EPiC Series in Computing

Volume 104, 2024, Pages 84–98

Proceedings of 3rd International Workshop on
Mathematical Modeling and Scientific Computing

Phoenix-OC: Applied Optimal Control using Advanced Structure Exploitation and Multi-Level Parallelism

Johannes Diepolder

Diepolder Dynamic Optimization GmbH, Germany,
Johannes.Diepolder@DynamicOptimization.de

Abstract

This paper introduces Phoenix-OC, a novel optimal control software for the solution of large-scale, multi-phase Optimal Control (OC) problems. Phoenix-OC employs segmented collocation methods for the state discretization and B-Splines for the control parameterization. Each of the parameterized controls is allowed to have a distinct degree and knot grid. Additionally, control derivatives of arbitrary order can be utilized in the model, as well as constraint and cost functions. User-defined functions can be modeled either through an automatic differentiation framework or via a generic C interface supporting the utilization of virtually arbitrary functions, external models, etc. Among other features, the software inherently supports table data interpolation, the computation of post-optimal sensitivities for parametric OC problems, bi-level optimization, homotopy formulations, parallel batch runs, and job dependencies. Furthermore, jobs can be executed locally or through a job scheduler on computer clusters.

Phoenix-OC operates on the Phoenix-CORE engine - a generic sparse evaluation framework for both the evaluation and derivative computation of vector-valued functions. Central to this computational engine is the notion of an Extended Sparsity Pattern (ESP). This novel type of sparsity pattern extends traditional binary-valued sparsity patterns to a new type of floating-point pattern, allowing for advanced structure exploitation. The exploitation of sparse structures based on the ESP, combined with the multi-level parallelism implemented in Phoenix-OC, yields high performance across a range of representative benchmarks from engineering applications.

1 Introduction

The success of direct optimal control methods in solving applied optimal control problems has led to the development of various general-purpose solvers and frameworks over the last decades (e.g., [3, 9, 1, 10]).

The key design principles of Phoenix-OC focus on the efficient solution of large-scale optimal control problems as well as the parallel batch processing of large problem sets. These principles are realized through a combination of advanced structure exploitation and parallelization strategies both within and across processes.

Phoenix-OC implements collocation methods in segmented form. These methods transcribe infinite-dimensional optimal control problems in continuous time to large-scale and sparse finite-dimensional parameter optimization problems [4, 7].

Much of the success of direct optimal control methods can be attributed, on the one side, to the advancements in computer hardware and the development of efficient numerical algorithms for large-scale and sparse parameter optimization problems (e.g., [8, 6, 11]). On the other side, efficient evaluation frameworks for direct optimal control methods are required, allowing for the solution of problems of realistic size arising from engineering applications.

Driving factors in this context include the efficient computation of derivative information (e.g., based on automatic differentiation), the exploitation of structural information, as well as parallelization strategies.

Phoenix-OC utilizes a custom automatic differentiation module that provides both first- and second-order derivatives, along with the associated structural information (sparsity patterns). A distinctive feature of the computational engine Phoenix-CORE, used by Phoenix-OC, is that it exploits the information contained in a novel extended form of sparsity pattern, which exhibits a richer information content compared to regular binary-valued patterns. Moreover, parallelization strategies are implemented to distribute the computational workload in batch runs and for individual problems.

This paper is organized as follows: First, several classes of optimal control problems that can be modeled and solved using Phoenix-OC are introduced in Section 2. Next, Section 3 discusses modeling aspects in Phoenix-OC related to these problem classes. Computational aspects regarding the evaluation framework Phoenix-CORE employed by Phoenix-OC are outlined in Section 4. Finally, Section 5 and Section 6 present numerical experiments as well as several case studies for advanced optimal control applications.

2 Optimal control problem formulations

Phoenix-OC allows for the solution of general multi-phase optimal control problems in parametric form.

Multi-phase optimal control problems can be viewed as single-phase problems connected by appropriate linkage conditions. A single-phase problem is formulated on a phase interval $t \in \mathcal{I} = [t_0, t_f]\,,\ t_f > t_0$: Find controls $\mathbf{u} : \mathcal{I} \to \mathbb{R}^{n_u}$ and corresponding states $\mathbf{x} : \mathcal{I} \to \mathbb{R}^{n_x}$ evolving according to the ordinary differential equation

$$\dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), t\right), \tag{1}$$

with boundary conditions

$$\mathbf{h}\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right)\right) = \mathbf{0}, \tag{2}$$

and subject to path constraints

$$\mathbf{g}\left(\mathbf{x}(t), \mathbf{u}(t), t\right) \leq \mathbf{0}, \tag{3}$$

that minimize the Bolza cost function

$$J = \int_{t_0}^{t_f} L\left(\mathbf{x}(t), \mathbf{u}(t), t\right)\, dt + M\left(\mathbf{x}(t_0), \mathbf{x}(t_f), t_0, t_f\right). \tag{4}$$

This Bolza cost function is defined as the sum of the Mayer (endpoint) cost function M and the integral of the Lagrange (running) cost function L.

The initial time t_0 and/or final time t_f can be either fixed or free. If the endpoints of the phase interval are free, a common modeling strategy is to introduce parameters for the endpoints satisfying $t_f > t_0$. The free interval $\mathcal{I}$ is then mapped to the fixed interval $[0, 1]$ via the affine transformation $(t-t_0)/(t_f-t_0)$. It is convenient to collect the parameters of this time parametrization as well as other static[1] parameters in a vector $\mathbf{p}$. If all functions are allowed to depend on $\mathbf{p}$, the following optimal control problem is obtained:

$$\begin{aligned}
\underset{\mathbf{x},\,\mathbf{u},\,\mathbf{p}}{\text{minimize}} \quad & \int_{t_0}^{t_f} L\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t\right)\, dt + M\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}\right) \\
\text{subject to} \quad & \dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t\right), \\
& \mathbf{g}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t\right) \leq \mathbf{0}, \\
& \mathbf{h}\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}\right) = \mathbf{0}, \\
& t \in [t_0, t_f].
\end{aligned}$$

This type of problem can also be considered in parametric form - a form suitable for performing post-optimal sensitivity analysis. For this purpose, a fixed parameter vector $\mathbf{q}$ is introduced which, in contrast to $\mathbf{p}$, is set to a nominal value in the problem formulation:

$$\begin{aligned}
\underset{\mathbf{x},\,\mathbf{u},\,\mathbf{p}}{\text{minimize}} \quad & \int_{t_0}^{t_f} L\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}\right)\, dt + M\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}; \mathbf{q}\right) \\
\text{subject to} \quad & \dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}\right), \\
& \mathbf{g}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}\right) \leq \mathbf{0}, \\
& \mathbf{h}\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}; \mathbf{q}\right) = \mathbf{0}, \\
& t \in [t_0, t_f].
\end{aligned}$$

Following the successful solution of a problem of this type, post-optimal sensitivity analysis can then reveal the sensitivities of the optimal states, controls, parameters, and cost with respect to $\mathbf{q}$.

Phoenix-OC also allows for several extensions to this standard problem. For example, homotopy formulations are inherently supported. This type of formulation is based on a homotopy parameter ρ that ranges from zero (representing a simple problem) to one (representing a difficult problem). The optimal control problem is then solved sequentially, starting from $\rho = 0$ through a sequence of intermediate problems ($0 < \rho < 1$), and finally for $\rho = 1$:

$$\begin{aligned}
\underset{\mathbf{x},\,\mathbf{u},\,\mathbf{p}}{\text{minimize}} \quad & \int_{t_0}^{t_f} L\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}, \rho\right)\, dt + M\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}; \mathbf{q}, \rho\right) \\
\text{subject to} \quad & \dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}, \rho\right), \\
& \mathbf{g}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}, \rho\right) \leq \mathbf{0}, \\
& \mathbf{h}\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}; \mathbf{q}, \rho\right) = \mathbf{0}, \\
& t \in [t_0, t_f].
\end{aligned}$$

After each homotopy step, the solution to the next problem is typically obtained very efficiently, as the previous solution often serves as an excellent initial guess.

[1] The term "static" should be understood here in the context of a phase interval and implies that the parameters exhibit the same value for all time points of a phase. Formally, static parameters can also be viewed as states with an associated zero rate, subject to appropriate bounds.

Another popular extension supported by Phoenix-OC is the modeling of batch runs. In this context the same problem is solved for a (potentially large) number of different parameter values from a batch set $\mathcal{B}$. Introducing the fixed batch parameter vector $\mathbf{b} \in \mathcal{B}$, the formulation for each individual problem is:

$$\begin{aligned}
\underset{\mathbf{x},\,\mathbf{u},\,\mathbf{p}}{\text{minimize}} \quad & \int_{t_0}^{t_f} L\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}, \rho, \mathbf{b}\right)\, dt + M\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}; \mathbf{q}, \rho, \mathbf{b}\right) \\
\text{subject to} \quad & \dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}, \rho, \mathbf{b}\right), \\
& \mathbf{g}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}, \rho, \mathbf{b}\right) \leq \mathbf{0}, \\
& \mathbf{h}\left(\mathbf{x}\left(t_0\right), \mathbf{x}\left(t_f\right), \mathbf{p}; \mathbf{q}, \rho, \mathbf{b}\right) = \mathbf{0}, \\
& t \in [t_0, t_f].
\end{aligned}$$

In Phoenix-OC, the parallelization of batch runs is achieved by distributing the workload across compute processes using a Message Passing Interface (MPI) implementation. Additionally, other levels of parallelism are implemented for each process using OpenMP, enabling parallelization within each individual optimal control problem.

Phoenix-OC also supports several extensions regarding the modeling concepts used in optimal control problems. For example, integral constraints of the form

$$\int_{t_0}^{t_f} \mathbf{r}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}, t; \mathbf{q}, \rho, \mathbf{b}\right) dt \leq \mathbf{0}, \tag{5}$$

are natively supported. Moreover, if a control parametrization of sufficient degree is employed, control derivatives can be directly utilized in model, cost, and constraint functions. In addition, cost and constraint functions can be introduced on arbitrary subintervals of a phase can and also mix state and control values at arbitrary points within and across phase boundaries. This can, for instance, be useful to introduce complex linkage conditions between phases. Another useful modeling concept is the use of subproblems that can be referenced within each problem setup. This implies that an optimal control problem can depend on one or more other optimal control problems. The referenced subproblems can be invoked as regular functions in all cost and constraint functions within a problem definition.

3 Modeling

There exist two main scopes in Phoenix-OC for modeling optimal control problems: The global and the local (phase) scope.

Within the global scope, the user can define an arbitrary number of parameters (optimization variables). Zero or more phases can be specified in a problem setup[2].

Parameters from the global scope are accessible in all phases. Conversely, all quantities (states, controls, etc.) defined within a phase are phase-private[3]. However, global parameters can be linked to arbitrary quantities from any phase and at any discretization point, making them effectively available to all model, cost, and constraint functions throughout the problem setup. This can be particularly useful to define complex links between phases. Notably, standard

[2] If no phases are defined, the optimal control problem effectively reduces to an optimization problem defined solely by the parameters and cost/constraints within the global scope.

[3] A phase-private parameter can be effectively modeled by introducing a control with B-Spline degree zero and a single knot interval. This introduces a single free (control) parameter that is only available within the specific phase scope.

linkage conditions are also supported for connecting phases. This includes linking the states of consecutive phases and/or linking the terminal states of the last phase to the initial states of the first phase for periodic problems.

By default, a parameter is introduced as a free parameter subject to optimization. However, the treatment of a parameter within the context of the optimal control problem can be altered by setting the following modifiers:

- `Passive`: The parameter is treated as a parameterizing constant input fixed to the initial value. If more than one value is specified for a parameter, the problem is solved for all values (batch run).
- `Morphogenic`: The values defined for the parameter create a homotopy sequence for the problem. This implies that the problem is initially solved for the first value and subsequently resolved for all other values. Each time the problem is resolved for a new value, the optimal solution from the preceding problem is used as the initial guess to the next problem.
- `Sensitive`: The parameter is fixed and the post-optimal sensitivities of the cost and optimization variables (parameters, states, and controls) are computed at the optimal solution.

Each phase requires the definition of an independent variable with defined interval endpoints (both can be either fixed or subject to optimization). The initial and final values of this variable can be set to an expression depending on the parameters from the global scope.

Several Legendre-Gauss Lobatto and Radau methods in integral form are available for state discretization within a phase. These include popular methods such as the Hermite-Simpson and trapezoidal method, the third-order RadauIIA method, and both the implicit and explicit Euler method. By default, the state grid points are distributed uniformly. However, the user has the option to provide a virtually arbitrary distribution for the grid points.

For the controls, Phoenix-OC implements B-Spline parameterizations of essentially arbitrary degree. B-Splines are often favored in optimal control formulations due to their flexibility in defining control functions of arbitrary smoothness and their local support property [7]. The default type of parametrization depends on the discretization defined for the phase. This default control parametrization can be overwritten for each control individually, meaning that the degree, the number of knot grid intervals, and the distribution of knot grid points can be chosen separately for each control. In addition, for each control, the derivatives of the interpolant can be incorporated into the problem formulation, e.g., to impose limits on the control rate or set boundary conditions of higher order. Moreover, complete control histories from one phase can be linked to those in other phases, provided they share the same knot grid and degree.

4 Evaluation

OC frameworks typically consist of two main components: the optimization module and the function evaluation module.

The optimization module aims to iteratively improve the optimization variable vector to ultimately locate a minimizer that satisfies the constraints. Phoenix-OC offers the option to dynamically link with IPOPT [11]. This optimization module is particularly known for its effectiveness and efficiency in solving high-dimensional and sparse optimization problems.

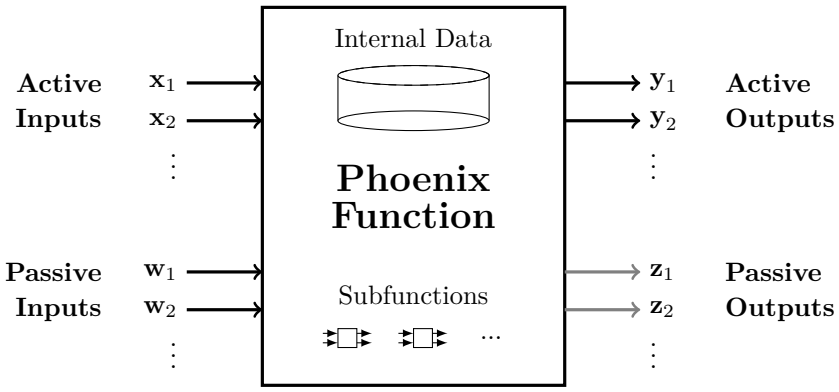

Figure 1: Phoenix Function.

Due to the high dimensionalities of the optimization variables and constraints arising from direct optimal control methods based on segmented collocation, gradient-based optimization algorithms are typically favored.

The task of the function evaluator in this context is to return the cost and constraint values, along with derivative information computed from the current iterate supplied by the optimization module. Note that during the iterative solution, the function evaluator evaluates the same functions, including derivatives, based on different input values multiple times. This fact motivates the exploitation of structural information related to the function evaluation and derivative computation in a preprocessing step.

In Phoenix-CORE, the primary structure of the function evaluator is represented as a rooted tree, with each branch corresponding to a vector-valued function. Each of these functions is a derived type of a base Phoenix Function type (see Fig. 1). A Phoenix Function has two sets of input arguments (active and passive) and two sets of output arguments (active and passive). Each input and output has a fixed size and must include a label, which serves as a unique identifier. Phoenix Functions are required to compute only the derivatives of active outputs with respect to active inputs. Conversely, passive inputs parameterize the function (i.e., 'external' constants), while passive outputs represent additional outputs for which no derivative information is required (e.g., to make internal variables observable without the need to compute derivatives). Additionally, constant data can be supplied to each function at build time (i.e., "internal" constants).

The Phoenix-CORE library supports a variety of built-in function types, including standard operators (inversion, addition, multiplication, etc.). In particular, it includes a graph operator for which an arbitrary number of subfunctions can be specified. This graph operator models a composite function represented as a directed acyclic call graph[4]. The connectivity of the call graph is defined based on the input/output labels assigned to the subfunctions.

[4]Although the graph operator can only model directed acyclic call graphs, this is essentially non-restrictive in practice; after all, each Phoenix Function can implement an arbitrary function.

A central concept heavily employed for structural exploitation throughout Phoenix-CORE is the notion of an Extended Sparsity Pattern (ESP).

Typically, binary-valued sparsity patterns are used to represent the pattern of zero and potentially non-zero derivatives of a function: *If a derivative value is constant zero, the corresponding entry in the sparsity pattern is zero; otherwise, it is one.* Such a binary-valued sparsity pattern reveals basic structural information about the function.

For example, the pattern regarding the first derivatives indicates whether there is an influence of a particular input on a particular output (entry one) or not (entry zero). In Phoenix-CORE, the sparsity pattern concept is extended from binary-valued to floating-point patterns. This extension allows for encoding more information about the function, thereby increasing the potential for structural exploitation.

The main advantage of the ESP is that it encodes information regarding constant derivative values: *If a derivative value is a constant number, the corresponding entry in the sparsity pattern is that number; otherwise, it is Not-a-Number (NaN).*

Clearly, the information contained in this type of sparsity pattern is a superset of that found in binary-valued sparsity patterns. After all, the binary-valued sparsity pattern can be trivially derived from the ESP. Notably, the ESP permits the automatic classification of functions as (multivariate) polynomials of degree zero, one, or two. This additional information can be heavily exploited in function call graphs. For example, it allows for the automatic elimination/replacement[5] of functions in a static call graph. Moreover, it enables the partial pre-computation of constant values as well as derivatives for evaluating the chain rule regarding the derivative computation of the composite function modeled by the graph.

Finally, it is worth noting that Phoenix-OC supports different environments for job submission and execution for a specific instance of an optimal control problem.

A job submission consists of two main parts. The first part is a description of the problem setup. The second part is a job description that references a specific problem setup and defines additional properties to execute the job. These job properties include settings related to the resources to be allocated (e.g., the number of MPI processes and OpenMP threads), custom initialization and finalization commands executed at the beginning and end of the job's execution lifecycle, the environment in which the job is executed (locally or via a load balancer), and job dependencies. After a job submission, the job (including all dependent jobs) is executed in the selected environment and the results are written to a file containing the structured output data. Phoenix-OC also includes a standard postprocessor to facilitate the import/export and visualization of output data.

[5]Note that if all outputs of a vector-valued function can be represented as multivariate polynomials of degree zero, one, or two, both the function and the derivative computation can be replaced by efficient matrix operations.

5 Computational experience

In the following, two examples are presented to analyze the computational time spent in Phoenix-CORE during optimization runs. The focus in this section lies on scalability aspects related to the number of discretization intervals as well as batch processing capabilities.

All examples are computed on a machine with 32 GB of RAM and a 64-bit x86 desktop microprocessor with eight cores operating at 1.7 GHz. The optimization module used for all examples is IPOPT version 3.14.4.

5.1 Discretization intervals

To study the time spent in Phoenix-CORE with respect to the number of intervals, a minimum time formulation for a vehicle moving in the x-y plane is used. The initial position at $t = 0$ is (0,0), and the final position at $t = t_f$ is (100,100). At both endpoints, the velocity v must equal five, and the heading, described by the angle ψ, must align with the direction of the x-axis. The box-bounded controls are the angular velocity ω and the normal acceleration a. Along the trajectory, the velocity must not exceed five, and a velocity-dependent rate constraint is imposed:

$$
\begin{aligned}
&\text{minimize} && t_f \\
&\text{subject to} && \dot{x}(t) = v(t)\cos(\psi(t)), \\
& && \dot{y}(t) = v(t)\sin(\psi(t)), \\
& && \dot{v}(t) = a(t), \\
& && \dot{\psi}(t) = \omega(t), \\
& && x(0) = y(0) = \psi(0) = \psi(t_f) = 0, \\
& && x(t_f) = y(t_f) = 100, \\
& && v(0) = v(t_f) = 5, \\
& && 0 \leq v(t) \leq 5, \\
& && -0.1 \leq a(t) \leq 0.1, \\
& && -0.4 \leq \omega(t) \leq 0.4, \\
& && -1 \leq 2v(t)\omega(t) \leq 1, \\
& && t \in [0, t_f].
\end{aligned}
\tag{6}
$$

The initial guess for the final time is set to $t_f = 30$ and the initial guess for the states and controls are constructed automatically by Phoenix-OC.

For illustration purposes, the optimal solution with 200 discretization intervals is depicted in Fig. 2. The problem is discretized using a low-order method (implicit Euler) and the computational time is analyzed with respect to the number of discretization intervals. The number of optimization variables, the number of constraints (equality and inequality), the number of iterations until convergence, and the total time spent in Phoenix-CORE during the optimization are reported in Table 1. The total time spent in Phoenix-CORE refers to the time spent on NLP function evaluations reported by IPOPT, excluding the time spent within IPOPT itself. This includes the evaluation of the values and gradients of the cost and constraints, as well as the Hessian of the Lagrangian in Phoenix-CORE.

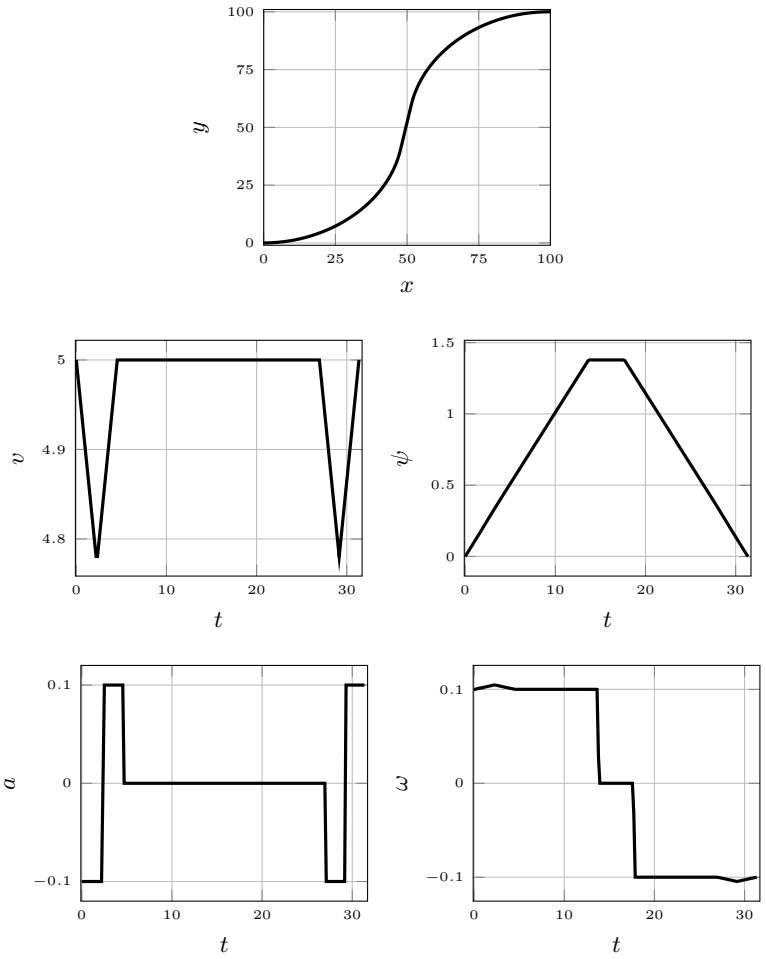

Figure 2: Optimal states and controls.

It should be noted that the largest problem in this study exhibits over one hundred thousand optimization variables, which is clearly excessive for this particular problem.

Intervals	Variables	Constraints	Iterations	Time Function Evaluations
20	118	103	20	1 Millisecond
200	1198	1003	23	7 Milliseconds
2000	11998	10003	22	67 Milliseconds
20000	119998	100003	19	559 Milliseconds

Table 1: Time comparison depending on the number of intervals.

5.2 Batch size

The computation of the backward reachable set for the game of two vehicles moving in a plane using optimal control methods [2] is studied to illustrate the capabilities of batch runs. The target state in this problem formulation is the origin $\mathbf{x} = \mathbf{0}$. A pointwise representation of the set of states from which the target state can be reached on the phase interval $[0, 1]$ is computed using the Distance Fields on Grids method [2]. Here, the batch set $\mathcal{B}$ contains three-dimensional points from a rectangular grid. For each $\mathbf{b} \in \mathcal{B}$, the following optimal control problem is solved:

$$
\begin{aligned}
\text{minimize} \quad & \|\mathbf{x}(t_f) - \mathbf{b}\|_2^2 \\
\text{subject to} \quad & \dot{x}_1(t) = -(-5 + 5\cos(x_3(t)) + u_1(t)x_2(t)), \\
& \dot{x}_2(t) = -(5\sin(x_3(t)) - u_1(t)x_1(t)), \\
& \dot{x}_3(t) = -(u_2(t) - u_1(t)), \\
& x_1(0) = x_2(0) = x_3(0) = 0, \\
& -1 \leq u_1 \leq 1, \\
& -1 \leq u_2 \leq 1, \\
& t \in [0, 1].
\end{aligned}
\tag{7}
$$

The problem is discretized using a higher-order method (Hermite-Simpson) with 20 intervals. The Cartesian grid from which the batch set $\mathcal{B}$ is constructed is defined on $[-4, 4] \times [-4, 4] \times [-4, 4]$ with a discretization of $50 \times 50 \times 50$ points.

For illustration purposes, the projections of the pointwise representation of the three-dimensional reachable set, obtained from the solution of the batch run using Phoenix-OC, are depicted in Fig. 3.

The solution time for the 125000 problems from the batch set distributed across seven MPI processes is approximately 52 seconds.

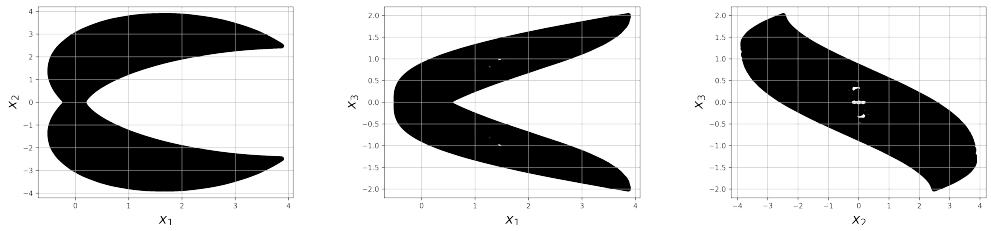

Figure 3: Projections of the reachable set.

6 Applications

In this section, several engineering applications are presented to showcase the modeling capabilities of Phoenix-OC. The same computational resources and optimization module are used as outlined in Section 5.

6.1 Low-thrust orbit transfer

The orbit transfer problem from [4], Chapter 6, considers the minimum-fuel transition from a low Earth orbit to a mission orbit. The model of the space vehicle has seven states (six modified equinoctial coordinates and weight) as well as three controls (normalized thrust direction) and an additional throttle parameter. The optimal trajectory in Cartesian coordinates (x, y, z) in relation to the Earth (radius R_E) computed by Phoenix-OC is shown in Fig. 4.

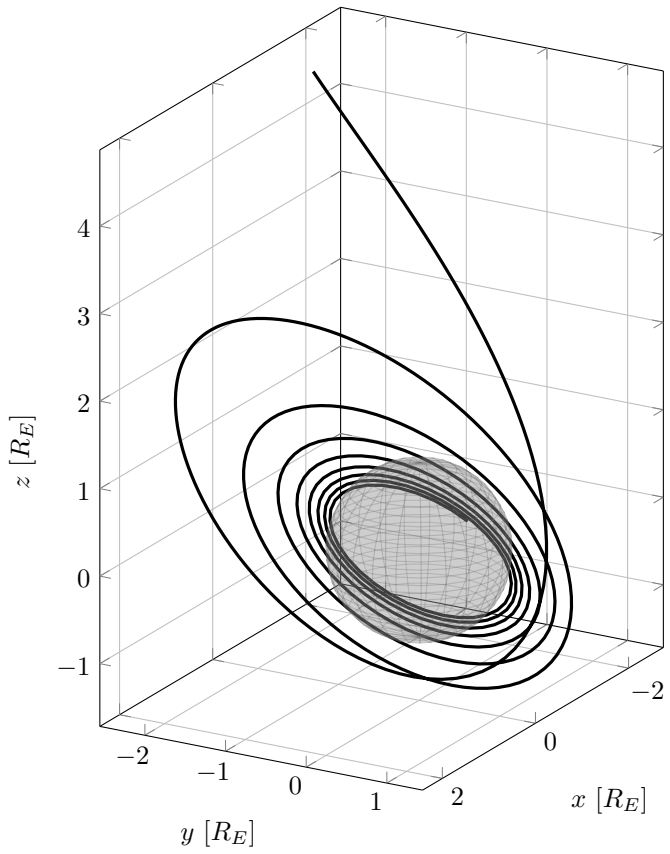

Figure 4: Optimal orbit transfer.

6.2 Manipulator arm

In Reference [7], Chapter 5, the optimal control of a manipulator arm is studied. The configuration space features the three angles $q_{1,\ldots,3}$ of the links. The associated (normalized) torque controls are $u_{1,\ldots,3}$. The control objective is to transfer the manipulator, rest to rest, from an initial configuration to a final position (prescribed in Cartesian space), while minimizing a combined cost of final time and control effort. The optimal states and controls computed by Phoenix-OC are shown in Fig. 5.

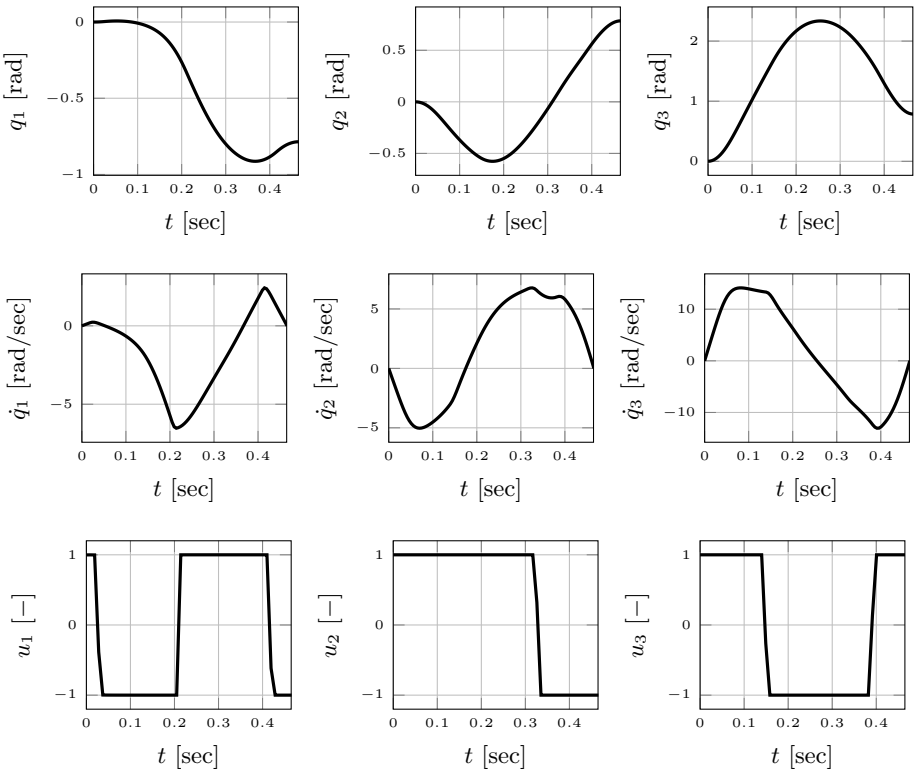

Figure 5: Optimal states and controls.

6.3 Air race

The optimization of air race trajectories represents a challenging optimal control problem due to the high fidelity and large problem size. In Reference [5], Chapter 11, a minimum-time optimal control problem with seven race gates is defined. Each race gate must be passed wings level. The problem is modeled with 16 phases and a rigid-body aircraft model featuring thirteen states (12 rigid-body states and one engine state) and four controls (three inputs for the primary control surfaces and one engine command). The minimum-time path through the race course described in Cartesian coordinates (x, y, h) computed by Phoenix-OC is presented in Fig. 6.

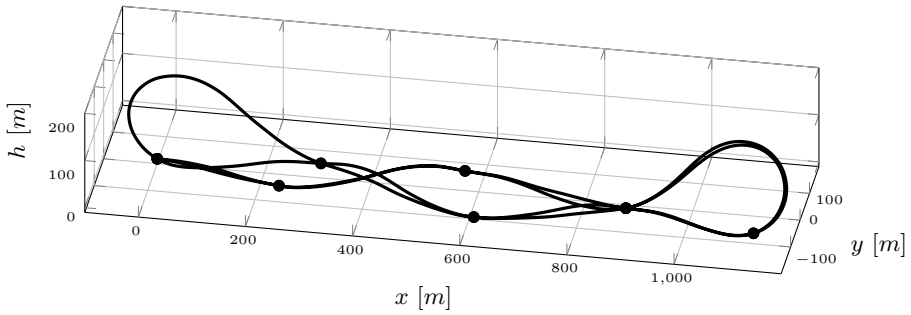

Figure 6: Air race optimal trajectory (rigid-body model).

6.4 Double lane-change

In Reference [7], Chapter 7, the test drive of a car is studied. The problem represents a double lane-change maneuver that should be completed in minimum time. The vehicle is modeled using a single-track car model with seven states, three continuous controls (steering angle velocity, brake force, throttle), and one discrete control (gear setting). The boundaries of the lane and the optimal trajectory in the x-y plane computed by Phoenix-OC are illustrated in Fig. 7. The optimal discrete gear setting μ is shown in Fig. 8.

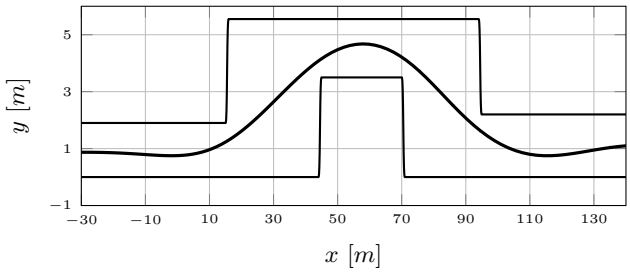

Figure 7: Optimal lane-change.

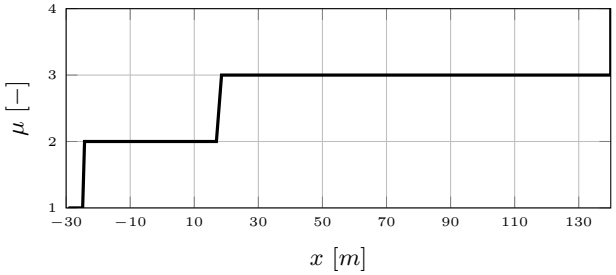

Figure 8: Optimal gear setting.

6.5 Heat partial differential equation

Reference [4], Chapter 4, considers the optimal control of a heating process for a probe. The objective is to control the applied heat on the outside of the probe to obtain a desired temperature profile on the inside. The partial differential equation for the temperature field is discretized in the spatial dimensions using the method of lines.

The optimal distribution of temperature T over the spatial and temporal dimensions (x, t) computed by Phoenix-OC is shown in Fig. 9. The optimal control u is given in Fig. 10.

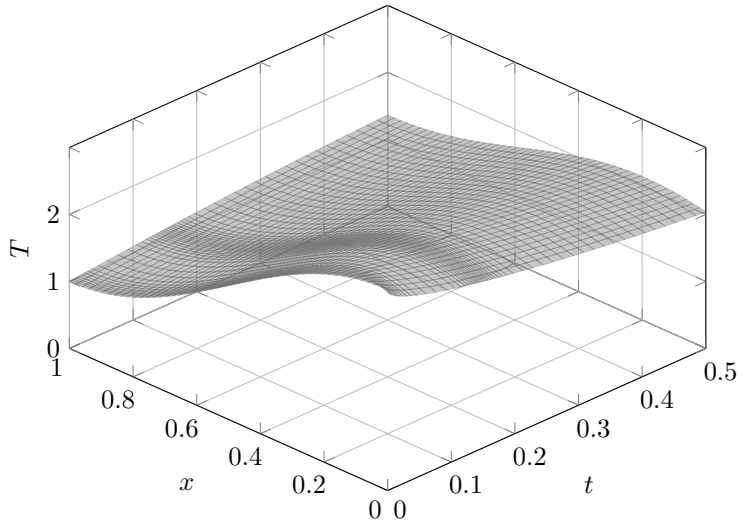

Figure 9: Temperature surface.

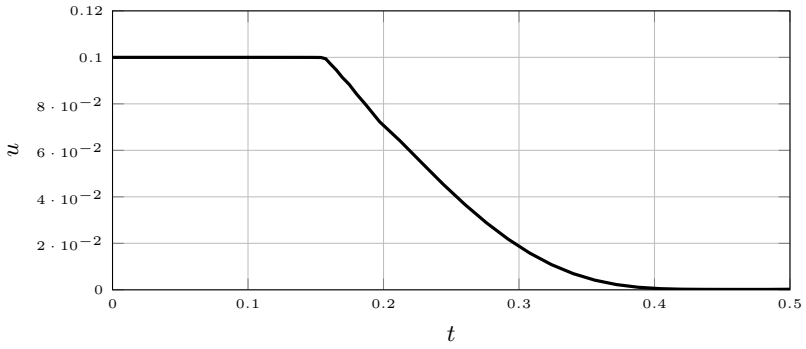

Figure 10: Heat control.

7 Summary

Phoenix-OC is a general-purpose optimal control solver with a focus on large-scale optimal control problems and batch runs over large problem sets. The supported problem classes include multi-phase problems, parametric formulations (including post-optimal sensitivity analysis), homotopy sequences, batch analysis, and bilevel problems.

The underlying computational engine (Phoenix-CORE) implements advanced structure exploitation strategies based on the notion of an extended sparsity pattern to efficiently evaluate the cost and constraint functions, as well as first- and second-order derivatives.

The computational capabilities of Phoenix-CORE and the ability of Phoenix-OC to solve realistic optimal control problems arising from engineering applications in various domains, such as automotive, aerospace, and robotics, are demonstrated by several numerical examples.

References

[1] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. CasADi - a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2018.

[2] R. Baier, M. Gerdts, and I. Xausa. Approximation of reachable sets using optimal control algorithms. *Numerical Algebra, Control and Optimization*, 3(3):519–548, 2013.

[3] V. M. Becerra. Solving complex optimal control problems at no cost with PSOPT. In *IEEE International Symposium on Computer-Aided Control System Design*, pages 1391–1396, Yokohama, Japan, 2010.

[4] J. T. Betts. *Practical Methods for Optimal Control and Estimation using Nonlinear Programming*. SIAM Advances in Design and Control, Philadelphia, PA, 2010.

[5] M. Bittner. *Utilization of Problem and Dynamic Characteristics for Solving Large Scale Optimal Control Problems*. PhD thesis, Technical University of Munich, Munich, 2017. Available online July 30, 2024, mediatum.ub.tum.de/?id=1343164.

[6] C. Büskens and D. Wassel. The ESA NLP solver WORHP. In Giorgio Fasano and János D. Pintér, editors, *Modeling and Optimization in Space Engineering*, volume 73, pages 85–110. Springer New York, 2013.

[7] M. Gerdts. *Optimal Control of ODEs and DAEs*. Walter de Gruyter, Berlin/Boston, 2012.

[8] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002.

[9] M. A. Patterson and A. V. Rao. GPOPS-II: A MATLAB software for solving multiple-phase optimal control problems using Hp-adaptive gaussian quadrature collocation methods and sparse nonlinear programming. *ACM Transactions on Mathematical Software*, 41(1):1–37, 2014.

[10] M. Rieck, M. Bittner, B. Grüter, J. Diepolder, P. Piprek, C. Göttlicher, F. Schwaiger, B. Hosseini, F. Schweighofer, T. Akman, and F. Holzapfel. *FALCON.m User Guide v1.31*. Institute of Flight System Dynamics, Technical University of Munich, 2024. Available online July 30, 2024, www.fsd.ed.tum.de/software/falcon-m.

[11] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.