the first-order necessary optimality conditions (KKT conditions) defined as,

$$\begin{aligned} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) &= \mathbf{0} \\ \mathbf{h}(\mathbf{x}^*) &= \mathbf{0} \\ \mathbf{g}(\mathbf{x}^*) &\leq \mathbf{0} \\ \boldsymbol{\mu}^{*\top} \mathbf{g}(\mathbf{x}^*) &= 0 \\ \boldsymbol{\mu}^* &\geq \mathbf{0} \end{aligned} \tag{KKT}$$

where the gradients of all active constraints at $\mathbf{x}^*$ are required to be linearly independent. In addition to the first-order optimality conditions, further verification is needed to sufficiently assess whether the KKT point is a true minimizer of Problem P.4.3 and not a possible maximizer. Accordingly, second-order sufficient optimality conditions can be defined which require the Lagrangian Hessian to be strictly positive definite at the optimal solution. Due to the expensive nature of computing accurate second-order derivatives, many numerical algorithms do not check for second-order sufficient conditions of optimality. Instead many algorithms (and SQP in particular) try to build in positive definite approximations, thus implicitly ensuring such conditions while searching for a KKT point. Nevertheless, some SQP algorithms are capable of utilizing second-order information directly, and as such provide appropriate steps to check and correct second order derivatives [27].

The SQP approach to determining a KKT point successively solves quadratic programming approximations to Problem P.4.3, where for each succession a search direction $\mathbf{d}$ is determined that satisfies the QP optimality conditions [26], followed by determining an appropriate step size, and subsequent iterate update and re-evaluation of the QP approximation until the desired termination criteria are satisfied. At a given primal-dual iterate $(\mathbf{x}^{[k]}, \boldsymbol{\lambda}^{[k]}, \boldsymbol{\mu}^{[k]})$, a linearly constrained quadratic approximation to Problem P.4.3 can be stated as,

$$\begin{aligned} \min_{\mathbf{d}} \quad & \nabla_x f(\mathbf{x}^{[k]})^\top \mathbf{d} + \tfrac{1}{2}\, \mathbf{d}^\top \nabla^2_{xx} \mathcal{L}(\mathbf{x}^{[k]}, \boldsymbol{\lambda}^{[k]}, \boldsymbol{\mu}^{[k]})\, \mathbf{d} \\ \text{st}: \quad & \nabla_x \mathbf{h}(\mathbf{x}^{[k]})\, \mathbf{d} + \mathbf{h}(\mathbf{x}^{[k]}) = \mathbf{0} \\ & \nabla_x \mathbf{g}(\mathbf{x}^{[k]})\, \mathbf{d} + \mathbf{g}(\mathbf{x}^{[k]}) \leq \mathbf{0} \end{aligned} \tag{P.4.4}$$

where the QP objective function can be derived based on a second-order Taylor series expansion of $\mathcal{L}$ around $\mathbf{d}$ in which constant terms are dropped and second-order terms could be computed as $\nabla^2_{xx}\mathcal{L}(\mathbf{x}^{[k]}, \boldsymbol{\lambda}^{[k]}, \boldsymbol{\mu}^{[k]}) := \nabla^2_{xx} f(\mathbf{x}^{[k]}) + \sum_{j=1}^{m_1} \nabla^2_{xx} h_j(\mathbf{x}^{[k]})\lambda_j + \sum_{j=1}^{m_2} \nabla^2_{xx} g_j(\mathbf{x}^{[k]})\mu_j$. However, most SQP algorithms rely on generating convex approximations or appropriate regularizations to the exact Lagrangian Hessian $\nabla^2_{xx}\mathcal{L}$ which ensure positive definiteness, such that convex quadratic programming algorithms can be employed. Furthermore, a direct

solution to Problem P.4.4 may not be possible due to inconsistencies within the linearized constraint approximation which causes an infeasible solution. Accordingly, constraint relaxation techniques can be used to restore feasibility [27, 30].

Once the step direction is known, a suitable step size α must be determined which is able to drive the algorithm to a local solution. The particular method we use is an inexact line search with an ℓ_1-penalty function, which seeks to approximate,

$$\alpha^{[k]} := \underset{\bar{\alpha}\in(0,1]}{\arg\min}\, \phi(\mathbf{x}^{[k]} + \bar{\alpha}\,\mathbf{d}, \boldsymbol{\eta}^{[k]}) \tag{4.4}$$

where the penalty function is defined as,

$$\phi(\mathbf{x}^{[k+1]}, \boldsymbol{\eta}^{[k]}) := f(\mathbf{x}^{[k+1]}) + \sum_{i=1}^{m_1} \eta_i^{[k]} |\mathbf{h}_i(\mathbf{x}^{[k+1]})| + \sum_{i=1}^{m_2} \eta_{m_1+i}^{[k]} \max\{0, \mathbf{g}_i(\mathbf{x}^{[k+1]})\} \tag{4.5}$$

where $\mathbf{x}^{[k+1]} := \mathbf{x}^{[k]} + \bar{\alpha}\,\mathbf{d}$ and $\boldsymbol{\eta}^{[k]}$ is a penalty vector for constraint violation that is updated at each major SQP iteration (see, pg. 542 and Equation 18.36 of [26] for further details). The exact solution of Equation 4.4 is unnecessary and it is sufficient only to satisfy the Armijo condition given by,

$$\phi(\bar{\alpha}) \leq \phi(0) + \sigma\bar{\alpha}D(0) \tag{4.6}$$

where σ is a small positive constant and $D(0)$ represents the directional derivative of $\phi(0)$ (see, pg 541 and Equation 18.31 from [26]). Now, using the condition in Equation 4.6, defined as $\delta := \phi(0) + \sigma\bar{\alpha}D(0) - \phi(\bar{\alpha})$, and an initial step size guess of $\bar{\alpha} = 1$, the step size is accepted if $\delta > 0$ and modified otherwise. The update model we choose follows a quadratic interpolation (see pg. 58 and Equation 3.58 of [26] for further details) and can be defined as,

$$\bar{\alpha} \leftarrow \frac{0.5\,\bar{\alpha}^2 D(0)}{\phi(0) + \bar{\alpha}D(0) - \phi(\bar{\alpha})} \tag{4.7}$$

Once both step direction and size are known, both primal and dual variables are updated as,

$$\mathbf{x}^{[k+1]} \leftarrow \mathbf{x}^{[k]} + \alpha^{[k]}\,\mathbf{d} \tag{4.8}$$

$$\{\boldsymbol{\lambda}^{[k+1]}, \boldsymbol{\mu}^{[k+1]}\} \leftarrow \{\boldsymbol{\lambda}^{[k]}, \boldsymbol{\mu}^{[k]}\} + \alpha^{[k]}(\{\mathbf{y}, \mathbf{v}\} - \{\boldsymbol{\lambda}^{[k]}, \boldsymbol{\mu}^{[k]}\}) \tag{4.9}$$

where $\mathbf{y}$ and $\mathbf{v}$ represent multipliers returned from the QP solver. The process of forming the QP and determining the step direction and size continues until the KKT conditions are satisfied to within a specified tolerance. A set of particular termination criteria that we adopt are defined as,

$$\|\nabla_x \mathcal{L}(\mathbf{x}^{[k]}, \boldsymbol{\lambda}^{[k]}, \boldsymbol{\mu}^{[k]})\| \leq \epsilon_{\text{opt}} \tag{4.10}$$

$$\|\mathbf{h}(\mathbf{x}^{[k]})\| \leq \epsilon_{\text{feas}} \tag{4.11}$$

$$\boldsymbol{\mu}_i^{[k]} \mathbf{g}_i(\mathbf{x}^{[k]}) \leq \epsilon_{\text{comp}} \quad i = 1, \ldots, m_2 \tag{4.12}$$

where ϵ_{opt}, ϵ_{feas}, ϵ_{comp} are user specified tolerances for optimality, feasibility and complementarity, respectively.

In order to set the stage for approximating the second-order Lagrangian derivative posed in Problem P.4.4, with reference to Problem P.4.2, we note that primal variables can be stated as $\mathbf{x} := [\mathbf{z}_1^\top, \ldots, \mathbf{z}_{n_s}^\top, \mathbf{p}^\top]^\top$ and define the multiperiod Lagrangian function as,

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) := \mathcal{L}_0(\mathbf{p}, \boldsymbol{\lambda}_0) + \sum_{i=1}^{n_s} \mathcal{L}_i(\mathbf{z}_i, \mathbf{p}, \boldsymbol{\lambda}_i, \boldsymbol{\mu}_i) \tag{4.13}$$

where the partially separable Lagrangian components are further defined as,

$$\mathcal{L}_0(\mathbf{p}, \boldsymbol{\lambda}_0) := \phi_0(\mathbf{p}) + \boldsymbol{\lambda}_0^\top \mathbf{c}_0(\mathbf{p}) \tag{4.14}$$

$$\mathcal{L}_i(\mathbf{z}_i, \mathbf{p}, \boldsymbol{\lambda}_i, \boldsymbol{\mu}_i) := w_i \phi_i(\mathbf{z}_i) + \boldsymbol{\lambda}_i^\top \begin{bmatrix} \mathbf{c}_i(\mathbf{z}_i) \\ \mathbf{M}_i \mathbf{z}_i - \mathbf{p} \end{bmatrix} + \boldsymbol{\mu}_i^\top [\mathbf{F}_i \mathbf{z}_i - \bar{\mathbf{z}}] \tag{4.15}$$

Considering this Lagrangian form, the approach adopted to approximate the Lagrangian

Hessian follows the well known damped BFGS method. However, given the form of Equation 4.13, our particular approach approximates each separate Hessian block $\mathbf{H}_0 \approx \nabla^2_{pp}\mathcal{L}_0$ and $\mathbf{H}_i \approx \nabla^2_{z_i z_i}\mathcal{L}_i$ for $i = 1, \ldots, n_s$ as,

$$\nabla^2_{xx}\mathcal{L} := \mathbf{H} \approx \operatorname{diag}\left(\mathbf{H}_1, \ldots, \mathbf{H}_{n_s}, \mathbf{H}_0\right) \tag{4.16}$$

which corresponds to a partitioned quasi-Newton update. Due to the introduction of the linear linking constraints in Problem P.4.2, this second-order update approximation can be considered completely separable and as such each block component is updated independently [7, 26, 31]. The update for each block follows the standard BFGS formula defined as,

$$\mathbf{H}_i^{[k]} \leftarrow \mathbf{H}_i^{[k]} - \frac{\mathbf{H}_i^{[k]}\mathbf{s}_i\,\mathbf{s}_i^\top\mathbf{H}_i^{[k]}}{\mathbf{s}_i^\top\mathbf{H}_i^{[k]}\mathbf{s}_i} + \frac{\mathbf{r}_i\,\mathbf{r}_i^\top}{\mathbf{r}_i^\top\mathbf{s}_i} \quad i = 0, \ldots, n_s \tag{4.17}$$

where the vectors $\mathbf{s}_i \in \mathbb{R}^{n_z}$ and $\mathbf{r}_i \in \mathbb{R}^{n_z}$ are defined as,

$$\mathbf{s}_i := \mathbf{z}_i^{[k+1]} - \mathbf{z}_i^{[k]} \tag{4.18}$$

$$\mathbf{r}_i := \nabla_{z_i}\mathcal{L}_i(\mathbf{z}_i^{[k+1]}, \boldsymbol{\lambda}_i^{[k+1]}, \boldsymbol{\mu}_i^{[k+1]}) - \nabla_{z_i}\mathcal{L}_i(\mathbf{z}_i^{[k]}, \boldsymbol{\lambda}_i^{[k+1]}, \boldsymbol{\mu}_i^{[k+1]}) \tag{4.19}$$

where $\mathbf{z}_i$, $\boldsymbol{\lambda}_i$, $\boldsymbol{\mu}_i$ represent local primal and dual variables for $i = 1, \ldots, n_s$ blocks. The bottom diagonal block, comprised solely of the complicating common parameters $\mathbf{p}$, as seen from $\mathbf{c}_0(\mathbf{p})$ within Problem P.4.2, is updated using,

$$\mathbf{s}_0 := \mathbf{p}^{[k+1]} - \mathbf{p}^{[k]} \tag{4.20}$$

$$\mathbf{r}_0 := \nabla_p\mathcal{L}_0(\mathbf{p}^{[k+1]}, \boldsymbol{\lambda}_0^{[k+1]}) - \nabla_p\mathcal{L}_0(\mathbf{p}^{[k]}, \boldsymbol{\lambda}_0^{[k+1]}) \tag{4.21}$$

where $\boldsymbol{\lambda}_0$ represents dual variables for $\mathbf{c}_0(\mathbf{p})$. The vector $\mathbf{r}_i$ is modified if $\mathbf{s}_i^\top\mathbf{r}_i < 0.2\,\mathbf{s}_i^\top\mathbf{H}_i^{[k]}\mathbf{s}_i$ using a damping parameter defined as,

$$\theta := 0.8\,\mathbf{s}_i^\top\mathbf{H}_i^{[k]}\mathbf{s}_i / (\mathbf{s}_i^\top\mathbf{H}_i^{[k]}\mathbf{s}_i - \mathbf{s}_i^\top\mathbf{r}_i) \tag{4.22}$$

where the final modification, if necessary, is performed as,

$$\mathbf{r}_i \leftarrow \theta\, \mathbf{r}_i + (1-\theta)\mathbf{H}_i^{[k]}\mathbf{s}_i \tag{4.23}$$

This approach, as highlighted in Equations 4.17 to 4.23 (which we adapted herein from Procedure 18.2 of [26]), is sufficient for block sizes that are relatively small (perhaps less than a 1000 variables). However, as the blocks $\mathbf{H}_i$ become large (e.g., with finer levels of discretization within the multiple-shooting approach), more compact BFGS representations are preferred [32].

4.3.2 IPM QP Approach

Interior-point methods have been shown highly effective at solving large-scale mathematical programming formulations with a relatively insensitive increase in iteration count as the problem size increases [9]. Traditionally, the solution of the quadratic program given by Problem P.4.4 is handled by an active-set strategy [26]. In this approach, all equalities and some inequality constraints are used to establish a working active-set of constraints, where a choice is made a priori of an active inequality (i.e., primal slack variable of zero) or inactive inequality (i.e., dual variable of zero). Using the current working-set, a Newton direction for the QP is determined and using constraint gradient and dual variable values a new working set is established (see, Algorithm 16.3 from [26]). Interior-point methods, on the other hand, function by shifting the bound constraints into the primal objective function to penalize primal slack variables and ensuring they remain strictly positive. From another perspective, one can view this approach as perturbing the complementarity constraints by a factor of μ, followed by determining a Newton direction using a fixed μ, which is subsequently repeated for $\mu \rightarrow 0$ and terminated once the optimality conditions are satisfied. Using this approach avoids the combinatorial active-set approach of selecting a priori the active and inactive inequality constraints, and as the interior-point algorithm proceeds the partitioning of primal

slack and dual variables into zero and nonzero elements is gradually revealed.

It is this second approach we utilize in this work to solve the QP subproblems using a third-party software package. However, we sketch out the approach to show how our formulation is arranged to fit within the structure that can be exploited by the QP decomposition approach. Accordingly, such QP formulations can be written as,

$$\begin{aligned} \min_{\mathbf{d},\,\mathbf{s}} \;& \mathbf{c}_k^\top \mathbf{d} + \tfrac{1}{2}\,\mathbf{d}^\top \mathbf{Q}_k\,\mathbf{d} \\ \text{st}: \;& \mathbf{A}_k\,\mathbf{d} - \mathbf{b}_k = \mathbf{0} \\ & \mathbf{C}_k\,\mathbf{d} - \mathbf{n}_k + \mathbf{s} = \mathbf{0} \\ & \mathbf{s} \geq \mathbf{0} \end{aligned} \tag{P.4.5}$$

where we re-write the inequality constrained formulation into standard QP form through the introduction of nonnegative slack variables $\mathbf{s} \in \mathbb{R}^{m_2}$. For convenience, we relate this formulation back to Problem P.4.4 using the following definitions $\mathbf{c}_k := \nabla_x f(\mathbf{x}^{[k]}) \in \mathbb{R}^n$, $\mathbf{b}_k := -\mathbf{h}(\mathbf{x}^{[k]}) \in \mathbb{R}^{m_1}$, $\mathbf{n}_k := -\mathbf{g}(\mathbf{x}^{[k]}) \in \mathbb{R}^{m_2}$, $\mathbf{A}_k := \nabla_x \mathbf{h}(\mathbf{x}^{[k]}) \in \mathbb{R}^{m_1 \times n}$, $\mathbf{C}_k := \nabla_x \mathbf{g}(\mathbf{x}^{[k]}) \in \mathbb{R}^{m_2 \times n}$, $\mathbf{Q}_k := \nabla^2_{xx} \mathcal{L}(\mathbf{x}^{[k]}, \boldsymbol{\lambda}^{[k]}, \boldsymbol{\mu}^{[k]}) \in \mathbb{R}^{n \times n}$, where $\mathbf{Q}_k$ is at least positive semi-definite.

A central aspect of interior-point methods is the reformulation of primal variable bounds by adding them to the objective function through the use of a log-barrier term with a positive penalty parameter $\mu > 0$. Accordingly, we can re-write Problem P.4.5 as,

$$\begin{aligned} \min_{\mathbf{d},\,\mathbf{s}} \;& \mathbf{c}_k^\top \mathbf{d} + \tfrac{1}{2}\,\mathbf{d}^\top \mathbf{Q}_k\,\mathbf{d} - \mu \textstyle\sum_{j=1}^{m_2} \log\, s_j \\ \text{st}: \;& \mathbf{A}_k\,\mathbf{d} - \mathbf{b}_k = \mathbf{0} \\ & \mathbf{C}_k\,\mathbf{d} - \mathbf{n}_k + \mathbf{s} = \mathbf{0} \end{aligned} \tag{P.4.6}$$

where μ can be seen to control the relation between the barrier formulation and the original QP problem. The first-order QP optimality conditions of the barrier formulation given by

Problem P.4.6 can now be derived (see, [27, 33]) as,

$$\begin{aligned} \mathbf{c}_k + \mathbf{Q}_k\,\mathbf{d} + \mathbf{A}_k^\top \mathbf{y} + \mathbf{C}_k^\top \mathbf{v} &= \mathbf{0} \\ \mathbf{A}_k\,\mathbf{d} - \mathbf{b}_k &= \mathbf{0} \\ \mathbf{C}_k\,\mathbf{d} + \mathbf{s} - \mathbf{n}_k &= \mathbf{0} \\ \mathbf{S}\,\mathbf{V}\,\mathbf{e} - \mu\mathbf{e} &= \mathbf{0} \end{aligned} \tag{qpKKT}$$

where $\mathbf{y} \in \mathbb{R}^{m_1}$, $\mathbf{v} \in \mathbb{R}^{m_2}$ represent the QP Lagrange multipliers, and the last equation represents the perturbed complementarity condition, which implies that at least one of the two variables, $\mathbf{s}$ or $\mathbf{v}$, have to be zero at the optimum where $\mu = 0$. $\{\mathbf{S}, \mathbf{V}\} = \mathrm{diag}(\{\mathbf{s}, \mathbf{v}\})$ are diagonal matrices and $\mathbf{e} = [1, \ldots, 1]^\top$ is a unit vector. The interior-point solution of Problem P.4.5 involves a central path following algorithm which keeps the iterate $(\mathbf{d}^{[j]}, \mathbf{s}^{[j]}, \mathbf{y}^{[j]}, \mathbf{v}^{[j]})$ biased towards the interior of the feasible region $(\mathbf{s}, \mathbf{v}) > \mathbf{0}$ [25, 34]. The Newton step of the *j*th iterate in the QP solution could be determined from the solution of the following linear system,

$$\begin{bmatrix} \mathbf{Q}_k & \mathbf{A}_k^\top & \mathbf{C}_k^\top & \mathbf{0} \\ \mathbf{A}_k & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{C}_k & \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{S}^{[j]} & \mathbf{V}^{[j]} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{d} \\ \Delta\mathbf{y} \\ \Delta\mathbf{v} \\ \Delta\mathbf{s} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \\ \mathbf{r}_4 \end{bmatrix} = \begin{bmatrix} -\mathbf{c}_k - \mathbf{Q}_k\mathbf{d}^{[j]} - \mathbf{A}_k^\top\mathbf{y}^{[j]} - \mathbf{C}_k^\top\mathbf{v}^{[j]} \\ -\mathbf{A}_k\mathbf{d}^{[j]} + \mathbf{b}_k \\ -\mathbf{C}_k\mathbf{d}^{[j]} - \mathbf{s}^{[j]} + \mathbf{n}_k \\ -\mathbf{S}^{[j]}\mathbf{V}^{[j]}\mathbf{e} + \mu\mathbf{e} \end{bmatrix} \tag{4.24}$$

However, rather than solving this non-symmetric and indefinite system directly it is more amenable to transform this system into the augmented form given by,

$$\begin{bmatrix} \mathbf{Q}_k & \mathbf{A}_k^\top & \mathbf{C}_k^\top \\ \mathbf{A}_k & \mathbf{0} & \mathbf{0} \\ \mathbf{C}_k & \mathbf{0} & -(\mathbf{V}^{[j]})^{-1}\mathbf{S}^{[j]} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{d} \\ \Delta\mathbf{y} \\ \Delta\mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 - (\mathbf{V}^{[j]})^{-1}\mathbf{r}_4 \end{bmatrix} \tag{4.25}$$

where through some algebraic manipulation we eliminate $\Delta\mathbf{s}$, which can be recovered as $\Delta\mathbf{s} := (\mathbf{V}^{[j]})^{-1}(\mathbf{r}_4 - \mathbf{S}^{[j]}\Delta\mathbf{v})$ [7]. Further elimination of $\Delta\mathbf{v}$ produces the better known

augmented form given by,

$$\underbrace{\begin{bmatrix} \mathbf{Q}_k + \mathbf{G}_k^{[j]} & \mathbf{A}_k^\top \\ \mathbf{A}_k & \mathbf{0} \end{bmatrix}}_{\mathbf{\Phi}} \begin{bmatrix} \Delta \mathbf{d} \\ \Delta \mathbf{y} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{r}}_1 \\ \mathbf{r}_2 \end{bmatrix} \tag{4.26}$$

where $\Delta \mathbf{v}$ is recovered as $\Delta \mathbf{v} := -\mathbf{V}^{[j]}(\mathbf{S}^{[j]})^{-1}(\mathbf{r}_3 - (\mathbf{V}^{[j]})^{-1}\mathbf{r}_4 - \mathbf{C}_k \Delta \mathbf{d})$ and the diagonal matrix $\mathbf{G}_k^{[j]}$ and the modified right-hand-side vector $\bar{\mathbf{r}}_1$ are defined as,

$$\mathbf{G}_k^{[j]} := \mathbf{C}_k^\top \mathbf{V}^{[j]}(\mathbf{S}^{[j]})^{-1}\mathbf{C}_k \tag{4.27}$$

$$\bar{\mathbf{r}}_1 := \mathbf{r}_1 + \mathbf{C}_k^\top \mathbf{V}^{[j]}(\mathbf{S}^{[j]})^{-1}(\mathbf{r}_3 - (\mathbf{V}^{[j]})^{-1}\mathbf{r}_4) \tag{4.28}$$

The symmetric indefinite linear system given by Equation 4.26 can be directly solved by a number of direct factorization or iterative methods. However, possible rank deficiency of $\mathbf{A}_k$ or near singularity in $\mathbf{Q}_k + \mathbf{G}_k^{[j]}$ can present problems. Thus, adaptive primal-dual regularization methods are typically used to ensure the matrix $\mathbf{\Phi}$ is at least symmetric quasi-definite which is known to be strongly factorizable [35, 36].

4.3.3 Decomposition within IPM

The formulation of Problem P.4.2 aims to facilitate the decoupling of the common design parameters $\mathbf{p}$ from each scenario/period realization. Considering the QP subproblem of Problem P.4.5 and the augmented system given by Equation 4.26, our particular multiperiod formulation produces several block diagonal and block bordered diagonal matrices given by,

$$\mathbf{A}_k := \begin{bmatrix} \mathbf{D}_1 & & & \mathbf{E}_1 \\ & \ddots & & \vdots \\ & & \mathbf{D}_{n_s} & \mathbf{E}_{n_s} \\ & & & \mathbf{D}_0 \end{bmatrix}, \quad \begin{aligned} \mathbf{Q}_k &:= \operatorname{diag}(\bar{\mathbf{Q}}_1, \ldots, \bar{\mathbf{Q}}_{n_s}, \bar{\mathbf{Q}}_0) \\ \mathbf{G}_k^{[j]} &:= \operatorname{diag}(\bar{\mathbf{G}}_1^{[j]}, \ldots, \bar{\mathbf{G}}_{n_s}^{[j]}, \mathbf{0}_{n_p \times n_p}) \\ \mathbf{C}_k &:= \operatorname{diag}(\mathbf{F}_1, \ldots, \mathbf{F}_{n_s}) \end{aligned} \tag{4.29}$$

Note, we describe this decomposition by considering that a single scenario realization maps directly to a single block of equations/variables within the Jacobian/Hessian matrices. However, one could further group multiple scenario realizations into a single block which is often required in order to map each block to a limited number of processors. We will elaborate this point further in the example problems of this chapter. The constituent matrix blocks in $\mathbf{A}_k$ are diagonal blocks $\mathbf{D}_i$ and right-border blocks $\mathbf{E}_i$ for each scenario/period $i = 1, \ldots, n_s$, and bottom diagonal block $\mathbf{D}_0$ for constraints involving only the complicating parameters $\mathbf{p}$. These block definitions relate back to the QP approximation of Problem P.4.2 through,

$$\mathbf{D}_i := \begin{bmatrix} \nabla_{z_i} \mathbf{c}_i(\mathbf{z}_i^{[k]}) \\ \mathbf{M}_i \end{bmatrix}, \ \mathbf{E}_i := \begin{bmatrix} \mathbf{0}_{n_z \times n_p} \\ -\mathbf{I}_{n_p} \end{bmatrix}, \ \mathbf{D}_0 := \nabla_p \mathbf{c}_0(\mathbf{p}^{[k]}) \tag{4.30}$$

Similarly for $\mathbf{Q}_k$, we have diagonal blocks representing Hessian contributions for each scenario, given by $\bar{\mathbf{Q}}_i := \nabla^2_{z_i z_i} \mathcal{L}_i(\mathbf{z}_i^{[k]}, \boldsymbol{\lambda}_i^{[k]}, \boldsymbol{\mu}_i^{[k]}) \approx \mathbf{H}_i$ and a single bottom diagonal block $\bar{\mathbf{Q}}_0 := \nabla^2_{pp} \mathcal{L}_0(\mathbf{p}^{[k]}, \boldsymbol{\lambda}_0^{[k]}) \approx \mathbf{H}_0$ for the complicating parameters, where the $\mathbf{H}_i$ represent BFGS approximations. For the augmented system block $\mathbf{G}_k^{[j]}$, we have constituent sub-blocks given by $\bar{\mathbf{G}}_i^{[j]} := \mathbf{F}_i^\top \mathbf{V}_i^{[j]} (\mathbf{S}_i^{[j]})^{-1} \mathbf{F}_i$, with $\mathbf{F}_i$ defined as in Problem P.4.2.The remaining vectors in Equation 4.26 are determined through the specification of $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{r}_3$ and $\mathbf{r}_4$, which in terms of the functions and variables of our multiperiod optimization problem P.4.5 may be expressed as,

$$\begin{aligned} \mathbf{r}_1 &:= [\mathbf{r}_1^{d\top}, \ldots, \mathbf{r}_{n_s}^{d\top}, \mathbf{r}_0^{d\top}]^\top \\ \mathbf{r}_2 &:= [\mathbf{r}_1^{y\top}, \ldots, \mathbf{r}_{n_s}^{y\top}, \mathbf{r}_0^{y\top}]^\top \\ \mathbf{r}_3 &:= [\mathbf{r}_1^{v\top}, \ldots, \mathbf{r}_{n_s}^{v\top}]^\top \\ \mathbf{r}_4 &:= [\mathbf{r}_1^{s\top}, \ldots, \mathbf{r}_{n_s}^{s\top}]^\top \end{aligned} \tag{4.31}$$

with subvectors defined as,

$$\begin{aligned}
\mathbf{r}_i^d &:= -\bar{\mathbf{c}}_i - \bar{\mathbf{Q}}_i\,\mathbf{d}_i^{[j]} - \mathbf{D}_i^\top \mathbf{y}_i^{[j]} - \mathbf{F}_i^\top \mathbf{v}_i^{[j]} \\
\mathbf{r}_0^d &:= -\bar{\mathbf{c}}_0 - \bar{\mathbf{Q}}_0\,\mathbf{d}_0^{[j]} - \mathbf{D}_0^\top \mathbf{y}_0^{[j]} - \textstyle\sum_{l=1}^{n_s} \mathbf{E}_l^\top \mathbf{y}_l^{[j]} \\
\mathbf{r}_i^y &:= -\mathbf{D}_i\,\mathbf{d}_i^{[j]} - \mathbf{E}_i\,\mathbf{d}_0^{[j]} + \bar{\mathbf{b}}_i \\
\mathbf{r}_0^y &:= -\mathbf{D}_0\,\mathbf{d}_0^{[j]} + \bar{\mathbf{b}}_0 \\
\mathbf{r}_i^v &:= -\mathbf{F}_i\,\mathbf{d}_i^{[j]} - \mathbf{s}_i^{[j]} + \bar{\mathbf{n}}_i \\
\mathbf{r}_i^s &:= -\mathbf{S}_i^{[j]}\mathbf{V}_i^{[j]}\mathbf{e}_i + \mu \mathbf{e}_i
\end{aligned} \tag{4.32}$$

where,

$$\begin{aligned}
\bar{\mathbf{c}}_i &:= w_i \nabla_{z_i} \phi(\mathbf{z}_i^{[k]}) \\
\bar{\mathbf{c}}_0 &:= \nabla_p \phi(\mathbf{p}^{[k]}) \\
\bar{\mathbf{b}}_i &:= [-\mathbf{c}_i(\mathbf{z}_i^{[k]})^\top, -(\mathbf{M}_i\,\mathbf{z}_i^{[k]} - \mathbf{p}^{[k]})^\top]^\top \\
\bar{\mathbf{b}}_0 &:= -\mathbf{c}_0(\mathbf{p}^{[k]}) \\
\bar{\mathbf{n}}_i &:= -(\mathbf{F}_i \mathbf{z}_i^{[k]} - \bar{\mathbf{z}})
\end{aligned} \tag{4.33}$$

We further note that $\mathbf{c}_k$, $\mathbf{b}_k$ and $\mathbf{n}_k$ in Problem P.4.5 are defined as,

$$\begin{aligned}
\mathbf{c}_k &:= [\bar{\mathbf{c}}_1^\top, \ldots, \bar{\mathbf{c}}_{n_s}^\top, \bar{\mathbf{c}}_0^\top]^\top \\
\mathbf{b}_k &:= [\bar{\mathbf{b}}_1^\top, \ldots, \bar{\mathbf{b}}_{n_s}^\top, \bar{\mathbf{b}}_0^\top]^\top \\
\mathbf{n}_k &:= [\bar{\mathbf{n}}_1^\top, \ldots, \bar{\mathbf{n}}_{n_s}^\top]^\top
\end{aligned} \tag{4.34}$$

with $\bar{\mathbf{c}}_i$, $\bar{\mathbf{c}}_0$, $\bar{\mathbf{b}}_i$, $\bar{\mathbf{b}}_0$ and $\bar{\mathbf{n}}_i$ defined as above in Equation 4.33. Note that within these vector definitions, k represents the SQP iterate, j represents the QP iterate and i represents the particular period within the multiperiod formulation.

The solution efficiency of the QP subproblem, based on the multiperiod formulation of Problem P.4.2, can be significantly improved through exploiting the formulation structure given in Equation 4.29. This advantage is primarily achieved through the particular solution approach used to solve the augmented system given by Equation 4.26. One approach that

has shown great promise for solving the augmented linear system is the explicit Schur-complement decomposition, which has seen considerable application in the context large-scale multiperiod uncertain design, multi-data set parameter estimation, and discretized dynamic optimization formulations [10, 11]. This technique is able to break up the direct factorization of $\mathbf{\Phi}$ into several smaller independent steps, based on the permutation $\mathbf{\Phi} \mapsto \mathbf{P\Phi P}^{-1}$, and thus facilitate the parallel solution of each subsystem and further reduce overall memory requirements [14, 29]. The permuted system (equivalent to Equation 4.26) is defined according to the double bordered diagonal structure given by Equation 4.35,

$$\underbrace{\begin{bmatrix} \mathbf{\Phi}_1 & & & \mathbf{B}_1 \\ & \ddots & & \vdots \\ & & \mathbf{\Phi}_{n_s} & \mathbf{B}_{n_s} \\ \mathbf{B}_1^\top & \cdots & \mathbf{B}_{n_s}^\top & \mathbf{\Phi}_0 \end{bmatrix}}_{\mathbf{P\Phi P}^{-1}} \begin{bmatrix} \Delta\mathbf{w}_1 \\ \vdots \\ \Delta\mathbf{w}_{n_s} \\ \Delta\mathbf{w}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_{n_s} \\ \mathbf{r}_0 \end{bmatrix} \tag{4.35}$$

where $\Delta\mathbf{w}_i := [\Delta\mathbf{d}_i^\top, \Delta\mathbf{y}_i^\top]^\top$ represent the desired QP Newton direction for $i = 0, \ldots, n_s$, with corresponding right-hand-side vectors given by,

$$\mathbf{r}_i := [(\mathbf{r}_i^d + \mathbf{F}_i^\top \mathbf{V}_i^{[j]} (\mathbf{S}_i^{[j]})^{-1} (\mathbf{r}_i^v - (\mathbf{V}_i^{[j]})^{-1} \mathbf{r}_i^s))^\top, \mathbf{r}_i^{y\top}]^\top \tag{4.36}$$

$$\mathbf{r}_0 := [\mathbf{r}_0^{d\top}, \mathbf{r}_0^{y\top}]^\top \tag{4.37}$$

The remaining matrix blocks are defined as,

$$\mathbf{\Phi}_i := \begin{bmatrix} \bar{\mathbf{Q}}_i + \bar{\mathbf{G}}_i^{[j]} & \mathbf{D}_i^\top \\ \mathbf{D}_i & \mathbf{0} \end{bmatrix}, \ \mathbf{\Phi}_0 := \begin{bmatrix} \bar{\mathbf{Q}}_0 & \mathbf{D}_0^\top \\ \mathbf{D}_0 & \mathbf{0} \end{bmatrix}, \ \mathbf{B}_i^\top := \begin{bmatrix} \mathbf{0} & \mathbf{E}_i^\top \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{4.38}$$

Note that the permutation matrix $\mathbf{P}$ is used to illustrate the transformation of Equation 4.26 to Equation 4.35, and it is not explicitly defined or used within the linear algebra of the transformation. Instead, within the algorithm implementation, an appropriate mapping is performed to decompose the matrices in Equation 4.29 to those shown in Equation 4.38.

The solution of this linear system proceeds in several steps [14, 37]. The first and most computationally demanding step is to form the Schur-complement defined according to,

$$\mathbf{C} := \mathbf{\Phi}_0 - \sum_{i=1}^{n_s} \mathbf{B}_i^\top \mathbf{\Phi}_i^{-1} \mathbf{B}_i \tag{4.39}$$

which involves several smaller steps based on the individual contributions from $\mathbf{B}_i^\top \mathbf{\Phi}_i^{-1} \mathbf{B}_i$, each of which can be computed in parallel. Following this, the Schur linear system is solved in serial for $\Delta \mathbf{w}_0$ according to,

$$\mathbf{C}\, \Delta \mathbf{w}_0 = \mathbf{r}_0 - \sum_{i=1}^{n_s} \mathbf{B}_i^\top \mathbf{\Phi}_i^{-1} \mathbf{r}_i \tag{4.40}$$

where the contributions from each $\mathbf{B}_i^\top \mathbf{\Phi}_i^{-1} \mathbf{r}_i$ can be computed in parallel. Following this, the second-stage linear systems for $i = 1, \ldots, n_s$, can be solved in parallel for each $\Delta \mathbf{w}_i$ according to,

$$\mathbf{\Phi}_i \Delta \mathbf{w}_i = \mathbf{r}_i - \mathbf{B}_i \Delta \mathbf{w}_0 \tag{4.41}$$

A detailed account of the specified steps involved in solving Equations 4.39, 4.40 and 4.41 are highlighted in the following section.

4.3.4 Implementation Details

The proposed SQP-IPM algorithm was implemented in `C++` using appropriate object-oriented software design principles. Our implementation incorporates the QP solver `OOPS` and relies heavily on the structured linear algebra classes within this package for all vector-matrix computations [13, 14]. The particular IPM used in `OOPS` is a higher-order primal-dual technique which follows a predictor-corrector algorithm [25]. Dynamic optimization formulations are handled using the multiple-shooting discretization approach [38] whereby the embedded integration tasks are performed using the ODE solver `CVODES` [24]. The chosen integration routine provides the necessary parameter sensitivity computations, and using this informa-

tion the constraint Jacobian of Equation 4.2 is constructed. An illustrative algorithm for the NLP solution with embedded ODE/DAE's was previously provided in our work [39], which we follow in this chapter with the exception that we now use an IPM QP solver and a block-based BFGS update scheme according to Equation 4.17. This revised algorithm is given in Algorithm 3 using the general notation of Problem P.4.3 where we highlight the associated communication costs of the parallel implementation.

Algorithm 3 Parallel SQP algorithm using IPM decomposition for the QP solution

1: set initial primal-dual point: $\mathbf{x}^{[0]}$, $\boldsymbol{\lambda}^{[0]}$, $\boldsymbol{\mu}^{[0]}$
   (vector/matrix memory stored based on structure, see [14])
2: evaluate initial function and derivatives:
   $f^{[0]}$, $\mathbf{h}^{[0]}$, $\mathbf{g}^{[0]}$, $\nabla\mathbf{f}^{[0]}$, $\nabla\mathbf{h}^{[0]}$, $\nabla\mathbf{g}^{[0]}$
   (scalar $f^{[0]}$ incurs communication: MPI `all-reduce`)
   (vector/matrix pieces evaluated on select processors)
3: set Lagrangian Hessian approximation: $\mathbf{H}^{[0]} \leftarrow \sigma\mathbf{I}$
   repeat until termination criteria satisfied
4: check termination (possibly exit) via Eqns. 4.10–4.12
   (norm calculation incurs communication: MPI `all-reduce`)
5: form QP approximation: $\mathbf{c}^{[k]}$, $\mathbf{b}^{[k]}$, $\mathbf{n}^{[k]}$
   $\mathbf{A}^{[k]} := \nabla\mathbf{h}^{[k]}$, $\mathbf{C}^{[k]} := \nabla\mathbf{g}^{[k]}$, $\mathbf{Q}^{[k]} := \mathbf{H}^{[k]}$
   (vector/matrix pieces evaluated on select processors)
6: compute **Newton direction** via IPM using `OOPS`
   QP solver: $\mathbf{d}^{[k]}$, $\mathbf{y}^{[k]}$, $\mathbf{v}^{[k]}$ (see [13, 14, 25])
7: compute **step size** $\alpha^{[k]}$ via line search, Eqns. 4.4–4.7
   (incurs communication: MPI `all-reduce`)
8: update primal-dual point: $\mathbf{x}^{[k]}$, $\boldsymbol{\lambda}^{[k]}$, $\boldsymbol{\mu}^{[k]}$
9: update Lagrangian Hessian approximation $\mathbf{H}^{[k]}$ via Eqns. 4.16–4.23
   (matrix blocks $\mathbf{H}_i^{[k]}$ evaluated on select processors)
10: $k \leftarrow k + 1$
   end

The software implementation is tailored to the distributed computing framework and makes use of the message passing interface (MPI) for all parallel computation. Conceptually, one can think of this as having several independent copies of the main program running simultaneously (based on a number of defined processes) whereby certain computations are simultaneously performed on select processes and then communicated to select or all processes (e.g., MPI `all-reduce` calls, see [40]) as per the software design. The computation aspects performed in parallel within our implementation, beyond the internal QP solution handled by `OOPS`, include the matrix-vector computations within the line-search algorithm, termination criteria norm calculations, all major SQP user-function evaluations, and the BFGS evaluation. This is primarily achieved through the unique linear-algebra class design within `OOPS`, which abstracts much of lower level MPI communications away from the user, allowing for a rather straightforward user-level algorithm implementation.

To provide further insight on the parallel solution of the QP subproblem, we identify the key steps in solving the augmented linear system given by Equation 4.35 according to Algorithm 4. We note that this particular parallelization approach is fairly well established and its discussed, as well as some further improvements, by a number of independent research groups (see, [10, 12, 37]). For the particular algorithm presented, we provide the following remarks. The solution of the augmented linear system is performed in the usual two phase procedure of factorization (Schur-complement formation) and back-solve. Steps 1 to 6 involve forming the Schur-complement $\mathbf{C}$ as per Equation 4.39 where: in step 1 $\mathbf{\Phi}_i$ is factorized for $i = 1, \ldots, n_s$ (on each processor for a predefined scenario/block allocation) using a modified Cholesky procedure into lower triangular $\mathbf{L}_i$ and diagonal $\mathbf{D}_i$ matrices; in step 2 an intermediary matrix $\mathbf{V}_i$ is formed, which requires the solution of separate linear system for each corresponding matrix column; in step 3 we form another intermediary matrix $\mathbf{S}_i$, which is accumulated into $\mathbf{S}$ on each processor in step 4; next we form the Schur-complement $\mathbf{C}$ on each processor in step 5, which is subsequently factored into $\mathbf{L}_0$ and $\mathbf{D}_0$ matrices in step 6. Note, the specific formation and factorization of $\mathbf{C}$ is done on

all processes (i.e., a potential serial bottleneck) through the appropriate communication of each $\mathbf{S}_i$, or alternatively through a slightly modified scheme according to [37]. Next, a back solve phase is performed in steps 7 to 14 in order to solve Equations 4.40 and 4.41. Steps 7 and 8 involve determining intermediary vectors $\mathbf{z}_i$ and matrices $\mathbf{T}_i$ for $i = 1, \ldots, n_s$ (on each processor for a predefined scenario/block allocation), and step 9 accumulates $\mathbf{T}_i$ into $\mathbf{T}$ on all processors. Note, steps 7 to 9 relate back to Equation 4.40 through the following relation $\mathbf{T} := \sum_{i=1}^{n_s} \mathbf{V}_i^\top \mathbf{L}_i^{-1} \mathbf{r}_i \Leftrightarrow \sum_{i=1}^{n_s} \mathbf{B}_i^\top \mathbf{\Phi}_i^{-1} \mathbf{r}_i$. Steps 10 to 12 solve, in sequence, for intermediary vectors $\mathbf{z}_0$, $\mathbf{y}_0$ and $\mathbf{x}_0$ on all processors, where this last vector $\mathbf{x}_0 \equiv \Delta\mathbf{w}_0$ relates back to the complete solution of Equation 4.40. Finally, in steps 13 and 14, the vectors $\mathbf{y}_i$ and $\mathbf{x}_i$ are determined in sequence on each processor for a predefined scenario/block allocation. Again, we note that this last vector $\mathbf{x}_i \equiv \Delta\mathbf{w}_i$ relates back to Equation 4.41. In this algorithm, we represent the complete solution as $[\mathbf{x}_0, \ldots, \mathbf{x}_{n_s}]^\top \equiv [\Delta\mathbf{w}_0, \ldots, \Delta\mathbf{w}_{n_s}]^\top$ for notational convenience.

Implementation aspects unique to our approach are primarily related to the evaluation of the user-supplied functions and derivatives, which based on Problem P.4.2, are performed across several processes that govern the predefined location of each scenario/period group (see [14] for a description of how the vector/matrices are allocated). The most expensive computations in our implementation are associated with the evaluation of each ODE and sensitivity system required to formulate Equation 4.2 and respective derivatives. In our multiperiod multiple-shooting implementation (see, [39]), the equations/variables associated with each period/scenario are grouped into so-called blocks (as defined by the user) where each block is assigned to a particular computing process/processor. On each processor, we further allocate separate memory associated with `CVODES` for the system of ODEs of a single shooting interval and then reuse this memory to solve each successive ODE system for the remaining shooting intervals. This is further repeated for each period/scenario within each block, all of which occurs simultaneously across each defined processor. Thus, we are parallelizing the solution of the defined block groups, as opposed to each integration

task. Accordingly, this leaves further possibilities for future work in terms of using shared-memory `OpenMP` constructs for finer levels of parallelization associated with each block on each computing node.

Algorithm 4 Parallel Schur-complement decomposition algorithm for linear IPM augmented system: $\mathbf{P\Phi P}^{-1}\Delta\mathbf{w} = \mathbf{r}$

procedure $\{\Delta\mathbf{w}_0, \ldots, \Delta\mathbf{w}_{n_s}\} \leftarrow$ SCHUR_SOLVE($\mathbf{\Phi}$, $\mathbf{r}$)

Schur-complement formation phase

factorize via modified Cholesky $\mathbf{\Phi}_i \rightarrow \mathbf{L}_i\mathbf{D}_i\mathbf{L}_i^\top$ for each $i = 1, \ldots, n_s$ (on each process)

compute intermediary matrix $\mathbf{V}_i \leftarrow \mathbf{D}_i^{-1}\mathbf{L}_i^{-1}\mathbf{B}_i$ for each $i = 1, \ldots, n_s$ (on each process)
where each column j in $\mathbf{V}_i$ is determined by solving $(\mathbf{L}_i\mathbf{D}_i)\,\mathbf{V}_i^{<j>} = \mathbf{B}_i^{<j>}$

compute $\mathbf{S}_i \leftarrow \mathbf{V}_i^\top\mathbf{D}_i\mathbf{V}_i$ for each $i = 1, \ldots, n_s$ (on each process)

compute $\mathbf{S} \leftarrow \sum_{i=1}^{n_s}\mathbf{S}_i$ (`all-reduce` communication from each process)

compute $\mathbf{C} \leftarrow \mathbf{\Phi}_0 - \mathbf{S}$ (on all processes)

factorize via modified Cholesky $\mathbf{C} \rightarrow \mathbf{L}_0\mathbf{D}_0\mathbf{L}_0^\top$ (on all processes)

Back-solve phase

solve $\mathbf{L}_i\,\mathbf{z}_i = \mathbf{r}_i$ for each $i = 1, \ldots, n_s$ (on each process)

compute $\mathbf{T}_i \leftarrow \mathbf{V}_i^\top\mathbf{z}_i$ for $i = 1, \ldots, n_s$ (on each process)

compute $\mathbf{T} \leftarrow \sum_{i=1}^{n_s}\mathbf{T}_i$ (`all-reduce` communication from each process)

solve $\mathbf{L}_0\,\mathbf{z}_0 = \mathbf{r}_0 - \mathbf{T}$ (on all processes)

solve $\mathbf{D}_0\,\mathbf{y}_0 = \mathbf{z}_0$ (on all processes)

solve $\mathbf{L}_0^\top\mathbf{x}_0 = \mathbf{y}_0$ (on all processes)

solve $\mathbf{D}_i\,\mathbf{y}_i = \mathbf{z}_i$ for each $i = 1, \ldots, n_s$ (on each process)

solve $\mathbf{L}_i^\top\mathbf{x}_i = \mathbf{y}_i - \mathbf{V}_i\,\mathbf{x}_0$ for each $i = 1, \ldots, n_s$ (on each process)

end procedure

4.4 Example Problems

To demonstrate the potential of our proposed SQP-IPM algorithm for tackling large-scale multiperiod NLP formulations, we consider several example problems. The first example aims to highlight the QP solution performance of `OOPS` on a constrained linear least-square

problem, the second example uses a static NLP representing an uncertain design formulation, and the final example considers an uncertain dynamic optimization formulation using a multiperiod multiple-shooting discretization approach. In these examples, the multiperiod nature of the formulation is related to using either multiple data sets or multiple uncertain parameter realizations. This data are then grouped into independent blocks which resemble the form given in Equation 4.35. For example, if one generated 10 uncertain parameter realizations, then they could be grouped into 2 blocks of 5 scenarios each, or 5 blocks with 2 scenarios each. In either case, we only parallelize based on the number of independent blocks; however, the manner in which this grouping is performed can facilitate the assessment of different scaling properties of the algorithm. These scaling/performance properties are measured using strong and weak scaling metrics which are typically presented in the form of speedup and efficiency. For strong scaling, these metrics are formed by measuring the computation time to handle a fixed amount of work balanced over the available processors, while increasing the number of processors (i.e., the work per process would decrease for an increase in processors). In this manner, one would access the strong scaling of the algorithm using the metrics of speedup defined as $\mathrm{S} := \mathrm{T}_{\mathrm{serial}}/\mathrm{T}_{\mathrm{parallel}}$ and efficiency as $\mathrm{E} := \mathrm{T}_{\mathrm{serial}}/(\mathrm{T}_{\mathrm{parallel}} \cdot \mathrm{N})$, where N is the number of processors used, $\mathrm{T}_{\mathrm{serial}}$ is the serial wall clock time required to run the program, and $\mathrm{T}_{\mathrm{parallel}}$ is the corresponding time using multiple processors. Based on how parallelization is achieved within the algorithm, measuring strong scaling requires changing the size of each block as processors are added so that the overall problem size remains constant. The process of measuring weak scaling, on the other hand, requires that the number of blocks used be increased in direct relation to the number of processors. Therefore, for weak scaling, one defines the efficiency metric as $\mathrm{E} := \mathrm{T}_{\mathrm{serial\_one\_unit}}/\mathrm{T}_{\mathrm{parallel\_N\_units}}$, where $\mathrm{T}_{\mathrm{serial\_one\_unit}}$ is the serial computation time for a single unit of work (i.e., single block) and $\mathrm{T}_{\mathrm{parallel\_N\_units}}$ is the parallel time for N units of work (i.e., N blocks) using N processors, such that the units of work and processors are increased in a one to one ratio.

4.4.1 Example 1: Parameter Estimation

In this first example we consider a scalable demonstrative QP in the form of a constrained linear least-squares multi-data-set parameter estimation problem, similar to that used in Kang et al. [12]. The purpose here is to set the stage by providing a quantitative comparison between IPMs that use a serial full-space factorization versus a serial and parallel Schur-complement decomposition of the augmented system matrix. The IPM QP solver we selected that uses a full-space factorization approach is `OOQP` [34], which further uses the linear solver `MA27` [41] and Gondzio's modification to Mehrotra's predictor-corrector algorithm (as implemented therein). For the Schur-complement decomposition we utilize the solver `OOPS` [14]. In order to provide a systematic comparison, we assess several scalability factors of the QP algorithm by increasing the size of each data set n_y and, independently, the number of complicating model parameters n_p within the formulation. More specifically, for our numerical experiments we consider a data set size (i.e., size of model response or data vector) defined arbitrarily as $n_y = 2 \cdot n_q$ where n_q represents the number of local model parameters for each data set selected as $n_q = \{5000, 50000\}$. The number of global model parameters to be estimated from the combination of each data set are selected as $n_p = \{1000, 5000\}$, and the number of independent data sets used is defined as $n_s = \{2, 4, 8, 16\}$. The multi-data-set parameter estimation formulation can be generally stated as,

$$\begin{aligned} \min_{\mathbf{z}_i \, \forall \, i, \, \mathbf{p}} \quad & \textstyle\sum_{i=1}^{n_s} \frac{1}{2} \| \mathbf{y}_i - \mathbf{y}_i^* \|_2^2 \\ \text{st} : \quad & \mathbf{y}_i - \mathbf{G} \, \mathbf{q}_i = \mathbf{0} \\ & \mathbf{M} \, \mathbf{z}_i - \mathbf{p} = \mathbf{0} \\ & \mathbf{z}_i \in [\mathbf{z}_L, \mathbf{z}_U] \, , \; \forall \; i = 1, \ldots, n_s \\ & \mathbf{p} \in [\mathbf{p}_L, \mathbf{p}_U] \end{aligned} \tag{E.4.1}$$

where $\mathbf{z}_i := [\mathbf{y}_i^\top, \mathbf{q}_i^\top]^\top \in \mathbb{R}^{n_z}$, $n_z = n_y + n_q$, represent scenario-dependent parameters (second-stage variables), $\mathbf{p} \in \mathbb{R}^{n_p}$ are common or complicating parameters (first-stage vari-

ables), and $\mathbf{y}_i^* \in \mathbb{R}^{n_y}$ represents a vector for the ith set of randomly generated measurement data of dimension n_y for $i = 1, \ldots, n_s$ data sets. The coefficient matrix $\mathbf{G} \in \mathbb{R}^{n_y \times n_q}$ is arbitrarily defined as $\mathbf{G} := [\mathbf{T}^\top, \mathbf{T}^\top]^\top$ where $\mathbf{T} \in \mathbb{R}^{n_q \times n_q}$ is a tridiagonal matrix with sub-diagonal and super-diagonal coefficient vectors of -1 and a diagonal coefficient vector of 2. Furthermore, the mapping matrix $\mathbf{M} := [\mathbf{0}_{n_p \times n_y}, \mathbf{I}_{n_p}, \mathbf{0}_{n_p \times (n_q - n_p)}]$ is used to extract the appropriate subset of linking parameters $\mathbf{q}_i$ from $\mathbf{z}_i$, which are linked back to $\mathbf{p}$. Note, for this particular formulation $n_q > n_p$. The primal variable initial guesses were set as $\{\mathbf{z}_1, \ldots, \mathbf{z}_{n_s}, \mathbf{p}\} = 1$, while the corresponding bounds were set as $\{\mathbf{z}_L, \mathbf{p}_L\} = -10$ and $\{\mathbf{z}_U, \mathbf{p}_U\} = 10$. This formulation can be written in standard QP form as per Problem P.4.5 and Equation 4.29 through the following definitions,

$$
\begin{aligned}
\mathbf{d} &:= [\mathbf{z}_1^\top, \ldots, \mathbf{z}_{n_s}^\top, \mathbf{p}^\top]^\top \\
\mathbf{c} &:= [-\mathbf{y}_1^{*\top}, \mathbf{0}_{n_q}^\top, \ldots, -\mathbf{y}_{n_s}^{*\top}, \mathbf{0}_{n_q}^\top, \mathbf{0}_{n_p}^\top]^\top, \; \mathbf{b} := \mathbf{0}_{(n_y + n_p) n_s} \\
\mathbf{Q} &:= \operatorname{diag}\left(\mathbf{I}_{n_y}, \mathbf{0}_{n_q \times n_q}, \ldots, \mathbf{I}_{n_y}, \mathbf{0}_{n_q \times n_q}, \mathbf{0}_{n_p \times n_p}\right) \\
\mathbf{A} &:= \begin{bmatrix} \mathbf{D}_1 & & & \mathbf{E}_1 \\ & \ddots & & \vdots \\ & & \mathbf{D}_{n_s} & \mathbf{E}_{n_s} \end{bmatrix}, \; \mathbf{D}_i := \begin{bmatrix} \mathbf{I}_{n_y} \mid -\mathbf{G} \\ \mathbf{M} \end{bmatrix}, \; \mathbf{E}_i := \begin{bmatrix} \mathbf{0}_{n_y \times n_p} \\ -\mathbf{I}_{n_p} \end{bmatrix}
\end{aligned}
\tag{4.42}
$$

where the data $\mathbf{y}_i^*$ for each $i = 1, \ldots, n_s$ was generated a priori by adding randomly generated noise to a known model response $\mathbf{y}_i$. Note that to improve convergence of the QP, a small constant regularization term (e.g., $\epsilon = 1 \times 10^{-4}$, in our case) was added to zero diagonal elements of $\mathbf{Q}$ such that all eigenvalues are strictly positive.

Figure 4.1 plots weak scaling results where the number of processors N are increased in relation to the number of blocks ($n_b \equiv n_s$), such that a single scenario is assigned to each block and each block is assigned to a dedicated processor. In Figure 4.1 (a) we plot the wall clock time for each QP iteration versus the number of blocks used, which in the parallel case is also the number of processors used. A comparison is shown between a serial full-space (FS) factorization via `MA27` within `OOQP` and serial and parallel Schur-complement decomposition

(SCD) as per `OOPS`. From this plot we see that the serial full-space factorization time grows exponentially as the problem size increases, the serial Schur-complement decomposition time grows almost linearly, while the parallel Schur-complement decomposition time remains relatively constant. In Figure 4.1 (b) we show the weak scaling efficiency at two different values of n_q, which in both cases indicates a degradation in parallel performance from the ideal value of one as we increase the number of blocks and processors. However, for the larger value of n_q the efficiency trends are generally improved, which can be attributed to a decrease in communication overhead relative to computing time on each processor. The overall declining efficiency behavior, in either case, is likely related to an inadequate balance of work and resources when the number of blocks is increased in direct proportion to the number of processors, as opposed to an alternative relation that maximizes processor loading, thus creating a sense of under utilization. Furthermore, this could be attributed to a greater influence of communication operations as the processors are increased.

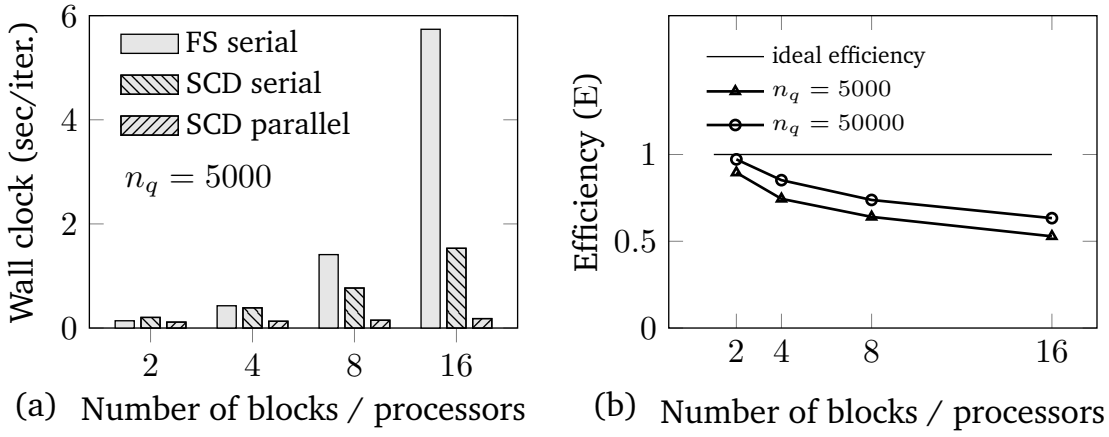

Figure 4.1: Example 1 – weak scaling results for increasing n_s and processors

Table 4.1: Example 1 – computation results for QP-IPM algorithm comparing increasing n_s, n_q and n_p using full-space (FS) and Schur-complement decomposition (SCD) approaches

Problem Dimension					FS (OOQP)		SCD (OOPS)				
n_p	n_q	n_s *	n †	m ‡	#iter *	wcs $^\triangleleft$	#iter *	wcs $^\triangleleft$	wcp $^\diamond$	S $^\bullet$	E $^\bullet$
$1 \cdot 10^3$	$5 \cdot 10^3$	2	$3.1 \cdot 10^4$	$5.3 \cdot 10^4$	7	0.978	12	2.446	1.365	1.79	0.90
		4	$6.1 \cdot 10^4$	$1.05 \cdot 10^5$	7	2.995	12	4.656	1.564	2.98	0.74
		8	$1.21 \cdot 10^5$	$2.09 \cdot 10^5$	7	9.874	12	9.223	1.799	5.12	0.64
		16	$2.41 \cdot 10^5$	$4.17 \cdot 10^5$	7	40.17	12	18.38	2.173	8.46	0.53
$5 \cdot 10^3$	$5 \cdot 10^3$	2	$3.5 \cdot 10^4$	$6.5 \cdot 10^4$	7	1.72	12	22.13	19.98	1.11	0.55
		4	$6.5 \cdot 10^4$	$1.25 \cdot 10^5$	7	5.77	12	31.41	23.25	1.35	0.34
		8	$1.25 \cdot 10^5$	$2.45 \cdot 10^5$	7	21.57	12	49.29	26.47	1.86	0.23
		16	$2.45 \cdot 10^5$	$4.85 \cdot 10^5$	7	102.56	12	86.67	31.89	2.72	0.17
$1 \cdot 10^3$	$5 \cdot 10^4$	2	$3.01 \cdot 10^5$	$5.03 \cdot 10^5$	7	54.78	12	38.86	19.99	1.94	0.97
		4	$6.01 \cdot 10^5$	$1.005 \cdot 10^6$	7	239.02	12	75.94	22.28	3.41	0.85
		8	$1.201 \cdot 10^6$	$2.009 \cdot 10^6$	7	1497.10	12	153.17	25.94	5.90	0.74
		16	$2.401 \cdot 10^6$	$4.017 \cdot 10^6$	7	7187.57	12	342.11	33.77	10.13	0.63
$5 \cdot 10^3$	$5 \cdot 10^4$	2	$3.05 \cdot 10^5$	$5.15 \cdot 10^5$	7	62.91	12	167.97	100.38	1.67	0.83
		4	$6.05 \cdot 10^5$	$1.025 \cdot 10^6$	7	250.05	12	287.73	114.04	2.52	0.63
		8	$1.205 \cdot 10^6$	$2.045 \cdot 10^6$	7	1642.94	12	566.67	128.48	4.41	0.55
		16	$2.405 \cdot 10^6$	$4.085 \cdot 10^6$	7	7640.42	12	1121.78	171.87	6.53	0.41

* number of data sets n_s = number of blocks n_b; † number of variables: $n = (n_y + n_q)n_s + n_p$;

‡ number of constraints: $m = (n_y + n_p)n_s + n$; * QP convergence tolerance 1×10^{-8};

$^\triangleleft$ total program serial wall clock time (seconds); $^\diamond$ total program parallel wall clock time (seconds);

$^\bullet$ weak scaling S = speedup and E = efficiency, both on a per QP iteration basis

Table 4.1 provides a more comprehensive display of timing results, where we assess the influence of increasing the size of each data set through n_q, the number of complicating/global parameters n_p and the number of scenario realizations n_s (i.e., blocks – in this case). Key observations are: (1) increasing n_q for a fixed n_p creates larger blocks that can be solved with

an improved efficiency as n_s is increased (i.e., greater processor utilization); (2) increasing n_p, regardless of n_s, creates a larger dense Schur-complement matrix $\mathbf{C}$, as per Equation 4.39, which ultimately increases the time of the serial direct factorization and back solve operations thus degrading parallel performance, as seen by the drop in efficiency. These observations are generally well known in the literature, and exemplify the bottleneck in the direct factorization of $\mathbf{C}$ when n_p becomes exceptionally large. An established remedy is the use of an iterative conjugate gradient approach for the solution of the Schur-complement system of Equation 4.39 which avoids direct factorization altogether [12].

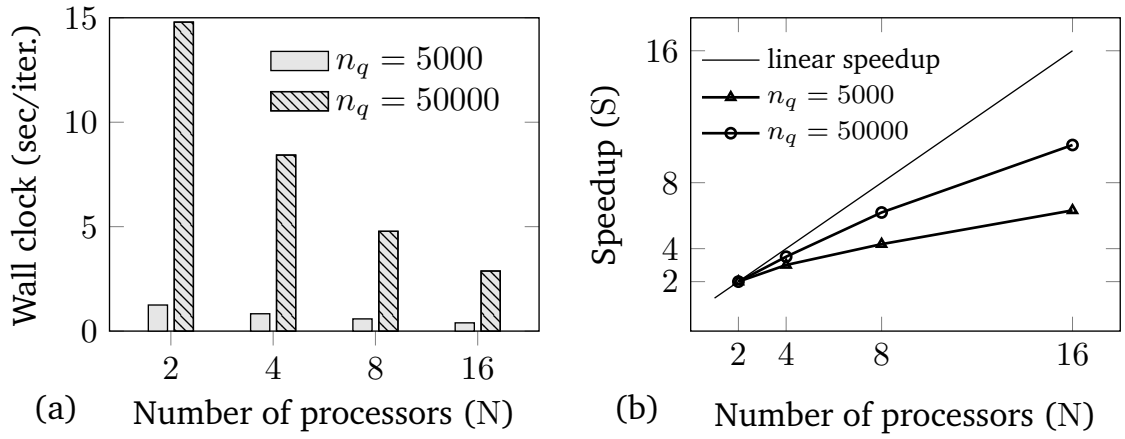

Figure 4.2: Example 1 – strong scaling results for fixed $n_s = 16$ and increasing processors

Next, we assess the strong scaling properties of `OOPS`, where we fix n_s and n_q and then increase the number of processors N such that the number of blocks is kept evenly distributed among each processor. Accordingly, as N is increased the number of blocks dedicated to each processor decreases and thus the computational work per process decreases. The consequence of decreasing the work load per processor is that communication overhead can become more pronounced at higher processor numbers which will deteriorate parallel performance. In Figure 4.2, we illustrate this parallel performance, considering processor work loads of $n_q = \{5000, 50000\}$ where $n_p = 1000$, by plotting both the wall clock time per QP iteration and speedup versus an increasing number of processors. As illustrated,

the speedup drifts from the ideal linear speedup at higher processor levels, and when the work load is increased the influence of the associated parallel overhead and serial Schur-complement formation/solution can be masked, thus allowing improved performance.

4.4.2 Example 2: Design Under Uncertainty

The next example considered formulates a multiperiod nonlinear program for design under uncertainty. With this problem we seek to provide an initial assessment of our proposed SQP-IPM algorithm using a significant number of scenario realizations evenly grouped into blocks. The problem was adapted from Bhatia and Biegler [7] who use a serial reduced-space SQP-IPM algorithm, and our purpose here is to use this benchmark formulation for comparison to our parallel SQP-IPM implementation. The particular mathematical program is written as,

$$\begin{aligned}
\min_{\mathbf{z}_i \,\forall\, i,\mathrm{p}} \quad & \mathrm{p}^2 + \textstyle\sum_{i=1}^{n_s} \alpha_i \,\mathsf{exp}(-\mathrm{y}_{1,i}) - 5\,\mathrm{y}_{2,i} + \mathrm{y}_{3,i}^2 \\
\mathrm{st}: \quad & \mathrm{y}_{1,i} + \beta_i\,\mathrm{y}_{2,i}^2 - 2\,\mathrm{y}_{3,i} - 5\,\mathrm{q}_i - 2 = 0 \\
& -\mathrm{y}_{2,i} - \gamma_i\,\mathrm{y}_{3,i} + 0.1\,\mathrm{q}_i^2 \leq 0 \\
& \mathbf{m}^\top \mathbf{z}_i - \mathrm{p} = 0 \\
& \mathbf{z}_i \in [\mathbf{z}_L, \mathbf{z}_U]\,,\ \forall\ i = 1, \ldots, n_s \\
& \mathrm{p} \in [\mathrm{p}_L, \mathrm{p}_U]
\end{aligned} \tag{E.4.2}$$

where $\mathbf{z}_i := [\mathbf{y}_i^\top, \mathrm{q}_i]^\top$, $\mathbf{m} \in \mathbb{R}^{n_z}$ is an appropriately defined mapping vector which extracts $\mathrm{q}_i \in \mathbb{R}$ from $\mathbf{z}_i$; $\alpha \in [0.5, 10]$, $\beta \in [1, 9.2]$, and $\gamma \in [1, 8.5]$ represent uncertain parameters which are generated a priori by sampling uniformly within the defined intervals. In this formulation there is only one degree of freedom given by the design parameter $\mathrm{p} \in \mathbb{R}$, one nonlinear equality and inequality constraint, respectively, and one linear linking constraint per scenario. The primal variable initial guesses were set as $\{\mathbf{z}_1, \ldots, \mathbf{z}_{n_s}, \mathrm{p}\} = 1$, while the corresponding bounds were set as $\mathbf{y}_L = 0$, $\{\mathrm{q}_L, \mathrm{p}_L\} = 1$, $\mathbf{y}_U = 3$ and $\{\mathrm{q}_U, \mathrm{p}_U\} = 50$.

Table 4.2: Example 2 – computation results for SQP-IPM algorithm comparing increasing n_s using full-space (FS) and Schur-complement decomposition (SCD) approaches

Problem Dimension					FS (WORHP)		SCD (OOPS)				
n_s ⋆	n_b †	n_g ‡	n	m	#iter *	wcs	#iter *	wcs	wcp	S •	E •
10	2	5	51	30	6/42	0.011	29/305	0.24	0.25	0.96	0.48
20	4		101	60	10/63	0.027	53/537	0.79	0.51	1.56	0.39
40	8		201	120	9/77	0.044	41/416	1.08	0.45	2.41	0.30
80	16		401	240	18/83	0.113	44/486	2.85	1.08	2.63	0.16
100	2	50	501	300	7/110	0.125	7/61	0.64	0.47	1.36	0.68
200	4		1001	600	20/132	0.306	58/713	12.24	5.18	2.36	0.59
400	8		2001	1200	12/125	0.478	63/893	36.39	10.23	3.56	0.44
800	16		4001	2400	6/134	1.135	7/75	10.83	2.54	4.27	0.27
1000	2	500	5001	3000	9/128	1.153	12/156	21.92	11.98	1.83	0.91
2000	4		10001	6000	22/326	5.706	45/585	107.66	35.29	3.05	0.76
4000	8		20001	12000	10/210	9.609	41/492	229.21	42.78	5.36	0.67
8000	16		40001	24000	17/250	23.95	20/262	591.24	72.37	8.17	0.51

⋆ number of scenario realizations; † number of blocks; ‡ number of scenarios per block; * major SQP iter./minor QP iter. where SQP convergence tolerances are: $\{\epsilon_{\text{opt}}, \epsilon_{\text{feas}}, \epsilon_{\text{comp}}\} = \{1 \times 10^{-8}, 1 \times 10^{-8}, 1 \times 10^{-3}\}$; • S = speedup and E = efficiency, both on a per SQP iteration basis

Table 4.2 provides an assessment of the SQP-IPM algorithm performance, where we consider several different numbers of scenario realizations n_s of increasing magnitude grouped into blocks of $n_b = \{2, 4, 8, 16\}$ where each block is comprised of scenario groups of size $n_g = \{5, 50, 500\}$. Note, the chosen scenario groupings and the number of blocks used allow each block to be stored and solved on each processor in a one to one relation. Additionally, we list the NLP dimension in terms of the number of variables (n) and constraints (m) and provide solution statistics based on the number of major SQP and minor QP iterations (#iter), the serial (wcs) and parallel (wcp) wall clock times for the entire program to run and speedup and efficiency statistics associated with the parallel implementation (based on the

average wall clock time for a single major SQP iteration, in order to normalize these statistics for comparison purposes). Note, the manner in which we assess parallel performance in this example is based on increasing both the number of processors and blocks in direct relation and thus constitutes a weak scaling evaluation. In addition to providing timing statistics of our SQP-IPM implementation (using OOPS), we also list serial computation statistics for the commercial state-of-the-art SQP-IPM solver WORHP which uses the linear solver MA97 [27, 41]. It should be noted that this commercial solver performs much better in serial than our proposed implementation and is equipped with many more features, such as warm starting each successive QP and better recourse to QP infeasibility failures. However, our intent here is not to directly compare this solver with our implementation and instead use the serial WORHP solution timings as a benchmark for reference. Currently, our SQP-IPM implementation does not warm start each QP, and as a result we see considerably more minor QP iterations at a significantly greater computational expense than the WORHP solver. Key observations from the results of our implementation include: (1) at high work loads of $n_g = 500$ scenarios per block/processor, the parallel computation time of each major SQP iteration (for a simultaneous increase of processors and blocks given by $n_b = \{2, 4, 8, 16\}$) is maintained relatively constant at $\mathrm{T}^{\mathrm{oops}}_{\mathrm{parallel}} = \{0.99, 0.78, 1.04, 3.62\}$ seconds compared to $\mathrm{T}^{\mathrm{oops}}_{\mathrm{serial}} = \{1.83, 2.39, 5.59, 29.56\}$ seconds in serial, which translates to a parallel efficiency of $\mathrm{E} = \{0.91, 0.76, 0.67, 0.51\}$; (2) for lower values of n_g and n_s, where the NLP dimension is rather small, performance is comparatively poorer; (3) through parallelization, for a $\times 100$ increase in problem size using groups $n_g = \{5, 500\}$ (see the rows for $n_s = \{80, 8000\}$ and the column for wcp in Table 4.2), the total program wall clock time increases by about $\times 67$, while in the purely serial case (see the respective rows and column wcs) the increase is much larger at $\times 207$. The postulated reason for the poorer parallel performance at smaller NLP sizes (see the column labeled E for rows $n_g = 5$ versus $n_g = 500$) is that the serial components of the algorithm (i.e., the line search algorithm highlighted in Equations 4.4 to 4.7 and termination criteria in Equations 4.10 to 4.12) are likely more dominant relative to the parallel components.

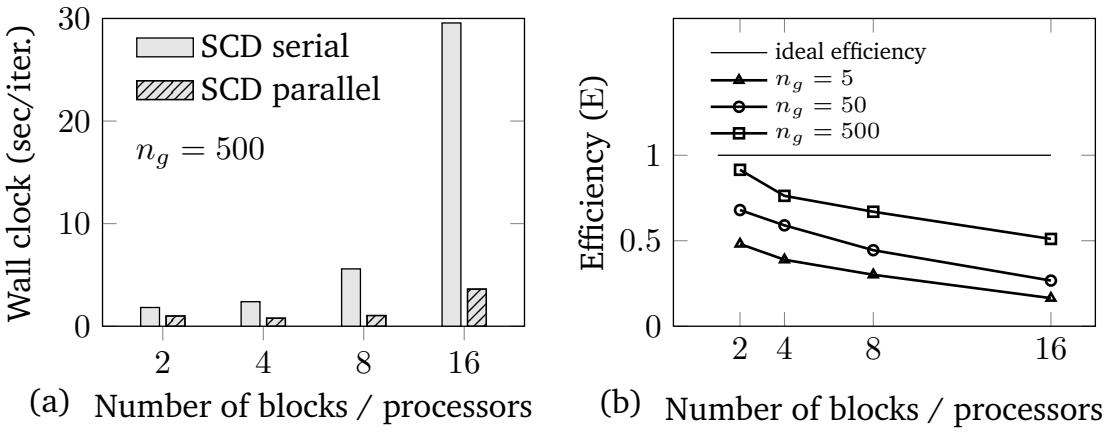

Figure 4.3: Example 2 – weak scaling results for increasing n_s and processors

The weak scaling results are further plotted in Figure 4.3, where the time per major SQP iteration and overall parallel efficiency are given for a simultaneous increase of blocks and processors defined by $\mathrm{N} = n_b = \{2, 4, 8, 16\}$. Furthermore, for the parallel efficiency shown in Figure 4.3 (b) we consider problem sizes described by $n_g = \{5, 50, 500\}$ and illustrate an improved performance as n_g increases. Ideally, we would like the efficiency to remain constant at unity as we increase the number of blocks and available computing resources. However, in the case of our implementation, we see a progressive decline in efficiency. It is again postulated that this behavior is a result of communication overhead (mainly through `all-reduce` MPI operations prior to evaluating the objective function) particularly within the line search algorithm, which consequently becomes more pronounced when using higher processor numbers.

4.4.3 Example 3: Dynamic Optimization Under Uncertainty

In our final example we consider a multiperiod dynamic optimization formulation for design under uncertainty. The problem was again adapted from [7], however unlike the full discretization approach used therein, we consider a multiple-shooting approach as per Equation 4.2, which requires the solution of an embedded dynamic system in the form of

Equation 4.1. The design problem considers determining the shortest time-span given by t_f (design variable) in order to achieve a maximum yield of product $x_{\mathsf{B}}(t_f)$ from a batch reactor, all while incorporating uncertainty within the reaction kinetics through the parameters $\boldsymbol{\theta} = [\theta_1, \theta_2]^\top$. The multiperiod dynamic optimization formulation can be stated over a normalized time horizon $\tau \in [0, 1]$ as,

$$
\begin{aligned}
\min_{t_f,\, u_i(\tau)\ \forall\, i} \quad & c_1\, {t_f}^{c_2} - \textstyle\sum_{i=1}^{n_s} w_i\, c_0\, x_{\mathsf{B},i}(1) \\
\text{st}: \quad & \dot{x}_{\mathsf{A},i}(\tau) = -[\theta_{1,i}\, u_i(\tau)^{\theta_{2,i}} + u_i(\tau)]\, x_{\mathsf{A},i}(\tau)\, t_f \\
& \dot{x}_{\mathsf{B},i}(\tau) = \theta_{1,i}\, u_i(\tau)\, x_{\mathsf{A},i}(\tau)\, t_f \\
& x_{\{\mathsf{A},\,\mathsf{B}\},i}(0) = x_{\{\mathsf{A0},\,\mathsf{B0}\}} \\
& x_{\{\mathsf{A},\,\mathsf{B}\},i}(\tau) \in [0, 1] \\
& u_i(\tau) \in [0, 5],\ t_f \in [0.5, 1.25]\ \forall\ \tau \in [0, 1],\ i = 1, \ldots, n_s \\
& \theta_1 \in (0.45, 0.55),\ \theta_2 \in (2.15, 2.25)
\end{aligned}
\tag{E.4.3a}
$$

where the objective function represents the combined costs of the operation time and the product revenue; the cost coefficients are defined as $\{c_0, c_1, c_2\} = \{700, 50, 2\}$; the weights associated with each scenario realization are given by $w_i = 1/n_s$. In the context of two-stage stochastic programming, the variable t_f represents the first stage decision, while the second-stage decisions are given by the open-loop control action $u_i(\tau)$ which provides compensatory action to uncertainty within the uncertain $\boldsymbol{\theta}$ parameter space. The fully discretized NLP

formulation can be further stated as,

$$\begin{aligned}
\min_{t_f,\,\mathbf{z}_i\,\forall\, i} \quad & c_1\, {t_f}^{c_2} - \textstyle\sum_{i=1}^{n_s} w_i\, c_0\, s_{2,i,n} \\
\text{st}: \quad & \mathbf{x}_0 - \mathbf{s}_{i,0} = \mathbf{0} \\
& \mathbf{x}(t_{i,j}; \mathbf{s}_{i,j-1}, u_{i,j-1}, q_i) - \mathbf{s}_{i,j} = \mathbf{0} \\
& q_i - t_f = 0 \\
& \mathbf{s}_{i,j} \in [0,1],\; u_{i,j} \in [0,5] \\
& \{q_i, t_f\} \in [0.5, 1.25] \\
& \forall\; i = 1, \ldots, n_s, \;\; j = 1, \ldots, n
\end{aligned} \tag{E.4.3b}$$

where $\mathbf{s}_{i,j} := [s_{1,i,j}, s_{2,i,j}]^\top$ represent shooting node state parameters, $u_{i,j}$ are control parameters for a piecewise constant profile, $\mathbf{x}_0 := [x_{\mathsf{A0}}, x_{\mathsf{B0}}]^\top$ are specified initial conditions with $x_{\mathsf{A0}} = 1$ and $x_{\mathsf{B0}} = 0$, and $\mathbf{z}_i := [\mathbf{s}_{i,0}^\top, u_{i,0}, \ldots, \mathbf{s}_{i,n-1}^\top, u_{i,n-1}, \mathbf{s}_{i,n}^\top, q_i]^\top$ represents a concatenation of all scenario-related parameters. The embedded ODE solution at each shooting node is represented by $\mathbf{x}(t_{i,j}; \mathbf{s}_{i,j-1}, u_{i,j-1}, q_i)$, which is functionally dependent on shooting node and local design parameters.

In this example, the performance of the SQP-IPM algorithm is further investigated where a more in-depth analysis is performed on embedded model ODE formulations. Table 4.3 is displayed in a similar manner to the previous example with similar connotations for each column heading. In addition, we include the fraction of the total program wall clock time spent solving the embedded ODEs, as represented by `fem`. Again, for reference purposes we compare the Schur-complement SQP-IPM algorithm developed here to the full space SQP-IPM solver `WORHP`. Considering several levels of scenario realizations $n_s = \{100, 200, 400, 800\}$, the serial wall clock time per major SQP iteration for `WORHP` is $\mathrm{T}^{\mathrm{worhp}}_{\mathrm{serial}} = \{0.62, 0.99, 3.17, 4.38\}$ seconds, while that of our implementation run in serial is about 1.5 times greater at $\mathrm{T}^{\mathrm{oops}}_{\mathrm{serial}} = \{0.67, 1.64, 4.46, 7.05\}$ seconds. These results are expected given that the number of minor QP iterations is over 2 times greater in our implementation. Now, if we consider using

$\mathrm{N} = n_b = \{2, 4, 8, 16\}$ processors and blocks, the wall clock timings per iteration of our implementation drop to $\mathrm{T}^{\mathtt{oops}}_{\mathrm{parallel}} = \{0.37, 0.47, 0.81, 1.06\}$ seconds, which can be considered a significant improvement over the full space approach, particularly when considering the additional work incurred by neglecting the warm starting of each QP in our implementation. This additional work is further manifested by an increased in-solver NLP time when compared to the total ODE solution time. This is evident as the problem size grows and the `fem` statistic decreases, which is primarily due to an increase in the total number of QP iterations associated with our particular implementation and the additional communication overhead as the number of blocks n_b and processors N increase. If we compare our implementation to the full space serial solver, results suggest that the `fem` should otherwise remain fairly constant.

Table 4.3: Example 3 – computation results for SQP-IPM algorithm with embedded ODE comparing increasing n_s using full-space (FS) and Schur-complement decomposition (SCD) approaches

Problem Dimension					FS (`WORHP`)			SCD (`OOPS`)						
n_s ⋆	n_b †	n ◇	m	n_{eq} ‡	`#iter` *	`wcs`	`fem`	`#iter` *	`wcs`	`fem`	`wcp`	`fem`	S •	E •
100	2	7801	5300	5000	101/449	63.01	0.84	88/805	59.41	0.64	32.96	0.72	1.80	0.90
200	4	15601	10600	10000	177/589	175.62	0.84	121/1175	198.07	0.54	58.07	0.67	3.41	0.85
400	8	31201	21200	20000	102/439	323.80	0.87	164/1874	731.12	0.43	132.03	0.52	5.54	0.69
800	16	62401	42400	40000	51/154	223.18	0.85	75/874	529.83	0.36	79.80	0.46	6.64	0.41

⋆ number of scenario realizations; † number of blocks ($n_g = 50$ scenarios per block);

‡ total number of embedded ODEs (2 per (j, i)) with number shooting intervals $n = 25$;

◇ total number of NLP variables; * major SQP iter./minor QP iter. where SQP convergence tolerances are: $\{\epsilon_{\mathrm{opt}}, \epsilon_{\mathrm{feas}}, \epsilon_{\mathrm{comp}}\} = \{1 \times 10^{-6}, 1 \times 10^{-8}, 1 \times 10^{-3}\}$; • S = speedup and E = efficiency, both on a per SQP iteration basis

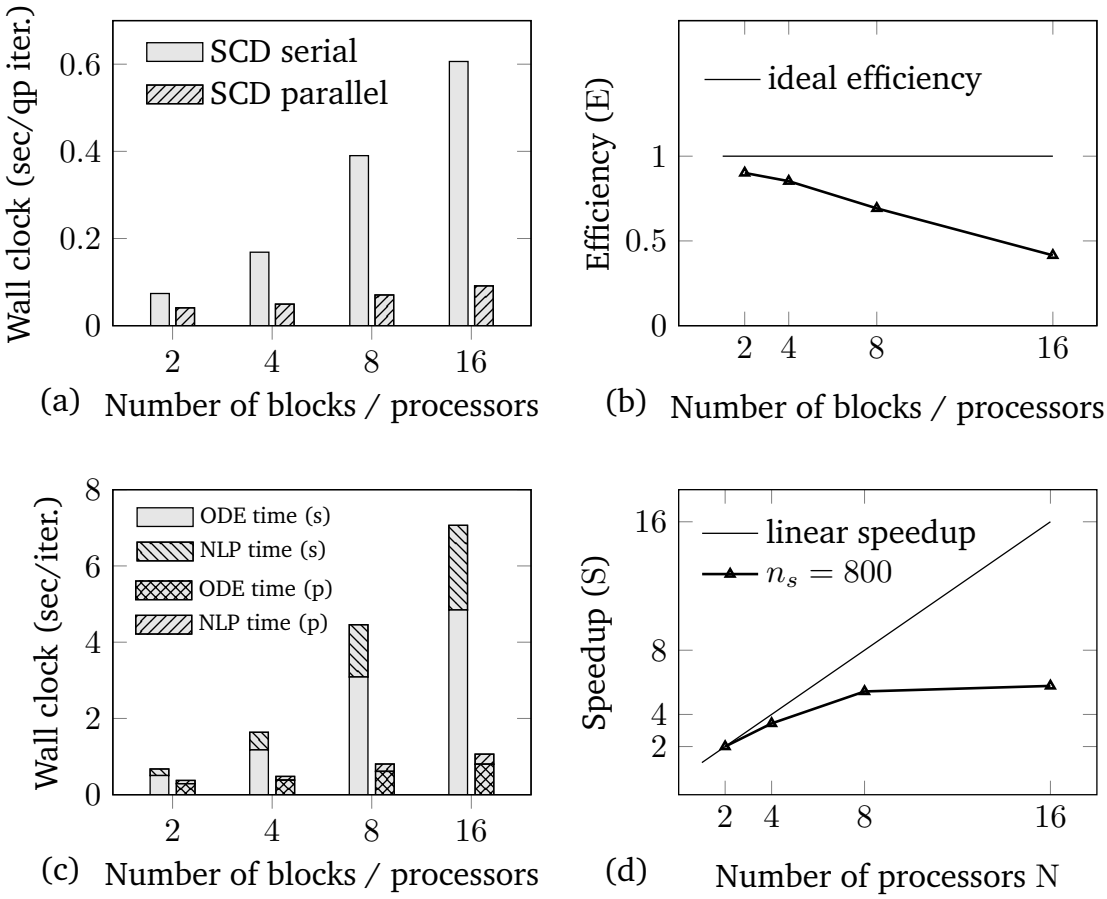

Figure 4.4: Example 3 – (a)–(c) weak scaling results for increasing n_s and processors; (d) strong scaling speedup for fixed $n_s = 800$ and increasing processors

Figures 4.4 (a) to (c) provide a visual assessment of weak scaling performance, while Figure 4.4 (d) gives strong scaling speedup results. In Figure 4.4 (a), the wall clock time is broken down based on the time per QP iteration, where the parallel time only slightly increases when both the number of blocks n_b and processors N are increased simultaneously. Figure 4.4 (b) provides the corresponding parallel efficiency, and is observed to decrease with an increase in n_b/N, which ideally should remain constant. However, for the same reasons in the previous examples (serial and communication overhead) we did not observe this ideal situation within our implementation. One approach to curb this declining performance would be to increase the amount of work per processor; however, the better approach would be to re-examine or

re-design the particular communication calls used in the implementation to limit the overhead as much as possible. Figure 4.4 (c) provides a further comparison of the wall clock times, on a major iteration basis, where the timings are broken down into the ODE solution time and in-solver NLP time. As evident, we observe a speedup of $\mathrm{S_{ODE}} = \{1.6, 2.75, 4.54, 5.16\}$ for the ODE solution and $\mathrm{S_{NLP}} = \{2.29, 4.75, 6.58, 7.95\}$ for the in-solver NLP computations, which together produce an overall parallel efficiency of $\mathrm{E} = \{0.90, 0.85, 0.69, 0.41\}$. As a final analysis, we consider the strong scaling performance in Figure 4.4 (d), where the problem size is kept constant at $n_s = 800$ and the number of processors is increased such that the number of blocks remain evenly balanced on each processor. As evident, we see a significant degradation in speedup at the higher processor levels, which again is related to a combination of decreased work and increased communication overhead.

4.5 Concluding Remarks

In this chapter, we have proposed and demonstrated an SQP algorithm for solving large-scale NLP formulations that is capable of exploiting the structure of the formulation and thus facilitate the parallelization of the underlying linear algebra within the algorithm. The algorithm was applied to two NLP formulations, one of which was a discretized uncertain dynamic optimization formulation that utilized a combined multiperiod multiple-shooting discretization approach. Our SQP implementation utilized an existing parallel IPM QP solver that was designed for the distributed computing paradigm using the message passing interface (MPI), and to this we added an appropriate globalization technique via a quadratic line search method and block-wise BFGS quasi-Newton update method. The parallel performance of the algorithm was assessed considering weak and strong scaling indicators and results suggest good scalability if the processor work load is significantly more dominant than the combined aspects of the parallel communication overhead and serial computation within the algorithm. The current quasi-Newton approach used herein is based on updating dense block matrices,

which can be quite expensive for large blocks. Future work will consider a block-wise BFGS update using a limited memory implementation [32], which is more appropriate for large-scale models; a feasibility restoration phase to handle infeasible QPs; further investigation of different MPI communication protocols within the line search and function evaluation routines to limit the associated parallel overhead and resulting performance degradation; an investigation of fine-grain parallelization using `OpenMP` constructs when solving the embedded ODEs/DAEs over each shooting interval; and additional applications that specifically consider the use of large-scale DAE process models.

List of References

[1] F. Allgower et al. “Nonlinear Predictive Control and Moving Horizon Estimation – An Introductory Overview”. In: *Advances in Control*. Ed. by P. M. Frank. Springer, 1999, pp. 391–449 (cit. on p. 110).

[2] A. Shapiro, D. Dentcheva, and A. Ruszczynski. *Lectures on stochastic programming: Modeling and Theory*. 2nd. SIAM, 2014 (cit. on pp. 110, 113).

[3] J. S. Logsdon, U. M. Diwekar, and L. T. Biegler. “On the simultaneous optimal design and operation of batch distillation columns”. In: *Chemical Engineering Research and Design* 68.5 (1990), pp. 434–444 (cit. on p. 111).

[4] T. K. Bhatia and L. T. Biegler. “Dynamic Optimization for Batch Design and Scheduling with Process Model Uncertainty”. In: *Industrial & Engineering Chemistry Research* 36.9 (1997), pp. 3708–3717 (cit. on p. 111).

[5] D. K. Varvarezos, L. T. Biegler, and I. E. Grossmann. “Multiperiod design optimization with SQP decomposition”. In: *Computers & Chemical Engineering* 18.7 (1994), pp. 579–595 (cit. on pp. 111, 114).

[6] R. A. Bartlett and L. T. Biegler. “QPSchur: A dual, active-set, Schur-complement method for large-scale and structured convex quadratic programming”. In: *Optimization and Engineering* 7.1 (2006), pp. 5–32 (cit. on p. 111).

[7] T. K. Bhatia and L. T. Biegler. “Multiperiod design and planning with interior point methods”. In: *Computers & Chemical Engineering* 23.7 (1999), pp. 919–932 (cit. on pp. 111, 116, 121, 124, 140, 143).

[8] D. J. Ternet and L. T. Biegler. “Interior-point methods for reduced Hessian successive quadratic programming”. In: *Computers & Chemical Engineering* 23.7 (1999), pp. 859–873 (cit. on p. 111).

[9] J. Albuquerque et al. “Interior point SQP strategies for large-scale, structured process optimization problems”. In: *Computers & Chemical Engineering* 23.4-5 (1999), pp. 543–554 (cit. on pp. 111, 116, 122).

[10] V. M. Zavala, C. D. Laird, and L. T. Biegler. “Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems”. In: *Chemical Engineering Science* 63.19 (2008), pp. 4834–4845 (cit. on pp. 111, 128, 131).

[11] D. P. Word et al. “Efficient parallel solution of large-scale nonlinear dynamic optimization problems”. In: *Computational Optimization and Applications* 59.3 (2014), pp. 667–688 (cit. on pp. 111, 128).

[12] J. Kang et al. “An interior-point method for efficient solution of block-structured NLP problems using an implicit Schur-complement decomposition”. In: *Computers & Chemical Engineering* 71 (2014), pp. 563–573 (cit. on pp. 111, 131, 135, 139).

[13] J. Gondzio and A. Grothey. “Parallel interior-point solver for structured quadratic programs: Application to financial planning problems”. In: *Annals of Operations Research* 152.1 (2007), pp. 319–339 (cit. on pp. 111, 129, 130).

[14] J. Gondzio and A. Grothey. “Exploiting structure in parallel implementation of interior point methods for optimization”. In: *Computational Management Science* 6.2 (2009), pp. 135–160 (cit. on pp. 111, 112, 128–130, 132, 135).

[15] J. Gondzio and A. Grothey. “Solving non-linear portfolio optimization problems with the primal-dual interior point method”. In: *European Journal of Operational Research* 181.3 (2007), pp. 1019–1029 (cit. on p. 111).

[16] A. Domahidi et al. “Efficient interior point methods for multistage problems arising in receding horizon control”. In: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. Dec. 2012, pp. 668–674 (cit. on p. 112).

[17] J. V. Frasch, S. Sager, and M. Diehl. “A parallel quadratic programming method for dynamic optimization problems”. In: *Mathematical Programming Computation* 7.3 (2015), pp. 289–329 (cit. on p. 112).

[18] I. D. Washington and C. L. E. Swartz. “Multi-Period Dynamic Optimization for Large-Scale Differential-Algebraic Process Models under Uncertainty”. In: *Processes* 3.3 (2015), pp. 541–566 (cit. on p. 112).

[19] J. Kang et al. “Nonlinear Programming Strategies on High-Performance Computers”. In: *IEEE Conference on Decision and Control*. Osaka, Japan, 2015 (cit. on p. 113).

[20] V. Sakizlis, J. D. Perkins, and E. N. Pistikopoulos. “Recent advances in optimization-based simultaneous process and control design”. In: *Computers & Chemical Engineering* 28.10 (2004), pp. 2069–2086 (cit. on p. 113).

[21] D. Navia et al. “A comparison between two methods of stochastic optimization for a dynamic hydrogen consuming plant”. In: *Computers & Chemical Engineering* 63 (2014), pp. 219–233 (cit. on p. 113).

[22] R. Huang and L. T. Biegler. “Robust nonlinear model predictive controller design based on multi-scenario formulation”. In: *Proceedings of the 2009 Conference on American Control Conference*. ACC’09. 2009, pp. 2341–2342 (cit. on p. 113).

[23] S. Lucia et al. “Handling uncertainty in economic nonlinear model predictive control: A comparative case study”. In: *Journal of Process Control* 24.8 (2014), pp. 1247–1259 (cit. on p. 113).

[24] A. C. Hindmarsh et al. “SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers”. In: *ACM Transactions on Mathematical Software* 31.3 (2005), pp. 363–396 (cit. on pp. 116, 129).

[25] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997 (cit. on pp. 116, 124, 129, 130).

[26] J. Nocedal and S. J. Wright. *Numerical Optimization*. 2nd. Springer, 2006 (cit. on pp. 116, 118, 119, 121, 122).

[27] C. Buskens and D. Wassel. “The ESA NLP Solver WORHP”. In: *Modeling and Optimization in Space Engineering*. Ed. by G. Fasano and J. D. Pinter. Vol. 73. Optimization and Its Applications. Springer, 2013, pp. 85–110 (cit. on pp. 116, 118, 119, 124, 142).

[28] B. Sachsenberg and K. Schittkowski. “A combined SQP-IPM algorithm for solving large-scale nonlinear optimization problems”. In: *Optimization Letters* 9.7 (2015), pp. 1271–1282 (cit. on p. 116).

[29] C. D. Laird and L. T. Biegler. “Large-Scale Nonlinear Programming for Multi-scenario Optimization”. In: *Modeling, Simulation and Optimization of Complex Processes*. Ed. by H. G. Bock et al. Springer Berlin Heidelberg, 2008, pp. 323–336 (cit. on pp. 116, 128).

[30] P. E. Gill, W. Murray, and M. A. Saunders. “SNOPT: An SQP algorithm for large-scale constrained optimization”. In: *SIAM Review* 47.1 (2005), pp. 99–131 (cit. on p. 119).

[31] M. J. Tenny, S. J. Wright, and J. B. Rawlings. “Nonlinear Model Predictive Control via Feasibility-Perturbed Sequential Quadratic Programming”. In: *Computational Optimization and Applications* 28.1 (2004), pp. 87–121 (cit. on p. 121).

[32] R. H. Byrd, J. Nocedal, and R. B. Schnabel. “Representations of quasi-Newton matrices and their use in limited memory methods”. In: *Mathematical Programming* 63.1-3 (1994), pp. 129–156 (cit. on pp. 122, 149).

[33] J. Gondzio. “Interior point methods 25 years later”. In: *European Journal of Operational Research* 218.3 (2012), pp. 587–601 (cit. on p. 124).

[34] E. M. Gertz and S. J. Wright. “Object-oriented software for quadratic programming”. In: *ACM Transactions on Mathematical Software* 29.1 (2003), pp. 58–81 (cit. on pp. 124, 135).

[35] A. Altman and J. Gondzio. “Regularized symmetric indefinite systems in interior point methods for linear and quadratic optimization”. In: *Optimization Methods and Software* 11.1-4 (1999), pp. 275–302 (cit. on p. 125).

[36] M. P. Friedlander and D. Orban. “A primal-dual regularized interior-point method for convex quadratic programs”. In: *Mathematical Programming Computation* 4.1 (2012), pp. 71–107 (cit. on p. 125).

[37] C. G. Petra et al. “An Augmented Incomplete Factorization Approach for Computing the Schur Complement in Stochastic Optimization”. In: *SIAM Journal on Scientific Computing* 36.2 (2014), pp. C139–C162 (cit. on pp. 129, 131, 132).

[38] H. G. Bock and K. J. Plitt. “A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems”. In: *Ninth IFAC World Congress*. Budapest, 1984 (cit. on p. 129).

[39] I. D. Washington and C. L. E. Swartz. “Design under uncertainty using parallel multiperiod dynamic optimization”. In: *AIChE Journal* 60.9 (2014), pp. 3151–3168 (cit. on pp. 130, 132).

[40] P. S. Pacheco. *Parallel Programming with MPI*. San Francisco, CA, USA: Morgan Kaufmann, 1996 (cit. on p. 131).

[41] *HSL. A collection of Fortran codes for large scale scientific computation*. 2015. URL: `http://www.hsl.rl.ac.uk/` (cit. on pp. 135, 142).

Chapter 5

Conclusions and Future Work

In this final chapter, the main contributions of this thesis are summarized and further directions are noted.

5.1 Concluding Remarks

In this work, we investigated an approach for solving stochastic dynamic optimization formulations using modern computational tools and algorithms which specifically utilized parallel computing approaches.

In Chapter 2, we proposed a multiperiod optimization approach for stochastic dynamic optimization formulations and demonstrated a prototype solution algorithm. Our focus here was on demonstrating potential algorithm performance improvement through parallelization where we targeted improving the speedup of the embedded dynamic model solution. Our approach specifically handles explicit ordinary differential equation (ODE) model representations and our applications were related to the integration of process systems design and control. The dynamic optimization application was to simultaneously determine economically optimal design/operation costs and control performance through the selection of optimal design parameters subject to closed-loop control profiles and parametric uncertainly within input disturbances. We addressed several computational performance aspects that arise when using multiperiod multiple-shooting dynamic optimization discretizations, in particular the exponential increase in computing requirements that occurs when increasing the granularity

of the discretization.

In Chapter 3, we extended our contributions made in Chapter 2 to include model representations given by semi-explicit differential-algebraic equations (DAEs). Furthermore, our intended applications and model representations were of a large-scale nature, which required the re-development of our proposed multiperiod multiple-shooting dynamic optimization algorithm presented in Chapter 2, into a more appropriate software tool written in C/C++ that utilized appropriate OpenMP complier constructs for parallel computing. Additionally, we investigated the use of second-order parameter sensitivity methods for generating explicit Lagrangian Hessian representations typically used in NLP algorithms that utilize exact second-order information. To demonstrate the performance of our parallel multiperiod multiple-shooting dynamic optimization tool we considered a large-scale air separation problem where the objective was to determine a robust open-loop control profile for transitioning the plant between two operating points, subject to uncertainty in the underlying plant model. A key result of the investigation was that the solution time of the embedded dynamic model within the overall nonlinear programming implementation was dramatically reduced, so much so that the computation bottleneck shifted to the interval in-solver NLP computation within the chosen active-set reduced SQP algorithm. Consequently, this result suggested a switch from an active-set method to an interior-point solution approach within the SQP algorithm.

In Chapter 4, we developed and investigated an SQP algorithm that incorporates an interior-point method for the solution of each quadratic programming (QP) subproblem. Our proposed approach was able to efficiently decompose and solve in parallel large-scale multiperiod nonlinear programming formulations with embedded dynamic model representations. Our SQP algorithm development was performed in C++ using the MPI library and additionally utilized an existing interior-point QP solver, to which we added several components making up the complete SQP algorithm. Several examples were used to assess the algorithms' parallel performance, which indicated a significant time reduction within the solution of the

algorithms' linear algebra with relatively good parallel scalability.

Ultimately this thesis has demonstrated progress towards using parallel computing techniques for solving large-scale multiperiod approximations to stochastic dynamic optimization formulations. Throughout the project several pieces of software were written that resulted in a multiperiod dynamic optimization tool and additionally a standalone nonlinear programming solver. The first tool uses OpenMP parallelization for solving the embedded ODE/DAE model with a multiple-shooting discretization, the second tool uses the MPI library for parallelizing the solution blocks of scenario groupings within the linear algebra of an NLP solver.

5.2 Future Work

Aspects for further work are noted as follows,

- The application of our developed software tools within model-based control and estimation studies which explicitly account for uncertainty as approximated through the multiperiod approach.
- The SUNDIALS ODE/DAE solvers used within the proposed multiperiod multiple-shooting algorithm were not utilized with the currently available OpenMP or MPI features. Thus, further parallelization aspects within the underlying linear algebra within these solvers could be investigated without much effort.
- The addition of several algorithmic aspects to the developed SQP-IPM algorithm which would allow greater recourse to potential algorithm failures when applied to difficult NLP problems and facilitate the application to larger formulations.
- Further investigation within our SQP-IPM algorithm to solving multiperiod multiple-shooting dynamic optimization formulations where the solution of the embedded dynamic model is parallelized using `OpenMP` constructs while the solution of the NLP

utilizes MPI. This hybrid approach would allow the embedded model solution to be evaluated using considerably more processors/threads than that needed for each block of the Schur-complement decomposition; thus, facilitating a greater utilization of parallel computing within the overall solution algorithm.