SAND2020-5271C

Open-Source, Object-Oriented, Multi-Phase Pseudospectral Optimization Using Pyomo

Rachel Schlossman, * Kyle R. Williams†, David Kozlowski, ‡ and Julie Parish §
Sandia National Laboratories, Albuquerque, NM 87185-1157

Multi-phase, pseudospectral optimization is employed in a variety of applications, but many of the world-class optimization libraries are closed-source. In this paper we formulate an open-source, object oriented framework for dynamic optimization using the Pyomo modeling language. This strategy supports the reuse of common code for rapid, error-free model development. Flexibility of our framework is demonstrated on a series of dynamic optimization problems, including multi-phase trajectory optimization using highly accurate pseudospectral methods and controller gain optimization in the presence of stability margin constraints. We employ numerical procedures to improve convergence rates and solution accuracy. We validate our framework using GPOPS-II, a commercial MATLAB-based optimization program, for a boost-glide problem. The results show close alignment in trajectory results with this state-of-the-art optimization suite.

I. Nomenclature

α	=	angle of attack, rad
γ	=	vertical flight path angle, rad
γ_{air}	=	heat capacity ratio for air, 1.4
θ	=	longitude, rad
μ	=	Earth's gravitational constant, 3.99e14 m^3/s^2
β	=	sideslip angle, rad
ρ	=	density, kg/m^3
ρ_0	=	atmospheric density on Earth's surface, 1.23 kg/m^3
σ	=	bank angle, rad
ϕ	=	latitude, rad
ψ	=	heading, rad
C_A, C_N, C_Y	=	coefficient of axial, normal, and side force
C_L, C_D, C_S	=	coefficient of lift, drag, and side force
D	=	drag, N
g	=	gravity, 9.81 m/s^2
h	=	altitude, m
h_0	=	density scale height for exponential density model, 7254.24 m
I	=	moment of inertia, $kg\text{-}m^2$
L	=	lift, N
m	=	mass, kg
r	=	geocentric radius, m
R_{air}	=	specific gas constant for air, J/(kg-K)
R_p	=	Earth polar radius, 6.45e6 m

*Member of Technical Staff, 05447 Autonomous Sensing and Control, P.O. Box 5800 MS 1163.
†Principal Member of Technical Staff, 05447 Autonomous Sensing and Control, P.O. Box 5800 MS 1161
‡Distinguished Member of Technical Staff, 05442 Navigation Guidance & Control, P.O. Box 5800 MS 1174
§Manager, 05449 Autonomy for Hypersonics, P.O. Box 5800 MS 1174

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

S	=	side force, N
S_{ref}	=	reference area, m^2
T	=	thrust, N
T_{atmos}	=	atmospheric temperature, Kelvin
v	=	velocity, m/s

II. Introduction

In recent decades, information technology has had an explosion of technological progress that many other fields have not shared. Much of that progress has been built on the backs of open-source software. Pseudospectral optimization has played a key role in quick generation of trajectories that are guaranteed to respect dynamic and problem-specific constraints. Multi-phase trajectory optimization, which breaks a trajectory into sub-components, allows the additional benefit of applying phase-specific constraints along a trajectory as needed. Multi-phase trajectory optimization is of interest in a variety of applications, including in civil aviation [1], underwater target tracking [2], and hypersonic vehicle missions. However, much of the quality software in this domain is behind license pay walls.

Pseudospectral optimization software has been available for years, but many of the widely-used used software packages are closed-source or restricted in availability. Fortran-based OTIS4 [3] has been used to apply pseduospectral optimization to spacecraft trajectory design [4], but it is subject to International Traffic in Arms Regulations restrictions. The Sparse Optimization Suite [5] and DIDO have been used to solve multi-phase problems, and are subject to license restrictions. GPOPS, a now closed-source, Matlab-based optimization solver that leverages the Gauss pseudospectral method, is leveraged frequently to solve multi-phase problems. GPOPS-II is an extension of GPOPS and has also been leveraged for path planning for flight vehicles [6],[7]. Our multi-phase optimization scheme relies on fully open-source software: Python, Pyomo, and IPOPT.

Pyomo [8] is a general, open-source, Python-based algebraic modeling language developed at Sandia National Labs. This R&D100-winning (2016) package allows the user to implement a mathematical model in natural syntax; this model is automatically differentiated and packaged to interface with a solver. The Pyomo extension `pyomo.dae` [9] is a Python-based modeling framework that is integrated with Pyomo. `pyomo.dae` enables high-level specification of multi-phase optimization problems with differential and algebraic equations (DAEs). In this paper, `pyomo.dae` is coupled with IPOPT [10], a nonlinear interior point optimization software. While Pyomo was not originally devised with a flight-application focus, its general framework facilitates the formulation of trajectory planning problems for a variety of problems in this paper, including a multi-phase boost-glide problem.

Optimization problems with phase-specific constraints have been solved with phases considered individually, in separate planning problems, or as a whole. In [11] pseudospectral methods and DIDO are used in two of three sequentially-connected optimization problems for a three-stage launch system. The employed approach requires optimizing the final-stage objective and solving backwards for the rest of the trajectory, thereby creating trajectory segments that do not directly support the final objective. In contrast to this work, our multi-phase optimization approach allows for the design of a full trajectory that is guaranteed to directly optimize the single objective of interest. The authors of [12] formulate a three-phase trajectory for a lunar landing problem using DIDO. We demonstrate the robustness of our approach with a six-phase trajectory, while emphasizing an object-oriented framework that allows ease in changing phase-specific constraints

GPOPS and GPOPS-II are widely-used software, but obtaining licenses for this software and Matlab are required to leverage these libraries. In [1] a five-phase flight scenario with a minimum-fuel consumption objective is solved with GPOPS. In addition to solving a similarly-sized problem efficiently*, our multi-phase boost-glide example problem includes obstacle avoidance. While the code in [13] demonstrates the GPOPS-II multi-phase capabilities, it is comprised of a sequential build-up of the phases, with the dynamic equations serially repeated for each phase. A contribution of our work is an object-oriented formulation of the system model and phases, in order to minimize code repetition and to ease switching out model components.

A. Contributions

The goal of this paper is to demonstrate a robust framework for multi-phase pseudospectral optimization that leverages only open-source software. We accomplish this by devising an object-oriented structure that captures the

*1–10 minutes and "up to 12,500 constraints and 10,000 nonlinear decision variables" [1] compared to our problem with 12,405 decision variables and 11,877 constraints in less than a minute.

common dynamics and environmental influences among phases, but also allows differing phase constraints to be easily encoded. Our approach allows nonlinear dynamics to be imposed as equality constraints, while guaranteeing that phase-specific constraints are also respected. Robustness is encouraged through adaptive meshing and using a two-iteration approach with warm starting when obstacles are present. While we are not the first to study multi-phase optimization, our work is novel in leveraging the modern and open-source tool, Pyomo, along with other open-source software, for this application. A second key contribution of this work is the construction of an object-oriented optimization framework, which allows reusability of structure between different problems, and ease of testing and modifying an optimization problem. We demonstrate our approach's capabilities through three simulated example problems. In a simulated boost-glide problem with a final speed objective, our results are comparable to those produced by the GPOPS-II software. We also demonstrate the versatility of our object-oriented approach in a planar, multi-phase toy problem and a single-phase, control system gain selection problem.

III. Multi-Phase, Pseudospectral Optimization Problem Formulation

We consider multi-phase dynamic optimization problems constrained by differential and algebraic equations. For any given phase, $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, and $y \in \mathbb{R}^{n_y}$ are the state, control, and algebraic variable vectors at a single time step, respectively. Define the state, input, and algebraic variable trajectories for a single phase as $X^{(p)} \triangleq x^{(p)}(t)$, $U \triangleq u^{(p)}(t)$, and $Y^{(p)} \triangleq y^{(p)}(t)$, $t \in \left[t_0^{(p)}, t_f^{(p)}\right]$, where phases $p = 1, \ldots, P$. The variable $t_0^{(p)}$ is the initial time and $t_f^{(p)}$ is the final time of phase p. The X, U, and Y trajectories for each phase are concatenated to form the trajectory for the entire problem: $X^* \triangleq \left[X^{(1)}, ..., X^{(P)}\right]$, $U^* \triangleq \left[U^{(1)}, ..., U^{(P)}\right]$, and $Y^* \triangleq \left[Y^{(1)}, ..., Y^{(P)}\right]$. The general optimal control formulation is then:

$$\underset{X^*, U^*, Y^*}{\text{minimize}} \quad J = \Phi\left(X^*, U^*, Y^*\right) \tag{1a}$$

$$\text{subject to} \quad \dot{x}^{(p)} = f^{(p)}\left(x^{(p)}, u^{(p)}, y^{(p)}, t^{(p)}\right), \quad p = 1, \ldots, P \qquad \text{Dynamics} \tag{1b}$$

$$0 \geq g_i^{(p)}\left(x^{(p)}, u^{(p)}, y^{(p)}, t^{(p)}\right), \quad p = 1, \ldots, P, \;\; i = 1, \ldots, n_i \qquad \text{Inequality Constraints} \tag{1c}$$

$$0 = g_e^{(p)}\left(x^{(p)}, u^{(p)}, y^{(p)}, t^{(p)}\right), \quad p = 1, \ldots, P, \;\; e = 1, \ldots, n_e \qquad \text{Equality Constraints} \tag{1d}$$

$$x^{(p)}(t_f) = x^{(p+1)}(t_0), \quad p = 1, \ldots, P-1 \qquad \text{Boundary Continuity} \tag{1e}$$

Phase time $t^{(p)} \in [t_0^{(p)}, t_f^{(p)}]$ is changed to $\tau^{(p)} \in [0, 1]$, and the dynamics are redefined with a now-standard transformation [14]:

$$t^{(p)} = (t_f^{(p)} - t_0^{(p)})\tau^{(p)} + t_0^{(p)} \tag{2a}$$

$$\frac{dx^{(p)}}{d\tau^{(p)}} = \frac{dt^{(p)}}{d\tau^{(p)}} \frac{dx^{(p)}}{dt^{(p)}} \tag{2b}$$

$$= (t_f^{(p)} - t_0^{(p)}) f^{(p)}\left(x(\tau), u(\tau), y(\tau), \tau; t_0, t_f\right)^{(p)} \tag{2c}$$

for $p = 1, \ldots, P$. This simple transformation of the time horizon allows `pyomo.dae` to treat phase final time, $t_f^{(p)}$, and phase initial time, $t_0^{(p)}$, as explicit optimization variables.

A. Discretization

`pyomo.dae` solves the optimal control problem (1) through direct collocation by applying a discretization scheme along the horizon $\tau \in [0, 1]$. This process transforms the optimal control problem into a nonlinear optimization problem [15]. Multiple discretization schemes are available in `pyomo.dae`. Two schemes are briefly discussed here, and more details can be found in [9]. The first scheme discussed is the backward Euler finite difference scheme, the second is an orthogonal collocation method based on Legendre-Gauss-Radau (LGR) nodes. Regardless of the scheme, discretization produces N_p indices with boundary conditions $x_0^{(p)} = x^{(p)}(0)$ and $x_{N_p-1}^{(p)} = x^{(p)}(1)$.

1. Finite Difference Scheme

For the numerical solution of trajectory optimization problems, `pyomo.dae` can implement the backward Euler finite difference scheme[†] in which the horizon is discretized into $\mathcal{N}$ finite elements. For a given phase, the discretization scheme is given by:

$$\frac{x_{k+1} - x_k}{h_k} = \frac{dx_{k+1}}{d\tau}, \quad k = 0, \ldots, \mathcal{N} - 1 \tag{3a}$$

$$\frac{dx_{k+1}}{d\tau} = t_f f(x_{k+1}, u_{k+1}, y_{k+1}, \tau_{k+1}; t_0, t_f), \quad k = 0, \ldots, \mathcal{N} - 1 \tag{3b}$$

where $x_k = x(kh_k)$ and $h_k \in (0, 1]$ is the size of finite element k, i.e., the variable step size. An additional index is inserted at the end of the horizon so that $x_{N_p-1} = x(1)$, where $N_p = \mathcal{N} + 1$ is the total number of discretized indices.

2. Orthogonal Collocation Scheme

`pyomo.dae` can also implement orthogonal collocation schemes [15], in which case the state vector is approximated with an orthogonal polynomial. The horizon is discretized into N_p finite elements, then over each finite element the profile of the differential variable $x(t)$ is approximated using a Lagrange interpolating polynomial of order $K - 1$. For a given finite element within a given phase the discretization scheme is given by:

$$\frac{1}{h}\sum_{j=0}^{K} D_{kj} x_j = \frac{dx_k}{d\tau}, \quad k = 1, \ldots, K \tag{4a}$$

$$\frac{dx_k}{d\tau} = t_f f(x_k, u_k, y_k, \tau_k; t_0, t_f), \quad k = 1, \ldots, K \tag{4b}$$

where $x_k = x(\tau_k h)$ for $k = 1, \ldots, K$, $h \in (0, 1]$ is the size of the finite element, $\tau_k \in (0, 1]$ is the location of the k^{th} LGR node in the finite element, K is the number of collocation nodes in the finite element, and $[D_{kj}]$ is the $K \times (K + 1)$ differentiation matrix associated with the j^{th} Lagrange polynomial basis evaluated at node τ_k [16]. In `pyomo.dae` the user must specify $\mathcal{N}$ and K within each finite element. An additional discretization point is added at $\tau_0 = 0$ within the first finite element so that $x_0 = x(0)$. The total number of discretized indices is therefore $N_p = \mathcal{N} \times K + 1$. Further details of the orthogonal collocation implementation in `pyomo.dae` are found in [9].

B. Derivatives Required by the NLP Solver

In the following investigations, `pyomo.dae` leverages IPOPT, which is compiled with the automatic differentiation library "AMPL Solver Library" (ASL). Once the differential and algebraic equations have been specified by the user, Pyomo writes the model into a format recognized by ASL so that exact first and second order derivatives can be constructed and exploited during the optimization process. Although exact second order derivatives are used in this work, an alternative option is a limited-memory quasi-Newton approximation such as the BFGS update. Switching between exact or approximate second order derivatives is performed in the Pyomo user interface.

C. Adaptive Meshing

By default, `pyomo.dae` distributes the $\mathcal{N}$ finite elements (plus an additional point at either the start or end of the horizon) uniformly along the horizon $[0, 1]$. In many problems the magnitudes of the state derivatives are comparably larger in specific regions of the horizon and smaller in others. To improve accuracy of the algebraic approximation of the differential state variables, it can be desirable to increase the density of the finite element distribution in areas where the state derivatives are large. To this end, a simple method is designed which redistributes the location of each finite element according to a target density function, $f(\tau)$:

$$\begin{aligned} f(\tau[n]) &= \left|\frac{dx/dt(\tau[n])}{\bar{x}}\right| \times \left|\tau[n] - \tau[n-1]\right| \text{ for } n = 2, \ldots, N_p \\ f(\tau[1]) &= 0 \end{aligned} \tag{5}$$

Here $\tau[n] \in [0, 1]$ is the location of the n^{th} time step, $dx/dt(\tau[n])$ is the derivative of state variable x at $\tau[n]$, and $\bar{x}$ is the average value of x. This density function is the normalized value of the state derivative scaled by the distance

[†] This can be derived by taking a Taylor series expansion of $x(t_k)$ about $x(t_k + h)$ to $O(h^2)$: $x(t_k) = x(t_k + h) - hx'(t_k + h) + O(h^2)$.

between successive indices. In the case of multiple state variables, (5) is averaged over all state variables. The location of each finite element is then redistributed according to:

$$\begin{aligned} F(\tau[j]) &= \frac{\sum_{i=1}^{j} f(\tau[i])}{\sum_{i=1}^{N_p} f(\tau[i])}, \text{ for } j = 1, \ldots, N_p \\ u(i) &= \frac{i}{\mathcal{N}}, \text{ for } i = 1, \ldots, \mathcal{N} \\ \Phi(i) &= \texttt{interp}(u(i), \{F(\tau[1]), \ldots, F(\tau[N_p])\}, \{\tau[1], \tau[2], \ldots, \tau[N_p]\}), \text{ for } i = 1, \ldots, \mathcal{N} \end{aligned} \tag{6}$$

where $\Phi(i) \in (0, 1]$ is the new end location of the i^{th} finite element and $F(\tau)$ is the cumulative distribution function for the target density $f(\tau)$.

The last line of (6) is an implementation of the inverse transformation method, which states that samples can be generated from any probability distribution given the inverse of the cumulative distribution function. The `interp` function fulfills the role of inverting $F(\tau)$ along $\mathcal{N}$ points between $[0, 1]$, since its domain is $\{\tau[1], \tau[2], \ldots, \tau[N_p]\}$. Therefore, the last line distributes the $\mathcal{N}$ finite element end locations between 0 and 1 according to the target density $f(\tau)$. Additionally, an automated regularization procedure is incorporated which limits the slope of $F(\tau)$. This slope limit is determined automatically to prevent the distance between each finite element from exceeding minimum and maximum user-defined values.

To implement the strategy on any given optimization problem, the problem is first solved using a default uniform mesh. The adaptive mesh scheme is then called and provided information from the previous solve. The mesh is then adjusted in preparation for the next solve. This procedure can then be repeated, typically until convergence in the mesh is observed. The adaptive mesh scheme is demonstrated on the *Hypersensitive Problem* taken from [17]. The code is available at [18].

D. Warm Start Procedure

The solution of complex multi-phase dynamic optimization problems can benefit from an iterative process. In this work we found it helpful to solve a sequence of simplified problems increasing in difficulty leading towards the original problem, as will be described in detail in Sec. IV.C and IV.D. To expedite this process we have formulated an automated warm-starting procedure which saves all Pyomo `Vars`, `DerivativeVars` and `ContinuousSet` components after each solve. On each iteration, all components of the previous solve are interpolated along the possibly new horizon discretization and provided as an initial guess to the new problem. The warm start code is provided at [18].

IV. Boost-Glide Example in Simulation

To demonstrate the capabilities of our approach, we consider trajectory generation for a boost-glide problem in simulation. A two-stage booster propels a glide-body to a desired altitude for hand-off. The constraints and key considerations of sections of the boost and glide trajectories vary, thus meriting a multi-phase problem. The attitude degrees of freedom (DOFs) for any given phase are angle of attack, α, bank angle, σ, and sideslip, β. We consider a bank-to-turn strategy for the boost problem ($\beta = 0$), while the glide vehicle can alter all three degrees of freedom.

Considering (2), in this work we utilize time-independent phases, allowing the assumption $t_0^{(p)} = 0$. Consequently, optimization variable $t_f^{(p)}$ is treated as the phase duration. (We discuss in the subsequent sections that time durations for Phases One, Two, and Three are fixed.) Phase horizons $\tau^{(p)}$ are constructed in Pyomo as `ContinuousSet` components. Phase-specific variables are created as `Var` components indexed along the associated `ContinuousSet` component. State derivative variables are likewise created as `DerivativeVar` components. We consider no algebraic variables in this problem. A portion of our model construction approach (contained within Vehicle3DOF, as discussed in Sec. IV.C) is shown in Listing 1, and a more extensive example is shown in [9].

Listing 1 Creating Phase-Dependent Components in Pyomo

```
m = ConcreteModel()  #Define the Pyomo model
m.tau1  = ContinuousSet(bounds=(0,1)) #Phase 1 horizon
m.tau2  = ContinuousSet(bounds=(0,1)) #Phase 2 horizon
m.x1 = Var(m.tau1) # Variable indexed along horizon 1
m.x2 = Var(m.tau2) # Variable indexed along horizon 2
m.dx1dtau1 = DerivativeVar(m.x1, wrt=m.tau1)
m.dx2dtau2 = DerivativeVar(m.x2, wrt=m.tau2)
```

A. Dynamics

In this 3-DOF model there are no rotational dynamics and attitude angles are the control input. The geocentric model used is similar to that developed in [19], the only exception being that we use an altitude, latitude, longitude coordinate system in place of a north, east, down coordinate system. This model is also similar to that developed in [20], except that the effect of Earth's rotation is ignored, and the effect of β is included in the glide phase. We define the state as $x \triangleq [h \;\; v \;\; \theta \;\; \phi \;\; \gamma \;\; \psi]^T \in \mathbb{R}^6$ and the input as $u \triangleq [\alpha \;\; \sigma \;\; \beta]^T \in \mathbb{R}^3$. The equations of motion are given by:

$$\dot{h} = v \sin\gamma \tag{7}$$

$$\dot{v} = \frac{T\cos\alpha - D}{m} - g\sin\gamma \tag{8}$$

$$\dot{\theta} = \frac{v}{h+Re}\cos\gamma\frac{\sin\psi}{\cos\phi} \tag{9}$$

$$\dot{\phi} = \frac{v}{h+Re}\cos\gamma\cos\psi \tag{10}$$

$$\dot{\gamma} = \frac{T\sin\alpha + L\cos\sigma - S\sin\sigma}{mv} + \left(\frac{v}{h+Re} - \frac{g}{v}\right)\cos\gamma \tag{11}$$

$$\dot{\psi} = \frac{L\sin\sigma + S\cos\sigma}{mv\cos\gamma} + \frac{v}{h+Re}\cos\gamma\sin\psi\frac{\sin\phi}{\cos\phi} \tag{12}$$

Each phase is discretized into N_p time steps along the horizon $\tau \in [0,1]$ according to one of the schemes discussed in Section III.A. N_p is allowed to vary between phases. Following Section III.A, $\tau_0^{(p)} = 0$, $x_0^{(p)} = x^{(p)}(0)$, $u_0^{(p)} = u^{(p)}(0)$, and $\tau_{N_p-1}^{(p)} = 1$, $x_{N_p-1}^{(p)} = x^{(p)}(1)$, $u_{N_p-1}^{(p)} = u^{(p)}(1)$.

1. Boost Phases

The booster trajectory is comprised of five phases. Phase One captures the dynamics with the first stage propelling the vehicle. We assume that the remaining mass of the Phase One stage drops immediately when all of the propellant mass is expended. A fixed time equal to $t_{\text{fixed}}^{(2)}$, during which there is no thrust, passes during Phase Two. The second stage then engages and the third phase initiates. The fourth phase begins once the vehicle drops the remaining second stage mass, and the vehicle follows a ballistic trajectory to a desired altitude. Though it is not guaranteed that Phase Four will terminate at apogee, we introduced this phase to tune the upper altitudes of the trajectory. The fifth phase is also ballistic, and terminates at a desired altitude, $h_f^{(5)}$, to hand off to the glide problem. The manner in which these constraints are encoded is discussed further in Sec. IV.D.

The vaccuum thrust profiles of the boosters in this simulation are known. We encode the thrust at a given time as a fifth order polynomial, $g_T^p(\hat{t})$, that is a function of time, $\hat{t} = \tau^{(p)} \times t^{(p)}$. We leverage a polynomial approximation of pressure as a function of altitude, $P(h^{(p)}(\tau^{(p)}))$. The available thrust is then vacuum thrust adjusted by the external pressure times the nozzle area, A_{nozzle}:

$$T^{(p)}(\tau_n^{(p)}) = g_T^{(p)}(\hat{t}) - P(h_n^{(p)}(\tau_n^{(p)})) \times A_{\text{nozzle}}^{(p)} \quad p = 1,3, \;\; n = 0,\ldots,N_p - 1 \tag{13}$$

The final times for these thrust profiles are also known, and so the final times of these phases, $t_{\text{fixed}}^{(1)}$ and $t_{\text{fixed}}^{(3)}$, are fixed. Expulsion of propellant and booster separations cause time-varying mass during in Phases One and Three. For

simplicity we assume the resulting mass flow rate is constant regardless of thrust. This detail can be incorporated easily, assuming that the relationship between thrust and mass flow rate is known. Each of these phases considers an initial mass, $m_{\text{init}}^{(p)}$ and a constant mass flow rate. During Phases Two, Four, and Five, the vehicle mass, $m_{\text{fixed}}^{(p)}$, is constant. Additionally, mass during phases Four through Six is equal to the glide body mass. These constraints are encapsulated in Sec. IV.D.

2. Glider Phase

The glide body motion phase occurs during Phase Six, where the glide body is detached from the booster. During this section of the trajectory, β can be nonzero and thrust is zero. The aerodynamic coefficients of the glider phase are generated by surrogate approximations to NASA X-43 vehicle data [21]. Further details are discussed in Section IV.B.2.

B. Environment

The 3-DOF vehicle dynamics rely on computations of L, D, and S. These external forces are affected by the surrounding atmospheric effects. A simple model for atmospheric temperature is given as:

$$T_{atmos} = -40 + 273.15. \tag{14}$$

This constant temperature serves as a pseudo-average for atmospheric temperature. To increase temperature accuracy, one could leverage a polynomial approximation of the US1976 atmospheric temperature model. However, we have found that this approach does not affect the optimization results and the problem runs slightly more slowly. Gravity is computed as:

$$g = \frac{\mu}{r^2}, \tag{15}$$

and geocentric radius is:

$$r = h + R_p. \tag{16}$$

1. Surrogate for Density Model

Accurately modeling Earth's atmospheric density, ρ, as a function of altitude is critical for accurate simulation. Ideally, numerical data would be used to directly implement density. However, Pyomo requires constraints to be formed as algebraic expressions, eliminating the possibility of a lookup table implementation. To this end a simple surrogate density function is designed below, using activation functions to represent density as a piecewise exponential model:

$$\rho = 1.225\exp(-0.105\times10^{-3}h)\sigma_1(h) + 1.997\exp(-0.156\times10^{-3}h)\sigma_2(h) \tag{17}$$

Here, $\sigma_i(h) = \frac{1}{2}\left(\tanh\frac{h-a_i}{\mu} + \tanh\frac{b_i-h}{\mu}\right)$ are activation functions whose outputs depend on h. These activation functions are differentiable with respect to the input variable, and are therefore compatible with Pyomo. A comparison of the 1976 U.S. Standard Atmosphere Density Model is shown in Fig. 1. The general use of activation functions to construct differentiable piecewise functions is helpful when modeling tabular data for use in Pyomo.

2. L,D,S Using Body Axis Forces

With T_{atmos}, g, r, and ρ computed, we can compute L, D, and S for the 3-DOF model. The lift, drag, and inertial side forces are defined relative to the *velocity vector*, defined by:

$$\begin{aligned} L &= F_N\cos\alpha - F_A\sin\alpha \\ D &= (F_N\sin\alpha + F_A\cos\alpha)\cos\beta - F_Y\sin\beta \\ S &= (F_N\sin\alpha + F_A\cos\alpha)\sin\beta + F_Y\cos\beta \end{aligned} \tag{18}$$

The normal, axial and body side force are defined relative to the *body frame*, defined by:

$$\begin{aligned} F_N &= 0.5\rho v^2 S_{ref} C_N(Mach,\alpha) \\ F_A &= 0.5\rho v^2 S_{ref} C_A(Mach,\alpha) \\ F_Y &= 0.5\rho v^2 S_{ref} C_Y(Mach,\alpha,\beta) \end{aligned} \tag{19}$$

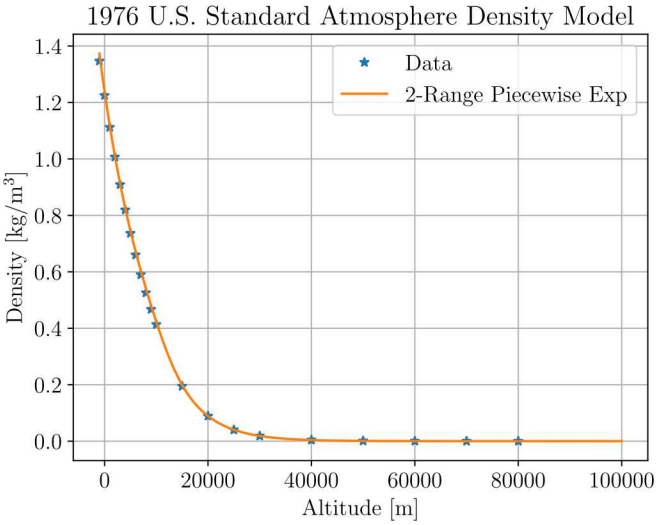

Fig. 1 Surrogate density model compared to 1976 U.S. Standard Atmosphere. The two-range piecewise exponential is created with (17).

The aerodynamic coefficients C_A, C_N, C_Y are dependent on Mach number,

$$Mach = \frac{v}{\sqrt{\gamma_{air} R_{air} T_{atmos}}},$$

α, and/or β. The effects of elevator, rudder and aileron deflections on the aerodynamic coefficients are ignored in this model.

As seen in (19), aero coefficients C_N, C_A, and C_Y depend on the optimization variables, v, α, and β. As discussed previously, use of lookup tables to directly compute these values is not possible. High-order (e.g. 9^{th} order) polynomial surrogates are used to approximate the aerodynamic coefficients to high accuracy ($R^2 > 0.96$) across a wide range of Mach Number, angle of attack, and sideslip angle.

3. L,D,S Using Simplified Model

Where aerodynamic data is classified, we employ an approximation for the lift, drag, and side force coefficients. The lift, drag, and inertial side force are defined relative to the *velocity vector*, defined by:

$$\begin{aligned} L &= 0.5\rho v^2 S_{ref} C_L \\ D &= 0.5\rho v^2 S_{ref} C_D \\ S &= 0.5\rho v^2 S_{ref} C_S \end{aligned} \tag{20}$$

with coefficients [20]:

$$\begin{aligned} C_L &= 1.57\alpha \\ C_D &= 1.65\alpha^2 + 0.06 \\ C_S &= 0 \end{aligned} \tag{21}$$

This simple model is employed for Phases Three, Four and Five in our optimization problem.

C. Object-Oriented Framework

A class-based framework facilitates straightforward construction of a multi-phase problem with minimal code repetition. Fig. 2 shows our architecture. The Vehicle3DOF class serves as the central component within this structure. When an object of this class is initialized, the Pyomo model is created, the phase-dependent components are defined (similar to Listing 1), and the variables are discretized. This class also contains all constraints that will be applied to the problem in a format that is agnostic to specific phases. Listing 2 shows how, for example, Eq. (8) can be applied to all

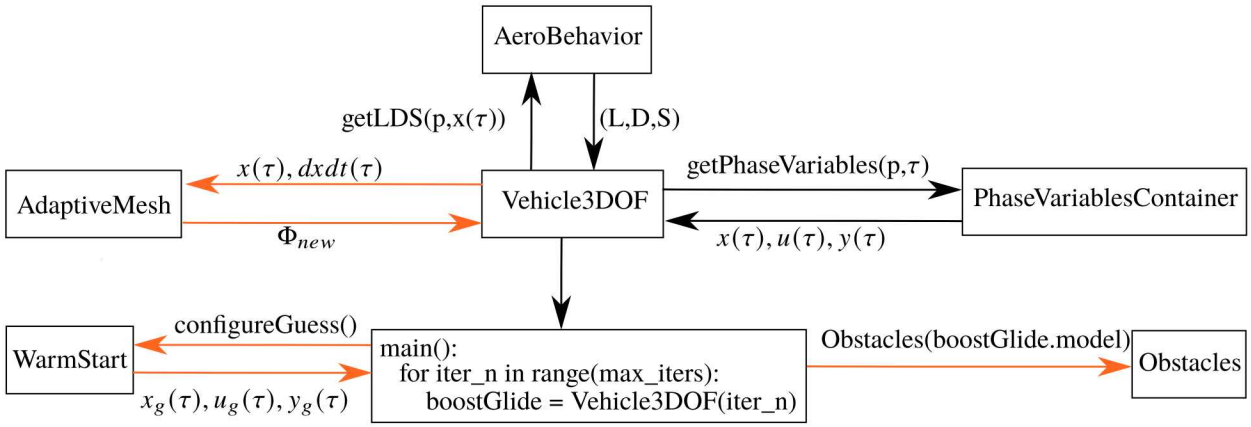

Fig. 2 Framework for the construction of a multi-phase optimization problem. Vehicle3DOF and AeroBehavior are the required classes to construct a multi-phase problem, and the getPhaseVariables function allows phase-specific constraints to be encoded. The arrows in orange indicate the components that can be in be implemented after the first iteration of the optimization problem. The number of iterations to execute is defined by max_iters. The variable iter_n is used as a flag within the main function and by the Vehcile3DOF to determine when to employ these optional components. Based on the results from the previous iteration, the AdaptiveMesh function returns the new finite element end locations in the array Φ_{new}. (See Sec. III.C.) If obstacles are present, the model is passed by reference to the Obstacles class in order to append avoidance constraints to the model. The x, u, and y trajectories from the previous iteration can also be used as the initial trajectory guesses, x_g, u_g, and y_g, respectively, for the next optimization problem.

phases without repeating the constraint explicitly for each phase. The integer value p serves as a flag to indicate to which phase the constraint is being applied. The function *getPhaseVariables* then returns all variables that correspond to Phase p.

The Vehicle3DOF class is a sub-class of AeroBehavior. The AeroBehavior class contains the aerodynamic data, which varies based on state. When the Vehicle3DOF object calls the aeroLiftDrag method, inherited from the AeroBehavior class, the phase variable p again indicates which data to use. This method then returns L, D, and S for the current state.

The main file instantiates an instance of the Vehicle3DOF class. Here we also add all constraints to the model. Listing 3 shows how the velocity constraint for Phase 1 is added to the model. (The flag p is passed to the model in an array to conform to Pyomo's expectations.) After all constraints are created, the objective function is defined, and the optimization problem is solved.

Our framework also allows for adaptive meshing, warm starting, and obstacle avoidance to be optionally included after the first optimization iteration. A multi-iteration approach may be leveraged for several reasons. Firstly, the solver often has difficulty converging if obstacles are included in the first iteration. We employ the Obstacles class to introduce obstructions after the first iteration. We have found that a warm start trajectory encourages problem convergence when obstacle avoidance is considered. (See Sec. III.D for more details.) An adaptive mesh may also be useful in problems where state derivatives become large in certain portions of the horizon. After each optimization problem iteration, the adaptive mesh scheme discussed in Sec. III.C is optionally called to refine the mesh using derivative information from the previous iteration.

Through this framework, it is simple to change out vehicle properties and constraints. One can easily switch out aerodynamic data in the AeroBehavior class to test the behavior of a new vehicle. Additionally, Listings 2 and 3 illustrate how one can easily add constraints to one or more phases during the development of an optimization problem without the hindrance of explicity repeating the constraint details for each phase. One can also easily remove constraints from phases to test the root cause of a problem not converging. This code structure thus encourages fast and straight-forward testing of multi-phase optimization problems.

Listing 2 Creating Phase-Dependent Constraints in Pyomo

```
    def _vdot(self, m, tau, p):
        (h, v, gamma, psi, theta, phi, alpha, beta, sigma, mass, thrust,
         dhdtau, dvdtau, dgammadtau, dpsidtau, dthetadtau, dphidtau,
         dalphadtau, dbetadtau, dsigmadtau, dmassdtau, dthrustdtau,
         alphadot, betadot, sigmadot, thrustdot, tf, Sref) = getPhaseVariables(m, p, tau)
        r = h + m.Re
        g = m.mu / r ** 2
        L, D, S = self.aeroLiftDrag(m, p, tau, alpha, beta, v, h, Sref)
        return dvdtau == tf * ((thrust * cos(alpha) - D) / mass - g * sin(gamma))
```

Listing 3 Encoding Model Constraints with Pyomo

```
boostGlide.model.vdot1Con = Constraint(boostGlide.model.tau1, p=[1], rule=vehicleLargeOpt._vdot)
```

D. Velocity Maximization

In this study we seek to devise a boost-glide trajectory in which the the final velocity of the glide body is maximized. We also seek to ensure that the glide body's final latitude, longitude, and altitude align with the desired positions $\phi_f^{(6)}$, $\theta_f^{(6)}$, and $h_f^{(6)}$, respectively. The objective function to minimize is:

$$J = -v_{N_p-1}^{(6)} \tag{22}$$

No-fly zone obstacles are included along the path in the form of two-dimensional ellipses in longitude and latitude:

$$\frac{(\phi_n^{(p)} - c_y)^2}{(\frac{b}{2})^2} + \frac{(\theta_n^{(p)} - c_x)^2}{(\frac{a}{2})^2} \geq, 1 \quad n = 0, \ldots, N_p - 1, \quad p = 1, \ldots, 6 \tag{23}$$

where (c_x, c_y) is the ellipse center, b is the diameter of the ellipse horizontal axis, and a is the diameter of the vertical axis. The optimization problem is subject to the following phase-specific constraints:

Dynamics:	(7)-(12), $p = 1, \ldots, 6$
Boundary Conditions:	(1e), $p = 1, \ldots, 5$
No Sidelip in Boost Phases:	$\beta_n^{(p)} = 0, \quad n = 0, \ldots, N_p - 1, \quad p = 1, \ldots 5$
Booster Thrust Profiles	(13)
Fixed-Time Phases:	$t_f^{(p)} = t_{\text{fixed}}^{(p)} \quad p = 1, 2, 3$
Zero Thrust Phases:	$T_n^{(p)} = 0, \quad n = 0, \ldots, N_p - 1, \quad p = 2, 4, 5, 6$
Booster Mass Profiles:	$m_0^{(p)} = m_{\text{init}}^{(p)}$
	$\dot{m}_n^{(p)} = -\left(\frac{dm}{dt}\right)^{(p)}, \quad n = 0, \ldots, N_p - 1, \quad p = 1, 3$
Fixed-Mass Phases:	$m^{(p)} = m_{\text{fixed}}^{(p)} \quad p = 2, 4, 5, 6$
Ballistic Dynamics:	$\alpha_n^{(p)} = 0, \quad n = 0, \ldots, N_p - 1, \quad p = 4, 5$
No-fly Zones:	(23)
Final Altitude Constraints:	$h_{N_p-1}^{(p)} = h_f^{(p)}, \quad p = 4, 5, 6$
Final Latitude and Longitude:	$\theta_{N_p-1}^{(6)} = \theta_f^{(6)}, \quad \phi_{N_p-1}^{(6)} = \phi_f^{(6)}$

The problem is solved using only open-source software: Python 3.7.3, Ipopt 3.12.3, and Pyomo 5.6.6.

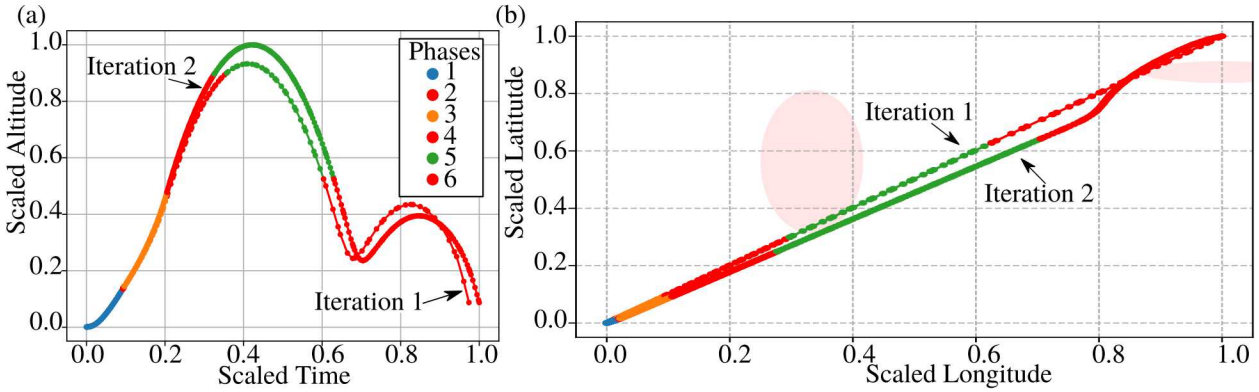

Fig. 3 (a) Optimal altitude trajectories devised over two iterations to maximize the final glide body velocity. Notice that the final altitude of Phase Four is constant between Iterations One and Two, but further work would be required to guarantee that this phase's final altitude constraint corresponds to apogee. The final glide body position corresponds to a desired latitude, longitude, and altitude per the problem's final condition equality constraints. (b) The trajectories' ground tracks, with obstacles shown as pink ellipses. In the first iteration obstacle avoidance is not included. The first iteration results are used to devise a warm start trajectory for the second iteration. The warm start aids in convergence to an optimal trajectory that successfully avoids obstacles.

1. Results

In the first iteration of our two-iteration approach, a coarse mesh is employed and obstacle constraints are not enforced. An initial guess of zero is used for nearly all optimization variables, although better guesses can be used, e.g., a linear profile between initial and final states. Thus, the primary goal of the first iteration is to quickly and approximately satisfy the differential state constraints and boundary constraints of the problem. The full solution from the first iteration, i.e., all Pyomo `Vars`, `DerivativeVars` and `ContinuousSets`, is provided to the second iteration as a warm-start as described in Section III.D. During the second iteration a finer mesh is employed to obtain a more accurate algebraic representation of the differential state constraints. Additionally, the obstacle constraints are enforced during this second iteration. The adaptive mesh scheme of Section III.C was run after the second iteration. However, the influence of the adaptive mesh on this problem is negligible.

During the first iteration (with obstacles inactive) a coarse mesh of 10 finite elements and 3 collocation points per finite element is employed for each phase, based on the orthogonal collocation scheme discussed in Section III.A. During the second iteration (with obstacles active) the mesh is made finer with 15 finite elements and 7 collocation points per finite element. As is common in optimization variables, we found that scaling proved essential to encourage numerical stability and convergence. We chose multipliers for each variable such that each optimization variable ranges from approximately 0 to 10 in the optimal trajectory. The first iteration solves in 11.4 seconds on a Intel(R) Xeon(R) W-2125 CPU. The second iteration solves in 38.2 seconds. A further breakdown of the total computation time into building the model and solving for the optimal trajectory is included in Table 1. The problem with obstacle avoidance has 12,405 variables, 11,238 equality constraints and 639 inequality constraints. The resulting trajectories from the two iterations of our approach are shows in Fig. 3 and Fig. 4. (Due to the Official Use Only nature of portions of these trajectories, the states and inputs have been nondimensionalized.)

Table 1 Model Build and Solve Times. The build time in Iteration 2 includes warm starting, and comprises 47 ms of the total build time.

	Build (s)	Solve (s)	Total (s)
Iteration 1	2.1	9.3	11.4
Iteration 2	9.0	30.2	38.2

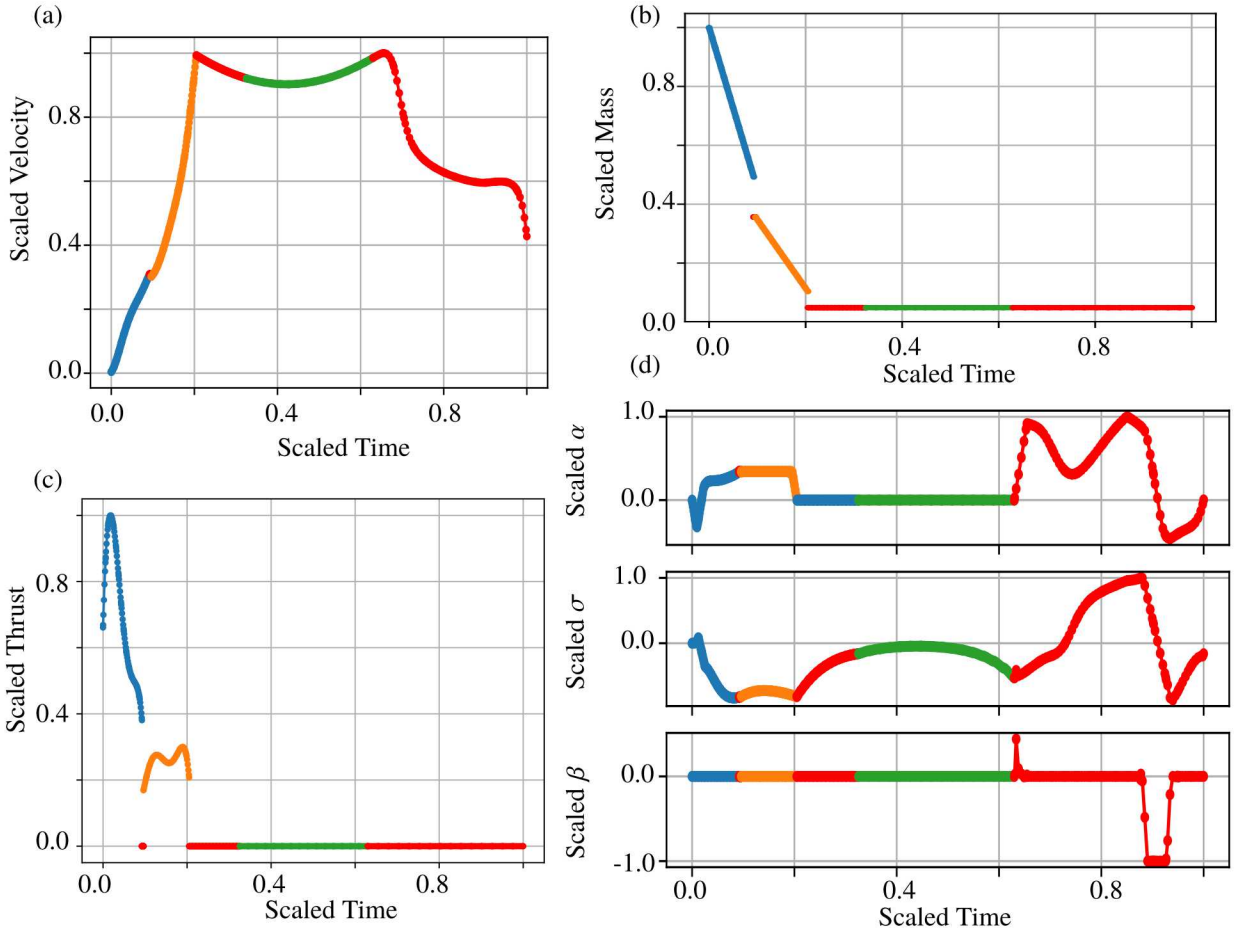

Fig. 4 Optimal trajectories of the second iteration with obstacle avoidance and a final speed objective. (a) Speed trajectory, with the intial and final conditions of each phase stitched together via boundary condition constraints. (b) Mass profile. Discontinuities in the profile correspond to when the first and second stage remaining masses are dropped. (c) Thrust profiles, with non-zero thrusts when the two booster stages are active. (d) Optimal attitude input commands.

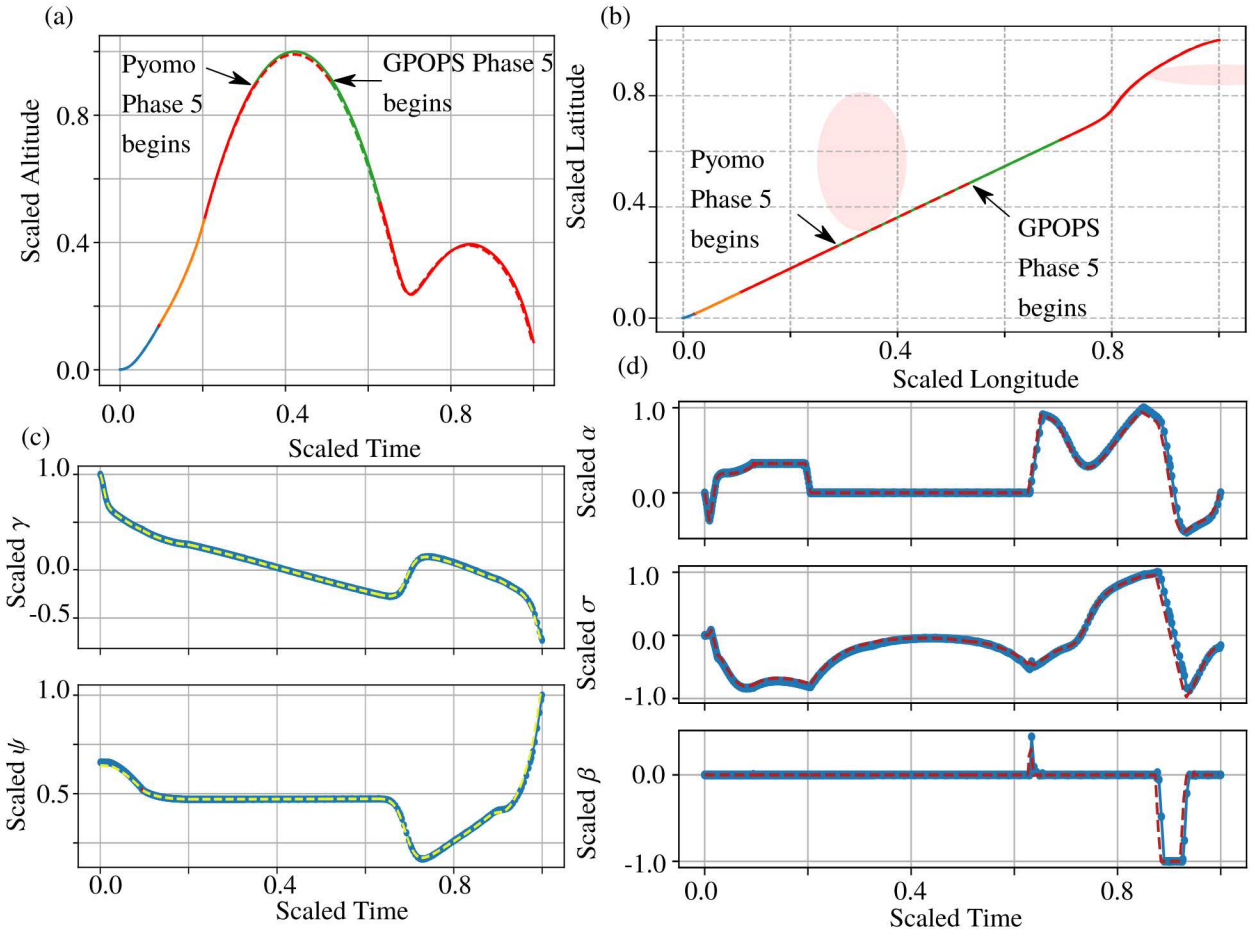

Fig. 5 Pyomo and GPOPS-II trajectories, with the GPOPS trajectories shown by dotted lines.

2. Validation with GPOPS-II

We validate the results from Pyomo with the GPOPS-II software, executing the problem solution in an identical manner: obstacle avoidance is deferred until the second iteration in GPOPS, and the results from the first GPOPS iteration are provided to the second iteration as a warm-start. Identical scaling factors for all state and control variables are used between Pyomo and GPOPS. Additionally, identical iteration-dependent meshes are used between Pyomo and GPOPS. The GPOPS mesh error tolerance was set to 1e-3, which was always satisfied with the predefined meshes (that is, GPOPS did not require any mesh sub-iterations). Fig. 5(a)-(d) show that Pyomo and GPOPS-II produce nearly identical trajectories. The main difference is highlighted in Fig. 5(a) and (b). The end of Phase 5 is triggered by the vehicle reaching an altitude of h_f^6. This event occurs two times, once before apogee, and once after. As a result, Phase 5 terminates at a scaled time, t_{scale} = 0.52 in the Pyomo trajectory and at t_{scale} = 0.33 in the GPOPS results. This phase difference makes minimal impact: the final velocities of the two approaches differ by 0.0132%.

Table 2 compares run times for the Pyomo and GPOPS-II problems. While both GPOPS and Pyomo leverage IPOPT, the differences in computation times may be due to differences in software versions: Pyomo is using IPOPT 3.12.4, whereas GPOPS is using IPOPT 3.11.0. Another source contributing to the computation time discrepancy may be the way first and second order derivatives, used by IPOPT during optimization iterations, are formed. As stated earlier, Pyomo supplies IPOPT with formatted model equations, from which derivatives are created through the compiled automatic differentiation library ASL. GPOPS, on the other hand, formulates these first and second order derivatives[‡] directly using finite differences [23] and then hands these over to IPOPT.

[‡]An alternative to finite differencing is to use the open-source ADiGator software [22] which is compatible with GPOPS-II.

Table 2 Warm Start Problem and Full Optimization Computation Times

	Iteration 1 (s)	Iteration 2 (s)	Total (s)
Pyomo	11.4	38.2	49.6
GPOPS-II	60.2	170.3	230.5

V. Planar Motion Toy Problem

This toy problem shows how our object-oriented framework can be applied to a different problem, and allows us to release our supporting code. We consider a vehicle that can move within a 2D plane, whose dynamics are based on [24] and [25]. The fifth-order dynamics are given by:

$$\dot{x} = v\cos\theta_v \tag{25}$$

$$\dot{y} = v\sin\theta_v \tag{26}$$

$$\dot{\theta}_v = \omega \tag{27}$$

$$m\dot{v} = F_{aero} + F_{control} \tag{28}$$

$$I\dot{\omega} = T_{aero} + T_{control} \tag{29}$$

where x, y, θ_v, and ω are x and y positions, heading angle, and angular velocity, respectively. F_{aero}, and T_{aero} are the environmental forces and torques respectively. $F_{control}$ and $T_{control}$ are the control input force and torque, respectively. The state is $x \triangleq [x \;\; y \;\; \theta_v \;\; v \;\; \omega]^T \in \mathbb{R}^5$ and the input is $u \triangleq [F_{control} \;\; T_{control}]^T \in \mathbb{R}^2$. The environmental forces and torques are computed as:

$$\begin{aligned} F_{aero} &= C_\theta \omega + C_v v \\ T_{aero} &= C_{\theta\omega}\theta_v + C_{v\omega} v \end{aligned} \tag{30}$$

The state-dependent coefficients are based on the following table:

Table 3 Planar Example Aerodynamic Coefficients

Velocity	C_θ	$C_{\theta\omega}$	C_v	$C_{v\omega}$
$0 \le v < 10$	0.1	0.2	-0.1	-0.3
$10 \le v < 20$	0.1	0.2	-0.2	-0.2
$20 \le v$	0.1	0.2	-0.3	-0.1

The goal is to solve for the optimal time history of control values $(F_{control}, T_{control})$ to move the vehicle from point (x,y) = (0 m,0 m) to point (4 m,8 m) in the minimum time. We include the constraint that $(F_{control}, T_{control})$ be bounded by the value ±5 N and ±5 N-m, respectively. An obstacle is present at (3 m, 1.5 m). It is also desired to guarantee that the trajectory will pass through the point (3.5 m, 2.5 m).

A. Results

To enforce that the trajectory passes through (3.5 m, 2.5 m), we formulate this problem as a two-phase problem. The initial conditions are $x_0^{(1)} = [0\text{ m} \;\; 0\text{ m} \;\; 0\text{ rad} \;\; 0\text{ m/s} \;\; 0\text{ rad/s}]^T$. The mid-trajectory position requirement of this problem is imposed as two final condition constraints on Phase One:

$$x_{N_p}^{(1)} = 3.5, \quad y_{N_p}^{(1)} = 2.0 \tag{31}$$

We solve the problem with m = 10 kg and I = 40 kg-m^2. To move the vehicle to the desired position as quickly as possible, the cost function (to be minimized) is:

$$J = -t_f^{(1)} - t_f^{(2)} \tag{32}$$

We employ the exact same object-oriented structure that was used in the boost-phase problem (Fig. (2)). Again, through this approach, dynamic and obstacle avoidance constraints are applied to both phases without explicit code repetition. As in the boost-glide problem, phase-specific initial, final, and boundary conditions are also included. The code is available at [18]

Since Pyomo does not allow state-dependent conditional constraints, we leverage smoothing functions to calculate C_v and $C_{w\omega}$ throughout the trajectory. We leverage the approximation of a step function using [26]. To summarize, to transition from $\epsilon \in [\lambda, \tau]$ to $\epsilon \in [\tau, \mathcal{V}]$, a piecewise continuous function :

$$\rho(\epsilon) = \begin{cases} \rho_I(\epsilon), & \lambda \le \epsilon \le \tau \\ \rho_{II}(\epsilon), & \tau \le \epsilon \le \mathcal{V} \end{cases} \tag{33}$$

can be approximated as:

$$\rho(\epsilon) \approx \rho_I(\epsilon) \times \frac{1}{2}\left[\frac{(\epsilon-\lambda)}{[(\epsilon-\lambda)^2+\eta^2]^{\frac{1}{2}}} + \frac{(\tau-\epsilon)}{[(\tau-\epsilon)^2+\eta^2]^{\frac{1}{2}}}\right] + \rho_{II}(\epsilon) \times \frac{1}{2}\left[\frac{(\epsilon-\tau)}{[(\epsilon-\tau)^2+\eta^2]^{\frac{1}{2}}} + \frac{(\mathcal{V}-\epsilon)}{[(\mathcal{V}-\epsilon)^2+\eta^2]^{\frac{1}{2}}}\right] \tag{34}$$

We extend this formulation to transition between the three velocity regions specified in Table 3. The results of this case study are shown in Fig. 6(c)-(e). The trajectory performs hand-off from Phase One to Phase Two at the required mid-trajectory position, and achieves obstacle avoidance. Figures 6 (d) and (e) demonstrate that this minimum-time trajectory results in bang-bang control.

For comparison, we also run this problem without the mid-trajectory position constraint, as shown in Fig. 6(b). Interestingly, while hand-off from Phase One to Phase Two could occur at any time, the solver selects for the second phase to only last 0.02 s. A comparison of the trajectory times is shown in Table 4. These results show that requiring the trajectory to pass through (3.5 m, 2.5 m) adds 0.72 s to the vehicle's travel time.

Table 4 Trajectory Times with and without (3.5 m, 2.5 m) Trajectory Requirement

	Phase 1 (s)	Phase 2 (s)	Total (s)
No Mid-Trajectory Requirement	6.52	0.02	6.54
No Mid-Trajectory Requirement	4.94	2.32	7.26

VI. Control System Gain Optimization with Stability Margin Constraints

We now demonstrate the flexibility of our framework in a single-phase dynamic optimization problem involving complex valued constraints. Unlike the previous examples, complex valued constraints are handled through the creation, manipulation, and lambdification of symbolic functions. These symbolic functions are then converted into Pyomo Expressions involving the Pyomo `Vars`. Specifically, a control system optimization subject to gain and phase stability margin constraints is performed. The overall goal is to maximize performance while ensuring a sufficient margin of closed-loop stability, which are typically competing objectives. Making this performance trade-off can be a non-trivial task, often resulting in significant trial-and-error evaluation. Using our framework, we demonstrate Pyomo's ability to solve this problem in seconds. By employing an object-oriented framework, various controller and plant structures can be investigated easily without needing to modify large amounts of the core code.

A. Problem Formulation

The following problem is adapted from [27]. Consider the following controller and plant transfer functions, $C(s)$ and $G(s)$, respectively, of a standard feedback loop

$$C(s) = \frac{K_p s + K_i}{s}, \qquad G(s) = \frac{1}{(s+2)(s^2+2s+4)}$$

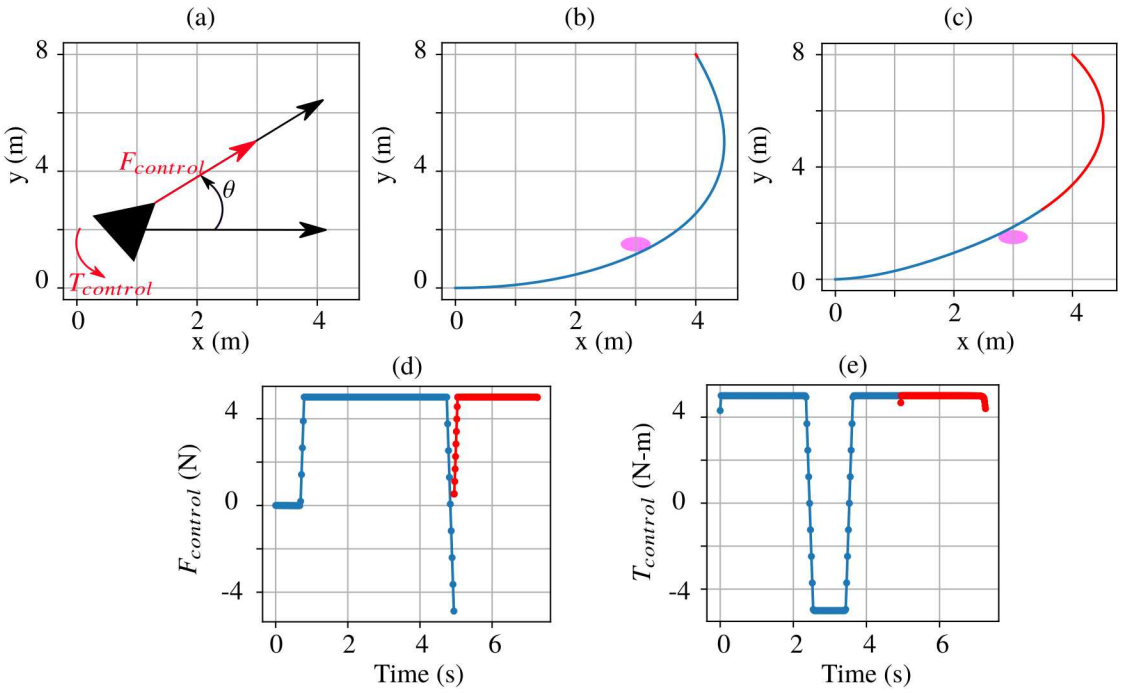

Fig. 6 Planar problem results. (a) Coordinate system employed, with the positive direction of $F_{control}$ and $T_{control}$ indicated in red. (b) Optimal trajectory with obstacle avoidance, and no requirement that the trajectory pass through (3.5 m, 2.5 m). Phase One is in blue and (the very short) Phase Two is in red. (c) Trajectory with both obstacle avoidance and the (3.5 m, 2.5 m) position requirement. (d) Optimal $F_{control}$ inputs to produce the results in (c). (e) Optimal $T_{control}$ inputs to produce the results in (c).

Here $s = \sigma + j\omega$ is the standard complex Laplace variable. The associated negative feedback loop is shown in Fig. 7.

This setup represents a proportional-integral (PI) type controller influencing the input to a third-order plant. The goal is to design the controller gains K_p and K_i such that the total tracking error response along some fixed horizon, $J = \int_{t=0}^{T} (r(t) - y(t))^2 \, dt$, is minimized, while preserving gain and phase stability margins [27].

To perform the required frequency domain analysis, we start by defining the forward path transfer function $L(j\omega) = G(j\omega)C(j\omega)$, where the complex argument is evaluated at $s = j\omega$. With some simple manipulation involving complex conjugates, this forward path transfer function can be expressed as $L(j\omega) = a(\omega; K_p, K_i) + jb(\omega; K_p, K_i)$, where the complex functions, a and b, are explicit functions of the controller gains, K_p and K_i. The phase of $L(j\omega)$ describes the angular distance by which the output will lag a sinusoidal input and is given by:

$$\angle L(j\omega) = \frac{180}{\pi} \tan_2^{-1} \frac{b(\omega; K_p, K_i)}{a(\omega; K_p, K_i)} \tag{35}$$

The so-called *phase-crossover* frequency, ω_{pc}, is the frequency at which phase becomes $-180°$, which is equivalent to the condition $0 = b(\omega_{pc}; K_p, K_i)$. The gain of $L(j\omega)$ is the multiplicative factor by which a sinusoidal input to L is increased, given by:

$$|L(j\omega)| = \sqrt{a(\omega; K_p, K_i)^2 + b(\omega; K_p, K_i)^2} \tag{36}$$

The so-called *gain-crossover* frequency, ω_{gc}, is the frequency at which the gain becomes unity and is equivalent to the condition $1 = a(\omega_{gc}; K_p, K_i)^2 + b(\omega_{gc}; K_p, K_i)^2$. Finally the gain margin, g_m, and phase margin, p_m, describe the distance the system is away from instability, defined as:

$$g_m = \frac{1}{|L(j\omega_{pc})|}, \qquad p_m = 180° + \angle L(j\omega_{gc}) \tag{37}$$

This entire problem is concisely described as the following mathematical programming problem, where gain and phase

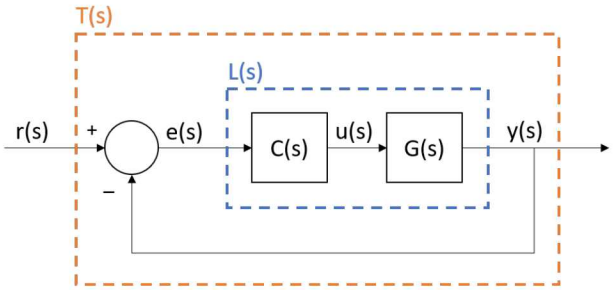

Fig. 7 Standard feedback loop for control system gain optimization problem.

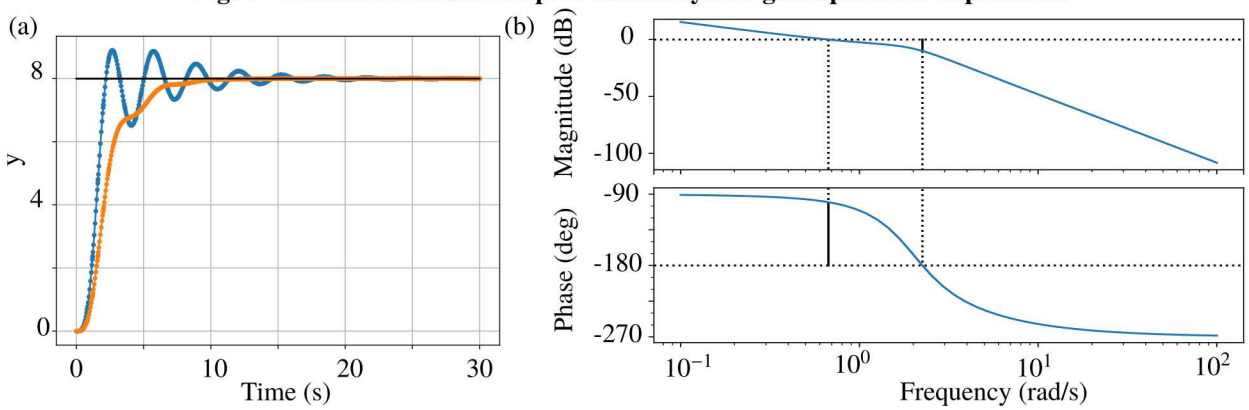

Fig. 8 Control system gain optimization problem results. (a) Responses with stability constraints inactive (blue), $J = 80.3$**,** $g_m = 3.75dB$**,** $p_m = 29.2°$ **and stability constraints active (orange),** $J = 108.8$**,** $g_m = 10dB$**,** $p_m = 80°$**. (b) Bode plot with stability constraints active. The plots show** $g_m = 10dB$ **(at 2.25 rad/s) and** $p_m = 79.9°$ **(at 0.67 rad/s)**

margin are constrained to be at least 10 dB and 80°, respectively.

$$\underset{K_p,K_i}{\text{minimize}} \quad J = \int_{t=0}^{T} (r(t) - y(t))^2 \, dt \tag{38a}$$

$$\text{subject to} \quad \text{time-domain dynamics derived from } Y(s) = G(s)U(s) \tag{38b}$$

$$\text{state and control inequalities} \tag{38c}$$

$$0 = b(\omega_{pc}; K_p, K_i) \quad \text{(phase crossover constraint)} \tag{38d}$$

$$1 = a(\omega_{gc}; K_p, K_i)^2 + b(\omega_{gc}; K_p, K_i)^2 \quad \text{(gain crossover constraint)} \tag{38e}$$

$$\frac{1}{|L(j\omega_{pc})|} \geq 3.1623 \quad \text{(gain margin constraint)} \tag{38f}$$

$$180° + \angle L(j\omega_{gc}) \geq 80° \quad \text{(phase margin constraint)} \tag{38g}$$

Python's `sympy` toolbox was leveraged in order to program the constraints above. Specifically, Pyomo `Vars` are passed into a `sympy lambda` function. This is accomplished by first constructing a symbolic function of *symbolic versions* of the Pyomo `Vars`. The function is then lambdified with the symbolic versions of the Pyomo `Vars` as arguments. This lambdified function is then provided the Pyomo `Vars` for conversion into a Pyomo Expression. The benefit of this approach is on-the-fly construction of the constraints in (38) from basic definitions of the controller, $C(s)$, and plant, $G(s)$.

This problem requires only the core component of our object-oriented framework. To solve this problem, we instantiate an instance of the GainScheduler class, which serves the same function as the Vehicle3DOF class in Fig. 2. Within the main function we can easily add and remove constraints during problem development. Full implementation details and associated code are provided at [18]. The results are shown in Fig. 8.

VII. Discussion

While we have demonstrated the ability for our pseudospectral optimization approach to respect a variety of constraints, other constraints could be considered for the boost-glide problem. Introduction of algebraic variables to constrain parameters such as heating may be of interest in glide problems. Additionally, it is of interest to refine the Phase Five final condition to guarantee that a final apogee limit is guaranteed.

Our comparison between equivalent boost-glide problems in our framework and GPOPS-II shows that we can achieve nearly identical results faster, while leveraging only open-source software. A main difference between our framework's capabilities compared to those of GPOPS-II is that currently our adaptive mesh approach can only move finite elements, whereas GPOPS-II can add and move them. As we mentioned in Sec. IV.D.2, the GPOPS-II boost-glide problem does not require any mesh sub-iterations. In other problems, this may prove critical in achieving an accurate solution. In the future we are interested to include the functionality in our framework to increase the number of finite elements based on current numerical accuracy.

In this paper we demonstrate the versatility of our object-oriented framework, coupled with only open-source software. By employing multi-phase optimization we devise full trajectories in our example scenarios that directly support the objective of interest. The object-oriented framework is highly useful for our multi-phase examples in order to minimize code repetition and to experiment easily with including constraints. Our optimization framework can be leveraged in a variety of applications, from high-level flight mission planning, to low-level controller design.

References

[1] Hartjes, S., van Hellenberg Hubar, M. E., and Visser, H. G., "Multiple-phase trajectory optimization for formation flight in civil aviation," *CEAS Aeronautical Journal*, Vol. 10, No. 2, 2019, pp. 453–462.

[2] Wu, Y., Li, L., Su, X., and Cui, J., "Multi-phase trajectory optimization for an aerial-aquatic vehicle considering the influence of navigation error," *Engineering Applications of Artificial Intelligence*, Vol. 89, 2020, p. 103404.

[3] Riehl, J. P., Sjauw, W. K., Falck, R. D., and Paris, S. W., "Trajectory Optimization: OTIS 4," 2010.

[4] Williams, J., Falck, R. D., and Beekman, I. B., "Application of Modern Fortran to Spacecraft Trajectory Design and Optimization," *2018 Space Flight Mechanics Meeting*, 2018, p. 1451.

[5] Betts, J., "Sparse optimization suite, SOS, User's guide, release 2015.11," , 2016.

[6] Pradeep, P., Park, S. G., and Wei, P., "Trajectory optimization of multirotor agricultural uavs," *2018 IEEE Aerospace Conference*, IEEE, 2018, pp. 1–7.

[7] Benito, J., and Johnson, B., "Trajectory Optimization for a Mars Ascent Vehicle," *AIAA/AAS Astrodynamics Specialist Conference*, 2016, p. 5441.

[8] Hart, W. E., Laird, C. D., Watson, J.-P., Woodruff, D. L., Hackebeil, G. A., Nicholson, B. L., and Siirola, J. D., *Pyomo-optimization modeling in python*, Vol. 67, Springer, 2017.

[9] Nicholson, B., Siirola, J. D., Watson, J.-P., Zavala, V. M., and Biegler, L. T., "pyomo. dae: a modeling and automatic discretization framework for optimization with differential and algebraic equations," *Mathematical Programming Computation*, Vol. 10, No. 2, 2018, pp. 187–223.

[10] Biegler, L. T., and Zavala, V. M., "Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization," *Computers & Chemical Engineering*, Vol. 33, No. 3, 2009, pp. 575–582.

[11] Forbes-Spyratos, S. O., Kearney, M. P., Smart, M. K., and Jahn, I. H., "Trajectory design of a rocket–scramjet–rocket multistage launch system," *Journal of Spacecraft and Rockets*, Vol. 56, No. 1, 2019, pp. 53–67.

[12] Mathavaraj, S., Pandiyan, R., and Padhi, R., "Constrained optimal multi-phase lunar landing trajectory with minimum fuel consumption," *Advances in Space Research*, Vol. 60, No. 11, 2017, pp. 2477–2490.

[13] Patterson, M. A., and Rao, A. V., "A General-Purpose MATLAB Software for Solving Multiple-Phase Optimal Control Problems Version 2.3," 2016.

[14] Biegler, L. T., *Nonlinear programming: concepts, algorithms, and applications to chemical processes*, Vol. 10, Siam, 2010.

[15] Kelly, M., "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, Vol. 59, No. 4, 2017, pp. 849–904.

[16] Garg, D., "Advances in global pseudospectral methods for optimal control," Ph.D. thesis, University of Florida USA, 2011.

[17] Patterson, M. A., and Rao, A. V., "GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming," *ACM Trans. Math. Softw.*, Vol. 41, 2014, pp. 1:1–1:37.

[18] Schlossman, R., and Williams, K. R., `https://github.com/sandialabs/repo_will_be_available_in_final_submission`, 2020.

[19] Hood, L., Bennett, G., and Parish, J. J., "Model Fidelity Studies for Rapid Trajectory Optimization," *AIAA Scitech 2019 Forum*, 2019, p. 0430.

[20] Grant, M., "A Simple Hypersonic Trajectory Formulation," *Sandia Internal Technical Note*, January 2018.

[21] McClinton, C., "X-43-Scramjet power breaks the hypersonic barrier: Dryden lectureship in research for 2006," *44th AIAA aerospace sciences meeting and exhibit*, 2006, p. 1.

[22] Weinstein, M. J., and Rao, A. V., "Algorithm 984: ADiGator, a toolbox for the algorithmic differentiation of mathematical functions in MATLAB using source transformation via operator overloading," *ACM Transactions on Mathematical Software (TOMS)*, Vol. 44, No. 2, 2017, pp. 1–25.

[23] Patterson, M. A., and Rao, A. V., "Exploiting sparsity in direct collocation pseudospectral methods for solving optimal control problems," *Journal of Spacecraft and Rockets*, Vol. 49, No. 2, 2012, pp. 354–377.

[24] Parish, J., "Simple Example Problem for Sensitivity Analysis," *Sandia Internal Technical Note*, December 2018.

[25] Weitz, L. A., *Decentralized, cooperative control of multivehicle systems: Design and stability analysis*, Texas A&M University, 2009.

[26] Arora, N., "Nonlinear Programming for Data Reconciliation and Parameter Estimation," Ph.D. thesis, PhD Dissertation, Carnegie Mellon, Pittsburgh, 2003.

[27] Åström, K. J., and Murray, R. M., *Feedback systems: an introduction for scientists and engineers*, Princeton university press, 2010.