Collocation Methods for Optimization in a Modelica Environment

Fredrik Magnusson[a] Johan Åkesson[a,b]
[a]Department of Automatic Control, Lund University, Sweden
[b]Modelon AB, Lund, Sweden

Abstract

The solution of generic dynamic optimization problems described by Modelica, and its extension Optimica, code using direct collocation methods is discussed. We start by providing a description of dynamic optimization problems in general and how to solve them by means of direct collocation. Next, an existing implementation of a collocation algorithm in JModelica.org, using CasADi and IPOPT, is presented. The extensions made to this implementation are reported.

The new implementation is compared to an old C-based collocation algorithm in JModelica.org in two benchmarks. The presented benchmarks are based on a continuously stirred tank reactor and a combined cycle power plant. The new algorithm and its surrounding framework is more flexible and shown to be several times more efficient than its predecessor.

Keywords: dynamic optimization; JModelica.org; collocation; nonlinear programming; CasADi

1 Introduction

Optimization of large-scale dynamic systems is becoming a standard industrial technology. Applications include minimization of material and energy consumption during set-point transitions in power plants and chemical processes, minimizing lap times for vehicle systems and trajectory optimization in robotics.

There are different kinds of dynamic optimization problems and in this paper we consider two categories. The first is optimal control, where the aim is to find control variable trajectories (and possibly parameters) that minimize, for example, the amount of resources spent to perform a specified action. The second category is parameter estimation, where the problem is to find the values of unknown model parameters that allow the model to behave according to some given measurement data.

Solving dynamic optimization problems is useful in many different fields and applications. Parameter estimation is used to improve physical models in general. Optimal control has many applications, in both on-line and off-line settings. On-line optimal control is usually done in the form of model predictive control. Off-line applications include finding optimal trajectories for the transition between two stationary operating conditions in a system, which can be used either as a reference during manual control or as a target for automatic control if combined with feedback. Another example is the identification of system bottlenecks, for example by analyzing adjoint variables.

There are many approaches to solving dynamic optimization problems. Until the 1970s, problems were typically solved using dynamic programming or Pontryagin's maximum principle. These approaches are ill-suited for large-scale problems and have trouble handling inequality constraints. Modern techniques often involve finding an approximate solution to the infinite-dimensional optimization problem by transcribing it into a finite-dimensional nonlinear program (NLP). These are called direct methods. The main difference among direct approaches is how to handle the constraints describing the system dynamics. In this paper, direct collocation is used. Another common approach is direct multiple shooting. See [1] and [2] for overviews on different direct methods.

JModelica.org [3] is a tool targeting large-scale dynamic optimization. The system dynamics are described using Modelica, and the optimization formulation is done with the use of the Modelica extension Optimica [4]. In this paper, we implement an optimization algorithm in JModelica.org for solution of dynamic optimization problems described by Modelica and Optimica code. This work is a continuation of the work begun in [5], where CasADi and JModelica.org were integrated and a prototypical collocation method was implemented based on this integra-

This work was supported by the Swedish Research Council through the LCCC Linnaeus Center. We would also like to thank Francesco Casella for letting us use the combined cycle power plant model.

tion. This prototype has since been refined and extended to support additional problem formulations and solution techniques. Additional benchmarks have also been performed, as reported in [6].

The outline of the paper is as follows. In Section 2, a general class of dynamic optimization problems is presented. In Section 3, we discuss how to solve this class of problems using direct collocation. In Section 4, the prominent tools used to implement the described collocation method in a Modelica environment are presented. In Section 5, we present the extensions made to the implementation from previous work. In Section 6, the implemented algorithm is compared to a similar existing algorithm. The two algorithms are applied to a continuously stirred tank reactor and to a combined cycle power plant. Finally, in Section 7, the paper is summarized and some future work is discussed. The work presented in this paper is a result of [6], where additional details are available.

Throughout the paper, the following notation is used. The integer interval from $a \in \mathbb{Z}$ to $b \in \mathbb{Z}$ is denoted by $[a..b]$. All kinds of products between scalars, vectors and matrices are denoted by the binary operator $\cdot$. The space of functions continuous of order k from $\mathbb{R}^m$ into $\mathbb{R}^n$ is denoted by $C^k(\mathbb{R}^m, \mathbb{R}^n)$, where $k = -1$ means that the functions may be discontinuous. No distinction between tuples and vectors is made.

2 Dynamic optimization

We consider systems whose dynamics are described by a single and fully implicit differential algebraic equation (DAE) system of index one (or zero). That is, an equation system of the form

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

where $t \in \mathbb{R}$ is the sole independent variable: time, $x \in C^0(\mathbb{R}, \mathbb{R}^{n_x})$ is the state, $u \in C^{-1}(\mathbb{R}, \mathbb{R}^{n_u})$ is the vector-valued control variable, $w \in C^{-1}(\mathbb{R}, \mathbb{R}^{n_w})$ is the vector-valued algebraic variable and $p \in \mathbb{R}^{n_p}$ is the vector of parameters to be optimized, that is, the free parameters. Initial conditions are also given on a fully implicit form, i.e.,

$$F_0(\dot{x}(t_0), x(t_0), u(t_0), w(t_0), p) = 0,$$

where t_0 is the start time. For ease of notation, we compose the time-dependent variables into a single variable z, that is,

$$z := (\dot{x}, x, u, w).$$

The system dynamics are thus fully described by

$$\begin{aligned} F(t, z(t), p) &= 0, \quad \forall t \in [t_0, t_f], \\ F_0(z(t_0), p) &= 0, \end{aligned}$$

where t_f is the final time and

$$\begin{aligned} F &\in C^2(\mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R}^{n_x + n_w}), \\ F_0 &\in C^2(\mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R}^{n_x}), \\ n_z &:= 2 \cdot n_x + n_u + n_w. \end{aligned}$$

These continuity requirements, and some of the continuity requirements stated later in this section, are needed to establish the second-order optimality conditions and also to find a solution to the first-order optimality condition using some variation of Newton's method.

The general problem studied in this paper is to

$$\begin{aligned} &\text{minimize} && f(t_0, t_f, z, p), && \text{(1a)} \\ &\text{with respect to} && t_0, t_f, z, p, \\ &\text{subject to} && F(t, z(t), p) = 0, && \text{(1b)} \\ & && F_0(z(t_0), p) = 0, && \text{(1c)} \\ & && z_L \le z(t) \le z_U, && \text{(1d)} \\ & && p_L \le p \le p_U, && \text{(1e)} \\ & && g_e(t_0, t_f, t, z(t), p) = 0, && \text{(1f)} \\ & && g_i(t_0, t_f, t, z(t), p) \le 0, && \text{(1g)} \\ & && G_e(t_0, t_f, Z_e, p) = 0, && \text{(1h)} \\ & && G_i(t_0, t_f, Z_i, p) \le 0, && \text{(1i)} \\ & && \forall t \in [t_0, t_f]. \end{aligned}$$

The objective (1a) can take on many forms. For optimal control problems, it is typically a Bolza functional, that is, a function on the form

$$\begin{aligned} f(t_0, t_f, z, p) = &\phi(t_0, z(t_0), t_f, z(t_f), p) + \\ &\int_{t_0}^{t_f} L(t, z(t), p)\, \mathrm{d}t, \end{aligned} \tag{2}$$

where

$$\phi \in C^2(\mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R})$$

is called the Mayer term and

$$L \in C^2(\mathbb{R} \times \mathbb{R}^{n_z} \times \mathbb{R}^{n_p}, \mathbb{R})$$

is called the Lagrange integrand.

For parameter estimation, the objective function is typically formulated using a weighted least squares sum, penalizing the deviation of the measured variables from the discrete measurement data. However,

DOI 10.3384/ecp12076649

in this paper we choose a slightly different approach. We first interpolate the discrete measurement data to form $y_m \in C^0(\mathbb{R}, \mathbb{R}^{n_y})$, where n_y is the number of measured variables. This function gives the approximated trajectories for the vector-valued measured variable $y \in C^{-1}(\mathbb{R}, \mathbb{R}^{n_y})$. Any of the states, algebraic variables and control variables can be measured variables. The objective is then chosen as a continuous weighted least squares function, given by

$$f(z,p) = \int_{t_0}^{t_f} (y(t) - y_m(t)) \cdot Q \cdot (y(t) - y_m(t)) \, \mathrm{d}t, \quad (3)$$

where $Q \in \mathbb{R}^{n_y \times n_y}$ is the weighting matrix. The reason for this approach is discussed in Section 3.

The constraints (1b) and (1c) enforce the system dynamics and initial conditions. The constraints (1d) and (1e) are variable bounds, which are enforced during the entire time horizon $[t_0, t_f]$, where $z_L \in (\mathbb{R} \cup \{-\infty\})^{n_z}$ and $p_L \in (\mathbb{R} \cup \{-\infty\})^{n_p}$ are the lower bounds and $z_U \in (\mathbb{R} \cup \{\infty\})^{n_z}$ and $p_U \in (\mathbb{R} \cup \{\infty\})^{n_p}$ are the upper bounds. The constraints (1f) and (1g) are called path constraints. These can for example be used to describe that a vehicle must follow a certain path. Finally, the constraints (1h) and (1i) are called point constraints. These are similar to the path constraints, with the difference being that they are only enforced at specific time points, rather than during the entire time horizon. The vectors Z_e and Z_i contain the variable values at all the time points used in the point constraints, i.e.

$$Z_e = (z(T_1), z(T_2), \ldots, z(T_m)),$$

where T_i is the time point at which point constraint i is enforced and m is the number of constraint points. A typical example of a point constraint is terminal constraints, where variable values are specified at the end of the time horizon. The path constraint functions g_e and g_i as well as the point constraint functions G_e and G_i must be twice continuously differentiable.

The general problem formulation (1) covers a large class of problems. The constraints (1d) to (1i) are optional, whereas the constructs in (1a) to (1c) are required to get a well-posed problem. The start and final time can be either fixed or free. For example, letting the final time be free and choosing the cost function as $f(t_0, t_f, z, p) = t_f$ allows for the formulation of minimum time problems, where the goal is to minimize the time required to perform some action, often specified in the form of terminal constraints.

A possible generalization of (1) is the division of the time horizon into multiple phases, where at the phase boundaries the DAE system is allowed to change and/or the states may be discontinuous. Another possible generalization is enforcing continuity for control and algebraic variables and then including their respective derivatives in the constraints and cost function. These generalizations are however outside the scope of this paper.

3 Collocation methods

3.1 Collocation polynomials

We will now describe how to solve the dynamic optimization problem (1) by means of direct collocation, using an approach similar to the ones described in [1] and [7]. The time horizon is discretized into n_e elements, and within element i the time-dependent variable z is approximated using a vector-valued polynomial $z_i = (\dot{x}_i, x_i, u_i, w_i)$, called a collocation polynomial. In element i, the time is normalized according to

$$\tilde{t}_i(\tau) = t_{i-1} + h_i \cdot (t_f - t_0) \cdot \tau, \quad \tau \in [0,1], \quad \forall i \in [1..n_e], \quad (4)$$

where t_i is the time at the end of element i, which is called the mesh point of element i, and h_i is the length of element i. The element lengths have been normalized so that the sum of all lengths equals 1. This normalization facilitates the optimization of t_f and t_0 by keeping element lengths constant.

The collocation polynomials are formed by choosing a number n_c of collocation points (which is assumed to be the same for each element). Let $\tau_{i,k}$ denote collocation point k in element i, and let $z_{i,k} = (\dot{x}_{i,k}, x_{i,k}, u_{i,k}, w_{i,k})$ denote the value of $z(\tau_{i,k})$. The collocation polynomials are then formed using Lagrange interpolation polynomials, using the collocation points as interpolation points. Since the states need to be continuous even at the element boundaries, we introduce an additional interpolation point at the start of each element for the state collocation polynomials, denoted by $\tau_{i,0} := 0$. We thus get the collocation polynomials

$$\begin{aligned} x_i(\tau) &= \sum_{k=0}^{n_c} x_{i,k} \cdot \tilde{\ell}_k(\tau), \quad (5) \\ u_i(\tau) &= \sum_{k=1}^{n_c} u_{i,k} \cdot \ell_k(\tau), \\ w_i(\tau) &= \sum_{k=1}^{n_c} w_{i,k} \cdot \ell_k(\tau), \end{aligned}$$

where $\tilde{\ell}_k$ and ℓ_k are the Lagrange basis polynomials, respectively with and without the additional interpola-

tion point $\tau_{i,0}$. The basis polynomials are given by

$$\tilde{\ell}_k(\tau) = \prod_{l \in [0..n_c] \setminus \{k\}} \frac{\tau - \tau_l}{\tau_k - \tau_l}, \quad \forall k \in [0..n_c],$$
$$\ell_k(\tau) = \prod_{l \in [1..n_c] \setminus \{k\}} \frac{\tau - \tau_l}{\tau_k - \tau_l}, \quad \forall k \in [1..n_c].$$

Note that the basis polynomials are the same for all elements, due to the normalized time.

In order to obtain the polynomial approximation of the state derivative $\dot{x}$ in element i, the collocation polynomial x_i is differentiated with respect to time. Using (4), (5) and the chain rule, we obtain

$$\dot{x}_i(\tau) = \frac{\mathrm{d}x_i}{\mathrm{d}\tilde{t}_i}(\tau) = \frac{\mathrm{d}\tau}{\mathrm{d}\tilde{t}_i} \cdot \frac{\mathrm{d}x_i}{\mathrm{d}\tau}(\tau) = \frac{1}{h_i \cdot (t_f - t_0)} \cdot \sum_{k=0}^{n_c} x_{i,k} \cdot \frac{\mathrm{d}\tilde{\ell}_k}{\mathrm{d}\tau}(\tau). \tag{6}$$

There are different schemes for choosing the collocation points $\tau_{i,k}$, with different numerical properties, in particular regarding stability and order of convergence. The most common ones are called Gauss, Radau and Lobatto collocation. In this paper we use Radau collocation, which always places a collocation point at the end of each element, and the rest are chosen in a manner that maximizes accuracy.

Collocation methods are not only used for optimization purposes, but are also widely used for numerical solution of both ODE and DAE systems, i.e. simulation. The concepts are the same in both simulation and optimization, and there is a theoretical basis shared by collocation methods in the two areas. See [8] for more on simulation using collocation methods, which are a special case of implicit Runge-Kutta methods.

3.2 Transcription of the dynamic optimization problem

In this section, the infinite-dimensional dynamic optimization problem (1) is transcribed into a finite-dimensional NLP, using the collocation polynomials constructed in the previous section. The main idea is that the infinite-dimensional time-dependent variable z is approximated using polynomials, which can be represented using a finite number of values: the collocation point values. This finite-dimensional approximation of the solution z is more suitable when employing numerical optimization methods.

As decision variables in the NLP we choose all the collocation point values $z_{i,k}$, the state values at the start of each element $x_{i,0}$ and the free parameters p. We also choose the initial condition values as NLP variables, which we denote by $z_{1,0}$. Finally, we choose t_0 and t_f as optimization variables if they are free. We thus let

$$\begin{aligned} Z = (&z_{1,0}, z_{1,1}, z_{1,2}, \ldots, z_{1,n_c}, \\ &x_{2,0}, z_{2,1}, z_{2,2}, \ldots, z_{2,n_c}, \\ &x_{3,0}, z_{3,1}, z_{3,2}, \ldots, z_{3,n_c}, \\ &\vdots, \\ &x_{n_e,0}, z_{n_e,1}, z_{n_e,2}, \ldots, z_{n_e,n_c}, p, t_0, t_f). \end{aligned}$$

be the vector containing all the NLP variables. There are other possibilities in the choice of NLP decision variables, and the choice depends on the collocation scheme. With Radau collocation and the above choice, the transcription of (1) results in the following NLP:

$$\begin{aligned}
&\text{min.} && \tilde{f}(Z), && \text{(7a)}\\
&\text{w.r.t.} && Z \in \mathbb{R}^{n_Z}, \\
&\text{s.t.} && F(t_{i,k}, z_{i,k}, p) = 0, && \text{(7b)}\\
& && F_0(z_{1,0}, p) = 0, && \text{(7c)}\\
& && u_{1,0} - \sum_{k=1}^{n_c} u_{1,k} \cdot \ell_k(0) = 0, && \text{(7d)}\\
& && z_L \le z_{i,k} \le z_U, && \text{(7e)}\\
& && p_L \le p \le p_U, && \text{(7f)}\\
& && g_e(t_{i,k}, z_{i,k}, p) = 0, && \text{(7g)}\\
& && g_i(t_{i,k}, z_{i,k}, p) \le 0, && \text{(7h)}\\
& && G_e(Z_e) = 0, && \text{(7i)}\\
& && G_i(Z_i) \le 0, && \text{(7j)}\\
& && \forall (i,k) \in \{(1,0)\} \cup ([1..n_e] \times [1..n_c]), \\
& && \dot{x}_{j,l} = \frac{1}{h_j \cdot (t_f - t_0)} \cdot \sum_{m=0}^{n_c} x_{j,m} \cdot \frac{\mathrm{d}\tilde{\ell}_m}{\mathrm{d}\tau}(\tau_l), \\
& && \forall (j,l) \in [1..n_e] \times [1..n_c], && \text{(7k)}\\
& && x_{n,n_c} = x_{n+1,0}, \quad \forall n \in [1..n_e - 1], && \text{(7l)}
\end{aligned}$$

where

$$n_Z = (1 + n_e \cdot n_c) \cdot n_z + (n_e - 1) \cdot n_x + n_p + 2$$

is the number of scalar NLP variables and

$$t_{i,k} := \tilde{t}_i(\tau_k)$$

denotes collocation point k in element i. The objective (1a) is transcribed into (7a). In the case of optimal control, the Mayer term of the Bolza functional (2) is straightforward to transcribe as

$$\phi(t_0, z(t_0), t_f, z(t_f), p) = \phi(t_0, z_{1,0}, t_f, z_{n_e,n_c}, p).$$

DOI 10.3384/ecp12076649

To transcribe the Lagrange term, we start by using (4) to define the Lagrange integrand in element i as

$$L_i(\tau, z_i(\tau), p) := L\left(\tilde{t}_i(\tau), z\left(\tilde{t}_i(\tau)\right), p\right).$$

The Lagrange term is then approximated as follows.

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

where the quadrature weights ω_k are given by

$$\omega_k = \int_0^1 \ell_k(\tau)\,\mathrm{d}\tau.$$

These quadrature weights provides the best approximation for these interpolation points, as shown in [1]. The optimal control objective is thus transcribed as

$$\begin{aligned}
f(z,p) \approx\ & \phi(t_f, z_{n_e,n_c}, p) + \\
& \sum_{i=1}^{n_e} \left(h_i \cdot (t_f - t_0) \cdot \sum_{k=1}^{n_c} \omega_k \cdot L_i\left(\tau_k, z_{i,k}, p\right) \right) \\
=:\ & \tilde{f}(Z).
\end{aligned}$$

For the parameter estimation problem, the continuous weighted least squares integral (3) is approximated using the same Gaussian quadrature, resulting in

$$\begin{aligned}
f(z,p) &= \int_{t_0}^{t_f} (y(t) - y_m(t)) \cdot Q \cdot (y(t) - y_m(t))\,\mathrm{d}t \\
&\approx \sum_{i=1}^{n_e} \left(h_i \cdot (t_f - t_0) \cdot \sum_{k=1}^{n_c} \omega_k \cdot (y_{i,k} - y_m(t_{i,k})) \cdot Q \cdot \right. \\
&\qquad \left. (y_{i,k} - y_m(t_{i,k})) \right) =: \tilde{f}(Z),
\end{aligned}$$

where y_i denotes the collocation polynomials for the measured variables, and $y_{i,k}$ denotes the corresponding collocation point values.

The system dynamics constraint (1b) is only enforced at the collocation points and the start time in the NLP, rather than during the entire time horizon. The initial conditions (1c) are straightforward to transcribe into (7c), since all the initial values have been chosen as NLP variables. The consistency of the user-provided initial conditions is ensured by enforcing all the dynamic constraints at the start time.

The initial values for the states and algebraic variables are determined by the dynamic and initial constraints. The initial value for the control variable is however not governed by the dynamic or initial equations, but is instead given by the collocation polynomial u_1. To obtain the value for $u_{1,0}$, we thus need to add the extrapolation constraint (7d).

The bounds and path constraints (1d) to (1g) are straightforward to transcribe into (7e) to (7h), by only enforcing them at the collocation points. How to transcribe the point constraints (1h) and (1i) is less obvious. The approach we have chosen is to assume that each constraint point coincides with a collocation or mesh point. It is then just a matter of identifying the NLP variables that correspond to the constraint point values Z_e and Z_i in order to transcribe the point constraints into (7i) and (7j). The other possibility is to evaluate the collocation polynomials at the constraint points. These constraints are however more computationally expensive to evaluate. But adding elements in order to line up the mesh with the constraint points is prone to be even more expensive. However, as long as the number of constraint points are few in number, which often is the case, this is not a critical issue.

A similar situation occurs during parameter estimation if a discrete least squares sum is used as the objective. The measured variable values are then needed at each of the measurement time points, and these are typically not few in number. The question of whether to line up the mesh (or even collocation) points with the measurement time points, or to simply evaluate the collocation polynomials, is then a critical choice. In this paper however, we avoid this issue by instead using the continuous least squares objective (3). This allows us to evaluate the objective using quadrature, for which we only need the variable values at the collocation points, which are readily available.

In order to determine the state derivative values at the collocation points, we enforce equation (6) at all the collocation points, giving us the collocation equations (7k). These are not enforced at the start time, where the state derivative values instead are determined by the DAE system and initial conditions.

Finally, we add the continuity constraints (7l), to get continuity for the state. An NLP has the general form

$$\begin{aligned}
&\text{minimize} && f(x) \\
&\text{with respect to} && x \in \mathbb{R}^{n_x}, \\
&\text{subject to} && x_L \le x \le x_U, \\
& && g(x) = 0, \\
& && h(x) \le 0,
\end{aligned}$$

which the transcription (7) is a special case of. By solving the NLP (7), we may obtain an approximate solution to the dynamic optimization problem (1).

4 Tools

4.1 CasADi

Obtaining the first and second-order derivatives of the NLP cost and constraints functions with respect to the NLP variables allows for efficient solution of an NLP. To this end, CasADi [9] (**C**omputer **a**lgebra **s**ystem with **A**utomatic **Di**fferentaion) is used. CasADi is a low-level tool for efficiently computing derivatives using automatic differentiation (AD) and is tailored for dynamic optimization. Once a symbolic representation of an NLP has been created using CasADi, the needed derivatives are efficiently and conveniently obtained and sparsity patterns are preserved.

To solve the NLP (7), we use IPOPT [10]. IPOPT uses a sparse primal-dual interior point method to find local optima to large-scale NLPs. CasADi comes with an interface to IPOPT, which is used in the implementation.

4.2 JModelica.org

4.2.1 The JModelica.org platform

JModelica.org [3] is an open-source platform for simulation and optimization of Modelica models. Whereas standard Modelica tools, such as Dymola[1] and OpenModelica[2], mostly focus on the simulation of physical systems, JModelica.org also targets large-scale dynamic optimization. A common problem is that a large amount of research goes into developing algorithms without accompanying means of describing complex physical systems, making these algorithms difficult to use in practical applications. One goal of JModelica.org is to open up the Modelica language and the vast amount of existing Modelica models to algorithms developed in academia.

The Modelica language is largely designed with simulation-based analysis in mind. To accommodate the need for conveniently formulating dynamic optimization problems based on models described by Modelica code, the Modelica extension Optimica [4] has been developed and integrated into JModelica.org. Optimica enables the extension of a Modelica model to include the constructs used to formulate a dynamic optimization problem, such as (1), where the pure Modelica code describes the dynamic constraints (1b) and (1c).

[1] http://www.3ds.com/products/catia/portfolio/dymola

[2] http://www.openmodelica.org/

The main components of JModelica.org are the Modelica and Optimica compilers, which are implemented in Java, and the three modeling interfaces Functional Mock-up Interface (FMI)[3], JModelica.org Model Interface (JMI) and a new symbolic XML-based format based on the FMI XML format, which includes equations in symbolic form. The user interacts with the various components of JModelica.org via the scripting language Python.

While FMI is a standard defining a tool-independent format for representation of hybrid dynamic models on ordinary differential equation (ODE) form, JMI is a runtime library designed solely for JModelica.org, and has long been the main interface for dynamic optimization in JModelica.org. The main optimization algorithm in JMI is collocation-based and implemented in C. It relies on CppAD[4] to compute and evaluate derivatives. However, in this paper the new XML-based format is instead used for the new collocation algorithm. This format is an extension of the XML format used in FMI and is described in [11]. The format uses a DAE representation of the model instead of an ODE representation. It is designed to use a model representation that is as general as possible, allowing for the formulation of a wide variety of problems based on Modelica code, in particular dynamic optimization problems described by Optimica code. CasADi supports import of models described by this XML format, allowing for smooth interaction between JModelica.org and CasADi.

4.2.2 The collocation algorithm toolchain

Figure 1 depicts an overview of the entire workflow for the implemented collocation algorithm.

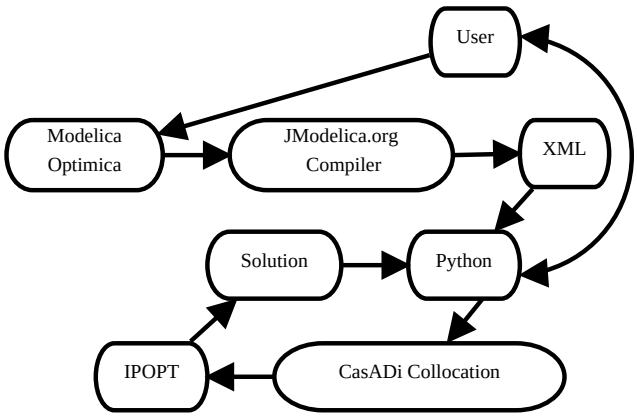

Figure 1: Overview of algorithm workflow

[3] http://www.functional-mockup-interface.org/

[4] http://www.coin-or.org/CppAD/

DOI 10.3384/ecp12076649

The user starts by defining the system model in Modelica and the dynamic optimization problem in Optimica. The user interaction is carried out in Python. The Optimica file is, via Python, sent to JModelica.org's compiler. The compiler generates an XML file from the Optimica file, which has a flat, rather than hierarchical, representation of the dynamic optimization problem similar to that of (1).

The XML file is parsed by CasADi and JModelica.org and the extracted information is used to transcribe the problem into an NLP by the collocation algorithm inside JModelica.org. This NLP problem is then solved by IPOPT. The solution is written to a result file in a format compliant with Dymola. The solution is also represented by a Python object which is returned to the user. This allows the user to freely analyze the data in Python, e.g. plotting it either manually or using JModelica.org's plotting GUI.

5 Implementation extensions

The work presented in this paper is a continuation of the work begun in [5], where a prototypical collocation algorithm was implemented in JModelica.org using CasADi in Python. In this section we describe the prominent extensions made to this implementation.

The algorithm supports problems with free start and final time. Since these are typically combined with terminal constraints, support for general point constraints has also been added.

Whereas the old implementation only supported Radau collocation with three collocation points per element, the new implementation supports an arbitrary number of collocation points (up to about 80 points, at which point the method for computing the collocation points runs into numerical problems). The new implementation also supports Gauss collocation as an alternative to Radau collocation.

CasADi has two separate approaches to performing automatic differentiation. The first approach is called SX and is a conventional AD approach, where the computation graph is only allowed to contain scalar, built-in unary and binary operations. The second approach is called MX and allows for more general operations in the computation graph, such as matrix operations (preserving sparsity), branches and user-defined functions. The novel MX graphs are less computationally efficient than SX graphs, but support a wider range of operations. This allows the resulting MX graphs to be smaller than SX graphs, thus consuming less memory, which may be critical. The collocation algorithm has been extended to enable the user to choose between SX and MX graphs. See [9] for more details regarding SX and MX graphs.

The collocation algorithm only deals with systems which are continuous in time. However, control signals are often inherently discrete in time, which can not be disregarded in for example model predictive control. In order to support discrete control signals, the possibility of adding blocking factors has been added. Blocking factors change the representation of control signals from piecewise polynomial to piecewise constant. Control signals can be forced to remain constant over single or multiple elements.

Finally, options have been added to allow the elimination of certain NLP variables. The state derivative variables $\dot{x}_{i,k}$ can be eliminated by inlining the collocation constraint (7k), and the state continuity variables $x_{i,0}$ can be eliminated by inlining the continuity constraint (7l). This allows for the trade-off between problem size and problem sparsity. Eliminating state derivatives also has the benefit of no longer needing to scale these variables, which often is difficult.

6 Benchmarks

6.1 Benchmark setting

In this section, we will compare the newly extended collocation algorithm based on CasADi and its Python interface with the old collocation algorithm implemented in C. Both of these algorithms are implemented in JModelica.org. We use Radau collocation with the same, low number of collocation points per element. The benchmarks are based on a continuously stirred tank reactor and a combined cycle power plant.

The two algorithms are based on the same theory and the constructed NLP problems are nearly identical, so the solutions can be expected to also be nearly identical. There are however a few differences. The most prevalent is that the new algorithm constructs AD graphs for the entire NLP. The computation of the Hessian of the Lagrangian function is thus easy and efficient. Obtaining this information for the old algorithm using CppAD, although possible, would require a tremendous effort to implement, which has not been done. Thus IPOPT employs a quasi-Newton method for the old algorithm, in which the Hessian instead is approximated. The computation of the Hessian and AD graphs for the entire NLP is expensive. These computations can however be performed off-line, and in turn make the on-line computations more effective.

In this benchmark, SX graphs are used for the new algorithm, since the generality offered by MX graphs are unnecessary for the presented benchmarks.

All the benchmarks are run on a Fedora 16 computer with an Intel® Core™ i7-2600 Quad processor @ 3.4 GHz. Revisions [3352] and [2594] of JModelica.org and of CasADi respectively are used, together with version 3.10.2 of IPOPT with the MA27 linear solver. For each benchmark, we provide the following run-time statistics:

- Off-line: The CPU time [s] spent doing off-line computations, which includes compilation of the Modelica and Optimica code, construction of AD graphs and computation of derivatives of NLP functions.
- On-line: The CPU time [s] spent doing on-line comptuations, which essentially is the time spent in IPOPT. This part consists of two parts, where the first one is the time spent internally in IPOPT, and the second part is the time spent evaluating NLP functions, which is done by CppAD or CasADi. The time spent by CasADi evaluating NLP functions is nearly negligible, whereas CppAD spends a significant amount of time evaluating functions on-line for the old algorithm.
- Total: The total CPU time [s] from the start of the compilation until the optimization result is returned.
- Iterations: The number of iterations required by IPOPT to solve the problem.

Minor variations in the collocation scheme or problem formulation can have a tremendous impact on the required number of iterations, for example if the solver has to enter a restoration phase, which in turn affect the overall solution time. But on average, the required number of iterations for the two algorithms should be similar for a specific problem. The only significant reason to expect a different number of iterations is due to that the new algorithm computes second-order derivatives analytically, whereas the old algorithm approximates them numerically. The number of iterations for the new algorithm can thus be expected to be lower on average.

6.2 Continuously stirred tank reactor

The continuously stirred tank reactor (CSTR) model used for this benchmark was developed in [12]. The system contains a highly nonlinear exothermic reaction and has two states: reactant concentration c [mol/m^3] and reactor temperature T [K]. The rate F_0 [m^3/s], concentration c_0 [mol/m^3] and temperature T_0 [K] of the reactant inflow are assumed to be constant. The reactor has a liquid cooling system, whose temperature T_c [K] is the sole control variable.

A formulation analogous to (1) of the considered problem is to

$$\begin{aligned}
&\text{min.} && \phi(t_f), && \text{(8a)}\\
&\text{w.r.t.} && c, T, T_c, \phi, \\
&\text{s.t.} && \dot{c}(t) = F_0 \cdot \frac{c_0 - c(t)}{V} - k_0 \cdot e^{-\frac{E_a}{T(t)}} \cdot c(t), && \text{(8b)}\\
& && \dot{T}(t) = F_0 \cdot \frac{T_0 - T(t)}{V} - \\
& && \qquad \frac{H}{\rho \cdot C_p} \cdot k_0 \cdot e^{-\frac{E_a}{T(t)}} \cdot c(t) + \\
& && \qquad \frac{2 \cdot U}{r \cdot \rho \cdot C_P} \cdot (T_c(t) - T(t)), && \text{(8c)}\\
& && \dot{\phi}(t) = \left|\left|(c(t), T(t), T_c(t)) - \left(c^{\text{ref}}, T^{\text{ref}}, T_c^{\text{ref}}\right)\right|\right|_2^2 && \text{(8d)}\\
& && (c(t_0), T(t_0), T_c(t_0), \phi(t_0)) = (c_0, T_0, T_{c0}, 0), && \text{(8e)}\\
& && (T(t), T_c(t)) \leq (350, 370), && \text{(8f)}\\
& && \forall t \in [t_0, t_f].
\end{aligned}$$

The objective (8a) is to move the system from the stationary operation point given by the initial condition (8e), where

$$(c_0, T_0, T_{c0}) \approx (956.3, 250.1, 370.0),$$

to the stationary operation point, given by

$$\left(c^{\text{ref}}, T^{\text{ref}}, T_c^{\text{ref}}\right) \approx (338.8, 280.1, 280.0).$$

The variable ϕ is introduced as a state and measures how the cost increases over time, and is governed by the dynamic equation (8d). This allows us to formulate the objective on Mayer form, instead of Lagrange.

The dynamics of the system are modelled by equations (8b) and (8c), where $V, k_0, E_A, H, \rho, C_p, U$ and r are physical parameters and constants. In order to avoid too high temperatures, we impose the variable bounds (8f). With $t_f = 200$ s, $n_e = 70$ and $n_c = 5$, we get the following result.

Table 1: Run-time statistics for the CSTR benchmark

	Off-line	On-line	Total	Iterations
New alg.	1.0	0.3	1.3	50
Old alg.	2.0	0.9	2.9	62

DOI 10.3384/ecp12076649

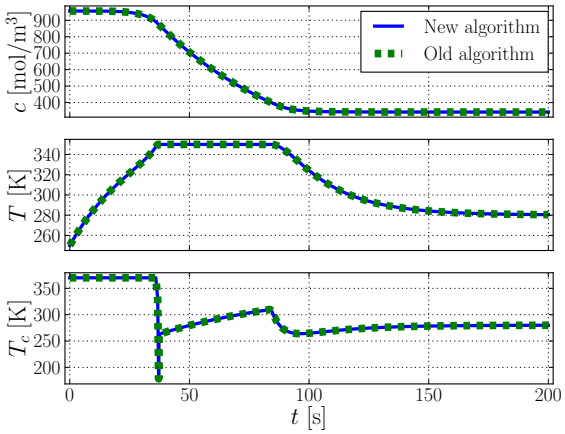

Figure 2: Comparison of the old and new algorithm on optimal control of a CSTR

We see that for this benchmark, the new algorithm is about twice as fast both off-line and on-line. They also produce the same solution (up to IPOPT tolerances). This problem is very small-scale, and in the next benchmark we will see that a larger problem will allow the new algorithm to truly outperform the old one

6.3 Combined cycle power plant

The combined cycle power plant (CCPP) model used for this benchmark is described in [13]. The model has 9 states, 128 algebraic variables and 1 control variable. The task is to minimize the time required to perform a warm start-up of the power plant. This problem has become highly industrially relevant during the last few years, due to an increasing need to improve power generation flexibility. The startup process is considered finished when the normalized load input signal u [1] to the steam turbine, starting at 15 %, has reached 100 % and the evaporator pressure p [Pa], which is a state with an initial value of approximately 3.47 MPa, has reached approximately 8.35 MPa.

In order to reduce the wear and tear on the steam turbine, which is one of the most expensive parts of the power plant, the thermal stress in the turbine σ [Pa], which is an algebraic variable, may not exceed 260 MPa. This is the main limiting factor in the startup process. Another imposed constraint is that the derivative of the load input signal u may not be negative and may not exceed $0.1/60$ s^{-1}. Since these bounds are on the derivative of the control variable, which is not supported by neither the old nor the new algorithm, we introduce the control variable $\dot{u}$ and add the equation

$$\frac{\mathrm{d}u}{\mathrm{d}t} = \dot{u},$$

to the DAE system. This converts the previous control variable u into a state, giving us a total of 10 states, and the sole control variable is now instead $\dot{u}$, which we can impose the mentioned bounds on.

We formulate a Lagrange cost function which penalizes the weighted deviation of the load input signal and the evaporator pressure from their respectively desired values, given by

$$f(z) = \int_{t_0}^{t_f} \Big(10^{-12} \cdot \big(p(t) - 8.35 \cdot 10^6\big)^2 + 0.5 \cdot (u(t) - 1)^2 \Big)\, \mathrm{d}t.$$

With $t_0 = 0$ s, $t_f = 4000$ s, $n_e = 40$ and $n_c = 4$, the following optimization result is obtained.

Table 2: Run-time statistics for the CCPP benchmark

	Off-line	On-line	Total	Iterations
New alg.	4.9	3.0	7.9	79
Old alg.	13.2	23.9	37.2	75

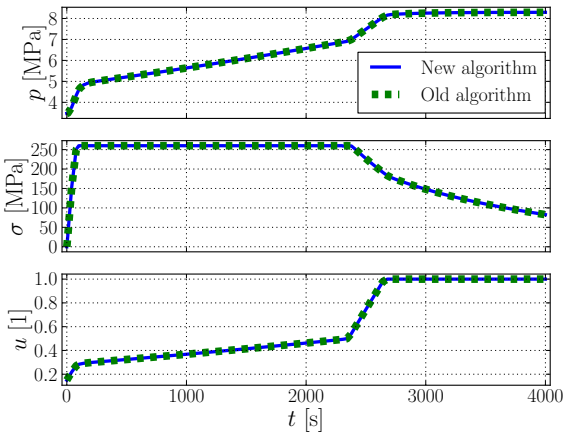

Figure 3: Comparison of the old and new algorithm for optimal start-up of a CCPP

In this case we clearly see the benefits of constructing AD graphs for the entire NLP problem using CasADi for large-scale problems, which is what allows for the exceptionally quick NLP on-line solution. Once again the algorithms find the same solution.

7 Conclusions

We have presented and implemented an optimization algorithm based on existing theory for direct collocation. The algorithm has been compared to an old and similar algorithm in JModelica.org. The solutions found by the two algorithms have shown to be as identical as can be expected, that is, up to IPOPT tolerances.

The overall performance of the new algorithm compared to the old algorithm, in terms of speed, is clearly superior, especially for large-scale problems. In terms of being fully-featured, there are still a few important features missing for the new algorithm. CasADi combined with Python is however very flexible, so adding new features is often straightforward, which is not the case for the old algorithm implemented in C.

Future work includes adding additional discretization schemes, adding support for multi-phase problems and allowing element lengths to be free in order to maximize accuracy. A related topic is the further development of Optimica, to support additional problem formulations.

References

[1] L. T. Biegler, *Nonlinear Programming: Concepts, Algorithms, and Applications to Chemical Processes*. MOS-SIAM Series on Optimization, Mathematical Optimization Society and the Society for Industrial and Applied Mathematics, 2010.

[2] T. Binder, L. Blank, H. Bock, R. Bulirsch, W. Dahmen, M. Diehl, T. Kronseder, W. Marquardt, J. Schlöder, and O. Stryk, "Introduction to model based optimization of chemical processes on moving horizons," in *Online Optimization of Large Scale Systems: State of the Art* (M. Grötschel, S. Krumke, and J. Rambau, eds.), pp. 295–340, Springer, 2001.

[3] J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit, "Modeling and optimization with Optimica and JModelica.org—languages and tools for solving large-scale dynamic optimization problem," *Computers and Chemical Engineering*, vol. 34, pp. 1737–1749, Nov. 2010.

[4] J. Åkesson, "Optimica—an extension of Modelica supporting dynamic optimization," in *In 6th International Modelica Conference 2008*, Modelica Association, Mar. 2008.

[5] J. Andersson, J. Åkesson, F. Casella, and M. Diehl, "Integration of CasADi and JModelica.org," in *8th International Modelica Conference*, Mar. 2011.

[6] F. Magnusson, "Collocation methods in JModelica.org," Master's Thesis ISRN LUTFD2/TFRT--5892--SE, Feb. 2012.

[7] J. T. Betts, *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*. SIAM's Advances in Design and Control, Society for Industrial and Applied Mathematics, 2nd ed., 2010.

[8] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and differential-algebraic problems*. Springer series in computational mathematics, Springer-Verlag, 2nd ed., 1996.

[9] J. Andersson, J. Åkesson, and M. Diehl, "CasADi – A symbolic package for automatic differentiation and optimal control," in *Recent Advances in Algorithmic Differentiation* (S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, eds.), Lecture Notes in Computational Science and Engineering, (Berlin), Springer, 2012.

[10] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[11] R. Parrotto, J. Åkesson, and F. Casella, "An XML representation of DAE systems obtained from continuous-time Modelica models," in *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, (Oslo, Norway), pp. 91–98, Oct. 3 2010.

[12] G. A. Hicks and W. H. Ray, "Approximation methods for optimal control synthesis," *The Canadian Journal of Chemical Engineering*, vol. 49, no. 4, pp. 522–528, 1971.

[13] F. Casella, F. Donida, and J. Åkesson, "Object-oriented modeling and optimal control: A case study in power plant start-up," in *18th IFAC World Congress*, (Milano, Italy), Aug. 2011.