Enhancing Collocation-Based Dynamic Optimization through Adaptive Mesh Refinement

Linus Langenkamp [1] Bernhard Bachmann [1]

[1]Institute for Data Science Solutions, Bielefeld University of Applied Sciences and Arts, Germany,
{first.last}@hsbi.de

Abstract

Direct collocation-based dynamic optimization plays an important role in the optimization of equation-based models. With this approach, continuous problems are transcribed into sparse nonlinear programs (NLPs) that can be solved efficiently. The open-source Modelica environment OpenModelica provides an implementation using Radau IIA collocation, but has major limitations, such as the lack of parameter optimization, no adaptive mesh refinement, and no support for higher-order integration schemes. This paper presents (1) a comprehensive reimplementation that addresses these limitations and (2) a novel h-method mesh refinement algorithm. Implemented in the custom Python / C++ optimization framework GDOPT, the approach demonstrates significant performance improvements, solving typical problems 2 to 3 times faster than OpenModelica under equivalent conditions. Using the proposed mesh refinement algorithm, the framework correctly identifies non-smooth regions and increases resolution accordingly, requiring only a small increase in computation time. The implementation lays the foundation for a future integration into the OpenModelica toolchain.

Keywords: Dynamic Optimization, Direct Collocation, Adaptive Mesh Refinement, Nonlinear Programming

1 Introduction

Dynamic Optimization has application in many different fields, including engineering, systems biology, economics, aerospace engineering and other disciplines where physical models are described by differential-algebraic equations (DAEs). A widely adopted strategy for solving such problems is direct collocation, a method that discretizes the time horizon and replaces the states by polynomials that satisfy the differential equations at discrete nodes. This approach leads to a sparse nonlinear program (NLP) that can be solved with state-of-the-art nonlinear optimizers. The accuracy of the approach relies on a well-designed discretization mesh. While smooth trajectories can be approximated with fewer intervals or higher degree polynomials, non-smooth behavior, e.g. rapid transitions, switching points or nonlinearities, demands high resolution and fine meshes. Adaptive mesh refinement methods, including h- (interval length adjustment), p- (polynomial degree adjustment), and hp-strategies (hybrid), are essential to efficiently obtain accurate solutions and are an active area of research (Liu, Hager, and Rao 2015; M. A. Patterson, Hager, and Rao 2015; Jain and Tsiotras 2008; Y. Zhao and Tsiotras 2009; J. Zhao and Shang 2018).

Several free and commercial tools implement direct collocation methods, including GPOPS-II (M. A. Patterson and Rao 2014), PSOPT (Becerra 2010), SPARTAN (Sagliano et al. 2018), JModelica (Magnusson and Åkesson 2015), and OpenModelica (Ruge et al. 2014; Fritzson, Pop, Abdelhak, et al. 2020). Among them, OpenModelica offers a fully open-source environment that combines expressive equation-based modeling in the Modelica language with optimization capabilities in a single tool. However, its current direct collocation implementation, using Radau IIA methods, has significant limitations, as it lacks support for adaptive mesh refinement, parameter optimization, higher-order collocation schemes and does not use exact second-order derivatives.

To address these limitations, this work proposes a novel h-method mesh refinement algorithm, implemented in a newly developed open-source framework called GDOPT, which is designed for future integration into the OpenModelica toolchain. The primary objective of this paper is to develop a general purpose mesh refinement algorithm that can accelerate the convergence of direct collocation as well as correctly identify non-smooth sections of control trajectories and increase the resolution appropriately.

This paper is organized as follows: Section 2 provides a general introduction and presents the mathematical formulation of dynamic optimization using direct collocation. Based on these considerations, we propose our novel mesh refinement algorithm L2-Boundary-Norm in Section 3, detailing its principal components and properties. Section 4 discusses the direct collocation implementation, focusing on its most important features and workflows. To demonstrate the capabilities of the algorithm and our implementation, Section 5 presents numerical results for a representative control problem, the fuel-optimal start-up of a diesel-electric power train (Sivertsson and Eriksson 2012; Mengist, Ruge, Gebremedhin, et al. 2013; Bachmann et al. 2012), comparing these results with the current OpenModelica implementation. Section 6 examines the limitations of both the framework and mesh refinement algorithm, while Section 7 summarizes the work and pro-

vides an outlook for future work.

2 Dynamic Optimization

2.1 Problem Definition

Consider the dynamic optimization problem

$$\min_{\boldsymbol{u}(t),\boldsymbol{p}} \; M(\boldsymbol{x}(t_f),\boldsymbol{u}(t_f),\boldsymbol{p},t_f) + \int_{t_0}^{t_f} L(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t)\,\mathrm{d}t \tag{1a}$$

s.t.

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t) \qquad \forall t \in T \tag{1b}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{1c}$$

$$\boldsymbol{g}^L \le \boldsymbol{g}(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t) \le \boldsymbol{g}^U \qquad \forall t \in T \tag{1d}$$

$$\boldsymbol{r}^L \le \boldsymbol{r}(\boldsymbol{x}(t_f),\boldsymbol{u}(t_f),\boldsymbol{p},t_f) \le \boldsymbol{r}^U \tag{1e}$$

$$\boldsymbol{a}^L \le \boldsymbol{a}(\boldsymbol{p}) \le \boldsymbol{a}^U \tag{1f}$$

over a fixed time horizon $T = [t_0, t_f]$ with time $t \in T$, referred to as the General Dynamic Optimization Problem (GDOP). The goal is to determine optimal control trajectories $\boldsymbol{u}(t) : T \to \mathbb{R}^{d_{\boldsymbol{u}}}$, time-invariant parameters $\boldsymbol{p} \in \mathbb{R}^{d_{\boldsymbol{p}}}$, and states $\boldsymbol{x}(t) : T \to \mathbb{R}^{d_{\boldsymbol{x}}}$ minimizing the cost functional (1a), subject to the constraints (1b) - (1f). The objective is divided into the *Mayer term* $M(\cdot)$, penalizing the final system state, and the *Lagrange term* $\int_{t_0}^{t_f} L(\cdot)\,\mathrm{d}t$, accounting for accumulated cost over time. The constraints include an initial value problem with *dynamic constraints* (1b) and an *initial condition* (1c), along with algebraic *path constraints* (1d), *final constraints* (1e), and *parametric constraints* (1f). All involved functions $M, L : \mathbb{R}^d \times T \to \mathbb{R}$, $\boldsymbol{f} : \mathbb{R}^d \times T \to \mathbb{R}^{d_{\boldsymbol{x}}}$, $\boldsymbol{g} : \mathbb{R}^d \times T \to \mathbb{R}^{d_{\boldsymbol{g}}}$, $\boldsymbol{r} : \mathbb{R}^d \times T \to \mathbb{R}^{d_{\boldsymbol{r}}}$, and $\boldsymbol{a} : \mathbb{R}^{d_{\boldsymbol{p}}} \to \mathbb{R}^{d_{\boldsymbol{a}}}$ with $d := d_{\boldsymbol{x}} + d_{\boldsymbol{u}} + d_{\boldsymbol{p}}$, are assumed to be twice continuously differentiable. The constraint bounds are given by $\boldsymbol{g}^L, \boldsymbol{g}^U \in \bar{\mathbb{R}}^{d_{\boldsymbol{g}}}$, $\boldsymbol{r}^L, \boldsymbol{r}^U \in \bar{\mathbb{R}}^{d_{\boldsymbol{r}}}$, and $\boldsymbol{a}^L, \boldsymbol{a}^U \in \bar{\mathbb{R}}^{d_{\boldsymbol{a}}}$ with $\bar{\mathbb{R}} := (\mathbb{R} \cup \{-\infty, \infty\})$. This formulation extends the Nonlinear Optimal Control Problem (NOCP) implemented in OpenModelica (Ruge et al. 2014) by including static parameters, which may be present in all model functions.

2.2 Direct Collocation

The continuous GDOP is reduced to an NLP by embedding a collocation scheme based on *flipped Legendre-Gauss-Radau (fLGR)* points. This discretization is equivalent to a transcription using the Radau IIA Runge-Kutta method, where the fLGR collocation nodes are scaled from $[-1, 1]$ to $[0, 1]$. For $[0, 1]$, the nodes c_j, $j = 1, \ldots, m$ are the m roots of the polynomial $(1-t)P^{(1,0)}_{m-1}(2t-1)$, where $P^{(1,0)}_{m-1}$ is an $(m-1)$-th Jacobi polynomial. The Radau IIA method has excellent stability properties, since it is A-, B-, and L-stable, and achieves order $2m-1$ for m collocation nodes.

The time horizon $[t_0, t_f]$ is divided into $n+1$ intervals $[t_i, t_{i+1}]$, $i = 0, \ldots, n$ with lengths $\Delta t_i := t_{i+1} - t_i$. On each interval $[t_i, t_{i+1}]$, collocation nodes $t_{ij} := t_i + c_j \Delta t_i$, $j = 1, \ldots, m_i$, along with the first grid point $t_{i0} := t_i + c_0 \Delta t_i$, where $c_0 = 0$, are introduced. Since $c_{m_i} = 1$ is always included in the Radau IIA scheme, the last grid point of interval $i-1$ matches the first of interval i, i.e. $t_{i-1,m_{i-1}} = t_{i0}$.

States and controls are approximated as $\boldsymbol{x}(t_{ij}) \approx \boldsymbol{x}_{ij}$ and $\boldsymbol{u}(t_{ij}) \approx \boldsymbol{u}_{ij}$, with the initial state fixed as $\boldsymbol{x}_{00} := \boldsymbol{x}_0$. In interval i, the state $\boldsymbol{x}(t)$ is replaced by a Lagrange interpolating polynomial $\boldsymbol{x}_i(t) = \sum_{j=0}^{m_i} \boldsymbol{x}_{ij}\, l_j(t)$ of degree m_i, where

$$l_j(t) := \prod_{\substack{k=0 \\ k \neq j}}^{m_i} \frac{t - t_{ik}}{t_{ij} - t_{ik}} \quad \forall j = 0, \ldots, m_i. \tag{2}$$

This polynomial must satisfy the differential equation (1b) at the collocation nodes t_{ij} for $j = 1, \ldots, m_i$ and match the initial condition $\boldsymbol{x}_{i0}$ from interval $i-1$. These conditions yield

$$\boldsymbol{0} = \begin{bmatrix} D^{(1)}_{10} I & \ldots & D^{(1)}_{1m_i} I \\ \vdots & \ddots & \vdots \\ D^{(1)}_{m_i 0} I & \ldots & D^{(1)}_{m_i m_i} I \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{i0} \\ \vdots \\ \boldsymbol{x}_{im_i} \end{bmatrix} - \Delta t_i \begin{bmatrix} \boldsymbol{f}_{i1} \\ \vdots \\ \boldsymbol{f}_{im_i} \end{bmatrix}, \tag{3}$$

with identity matrix $I \in \mathbb{R}^{d_{\boldsymbol{x}} \times d_{\boldsymbol{x}}}$, $\boldsymbol{f}_{ij} := \boldsymbol{f}(\boldsymbol{x}_{ij}, \boldsymbol{u}_{ij}, \boldsymbol{p}, t_{ij})$, and the first differentiation matrix $D^{(1)}_{jk} := \frac{\mathrm{d}\tilde{l}_k}{\mathrm{d}\tau}(c_j)$, where

$$\tilde{l}_k(\tau) := \prod_{\substack{j=0 \\ j \neq k}}^{m_i} \frac{\tau - c_j}{c_k - c_j} \quad \forall k = 0, \ldots, m_i. \tag{4}$$

Using the formulas provided in (Schneider and Werner 1986), these matrices can be computed efficiently *a priori*.

The Lagrange term is approximated via the corresponding Radau quadrature rule

$$\int_{t_0}^{t_f} L(\boldsymbol{x}(t),\boldsymbol{u}(t),\boldsymbol{p},t)\,\mathrm{d}t \approx \sum_{i=0}^{n} \Delta t_i \sum_{j=1}^{m_i} b_j L_{ij}, \tag{5}$$

where $L_{ij} := L(\boldsymbol{x}_{ij}, \boldsymbol{u}_{ij}, \boldsymbol{p}, t_{ij})$ and the quadrature weights are given by

$$b_j = \int_0^1 \prod_{\substack{k=1 \\ k \neq j}}^{m_i} \frac{\tau - c_k}{c_j - c_k}\,\mathrm{d}\tau \quad \forall j = 1, \ldots, m_i. \tag{6}$$

2.3 Resulting NLP

Discretizing the remaining components of the GDOP is straightforward. The path constraints are replaced by constraints, which are evaluated at all grid points t_{ij} and the Mayer term and final constraints are approximated by an evaluation at the final grid point t_{nm_n}. The time-invariant parameters and their parametric constraints need not be

DOI 10.3384/ecp218127

discretized, since these are discrete. By flattening the collocated dynamics (3), the discretized GDOP becomes

$$\min_{\boldsymbol{x}_{ij},\boldsymbol{u}_{ij},\boldsymbol{p}} M(\boldsymbol{x}_{nm_n},\boldsymbol{u}_{nm_n},\boldsymbol{p},t_{nm_n}) + \sum_{i=0}^{n}\Delta t_i \sum_{j=1}^{m_i} b_j L(\boldsymbol{x}_{ij},\boldsymbol{u}_{ij},\boldsymbol{p},t_{ij}) \tag{7a}$$

s.t.

$$\boldsymbol{0} = \sum_{k=0}^{m_i} D^{(1)}_{jk}\boldsymbol{x}_{ik} - \Delta t_i \boldsymbol{f}(\boldsymbol{x}_{ij},\boldsymbol{u}_{ij},\boldsymbol{p},t_{ij}) \tag{7b}$$

$$\boldsymbol{x}_{00} = \boldsymbol{x}_0 \tag{7c}$$

$$\boldsymbol{g}^L \le \boldsymbol{g}(\boldsymbol{x}_{ij},\boldsymbol{u}_{ij},\boldsymbol{p},t_{ij}) \le \boldsymbol{g}^U \tag{7d}$$

$$\boldsymbol{r}^L \le \boldsymbol{r}(\boldsymbol{x}_{nm_n},\boldsymbol{u}_{nm_n},\boldsymbol{p},t_{nm_n}) \le \boldsymbol{r}^U \tag{7e}$$

$$\boldsymbol{a}^L \le \boldsymbol{a}(\boldsymbol{p}) \le \boldsymbol{a}^U \tag{7f}$$

where the dynamic constraints (7b) and path constraints (7d) must be satisfied for all $i = 0,\dots,n$ and $j = 1,\dots,m_i$.

This sparse, large-scale NLP can be implemented and solved efficiently with state-of-the-art nonlinear optimizers such as Ipopt (Wächter and Biegler 2006) or SNOPT (P. E. Gill et al. 2007; Philip E. Gill, Murray, and Saunders 2005), which require first and second order derivatives of the objective and constraints. By properly sorting the constraint vector, the cyclic block structure in the Jacobian and Hessian can be exploited. A detailed overview of the resulting blocks is given in (Langenkamp 2024).

3 Adaptive Mesh Refinement

An important modeling decision is the choice of grid points, i.e. the mesh $\mathcal{M} = \{t_0, t_1, \dots, t_n\}$, and the number of collocation nodes m_i per interval. Without prior knowledge of the problem, one might use an equidistant mesh $\Delta t_i \equiv$ const and a fixed number of nodes $m_i \equiv$ const. However, this often leads to large errors and unrealistic results, especially when the optimal solution exhibits discontinuities, switches, or steep gradients. Furthermore, the optimization often takes a significant amount of computation time due to poor initial guesses.

To address this, direct collocation methods typically employ adaptive mesh refinement algorithms, which balance accuracy and efficiency. In adaptive refinement, an initial NLP is solved on a coarse mesh, then the mesh is refined, and the NLP is solved again using updated initial guesses. This process continues until a termination criterion is met. The main classes of mesh refinement methods are h-, p-, and hp-methods.

h-methods (J. Zhao and Shang 2018; Y. Zhao and Tsiotras 2009; Jain and Tsiotras 2008; Betts and Huffman 1998) divide the time horizon into many intervals and use low-degree polynomials on each, i.e. $m_i \equiv m$. Convergence is achieved by increasing the number of intervals based on error criteria. These methods are robust and effective for non-smooth solutions but converge only polynomially fast for smooth problems. Several h-methods have been described in the literature. For example, in (J. Zhao and Shang 2018) a *multiresolution technique* is proposed that compares the current solution with a smoothed trajectory using non-oscillatory (ENO) interpolation. It detects non-smooth regions and merges intervals with constant control. Another method, presented in (Y. Zhao and Tsiotras 2009), encodes the curvature of the control trajectory into a density function that governs the distribution of mesh points, effectively capturing irregularities.

p-methods usually use a single interval and high-degree global polynomials. For smooth problems, they achieve *spectral*, i.e. exponential, convergence, but for non-smooth cases, errors grow significantly.

Modern hp- or ph-adaptive methods (M. A. Patterson and Rao 2014; M. A. Patterson, Hager, and Rao 2015; Liu, Hager, and Rao 2015; Sagliano et al. 2018; Garg, M. Patterson, Francolin, et al. 2011) combine both strategies. They adapt both the number of intervals and polynomial degrees. These hybrid methods also provide spectral convergence and are implemented in state-of-the-art tools such as GPOPS II (M. A. Patterson and Rao 2014). However, they may still struggle to detect switches and often generate large meshes with many collocation points (J. Zhao and Shang 2018; M. A. Patterson, Hager, and Rao 2015).

3.1 L2-Boundary-Norm

In this paper, the novel h-method *L2-Boundary-Norm (L2BN)* is proposed, which uniformizes the control trajectory in a similar way to density function-based approaches (Y. Zhao and Tsiotras 2009). L2BN applies consecutive bisection mesh refinement, following a workflow similar to the *multiresolution* technique (J. Zhao and Shang 2018), but without removing mesh points to maintain maximum stability. The method is designed to detect discontinuities, kinks, bends, and steep sections, while requiring limited computation time. L2BN is divided into 2 distinct phases:

3.2 Phase I

Phase I is used to accelerate the optimization by performing a simple multigrid refinement. Here the possibly poor initial guess of the control trajectory $\boldsymbol{u}_{init}(t)$, e.g. constant or linear, is only used for an initial optimization on an extremely coarse mesh. To ensure that the state trajectory guess $\boldsymbol{x}_{init}(t)$ is feasible, i.e. satisfies the initial value problem (1b) - (1c), the initial states are obtained by performing a simulation using the provided guesses $\boldsymbol{u}_{init}(t)$ and $\boldsymbol{p}_{init}$.

The initial optimization is exceptionally fast compared to one using the poor guess on a fine mesh. The optimal solution of the problem is then interpolated onto a mesh, where every interval has been bisected. This interpolated solution is then used as a new, more appropriate initial guess and optimized again. Repeating this process several times, yields a realistic, equidistant optimal solution

that solely suffers from steep sections, kinks or jump discontinuities. Overall, the first phase can be expressed as Algorithm 3.1:

Algorithm 3.1: Phase I: Multigrid Refinement

Input: GDOP, equidistant mesh $\mathcal{M}_0$, number of iterations k_{max}, initial guess on the control trajectory $\boldsymbol{u}_{init}(t)$ and parameters $\boldsymbol{p}_{init}$
Output: Equidistant solutions $\boldsymbol{x}^*(t)$, $\boldsymbol{u}^*(t)$, $\boldsymbol{p}^*$

$\boldsymbol{x}_{init}(t) \leftarrow$ Solve the ODE $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\cdot), \boldsymbol{x}(t_0) = \boldsymbol{x}_0$ using the guesses $\boldsymbol{u}_{init}(t), \boldsymbol{p}_{init}$
for $k = 0, \ldots, k_{max}$ **do**
  $\boldsymbol{x}_k^*(t), \boldsymbol{u}_k^*(t), \boldsymbol{p}_k^* \leftarrow$ Solve the NLP (7a) - (7f) on $\mathcal{M}_k$ using $\boldsymbol{x}_{init}(t), \boldsymbol{u}_{init}(t), \boldsymbol{p}_{init}$ as initial guesses
  if $k = k_{max}$ **then**
    return $\boldsymbol{x}_k^*(t), \boldsymbol{u}_k^*(t), \boldsymbol{p}_k^*$
  end
  $\mathcal{U}_k \leftarrow \left\{ \frac{t_i + t_{i+1}}{2} \mid i = 0, \ldots, |\mathcal{M}_k| - 1 \right\}$
  $\mathcal{M}_{k+1} \leftarrow \mathcal{M}_k \cup \mathcal{U}_k$
  $\boldsymbol{p}_{init} \leftarrow \boldsymbol{p}_k^*$
  $\boldsymbol{x}_{init}(t), \boldsymbol{u}_{init}(t) \leftarrow$ Interpolate $\boldsymbol{x}_k^*(t), \boldsymbol{u}_k^*(t)$ onto $\mathcal{M}_{k+1}$
end

3.2.1 Illustration of Phase I

To further illustrate phase I, we consider the *Oil Shale Pyrolysis*[1], studied in (Wen and Yen 1977) and (Langenkamp 2024), aiming to find an optimal temperature control. Using a poor, constant initial temperature guess two approaches are compared. The *Default* method involves optimization on a mesh with $|\mathcal{M}_0| = 80$ intervals and $m = 3$ collocation nodes. Alternatively, Algorithm 3.1 (Phase I) starts with a much coarser initial mesh of 5 intervals and performs $k_{max} = 4$ refinement iterations. Both methods ultimately yield the same optimal solution.

Method	$\lvert\mathcal{M}_0\rvert$	m	k_{max}	t_{opt}	Iter
Default	80	3	0	0.3492	161
Algorithm 3.1	5	3	4	0.1242	105

Table 1. Performance Evaluation of Phase I Refinement for an Oil Shale Pyrolysis Optimization

As detailed in Table 1, the optimization using Ipopt and MUMPS demonstrates that Algorithm 3.1 required fewer NLP iterations (Iter) and approximately one-third of the computation time t_{opt} in seconds compared to the default implementation. This significant speedup is attributed to the multigrid refinement strategy: instead of optimizing with a poor initial guess on a large mesh, the guesses progressively improve through successive refinements. Figure 1 depicts this process, showing how the temperature control trajectory evolves and approaches the final solution across refinement iterations.

[1]The GDOPT implementation can be found under `https://github.com/linuslangenkamp/GDOPT/blob/master/examples/oilShalePyrolysis.py`.

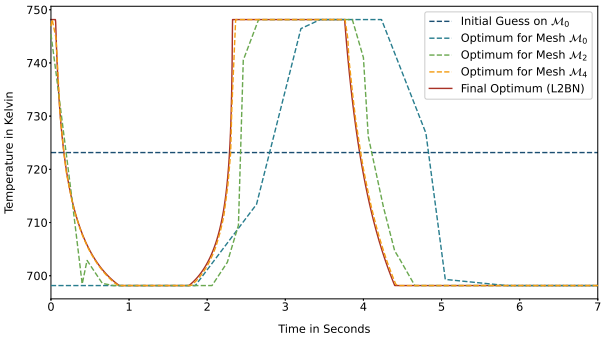

Figure 1. Multigrid refinement applied to an Oil Shale Pyrolysis (Wen and Yen 1977). The optimal solution after performing 3 additional phase II iterations is given as reference.

3.3 Phase II

Phase II addresses the problem of remaining steep sections, kinks or switches and iteratively refines the solution of phase I by investigating certain L^2-norms of the control trajectory (*on-interval condition*) as well as error terms of the trajectories on the boundary (*boundary condition*). It identifies non-smooth behavior and splits intervals, which violate the conditions, such that resolution is increased appropriately and switching times are identified correctly.

3.4 On-Interval Condition

The idea behind the *on-interval condition* is to individually inspect the current solution of the control variables $\boldsymbol{u}_{ij}$ within each mesh interval $[t_i, t_{i+1}]$ of the mesh $\mathcal{M}_k$, and to measure the *slope* and *curvature* of the control variables over the full interval. If these measures exceed a certain threshold, the interval is bisected, and the midpoint $\frac{t_i + t_{i+1}}{2}$ is added to the next mesh $\mathcal{M}_{k+1}$.

Consider an arbitrary interval $[t_i, t_{i+1}]$ from the mesh $\mathcal{M}_k$ and the current optimal solution of the l-th control variable $u^{(l)}$ on this interval. This solution is defined by $m+1$ sample points $u_{ij}^{(l)}$, $j = 0, \ldots, m$. The first step is to construct an interpolating polynomial $p_i^{(l)}(t)$ for the control variable $u^{(l)}$ on the normalized interval $[0, 1]$, such that $p_i^{(l)}(c_j) = u_{ij}^{(l)}$, $j = 0, \ldots, m$, with $p_i^{(l)} \in P^m$. This rescaling is performed because the condition should not be influenced by the length of the interval Δt_i and, moreover, only the magnitude of the controls should be relevant. In order to estimate the change on the interval, it is natural to consider a function norm of $\dot{p}_i^{(l)}$ and $\ddot{p}_i^{(l)}$ such as L^1, L^2 or L^∞. The L^2 norm

$$\|p\| := \sqrt{\int_0^1 p(\tau)^2 \, \mathrm{d}\tau} \tag{8}$$

is chosen, since it is a good middle ground and allows for

DOI 10.3384/ecp218127

a fast computation of the condition. All in all, the *on-interval condition* says:

Condition 3.1 (On-Interval Condition)**.** *If for a given interval* $[t_i, t_{i+1}]$ *there exists at least one control variable* $u^{(l)}$ *such that the interpolating polynomial* $p_i^{(l)}$ *on the nominal interval* $[0,1]$ *satisfies*

$$\left\|\dot{p}_i^{(l)}\right\| > TOL_1^{(l)} \quad \vee \quad \left\|\ddot{p}_i^{(l)}\right\| > TOL_2^{(l)},$$

for specified tolerances $TOL_1^{(l)}, TOL_2^{(l)} > 0$*, then the interval is bisected.*

3.4.1 Fast Computation

The following procedure enables an efficient and exact evaluation of the *on-interval condition* for any number of collocation nodes m. By construction, it holds that $\deg\left(\dot{p}_i^{(l)}(\tau)\right) = m-1$, $\deg\left(\ddot{p}_i^{(l)}(\tau)\right) = m-2$, which implies $\deg\left(\dot{p}_i^{(l)}(\tau)^2\right) = 2m-2$, $\deg\left(\ddot{p}_i^{(l)}(\tau)^2\right) = 2m-4$ for all $m \geq 2$. In the special case of $m = 1$, the second derivative of $p_i^{(l)}$ vanishes and the first derivative is constant. Since Radau quadrature with m collocation nodes integrates all polynomials of degree up to $2m-2$ exactly, both norms can be exactly evaluated using the quadrature rule. For simplicity, the calculations are only shown for the first derivative. These are analogous for the second derivative. Hence

$$\left\|\dot{p}_i^{(l)}\right\| = \sqrt{\int_0^1 \dot{p}_i^{(l)}(\tau)^2 \,\mathrm{d}\tau} = \sqrt{\sum_{k=1}^{m} b_k \dot{p}_i^{(l)}(c_k)^2}, \qquad (9)$$

where b_k are the Radau quadrature weights (6). By writing $p_i^{(l)}(\tau) = \sum_{j=0}^{m} u_{ij}^{(l)} l_j(\tau)$ as a Lagrange interpolating polynomial, we get $\dot{p}_i^{(l)}(c_k) = \sum_{j=0}^{m} u_{ij}^{(l)} D_{kj}^{(1)}$ using the first differentiation matrix. Finally, it follows that

$$\left\|\dot{p}_i^{(l)}\right\| = \sqrt{\sum_{k=1}^{m} b_k \left(\sum_{j=0}^{m} u_{ij}^{(l)} D_{kj}^{(1)}\right)^2}. \qquad (10)$$

Since applying the first differentiation matrix twice yields the second derivatives at the collocation nodes, a similar formula can be calculated for $\left\|\ddot{p}_i^{(l)}\right\|$. Therefore, we obtain Algorithm 3.2 by utilizing matrix-vector notation.

Because the algorithm has to be applied for each mesh interval and control variable, the overall runtime of the condition becomes $O(d_{\boldsymbol{u}} n m^2)$, which is just m times the number of discretized controls $d_{\boldsymbol{u}} n m$. This runtime is neglectable, compared to the several expensive NLP iterations that are performed for each mesh iteration.

3.4.2 Termination

An important idea of the Phase II iteration is that the final solution should satisfy Condition 3.1 for all intervals and control variables. This condition ensures that the local variability (slope and curvature) of the control trajectories remains below specific tolerances across all intervals. When this is achieved, the mesh points become equidistributed relative to this local variability. Since Condition 3.1 takes the same form as conditions used to define mesh density functions (e.g. ensuring an integral of some monitor function over each interval is below a tolerance), the resulting mesh distribution is at least as strong as, and can be characterized by, the density function implicitly defined by the satisfied condition itself. The derivation of this implicit density function is detailed in (Langenkamp 2024). Because of this, it is useful to have a criterion for the termination of a sequence of *on-interval* iterations. A sufficient condition for this termination can be deduced from Lemma 1, where the L^2 norm is again taken on $[0,1]$:

Algorithm 3.2: Fast *On-Interval* Computation

Input: Sample values $\hat{\boldsymbol{u}} = \left(u_{i0}^{(l)}, u_{i1}^{(l)}, \ldots, u_{im}^{(l)}\right)^T$, m-step Radau quadrature weights $\boldsymbol{b}$, differentiation matrix $D^{(1)}$

Output: $\left\|\dot{p}_i^{(l)}\right\|, \left\|\ddot{p}_i^{(l)}\right\|$

$\boldsymbol{p}' \leftarrow D^{(1)} \hat{\boldsymbol{u}}$

$\boldsymbol{q}' \leftarrow \left((p_1')^2, \ldots, (p_m')^2\right)^T$

$\left\|\dot{p}_i^{(l)}\right\| \leftarrow \sqrt{\boldsymbol{b}^T \boldsymbol{q}'}$

$\boldsymbol{p}'' \leftarrow D^{(1)} \boldsymbol{p}'$

$\boldsymbol{q}'' \leftarrow \left((p_1'')^2, \ldots, (p_m'')^2\right)^T$

$\left\|\ddot{p}_i^{(l)}\right\| \leftarrow \sqrt{\boldsymbol{b}^T \boldsymbol{q}''}$

return $\left\|\dot{p}_i^{(l)}\right\|, \left\|\ddot{p}_i^{(l)}\right\|$

Lemma 1. *Let* $p \in P^m$ *be a polynomial and* $p_\varepsilon(t) = p\left(\frac{t}{2}\right) + \varepsilon(t)$ *be a variation of* $p\left(\frac{t}{2}\right)$ *by* $\varepsilon \in P^m$*, then for a given* $\delta \in \mathbb{R}$ *with* $\frac{\sqrt{2}}{2} < \delta < 1$

$$\frac{\|\dot{p}_\varepsilon(t)\|}{\|\dot{p}(t)\|} < \delta, \quad \text{if} \quad \|\dot{\varepsilon}(t)\| < \frac{\sqrt{2}\delta - 1}{2} \left\|\dot{p}\left(\frac{t}{2}\right)\right\|.$$

A proof of Lemma 1 is given in the Appendix. Let $p(t)$ be a polynomial that violates Condition 3.1. The corresponding interval is bisected and both subintervals are examined in the next iteration. Without loss of generality, we consider the first subinterval, while the second follows analogously with $p_\varepsilon(t) = p\left(\frac{t+1}{2}\right) + \varepsilon(t)$. Rescaling the old polynomial from $[0, \frac{1}{2}]$ to $[0,1]$, as required by the *on-interval condition*, yields $p\left(\frac{t}{2}\right)$, so the new polynomial becomes $p_\varepsilon(t) = p\left(\frac{t}{2}\right) + \varepsilon(t)$, with a perturbation ε. By Lemma 1, the new norm on the subinterval is smaller if

$$\left\|\dot{p}_\varepsilon(t) - \dot{p}\left(\frac{t}{2}\right)\right\| = \|\dot{\varepsilon}(t)\| < \frac{\sqrt{2}\delta - 1}{2} \left\|\dot{p}\left(\frac{t}{2}\right)\right\|. \qquad (11)$$

Thus, we get a decreasing sequence of L^2-norms, provided the difference of the new and old polynomial derivatives is suitably bounded proportional to the derivative of the old polynomial itself. Assuming the convergence of the collocation method, the variations $\varepsilon, \dot{\varepsilon}$, and $\ddot{\varepsilon}$ should vanish for smooth problems. Since it is easy to find an analogous lemma for the second derivative with

$$\frac{\|\ddot{p}_\varepsilon(t)\|}{\|\ddot{p}(t)\|} < \delta, \text{ if } \|\ddot{\varepsilon}(t)\| < \frac{2\sqrt{2}\delta - 1}{4}\left\|\ddot{p}\left(\frac{t}{2}\right)\right\| \tag{12}$$

for $\frac{1}{2\sqrt{2}} < \delta < 1$, the same holds for $\ddot{p}$ in both subintervals. Hence, for smooth problems under suitable convergence assumptions, the refinement procedure terminates after finitely many iterations. However, for non-smooth solutions, e.g. bang-bang optimal controls, the method may never terminate, as the identical polynomial can be obtained on one of the subintervals, i.e. $p_\varepsilon(t) - p(t) \equiv 0$.

3.5 Boundary Condition

The described method could be implemented using only the *on-interval condition*. However, this approach has a critical weakness in detecting non-smooth behavior on the boundary of two adjacent intervals.

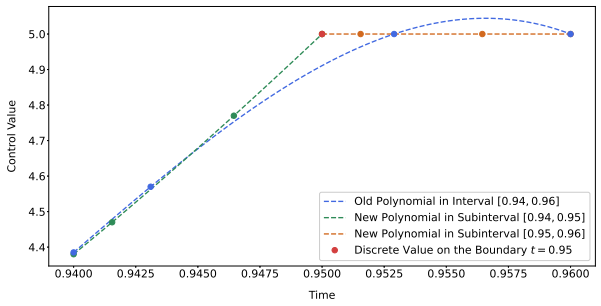

Figure 2. Poor corner detection

Consider Figure 2, where the interval $[0.94, 0.96]$ is split into $[0.94, 0.95]$ and $[0.95, 0.96]$ using the *on-interval condition*. The interpolating polynomials based on the 3-step Radau collocation method are shown. The first subinterval exhibits a steep slope throughout and will likely be bisected again. In contrast, the second subinterval appears nearly constant and will not trigger further refinement. However, this is problematic, as their shared boundary represents a sharp corner. To determine the switching time more accurately, the algorithm should further refine the mesh around this corner.

To resolve this, we introduce the *boundary condition*. It compares the first and second derivatives of adjacent polynomials at their shared boundary point. Rather than using purely relative or absolute error, we adopt the *plus-1 error* with $E(x,y) := \frac{|x-y|}{1+\min|x|,|y|}$, which combines both measures. Overall, the *boundary condition* says:

Condition 3.2 (Boundary Condition)**.** *If for two adjacent intervals i and $i+1$ there exists at least one control variable $u^{(l)}$, such that the interpolating polynomials $p_i^{(l)}, p_{i+1}^{(l)}$ on the nominal interval $[0,1]$ satisfy*

$$E(\dot{p}_i^{(l)}(t_{i+1}), \dot{p}_{i+1}^{(l)}(t_{i+1})) \geq CTOL_1^{(l)} \ \vee$$
$$E(\ddot{p}_i^{(l)}(t_{i+1}), \ddot{p}_{i+1}^{(l)}(t_{i+1})) \geq CTOL_2^{(l)}$$

for the common boundary point t_{i+1} and specified corner tolerances $CTOL_1^{(l)}, CTOL_2^{(l)} > 0$, then both intervals are bisected.

Evaluating the condition is inexpensive, since the values are side products of Algorithm 3.2.

3.6 Algorithm

Based on the previous considerations the full mesh refinement algorithm L2-Boundary-Norm (L2BN) can be formulated. By default, the constant corner tolerances are given as $CTOL_1^{(l)} = CTOL_2^{(l)} = 0.1$, but can be adjusted for a specific problem. The default tolerances for Condition 3.1 are given by

$$TOL_1^{(l)} = \frac{\text{range}(u^{(l)})}{|\mathcal{M}_0|} 10^{-\Lambda} \tag{13a}$$

$$TOL_2^{(l)} = \frac{\text{range}(u^{(l)})}{2|\mathcal{M}_0|} 10^{-\Lambda}, \tag{13b}$$

where $\text{range}(u^{(l)}) := \max_{t\in T} u^{(l)}(t) - \min_{t\in T} u^{(l)}(t)$ and $\Lambda \in \mathbb{R}$ is the level of refinement with default value $\Lambda = 0$.

Altogether we get Algorithm 3.3, which, for the b-th mesh refinement, requires $O(d_{\boldsymbol{u}} m^2 |\mathcal{M}_b|)$ operations. In this version of L2BN, Condition 3.1 enforces a bisection of the interval itself and both neighbors to prevent bisections in future iterations and enhance stability.

4 Implementation

The novel mesh refinement algorithm L2BN is integrated into a newly developed, open source dynamic optimization framework called GDOPT (General Dynamic Optimizer), which is publicly available on GitHub[2]. The framework is split into a Python modeling environment *gdopt*, which allows for expressive and accessible symbolic modeling of dynamic optimization problems and a C++ library *libgdopt* that performs the computationally intensive task of solving the large-scale NLPs with Ipopt (Wächter and Biegler 2006).

The framework offers a range of noteworthy features, of which the most relevant are presented in this paper. A comprehensive description of the GDOPT implementation can be found in (Langenkamp 2024) and an introduction to modeling with the framework is provided in the GDOPT User's Guide[3]. The principle workflows are illustrated in Figure 3. First, users can define their optimization problems in Python using purely symbolic ex-

[2] https://github.com/linuslangenkamp/GDOPT

[3] https://github.com/linuslangenkamp/GDOPT/blob/master/usersguide/usersguide.pdf

DOI 10.3384/ecp218127

Algorithm 3.3: L2-Boundary-Norm (L2BN)

Input: GDOP, phase I iteration count k_{max}, guesses $\boldsymbol{u}_{init}(t)$, $\boldsymbol{p}_{init}$, mesh $\mathcal{M}_0$, Radau IIA scheme with m nodes, phase II iteration count b_{max}, tolerances $TOL_1^{(l)}$, $TOL_2^{(l)}$, $CTOL_1^{(l)}$, $CTOL_2^{(l)}$

Output: $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{p}^*$

$\boldsymbol{x}_0^*(t), \boldsymbol{u}_0^*(t), \boldsymbol{p}_0^* \leftarrow$ Perform k_{max} iterations of Algorithm 3.1 on $\mathcal{M}_0$ with m collocation nodes and guesses $\boldsymbol{u}_{init}(t)$, $\boldsymbol{p}_{init}$

for $b = 0, \ldots, b_{max} - 1$ **do**
  $\mathcal{U} \leftarrow \{\}$
  for $i = 0, \ldots, |\mathcal{M}_b| - 1$ **do**
    for $l = 1, \ldots, d_{\boldsymbol{u}}$ **do**
      if *Condition 3.1 is violated* **then**
        $\mathcal{U} \leftarrow \mathcal{U} \cup \{\max\{0, i-1\}, i, \min\{|\mathcal{M}_b| - 1, i+1\}\}$
        break
      end
      if $(i \neq |\mathcal{M}_b| - 1) \wedge$ *Condition 3.2 is violated* **then**
        $\mathcal{U} \leftarrow \mathcal{U} \cup \{i, i+1\}$
      end
    end
  end
  if $\mathcal{U} = \{\}$ **then**
    return $\boldsymbol{x}_b^*(t), \boldsymbol{u}_b^*(t), \boldsymbol{p}_b^*$
  end
  $\mathcal{U}_b \leftarrow$ Bisect the intervals in $\mathcal{U}$
  $\mathcal{M}_{b+1} \leftarrow \mathcal{M}_b \cup \mathcal{U}_b$
  $\boldsymbol{p}_{init} \leftarrow \boldsymbol{p}_b^*$
  $\boldsymbol{x}_{init}(t), \boldsymbol{u}_{init}(t) \leftarrow$ Interpolate $\boldsymbol{x}_b^*(t), \boldsymbol{u}_b^*(t)$ onto $\mathcal{M}_{b+1}$
  $\boldsymbol{x}_{b+1}^*(t), \boldsymbol{u}_{b+1}^*(t), \boldsymbol{p}_{b+1}^* \leftarrow$ Solve the NLP (7a) - (7f) on $\mathcal{M}_{b+1}$ using $\boldsymbol{x}_{init}(t)$, $\boldsymbol{u}_{init}(t)$, $\boldsymbol{p}_{init}$ as initial guesses
end
return $\boldsymbol{x}_{b_{max}}^*(t), \boldsymbol{u}_{b_{max}}^*(t), \boldsymbol{p}_{b_{max}}^*$

pressions for differential equations, constraints and objectives. The Python package *SymEngine*[4] is used to efficiently calculate the symbolic first and second derivatives and their common subexpressions to remove redundant calculations. These symbolic expressions are translated into efficient C++ code with exact sparsity patterns. After that, the code is compiled and linked with *libgdopt*, which implements the full collocation-based dynamic optimization pipeline, including discretization, solver interface, L2BN mesh refinement algorithm, and precomputed Radau IIA schemes for $m = 1, \ldots, 70$ accurate to machine precision.

GDOPT separates code generation and optimization, allowing parameters, solver settings, and initial guesses to be modified after compilation. This enables rapid repeated solves without regenerating and recompiling the model. At runtime, the executable reads all parameters, initial values, and solver settings, and solves the resulting NLP using Ipopt (Wächter and Biegler 2006) with a chosen linear solver, e.g. MUMPS (Amestoy et al. 2001) or one from the HSL suite (HSL 2013). Results are written to an output file, which is automatically read by the Python interface for post-processing and visualization using *matplotlib* (Hunter 2007).

4.1 Acceleration of Refinement Iterations

An important note on the implementation is, that Ipopt is an interior-point solver that uses logarithmic barriers. The implementation of L2BN heavily benefits from this concept, by setting the barrier parameter to a minimal value (e.g. $\mu_0 = 10^{-15}$) for repeated solves. This strategy significantly accelerates mesh refinement iterations even further, since the new initial guess, i.e. the previous optimal solution interpolated onto the new mesh, is known to be almost optimal by construction. Thus, Ipopt solves an unperturbed problem, avoiding unnecessary calculations and achieving rapid convergence.

[4] https://github.com/symengine/symengine

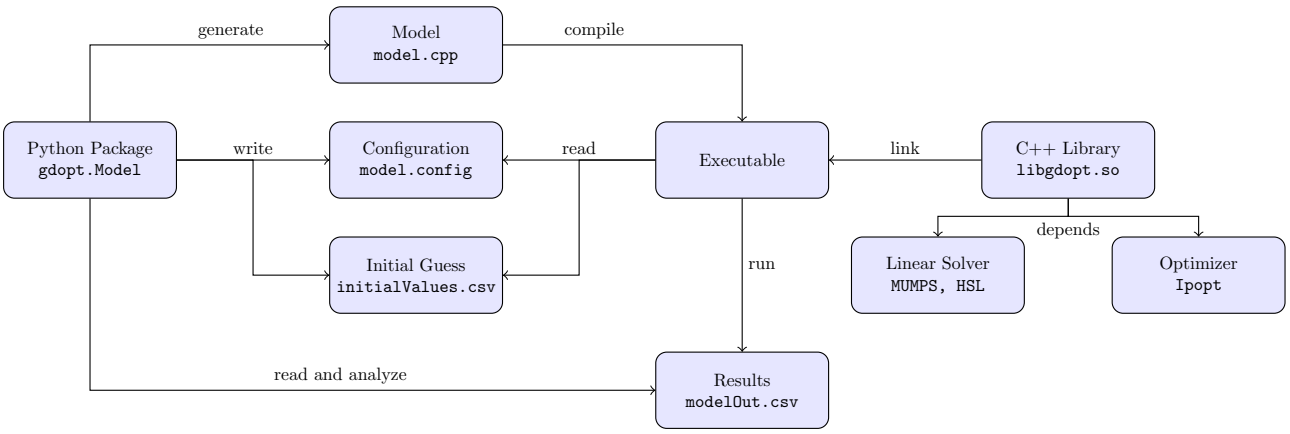

Figure 3. Principle workflows in GDOPT. For a detailed introduction to workflows in GDOPT, please refer to (Langenkamp 2024).

4.2 Optimization of DAEs

Currently, the framework does not support the direct handling of differential-algebraic equations (DAEs). While algebraic variables can be modeled as control inputs, this approach artificially bloats the NLP, making it more numerically unstable and computationally demanding. An integration of *libgdopt* into a proper modeling environment would clearly be beneficial. Previous work on OpenModelica (Ruge et al. 2014) has already demonstrated such an integration by performing a transformation to semi-explicit ODE form and solving for algebraic variables in each optimization step.

5 Results and Performance

To evaluate the performance of the proposed mesh refinement algorithm and the newly developed framework, a relevant dynamic optimization problem from the open literature is considered. For additional benchmarks of GDOPT, including demonstrations of the termination property for smooth problems, we refer to (Langenkamp 2024). All optimizations are performed on a laptop with a Intel Core i7-12800H, 32 GB RAM, running Ubuntu 24.04.2 and using GCC v13.3.0 with flags -O3 -ffast-math for compilation.

5.1 Diesel Electric Power Train

Consider the diesel-electric power train[5] that has been studied extensively in (Sivertsson and Eriksson 2012; Mengist, Ruge, Gebremedhin, et al. 2013; Bachmann et al. 2012). This optimization problem deals with the fuel optimal start-up of a diesel engine from an idling condition to a certain power level. The model features 2 control variables in relative units and 4 states, while each variable is associated with a given box constraint to guarantee physical validity. The diesel-electric power train has the following principle structure:

$$\min_{u_1(t),u_2(t)} \sum_{i=1}^{4} (x_i(t_f) - c_i)^2 \tag{14a}$$

s.t.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, x_3, u_1) \\ f_2(x_1, x_2, x_4) \\ f_3(x_1, x_2, x_3, u_1, u_2) \\ f_4(x_1, x_2, x_3, x_4, u_1) \end{bmatrix} \forall t \in [t_0, t_f] \tag{14b}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{14c}$$

$$\boldsymbol{x}^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U \forall t \in [t_0, t_f] \tag{14d}$$

$$\mathbf{0} \leq \boldsymbol{u} \leq \mathbf{1} \quad \forall t \in [t_0, t_f] \tag{14e}$$

$$t_0 = 0, t_f = 0.5, \boldsymbol{c} \in \mathbb{R}^4 \text{ are constants.} \tag{14f}$$

The optimization problem is solved with both GDOPT and OpenModelica v1.24.0 using the same hardware and settings for various configurations. All runs use an Ipopt tolerance of 10^{-14}, the linear solver MUMPS, and identical constant initial guesses. All other solver parameters are left at default and the times are averaged over 5 runs.

[5]The GDOPT implementation can be found under `https://github.com/linuslangenkamp/GDOPT/blob/master/examples/dieselMotor.py`

5.2 Code Generation and Compilation

The code generation and compilation timings for GDOPT are given by: 0.0268 s for the derivative calculations and code generation, 1.1561 s for compiling the generated C++ code and 0.0335 s for obtaining the initial guess by simulating the dynamics. However, the OpenModelica compiler requires just 0.2011 seconds for the translation and compilation. Note that this overhead is independent of the specific configuration.

5.3 Results Without Mesh Refinement

For equidistant meshes with sizes $|\mathcal{M}_0| = 25, 100, 250$, the same optimization has been performed with both implementations and the results are displayed in Table 2.

Alg.	$\lvert\mathcal{M}_0\rvert$	m	$\phi^* \cdot 10^3$	t_{opt}	Speedup
OM	25	3	1.11171326	0.3592	1.0000
GD	25	3	1.11171326	0.1877	1.9136
OM	100	3	1.11155875	0.6730	1.0000
GD	100	3	1.11155875	0.3214	2.0940
OM	250	3	1.11155972	1.8256	1.0000
GD	250	3	1.11155972	0.6086	2.9997

Table 2. Performances of GDOPT (GD) and OpenModelica (OM) on equidistant meshes for the *Diesel Electric Power Train*. The columns are the algorithm (Alg.), initial mesh size ($|\mathcal{M}_0|$), number of collocation nodes (m), scaled final objective ($\phi^* \cdot 10^3$), total optimization time (t_{opt}), and speedup factor relative to OpenModelica.

It is noteworthy that GDOPT is 2 to 3 times faster than OpenModelica without using any mesh refinement. Moreover, both implementations provide the exact same objectives to all digits. This clearly shows that GDOPT efficiently deploys the relevant model information and additionally, performs internal calculations very fast.

5.4 Phase I Refinement

By incorporating $k = 2$ phase I refinements, the same optimal solution for an equidistant mesh size of $|\mathcal{M}| = 100$ is obtained in 0.2291 s as seen in Table 3, compared to 0.3214 s with GDOPT without refinement and 0.6730 s with OpenModelica (Table 2). Therefore, the multigrid refinement (Algorithm 3.1) accelerates the GDOPT procedure by roughly 40%.

5.5 Results Using L2-Boundary-Norm

Additional Phase II iterations enhance accuracy in regions with non-smooth control behavior. Since the optimal control remains constant over large parts of the time horizon, high resolution is not required everywhere. However, the problem also features sharp switching behavior, where accurately resolving the exact timing of the switches is crit-

DOI 10.3384/ecp218127

ical. This is illustrated in Figure 4, which shows the optimal control solution on the final refined mesh:

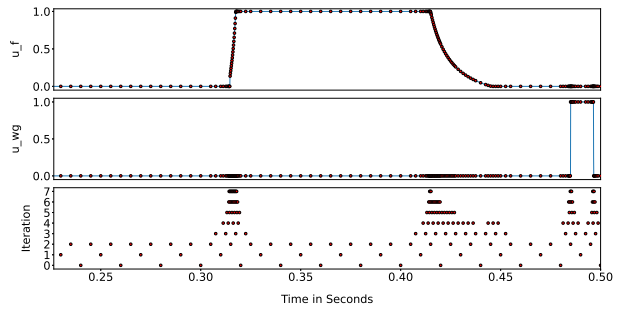

Figure 4. Mesh refinement history of the *Diesel Electric Power Train* for configuration: GDOPT, $k = 2$, $b = 5$, $|\mathcal{M}_0| = 25$, $m = 3$

To further investigate the impact of the refinement iterations, control trajectories in different stages of the refinement process are presented. In Figure 5 the controls are displayed for the configuration $k = 2$, $b = 5$, $|\mathcal{M}_0| = 25$, $m = 3$.

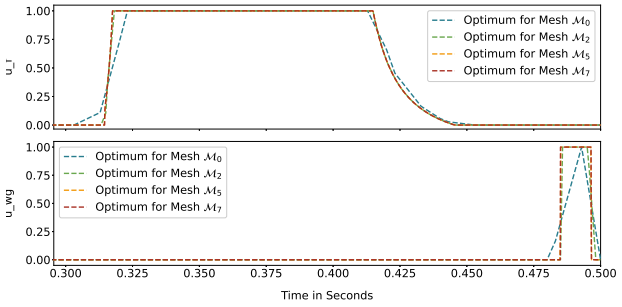

Figure 5. Refinement iterations for both control variables of the *Diesel Electric Power Train* for configuration: GDOPT, $k = 2$, $b = 5$, $|\mathcal{M}_0| = 25$, $m = 3$

Note that in this case the mesh subscript denotes the cumulative mesh iteration. By simply performing 2 multigrid iterations, the solution on $\mathcal{M}_2$ looks almost identical to the final solution using 5 additional phase II iterations. Nevertheless, by increasing the size of the plot, it becomes obvious that the switching times are not precisely located as seen for the second control variable in Figure 6. For this control, the placements of the switches are roughly $t \approx 0.485\,\mathrm{s}$ and $t \approx 0.496\,\mathrm{s}$ for $\mathcal{M}_2$, but the resolution is too low to capture the exact position of the switch. By performing additional iterations, the resolution of the switches increases drastically, such that the final solution on $\mathcal{M}_7$, which was obtained in just 0.9522 s (Table 3), describes a perfect rectangular shape. This final resolution is equivalent to that of an equidistant mesh with 3200 intervals. Performing the optimization on this grid without refinement takes 20.3825 s using GDOPT and 84.6323 s using OpenModelica. Furthermore, OpenModelica did not find the optimal solution but terminated with an oscillating solution.

Alg.	k, b	$\lvert\mathcal{M}_0\rvert$	m	$\lvert\mathcal{M}^*\rvert$	$\phi^* \cdot 10^3$	t_{opt}
GD	2,0	25	3	100	1.11155875	0.2291
GD	2,3	25	3	167	1.11155842	0.4301
GD	2,3	25	5	170	1.11155852	0.6849
GD	2,3	25	7	171	1.11155856	1.1687
GD	2,5	25	3	240	1.11155853	0.9522
GD	2,5	25	5	243	1.11155853	1.5512
GD	2,5	25	7	249	1.11155853	4.4676
GD	2,8	25	3	266	1.11155853	1.3536

Table 3. Performance of GDOPT (GD) using Algorithm 3.3 for the *Diesel Electric Power Train*. The columns are the algorithm (Alg.), maximum number of Phase I and II iterations (k, b), initial mesh size ($|\mathcal{M}_0|$), number of collocation nodes (m), final mesh size ($|\mathcal{M}^*|$), scaled final objective ($\phi^* \cdot 10^3$), and total optimization time (t_{opt}) in seconds.

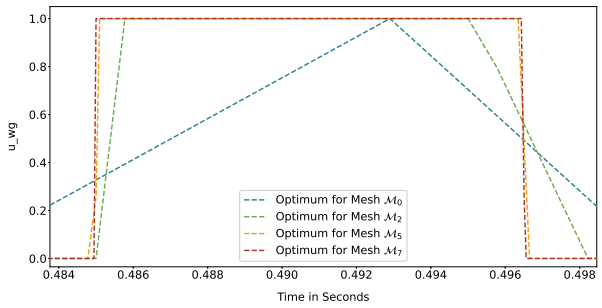

Figure 6. Refinement iterations for jump discontinuities of the *Diesel Electric Power Train* for configuration: GDOPT, $k = 2$, $b = 5$, $|\mathcal{M}_0| = 25$, $m = 3$

5.6 Resolution Under Runtime Constraints

To benchmark resolution efficiency, GDOPT was run on equidistant meshes under a fixed runtime limit of 1 s. The densest such mesh solved within this limit had 400 intervals and required 1.0273 s. In contrast, mesh refinement using L2BN with $k = 2$ Phase I and $b = 8$ Phase II iterations achieved an effective resolution equivalent to 25600 equidistant intervals in just 1.3536 s. For comparison, equidistant meshes with 550 and 600 intervals required 1.2349 s and 2.0783 s, respectively. These resolutions are over 40 times coarser than the refined result. This demonstrates that mesh refinement with L2BN significantly improves resolution, as needed for problems with switching controls, while maintaining low computational cost.

5.7 Discussion of Results

Overall, L2BN enables GDOPT to accelerate convergence from poor initial guesses, efficiently detect non-smooth regions, and adaptively increase resolution, far outperforming traditional equidistant mesh approaches. Combined with its rapid execution under equivalent conditions, GDOPT shows striking advantages over direct collocation in OpenModelica in both speed and solution quality.

6 Limitations

While effective, L2BN and GDOPT have several limitations that should be considered. First, L2BN does not take advantage of the spectral convergence properties of flipped Legendre-Gauss-Radau (fLGR) collocation points. Therefore, it is unable to achieve the convergence rates of hp-adaptive methods and may require excessive mesh iterations or intervals to reach high accuracy, potentially leading to numerical instabilities.

The performance of L2BN also depends on the initial mesh size and refinement parameters, such as the number of phase I and phase II iterations. Clearly, the initial mesh must resemble the principle behavior of the subsequent iterations, otherwise the phase I iterations can have negative effects. Poor choices for these settings can result in unnecessary computational effort or failure to resolve key features of the solution.

Another limitation is the absence of direct error estimation in the algorithm. Since L2BN does not explicitly compute or use error estimates, it may not always refine the mesh optimally. For example, regions with smooth but rapidly changing controls might be over-refined, because of the large change on the interval. This can lead to a lot of unnecessary computations and considerably larger meshes than sufficient.

The most significant limitation, however, is that GDOPT is unable to handle DAEs directly and must model algebraic variables as control inputs. An integration into an elaborate modeling environment like OpenModelica would allow solving for algebraic variables at every step, thus reducing the solver space. Furthermore, direct error estimates could be incorporated by leveraging established simulation capabilities of OpenModelica.

7 Discussion and Future Work

This work presents two significant contributions to direct collocation-based dynamic optimization: (1) a comprehensive re-implementation that addresses significant limitations in optimizing with OpenModelica, and (2) the novel L2BN mesh refinement algorithm. The results demonstrate that GDOPT achieves 2 to 3 times speedups over OpenModelica under equivalent conditions, while introducing additional, essential capabilities such as adaptive mesh refinement, parameter optimization, exact Hessians, and support for higher-order collocation schemes. Tested on a representative optimal control problem, the proposed mesh refinement algorithm proves to be highly efficient. It speeds up convergence from poor initial guesses, correctly detects non-smooth regions, and enhances solution accuracy with low computational effort. Additionally, it reaches a resolution at switching points that would be unrealistic to achieve with equidistant meshes in a reasonable amount of time.

Ongoing work focuses on integrating *libgdopt* into the OpenModelica toolchain to replace its legacy optimization runtime. This integration will bring several substantial improvements, including full parameter optimization capabilities, adaptive mesh refinement as presented in this work, variable polynomial degrees, and exact Hessian computation for improved convergence. Moreover, native DAE support, already available in OpenModelica, will be preserved through algebraic system resolution at each optimization step, expanding the capabilities of GDOPT, which is currently limited to ODEs. The performance gains demonstrated in this paper suggest that such an integration could significantly enhance the optimization capabilities of OpenModelica.

Acknowledgements

This work was conducted as part of the OpenSCALING project (Grant No. 01IS23062E) at the University of Applied Sciences and Arts Bielefeld, in collaboration with Linköping University. The authors would like to express their sincere appreciation to both the OpenSCALING project and the Open Source Modelica Consortium (OSMC) for their support, collaboration, and shared commitment to advancing open-source modeling and simulation technologies.

References

Amestoy, Patrick R. et al. (2001). “A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling”. In: *SIAM Journal on Matrix Analysis and Applications* 23.1, pp. 15–41.

Bachmann, Bernhard et al. (2012-09). “Parallel Multiple-Shooting and Collocation Optimization with OpenModelica”. In: DOI: 10.3384/ecp12076659.

Becerra, V. M. (2010). “Solving complex optimal control problems at no cost with PSOPT”. In: *2010 IEEE International Symposium on Computer-Aided Control System Design*, pp. 1391–1396. DOI: 10.1109/CACSD.2010.5612676.

Betts, John T. and William P. Huffman (1998). “Mesh refinement in direct transcription methods for optimal control”. In: *Optimal Control Applications and Methods* 19.1, pp. 1–21.

Fritzson, Peter, Adrian Pop, Karim Abdelhak, et al. (2020-10). “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development”. In: *Modeling, Identification and Control: A Norwegian Research Bulletin* 41, pp. 241–295. DOI: 10.4173/mic.2020.4.1.

Garg, Divya, Michael Patterson, Camila Francolin, et al. (2011-06). “Direct Trajectory Optimization and Costate Estimation of General Optimal Control Problems Using a Radau Pseudospectral Method”. In: *Computational Optimization and Applications* 49, pp. 335–358. DOI: 10.1007/s10589-009-9291-0.

Gill, P. E. et al. (2007). *SNOPT 7.7 User’s Manual*. Tech. rep. CCoM Technical Report 18-1. San Diego, CA: Center for Computational Mathematics, University of California, San Diego.

Gill, Philip E., Walter Murray, and Michael A. Saunders (2005-01). “SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization”. In: *SIAM Rev.* 47.1, pp. 99–131. ISSN: 0036-1445. DOI: 10.1137/S0036144504446096. URL: https://doi.org/10.1137/S0036144504446096.

HSL (2013). *HSL: A collection of Fortran codes for large-scale scientific computation*. Available at http://www.hsl.rl.ac.uk. Accessed: 15-04-2025.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Jain, Sachin and Panagiotis Tsiotras (2008-09). "Trajectory Optimization Using Multiresolution Techniques". In: *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM* 31. DOI: 10.2514/1.32220.

Langenkamp, Linus (2024-12). "Adaptively Refined Mesh for Collocation-Based Dynamic Optimization". MA thesis. DOI: 10.13140/RG.2.2.18499.72484.

Liu, Fengjin, William W. Hager, and Anil V. Rao (2015). "Adaptive mesh refinement method for optimal control using nonsmoothness detection and mesh size reduction". In: *Journal of the Franklin Institute* 352.10, pp. 4081–4106. ISSN: 0016-0032. DOI: https://doi.org/10.1016/j.jfranklin.2015.05.028. URL: https://www.sciencedirect.com/science/article/pii/S0016003215002045.

Magnusson, Fredrik and Johan Åkesson (2015). "Dynamic Optimization in JModelica.org". In: *Processes* 3.2, pp. 471–496. ISSN: 2227-9717. DOI: 10.3390/pr3020471. URL: https://www.mdpi.com/2227-9717/3/2/471.

Mengist, Alachew, Vitalij Ruge, Mahder Gebremedhin, et al. (2013-09). "Model-Based Dynamic Optimization with OpenModelica and CasADi". In.

Patterson, Michael A., William W. Hager, and Anil V. Rao (2015). "A ph mesh refinement method for optimal control". In: *Optimal Control Applications and Methods* 36.4, pp. 398–421. DOI: https://doi.org/10.1002/oca.2114. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/oca.2114. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/oca.2114.

Patterson, Michael A. and Anil V. Rao (2014-10). "GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming". In: *ACM Trans. Math. Softw.* 41.1. ISSN: 0098-3500. DOI: 10.1145/2558904. URL: https://doi.org/10.1145/2558904.

Ruge, Vitalij et al. (2014-03). "Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C". In: pp. 1017–1025. ISBN: 978-91-7519-380-9. DOI: 10.3384/ecp140961017.

Sagliano, Marco et al. (2018-01). "SPARTAN: A Novel Pseudospectral Algorithm for Entry, Descent, and Landing Analysis". In: pp. 669–688. ISBN: 978-3-319-65282-5. DOI: 10.1007/978-3-319-65283-2_36.

Schneider, Claus and Wilhelm Werner (1986-07). "Some new aspects of rational interpolation". In: *Math. Comput.* 47.175, pp. 285–299. ISSN: 0025-5718. DOI: 10.2307/2008095. URL: https://doi.org/10.2307/2008095.

Sivertsson, Martin and Lars Eriksson (2012). "Time and Fuel Optimal Power Response of a Diesel-Electric Powertrain". In: *IFAC Proceedings Volumes* 45.30. 3rd IFAC Workshop on Engine and Powertrain Control, Simulation and Modeling, pp. 262–269. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20121023-3-FR-4025.00036. URL: https://www.sciencedirect.com/science/article/pii/S1474667015351673.

Wächter, Andreas and Lorenz T. Biegler (2006-03). "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical Programming* 106.1, pp. 25–57. ISSN: 1436-4646. DOI: 10.1007/s10107-004-0559-y. URL: https://doi.org/10.1007/s10107-004-0559-y.

Wen, C.S. and T.F. Yen (1977). "Optimization of oil shale pyrolysis". In: *Chemical Engineering Science* 32.3, pp. 346–349. ISSN: 0009-2509. DOI: https://doi.org/10.1016/0009-2509(77)80221-2. URL: https://www.sciencedirect.com/science/article/pii/0009250977802212.

Zhao, Jisong and Teng Shang (2018). "Dynamic Optimization Using Local Collocation Methods and Improved Multiresolution Technique". In: *Applied Sciences* 8.9. ISSN: 2076-3417. DOI: 10.3390/app8091680. URL: https://www.mdpi.com/2076-3417/8/9/1680.

Zhao, Yiming and Panagiotis Tsiotras (2009-04). "Mesh Refinement Using Density Function for Solving Optimal Control Problems". In: *AIAA Infotech at Aerospace Conference and Exhibit and AIAA Unmanned...Unlimited Conference*. DOI: 10.2514/6.2009-2019.

Appendix: Proof of Lemma 1

Lemma 1. *Let $p \in P^m$ be a polynomial and $p_\varepsilon(t) = p\left(\frac{t}{2}\right) + \varepsilon(t)$ be a variation of $p\left(\frac{t}{2}\right)$ by $\varepsilon \in P^m$, then for a given $\delta \in \mathbb{R}$ with $\frac{\sqrt{2}}{2} < \delta < 1$*

$$\frac{\|\dot{p}_\varepsilon(t)\|}{\|\dot{p}(t)\|} < \delta, \quad \text{if} \quad \|\dot{\varepsilon}(t)\| < \frac{\sqrt{2}\delta - 1}{2}\left\|\dot{p}\left(\frac{t}{2}\right)\right\|.$$

Proof.

$$\begin{aligned} \|\dot{p}(t)\|^2 &= \int_0^1 \dot{p}(t)^2 \,\mathrm{d}t = \int_0^{\frac{1}{2}} \dot{p}(t)^2 \,\mathrm{d}t + \int_{\frac{1}{2}}^1 \dot{p}(t)^2 \,\mathrm{d}t = \frac{1}{2}\left(\int_0^1 \dot{p}\left(\frac{t}{2}\right)^2 \mathrm{d}t + \int_0^1 \dot{p}\left(\frac{t+1}{2}\right)^2 \mathrm{d}t\right) \\ \implies \|\dot{p}(t)\| &= \frac{1}{\sqrt{2}}\left(\int_0^1 \dot{p}\left(\frac{t}{2}\right)^2 \mathrm{d}t + \int_0^1 \dot{p}\left(\frac{t+1}{2}\right)^2 \mathrm{d}t\right)^{\frac{1}{2}} = \frac{1}{\sqrt{2}}\left(\left\|\dot{p}\left(\frac{t}{2}\right)\right\|^2 + \left\|\dot{p}\left(\frac{t+1}{2}\right)\right\|^2\right)^{\frac{1}{2}} \end{aligned} \tag{15}$$

Also $\dot{p}_\varepsilon(t) = \frac{1}{2}\dot{p}\left(\frac{t}{2}\right) + \dot{\varepsilon}(t)$ and by the *Minkowski inequality* $\|\dot{p}_\varepsilon(t)\| \leq \frac{1}{2}\left\|\dot{p}\left(\frac{t}{2}\right)\right\| + \|\dot{\varepsilon}(t)\|$, implying

$$\frac{\|\dot{p}_\varepsilon(t)\|}{\|\dot{p}(t)\|} \leq \frac{\frac{1}{2}\left\|\dot{p}\left(\frac{t}{2}\right)\right\| + \|\dot{\varepsilon}(t)\|}{\frac{1}{\sqrt{2}}\left(\left\|\dot{p}\left(\frac{t}{2}\right)\right\|^2 + \left\|\dot{p}\left(\frac{t+1}{2}\right)\right\|^2\right)^{\frac{1}{2}}} \leq \frac{1}{\sqrt{2}}\frac{\left\|\dot{p}\left(\frac{t}{2}\right)\right\| + 2\|\dot{\varepsilon}(t)\|}{\left\|\dot{p}\left(\frac{t}{2}\right)\right\|} = \frac{1}{\sqrt{2}}\left(1 + \frac{2\|\dot{\varepsilon}(t)\|}{\left\|\dot{p}\left(\frac{t}{2}\right)\right\|}\right). \tag{16}$$

By setting $\frac{1}{\sqrt{2}}\left(1 + \frac{2\|\dot{\varepsilon}(t)\|}{\|\dot{p}\left(\frac{t}{2}\right)\|}\right) < \delta$ for $\delta \in \mathbb{R}$ with $\frac{\sqrt{2}}{2} < \delta < 1$ and rearranging terms, the proposition follows. □

DOI 10.3384/ecp218127