Large-scale dynamic optimization using code generation and parallel computing

J O S E S A N T I A G O R O D R I G U E Z

Master of Science Thesis
Stockholm, Sweden 2014

Large-scale dynamic optimization using code generation and parallel computing

J O S E S A N T I A G O R O D R I G U E Z

Master's Thesis in Scientific Computing (30 ECTS credits)
Master Programme in Computer simulation for Science and Engineering (120 credits)
Royal Institute of Technology year 2014
Supervisor at Modelon AB was Johan Åkesson
Supervisor at KTH was Michael Hanke
Examiner was Michael Hanke

TRITA-MAT-E 2014:50
ISRN-KTH/MAT/E--14/50--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden

URL: www.kth.se/sci

Acknowledgements

I am extremely grateful with the COSSE program and the European Union for all the support given along these two years of graduate studies. I feel very lucky to be chosen by the Erasmus Mundus consortium to be part of such an amazing inter-cultural graduate program. I wish more students could have the opportunity to participate and be financially supported by the program.

Thanks to my family and friends because without them this work would have not been possible. I would like to specially thank Prof. Carl Laird from Purdue University for all his advice and help. He has become an academic father and a great role model for me to follow. Thanks also to Dr. Daniel Word for all the support with the parallel framework. And last but not least, thanks to JModelica.org's development team for their patience and support during the master thesis.

Abstract

Complex physical models are becoming increasingly used in industry for simulation and optimization. Modeling languages such as Modelica allow creating model libraries of physical components, which in turn can be used to compose system models of, e.g., vehicle systems, power plants and electronic systems. JModelica.org is an open source tool suite for Modelica. It includes a compiler for Modelica and for the language extension Optimica, which is used to formulate dynamic optimization problems based on Modelica models.

Direct collocation methods are used in JModelica.org to transcribe a dynamic optimization problem into a large-scale nonlinear program with sparse structure. This structure can be exploited for parallel solution of the linear *Karush Kuhn Tucker* KKT system solved in each step of an interior point method. Currently, the non-linear programming solvers available in JModelica.org do not support parallel algorithms for the solution of the structured problem.

The optimization platform in JModelica.org was modified to generate efficient C-code that could be linked with an external non-linear programming solver with parallel computation capability. CasADi, JModelica.org's third party software, is used to generate C-code from the model equations, after collocation has been applied. Speed up factors over 20 were obtained for the compilation of the generated files, and the problem of non-compilable files was overcome.

A C++ interface was implemented to link the generated files with the parallel framework that solves non-linear problems with an interior point algorithm.A large-scale trajectory optimization problem, such as the start-up optimization of a power cycle model, was used to evaluate the performance of the interface and the parallel algorithm. Speedup of over 2.5 was obtained with 4 processors in a shared memory architecture.

Referat

Storskalig dynamisk optimering med hjälp kodgenerering och parallella beräkningar.

Komplexa fysikaliska modeller används alltmer inom industrin för simulering och optimering. Modelleringsspråk som Modelica möjliggör skapandet av modellbibliotek av fysikaliska komponenter, som i sin tur används för att skapa systemmodeller av t.ex. fordonssystem, kraftverk och elektronik. JModelica.org är ett open-source verktyg för Modelica. Det består av en kompilator för Modelica och dess utökning Optimica, som används för att formulera dynamiska optimeringsproblem baserat påModelica-modeller.

Direkta kollokationsmetoder används i JModelica.org för att transkribera dynamiska optimeringsproblem till storskaliga olinjära program med gles struktur. Denna struktur kan utnyttjas vid parallell lösning av det linjära KKT-systemet som löses i varje iteration i en inre-punkts-metod. De nuvarande lösarna för olinjära program i JModelica.org stödjer inte parallella algoritmer.

Optimeringsplattformen i JModelica.org har modifierats för att generera effektiv C-kod som kan länkas med en extern lösare för olinjära program som utför parallella beräkningar. CasADi, JModelica.org's tredjepartsmjukvara, används för att generera C-kod från modellekvationerna efter att kollokation har applicerats. Kompileringstiden har minskat med en faktor större än 20 och det tidigare problemet med ej kompilerbara filer har lösts.

Ett C++-gränssnitt implementerades för att länka de genererade filerna med det parallella ramverket. Ett storskaligt trajektorieoptimeringsproblem, såsom uppstartsoptimering av en modell av ett kraftverk, användes för att evaluera prestandan av gränssnittet och den parallella algoritmen. En uppsnabbningsfaktor på2.5 erhölls med 4 processor som delar minne.

Contents

Chapter 1

Introduction

Interest in simulating and optimizing dynamic problems has increased in the last decades. Both, the advancements in numerical methods for solving differential equations and the improvements of computational tools, have made possible the solution of large scale dynamic problems with considerable number of variables. Furthermore, the popularization of parallel computing in the last years has enhanced the formulation of fast algorithms for solving engineering problems. All of this have motivated us to integrate JModelica.org's dynamic optimization platform [1] with a C++ framework that uses the Message Passaging Interface (MPI) subroutines to solve non-linear programming (NLP) problems with an interior-point algorithm (IP) [2].

The integration strategy consisted in generating C-files in JModelica.org that can be later compiled with external solver packages such as the C++ framework [2]. This approach gives JModelica.org users flexibility to link the dynamic optimization platform with their own implementation, and solve the transcribed NLP with solvers not available within JModelica.org. In our case, we were interested in solving optimal control problems (OCP) with multiple processors. Therefore we compiled the generated files with the framework developed in [2] since it is capable of solving NLP problems in both shared and distributed memory architectures.

In this chapter we describe the overall goals of the master thesis and the research under consideration. We briefly introduce the advancements on strategies for dynamic process optimization. Unlike other optimization problems, dynamic optimization is constrained by a set of differential and algebraic equations (DAEs). For this reason, efficient algorithms are required in order to solve effectively the DAE constrained problems. In particular, in this project we consider "all-at-once" or *direct transcription* methods to tackle the optimization problem. Thus, through this chapter, we introduce background information and terminology used along this document.

1.1 Advancement of Numerical Methods for Optimal Control Problems

Optimal control seek for the solution of models described by dynamic equations in the form of ordinary differential equations (ODEs), differential-algebraic equations (DAEs) or partial differential equations (PDEs). Consider as an introductory example a basic OCP

$$\begin{aligned} \min \quad & \mathcal{J}(x(t), u(t)) \\ \text{s.t.} \quad & \tfrac{dx}{dt} - f(x(t), u(t)) = 0, \\ & x(t_0) = x_0 \end{aligned}$$

The idea is to determine the control signals u that will cause a system to satisfy the physical constraints $f(x, u) : \mathbb{R}^{n_x+n_u} \to \mathbb{R}^{n_x}$ and, at the same time, optimize some performance criterion $\mathcal{J}(x, u) : \mathbb{R}^{n_x+n_u} \to \mathbb{R}$. Where $x \in \mathbb{R}^{n_x}$ are known as the state variables and $u \in \mathbb{R}^{n_u}$ the control or input signals. A more general formulation will be given in Chapter 2.

Optimal control problems can be solved by two approaches. The first approach, known as the indirect or variational approach, is based on the solution of the first order necessary conditions for optimality obtained from Pontryagin's Maximum Principle. The second approach, converts the original infinite-dimensional problem into a discrete problem by fully or partially discretizing the system.

The indirect approach has solid theoretical foundations but notorious disadvantages. For problems without inequality constraints, one can formulate the first order optimality conditions and solve them as a two-point-boundary value problem. However, for problems with inequality constraints one is required to handle an active set of inequalities which is often complicated. In addition, since the analytical necessary conditions of optimality have to be derived analytically for every problem, and the resulting boundary value problem requires initial guesses for all variables, these approaches are less popular than direct approaches. Hence, in this thesis we opted for a direct approach and we do not go through indirect methods in detail. A review of these methods can be found however in [3]

Direct approaches transcribe the OCP into an NLP. Depending on the sort of discretization, these methods are classified as sequential or simultaneous. When only the control profiles are discretized, the methods are said to be sequential or control variable parametrized methods. The main advantage of using sequential methods relies in the fact that they generate smaller discrete problems than simultaneous methods. Nevertheless, sequential methods have the disadvantage that the DAE must be solved multiple times [4], while simultaneous methods on the other hand, solve the DAE only once. In addition, the KKT system of the resulting NLP

from the fully discretization method has the important feature of being a sparse structured system and this can be exploited by specialized sparse linear solvers.

1.1.1 Single-shooting Approach

The single shooting methods use an embedded DAE integrator in order to eliminate the continuous time dynamic system. The idea consist of parametrizing the continuous control trajectory $u(t)$ e.g. by polynomials or piecewise wise constants, and then pass discretized control to a DAE solver that integrates the DAEs over the entire time domain. Consider for instance, a discretized control function $u(t;q)$ where $q \in \mathcal{R}^{n_q}$ is the piecewise constant vector as illustrated in Figure 1.1

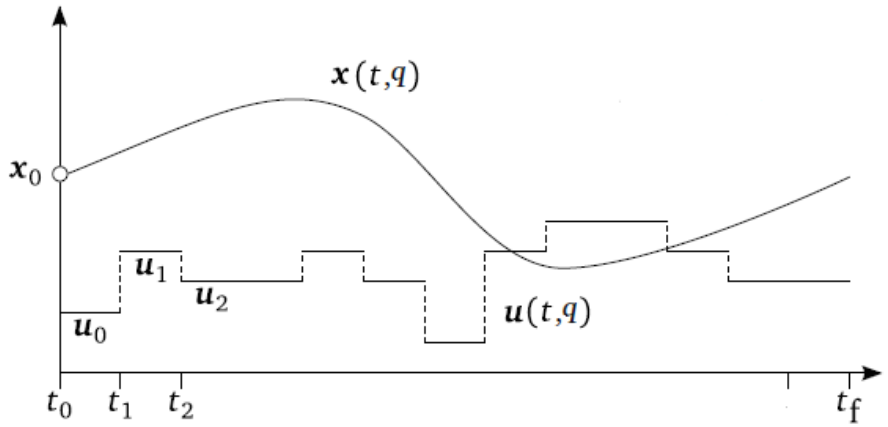


Figure 1.1. Illustration of the direct single shooting approach. Taken from [5]

Thus with the discretized control signals, the approximated states $x(t;q)$ are obtained by means of forward integration with an embeded DAE solver. This removes the DAE from the optimization problem and NLP solvers can be used to obtain a new guess of the controls.

Single shooting methods are often used in practice as the implementation is straightforward and the dimension of the NLP is relatively small. However, they suffer from a number of drawbacks. If the NLP solver requires first and second order gradient information to compute the search steps, additional time consuming sensitivity analysis must be done. On the same hand, repeated numerical integration of DAE model can be computationally expensive depending on the problem. For instance, the complexity of the shooting method does not scale well for problems with long time horizons.

An extension of the single-shooting approach is the multiple-shooting method which

is considered a hybrid between sequential and simultaneous methods. Further information regarding this method can be found in [5].

1.1.2 Simultaneous Approach

Another class of direct methods are the direct transcription methods, most notably the direct collocation. This approach approximates the continuous trajectories of the OCP through numerical quadrature schemes. Thus, the DAE model is discretized into a set of algebraic constraints, converting the OCP continuous problem into a large-scale NLP. This large-scale NLP needs to be solved by structure exploiting solvers, and commonly nonlinear interior point solvers such as IPOPT [6] are used.

Among the advantages of using direct collocation for solving an OCP, two attracted our attention. First, since the transcribed NLP is in completely algebraic form, first and second order derivative information can be efficiently computed through automatic differentiation tools, and considering that JModelica.org uses the third party software CasADi [7] to transcribe the optimal control problem, derivative information can be easily obtained. Secondly this discretization approach results in a sparse structure that can be exploited in parallel implementations. Consequently, we will focus on this approach in Chapter 3.

1.2 Thesis Overview

Chapter 2 describes the basic mathematical concepts that aid in the understanding of the direct transcription of an optimal control problem. The concepts of polynomial interpolation and non-linear constrained optimization are explored. Also, included in this chapter is a discussion of the interior point method for solving large scale non-linear programs. In addition, a basic introduction of the Schur-complement decomposition technique is given. Finally, a formulation of the continuous optimal control problem is presented.

In Chapter 3, a detailed description of the procedure used to transform the continuous dynamic problem into a discrete problem is given. Furthermore, in this chapter, we construct a discrete form for each of the terms of our optimal control problem.

Chapter 4 explains the method followed in this thesis to solve an OCP with code generation and parallel programming. It also presents the benchmark problems used throughout the thesis. It describes each of the models and shows the solution for each of the test problems.

Chapter 5 describes the code generation of the transcribed NLP. The concept of *checkpointing* is explained and a description of how to use it for transcribing the OCP in JModelica.org is given. Thus, in this chapter, we discuss how to use the

third party software CasADi effectively to generate compact and efficient C-Files.

Chapter 6 explains the parallel algorithm used for solving the dynamic optimization problem with multiple cores. We discuss the decomposition strategy proposed by D.Word et al in [2] to exploit the parallel structure of the dynamic problem. A brief explanation regarding the implementation is given. Finally, we present the results for one of the benchmark problems of Chapter 4.

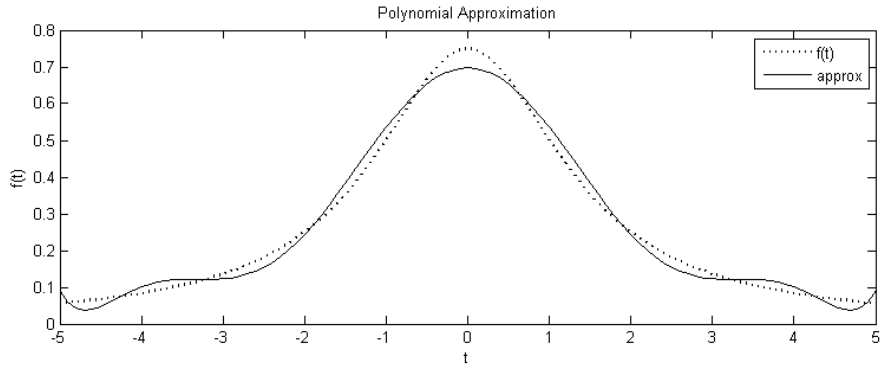
Finally, Chapter 7 gives an overall conclusion of the master's thesis project.

Chapter 2

Mathematical Background

In this chapter, we review some of the mathematical concepts that aid in the understanding of the direct transcription of an optimal control problem. The main objective of the transcription is to approximate an infinite-dimensional problem by a finite-dimensional one. Consequently, we first describe the concept of polynomial approximation since the trajectories of the optimal control problem are approximated by using a basis of Lagrange polynomials in the transcription procedure. Once the problem is transcribed, a nonlinear programming problem (NLP) must be solved, and since the NLP is solved through an interior point solver in this thesis, we introduce both the NLP formulation and the interior point method. Later, we discuss the basics of a Schur-complement decomposition that can be used to parallelize an interior point algorithm. Finally, a detailed review of the optimal control formulation is included.

2.1 Polynomial Approximation



The study and analysis of engineering problems often requires solving initial and

boundary value problems. A large variety of solution methods have been developed to help solve such classes of problems, e.g [8]. Choosing the most appropriate method depends heavily on the structure of the desired problem to solve. In the case of DAE-constrained optimization problems, the high accuracy, numerical stability, and convergence properties of orthogonal collocation methods [9], make this approach attractive for optimal control [10][11]. Since this thesis focuses on dynamic optimization, specifically in the collocation direct transcription method, this section describes the mathematics of the approximation methods used.

Collocation methods employ interpolating polynomials to approximate a function $\zeta(t)$ across an interval of time. Although this technique uses local orthogonal collocation, let us first consider a global approximation in order to understand the discretization technique, and then proceed to describe how to apply it to multiple finite elements within a time horizon. In an orthogonal collocation scheme, $\zeta(t)$ is approximated by using a family of polynomials defined at a set of collocation points $t_1, t_2, t_3, .., t_{n_c}$. The approximation can be represented in various ways, including power series, B-splines, or Lagrange interpolation, among many others. However, to transcribe an optimal control problem into an NLP, it is common to use a representation based on Lagrange interpolation polynomials [11]:

$$\zeta(t) \approx \tilde{\zeta}(t) = \sum_{i=1}^{n_c} \zeta(t_i) \cdot \ell_i(t)$$

$$\ell_i(t) = \prod_{j=1, j\neq i}^{n_c} \frac{t - t_j}{t_i - t_j}$$

Here $\tilde{\zeta}(t)$ is a $(n_c - 1)^{th}$ order polynomial approximation,$\zeta(t_i) \in \mathbb{R}$ the basis coefficient, and ell_i the ith Lagrange basis polynomial in the set of Lagrange interpolation polynomials. $\dot{\zeta}$ and $\dot{\ell}_i$ are the corresponding derivatives.

$$\dot{\zeta}(t) \approx \dot{\tilde{\zeta}}(t) = \sum_{i=1}^{n_c} \zeta(t_i) \cdot \dot{\ell}_i(t)$$

$$\dot{\ell}_i = \sum_{l=1 l\neq i}^{n_c} \left(\frac{1}{t_i - t_l} \prod_{j=1, j\neq i,l}^{n_c} \frac{t - t_j}{t_i - t_j} \right)$$

Lagrange polynomials have the property that

$$\ell_i(t_j) = \begin{cases} 1 & \text{if } i = j; \\ 0 & \text{if } i \neq j. \end{cases} \tag{2.1}$$

which results in the fact that $\zeta(t) = \tilde{\zeta}(t_i), i.e$, the function's approximation is equal to the true function at the collocation points. Thus, the idea consists of choosing a family of basis functions in such a way that the approximation $\tilde{\zeta}(t)$ matches the

true $\zeta(t)$ exactly at a set of collocation points $t_1, t_2, t_3, .., t_{n_c}$.

Even though the selection of the collocation points seems to be arbitrary, and one can naively or intuitively break the time interval at equidistant points, the position of the collocation points is not arbitrary. These are chosen to minimize the integral of the residuals between the approximation and the true function [12]. For instance for a uniform discretization scheme, the approximation error near the boundaries increases as the number of collocation points increase [13]. On the other hand, choosing the collocation points as the roots of Legendre and Chebyshev polynomials

$$P_K(\tau) = \frac{1}{2^K \cdot K!} \frac{d^K}{d\tau^K} \left[(\tau^2 - 1)^K \right]$$

guarantee that the approximation error monotonically decreases as the number of collocation points increases. Observe that $\tau \in [-1, 1]$ since the polynomials are orthogonal in that domain. However, we can shift the Legendre polynomials $P_K(2\tau - 1)$ to the domain $[0, 1]$

$$\tilde{P}_K(\tau) = \frac{1}{K!} \frac{d^K}{d\tau^K} \left[(\tau^2 - \tau)^K \right]$$

Thus, the collocation points in the time interval can be computed as $t_i = (t_f - t_0)\tau_i + t_0$. This leads to a Gauss-Legendre pseudospectral or a global scheme in which the collocation points lie within the interval (t_0, t_f). Consequently, solving an initial value problem reduces to finding the coefficients $\zeta(t_i) = \zeta_i$ of a system of equations. Consider the following initial value problem:

$$\begin{aligned} &\frac{dz}{dt} = f(t, z) \\ &z(0) = \bar{z_0} \end{aligned}$$

The discretization of the previous problem would look like:

$$\begin{aligned} &\sum_{i=0}^{n_c} z_i \dot{\ell}_i(t_j) = f(t_j, z_j), \qquad \forall j = 1, ..., n_c \\ &z_0 = \bar{z_0} \end{aligned}$$

Observe that this approach considers only one finite element, and within it the z_j are the variables of an algebraic system with n_c+1 equations. However, since we are interested in a local collocation scheme due to its property of local support, which bounds the error locally along the trajectory, we can divide our time interval into

n_e finite elements and represent the function $\zeta(t)$ with a polynomial approximation $\tilde{\zeta}_k(t)$ in each finite element. Further explanations about this procedure are given in Chapter 3.

2.2 Static Optimization NLP

Another very important concept in the understanding of how to solve a dynamic optimization problem is the formulation and solution of nonlinear programs (NLP). From a practical point of view, this kind of optimization consists of finding a set of parameters z that minimize or maximize an objective function, which could be the cost, profit or yield of a process. Such a set of parameters must also satisfy a set of algebraic equations which describe the behaviour of the process. These are called equality and inequality constraints of the optimization problem and define the limits for the optimal set of parameters. For a continuous variable optimization problem the NLP formulation can be written as

$$\begin{array}{ll} \min & f(z) \\ \text{s.t.} & g(z) \leq 0, \\ & h(z) = 0 \\ & Z_L \leq z \leq Z_U \end{array} \tag{2.2}$$

where $f(z) : \mathbb{R}^n \to \mathbb{R}$, $g(z) : \mathbb{R}^n \to \mathbb{R}^r$, and $h(z) : \mathbb{R}^n \to \mathbb{R}^m$, are assumed to be twice continuously differentiable $f(z), g(z), h(z) \in C^2$ [11]. The optimization problem is often also defined by introducing nonnegative slack variables $s \geq 0$ with the aim of shifting the nonlinearities away from the inequalities:

$$\begin{array}{ll} \min & f(z) \\ \text{s.t.} & g(z) + s = 0, \\ & h(z) = 0 \\ & Z_L \leq z \leq Z_U \\ & s \geq 0 \end{array} \tag{2.3}$$

Hence an equivalent formulation of (2.2), that will be of use in describing the interior point algorithm in Section 2.3 is as follows

$$\begin{array}{ll} \min & f(z) \\ \text{s.t.} & c(z) = 0 \\ & Z_L \leq z \leq Z_U \end{array} \tag{2.4}$$

Here $c(z) : \mathbb{R}^n \to \mathbb{R}^m$. Recalling (2.2) let us define the Lagrangian function $L(z, u, \lambda) = f(z) + g(z)^T u + h(z)^T \lambda$ where $u \in \mathbb{R}^r$ and $\lambda \in \mathbb{R}^m$ are called the dual variables and give different "weights" to the inequality and equality constraints. At the optimal point the solution must satisfy the first order necessary optimality or Karush-Kuhn-Tucker conditions

$$\nabla_x L(z^*, u^*, \lambda^*) = \nabla f(z^*) + \nabla g(z^*)^T u^* + \nabla h(x^*)^T \lambda^* = 0 \tag{2.5}$$

$$\begin{aligned} g(z^*) &\leq 0, \\ h(z^*) &= 0 \end{aligned} \tag{2.6}$$

$$\begin{aligned} g(z^*)^T u &= 0, \\ u^* &\geq 0 \end{aligned} \tag{2.7}$$

Equation (2.5) ensures the stationarity of the optimal point and is associated with the force balance in the kinematic interpretation proposed in [11]. Besides, the optimal solution must lie in a feasible region defined by both equality and inequality constraints (2.6). Additionally, (2.7) ensures complementarity meaning that inactive inequality constraints, which are irrelevant to the solution, have a dual variable equal to zero, and that the active inequality constraints "push" the solution to the interior of the feasible region. Another requirement for z^* to be an optimal point is that it must satisfy the concept of constraint qualification. One typical qualification is the linear independence constraint qualification (LICQ) which states that the active constraints $\nabla g_i(z^*)$ and $\nabla h(z^*)$ must be linearly independent.

Finally, through the second order sufficient optimality conditions (2.8), we ensure that the solution z^* is a local minimum and not a local maximum.

$$\begin{aligned} &\nabla p^T \nabla_{zz} L(z^*, u^*, \lambda^*) p \geq 0 \\ &\nabla h(z^*)^T p = 0 \\ &\nabla g_i(z^*)^T p = 0, \quad i \in \{i | g_i(z^*) = 0, u_i^* > 0\} \\ &\nabla g_i(z^*)^T p \leq 0, \quad i \in \{i | g_i(z^*) = 0, u_i^* = 0\} \, \forall p \neq 0 \end{aligned} \tag{2.8}$$

Here p represents all possible nonzero directions.

2.3 Interior Point Algorithm

In this thesis the solution method for solving the DAE constrained optimization problems consists of applying a direct transcription to a nonlinear program, and then solving it through an interior point algorithm. For this reason, we find it important to give a brief explanation to the interior point algorithm so that the reader gets a general idea of how the transcribed NLP is solved [6].

Consider a primal-dual barrier method to solve nonlinear problems of the form (2.4) with n variables and m equality constraints. To handle the lower and upper boundaries of z we form a barrier function and then compute a sequence of solutions to the barrier problem

$$\begin{array}{ll} \min & f(z) - \mu \sum_{i=1}^{n} \ln\left(Z_U^{(i)} - z^{(i)}\right) - \mu \sum_{i=1}^{n} \ln\left(z^{(i)} - Z_L^{(i)}\right) \\ \text{s.t.} & c(z) = 0 \end{array} \tag{2.9}$$

Here μ is the barrier parameter for a single barrier iteration that converges to zero as the number of iterations goes to infinity, and z^i is the i-th component of the vector of $z \in \mathbb{R}^n$. Note that this formulation enforces $z^{(i)} - Z_L^{(i)} > 0$ and $Z_U^{(i)} - z^{(i)} > 0$ in order to determine the solution vector. Similar to the previous section, we now define the Lagrangian function of the optimization subproblem (2.9)

$$L(z, \lambda) = f(z) - \mu \sum_{i=1}^{n} \ln\left(Z_U^{(i)} - z^{(i)}\right) - \mu \sum_{i=1}^{n} \ln\left(z^{(i)} - Z_L^{(i)}\right) + c(z)^T \lambda \tag{2.10}$$

The *primal* optimality conditions are then

$$\begin{aligned} \nabla f(z) + \mu(\overline{Z})^{-1} e - (\underline{Z})^{-1} e + \nabla c(z)^T \lambda = 0 \\ c(z) = 0 \end{aligned} \tag{2.11}$$

where $\overline{Z}$ =diag$(Z_U - z)$, $\underline{Z}$ =diag$(z - Z_L)$, and $e = \mathbb{1}$. At this point, one can see that through this approach there is no need to handle sets of active inequality constraints, and therefore Newton's method can be applied to solve the barrier subproblem. Defining the dual variables $\overline{d} = \mu(\overline{Z})^{-1} e$ and $\underline{d} = \mu(\underline{Z})^{-1} e$, we can substitute them in (2.11) and get the primal dual system

$$\begin{aligned} \nabla f(z) + \overline{d} - \underline{d} + \nabla c(z)^T \lambda = 0 \\ c(z) = 0 \\ \underline{Z}\underline{d} - \mu e = 0 \\ \overline{Z}\overline{d} - \mu e = 0 \end{aligned} \tag{2.12}$$

Newton's method relies on the generation of Newton steps from the following linear system:

$$\begin{bmatrix} \nabla^2_{zz} L(z^k, \lambda^k) & \nabla_z c(z^k) & -I & I \\ \nabla_z c(z^k)^T & 0 & 0 & 0 \\ \underline{D}^k & 0 & \underline{Z}^k & 0 \\ \overline{D}^k & 0 & 0 & \overline{Z}^k \end{bmatrix} \begin{bmatrix} \delta z^k \\ \delta \lambda^k \\ \delta \underline{d}^k \\ \delta \overline{d}^k \end{bmatrix} = - \begin{bmatrix} \nabla_z f(z^k) + \overline{d}^k - \underline{d}^k + \nabla_z c(z^k) \lambda^k \\ c(z^k) \\ \underline{Z}^k \underline{d}^k - \mu e \\ \overline{Z}^k \overline{d}^k - \mu e \end{bmatrix} \tag{2.13}$$

Here $\underline{D}^k$ =diag$(\underline{d}^k)$, $\overline{D}^k$ =diag$(\overline{d}^k)$ and δz^k, $\delta \lambda^k$, $\delta \underline{d}^k$, and $\delta \overline{d}^k$ are the respective steps for the corresponding variable. Multiplying the third block row by $(\underline{Z}^k)^{-1}$, the fourth block row by $(-\overline{Z}^k)^{-1}$, and adding these two rows we get the augmented form (2.14)

$$\begin{bmatrix} H^k & \nabla_z c(z^k) \\ [\nabla_z c(z^k)]^T & 0 \end{bmatrix} \begin{bmatrix} \Delta z^k \\ \Delta \lambda^k \end{bmatrix} = - \begin{bmatrix} \tilde{r}_z^k \\ c(z^k) \end{bmatrix} \tag{2.14}$$

Where,

$$H^k = \nabla^2_{zz} L(z^k) + (\underline{Z}^k)^{-1} \underline{D}^k + (\overline{Z}^k)^{-1} \overline{D}^k \tag{2.15}$$

and

$$\tilde{r}_z^k = \nabla_z f^k + \nabla_z c(z^k)\lambda - (\underline{Z}^k)^{-1}\mu e + (\overline{Z}^k)^{-1}\mu e \tag{2.16}$$

We will return to this augmented form later when we cover an efficient parallel solution for large-scale nonlinear dynamic optimization problems. Now that we know how to solve the subproblem (2.9), we just need to define a stopping criteria and an updating formula for μ.

$$E_\mu\left(z^*, \lambda^*, \underline{d}^*, \overline{d}^*\right) = \max \left\{ \begin{matrix} \|\nabla_z f(z^*) + \overline{d}^* - \underline{d}^* + \nabla_z c(z^*)\lambda^*\|_\infty, \\ \|c(z^*)\|_\infty, \\ \|\underline{Z}\underline{d}^* - \mu e\|_\infty, \\ \|\overline{Z}\overline{d}^* - \mu e\|_\infty \end{matrix} \right\} \leq \epsilon_{tol}$$

To conclude this section we present the interior point algorithm proposed in [11]

ALGORITHM

1 Choose constants $\epsilon_{tol} > 0, \kappa_\epsilon > 0, \kappa_\mu \in (0,1)$ and $\theta_\mu \in (1,2)$

2 **while** $E_\mu\left(\tilde{z}, \tilde{\lambda}, \tilde{\underline{d}}, \tilde{\overline{d}}\right) \geq \epsilon_{tol}$ *and* $l \geq 0$ **do**

3 Form the subproblem (2.9) for μ_l **while** $E_\mu\left(\tilde{z}, \tilde{\lambda}, \tilde{\underline{d}}, \tilde{\overline{d}}\right) \geq \kappa_\epsilon \mu_l$ **do**

4 Find the approximate solution $(\tilde{z}, \tilde{\lambda}, \tilde{\overline{d}}, \tilde{\underline{d}})$ by solving the linear system (2.12) using Newton's method with the step defined in (2.13)

5 **end**

6 Update the barrier parameter

$$\mu_l = \max\left\{\frac{\epsilon_{tol}}{10}, \min\left\{\kappa_\mu \mu_l, \mu_l^{\theta_\mu}\right\}\right\}$$

Update

$$l = l - 1$$

7 **end**

2.4 Schur Complement Decomposition

Let us introduce a sub-structuring technique for solving large-scale systems. The Schur complement method, a decomposition technique initially used for solving linear systems in limited memory computers [14] is nowadays used to exploit the parallelism of new computer architectures. The idea consists of sub-structuring a domain into sub-domains. Consider the linear system $Mz = f$ where $M \in \mathbb{R}^{n\times n}$ is a 2×2 block matrix, $z \in \mathbb{R}^n$ and $f \in \mathbb{R}^n$

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix}$$

Substructuring the linear system

$$\begin{aligned} Ax + By &= c \\ Cx + Dy &= d \end{aligned}$$

We can solve the second equation for

$$y = D^{-1}\left(d - Cx\right)$$

Substituting in the first equation

$$\left(A - BD^{-1}C\right)x = c - BD^{-1}d$$

The matrix $\left(A - BD^{-1}C\right)$ is called the Schur complement of D and if it is invertible, the solution of the linear system is as follows:

$$\begin{aligned} x &= \left(A - BD^{-1}C\right)^{-1}\left(c - BD^{-1}d\right) \\ y &= D^{-1}\left(d - C\left(A - BD^{-1}C\right)^{-1}\left(c - BD^{-1}d\right)\right) \end{aligned}$$

We will retake this topic in Chapter 6 when we talk about the parallel interior point algorithm proposed in [2].

2.5 Optimal Control Problem

Consider the optimization problem where the objective is to find the variables $z(t) \in \mathbb{R}^{n_z}$ on the interval $[t_0, t_f]$, and the set of parameters $p \in \mathbb{R}^{n_p}$ that minimizes a cost function

$$\mathcal{J}(z,p) = \Phi(z(t_f),p) + \int_{t_0}^{t_f} \mathscr{L}(t,z(t),p)dt \tag{2.17}$$

Here $\Phi : \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}$ is known as the *Mayer term* and $\mathscr{L} : \mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p} \to \mathbb{R}$

is the *Lagrange term*. Both Φ and $\mathscr{L} \in C^2$. Equation (2.17) is said to be of the *Bolza* form. Special cases of the Bolza problem are when either the Mayer term $\Phi(z(t_f), p) := 0$ or the Lagrange term $\mathscr{L}(t, z(t), p) := 0$. Moreover, the three cases are equivalent in the sense that each of them can be converted to any other. Given a Bolza problem, the objective function can be reduced into a Mayer type one. This can be done by introducing an additional variable

$$x_L(t) = \int_{t_0}^{t} \mathscr{L}(t, z(t), p)dt$$

and an additional ODE constraint

$$\begin{aligned} \tfrac{dx_L(t)}{dt} &= \mathscr{L}(t, z(t), p) \\ x_L(t_0) &= 0 \end{aligned}$$

so that the *Bolza* objective is converted to

$$\mathcal{J}(z, p) = \hat{\Phi}(z(t_f), p) + x_L(t_f) = \Phi(\tilde{z}(t_f), p)$$

Conversely, *Bolza* problems can be transformed into *Lagrange* problems by introducing an additional variable

$$x_L(t) = \frac{1}{t_f - t_0}\Phi(z(t_f), p)$$

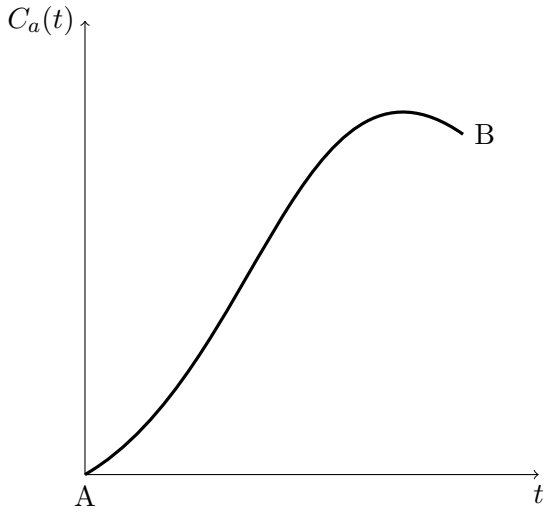
and the additional ODE constraint

$$\begin{aligned} \tfrac{dx_L(t)}{dt} &= 0 \\ x_L(t_0) &= \tfrac{1}{t_f - t_0}\phi(z(t_f), p) \end{aligned}$$

which converts the *Bolza* objective into

$$\mathcal{J}(z, p) = \int_{t_0}^{t_f} x_L(t) + \hat{\mathscr{L}}(t, z(t), p)dt = \int_{t_0}^{t_f} \mathscr{L}(t, \tilde{z}(t), p)dt$$

Thus far, we have treated the variables of our optimization problem as if they were all of the same kind. Notwithstanding, in optimal control problems, variables can be separated into three classes, namely the state variables, the control or input variables, and the algebraic variables. As an example, consider the chemical engineering problem of a batch reactor in which, from a prescribed initial state, one may be interested in finding a feasible control (one that satisfies the physical constraints) of the reactor's cooling temperature such that the production of a certain substance is maximized. This answers the question of how to transfer the chemical system from an initial state A to a desired final state B.



Hence, we are looking for a feasible profile of the control variable $T_c(t)$ that maximizes one of our states $C_a(t)$ at a time t_B, starting from a time t_A. Here, one can see how the *Mayer* problem presented at the beginning of this section, fits perfectly to formulate our example. The $z(t)$ variables from (2.17) turn out to be

$$z := (\dot{x}, x, w, u)$$
$$n_z := 2n_x + n_u + n_w$$

where $x \in \mathbb{R}^{n_x}$ is the vector of states, $w \in \mathbb{R}^{n_w}$ the vector of algebraic variables, and $u \in \mathbb{R}^{n_u}$ the vector of inputs or control variables. With these newly introduced variables and departing from our introductory chemical reactor example, we can proceed to define our OCP. In general, the main goal of an optimal control problem is to determine the control signals u that will cause a system to satisfy the physical constraints $F(t, \dot{x}(t), x(t), w(t), u(t))$, while minimizing or maximizing some performance criterion $\mathcal{J}(x, u, w, p)$.

We look for a model that adequately predicts the response of the physical system to changes in the inputs. Therefore, we shall model the system with an index one DAE. In this thesis, we restrict ourselves to an index one or zero system

$$F(t, \dot{x}(t), x(t), w(t), u(t)) = 0 \quad \forall t \in [t_0, t_f] \tag{2.18}$$

Here $F : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_w} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x + n_w}$ differential algebraic system, and a solution of (2.18) represents the response of the system to a change in the control u for an initial state $x(t_0) = x_0$. One difference from an *ordinary differential equation system* (ODE) is that the imposed initial conditions need to be consistent with a set of algebraic equations at t_0.

$$F(t, x(t_0), \dot{x}(t_0), w(t_0), u(t_0)) = 0$$

Thus, imposing an arbitrary initial condition may result in an unsolvable problem. This covers all that we need to understand the OCP we solve in this thesis work. See [9][15] for further information about DAE systems.

Apart from the physical model, a great variaty of constraints may be imposed in the optimal control problem. Among them, we will discuss *point constraints*, *path constraints* and *isoperimetric constraints*.

Point constraints are often part of optimal control problems. The more frequent ones are the terminal constraints, like the one imposed as $x(t_f) = x_f$. In general terminal constraints force the system response to a certain terminal range

$$\psi(x(t_f), \dot{x}(t_f), w(t_f), u(t_f), p) \leq 0 \tag{2.19}$$

Path constraints are encountered in many optimal control problems. With them, mixed functions of state, algebraic and control variables can be restricted to a defined range over an interval $[t_a, t_b] \subset [t_0, t_f]$. They are defined as follows:

$$\phi(x(t), \dot{x}(t), w(t), u(t), p) \leq 0 \quad \forall t \in [t_a, t_b] \tag{2.20}$$

Constraints in which only states are involved can also appear in the optimal control problem.

$$x^{(k)}(t) \leq x_U \quad \forall t \in [t_a, t_b]$$

Isoperimetric constraints are perhaps more difficult to handle than the previous two and are not often seen in optimal control problems. Even though we do not solve problems with these kind of constraints in this thesis, we introduce them so that the reader gets a brief idea of how they look. Isoperimetric constraints involve the integral of a given function $g_i(t, x(t), w(t), u(t))$ over a time interval $[t_a, t_b] \subset [t_0, t_f]$

$$\int_{t_a}^{t_b} g_i(t, x, w, u) dt \leq C$$

where $C \in \mathbb{R}$. Note that isoperimetric constraints can be converted into equivalent point constraints in a similar way to what we did to the *Lagrange* term of (2.17).

Having examined each of the elements of our optimal control problem we can formulate the general OCP

$$
\begin{aligned}
\min \quad & \mathcal{J}(z(t), p) \\
\text{s.t.} \quad & F(t, z(t), p) = 0, \\
& F_0(t_0, z(t_0), p) = 0 \\
& Z_L \leq z(t) \leq Z_U \qquad \forall t \in [t_0, t_f] \\
& P_L \leq p \leq P_U \\
& \psi(t_a, z(t_a), p) \leq 0 \\
& \phi(t, z(t), p) \leq 0 \\
& \int_{t_a}^{t_b} g_i(t, z(t), p) dt \leq C
\end{aligned}
\tag{2.21}
$$

Chapter 3

Transcription of Dynamic Optimization Problems

In Chapter 2, we introduced some essential concepts for the understanding of the transcription and solution approaches we use for solving optimal control problems. In this chapter, we use all the theoretical concepts previously described, to transcribe the continuous problem into a discrete one, that can later be solved with the interior point method. We start with a brief introduction of a standard discretization approach based on orthogonal collocation on finite elements. We continue with the construction of the discrete form of each of the terms in our optimal control problem (2.21).

3.1 Discretization Technique

Let us consider the optimal control problem as an infinite-dimensional extension of the NLP problem. Since numerical solvers require a finite number of variables and equations we aim to transcribe or convert an infinite-dimensional problem into a finite dimensional one through an approximation. According to [16], the first step for the transcription consists of converting the dynamic system into a problem with a finite set of variables, which we focus on in this section.

Assume that Figure 3.1 represents one of the states $x(t)$ of an optimal control problem, and that the solid line is the continuous trajectory of it. Geometrically, we are looking for a finite number of points that accurately approximate the continuous trajectory of the solid line. To do so, a great variety of numerical methods have been developed in the last decades [16][9][17].Due to its high accuracy and numerical stability properties, the orthogonal collocation on finite elements method is used.

As an introduction to the method, consider the chemical engineering example of optimizing the design of a set of tubular reactors connected in series. Once the reaction process is in the steady state phase, the state variables $x(t)$ evolve over

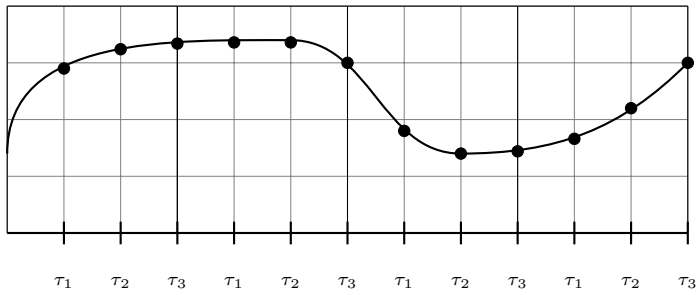


Figure 3.1. Collocation in finite elements scheme

distance but not time. Disregarding for now the mathematical model that describe each of the states, assume that it simply involves a DAE system. In Figure 3.2 each tubular reactor, except for the first one, is fed with the outlet of the reactor that precedes it. From that point of view each reactor is described by a certain set of equations plus two that connect it with its neighbours. This is precisely the case of the collocation on finite element method. Each reactor represents a finite element and within the element the set of collocation points determines the mathematical model which describes it. The collocation points in all elements do not necessarily need to be the same, since the reaction system of each tubular reactor in the process does not have the same reaction mechanism. In this sense, each finite element may be treated differently. However, all finite elements are similar in that they are coupled through the continuity equations, where the inlet and outlet streams of the tubular reactors link them together.

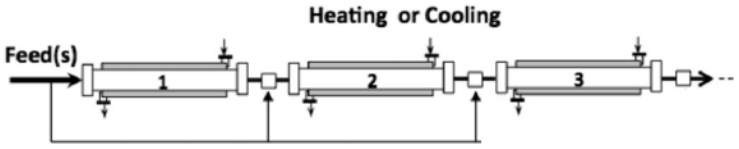


Figure 3.2. Tubular reactors. Taken from [11]

To understand the mechanics of the approximation scheme, assume that in Figure 3.1 the time interval $[t_0, t_f]$ is divided into four finite elements $[t_i, t_{i+1}]$ for $i = 1, 2, \cdots, 4$, and that in each finite element a polynomial approximation like the one presented in chapter 2 is used. Moreover, we transform the time domain at each element to $\tau \in [0, 1]$ by defining $h_i = t_i - t_{i-1}$ and $t_i(\tau) = t_{i-1} + h_i \cdot \tau$. For simplicity, let us also assume that $h_i = h$ for $i = 1, 2, \cdots, 4$. Hence, in each finite element a set of $\tau_k = 1, 2, \cdots, n_c$ collocation points is chosen in such a way that the approximated element $\tilde{x}_i(t)$ matches the true state $x(t)$ at the set of collocation points exactly as discussed e.g in [12].

Recall now the variables $z(t) : (\dot{x}, x, w, u)$ of our optimal control problem (2.21). We look for a Lagrange polynomial based approximation of our variable trajectories

$$\upsilon(t) := (x(t), w(t), u(t)), \quad \forall t \in [t_0, t_f]$$

in each finite element $i \in i = 1, ..., n_e$

$$\upsilon_i(\tau) := (x_i(\tau), w_i(\tau), u_i(\tau)) \in C^\infty \left(\mathbb{R}, \mathbb{R}^{n_x} \times \mathbb{R}^{n_w} \times \mathbb{R}^{n_u}\right)$$

Here we choose $n_c + 1$ interpolation points in the i-th element and approximate the trajectory variables as follows:

$$\tilde{\upsilon}_i(\tau) = \sum_{k=0}^{n_c} \upsilon_{i,k} \cdot \tilde{\ell}_k(\tau) \tag{3.1}$$

where

$$\tilde{\ell}_k(\tau) = \prod_{l=0, l\neq k}^{n_c} \frac{\tau - \tau_l}{\tau_k - \tau_l}$$

Equivalently, the time derivative of the state variables can be represented as a Lagrange polynomial

$$\dot{\tilde{x}}_i(\tau) = \frac{dx_i(\tau)}{dt} = \frac{dx_i(\tau)}{d\tau}\frac{d\tau}{dt} = \frac{1}{h_i}\sum_{k=0}^{n_c} x_{i,k} \cdot \dot{\tilde{\ell}}_k(\tau)$$

Note that we enforce continuity of our trajectories by including the interpolation point $k = 0$. Nevertheless, only the states are required to be continuous, but control or algebraic variables may be either continuous or discontinuous depending on the problem. For further information regarding continuity refer to [15].

Thus far, we know how to approximate the variables $\tilde{z}_i(\tau) : (\dot{\tilde{x}}_i, \tilde{\upsilon}_i(\tau))$ of the optimal control problem within a finite element, but we still need to discretize the time variable τ. Thus, the next step consists of selecting a set of discrete points that gives us the most accurate and stable approximation. Because of the properties of the collocation points, we choose the set of collocation points as our discrete points. Let us now discuss some approaches commonly found in literature.

As mentioned in Chapter 2, the selection of the collocation points is of great importance. The convergence, accuracy and stability properties of the method depend on the set of collocation points selected [9]. Since we are using orthogonal collocation,

the interpolation points that we use are chosen as the roots of orthogonal polynomials. In this thesis we choose the roots of the shifted Gauss-Jacobi polynomial of degree $K = n_c - \alpha - \beta$, described by the following equation:

$$P_K^{(\alpha,\beta)} = \sum_{j=0}^{K} (-1)^{K-1} \cdot \tau^j \cdot \gamma_j$$

where

$$\gamma_j = \begin{cases} 1 & \text{if } j = 0 \\ \frac{(K-j+1)\cdot(K+j+\alpha+\beta)}{j\cdot(j+\beta)} & \text{if } j = 1, \cdots, K \end{cases}$$

Here α and β lead to three possible cases:

- $\alpha = 0, \beta = 0$ leads to the roots of the Legendre polynomial $\tilde{P}_K$ from Chapter 2, and hence this is called the *Gauss-Legendre* (LG) collocation.
- $\alpha = 1, \beta = 0$ leads to the roots of $\tilde{P}_K = \tilde{P}_K(\tau) - \tilde{P}_{K-1}(\tau)$ and it is called *Gauss-Radau* (LGR) collocation.
- $\alpha = 1, \beta = 1$ leads to the roots of $(1 - \tau^2)\dot{\tilde{P}}_{K-1}(\tau)$ and it is called the *Gauss-Lobatto* (LGL) collocation.

The three sets of points are obtained from Legendre polynomials or through linear combinations of them or their derivatives. Also, since we are working with the shifted polynomials, the sets of points are defined in the interval $[0, 1]$, but depending on the collocation case, either one, two or neither of the end points is included in the set. Even though it sounds like a slight difference, the inclusion of the end points or not implies many changes in the approximation method. In this thesis, we will elaborate on the first two cases, not only because of their accuracy and stability properties, but also because of their compatibility with the NLP formulation resulting from the transcription.

The convergence properties of Gauss and Radau methods have been studied and presented in numerous documents. In [18] and [19], the convergence rates of the transcription of an unconstrained optimal control problem are analysed. Higher order convergence for the state variables are proved. Both [20] and [21], discuss approaches for proving convergence of discrete approximations and present, for instance, results of an implicit euler discretization. Runge-Kutta convergence results for optimal control problems is studied in [22]. In [23], an study of the effects of increase in the number of collocation points is presented. Finally, in [10] the convergence of Radau collocation in optimal control problems is discussed.

3.1.1 Radau Collocation

The set of collocation points τ_k in the Radau collocation are obtained from the roots of $\tilde{P}_K = \tilde{P}_K(\tau) - \tilde{P}_{K-1}(\tau)$. Contrary to LG points, Radau points are not unique, given that either of the end points can be included. In this thesis, we have chosen $\tau_{n_c} = 1$ so that the set of points are in the semi-open interval $\tau_k \in (0, 1]$ for $1, \cdots, n_c$. By forcing the terminal end point to lie in the boundary, one degree of freedom is reduced, and therefore the truncation error becomes $\mathscr{O}(h^{(2 \cdot n_c - 1)})$. This is commonly called *flipped* LGR. In Table 3.1 we present the numeric values of the collocation points of a Radau collocation.

Table 3.1. Radau collocation points

	τ_1	τ_2	τ_3	τ_4	τ_5
$n_c = 1$	1.00000	N/A	N/A	N/A	N/A
$n_c = 2$	0.33333	1.00000	N/A	N/A	N/A
$n_c = 3$	0.15505	0.64449	1.00000	N/A	N/A
$n_c = 4$	0.08858	0.40946	0.78765	1.00000	N/A
$n_c = 5$	0.05710	0.27684	0.58359	0.86024	1.00000

Despite the Radau method lacking symmetry, it is commonly used for solving optimal control problems through direct collocation. Moreover, LGR is one degree less precise than LG, but in many cases LGR is still preferred over other collocation schemes. The reader may wonder what makes LGR so special, that even though it presents important disadvantages, it is still preferred over other collocation methods in solving optimal control problems. There is not a single answer to that question, but in this thesis we mention a couple of them.

First, Radau collocation has the advantage that no additional extrapolations are required. Thanks to the definition of τ_{n_c} at the terminal point, the mesh points can be easily obtained:

$$\upsilon_{i,0} = \upsilon_{i-1}(\tau_{n_c}) = \sum_{k=0}^{n_c} \upsilon_{i-1,k} \cdot \tilde{\ell}_k(\tau_{n_c}) = \upsilon_{i-1,n_c}$$

This is one of the reasons why Radau is so commonly used in optimal control. As we discussed in the previous section, each finite element solves its own DAE system of equations independently, but it is coupled to its neighbouring elements by means of the continuity equations. Choosing the Radau collocation points makes it easier for this coupling, as it only requires the equalizing of the $\upsilon_i, 0$ with the terminal collocation point of the previous element υ_{i-1,n_c}. This is represented graphically in Figure 3.3.

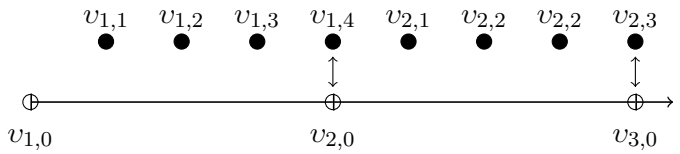


Figure 3.3. Radau collocation scheme.The vertical double arrows represent graphically the continuity equations. Note that the initial point $v_{1,0}$ must be equated to the initial condition of the DAE.

In the same way, the transcription of the Mayer term in the objective function is straightforward with Radau, since one of the collocation points is located at t_f

$$\Phi(z(t_f), p) = \Phi(z_{n_e,n_c}, p)$$

Additionally, for certain problems Gauss collocation produces oscillating profiles while Radau collocation leads to smoother profiles. According to [10], Radau collocation stabilizes the system of equations more efficiently in the presence of high index DAEs, and stiff ODEs. For this reason, Radau is used widely in chemical process control [11]. Furthermore, this scheme is recommended as the method of choice for solving initial value DAEs and for very stiff ODEs [9].

Lastly, both Radau and Gauss collocation are A-stable methods. However, Radau is also L-stable, meaning that it can quickly damp out the transient response of very stiff components in a differential equation [24].

For further information regarding Radau methods, we recommend the reader to look first for the low order Runge-Kutta methods called Radau methods, such as the Euler implicit method [17],[8].

3.1.2 Gauss Collocation

Another part of the Runge-Kutta methods are the well known Gauss collocation methods. The collocation points τ_k in LG are obtained as the roots of the shifted Gauss-Jacobi polynomial, and lie in the open interval $(0, 1)$. Therefore, in contrast to Lobatto and Radau methods, Gauss schemes impose the collocation strictly in the interior of the interval. Tables 3.2 and 3.3 present the numeric values of the collocation points and quadrature weights of a problem having up to 5 collocation points.
Note in Table 3.2 that the collocation points are symmetric around the midpoint $\tau_{mid} = 0.5$ of the interval, and thus Gauss, apart from having the highest precision $\mathcal{O}(h^{(2 \cdot n_c)})$ is considered a symmetric method. The quadrature weights are determined as

Table 3.2. Gauss collocation points

	τ_1	τ_2	τ_3	τ_4	τ_5
$n_c = 1$	0.50000	N/A	N/A	N/A	N/A
$n_c = 2$	0.21132	0.78867	N/A	N/A	N/A
$n_c = 3$	0.11270	0.50000	0.88729	N/A	N/A
$n_c = 4$	0.06943	0.33001	0.66999	0.93057	N/A
$n_c = 5$	0.04691	0.23076	0.50000	0.76923	0.95308

$$w_i = \int_{-1}^{1} \ell_i(\tau) d\tau = \frac{2}{(1-\tau^2)\left[\dot{P}_K(\tau_i)\right]^2}$$

Table 3.3. Gauss Quadrature weights

	τ_1	τ_2	τ_3	τ_4	τ_5
$n_c = 1$	1.00000	N/A	N/A	N/A	N/A
$n_c = 2$	0.50000	0.50000	N/A	N/A	N/A
$n_c = 3$	0.27777	0.44444	0.27777	N/A	N/A
$n_c = 4$	0.17392	0.32607	0.32607	0.17392	N/A
$n_c = 5$	0.11846	0.23931	0.28444	0.239314	0.11846

Since Gauss collocation does not have a collocation point at the end of each element, the continuity equations require an additional computation in order to couple or link two elements.

$$v_{i,0} = \sum_{k=0}^{n_c} v_{i-1,k} \cdot \tilde{\ell}_k(1)$$

A geometrical scheme of how this linking is done, is presented in Figure 3.4. There, one can see that in order to couple two elements, one requires all the interpolation points of the previous element. An alternative to avoid this computation, relies on the quadrature weights, but one can still see that coupling the elements is more complicated than the straightforward coupling offered with Radau collocation.

$$v_{i,0} \approx v_{i-1,0} + h_{i-1} \sum_{k=1}^{n_c} \omega_k \cdot v_{i-1,k}$$

Another disadvantage of not having a collocation point at the end point of each

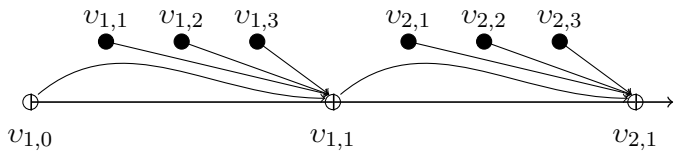


Figure 3.4. Gauss collocation scheme. The curve arrows represent graphically the continuity equations. Note that the initial point $\upsilon_{1,0}$ must be equated to the initial condition of the DAE

element is that the Mayer term of the objective function requires an extrapolation. In order to determine the value of the trajectories at the end of the time horizon, one must approximate them in a similar way as we did with the continuity equations. Thus, the transcription of the Mayer term

$$\Phi(z(t_f), p) = \Phi(\sum_{k=0}^{n_c} \upsilon_{n_e,k} \cdot \tilde{\ell}_k(1), p)$$

Also different from Radau, Gauss scheme does not have a collocation point at the mesh points. This slight difference implies that Gauss collocation does not satisfy the DAE constraints at the mesh points, which is considered to degrade the performance of the method for systems with high index. Although we do not investigate this fact in this thesis, since JModelica.org only consider index-1 problems, a variety of alternatives have been proposed to overcome this problem [25].

For further information regarding Gauss methods, we recommend the reader to look first for the low order Runge-Kutta methods, such as the mid point rule which is the simplest Gauss collocation scheme [17],[8], [16].

Based in this discretization technique, in which the time domain is divided in n_e finite elements, we proceed to describe how to construct the discrete form of each of the optimal control problem terms (2.21) according to the approximated trajectories $\tilde{z}_i(\tau)$ and the n_c collocation points τ_k within each element.

3.2 Transcription of the Objective Function

As discussed in the previous section, the transcription of the Mayer term of the objective function varies depending on the collocation scheme. However, for the Lagrange term $\mathscr{L}$ we use Gaussian quadrature in order to integrate our optimization variables $z(t)$ with the highest accuracy [11].

$$\begin{aligned}\int_{t_0}^{t_f} \mathscr{L}(t, z(t), p)dt &\approx \sum_{i=1}^{n_e} \left(h_i \int_0^1 \mathscr{L}(t_i(\tau), \tilde{z}_i(\tau), p)d\tau \right) \\ &\approx \sum_{i=1}^{n_e} \left(h_i \sum_{k=1}^{n_c} \omega_k \cdot \mathscr{L}(t_i(\tau_k), \tilde{z}_i(\tau_k), p) \right) \\ &= \sum_{i=1}^{n_e} \left(h_i \sum_{k=1}^{n_c} \omega_k \cdot \mathscr{L}(t_i(\tau_k), z_{i,k}, p) \right)\end{aligned}$$

Note that the last equality holds because of the collocation property (2.1) and that simplifies considerably the algebraic expression. Note also that the integration does not include the mesh points $z_{i,0}$, which is due to the fact that non-continuous approximations are not discretized at this point. Further information about this is in [15].

3.3 DAE Constraints Transcription

To explain how to transcribe the DAE constrained system, let us first write $F(t, z(t), p)$ into a simple way, in order to make the substitution of the variables explicit. Consider $f_d : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_w} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ as the set of ordinary differential equations, and $f_a : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_w} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_w}$ the set of algebraic equations. Hence, the DAE constraints in (2.21)

$$\begin{aligned}&\dot{x} = f_d(t, x(t), w(t), u(t), p) \\ &0 = f_a(x(t), w(t), u(t), p) \\ &x(t_0) = \bar{x}_0\end{aligned}$$

Substituting the approximation profiles $\tilde{z}_i(\tau)$ and evaluating in the set of collocation points τ_k

$$\begin{aligned}&\dot{\tilde{x}}_i(\tau_k) = f_d(t_i(\tau_k), \tilde{x}_i(\tau_k), \tilde{w}_i(\tau_k), \tilde{u}_i(\tau_k), p) \\ &0 = f_a(\tilde{x}_i(\tau_k), \tilde{w}_i(\tau_k), \tilde{u}_i(\tau_k), p) \\ &x_{1,0} = \bar{x}_0 \\ &x_0(0) = x(t_0), \\ &\forall (i,k) \in ([1..n_e] \times [1..n_c]) \\ &\sum_{j=0}^{n_c} x_{i,j} \cdot \tilde{\ell}_j(0) = \sum_{j=0}^{n_c} x_{i-1,j} \cdot \tilde{\ell}_j(1) \quad i = 1, \cdots, n_e\end{aligned}$$

Observe that because we use the approximated trajectories $\tilde{v}_i$, an additional set of n_e equations must be included. These are the continuity equations that link the neighbours together. Finally, recalling the collocation property (2.1) we obtain

$$
\begin{aligned}
&\tilde{\dot{x}}_i(\tau_k) = f_d(t_i(\tau_k), x_{i,k}, w_{i,k}, u_{i,k}, p) \\
&0 = f_a(x_{i,k}, w_{i,k}, u_{i,k}, p) \\
&x_{1,0} = \bar{x}_0 \\
&x_0(0) = x(t_0), \\
&\forall (i,k) \in ([1..n_e] \times [1..n_c]) \\
&\sum_{j=0}^{n_c} x_{i,j} \cdot \tilde{\ell}_j(0) = \sum_{j=0}^{n_c} x_{i-1,j} \cdot \tilde{\ell}_j(1) \quad i = 1, \cdots, n_e
\end{aligned}
$$

Thus, we enforce the DAE system only at the collocation points and we couple the elements with the continuity equations. Other kinds of approximation schemes are based on the so called monomial basis [12],[11]. On the other hand, one can consider the derivatives $\dot{x}$ as variables of the optimization problem and append the following collocation equations to the set of constraints. This way the $F(t, z(t), p)$ from the OCP can be evaluated directly. This is the approach JModelica.org follows.

$$
\dot{x}_{j,l} - \frac{1}{h_j} \sum_{m=0}^{n_c} x_{j,m} \cdot \dot{\tilde{\ell}}_{m,l} = 0, \quad \forall (j,l) \in ([1, \cdots, n_e] \times [1, \cdots, n_c])
$$

3.4 Path and Point Constraints

We have now described the transcription of the Bolza problem and the DAE constraints. These two elements compose the simplest dynamic optimization problem. Nevertheless, since in JModelica.org more general problems including path and point constraints $\phi(t, z(t))$, $\psi(t_a, z(t_a))$ can be solved, we will explain briefly how to deal with these sorts of restrictions.
Again there is more than one way to transcribe the path constraints and variable bounds [15]. In this thesis, we choose to enforce these constraints at the collocation points

$$
\begin{aligned}
&\phi(t_i(\tau_k), x_{i,k}, w_{i,k}, u_{i,k}) \leq 0 \\
&X_L \leq x_{i,k} \leq X_U \\
&W_L \leq w_{i,k} \leq W_U \\
&U_L \leq u_{i,k} \leq U_U
\end{aligned}
$$

Point constraints, on the other hand, are not straight forward to transcribe. In the best case, the constrained time point may coincide with one of the collocation points. In that situation, we just need to enforce that point constraint at that collocation point. However, if that is not the case we have two possibilities. The first alternative consists of using our approximated trajectory polynomial $\tilde{v}_i(t_i(\tau_a))$ to enforce the solution to satisfy the point constraint. The second alternative, and

the most attractive from our point of view, due to the fact that JModelica.org follows it, consists of mapping the constrained point to the nearest collocation point, and then restricting at the mapped point.

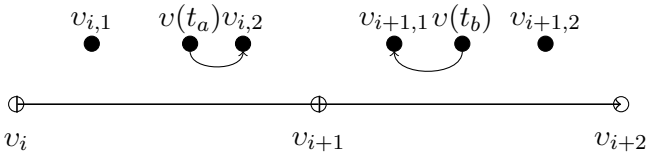


Figure 3.5. Point mapping for point constraints.

3.5 Concluding Remarks

The fundamental steps required for the direct transcription of an optimal control problem are now in place. The approach divides the time horizon into n_e elements where the states, inputs, and algebraic variables are approximated by means of a collocation method. The basic notion is quite simple: the differential equations placed in the constraints are replaced by a finite set of algebraic constraints, and as a result, the optimal control constraints are replaced by NLP constraints. Accordingly, this approach is usually called full discretization in literature.

In particular, the control variables are often parametrized with piecewise constant or piecewise linear controls, but more generally they are approximated in the same way the states and algebraics are approximated, with Lagrange polynomials but also monomial representation [12]. By using either representation, the problem (2.21) can be converted into the following form:

$$
\begin{aligned}
\min \quad & \mathcal{J}(z_{i,k}, p) \\
\text{s.t.} \quad & F(t, z_{i,k}, p) = 0, \\
& F_0(t, z_{1,0}, p) = 0 \\
& Z_L \leq z_{i,k} \leq Z_U \\
& P_L \leq p \leq P_U \\
& \psi(t_{i,k}, z_{i,k}, p) \leq 0 \\
& \phi(t, z_{i,k}, p) \leq 0 \\
& \dot{x}_{i,k} - \tfrac{1}{h_i} \sum_{l=0}^{n_c} x_{i,l} \cdot \dot{\tilde{\ell}}_{l,k} = 0 \\
& \forall (i,k) \in ([1, \cdots, n_e] \times [1, \cdots, n_c]) \\
& \sum_{l=0}^{n_c} x_{i,l} \cdot \tilde{\ell}_l(0) = \sum_{l=0}^{n_c} x_{i-1,l} \cdot \tilde{\ell}_l(1) \quad i = 1, \cdots, n_e
\end{aligned}
\tag{3.2}
$$

Observe that the differential equations are discretized in n_c equality constraints at each element $i = 1, 2, \cdots, n_e$, and that additional conditions are imposed so that

the state variables are continuous at the junctions between elements. Note also, that inequality constraints are discretized into a finite number of constraints, which must hold at the collocation points. Finally, the transcription results in an NLP problem with $n_e \cdot (n_x + n_c \cdot (2 \cdot n_x + n_u + n_w))$ variables that can be solved for instance with an interior point method.

Chapter 4

Solution Approach of the Optimal Control Problem

Along this chapter, we describe the procedure proposed to achieve the goal of solving an optimal control problem by using JModelca.org's optimization framework, and a parallel algorithm. First, we explain the strategy followed to successfully link JModelica.org with the package developed in [2]. Then we present the benchmark problems used to test both the code generation and the parallel algorithm. We describe the model equations and present the solutions of the benchmark problems.

4.1 Solving Strategy

Chapters 2 and 3 provide the theoretical knowledge and numerical procedure required to transcribe an OCP into a large-scale NLP by following a simultaneous approach based on local finite element orthogonal collocation. Both, the transcription and solution can be obtained within JModelica.org. First, Modelica and optimica compilers get the continuous problem formulated in Section 2.5. Once the OCP is compiled, different options can be chosen to solve the optimization problem [1]. In this thesis, we use an instance of the JModelica.org's LocalDAECollocator class for transcribing the problem by means of an automatic differentiation tool called CasADi. This package not only helps us with the transcription step, but also permits us to generate C-code of the NLP functions. Then, one can solve sequentially the NLP by calling the interior point solver IPOPT, which is also available in JModelica.org. However, currently IPOPT solves the NLP sequentially and because the transcription is done with orthogonal finite elements, we would like to make use of the structure of the problem, and solve the NLP with parallel computing. Therefore, we aim to use the transcription features of JModelica.org, but instead of solving the NLP within JModelica.org by calling IPOPT, we intend to solve the problem by means of parallel computing with the help of the C++ package developed in [2]. The procedure is represented in Figure 4.1.

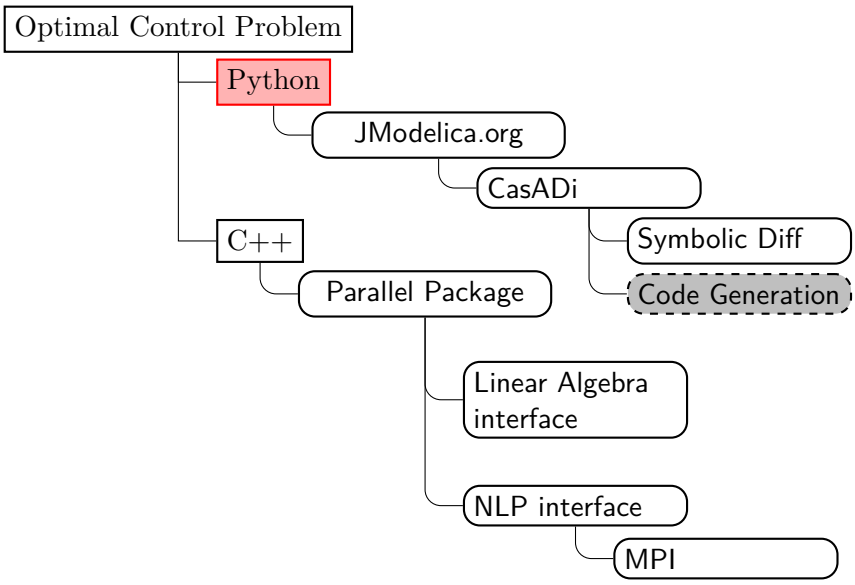


Figure 4.1. Software and tools used for solving the OCP with parallel computing. Both the Python and the C++ programs are linked together through code generation.

For a better understanding, and because generating code could be use for more than simply linking JModelica.org with the package developed in [2], we chose to first cover the topic of code generation in Chapter 5 and later explain in Chapter 6 how the parallel implementation works, and more importantly how it was linked through the generated code. Another reason to have two chapters dedicated to explain the solution approach and its results, is that both topics are extensive and independent of each other.

4.2 Benchmark Problems

In this section we present three optimal control problems used to test the code generation and parallel algorithm. All benchmarks are available in the optimization examples of JModelica.org. The considered problems are based on a VDP oscillator, a continuously stirred tank reactor, and a combined cycle power plant.

4.2.1 Van der Pol oscillator (VDP)

The first problem we analyzed was the well studied Van der Pol (VDP) oscilator which is mentioned in several JModelica.org documents [15][26][1]. Named after the scientist Balthazar van der Pol, a pioneer in the field of telecommunications. The Van der Pol oscillator is a non-conservative with non-linear damping oscillator, which dissipates and generates energy until it reaches a balance state known as the limit cycle. Since its formulation, the Van der Pol equation has become a fundamental equation for modeling nonlinear oscillatory systems

$$
\begin{aligned}
\min \quad & \int_{t_0}^{t_f} \left(x_1^2 + x_2^2 + u^2\right) dt \\
\text{s.t.} \quad & \\
& \tfrac{dx_1}{dt} = \left(1 - x_2^2\right) x_1 - x_2 + u, \\
& \tfrac{dx_2}{dt} = x_1 \\
& x_1(t_0) = 0 \\
& x_2(t_0) = 1 \\
& u \leq 0.75
\end{aligned}
$$

Due to its nature, the Van der Pol oscillator has been widely used in the study of limit cycle oscillations. For instance, in the field of biocomputing, the oscillator has been used for modelling electrical potential across neurons [27]. Additionally, the equation has been also used in the study of non-linear dynamical systems [28]. Thus there stands a good reason to test the software with it.

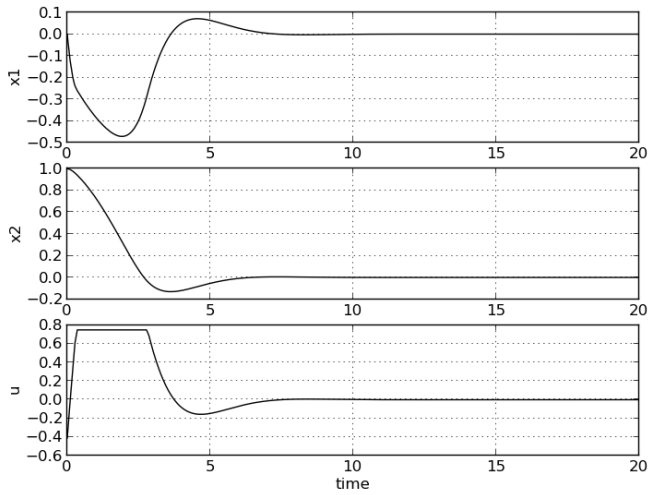


Figure 4.2. Van der Pol benchmark solution.

The optimization problem here consists of finding a control profile $u(t)$ such that the states x_1, x_2 are driven to zero as quickly as possible. In Figure 4.2 we present the solution of the problem obtained by means of JModelica.org. Additionally in Table 4.1 we present the number of variables for the transcribed problem, the number of non-zero values in the Jacobian of the constraints and the Hessian of the Lagrangian function since these values are relevant for the code generation and solution time of the interior point algorithm.

Table 4.1. Van der Pol oscilator NLP transcription. The number of discretization points n_e and n_cp is presented and the corresponding number of optimization variables n_z of the transcribed NLP.

n_e	n_{cp}	n_z	NNZ $\nabla_z c(z)$	NNZ $\nabla^2_{zz} L(z,\nu)$
80	4	1763	6089	1282

4.2.2 Continuous stirred tank reactor (CSTR)

For the second benchmark problem, we chose a typical chemical engineering problem.

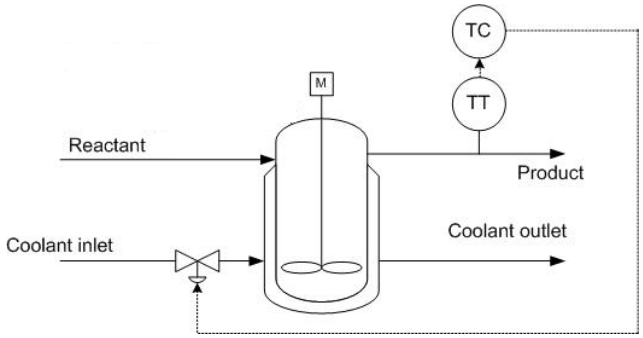


Figure 4.3. CSTR flow diagram.

The problem consists of a chemical reaction system which involves a continuous stirred tank reactor. The reaction mechanism consists only of a first order exothermic reaction. This problem was developed in [29] and considers only two states with a single control variable:

$$\dot{c}(t) = F_0 \frac{c_0 - c(t)}{V} - k(T) \cdot c(t)$$

$$\dot{T}(t) = F_0 \frac{T_0 - T(t)}{V} - \frac{H}{\rho C_p} \cdot k(T) c(t) + \frac{2U}{r \cdot \rho C_p} \left(T_c(t) - T(t)\right)$$

$$k(T) = k_0 \cdot \exp\left(-\frac{E_a}{RT(t)}\right)$$

Here, V is the volume of the reactor, k_0 the pre-exponential constant, F_0 the volumetric flow, E_a the activation energy, R the universal constant of gases, H the reaction enthalpy, C_p the calorific capacity, U the global heat transfer coefficient. For simplicity, it is assumed that the thermal parameters do not vary drastically

with temperature changes, so they are assumed to be constants.

For the optimization problem, we look for a control signal that moves the reactive system from one steady state operating point A to another point B

$$\begin{array}{ccc} c(t_0) = 956.3[\text{mol/m}^3] & & c(t_B) = 338.8[\text{mol/m}^3] \\ T(t_0) = 250[K] & \longrightarrow & T(t_B) = 280[K] \\ T_c(t_0) = 370[k] & & T_c(t_B) = 280[K] \end{array}$$

The results of the optimization problem are presented in Figure 4.4. Due to the non-linearity of the problem, in order to get accurate results a large number of elements is recommended [15].

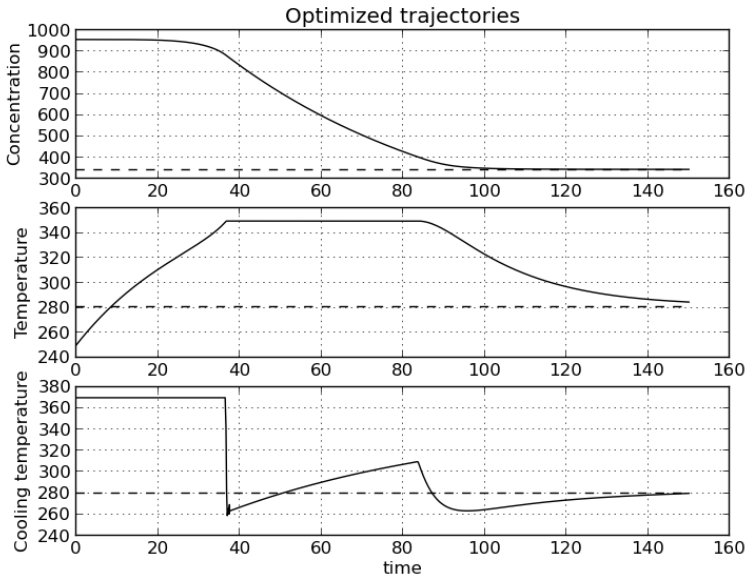


Figure 4.4. CSTR benchmark solution.

Finally the number of discretization points and the non-zero values of the gradient of the constraints and the Hessian of the transcribed Lagrangian function are presented in Table 4.2

Table 4.2. CSTR NLP transcription. The number of discretization points n_e and n_cp is presented and the corresponding number of optimization variables n_z of the transcribed NLP.

n_e	n_{cp}	n_z	NNZ $\nabla_z c(z)$	NNZ $\nabla^2_{zz} L(z, \nu)$
80	4	2405	7694	1603

4.2.3 Combined cycle power plant (CCPP)

For the third benchmark problem, we chose a more challenging problem, the combined cycle power plant (CCPP) model first described in [30] and studied in several JModelica documents. Recently, Word et al [2] studied a parallel Schur-decomposition for solving the CCPP problem in parallel. Hence, we choose this problem as our case study as well. The model has 9 states, 128 algebraics and one control variable. In contrast to the previous section, in this case, the goal is to minimize the time required to start up the power plant.

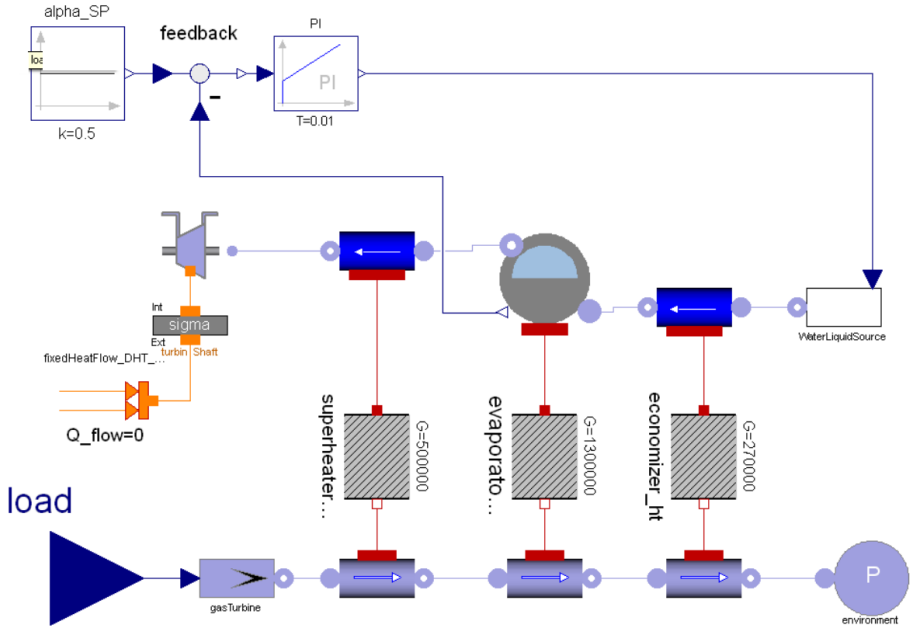


Figure 4.5. Combine cycle power plant flow diagram.

The objective function for the dynamic optimization problem only involves the Lagrange term of the Bolza problem described in Chapter 2 which penalizes the deviation of the process from the steady state. Therefore, the start up of the plant is considered finished once the evaporator reaches a pressure of 8.35 MPa and when the load input u has reached 100%.

$$f(z) = \int_{t_0}^{t_f} \left(10^{-12} \cdot (p(t) - 8.35 \cdot 10^6)^2 + 0.5 \cdot (u(t) - 1)^2\right) dt$$

where $t_0 = 0$s, $t_f = 4000$s and the load input

$$u(t) = 0.15 + 0.85 \cdot \frac{t}{T \cdot \left(1 + \left(\frac{t}{T}\right)^6\right)^{\frac{1}{6}}}$$

where $T = 10000$ s is the simulation duration. In Table 4.3, we present the number of discretization points and the non-zero values of the gradient of the constraints and the Hessian of the transcribed Lagrangian function of the CCPP problem.

Table 4.3. CCPP NLP transcription. The number of discretization points n_e and n_cp is presented and the corresponding number of optimization variables n_z of the transcribed NLP.

n_e	n_{cp}	n_z	NNZ $\nabla_z c(z)$	NNZ $\nabla^2_{zz} L(z, \nu)$
80	4	47014	146664	40446

From the results in Figure 4.6 one can see that in the optimal case the steady state of the plant is reached approximately after 2700 s.

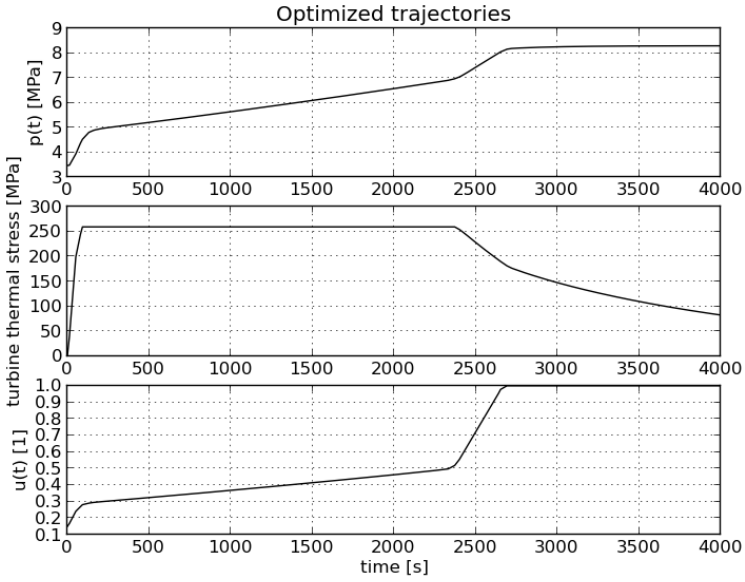


Figure 4.6. CCPP benchmark solution.

Chapter 5

Code Generation

Over the past few years, a new initiative of writing programs that generate program code has become more and more popular in the field of computing [31]. This code generation initiative consists of converting some intermediate representation into source code that a compiler can later read. In this thesis project, the goal of using code generation is to take an input in one language and produce an output in a different one. We aim to use the transcribing features in JModelica.org to convert the dynamic optimization problem into a large-scale NLP. Posterior to the transcription process, we will write C-code that can be compiled outside JModelica.org. This will allow users to interface JModelica.org with their own C or C++ programs, and use external solvers.

In the following chapter, we will describe how the code generation is done. First we will give a brief explanation of how the third party software CasADi generates C-code, and then we will discuss some implementation techniques that allow us to efficiently generate C-files.

5.1 CasADi and the Code Generation Framework

CasADi is a powerful automatic differentiation tool that relies on symbolic abstraction of mathematical functions to solve dynamic problems, in our case optimal control problems. According to the user guide [7], the open source software is based on the following fundamental classes:

- *SX* Scalar symbolic type
- *MX* Matrix symbolic type
- *FX* Functions
- *SX*Matrix Scalar matrix
- *D*Matrix Floating point matrix

CasADi [32] gives support for manipulation and construction symbolic expressions using scalar (SX) and matrix (MX) graph representations. In the scalar graph, the atomic operations of the expression graphs are scalar-valued expressions, while in the matrix graph the atomic operations of the expression are multiple-input, multiple-output. Thus, the scalar type is design for unitary operations ($\mathbb{R} \to \mathbb{R}$ or $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$) and the matrix type for more general vectorial operations ($\mathbb{R}^{n_1 \times m_1} \times \mathbb{R}^{n_N \times m_N} \to \mathbb{R}^{p_1 \times q_1} \times \mathbb{R}^{p_M \times q_M}$). Both expression graphs can be used to formulate function objects (FX). In the case of JModelica.org, these function objects are used to transcribe the OCP, and they are a key feature for the code generation task.

Through CasADi, one can generate code for almost every symbolic function, including both scalar and matrix functions which are generated into C-files.To understand it, let us summarise CasADi's code generation source code in Figure 5.1. In order to generate C-code, CasADi uses two objects: A code generator object which is mainly a printer that writes strings into a stream, and a function object that contains the mathematical information of a mapping, such as the inputs, outputs and sparsity among others. Depending on the type of function, CasADi will handle the code generation differently. In the case of a scalar function, the code generator object prints the information coming from the SX_Function while in the matrix case from the MX_Function.

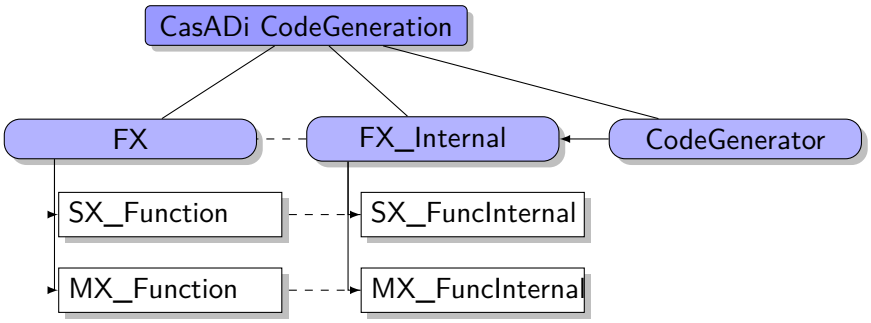


Figure 5.1. CasADi's code generation framework. The diagram presents the relevant CasADi classes used for the code generation.

Despite the code generator object seeming to be just a string translator, it plays perhaps the most important role in the code generation task. It acts as a compiler, making sure that every statement and operation in the C-file are written in the right place. Consider for example the expression

$$x * x + 2$$

where x is a scalar variable. If we intend to generate code for this simple scalar

function, the code generator will write a file with the necessary tools to evaluate the function, like for instance the sparsity pattern of inputs and outputs. Specifically, to evaluate our dummy function we get the following lines in the generated file:

```
#include <math.h>
#define d double
int init(int *n_in, int *n_out){...} //sets number of inputs/outputs

int getSparsity(int i, int *nrow, int *ncol,
        int **rowind, int **col){...} //gets the CSR sparsity

void evaluate(const d* x0,d* r0){
  d a0=x0[0];
  a0=sq(a0);
  d a1=2.0;
  a0=(a0+a1);
  if(r0!=0) r0[0]=a0;
}
```

The code generator behaves as a compiler in the sense that it gets all the information about the function from an intermediate representation within CasADi (FX_function), and then translates it to C code. It makes sure that the following operations will be done following that pattern:

a) Load variables and constants

b) Perform arithmetic operations

c) Store results

In general, the size of the generated file depends mainly on the algorithm size

$$\text{AlgorithmSize} = n_{\text{variables}} + n_{\text{constants}} + n_{\text{operations}} + n_{\text{outputs}}$$

The more complicated the functions are, the larger the files might get. Clearly, for our dummy function, we do not need to worry about size and complexity of the generated file. However, since we plan to generate code for a transcribed NLP with a large number of variables, constants, inputs and outputs is considerably large, we must certainly avoid unnecessary operations in the intermediate representation. The generated code's efficiency depends on the intermediate representation's efficiency. If unnecessary operations are done in the intermediate representation, then they are translated to the generated code making it quite inefficient. For this reason, important changes were done in JModelica's dynamic optimization framework with the aim of generating efficient C-files.

5.2 Restructuring of JModelica.org for Efficient Code Generation With CasADi

For the solution of optimal control problems JModelica.org offers different possibilities [1]. In the case of the local finite element discretization, JModelica.org relies on CasADi's symbolic expressions to transcribe the continuous problem. The method consists of creating FX functions for each term of the OCP (2.21), and by calling the FX functions at the corresponding collocation points, one gets a set of symbolic expressions that composes the NLP (3.2). Thus, one can create a wrapper FX function that takes the discrete point variables $z_{i,k}$ and get all the NLP expressions evaluated at the corresponding points.

One would think that no matter the number of variables and discretization points, the dimension of the generated file will remain approximately constant. Nevertheless, JModelica.org had the problem that an increase in the number of discretization points implied an increase of the size of the generated code. For instance, the generated C-file of a problem with fast dynamics, in which a fine discretization is required, was not even compiling with high performance compilers. For this reason, a restructuring in JModelica.org's code was performed.

With the aim of studying the code generation of a fully discretized optimal control problem, the transcription of the OCP was implemented in two different ways. In the first case, MXFunctions of the type $\mathbb{R}^{n_z} \to \mathbb{R}$ were used for all the constraints and the objective of the model, while in the second case the constraints and objective were implemented following a checkpointing scheme in which the MXFunctions were of the type $\mathbb{R}^{n_z \cdot n_c} \to \mathbb{R}^{n_c}$. While in the first case the MXFunctions must be called $n_e * n_c$ times, in the second one they are called only n_e times. To investigate the dependency of the dimension of the generated file with respect to the number of collocation points, the two implementations were tested with a simple problem with one control and one state.

5.2.1 First approach (Level 0)

The first approach consisted of generating code for a small problem with only 2 elements and 2 collocation points. The layout of the generated file was the following:

1. Fixed size integer arrays containing sparsity information
2. Fixed size double arrays containing the $\dot{\ell}_i(\tau_j)$ coefficients for each element
3. A set of 5 embedded functions:
   - Lagrange function $f_0(t, x, u)$
   - Mayer function $f_1(t, x(t_f))$
   - DAE residual function $f_2(t, x, u, dx)$

- Collocation equation $f_3(\dot{\ell}_{i,(j=0,\ldots,n_c)}(t), x_{i,(j=0,\ldots,n_c)}, dx_{i,j})$
- Initial residual $f_4(t, x)$

4. init function
5. getSparsity function
6. evaluate function

From this implementation, we observed several *for* loops of length one in the generated code. This is against our expectations where we assume that because our problem has 2 elements and 2 collocation points, the generated code will loop once over the elements and once over the collocation points within each element. However, CasADi's NLP function gives 4 loops of length one.

5.2.2 Second approach (Level 1)

C code was generated for the same problem as that in level 0, however the generated code was following a checkpointing scheme. Thus, instead of calling the DAE residual function four times, it was called twice for the two collocation points inside a wrapping function, and the wrapping function was called twice more for each of the two elements. The same scheme was used for the collocation equations and Lagrange integrand. Therefore, the layout of the C file in this case looked like:

1. Fixed size integer arrays containing sparsity information
2. Fixed size double arrays
3. A set of 7 embedded functions:
   - Lagrange function $f_1(t, x, u)$
   - Wrapper Lagrange function $f_0(t_{i=1,\ldots,n_c}, x_{i=1,\ldots,n_c}, u_{i=1,\ldots,n_c})$
   - Mayer function $f_2(t, x(t_f))$
   - Dae residual function $f_4(t, x, u, dx)$
   - Wrapper Dae residual function $f_3(t_{i=1,\ldots,n_c}, x_{i=1,\ldots,n_c}, u_{i=1,\ldots,n_c}, dx_{i=1,\ldots,n_c})$
   - Collocation equation $f_6(\dot{\ell}_{i,(j=0,\ldots,n_c)}(\tau_j), x_{i,(j=0,\ldots,n_c)}, dx_{i,j})$
   - Wrapper collocation equation $f_5(\dot{\ell}_{i,(j=0,\ldots,n_e*(n_c+1))}, x_{i,(j=0,\ldots,n_c)}, dx_{i,j=1,\ldots,n_c})$
   - Initial residual $f_7(t, x)$
4. init function
5. getSparsity function
6. evaluate function

By using the checkpointing scheme, we were able to generate nested loops and hence avoid multiple loops of length one in the generated file. Despite the size of the file decreasing, some other problems were found. Since the wrapper element functions required MX variables of dimension $n_z \cdot n_c$, we used CasADi's concatenation tool to set up the element wrapper function inputs. As mentioned earlier, almost every operation done in the intermediate representation is shown in the generated code. Therefore, some additional loops appeared in the C-file due to the concatenations done with CasADi. This problem consequently makes the checkpointing approach less attractive. However, if instead of concatenating the MX variables, the vector containing all optimization variables is reordered in such a way that one can extract packets of dimension n_c, the concatenation problem could be avoided. Consider the following orders of variables:

$$
V\left\{\begin{array}{l}
e_1\left\{\begin{array}{ll}
C_1\left\{\begin{array}{l} x_{1,1}^{(1)} \\ \dots \\ x_{1,1}^{(n_x)} \\ w_{1,1}^{(1)} \\ \dots \\ w_{1,1}^{(n_w)} \\ u_{1,1}^{(1)} \\ \dots \\ u_{1,1}^{(n_u)} \\ \dot{x}_{1,1}^{(1)} \\ \dots \\ \dot{x}_{1,1}^{(n_x)} \end{array}\right. \\
C_2\left\{\begin{array}{l} z_{1,2}^{(1)} \\ \dots \\ z_{1,2}^{(n_z)} \end{array}\right. \\
C_3\left\{\begin{array}{l} z_{1,n_c}^{(1)} \\ \dots \\ z_{1,n_c}^{(n_z)} \end{array}\right.
\end{array}\right. \\
e_2\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right.
\end{array}\right.
\qquad
\bar{V}\left\{\begin{array}{l}
x^{(1)}\left\{\begin{array}{ll}
e_1\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right. \\
e_2\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right. \\
e_3\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right.
\end{array}\right. \\
\dots \\
x^{(n_x)}\left\{\begin{array}{l} e_1 \\ e_2 \\ e_3 \end{array}\right. \\
w^{(1)}\left\{\begin{array}{l} e_1 \\ e_2 \\ e_3 \end{array}\right. \\
\dots \\
w^{(n_w)}\left\{\begin{array}{l} e_1 \\ e_2 \\ e_3 \end{array}\right. \\
u^{(1)}\left\{\begin{array}{l} e_1 \\ e_2 \\ e_3 \end{array}\right. \\
\dots \\
u^{(n_u)}\left\{\begin{array}{l} e_1 \\ e_2 \\ e_3 \end{array}\right.
\end{array}\right.
\qquad
\tilde{V}\left\{\begin{array}{l}
e_1\left\{\begin{array}{ll}
x^{(1)}\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right. \\
\dots \\
x^{(n_x)}\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right. \\
w^{(1)}\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right. \\
\dots \\
w^{(n_x)}\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right. \\
u^{(1)}\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right. \\
\dots \\
u^{(n_x)}\left\{\begin{array}{l} C_1 \\ C_2 \\ C_3 \end{array}\right.
\end{array}\right. \\
e_2\left\{\begin{array}{l} \dots x^{(n_x)} \\ \dots w^{(n_w)} \\ \dots u^{(n_u)} \end{array}\right.
\end{array}\right.
\tag{5.1}
$$

JModelica.org utilized the order V in equation (5.1). Nevertheless, we look for an order that allows us to extract packets of variables for the element wrapper functions, thus V is not the most appropriate order for the checkpointing scheme. For example, while setting up the input of f_3, one needs to pass a variable x_i of dimension n_c, hence with the V order one must extract $x_{i,j}$ separately and then concatenate them in a single MX variable. This is because of non-contiguous order of collocation points of a certain sort of variable. On the other hand, both the $\bar{V}$ and $\tilde{V}$ orders have the advantage that one can extract x_i directly with no concatenation being required. Hence, the ideal ordering of variables is one that allows us to extract directly the x_i, u_i, w_i, dx_i variables needed for the wrapper functions. If the problem involves several number of variables, the ideal order would be $\tilde{V}$ instead.

Until this point, we have discussed a checkpointing scheme with only one level

of packing. Nonetheless, the packing scheme could be further exploited if nested levels are implemented. Instead of having only wrapper functions for every element, one could have wrapper functions packing more than one element, and in that case the ideal mapping would look like:

$$
\breve{V}\left\{\begin{array}{l}
\ldots \\
e_1\left\{\begin{array}{l} C_0\left\{\begin{array}{l} x^{(1)} \\ \ldots \\ x^{(n_x)} \end{array}\right. \\ \ldots \\ C_{n_{cp}}\left\{\begin{array}{l} x^{(1)} \\ \ldots \\ x^{(n_x)} \end{array}\right. \end{array}\right. \\
\ldots \\
e_n\left\{\begin{array}{l} C_0\left\{\begin{array}{l} x^{(1)} \\ \ldots \\ x^{(n_x)} \end{array}\right. \\ \ldots \\ C_{n_{cp}}\left\{\begin{array}{l} x^{(1)} \\ \ldots \\ x^{(n_x)} \end{array}\right. \end{array}\right. \\
\ldots \\
e_1\left\{\begin{array}{l} C_1\left\{\begin{array}{l} \dot{x}^{(1)} \\ \ldots \\ \dot{x}^{(n_x)} \end{array}\right. \\ \ldots \\ C_{n_{cp}}\left\{\begin{array}{l} \dot{x}^{(1)} \\ \ldots \\ \dot{x}^{(n_x)} \end{array}\right. \end{array}\right. \\
\ldots \\
e_n\left\{\begin{array}{l} C_1\left\{\begin{array}{l} \dot{x}^{(1)} \\ \ldots \\ \dot{x}^{(n_x)} \end{array}\right. \\ \ldots \\ C_{n_{cp}}\left\{\begin{array}{l} \dot{x}^{(1)} \\ \ldots \\ \dot{x}^{(n_x)} \end{array}\right. \end{array}\right. \\
e_1\left\{\begin{array}{l} C_1\left\{\begin{array}{l} u^{(1)} \\ \ldots \\ u^{(n_u)} \end{array}\right. \\ \ldots \\ C_{n_{cp}}\left\{\begin{array}{l} u^{(1)} \\ \ldots \\ u^{(n_u)} \end{array}\right. \end{array}\right. \\
\ldots \\
e_n\left\{\begin{array}{l} C_1\left\{\begin{array}{l} u^{(1)} \\ \ldots \\ u^{(n_u)} \end{array}\right. \\ \ldots \\ C_{n_{cp}}\left\{\begin{array}{l} u^{(1)} \\ \ldots \\ u^{(n_u)} \end{array}\right. \end{array}\right. \\
\ldots
\end{array}\right. \tag{5.2}
$$

Although we did not implement the nested checkpointing scheme, the $\breve{V}$ order was implemented in JModelica, so that the nesting could easily be implemented in future work if necessary. Apart from modifying the order of the variables, the signature of CasADis functions in JModelica was also modified. Instead of declaring CasADi's functions as $f(z(t))$, the functions are now declared as $f(x(t), \dot{x}(t), u(t), w(t), p)$. The reason for this modification was to avoid unnecessary concatenations.

5.3 Results

The new order of variables and the checkpointing scheme were tested with the benchmark problems presented in Chapter 4, and with the simple problem containing only one state and one input. For these tests, all problems were run with 80 elements and 4 collocation points.

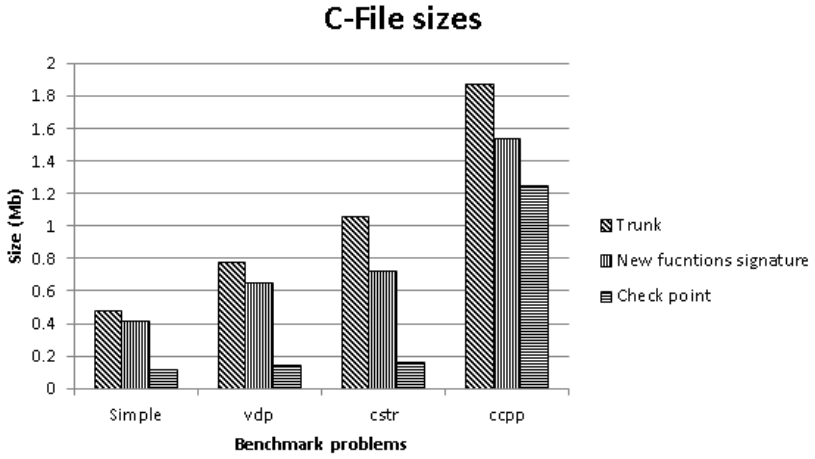


Figure 5.2. Generated code size

In Figure 5.2, we can see the improvement in the code generation size of the transcribed NLPs. There Trunk corresponds to previos JModelica.org implementation, new functions signature are the results after reordering the variables, and check point the results after implementing the checkpointing scheme. The reader might wonder why we bother about the size of the generated file, even though users in general would never see the file. There are two primary reasons. First, as we mentioned, the code generation consists of the translation of an intermediate representation into C code, which results in the explanation that a reduction in the size of the generated file is caused by a reduction of machine memory usage. Secondly, Figure 5.2 only shows the sizes of the transcribed NLP's code generation, but we are interested in more complex functions. Recall the discussion on interior point methods in section 2.3. Expensive operations such as computing the Jacobian of the constraints or the Hessian of the Lagrange function are required in order to solve the primal dual system with the Newton method (2.13). Thus, generating code for the evaluation of such functions would lessen the impact of those operations when solving the NLPs. For this reason, we are interested in generating code for such functions with CasADi, and by aiming to reduce the code generation size of the transcribed NLP, we hope to obtain a reasonable size for the more complex functions that we are interested in.

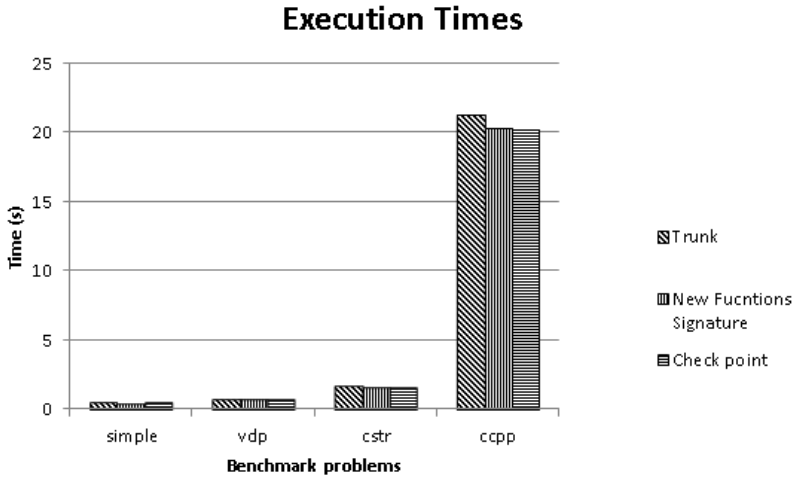


Figure 5.3. Solving execution time

To ensure that the new features added to JModelica.org's implementation do not worsen the performance, we check the solution time for each of our benchmark problems. The results are presented in Figure 5.3. There, one can see that the time required for solving the problems remained either constant or it barely decreases.

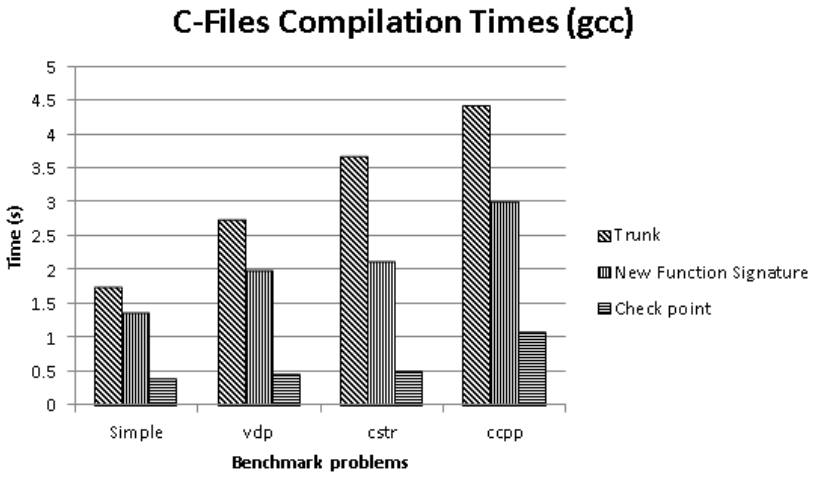


Figure 5.4. Compilation time of generated files

Another relevant result for this project is presented in Figure 5.4. In the Figure, the checkpointing scheme reduces the compilation time of the generated code in all of our benchmark problems remarkably. However, the results presented in Figure 5.4 are

for problems with a fixed number of discretization points. Therefore, to analyse the true benefits of checkpointing, we loop through the number of collocation points and elements, and summarize the results in the following Figures. Let us first analyse the case of the simple problem that only contains one state and one input.

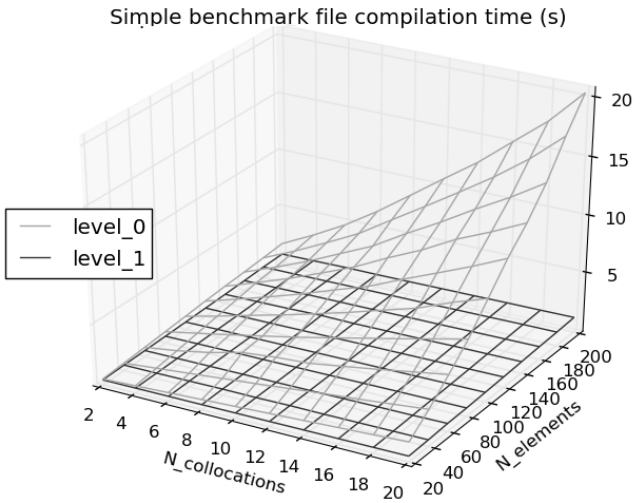


Figure 5.5. Compilation times for the Simple benchmark.

Figure 5.5 shows that the compilation time of the generated code for the simple NLP increases dramatically as the number of discretization points increases. Nonetheless, when we generate code following a checkpointing scheme, the compilation time remains almost constant. Observe also that the compilation time varies differently if the number of collocation points is increased when the number of elements increases. In order to understand this better, let us define the speed up factor as

$$S_p = \frac{t_{l_0}}{t_{l_1}} \tag{5.3}$$

Where t_{l_0} stands for the compilation time without checkpointing and t_{l_1} is the compilation time with checkpointing. Based on (5.3), we can study the benefits of the packing technique further, and determine how much faster the compilation time of the generated file is.

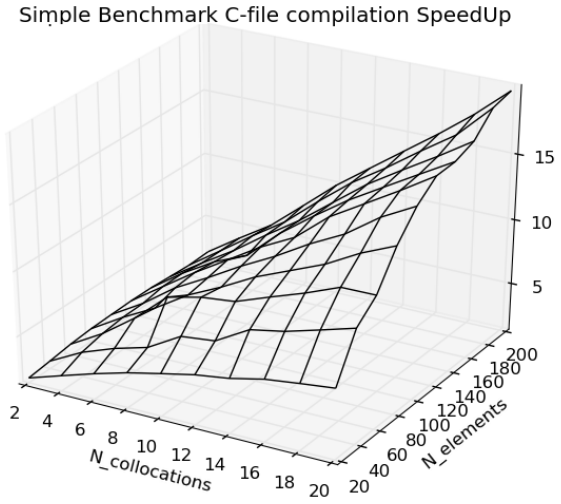


Figure 5.6. Compilation speed-up for the Simple benchmark.

Note that for the simple problem the checkpointing scheme greatly speeds up the compilation time of the generated code. Note also that the speed up with respect to the number of elements resembles a logarithmic behaviour, while with respect to the collocation points a linear behaviour is represented. This can be explained by the fact that we only used 2 levels of packing. If nesting check point is used instead, the compilation time most probably decreases more but the solving time might get compromised.

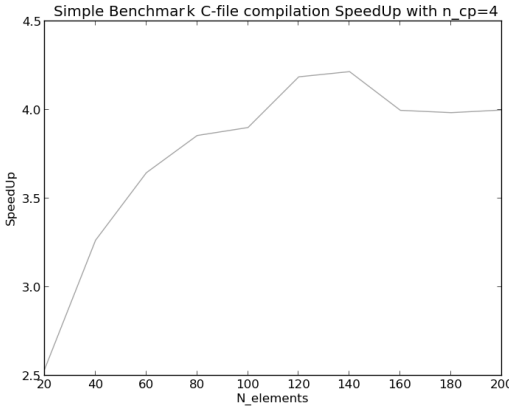


Figure 5.7. Fixed number of Collocations.

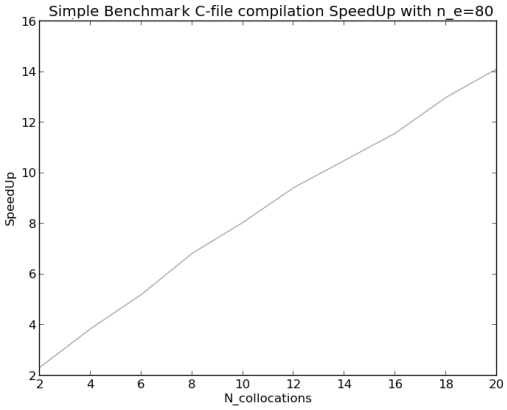


Figure 5.8. Fixed number of Elements.

Regarding the VDP and the CSTR benchmark problems we get similar results.

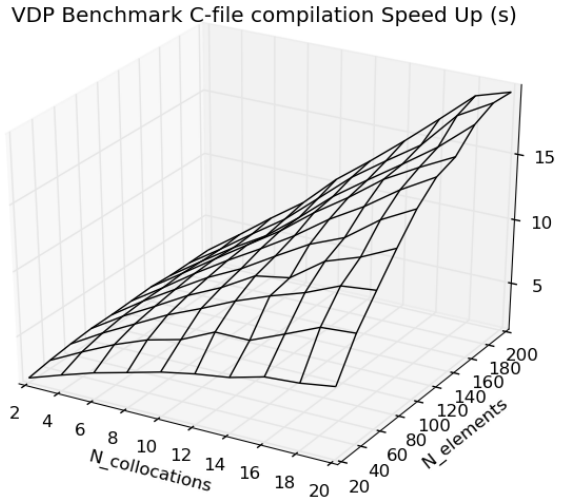


Figure 5.9. Compilation speed-up for the VDP benchmark.

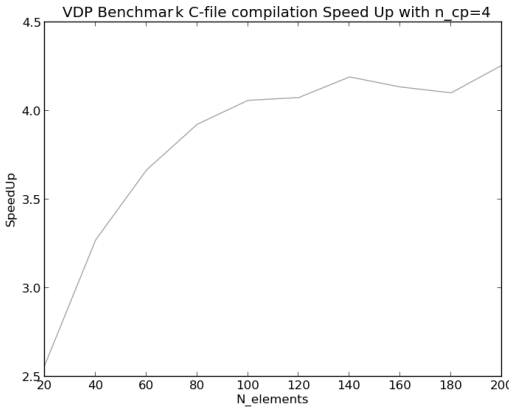


Figure 5.10. Fixed number of Collocations.

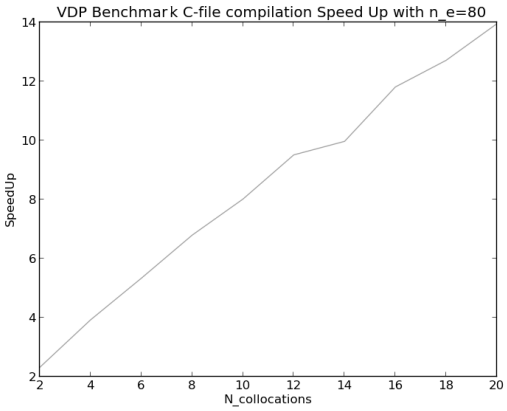


Figure 5.11. Fixed number of Elements.

Finally, Figures 5.15, 5.16 and 5.17 show the results of the most complicated problem among the benchmark problems we are dealing with. In the case of the CCPP benchmark, due to the number of variables, the improvements are not stunning as in the other three cases. We still get some speed up in the compilation times but not in the same order as in the case of the simple benchmark, the VDP or the

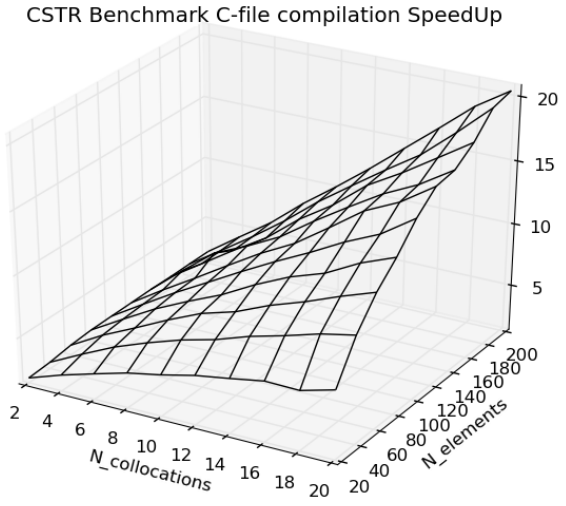


Figure 5.12. Compilation speed-up for the CSTR benchmark.

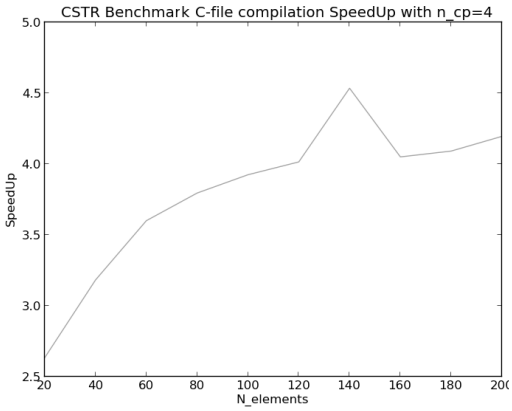


Figure 5.13. Fixed number of Collocations.

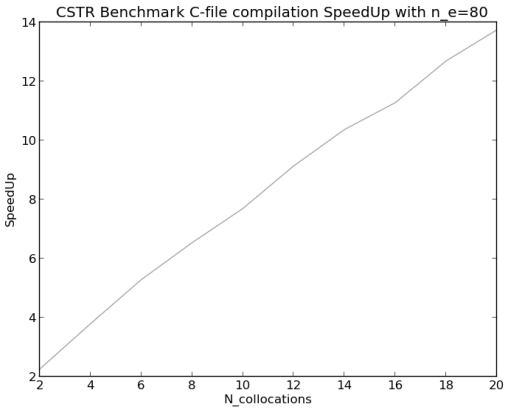


Figure 5.14. Fixed number of Elements.

CSTR problems. Further investigations concerning this problem should be carried out, but due to time constraints, we were restricted from doing so and continued on to the next step in the project. This is not considered a complication for the following parts of the thesis since the generated files still compiled in a reasonable time. Moreover, before implementing the changes described in this chapter, the code generation of functions like the Hessian of the Lagrange function or the gradient of the constraints could not even be compiled.

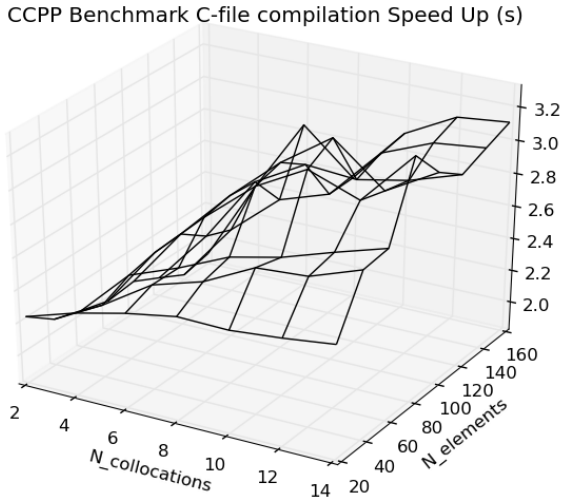


Figure 5.15. Compilation speed-up for the CCPP benchmark.

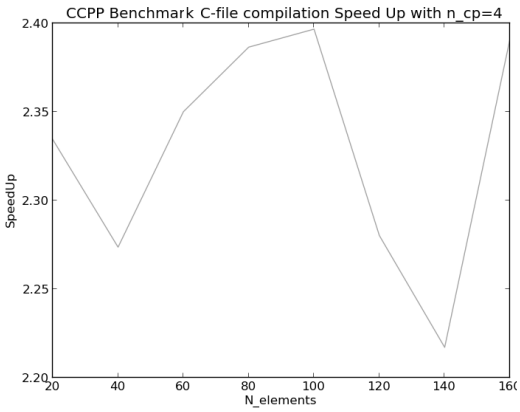


Figure 5.16. Fixed number of Collocations.

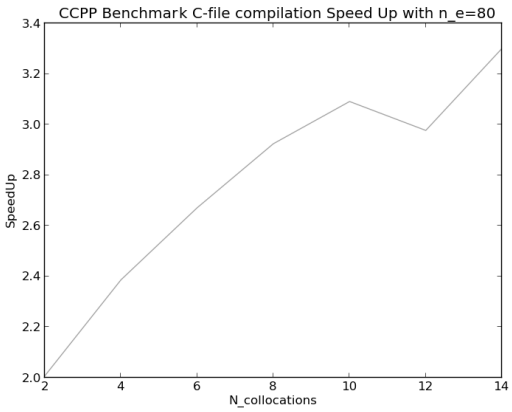


Figure 5.17. Fixed number of Elements.

With these improvements in the code generation side of our problem, we proceed to solve transcribed NLP problems in C++ by interfacing with the package developed in [2].

Chapter 6

C++ Framework

Direct Transcription has proven to be an efficient method for solving dynamic optimization problems[4]. Although the simultaneous approach presents significant advantages in comparison to the sequential approach, one noticeable disadvantage must be highlighted. Since the technique fully discretizes the OCP's state and algebraic variables, and converts the infinite dimensional problem into an NLP, specialized methods are required to solve the large scale non-linear programs. Among the different solution strategies, one can find variations of Sequential Quadratic Programing (SQP) and barrier or interior point techniques (IP). In this project, we choose an interior point algorithm to solve the non-linear program. The reason to do so is to exploit the structure of the KKT system in order to use parallel computations on multiple processors. In the following chapter we first discuss a decomposition strategy that allows efficient solution of OCPs in both distributed and shared memory architectures [33] is described. Then an interface to use the parallel subroutines provided by the package developed in [2]. Finally, we present the results obtained by linking JModelica.org's generated code with the package from [2].

6.1 Parallel Algorithm Approach

The solution of large-scale nonlinear programming problems is possible with a great variety of algorithms. In case of the interior point algorithm, the solution of the linear KKT system at each iteration, required to find the step in the primal and dual variables, dominates the cost. The discretization technique described in Chapter 3 results in a band-block structured KKT system that can be decomposed in blocks by selecting breaking points in time, see e.g [33]. D.Word et al C++ package [2] follows an interior point NLP strategy where a Schur-complement internal decomposition technique is used to efficiently solve in parallel the OCP.

Having introduced the sub-structuring in Section 2.4, let us now focus on the problem of interest. After discretizing the dynamic optimization problem into an

NLP, the KKT system solved with the interior point algorithm can be written in N sub-domains. For example, a problem with 64 finite elements can be divided into $N = 2, 4, 8, 16, 32, 64$ sub-domains. The number of blocks or sub-domains should be determined by the number of available processors.

$$\begin{bmatrix} K_1 & & & & A_1^T \\ & K_2 & & & A_2^T \\ & & \ddots & & \vdots \\ & & & K_N & A_N^T \\ A_1 & A_2 & \cdots & A_N^T & Q \end{bmatrix} \begin{bmatrix} \Delta\upsilon_1 \\ \Delta\upsilon_2 \\ \vdots \\ \Delta\upsilon_N \\ \Delta\upsilon_s \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \\ r_s \end{bmatrix} \tag{6.1}$$

Here A_i are block matrices that couple the individual blocks K_i, $\Delta\upsilon_i$ include the primal and dual variables for block i, and $\Delta\upsilon_s$ is the dual variables for the coupling constrains. To understand these coupling matrices let us assume that each block is a finite element of the discretized dynamic problem. Consider the discretized optimal control problem

$$\begin{array}{ll} \min & f(z) \\ \text{s.t.} & c(z) = 0 \\ & Z_L \leq z \leq Z_U \end{array} \tag{6.2}$$

where

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_{n_e} \end{bmatrix}, \text{and} \quad c(z) = \begin{bmatrix} \bar{G}z_1 - x_0 \\ R(z_1) \\ \underline{G}z_1 + \bar{G}z_2 \\ \vdots \\ \underline{G}z_{n_e-1} + \bar{G}z_{n_e} \\ R(z_{n_e}) \end{bmatrix} \tag{6.3}$$

Here, $\bar{G} \in \mathbb{R}^{n_x \times n_z}$ and $\underline{G} \in \mathbb{R}^{n_x \times n_z}$ are linking matrices that arise from the continuity equations that extract $x_{i,0}$ and x_{i,n_c} from each block. By means of these two matrices, we define the coupling constraints in $c(z)$, since $\underline{G}z_{i-1} + \bar{G}z_i$ are constraints that link the individual blocks, and the $R(z_i)$ constraints that correspond to the collocation and DAE residuals of each block. Note that both $\bar{G}$ and $\underline{G}$ only link the states, but not the algebraics or controls of the OCP. Thus, the number of coupling constraints depends only on the number of state variables, a feature that will be exploited by the parallel algorithm.

Now with the linking matrices $\bar{G}$ and $\underline{G}$ we can build the coupling matrices A

$$A_1 = \begin{bmatrix} 0 & 0 & 0 \\ \underline{G} & 0 & 0 \\ 0 & 0 & 0 \\ & \vdots & \end{bmatrix}, \quad A_i = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -I & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \underline{G} & 0 & 0 \\ 0 & 0 & 0 \\ & \vdots & \end{bmatrix}, \quad A_{n_e} = \begin{bmatrix} 0 & 0 & 0 \\ & \vdots & \\ 0 & 0 & 0 \\ 0 & -I & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.4}$$

For the individual K_i submatrices we have:

$$K_i = \begin{bmatrix} H_i + \delta_H I & \bar{G}^T & \nabla_{z_i} R(z_i) \\ \bar{G}^T & -\delta_c I & \\ \nabla_{z_i} R(z_i)^T & & -\delta_c I \end{bmatrix}, \quad \Delta\upsilon_i = \begin{bmatrix} \Delta z_i \\ \Delta\gamma_{\bar{G},i} \\ \Delta\gamma_{R,i} \end{bmatrix}, \quad r_i = \begin{bmatrix} -\tilde{r}_{z_i} \\ -\bar{G}z_i + q_{i-1} \\ -R_i \end{bmatrix} \tag{6.5}$$

Here H_i is the hessian described in (2.15) for the augmented form of the KKT system, δ_H and δ_c correct the intertia condition, $\tilde{r}_{z_i}$ is the residual defined in (2.16), and lastly $q_i = x_{i,n_c}$ the decoupling variables. Observe that the $\Delta\upsilon_i$ vectors only include the primal and dual variables of the individual blocks. The remaining coupling variables in the KKT system (6.1) are

$$Q = \begin{bmatrix} \delta_H I & I & & & & & \\ I & -\delta_c I & & & & & \\ & & \delta_H I & I & & & \\ & & I & -\delta_c I & & & \\ & & & & \ddots & & \\ & & & & & \delta_H I & I \\ & & & & & I & -\delta_c I \end{bmatrix} \tag{6.6}$$

$$\Delta\upsilon_s = \begin{bmatrix} \Delta q_1 \\ \Delta\lambda_{\underline{G},1} \\ \vdots \\ \Delta q_{n_e-1} \\ \Delta\lambda_{\underline{G},n_e-1} \end{bmatrix}, \text{and} \quad r_s = \begin{bmatrix} -\nabla_q \tilde{r}_1 \\ -\underline{G}z_1 - q1 \\ \vdots \\ -\nabla_q \tilde{r}_{n_e-1} \\ -\underline{G}z_1 - q_{n_e-1} \end{bmatrix} \tag{6.7}$$

To decouple the step of the υ_s variables we can then use a Schur-complement decomposition, like the one proposed at the beginning of this section. The idea is to

eliminate the A_i matrices in order to exploit the sparse structure of the discretized problem by means of the Schur-complement decomposition,

$$\left[Q - \sum_i A_i K_i^{-1} A_i^T\right] \Delta v_s = r_s - \sum_i A_i K_i^{-1} r_i \tag{6.8}$$

The discussed block structure (6.1) can be exploited through the Schur-complement decomposition (6.8) in order to efficiently find the primal and dual variable step, by means of parallel computing, for each iteration of the interior point algorithm. Notwithstanding, the approach is considered beneficial only for problems with considerably more number of algebraic variables than states. The reason is that the computational time of the parallel algorithm is dominated by the construction of the Schur-complement, where the size of it is determined by the number of coupling variables. Hence, if the size of the Schur-complement grows, the parallel speed up is eroded.

ALGORITHM

1 **foreach** i *in* $1, \cdots, N$ **do**
2 | factor K_i (Using MA27 subroutines)
3 **end**
4 Initialize S by letting $S = Q$
5 Let $r_{sc} = r_s$
6 **foreach** i *in* $1, \cdots, N$ **do**
7 | **foreach** *nonzero column* j *in* A_i^T **do**
8 | | solve the system $K_i s_i^{<j>} = [A_i^T]^{<j>}$ for $s_i^{<j>}$
9 | | let $S^{<j>} = S^{<j>} - A_i s_i^{<j>}$
10 | **end**
11 | solve the system $K_i p_i = r_i$ for p_i
12 | let $r_{sc} = r_{sc} - A_i p_i$
13 **end**
14 solve $S\Delta v_s = r_{sc}$ for Δv_s
15 **foreach** i *in* $1, \cdots, N$ **do**
16 | solve $K_i \Delta v_i = r_i - A_i^T \Delta v_s for \Delta v_i$
17 **end**

The parallel algorithm outlined above has several levels of parallelism. Steps 1,6 and 15 can be parallelized if there are as many processors available as the number of N blocks. On the other hand, if more processors are available than blocks, individual column backsolves can be parallelized as well. Observe that the performance of this algorithm is a function of the number of states and the number of processors. An increase in the number of processors may potentially increase the parallelism, but the size of the Schur-complement also increases which is disadvantageous to the

parallelism. However, the algorithm has significant speed up potential for problems where the number of algebraic variables outnumbers the number of states [33].

6.2 Implementation

In this thesis project, we interface JModelica.org with the C++ framework developed in [2]. First, we make use of CasADi's code generation tool to write the transcribed NLP functions into a set of C-files as described in Chapter 5.

- g_nlp_info.hpp
- g_dynop_info.hpp
- g_init_cond.hpp
- g_objective.hpp
- g_grad_objective.hpp
- g_constraints
- g_jac_constraints.hpp
- g_hessian_lagrange.hpp

Eight files are generated. The first two files are self written files that include functions for setting the number of variables, the initial guesses, the state initial values, the number of discretization points, and the number of individual blocks among other parameters. On the other hand, the CasADi generated files have the structure described in Section 5.2.2. With these files one can evaluate all NLP functions in (3.2) of a single K_i block.

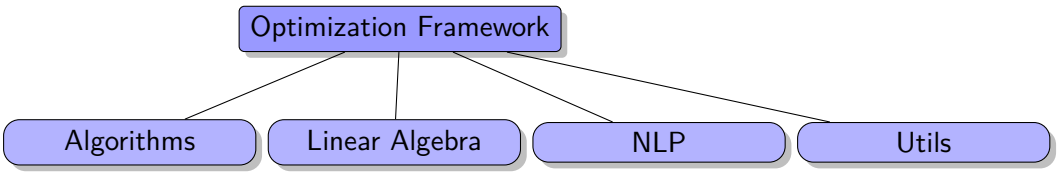


Figure 6.1. Optimization framework structure diagram.

The C++ optimization framework from [2] follows an object oriented programming paradigm. In Figure 6.1 a diagram with the corresponding packages of the optimization framework is presented. In the algorithm package, one can call the interior point solver which requires mainly two objects to be created, a KKTLinearSolver

object which in our case uses a Schur-complement decomposition, and an NLP object from the NLP package. With these three objects, one can run the code in serial. However, in order to run interior point solver with multiple processors, the KKTLinearSolver and the NLP objects must call the MPI wrappers from the Utils package.

The design of the optimization framework [2] allowed us to link the generated files in a straightforward manner. We first implemented a Generated_NLP class that inherits from the NLP abstract class provided in the framework. The evaluation functions of the generated files are called within the Generated_NLP class. Thus, the Generated_NLP object represents a single K_i block. Each Generated_NLP object is created in a different processor and blocks ids are assigned according to the MPI rank of the processors. Second, with help of the setting functions from the self written files, we build up the linking matrices $\bar{G}$ and $\underline{G}$ that are required to create the KKTLinearSolver object. This will use the MPI subroutines (Utils package) and the specialized sparse linear solvers (Algorithm package) to solve the KKT system with multiple processors. Finally, the interior point solver (Algorithm package) solves the transcribed NLP (3.2).

6.3 Results

To test the performance of the algorithm, we focused on timing only the combined cycle power plant benchmark problem. The main reason for this is that among the benchmark problems, the CCPP has significantly more algebraic variables than state variables. However the interfaced worked for all benchmark problems. Therefore, we test the speedup of the algorithm with 3 different discretization sizes. Figure 6.2 shows the speedup achieved with 2 and 4 processors. The timing results were obtained on an Intel Core i7-2640M Processor.

Detailed information about the average time per iteration of the main steps of the algorithm are shown in Table 6.1. Both the serial and parallel timings are shown for the CCPP problem while varying the number of elements. Observe that even for the serial case, an increase in the number of blocks reduces the time required for steps 1 and 15. This is due to the fact that increasing the number of blocks makes each individual block smaller, while factorizing K_i and performing backsolves becomes significantly faster with these smaller blocks as well. On the other hand, increasing the number of blocks makes step 6 slower since forming the Schur-complement with more blocks requires more operations and consequently more time when it is computed in serial.

The proposed parallel approach has potential for significant speedup for problems in which the number of state variables outnumbers the algebraic variables. As the parallel algorithm is based on a Schur-complement decomposition, a few number of

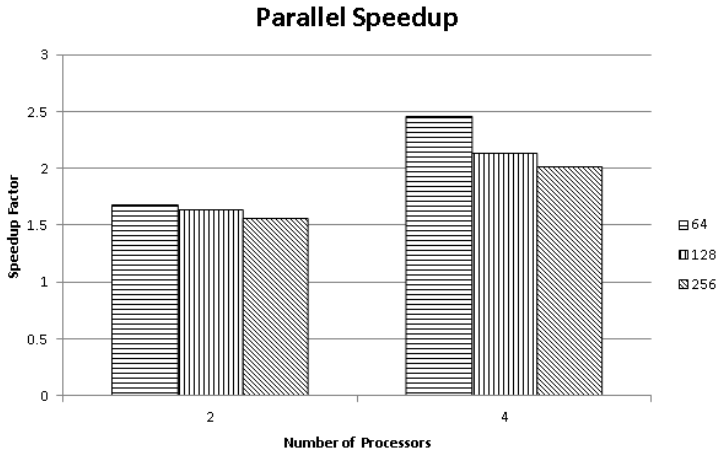


Figure 6.2. Speedups comparing the parallel Schur-complement linear solver with the Serial Schur-complement linear solver for the CCPP problem with 64,128 and 256 finite elements.

Table 6.1. Average time per iteration required in the steps of the Schur-complement decomposition algorithm. Both serial and parallel times are shown.

Schur-complement Algorithm Times									
n_e	N	Step 1		Step 6		Step 14		Step 15	
		Serial	Parallel	Serial	Parallel	Serial	Parallel	Serial	Parallel
64	2	0.0374	0.0214	0.0257	0.0171	5.9E-05	5.4E-05	0.0027	0.0018
	4	0.0351	0.0115	0.0378	0.0185	0.00012	0.00013	0.0026	0.0015
128	2	0.0756	0.0445	0.0558	0.0384	7.1E-05	5.5E-05	0.0057	0.0038
	4	0.0734	0.0257	0.0789	0.0513	0.00014	0.00012	0.0055	0.0037
256	2	0.1721	0.1053	0.1193	0.0875	6.9E-05	5.2E-05	0.0113	0.0085
	4	0.2216	0.0950	0.1724	0.11349	0.00015	0.00011	0.0114	0.00778

states reduced the coupling between the individual blocks and consequently inter-processor communication required for forming the Schur-complement decreases as well. For this reason, we tested the algorithm with the third benchmark problem of chapter 4. The results verify the potential benefit of the parallel algorithm observed by D.Word et al [2].

Chapter 7

Concluding Remarks

7.1 Conclusion

We have successfully implemented an interface that links JModelica.org optimization platform with an external interior point solver that solves large-scale NLP problems with parallel computations. This interface allows us to transform high-level descriptive Modelica models of dynamic optimization problems into algebraic nonlinear programming problems in JModelica.org. The optimization framework converts the infinite-dimensional dynamic problem into a finite-dimensional one through a direct collocation approach based on Radau or Gauss schemes. Then, with the help of the third party software CasADi [7], a set of C-files that contains the algebraic non-linear programming problem functions is generated. This set of files is taken by the interface and compiled together with the C++ package that uses a Schur-complement decomposition algorithm for solving the problem with an interior point approach in parallel [2].

For efficient generation of C-files in JModelica.org, a new approach for constructing a transcribed NLP (3.2) with CasADi was followed. Thus, some of the transcription features of JModelica.org optimization platform were implemented with a different approach. By means of this new implementation, through a checkpointing scheme, we were able to overcome the problem of non-compilable generated files. Moreover, this implementation not only allowed us to compile the generated code of transcribed NLPs with a large number of discretization points, but also allowed to speed up the compilation time of the generated C-files.

The non-linear programming problem information was passed to a C++ interface that compiled together the generated C-files with the parallel solver [2]. This allowed us to use all numerical features from JModelica.org for converting the dynamic problem into an NLP and then solve it efficiently in parallel. In addition, the integration of these two packages allowed us to use symbolic calculations of the third party software CasADi, which is also beneficial for the interior point algorithm due

to the involvement of efficient automatic differentiation.

The combined cycle power plant problem presented in Chapter 4 was used to test the interface. The shared memory architecture results reported in [2] were similar to those obtained in the present work. A speedup factor over 2.5 was obtained on Intel Core i7-2640M Processor when using 4 cores. This was possible because in that benchmark, the number of algebraic variables outnumbers the number of states. As reported in [2] for problems with few states and many algebraics, the Schur-complement decomposition algorithm has the potential for significant speedup. This must be considered when solving problems through the developed interface.

Bibliography

[1] *JModelica.org User Guide*. Modelon AB, September 2013.

[2] Daniel P Word, Jia Kang, Carl D Laird, and Johan Akesson. Efficient parallel solution of large-scale nonlinear dynamic optimization problems. *Journal of Computational Optimization and Applications*, 2014.

[3] Arturo Cervantes and L. T. Biegler. Optimization strategies for dynamic systems. *Encyclopedia of Optimization*, pages 2847–2858, 2009.

[4] Lorenz T. Biegler, Arturo M. Cervantes, and Andreas Wachter. Advances in simultaneous strategies for dynamic process optimization. *Chemical Engineering Science*, 57:575–593, 2002.

[5] Christian Kirches. *Fast Numerical Methods for Mixed-Integer Nonlinear Model-Predictive Control*. Advances in Numerical Mathematics. Springer-Verlag, 2011.

[6] Andreas Waechter and Lorenz T. Biegler. On the implementation of an interior point filter line-search algorithm for large scale nonlinear programming. *Mathematical Programming - Springer*, 106:25–57, 2006.

[7] Joel Andersson, Attila Kozma, Joris Gillis, and Moritz Diehl. *CasADi User's Guide*, 2014.

[8] N. Bellomo, B. Lods, R. Revelli, and L. Ridolfi. *Generalized Collocation Methods*. Birkhauser Boston, 2008.

[9] Uri M. Asher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic equations*. Society for Industrial and Applied Mathematics SIAM, 1998.

[10] Shivakumar Kameswaran and Lorenz T. Biegler. Convergence rates for direct transcription of optimal control problems using collocation at radau points. *Computational Optimization and Applications*, 41:81–126, 2008.

[11] Lorenz T. Biegler. *Nonlinear Programming. Concepts, Algorithms, Applications to Chemical Processes*. Society for Industrial and Applied Mathematics Philadelphia SIAM, second edition, 2010.

[12] Victor M. Zavala. *Computational Strategies for the Optimal Operation of Large-Scale Chemical Processes*. PhD thesis, Carnegie Mellon University, 2008.

[13] Geoffrey T. Huntington. *Advancement and Analysis of a Gauss Pseudospectral Transcription for Optimal Control Problems*. PhD thesis, Massachusetts Institute Of Technology, 2007.

[14] S. Koack and H.U. Akay. Parallel schur complement method for large-scale systems on distributed memory computer. *Applied Mathematical Modelling*, 25:873–886, 2001.

[15] Fredrik Magnusson. Collocation methods in jmodelica.org. Master's thesis, Lund University, 2012.

[16] John T. Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*. Society for Industrial and Applied Mathematics Philadelphia SIAM, 2001.

[17] Uri M. Asher, Robert M.M Mattheij, and Robert D.Russel. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic equations*. Society for Industrial and Applied Mathematics SIAM, 1995.

[18] W.W Hager. Rates of convergence for discrete approximations to unconstrained control problems. *SIAM Journal on Numerical Analysis*, 23:449–472, 1976.

[19] G.W Reddien. Collocation at gauss points as a discretization in optimal control. *SIAM Journal on Control and Optimization*, 17:298–306, 1979.

[20] K. Malanowski. Convergence of approximations vs. regularity of solutions for convex, control constrained optimal control problems. *Applied Mathematics And Optimization*, 13:69–95, 1981.

[21] A. Dontchev. Discrete approximations in optimal control. in: Nonsmooth analysis and geometric methods in deterministic optimal control. *Mathematics and its Applications*, 78:59–80, 1996.

[22] W.W Hager. Runge-kutta methods in optimal control and the transformed adjoint system. *Numerische Mathematik*, 87:247–282, 2000.

[23] J.A Pietz. Pseudospectral collocation methods for the direct transcription of optimal control problems. Master's thesis, Rice university Houston, 2003.

[24] Brian Fabien. *Analytical System Dynamics*. Modeling and Simulation. Springer US, 2009.

[25] Uri M. Ascher and Linda R. Petzold. Projected implicit runge-kutta methods for differential-algebraic equations. *Computational Optimization and Applications*, 28:1097–1120, 1991.

[26] Joel Andersson, Johan Akesson, and Moritz Diehl. Dynamic optimization with casadi.

[27] R. Fitzhugh. Impulses and physiological states in theoretical models of nerve membranes. *Biophysics J*, pages 455–466, 1961.

[28] John Guckenheimer and Philip Holmes. *Non-linear Oscillations, Dynamical systems, and Bifucartions of vector fields.* Applied mathematical sciences. Springer-Verlag, 1997.

[29] G. A Hicks and W. H Ray. Approximation methods for optimal control synthesis. *The Canadian Journal of Chemical Engineering*, 49:522–528, 1971.

[30] Francesco Casella, Filippo Donida, and Johan Akesson. Object oriented modeling and optimal control: A case study in power plant start-up. *18th IFAC World Congress, Milano, Italy*, 2011.

[31] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[32] Joel Andersson. *A General-Purpose Software Framework for Dynamic Optimization.* PhD thesis, University of Leuven, 2013.

[33] Carl D. Laird, Angelica V.Wong, and Johan Akesson. Parallel solution of large-scale dynamic optimization problems. *Elsevier B.V*, 2011.

TRITA-MAT-E 2014:50
ISRN-KTH/MAT/E—14/50-SE

www.kth.se