DYNAMIC OPTIMIZATION IN THE JULIA PROGRAMMING LANGUAGE

KUI XIE
Supervisor: Prof. Eric Kerrigan
Co-Supervisor: Mr. Eduardo Vila

A Thesis submitted in fulfilment of requirements for the degree of
Master of Science in Control and Optimization
of Imperial College London

Department of Electrical and Electronic Engineering
Imperial College London
October 3, 2022

Abstract

Direct collocation methods could transform a dynamic optimization problem (DOP) into a nonlinear programming problem (NLP). Followed by mesh refinement methods, the dynamic constraint residuals integrated along the whole solution trajectory would be decreased to the desired level.

Additionally, based on a nonlinear programming solver, the package, called DCMR, for direct collocation with mesh refinement would be developed in Julia. Meanwhile, during implementation, the advantages of Julia would be fully used, such as multiple dispatch.

Lastly, to lower the numerical computation burden, novel strategies are proposed to provide reasonable initial guesses and convergence tolerance settings in each iteration of mesh refinement. As shown in the results, the proposed schemes would achieve more than a 50% reduction in computation time.

Acknowledgment

To begin with, it is my great honour that the chance could be offered to explore cutting-edge research at this globally famous university, Imperial College London.

Meanwhile, I would like to thank the project supervisor, Prof. Eric Kerrigan. It is his instructive advice that guides me in this exciting research field. Then, I would also thank my second supervisor, Mr Eduardo M Vila, for his constant assistance and encouragement and for helping me learn the brand-new area from scratch. Thank all professors and lecturers at Imperial College London for their teaching, helps me build a solid foundation for this project.

Particular gratitude also goes to my dear friends, Haoyu Wei and Yuhao Yang, who accompany me through the most challenging time.

Finally, my deepest gratitude is expressed to my family for their continuous support and care.

Contents

List of Figures

List of Tables

Abbreviations

DOP: Dynamic Optimization Problem

MR: Mesh Refinement

NLP: Nonlinear Programming

TNIR: Time Normalized Integrated Residuals

TRP: Trapezoidal method

HSS: Hermite–Simpson (Separated) Collocation method

HSC: Hermite–Simpson (Compressed) Collocation method

TRPuC: Trapezoidal Collocation Method with Continuous Input

TRPuD: Trapezoidal Collocation Method with Discontinuous Input

HSSuC: Hermite–Simpson (Separated) Collocation Method with Continuous Input

HSSuD: Hermite-Simpson (Separated) Collocation Method with Discontinuous Input

IRNS: Integrated Residual Norm Squared

MIRNS: Mesn Integrated Residual Norm Squared

ODE: Ordinary Differential Equation

DAE: Differential Algebraic Equation

Chapter 1

Introduction

1.1 Literature Review

MOST of the optimal control and estimation problems could be transformed into a dynamic optimization problem (DOP). In the field of aerospace and robotics, these problems could also be called trajectory optimization problems, which are concerned with minimizing a functional $J(\cdot)$, such as minimizing time, energy, and cost [1]. Different from parameter optimization about minimizing some function $f(\cdot)$, trajectory optimization is more complicated due to the larger space of functions.

Then, there are many methods for trajectory optimization, such that trade-offs need to be found for specific problems based on a good understanding of different methods [1]. For example, in contrast to direct methods, indirect methods tend to produce a more accurate solution, at the expense of difficult construction and complicated solution process Since direct methods tend to only sole a discrete approximation of the optimal problems, while indirect methods solve the problem with necessary and sufficient conditions directly. Hence, due to simple implementation of direct methods, direct methods are usually chosen to solve practical DOPs [2]. Additionally, when DOP is solved with collocation grids initially, appropriate mesh refinement needs to be implemented to find a new mesh or increase the polynomial order inside the interval [3–5]. This iterative strategy aims at obtaining the most accurate solutions with the least computation effort: start solving coarse

mesh and get an inaccurate solution, then solutions with subsequent mesh are more accurate.

Additionally, there are various software programs for trajectory optimization problems written in MATLAB and C++. Nowadays, as an easy and fast scientific computing language, the innovation of Julia results from its combination of productivity and performance [6, 7]. There are many packages, such as JuMP.jl package [8] for mathematical programming written for forming an entire fast, high-level system for scientific and numerical computing.

1.2 Overview

To begin with, the theoretical background for optimal numerical control such as direct collocation methods, will be introduced in Chapter 2. Then, Julia's characteristics would be briefly shown in Chapter 3 to help the reader better comprehend the implementation in Chapter 4. After solving the DOP cart-pole swing-up problem, results are presented and discussed fully in Chapter 5. Lastly, Chapter 6 would conclude the thesis and tell future work.

Chapter 2

Numerical Optimal Control

THERE are various ways to formulate the dynamic optimization problem [9]. In this thesis, only single-phase continuous-time dynamic optimization problems are considered. Then, temporal discretization and trajectory parametrization for DOPs are introduced in general form firstly, followed by trapezoidal collocation and Hermite-Simpson collocation methods.

2.1 Dynamic Optimization Problem

The dynamic optimization problem could be formulated as the general Bolza form [10],

$$\min_{\substack{x(\cdot),u(\cdot)\\ t_0,t_f,p}} \Phi(x(t_0),x(t_f),t_0,t_f,p)+\int_{t_0}^{t_f} L(x(t),u(t),t,p)\,\mathrm{d}t \tag{2.1}$$

subject to

$$\dot{x}(t)=f(x(t),u(t),t,p),\ \forall t\in\mathcal{T}\ a.e., \tag{2.2}$$

$$g(\dot{x}(t),x(t),u(t),t,p)=0,\ \forall t\in\mathcal{T}\ a.e., \tag{2.3}$$

$$c(\dot{x}(t),x(t),u(t),t,p)\leq 0,\ \forall t\in\mathcal{T}\ a.e., \tag{2.4}$$

$$\psi_E(x(t_0),x(t_f),t_0,t_f,p)=0, \tag{2.5}$$

$$\psi_I(x(t_0),x(t_f),t_0,t_f,p)\leq 0. \tag{2.6}$$

where the continuous state trajectory $x : \mathbb{R} \to \mathbb{R}^{n_x}$; the control functions $u : \mathbb{R} \to \mathbb{R}^{n_u}$ are selected to minimize cost functions and on a time interval $\mathcal{T} := [t_0, t_f] \subset \mathbb{R}$; t_0 and t_f are initial and final values for time t; 'a.e.' denotes 'almost everywhere'; Equation 2.1 denotes the objective function in Bolza form, composed of Mayer term ($\Phi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}$) and Lagrange term ($L : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}$) [11]; The objective would be represented by a single functional J with optimal solution denoted J^*; Equation 2.2 represents the system dynamics in ODE form ($f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_x}$); Equations 2.3 and 2.4 are equality constraint ($g : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_g}$) and inequality path constraints ($c : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_c}$) respectively; Equations 2.5 and 2.6 define the boundary equality constraints ($\psi_E : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_{\psi_E}}$) and boundary inequality constraints ($\psi_I : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_{\psi_I}}$). Equations 2.2 and 2.3 could be referred as dynamic equations.

2.2 Direct Collocation Method

2.2.1 General Formulation

A dynamic optimization problem could be formulated as a nonlinear programming problem by numerical discretization schemes. To begin with, the time domain $\mathcal{T} := [t_0, t_f]$ would be subdivided into K intervals. For the k^{th} interval, $\mathcal{T}_k := [s_k, s_{k+1}]$ and $t_0 = t_1 < t_2 < \cdots < t_N < t_{N+1} = t_f$ distributing along knot points. Within the k^{th} interval, the minor nodes would be added according to the computation schemes.

Then, during each interval k, system dynamics would be depicted by a set of dynamic variables

$$\begin{bmatrix} \boldsymbol{x}^{(k)}(t) \\ \boldsymbol{u}^{(k)}(t) \end{bmatrix} \approx \begin{bmatrix} \tilde{\boldsymbol{x}}^{(k)}(t) \\ \tilde{\boldsymbol{u}}^{(k)}(t) \end{bmatrix} := \begin{bmatrix} \sum_{i=1}^{N^{(k)}} a_{x,i}^{(k)} \beta_{x,i}^{(k)}(t) \\ \sum_{i=1}^{N^{(k)}} a_{u,i}^{(k)} \beta_{u,i}^{(k)}(t) \end{bmatrix} \tag{2.7}$$

composed of the $n_x^{(k)}$ state variables and the $n_u^{(k)}$ control variables separately. The behaviour of the solution between the points would be constructed by C^1 state approximation and C^0 control approximation during the transcription process. The $\beta_{x,i}^{(k)}(\cdot)$ and $\beta_{u,i}^{(k)}(\cdot)$

denote the basis functions and $a_{x,i}^{(k)}$ and $a_{u,i}^{(k)}$ are the corresponding coefficients, known as the amplitude of the basis function for the i^{th} degree of freedom [5].

There are $N^{(k)}$ points on the polynomial function $\tilde{x}^{(k)}$ and the unknown coefficients are denoted by $\chi_i^{(k)}$ as parametrised states. Overall, $\chi := \left[\chi^{(1)}, \chi^{(2)}, \ldots, \chi^{(K)}\right]^T \in \mathbb{R}^{N \times n_x}$, with $N := \sum_{k=1}^{K} N^{(k)}$. Similar parametrization for $\tilde{u}$ could depicted by v.

As for continuous trajectories, there are two ways of forcing the continuity of the state trajectories between intervals. The first method is that the trajectories are subdivided directly by N intervals such that the same decision variables would be applied. The other method is making use of additional continuity constraints:

$$\tilde{x}^{(k)}(t_{k+1}) = \tilde{x}^{(k+1)}(t_{k+1}). \tag{2.8}$$

Typically, as for continuous inputs, they would be implemented as continuous trajectories as for the states. Otherwise, discontinuity of inputs could be allowed and the inputs trajectories would be piecewise continuous [10]. Hence, the solution to the discretized problem is denoted by $\mathcal{Z} := (\chi, v, p, t_0, t_f)$.

With temporal discretization and trajectory parametrization above, we could express the DOP as NLP:

$$\min_{\chi, v, t_0, t_f, p} \Phi(\chi_1^{(1)}, \chi_{N^{(k)}}^{(N)}, t_1, t_f, p) + \sum_{k=1}^{N} \sum_{i=1}^{Q^{(k)}} w_i^{(k)} L(\tilde{x}^{(k)}(q_i^{(k)}), u^{(k)}(q_i^{(k)}), t_1, t_f, p) \mathrm{d}t \tag{2.9}$$

subject to, for all k,

$$\sum_{l=1}^{N^{(k)}} \mathcal{A}_{il}^{(k)} \chi_l^{(k)} + \mathcal{D}_{il}^{(k)} f(\chi_l^{(k)}, v_l^{(k)}, t_0, t_f, p) = 0, \tag{2.10}$$

$$g(\dot{\chi}_i^{(k)}, \chi_i^{(k)}, v_i^{(k)}, t_0, t_f, p) = 0, \tag{2.11}$$

$$c(\dot{\chi}_i^{(k)}, \chi_i^{(k)}, v_i^{(k)}, t_0, t_f, p) \leq 0, \tag{2.12}$$

$$\phi_E(\chi_1^{(1)}, \chi_{N^{(k)}}^{(K)}, t_0, t_f, p) = 0, \tag{2.13}$$

$$\phi_I(\chi_1^{(1)}, \chi_{N^{(k)}}^{(K)}, t_0, t_f, p) \leq 0, \tag{2.14}$$

as well as the necessary continuity constraints 2.8, where $\dot{\chi}_i^{(k} := {}^{\cdot(k)}(d_i^{(k)}) \in \mathbb{R}^{n_x}$, $\mathcal{A}^{(k)} \in \mathbb{R}^{N^{(k)} \times N^{(k)}}$ is a discretization-dependent constant matrix, and the matrix $\mathcal{D}^{(k)} \in \mathbb{R}^{N^{(k)} \times N^{(k)}}$ contains time variables. For the integrals, quadrature is used as approximation, such as Simpson quadrature. The points inside each interval is denoted by $Q^{(k)}$. The scalers, $w_i^{(k)}$, are the interval-dependent quadrature weights.

In the next part, there are two direct collocation methods, Trapezoidal collocation method and Hermite-Simpson collocation method, introduced as follows.

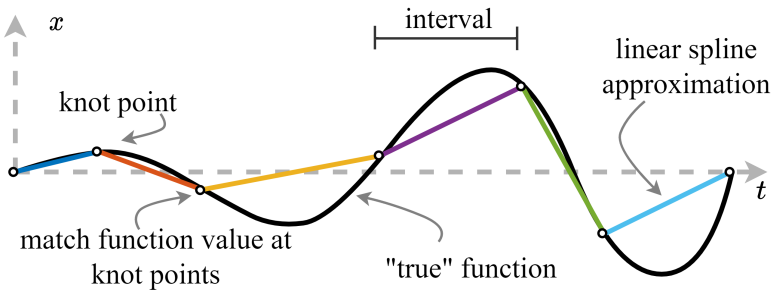

Figure 2.1: Function approximation with a linear spline

2.2.2 Trapezoidal Collocation Method

Trapezoidal collocation aims at translating a continuous-time dynamic optimization problem into a nonlinear program. In this part, only the state constraints in ODE explicit form would be considered. The continuous aspect of the problem is converted into a discretization approximation by the trapezoidal rule for integration. Meanwhile, There is no minor node inside each interval, and only the knot points would be applied for approximation. The transformation process will be introduced in this part.

Trapezoidal Collocation: Integrals

In the dynamic optimization problem, integral expressions often appear in the objective function, but sometimes they are also in constraints. The key point is applying the trapezoidal rule to approximate the integration only with the value of the integrand $w(t_k) = w_k$ at the collocation points t_k along the trajectory. The time interval k is set as $h_k = t_{k+1} - t_k$

The equation could be formulated as

$$\int_{t_0}^{t_F} w(x(\tau), u(\tau), \tau)\ \mathrm{d}\tau \approx \sum_{k=1}^{N} \frac{1}{2} h_k \cdot (w_k + w_{k+1}). \tag{2.15}$$

Trapezoidal Collocation: System Dynamics

The important step of a direct collocation method is that the dynamics of the system are converted to a set of constraints, known as collocation constraints. By trapezoidal rule, the dynamics in integrand form are written and changed into collocation constraints. Then, the integral is approximated by trapezoidal quadrature [5].

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}, \\ \int_{t_k}^{t_{k+1}} \dot{\boldsymbol{x}}\ \mathrm{d}t &= \int_{t_k}^{t_{k+1}} \boldsymbol{f}\ \mathrm{d}t, \\ \boldsymbol{x}_{k+1} - \boldsymbol{x}_k &\approx \frac{1}{2} h_k \cdot (\boldsymbol{f}_k + \boldsymbol{f}_{k+1}), \end{aligned}$$

where $\boldsymbol{x}_k$ is functioned as the decision variable, and the dynamics of the system would be computed as each collocation point to achieve $\boldsymbol{f}_k = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, t_k)$

The above approximation would be used for each pair of collocation points inside the corresponding interval:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k = \frac{1}{2} h_k \cdot (\boldsymbol{f}_k + \boldsymbol{f}_{k+1}), \quad k \in 1, \ldots, N. \tag{2.16}$$

Trapezoidal Collocation: Constraints

From Section 2.2.1, the limits on state, control, path constraints and boundary constraints would be formulated accordingly. Since there is no minor point between each pair of the collocation points, these constraints would only be enforced at the knot points. There are some other methods, such as the orthogonal collocation method, for which the trajectory boundaries should be dealt with carefully [12, 13].

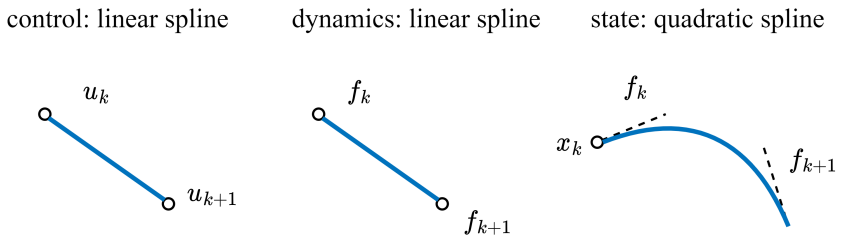

Figure 2.2: The control and state trajectories for trapezoidal collocation

Trapezoidal Collocation: Interpolation

As shown in Figure 2.1, the control trajectory and the system dynamics would be approximated as piecewise linear functions, also called linear splines with trapezoidal rule. The knot points would be applied as a connection between the segments and also coincident with the collocation points.

During each interval, the expression of linear spline could be formulated with the function values and time points at the knot points. As for the control trajectory in a single segment $t \in [t_k, t_{k+1}]$, the equations could be derived as:

$$\boldsymbol{u}(t) \approx \boldsymbol{u}_k + \frac{\tau}{h_k}(\boldsymbol{u}_{k+1} - \boldsymbol{u}_k), \tag{2.17}$$

where $\tau = t - t_k$ and $h_k = t_{k+1} - t_k$ are defined to make mathematics more readable.

Similarly, the system dynamics are also approximated linearly between the knot points with the trapezoidal rule. Then, with trapezoidal quadrature Equation 2.15, we could derive polynomials for states as:

$$\boldsymbol{x}(t) \approx \boldsymbol{x}_k + \boldsymbol{f}_k\tau + \frac{\tau^2}{2h_k}(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k). \tag{2.18}$$

the specific derivation could be found in Appendix A.1.1.

Generally, when the control trajectories are approximated with order n, the state trajectories are formulated with splines of order $n + 1$. Figure 2.2 shows the construction of the control and state trajectories. Equations 2.17 and A.3 would be mainly used for

trapezoidal collocation.

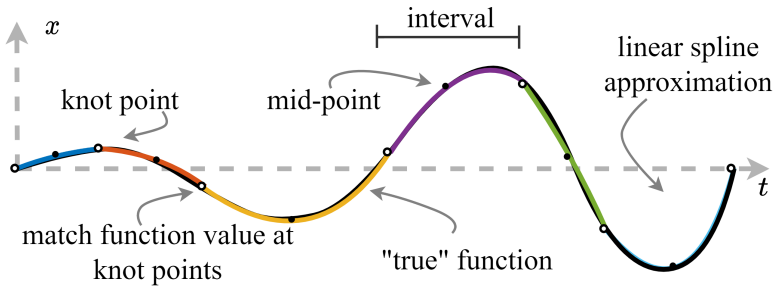

Figure 2.3: Function approximation with a quadratic spline

2.2.3 Hermite-Simpson Collocation Method

With higher-order spline, the Hermite-Simpson collocation method is more accurate than the trapezoidal collocation method with the same number of intervals. The control and system dynamics are approximated with piecewise quadratic functions as shown in Figure 2.3. Additionally, since one minor point is inserted into each interval, the constraints on the minor points should also be considered. Lastly, it could be estimated that piecewise cubic Simpson splines would be constructed for state trajectories.

Hermite-Simpson Collocation: Integrals

The integral terms would appear in both objective function and constraints for the dynamic optimization problem. They are approximated with Simpson quadrature in the Hermite-Simpson Collocation method. In this way, the integrand of the integrals are fitted with piecewise quadratic functions, also called as Simpson's rule for integration. From the derivation in Appendix A.1.2, the approximation could be obtained for the integrals:

$$\int_{t_0}^{t_F} w(\tau)\,\mathrm{d}\tau \approx \sum_{k=1}^{N} \frac{h_k}{6}(w_k + 4w_{k+\frac{1}{2}} + w_{k+1}), \tag{2.19}$$

where the time interval k is denoted as $h_k = t_{k+1} - t_k$.

Hermite-Simpson Collocation: System Dynamics

The dynamics of the system would be rewritten in integral form and converted into collocation constraints by the Simpson rule. For each time interval h_k between two-knot points, the approximation could be conducted with Equation 2.19:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}, \tag{2.20}$$

$$\int_{t_k}^{t_{k+1}} \dot{\boldsymbol{x}} \, \mathrm{d}t = \int_{t_k}^{t_{k+1}} \boldsymbol{f} \, \mathrm{d}t, \tag{2.21}$$

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k = \frac{h_k}{6}(\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} + \boldsymbol{f}_{k+1}). \tag{2.22}$$

Since there is one minor point added between knot points, there is another collocation constraint for enforcing the dynamics on the minor point. It is a key difference between the Trapezoidal collocation method and the Hermite-Simpson collocation method. The minor point state value $\boldsymbol{x}_{k+\frac{1}{2}}$ could be evaluated by evaluating the midpoint of an interplant for the state trajectory:

$$\boldsymbol{x}_{k+\frac{1}{2}} = \frac{1}{2}(\boldsymbol{x}_k + \boldsymbol{x}_{k+1}) + \frac{h_k}{8}(\boldsymbol{f}_k - \boldsymbol{f}_{k+\frac{1}{2}}), \tag{2.23}$$

which would be illustrated further in the next subsection.

Note that when Equation 2.22 and Equation 2.23 are separated, the method is named as Hermite-Simpson (Separated) Method or simply Separated Simpson method, abbreviated as HSS. Both of them could be combined into a single collocation constraint. It is named Hermite-Simpson (Compressed) method or simply Compressed Simpson method and could be abbreviated as HSC [5].

Hermite-Simpson Collocation: Constraints

Special care should be taken in this method compared with the trapezoidal collocation method since not only knot points but also the midpoint should be considered for the state, control, path constraints and boundary constraints. For example, simple boundary

conditions on state and control are shown as follows:

$$\boldsymbol{x} < \boldsymbol{x}^{upp} \rightarrow \begin{matrix} \boldsymbol{x}_k < \boldsymbol{x}^{upp}, \\ \boldsymbol{x}_{k+\frac{1}{2}} < \boldsymbol{x}^{upp}, \end{matrix} \tag{2.24}$$

$$\boldsymbol{u} < \boldsymbol{u}^{upp} \rightarrow \begin{matrix} \boldsymbol{u}_k < \boldsymbol{u}^{upp}, \\ \boldsymbol{u}_{k+\frac{1}{2}} < \boldsymbol{u}^{upp}. \end{matrix} \tag{2.25}$$

From Section 2.2.1, the conditions for state, control, path constraints and boundaries could be achieved respectively.

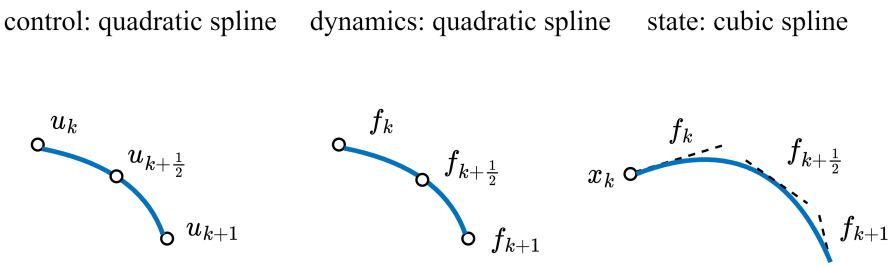

Figure 2.4: The control and state trajectories for Hermite-Simpson collocation

Hermite-Simpson Collocation: Interpolation

As shown in Figure 2.4, the polynomial interplant could be obtained from the collocation points. Similar to trapezoidal collocation, Simpson quadrature would be used by Hermite-Simpson collocation to approximate each segment of the trajectory.

Given the only one minor point located in the middle, the expression for $\boldsymbol{u}(t)$ could be fitted with an appropriate quadratic function:

$$\boldsymbol{u}(t) = \frac{2}{h_k^2}(\tau - \frac{h_k}{2})(\tau - h_k)\boldsymbol{u}_k - \frac{4}{h_k^2}(\tau)(\tau - h_k)\boldsymbol{u}_{k+\frac{1}{2}} + \frac{2}{h_k^2}(\tau)(\tau - \frac{h_k}{2})\boldsymbol{u}_{k+1}, \tag{2.26}$$

where $h_k = t_{k+1} - t_k$, $t_{k+\frac{1}{2}} = \frac{1}{2}(t_k + t_{k+1})$, and $\tau = t - t_k$.

Then, with quadratic expression of dynamics and Simpson quadrature, the expres-

sions for state trajectories could be achieved:

$$\begin{aligned}\boldsymbol{x}(t) = x_k + (\boldsymbol{f}_k)\tau + \frac{1}{2h_k}(-3\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1})(\tau^2) \\ + \frac{1}{3h_k^2}(2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1})\tau^3.\end{aligned} \tag{2.27}$$

The specific derivation of Equations 2.26 and 2.27 could be found in Appendix A.1.3.

2.3 Error Analysis

There are mainly two sources of numerical errors, transcription errors and errors of nonlinear program solutions. Within the direct collocation method, the former is focused on. Ideally, the accuracy of the solution could be measured by comparing $\tilde{x}$ with x and $\dot{\tilde{x}}$ with $\dot{x}$. Unfortunately, the real optimal solution might not be unique and not obtainable. Hence, the error matrices would be selected carefully from the solution for the dynamic optimization problem.

In practice, the residuals $\epsilon(t) \in \mathbb{R}^{n_x+n_g}$ could be defined as:

$$\epsilon(t) := \begin{bmatrix} \dot{\tilde{x}}(t) - f(\tilde{x}, \tilde{u}, t, p) \\ g(\dot{\tilde{x}}, \tilde{x}, \tilde{u}, t, p) \end{bmatrix}, \tag{2.28}$$

which is applicable for any numerical optimal solution. Furthermore, the integration of residual errors along certain intervals could be a suitable standard for the actual accuracy instead of the residuals at a certain location. Based on this, the error in each interval could be evaluated as:

$$\eta^{(k)} := \int_{t_k}^{t_{k+1}} \left\| \epsilon^{(k)}(\tau) \right\|_2^2 \mathrm{d}\tau, \tag{2.29}$$

with $\|\cdot\|_2$ the vector 2-norm squared. Practically, the integral could be evaluated with a standard quadrature method, such as Simpson quadrature. The value $\eta \in \mathbb{R}^N$ is known as the absolute local error [5].

Meanwhile, to estimate the residual error overall, the integrated residual norm

squared (IRNS) could be computed:

$$r(\tilde{x}, \tilde{u}, t_0, t_f, p) := \int_{t_0}^{t_f} \|\epsilon(\tau)\|_2^2 \, \mathrm{d}\tau. \tag{2.30}$$

There are other variants such as mean integrated residual norm squared(MIRNS) error defined as $\frac{r}{\Delta t}$ with $\Delta t := t_f - t_0$. Finally, the computation of residuals paves the way for remeshing the trajectory so that the optimal solution would converge to the continuous dynamics.

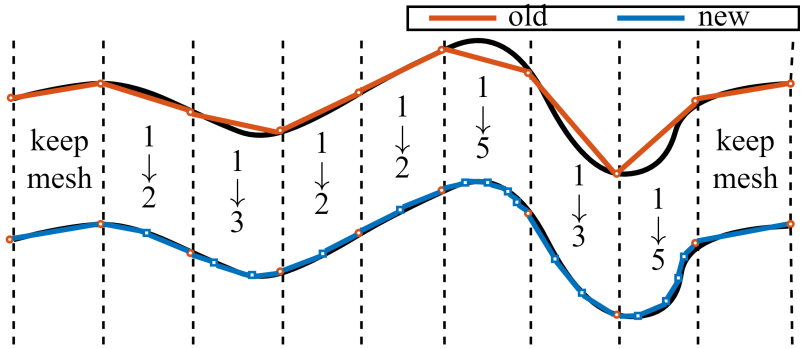

Figure 2.5: Mesh Refinement by subdividing segments

2.4 Mesh Refinement

During mesh refinement, a dynamic optimization problem would be solved on a series of various collocation meshes, also called collocation grids. The selection of discretization over the trajectory is defined as mesh. In the process, the most accurate solution would be achieved with the least computational efforts based on an iterative strategy. It always starts with coarse meshes and a low-order collocation method, and it is easy to solve but not precise. Sequentially, the larger number of collocation points would be chosen even with a higher-order of collocation method, and it would be solved with more computation efforts and a more accurate solution.

As shown in Figure 2.5, there is a simple example for illustrating how the accuracy of meshed with a trapezoidal rule is improved by adding more points into segments. It could be observed from the intervals 5, 6 and others that the higher Error is in the certain

interval, the more points are added.

In the following part, there are mainly three ways, Minimax Error, Bisection, and Equadistributed Error [5] introduced to construct a new mesh, and the order of collocation would be kept all the way. Other methods are referred to as subdividing and increasing the polynomial order, named hp-adaptive meshing. For example, the polynomial order is changed from trapezoidal to Hermite-Simpson collocation [3–5].

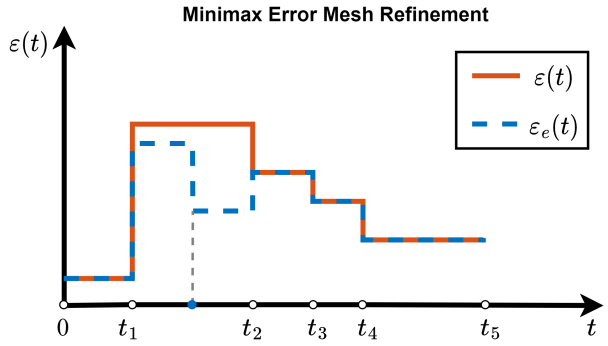

Figure 2.6: Illustration for Minimax Error Mesh Refinement Method

2.4.1 Minimax Error

The key idea of the method is adding a new point to the interval with the largest discretization error, such that the maximum error of all intervals would be effectively reduced. However, adding only one point to one interval for each mesh refinement is far not enough, since too many iterations would burden the computation. Hence, it provides a way to estimate the error distribution after adding new points such that new collocation points could be selected iteratively without solving the DOP.

Firstly, for a single interval, the initial discretization error during the time step size h could be estimated as:

$$\theta = ch^{p-r+1}, \tag{2.31}$$

where c is a constant coefficient, p denotes the order of accuracy of a method and is known, and r is the order of reduction and need to be estimated. There is an order reduction for

the numerical methods when it is applied to a DAE [5].

After I points are added to this interval, the new discretization error is

$$\eta = c(\frac{h}{1+I})^{p-r+1}, \tag{2.32}$$

where it is assumed that the order reduction r and the constant c would be kept the same for the initial and new grids.

Combining the two equations above, we could get the expression for $\hat{r}$:

$$\hat{r} = p + 1 - \frac{\log(\theta/\eta)}{\log(1+I)}. \tag{2.33}$$

To select a suitable integer, the order reduction could be estimated:

$$r = \max\left[0, \min(\text{nint}(\hat{r}), p)\right], \tag{2.34}$$

where nint means the nearest integer. Additionally, to make operation practical, it is assumed that the order reduction would be unchanged for all $I+1$ intervals in the new grid. Hence, the old, coarse grid would decide the estimated values of the order reduction.

Furthermore, when the original error is $\epsilon_k = ch_k^{p-r_k+1}$ and I_k points are added into the interval k, the approximation for the error on each of the $1+I_k$ subintervals would be

$$\hat{\epsilon}_k = (\frac{1}{1+I_k})^{p-r_k+1}, \tag{2.35}$$

with I_k is an integer and $I_k \geq 0$. Then, a set of integers I_k would be selected to build the new mesh and used for minimizing:

$$\phi(I_k) = \max_k \epsilon_k, \tag{2.36}$$

with the constraints

$$\sum_{k=1}^{N} I_k \leq M - 1, \tag{2.37}$$

$$I_k \leq M_1, \tag{2.38}$$

for $k = 1, \ldots, N$. The added constraints aim that no more than $M - 1$ total points are added to the current mesh, and for a single segment, at most M_1 points could be added to avoid an excessive number of subdivisions into one segment. Typically, $M_1 = 5$ and $M = N + 1$ is chosen.

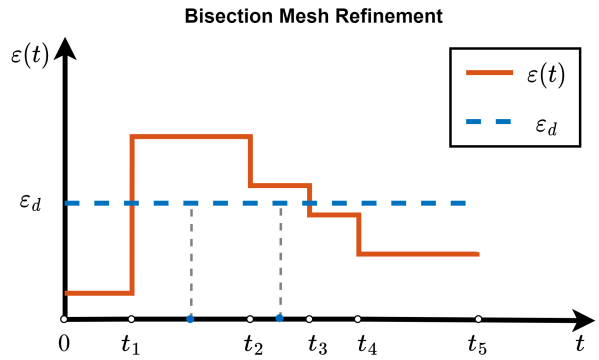

Figure 2.7: Illustration for Bisection Mesh Refinement Method

2.4.2 Bisection

The minimax error method severely depends on the original estimation for order reduction. Occasionally, the initial prediction from a coarse grid would not be accurate, and it would lead to seriously mistaken subdivisions. To avoid this situation, a bisection approach is provided by simply bisecting any interval whose error exceeds a specific error. Specifically, for each interval k of the old grid, if the residuals.

$$\epsilon_k > \epsilon_d, \tag{2.39}$$

then add a new grid point in the middle

$$t_{k+\frac{1}{2}} = \frac{1}{2}\left[t_k + t_{k+1}\right] \tag{2.40}$$

Obviously, for each refinement, the total added points would no more than N.

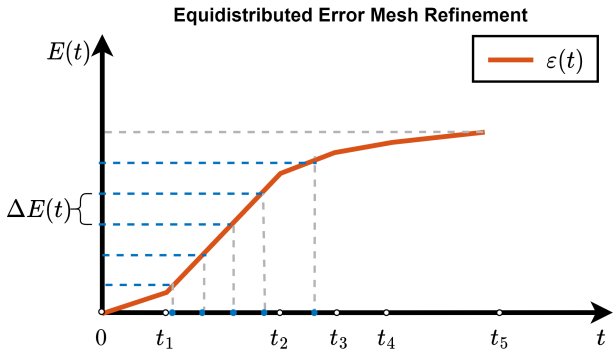

Figure 2.8: Illustration for Equidistributed Error Mesh Refinement Method

2.4.3 Equidistributed Error

The main idea of the Equadistributed Error method is to identify the newly inserted points by splitting the total error over the whole time span. As shown in Figure 2.8, accumulated error $E(t)$ increases, with time going on. Then, $E(t_F)$ would be divided by m (usually $m = N + 1$) uniformly and the error for each segment is $\Delta E(t)$. With these new breakpoints for total error, new points would be added into corresponding intervals.

2.5 Initial Guess

Initialization for nearly all initial guesses is important. A good initial guess will help the solver quickly find the globally optimal solutions. In the worst case, the optimization may fail to find a feasible solution. Hence, the setting of the initial guess should also be paid attention to. One simple way is to assume the trajectory is a straight line from initial to final points. Then, the variable values are found by linear interpolations. To deal with complex problems, the solution of simplified problems could be solved first to initialize

the original problem.

With this idea, we propose a novel strategy for an initial guess of direct collocation with the MR method. Similarly, since DOP solved with coarse mesh initially needs mesh refinement several times, the previous solution could be used as the next round of initial guess, named warm start. In contrast, the other method, cold start, means there is no initial guess provided to the NLP optimizer. To further compare the results, once warm start and quasi warm start are also provided. The former means that the initial guess is only provided for the NLP solver at the very beginning. The other one only assumes initial trajectories as straight lines and provides this in each iteration.

2.6 NLP Optimizer Tolerance

Our objective for solving a dynamic optimization problem is to find the optimal solution and make MIRNS lower than a threshold. In each iteration, DOP would be transformed to NLP by the transcription method. Then, the algorithm for NLP will terminate when (scaled) NLP error is smaller than desired convergence tolerance (*tol*, default 10^{-8}). At the early stage, there is no need for a too accurate solution by NLP optimizer. Hence, *tol* could be set as decreasing from large to small and no less than the default value (10^{-8}):

$$tol = max(\alpha N^k + \beta, 10^{-8}) \tag{2.41}$$

where parameters α, β and k are identified according to the specific problem.

2.7 Conclusion

In summary, the collocation method mainly has three steps:

- **Direct collocation:** Convert the DOP into an NLP problem by discretization,
- **Non-linear Program:** Solve the sparse nonlinear program
- **Mesh Refinement:** Evaluate the residuals, refine the discretization, and then repeat the optimization steps when necessary

Chapter 3

Julia Intro

BEFORE introducing the implementation of algorithms into a package, characteristics and advantages of Julia are discussed firstly as a warm start. There are typically two kinds of programming languages, dynamically typed languages like Python and statically typed languages like C. The former ones have better productivity, but less performance, and inversely, static languages are more performance but less productive. As a scientific computing language, Julia's innovation results from seeking the balance between productivity and performance. Then, a standard REPL help users interact with Julia and commands are collected in a .jl file or by typing directly in a Jupyter notebook. Most of its libraries are written in pure Julia to make it faster, such as JuMP.jl [8]. Other features that benefit this ability of Julia are listed as follows and also would be used during algorithm implementation [6]:

3.1 Expressive type system

In Julia, the type ecosystem is composed of built-in and customized types. Different from dynamically typed language, there is no meaningful distinction between them. For example, "ArrayT,dims" could be considered as a generic N-dimensions object that may contain any type T. Meanwhile, the vectorization improves performance.

3.2 Data-flow type inference

In general, the Julia compiler could automatically infer the types, even if there are minimal or no type annotations in the code. As a result, type annotations could be used for code selection. The implementation of data-flow type inference makes it so that the typing of code is regulated by the flow of data through it, like dynamic languages. In this way, there is no need to force the programmer to explicitly specify types, and programs would be automatically annotated with type bounds. Similar to MATLAB, vectorization is fully used in Julia. Since type information is considered in advance for vectorization, it would rescue "For Loops".

3.3 Multiple Dispatch

In Julia, the same name could be used for the different functions in different circumstances, referred to as "polymorphism". Then, a collocation of functions has the same name, and they often represent the same idea but operate on different structures. Finally, the right code would be selected based on argument types. The above process is called multiple dispatch. Take the built-in operator "+" as an example. "+" could not only deal with "+(x::Int64, y::Int64)" and "+(x::Int64, y::Float64)", but also "+(x::Vector{Int64}, y::Vector{Float64})". This characteristic helps code reused more efficiently compared with traditional object-oriented programming.

3.4 Meta Programming

In Julia, there are mainly two stages of source code execution in Julia parsing and then executing code. With Julia's meta-programming facilities, the code could be assessed after it's been parsed but before it's evaluated. Macros are used to accomplish this and operate at operating at the level of abstract syntax trees (AST). Therefore, macros allow the programmer to generate and include fragments of customized code before the full program is run. For example, given the "@time" macro, a "start the stopwatch" command

is inserted at the beginning of the code, and some codes are added at the end to "stop the stopwatch", and it calculates the elapsed time and memory usage. The modified code is then passed on for evaluation.

```
julia> @time [sin(cos(i)) for i in 1:100000];
0.108465 seconds (401.72 k allocations: 21.657 MiB, 97.05% compilation time)
```

In summary, the characteristics above benefit that Julia meets the needs for numerical computing and could run faster than other programming languages.

Chapter 4

DOP Solver - DCMR

IN this part, we mainly introduce how the DOP solver, named DCMR, with direct collocation and mesh refinement method is implemented in Julia. All code could be found by GitHub link: https://github.com/QuinnXie/DCMR.git.

4.1 Work Flow

Figure 4.1 and 4.2 provide the general view of work flow and whole algorithm structure.

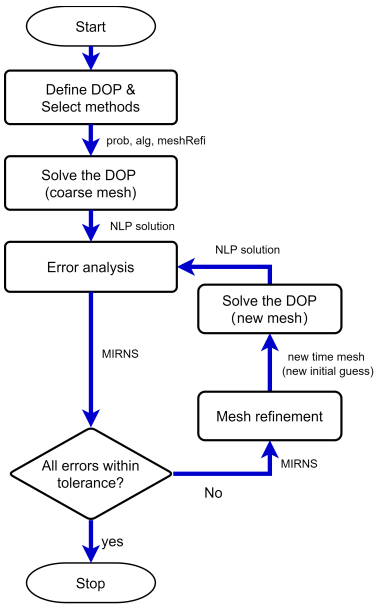

Figure 4.1: Flow Chart for Algorithm Structure

As shown in the flow chart, the algorithm starts with DOP definition and method selection for collocation and mesh refinement methods. With this information provided, the DOP is solved based on a coarse mesh, and then the collocation error is computed in the error analysis block. There is a cycle termination condition for judgement, that if all errors are within tolerance, such as MIRNS less than a residual threshold, then the algorithm would stop. Otherwise, a new time mesh would be built by the mesh refinement method and sent to the part of solving DOP. Again, with the NLP solution, discretization error will be computed for another judgement.

Additionally, more details in each block and whole algorithm structure could be found in Figure 4.2 and Algorithms 1-12.

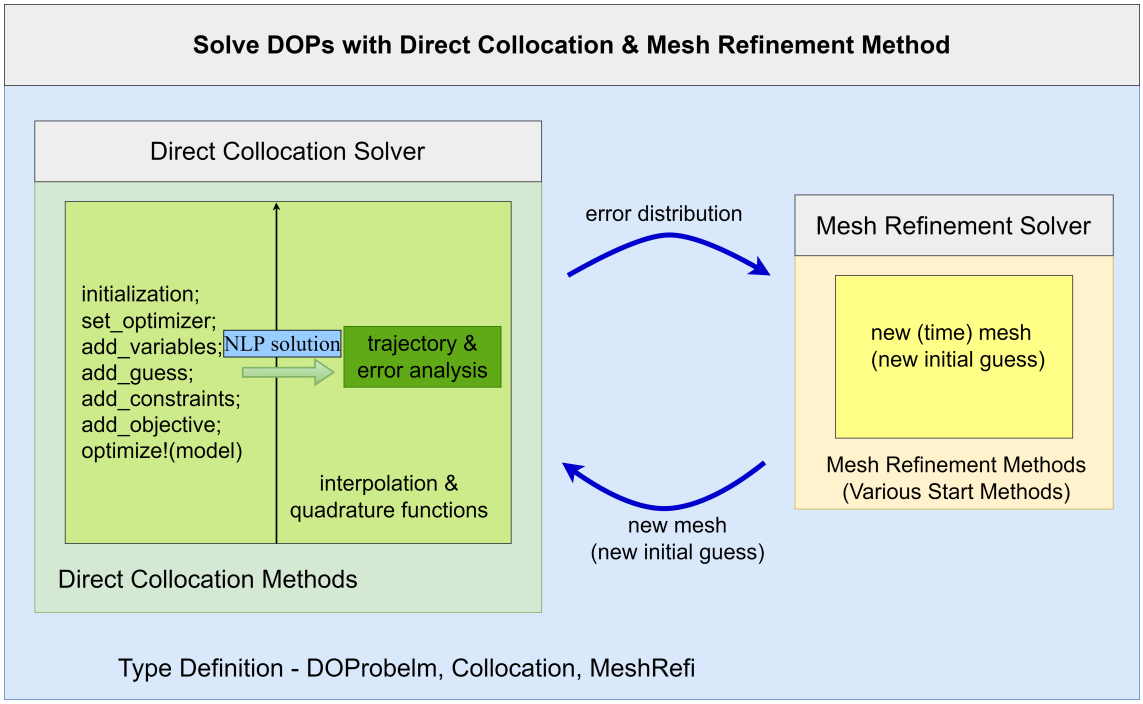

Figure 4.2: Block diagram for Algorithm Structure

4.2 Type Definition

Basically, many types and their connections form the whole ecosystem of types, which is a solid base for the algorithm. Various types are defined for DOProblem and multifarious methods, with the help of the expressive type system of Julia introduced in Chapter 3. In

the Julia code, type definitions are given by three files, "typeDOProblem.jl", "typeCollocation.jl" and "typeMeshRefinement.jl". They are mostly built with constructors, and there are also some sub-constructors inside. For example, *MRMinimax* constructor is composed of three parts: *resiThreshold*, *start* and *termination*, shown below. The type of each element has been originally defined, and the types of start and termination also work as parameters for this constructor. Hence, *MRMinimax* type will have lots of concrete subtypes, and meanwhile, it belongs to abstract type *MRMinimax*. In this way, constructors for other types could be built.

```
struct MRMinimax{S<:MRStart, T<:MRTerm}  <: MeshRefi
    resiThreshold :: AbstractFloat
    start :: S
    termination :: T
end
```

4.3 Direct Collocation Solver

Based on the type of ecosystem, two main subparts are built for direct collocation and mesh refinement solvers. Since there are so many methods and functions with the same idea, type inference and multiple dispatch are applied to select the right functions, introduced in Chapter 3. Algorithm 1 shows the specific process of solving DOP with direct collocation methods with JuMP [8]. As an open-source modelling language, JuMP allows users to formulate various classes of optimization problems with easy-to-read code. Then, these problems can be solved using state-of-the-art open-source and commercial solvers, such as IPOPT. Specifically, after initializing the JuMP model, an optimizer for solving the NLP would be set for the model. To cope with specific DOP, there are five key parts to prepare the model, *add_variables* 2, *add_constraints* 3, *add_objective* 4, *add_guess* 4 and *add_earlyTerm* 6. They are corresponding to the definition of DOP shown by Equations 2.1 - 2.6.

Algorithm 1: Function for solving dynamic optimization problem

Input: $prob$, alg, $meshRefi$
/* $prob$-DOP, alg-collocation method, $meshRefi$-MR method */
Output: $solution$, alg
1 **Initialize:** $JuMP\ model$;
2 set_optimizer to model; /* (e.g. Ipopt.Optimizer) */
3 add_variables Algorithm 2 with inputs $model$, $prob$, alg;
/* set variables for $model$ */
4 add_constraints Algorithm 3 with inputs $model$, $prob$, alg;
/* set constraints for $model$ */
5 add_objective Algorithm 4 with inputs $model$, $prob$, alg;
/* set objective function for $model$ */
6 add_guess Algorithm 5 with inputs $model$, $prob$, alg;
/* set initial guess for $model$ */
7 add_earlyTerm Algorithm 6 with inputs $model$, $prob$, alg;
/* set (scaled) NLP tolerance for $model$ */
8 optimize!($model$);/* solve the NLP */
9 trajectory function and TNIR ← soln_trajectory Algorithm 7 inputs $model$, $prob$, $value$, alg;
/* $value$ - NLP solution */
/* Get full piecewise trajectory of input and state */
/* calculate integrated residuals */
10 **Return**: $solution$, alg;

4.3.1 Add Variables

In *add_variables* 2, arrays for time, states and control are built and the simple bounds for them are also added at the same time. Additionally, the macro *@variable* is used to add defined variables to the model. A brief intro is presented in Chapter 3.

4.3.2 Add Constraints

Within *add_constraints* 3, the non-linear expressions for system dynamics are defined with macro *@NLexpression*. If expressions are quadratic and affine, linear macros like *@expression* should be used. In these algorithms, nonlinear problems are focused on, and linear would have a similar formulation. When dynamics expressions are prepared, collocation and interpolation constraints for TRP or HSS would be set to the model. Sections 2.2.2 and 2.2.3 show the formulation of them. In practical implementation, *if...else...* would appropriately be replaced by multiple dispatch with the type of *alg*.

Algorithm 2: Function for adding variables

Input: $model$, $prob$, alg
Output: $model$
Initialize: N, n;
`/* N - number of Intervals, n - number of points */`
$@variable\ t_i^{low} \leq t_i \leq t_i^{upp}$ for $i = 1 : N + 1$;
`/* set time variable constraints to model */`
$@variable\ \boldsymbol{x}_i^{low} \leq \boldsymbol{x}_i \leq \boldsymbol{x}_i^{upp}$ for $i = 1 : n$;
`/* set state variable constraints; initial and final states */`
$@variable\ \boldsymbol{u}_i^{low} \leq \boldsymbol{u}_i \leq \boldsymbol{u}_i^{upp}$ for $i = 1 : n$;
`/* set input variable constraints to model */`
Return: $model$;
`/* model with variables added */`

Algorithm 3: Function for adding constraints

Input: $model$, $prob$, alg
Output: $model$
Initialize: N, n;
`/* N - number of Intervals, n - number of points */`
$@NLexpression\ \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(t_i, \boldsymbol{x}_i, \boldsymbol{u}_i; \boldsymbol{p})$ for $i = 1 : n$;
`/* set system dynamics constraints */`
$h_i = t_{i+1} - t_i$;
if *Trapezoidal Collocation* **then**
  $@NLconstraint\ \boldsymbol{x}_{i+1} - \boldsymbol{x}_i = \frac{1}{2} h_i \cdot (\boldsymbol{f}_{i+1} + \boldsymbol{f}_i)$ for $i = 1 : N$;
  `/* collocation constraints */`
else if *Hermite-Simpson Collocation* **then**
  $@NLconstraint\ \boldsymbol{x}_{i+1} - \boldsymbol{x}_i = \frac{1}{6} h_i \cdot (\boldsymbol{f}_i + 4\boldsymbol{f}_{i+\frac{1}{2}} + \boldsymbol{f}_{i+1})$ for $i = 1 : N$;
  `/* collocation constraints */`
  $@NLconstraint\ \boldsymbol{x}_{i+\frac{1}{2}} = \frac{1}{2}(\boldsymbol{x}_i + \boldsymbol{x}_{i+1}) + \frac{1}{8} h_i (\boldsymbol{f}_i - \boldsymbol{f}_{i+1})$ for $i = 1 : N$;
  `/* interpolation constraints */`
else
  @error unrecognizable collocation method
Return: $model$; `/* with system and collocation constraints added */`
`/* Physical implementation of constraints adding would be various due to continuous and discontinuous input features */`

4.3.3 Add Objective Function

As for *add_objective* 4, nonlinear objective functions are discretized with trapezoidal rule or Hermite-Simpson rule. Quadrature would be applied for integration.

Algorithm 4: Function for adding objective

Input: $model$, $prob$, alg
Output: $model$
Initialize: N, n;
`/* N - number of Intervals, n - number of points */`
`/* objective:`$\min_{t_0,t_F,\boldsymbol{x},\boldsymbol{u}}\ J(t_0,t_F,\boldsymbol{x}(t_0),\boldsymbol{x}(t_F))+\int_{t_0}^{t_F}\omega(\tau,\boldsymbol{x}(\tau),\boldsymbol{u}(\tau))\mathrm{d}\tau$ `*/`
if *Trapezoidal Collocation* **then**
  $@NLobjective\ \min_{t_1,t_N,\boldsymbol{x},\boldsymbol{u}} J(t_1,t_N,\boldsymbol{x_1},\boldsymbol{x_n})+\Sigma_{i=1}^{i=N}\frac{1}{2}h_i\cdot(\omega_i+\omega_{i+1})$;
  `/* Trapezoidal quadrature for integration */`
else if *Hermite-Simpson Collocation* **then**
  $@NLobjective\ \min_{t_1,t_N,\boldsymbol{x},\boldsymbol{u}} J(t_1,t_N,\boldsymbol{x_1},\boldsymbol{x_n})+\Sigma_{i=1}^{i=N}\frac{1}{6}h_i\cdot(\omega_i+4\omega_{i+\frac{1}{2}}+\omega_{i+1})$;
  `/* Simpson quadrature for integration */`
else
  @error unrecognizable collocation method
Return: $model$; `/* with objective added */`
`/* Physical implementation of constraints adding would be various due to continuous and discontinuous input features */`

4.3.4 Add Initial Guess

During solving an optimization problem, an initial guess is the starting point for the solver. A great initial guess, even close to the optimal trajectory, could reduce iterations and lower the computation burden. As shown in Algorithm 5, we propose a warm start as the initial guess, which is achieved from the last solution. It fully uses the information of the last solution and uses linear interpolations to estimate the values of newly inserted points. For the other three, initial guess strategies are put in comparison.

4.3.5 Set NLP Tolerance

The last preparation for the model is setting NLP (scaled) tolerance in Algorithm 6. Early termination method applies function $\max\left(\alpha N^{-k}+\beta, 10^{-8}\right)$ to shorten running time.

Algorithm 5: Function for adding initial guess

Input: $model$, $prob$, alg
Output: $model$

if $cold\ start$ **then**
    no initial guess
else if $once\ warm\ start$ **then**
    /* add initial guess only once in the first solving */
    $vec\_initial\_guess \leftarrow$ initial and final time, states, inputs;
    set_start_value(all_variables($model$), $vec\_initial\_guess$);
else if $quasi\ warm\ start$ **then**
    $vec\_initial\_guess \leftarrow$ initial and final time, states, inputs;
    set_start_value(all_variables($model$), $vec\_initial\_guess$);/* vectorization */
    /* add simple initial guess to $model$ */
/* e.g. linear interpolations from initial states to final states */
else if $warm\ start$ **then**
    $vec\_initial\_guess \leftarrow (solnTime, solnState, solnInput)$;
    set_start_value(all_variables($model$), $vec\_initial\_guess$);
    /* new initial guess is achieved by trapezoidal interpolation and add new points to the location where new time horizons are added */
else
    @error unrecognizable initial start method
/* During mesh refinement with warm start, last solution would be functioned as initial guess for next round */
Return: $model$;

Algorithm 6: Function for adding NLP(scaled) tolerance

Input: $model$, alg, $meshRefi$
Output: tol

Initialize: N, tol; /* N - number of Intervals; tol - tolerance */
if $early\ Termination$ **then**
    $tol \leftarrow \max\left(\alpha N^{-k} + \beta, 10^{-8}\right)$;
    /* tol decreases with increasing N and no less than default */
    set new NLP tolerance to $model$;
else if $defualtTermination$ **then**
    keep NLP tolerance default; /* default tol = 10^{-8} */
else
    @error unrecognizable termination method
Return: tol;

After solving the NLP problem, *value*, the NLP solution is achieved and contains the optimal solution at each point. To get the whole trajectory, *soln_trajectory* Algorithm 7 is developed for finding the interpolation in each interval and error distribution over the whole time scale. Finally, full *DOSolution* is achieved containing NLP solution, polynomial function for each interval of states and inputs, and collocation error. It paves the way for the whole trajectory plots and meshes refinement.

Algorithm 7: Function for computing interpolations for NLP solution and collocation error

Input: $model$, $value$, $prob$, alg
Output: $solution$, alg
Initialize: N, $nstate$, $ninput$; optimization solution $\leftarrow$ $value$ and $model$;
interpolation polynomial matrix $\leftarrow$ interpolation functions with inputs optimization solution;
`/* given type of collocation method, right interpolation is selected for states and inputs */`
collocation error $\leftarrow$ simpsonQuadrature functions with inputs optimization solution and interpolation polynomial matrix;
`/* simpsonQuadrature used for computing integrals with Simpson quadrature rule */`
Return: $solution$, alg;

4.4 Mesh Refinement Solver

The other key part of the package is the mesh refinement (MR) solver. These three types of MR methods, Minimax Error 8, Bisection(Multi) 10 and Equidistributed Error 11. The basic workflow includes building a new mesh and using a direct collocation solver 1 to solve the DOP again with a new mesh within a *while* loop until the MIRNS requirement is satisfied. Different ways of finding new mesh or selecting points inserted into each interval are decided by each mesh refinement method. Meanwhile, with the new mesh, the initial guess would also be decided by the initialization strategies such as warm start. The formulation of each MR method can be found in Section 2.4.

Algorithm 8: Function for Minimax Error MR method

Input: $prob$, alg, $meshRefi$
Output: $prob.solution$, alg
Initialize: N, $nstate$ /* p - order of accuracy TRP: $p = 2$ HSS: $p = 4$ */
Estimate order reduction r by Equations 2.33 and 2.34;
/* simple mesh refinement carried on and add one point to each interval to estimate order reduction */
while *MIRNS* $> resiThreshold$ **do**
  /* $resiThreshold$ - target residual level (e.g.10^{-6}) */
  $numInsert \leftarrow$ Algorithm 9 with inputs r, $errorMean$, $resiThreshold$, $errorTime$, alg, $meshRefi$;
  /* $numInsert$ - number of points inserted to each interval */
  update $prob.solution$, alg;
  /* right function chosen for building initial guess according to start strategies */
  $solution$, $alg \leftarrow$ Algorithm 1 with inputs $prob$, alg, $meshRefi$;
Return: $prob.solution$, alg;

Algorithm 9: Function for finding points distribution for insertion (Minimax)

Input: r, $errorMean$, $resiThreshold$, $errorTime$, alg, $meshRefi$
Output: I
Initialize: N, M, I, ΔM, M_1, κ /* I - added points distribution */
while *(constraints like Equtions 2.37 and 2.38)* **do**
  find maximum error and the according interval;
  add one point to the interval and update I;
  predict new error distribution;
Return: I;

Algorithm 10: Function for Bisection (Multi) MR method

Input: $prob$, alg, $meshRefi$
Output: $prob.solution$, alg
while *MIRNS* $> resiThreshold$ **do**
  /* $resiThreshold$ - target residual level (e.g.10^{-6}) */
  $numInsert \leftarrow errorMean .> resiThreshold$;
  /* add one point to where local MIRNS larger than threshold */
  update $prob.solution$, alg;
  /* right function chosen for building initial guess according to start strategies */
  $solution$, $alg \leftarrow$ Algorithm 1 with inputs $prob$, alg, $meshRefi$;
Return: $prob.solution$, alg;

Algorithm 11: Function for Equidistributed Error MR method

Input: $prob$, alg, $meshRefi$
Output: $prob.solution$, alg

while *MIRNS* $> resiThreshold$ **do**
  /* $resiThreshold$ - target residual level (e.g.10^{-6}) */
  $numInsert \leftarrow$ Algorithm 12 with inputs $numInterval$, $errorMean$, $meshRefi$;
  /* $numInsert$ - number of points inserted to each interval */
  update $prob.solution$, alg;
  /* right function chose for building initial guess according to start strategies */
  $solution$, $alg \leftarrow$ Algorithm 1 with inputs $prob$, alg, $meshRefi$;
Return: $prob.solution$, alg;

Algorithm 12: Function for finding inserted points distribution (Equidistributed)

Input: $numInterval$, $errorMean$, $meshRefi$
Output: I

$M \leftarrow numInterval + 1$ $\epsilon \leftarrow errorMean$ $M_1 \leftarrow 5$;
compute error distribution $E_k = E_{k-1} + \epsilon_{k-1}$;
$\Delta E \leftarrow E_M/M$;
compute uniform error distribution $\hat{E}_k = \hat{E}_{k-1} + \Delta E$;
break points of $\hat{E}$ in each original interval set as inserted points and I_k no more than M_1;
Return: I;

Chapter 5

Results and Analysis

THIS chapter would take a concrete dynamic optimization problem as an example to compare the results of various methods. During algorithm implementation, the continuity of control is considered, such as continuous and discontinuous inputs. Meanwhile, trapezoidal and Hermite-Simpson collocation methods both are applied to them. Hence, there are a group of four circumstances given in the code accomplishment. Note that abbreviations are applied for classification, Trapezoidal Collocation Method with Continuous Input (TRPuC), Trapezoidal Collocation Method with Discontinuous Input (TRPuD), Hermite-Simpson Collocation Method with Continuous Input (HSSuC) and Hermite-Simpson Collocation Method with Discontinuous Input (HSSuD).

Then, each circumstance would be further optimized with various mesh refinement methods after initial solutions for coarse mesh are achieved. Since some of the results are almost similar despite using different mesh refinements, only the typical one would be presented for classification.

5.1 Cart-Pole Swing-Up Example

The cart-pole system comprises a cart travelling along a horizontal track and a pendulum hanging freely from the cart. The cart is driven by a motor forward and backwards. The objective is that the pendulum is swung up to the point of inverted balance above the cart

from the rest place. We would apply direct collocation and mesh refinement methods to this model and complete the "swung-up" manoeuvre.

As for the system dynamics, it is a second-order dynamical system, and there are four states. The position of the cart and the angle of the pole are q_1 and q_2, respectively, and the control force is given by u. There are also some system parameters, masses for cart and pole given by m_1 and m_2, the length of pole given by l and gravity acceleration, g. The dynamics ($\ddot{q}_1$ and $\ddot{q}_2$) for the cart pole system are:

$$\ddot{q}_1 = \frac{lm_2 \sin(q_2)\dot{q}_2^2 + u + m_2 g \cos(q_2)\sin(q_2)}{m_1 + m_2(1 - \cos^2(q_2))}, \tag{5.1}$$

$$\ddot{q}_2 = -\frac{lm_2 \cos(q_2)\sin(q_2)\dot{q}_2^2 + u\cos(q_2) + (m_1 + m_2)g\sin(q_2)}{lm_1 + lm_2(1 - \cos^2(q_2))}. \tag{5.2}$$

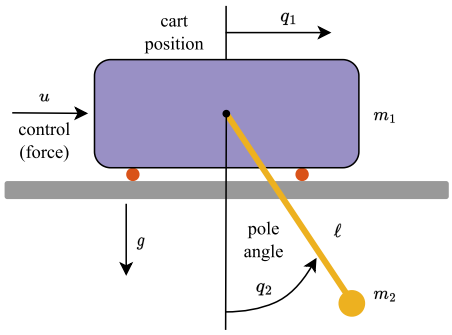

Figure 5.1: Cart-Pole Swing-Up Model

Besides the cart-pole position and pole angle, corresponding first order derivatives are also included in states, velocity and angular velocity.

$$\boldsymbol{x} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}, \qquad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u) = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix}. \tag{5.3}$$

Then, the objective function of this trajectory problem is set as the integral of

actuator-effort (control) squared:

$$J = \int_0^T u^2(\tau)\mathrm{d}\tau. \tag{5.4}$$

It helps to produce smooth trajectories and is easier to stabilize with conventional controllers physically.

Given the boundary constraints, the initial and final states are fixed, and it supposes that the cart starts from the centre of the rails and travels for a distance d during the pole's swing-up manoeuvre.

$$\begin{aligned} q_1(t_0) &= 0, \qquad & q_1(t_F) &= d, \\ q_2(t_0) &= 0, \qquad & q_2(t_F) &= \pi, \\ q_3(t_0) &= 0, \qquad & q_3(t_F) &= 0, \\ q_4(t_0) &= 0, \qquad & q_4(t_F) &= 0. \end{aligned} \tag{5.5}$$

Additionally, to make it more realistic, we limit the horizontal range of the cart and restrict the motor force to some maximal force.

$$\begin{aligned} -d_{max} \leq q_1(t) \leq d_{max}, \\ -u_{max} \leq u \leq u_{max}. \end{aligned} \tag{5.6}$$

5.2 Solution: Problem Feasibility

When we solve an optimal problem, the first key step is to confirm the feasibility of the problem. After several trials, it could be found that it needs at least 4 intervals of mesh for the trapezoidal collocation method and 3 intervals for the Hermite-Simpson collocation method enough. To fairly compare the results, the initial number of coarse mesh is set as 4 for all.

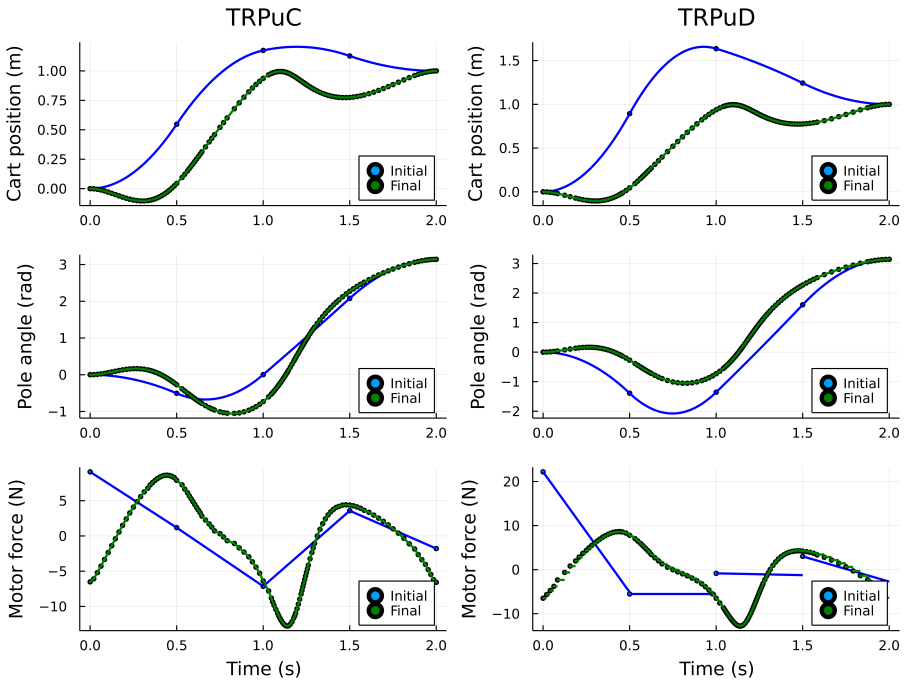

Figure 5.2: Cart-Pole Optimal Trajectory Given by Trapezoidal Collocation Method

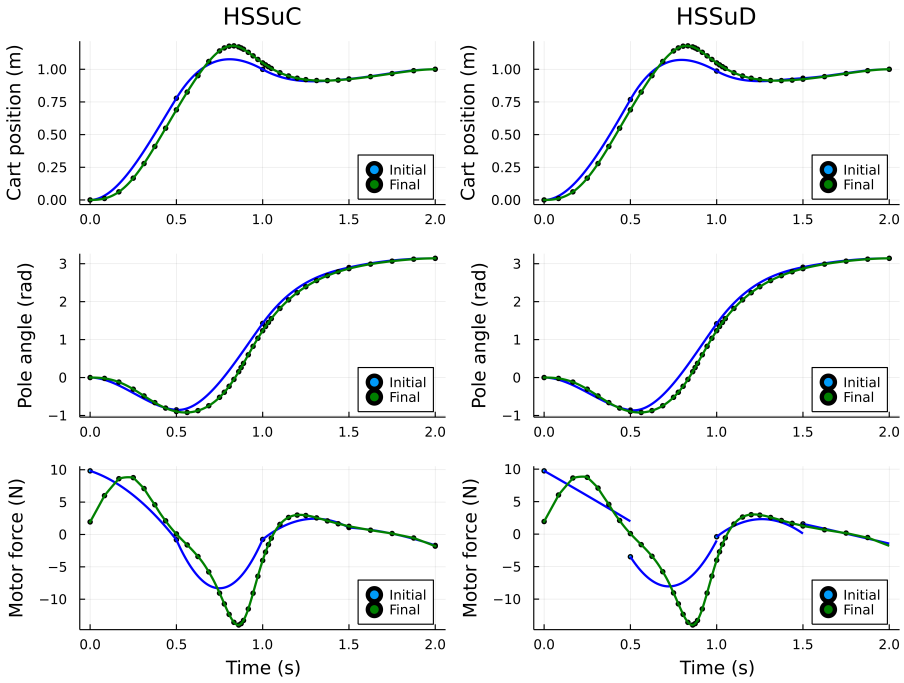

Figure 5.3: Cart-Pole Optimal Trajectory Given by Hermite-Simpson Collocation Method

5.3 Solution: Optimal Trajectory

Trapezoidal and Hermite-Simpson collocation methods solve the dynamic optimization problem. Since the final trajectories given by various mesh refinement methods are almost the same, we only propose the final trajectory given by Minimax Error as an example for discussion.

Under different methods and continuities, the optimal trajectories for cart position, pole angle and motor force are shown in Figure 5.2 and 5.3 separately.

Since figures are composed of many parts, some views are selected for discussion as follows:

5.3.1 Initial and final results in contrast

Generally, each figure shows that initial and final trajectories are labelled in contrast. It could be observed that they have a similar tendency for both input continuity circumstances. However, the coarse mesh would be given significant discretization error, and the solution would be improved with several points added.

For initial trajectories, they are all composed of four parts uniformly since the number of intervals all starts from 4 at the very beginning.

With the help of mesh refinement methods, final trajectories show that intervals are denser but not uniform. Dense points stay where discretization errors are significant. It could be inferred that where points are dense means, the dynamics would change rapidly. Points mainly concentrate in the middle of the trajectory.

5.3.2 Input continuities in contrast

When inputs are continuous, each interval of motor force is connected, and inversely, disconnections of inputs represent discontinuity. Despite apparent differences for continuous and discontinuous inputs at the early stage, they would converge gradually with an increase in interval numbers and finally achieve almost the same shapes. However, all states

are continuous due to integrals, whatever the continuity of inputs. Meanwhile, final states' trajectories are almost the same without being affected by input continuities.

5.3.3 Collocation methods in contrast

In the first four figures of Figure 5.2, both cart position and pole angle are depicted with piecewise quadratic functions, which are found by Equation A.3. Corresponding to Equation 2.17, inputs are piecewise linear, shown in the last two subplots of Figure 5.2.

Similarly, for the Hermite-Simpson rule, it could be found that two ends of each input interval are connected with a quadratic function, given by Equation 2.26. Then, the Hermite-Simpson rule gives the cubic splines of states, formulated by Equation 2.27.

Comparing the results given by two collocation methods, it could be found that the Hermite-Simpson rule would be more accurate than the trapezoidal rule. Firstly, when the initial number of intervals and required resolution are the same, the initial solutions by HSS are closer to the final solutions than those computed by TRP. The other reason is that obviously, fewer intervals are used with HSS and intervals used by TRP would be far denser at the final stage.

Additionally, there are still some discontinuous parts for the final trajectory of TRPuD, although it has a similar shape to that of TRPuC. Inversely, for HSS, both input trajectories are continuous and converge to the same shape.

Last but not least, with two collocation methods, the final solutions they converge to are different. It could be inferred that there are at least two optimal solutions for this dynamic optimization problem. Meanwhile, it means that sometimes the solutions are not unique for optimal control problems. Additionally, the initial setting affects the final results, which should be selected carefully. For TRP, when the initial intervals are set as 5, semblable final trajectories are achieved with HSS now.

5.4 Solution: Error Analysis

The error plots are almost the same for various input continuities. Therefore, only discontinuous inputs are considered in the body part, and continuous could be found in Appendix A.2 for reference. The left two figures of Figure 5.4 show the entire velocity distribution and angular velocity error over time. Then, the absolute error is integrated to evaluate the total error in each interval shown on the right-hand side. The width of each interval represents the corresponding time horizon. We could see that the errors are pretty significant, even nearly 1, and solutions are inaccurate due to a small number of intervals. Hence, mesh refinement is necessary to reduce collocation errors, and error distribution sets a solid foundation for mesh refinement.

After total MIRNS is decreased to less than 10^{-6}, the error distribution is shown in Figure 5.5. Overall, discretization error is reduced, and the most significant absolute local error is less than 10^{-3}. Hence, the final solutions are more accurate to meet practical demands. The various time intervals live up to the non-uniform points distribution shown in Figure 5.2.

Comparing Figures 5.4 and 5.6, we could find the collocation error of HSSuD is overall more minor than that of TRPuD when both numbers of intervals are 4. The comparison results are coincident with those given in Section 5.3

After the mesh refinement process, all absolute local errors are less than 5×10^{-4}. Especially lots of points are concentrated in the middle part, and the error is still small at the end despite fewer points.

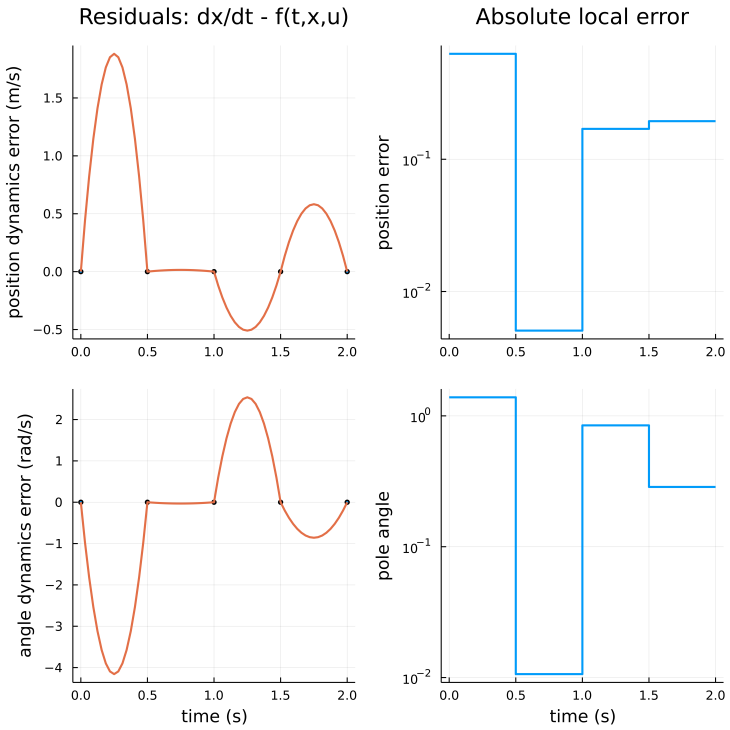

Figure 5.4: Initial Error Distribution by TRPuD

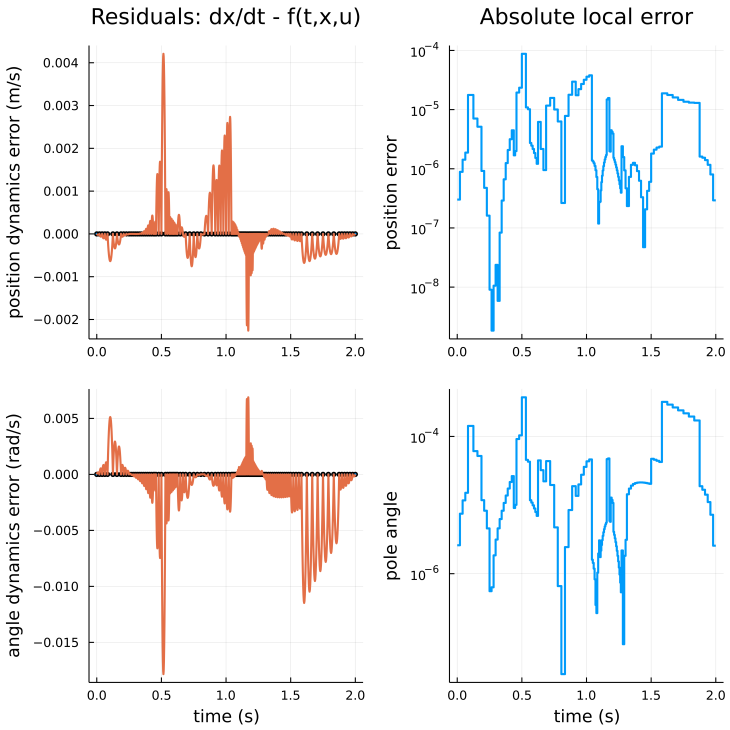

Figure 5.5: Final Error Distribution by TRPuD

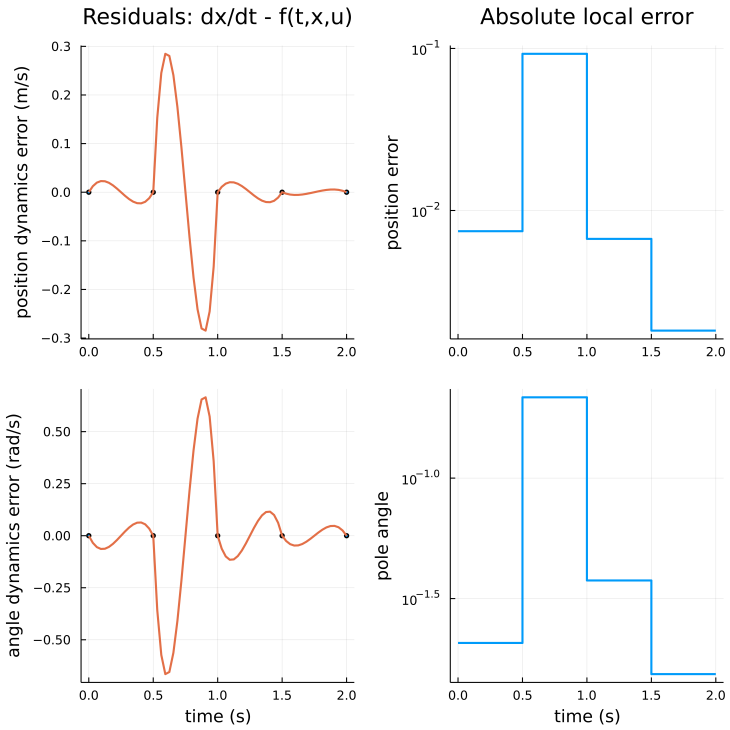

Figure 5.6: Initial Error Distribution by HSSuD

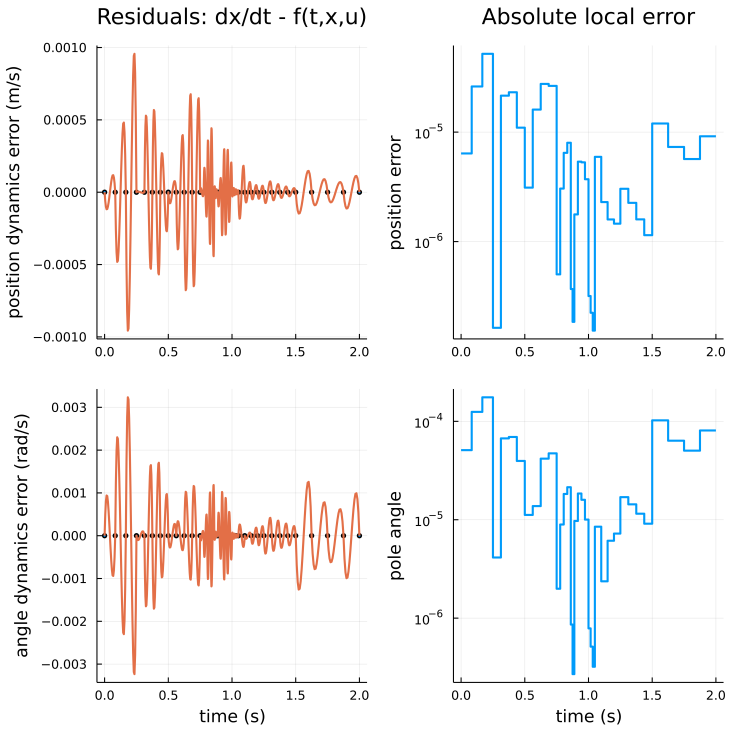

Figure 5.7: Final Error Distribution by HSSuD

5.5 Solution: Time Step

Furthermore, more details of the time step distribution can be found in Figure 5.8. Basically, the final time steps are only around one-tenth original ones for TRP and are reduced to one-fifth for HSS. Due to the high order of accuracy, HSS uses fewer intervals than TRP. Then, the whole distributions are not uniform. When the system dynamics change most quickly, the smallest part of TRP's time steps is between $1.0s$ and $1.25s$. Differently for HSS, they are located from $0.75s$ to $1.0s$. Given these parts in Figure 5.2 and 5.3, the time schedules are when the absolute of inputs are largest and change more quickly. With the same collocation method, time step distributions are similar for TRP, and even the same for HSS.

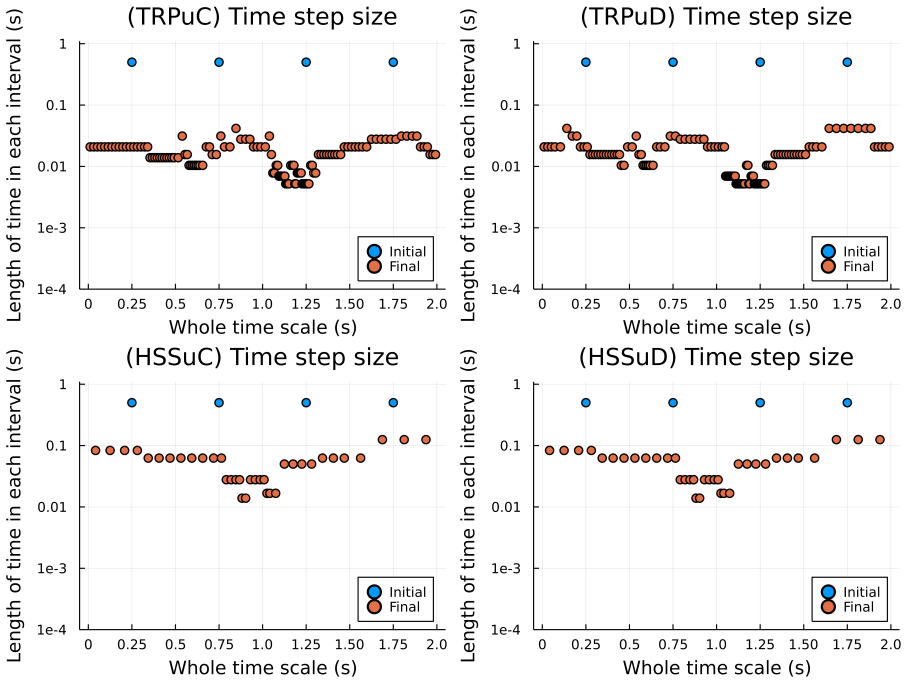

Figure 5.8: Time Horizon Distribution for Four Circumstances

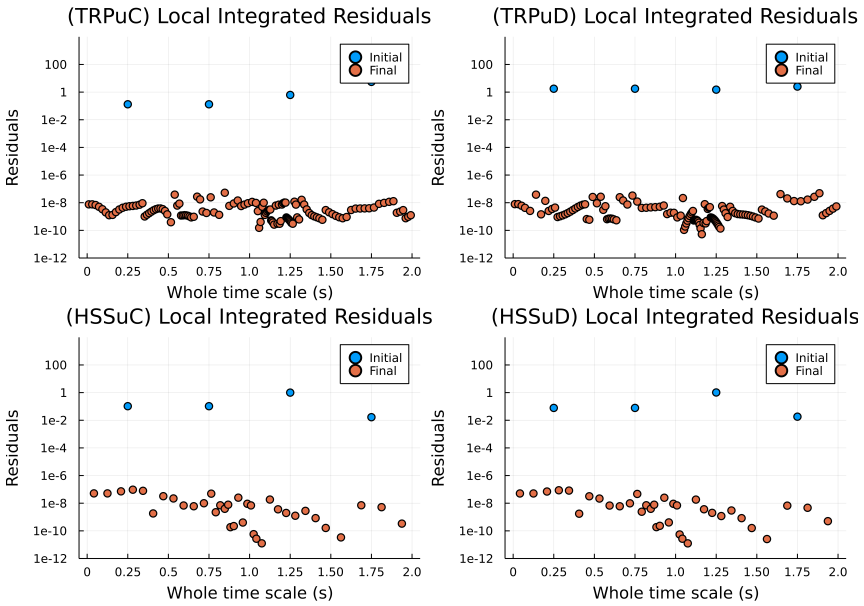

Figure 5.9: Local Integrated Residuals for Four Circumstances

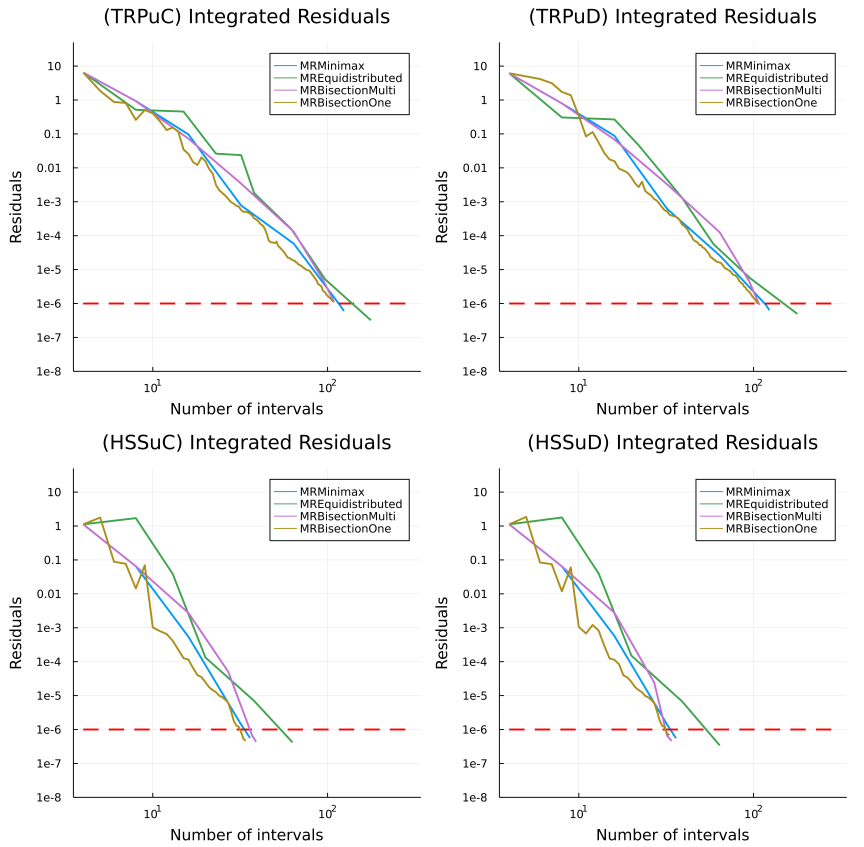

Figure 5.10: MIRNS Changes with Increasing Number of Intervals for Four Circumstances

5.6 Solution: Residuals

Equation 2.29 is used to compute the residuals of each interval considering all states to evaluate the collocation error. It paves the way for mesh refinement. Figure 5.9 shows the initial and final integrated residual distribution for four circumstances. The discretization error is reduced with an increase in the number of intervals. Generally, the local residuals reach 10^{-9} around for all at the end. As depicted before, the trapezoidal rule spends more intervals to reach the target accuracy. Lastly, the full resolution would be computed by MIRNS, and the mesh algorithm would run until the target MIRNS is met.

Figure 5.10 shows the process with various mesh refinement methods to depict the decreasing process of residuals. Firstly, Bisection(One) mesh refinement method means that each time only the interval with the largest error would be added one point such that the least points are added precisely, but more computation resources would cost. It works as a standard for comparison.

Generally, residuals would decrease gradually with points added despite some fluctuations in the trajectory. It is mainly affected by the computation accuracy of the NLP solver.

Firstly, let us compare the decreased speed of various MR methods for TRP. The Bisection (Multi) mesh refinement method uses the least number of points in TRP and is quite close to that of Bisection(One), although it is not the fastest at the beginning. It uses fewer points than Minimax Error because the order reduction is estimated inaccurately by coarse mesh, and it affects the efficiency obviously when the number of intervals increases. Typically, most points are applied by the Equidistributed Error MR method finally. Meanwhile, its decreased ratios change rapidly compared with other methods, telling that sometimes it is unstable.

A similar situation for Equidistributed also happens in HSS. However, the Minimax Error MR method is the fastest within HSSuC. It is beneficial from the accuracy of HSS such that the order of reduction could be predicted more precisely. As for HSSuD, Bisection(Multi) still uses the least number of intervals.

Comparing the results given by TRP and HSS, it could be found that the decrease ratio of HSS is more significant than that of TRP on the whole. Hence, the final number of intervals would be smaller for HSS.

In summary, Minimax Error could refine mesh efficiently when the estimation is precise. Significantly, the more points are added, the more pronounced the deviation is. Bisection(Multi) is a simple and stable way to select the needed points. It would replace when the precise estimation is unavailable. In this example, the performances of the Equidistributed Error MR method are unstable sometimes and still need to be improved.

5.7 Solution: Warm Start

A pivotal index to evaluate a dynamic optimization algorithm is the total running time, especially for solving complicated problems. With various collocation and mesh refinement methods applied to cart-pole swung-up problem, running time would be different. The novel strategy, warm start, is also applied to the refinement process. In comparison, there are three other initial guess strategies proposed, cold start (no initial guess provided to NLP solver), once warm start (initial guess only given once at the very beginning) and quasi warm start (initial guess provided and inferred by linear interpolations between initial and final states).

To begin with, Figures 5.11 - 5.14 show that there are mainly four parts of the bar plot, representing various MR methods. Under the corresponding MR method, four bars illustrate the estimated average running time with different initial guess strategies. The running time is achieved by commands @benchmark, and ten samples with one evaluation would be carried on for each circumstance. Additionally, the median time is more robust than the average time and is therefore selected as a final result.

It could be predicted that the more helpful information is provided as an initial guess, the better the algorithm's performance. It also coincides with the running time index, especially comparing cold and warm start results in each subpart. For example, the first, second and third subparts of Figure 5.11 depict that the running time cost of a warm

start is only one-third of that of a cold start. Generally, the tendency is to decrease within each part, and the lowest time is achieved by the warm start.

Given the time cost by various MR methods of TRP, less time is spent by Bisection overall, but Minimax Error achieves the least running time with a warm start down to $0.21s$. Then, Bisection(One) uses the running time exceeding $2.5s$ for both TRPuC and TRPuD, and even the longest one is up to $8.5s$. Due to more iterations and only one point added each time, the time cost would be longer than other MR methods.

Considering the effects of input continuity on execution time, it is predictable that it would take more time to solve the method with discontinuous inputs since more control variables are applied. Many bars of TRPuD would be longer than that in TRPuC, which could be observed from the first and fourth subparts. The main differences happen in Equidistributed Error MR. The time used by cold start in TRPuD is obviously shorter than that of TRPuC. The reason would be illustrated later due to the setting of target MIRNS.

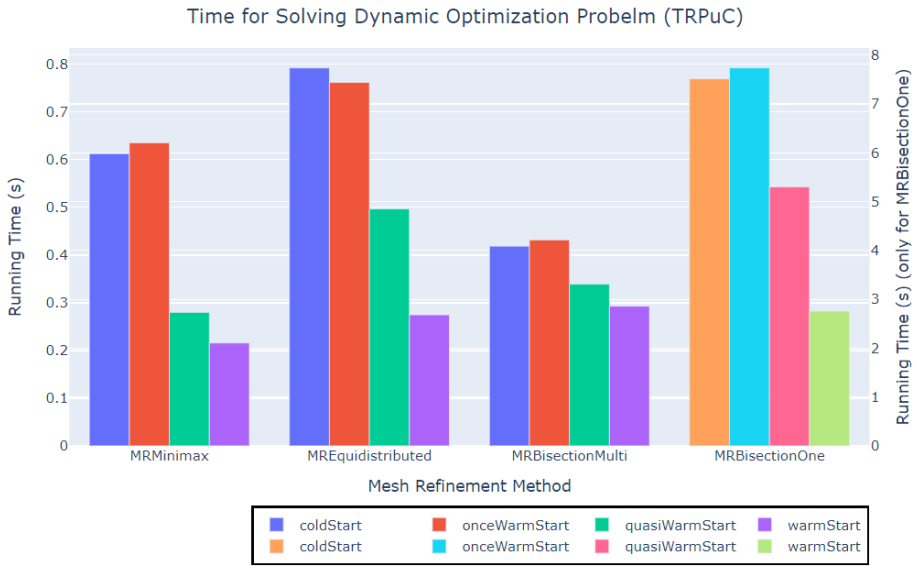

Figure 5.11: Total Running Time Cost by Various MR Methods for TRPuC

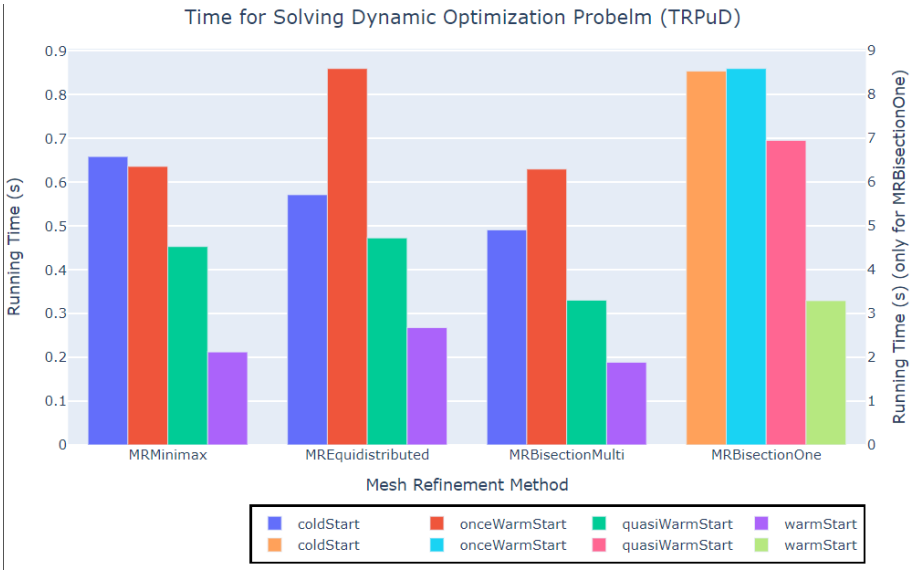

Figure 5.12: Total Running Time Cost by Various MR Methods for TRPuD

As for HSS in Figures 5.13 and 5.14, the decrease tendency in each subplot is not obvious. The running time spent by the warm start method is longer than that of the once quasi warm start in the third subpart of HSSuC and HSSuD. There are also similar circumstances happening in Equidistributed Error MR for HSSuD.

These circumstances happening does not necessarily mean the failure of the strategies. The reasons could be clarified as follows: Firstly, the target of mesh refinement is to reduce MIRNS smaller than some standard (e.g. 10^{-6}). When the same number of iterations are used by the programs with two different initial guesses ways, the MIRNS of quasi warm start method would just below the objective value such as 0.98×10^{-7}, and the program stops. Unfortunately, for the warm start, MIRNS is only a little larger than the expected value, such as 1.01×10^{-6}, and one more iteration would be carried on with more intense mesh, and much more running time would be used. Hence, this phenomenon is reasonable, and it is still worth using warn start as the beginning of NLP.

Then, Minimax Error MR saves more computation time due to a more accurate estimation, followed by Bisection(Multi) MR in HSS results. Similarly, The longest running time still belongs to Bisection(One).

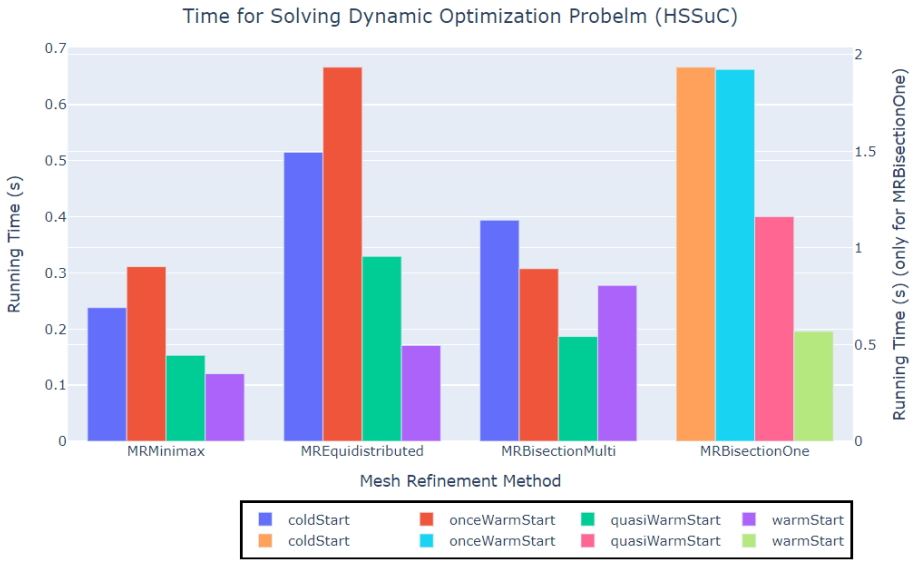

Figure 5.13: Total Running Time Cost by Various MR Methods for HSSuC

To compare the execution time with various collocation methods, Figure 5.15 shows them with different MR methods. Another comparison is put in Appendix A.3 to avoid repetition. Relatively, we could observe that the time used by HSS is less than that of TRP due to method accuracy. Especially in Figure A.7, the comparison is obviously with more iterations. Comparing each subpart, we could find that the shapes are similar with the same collocation method with different input continuities.

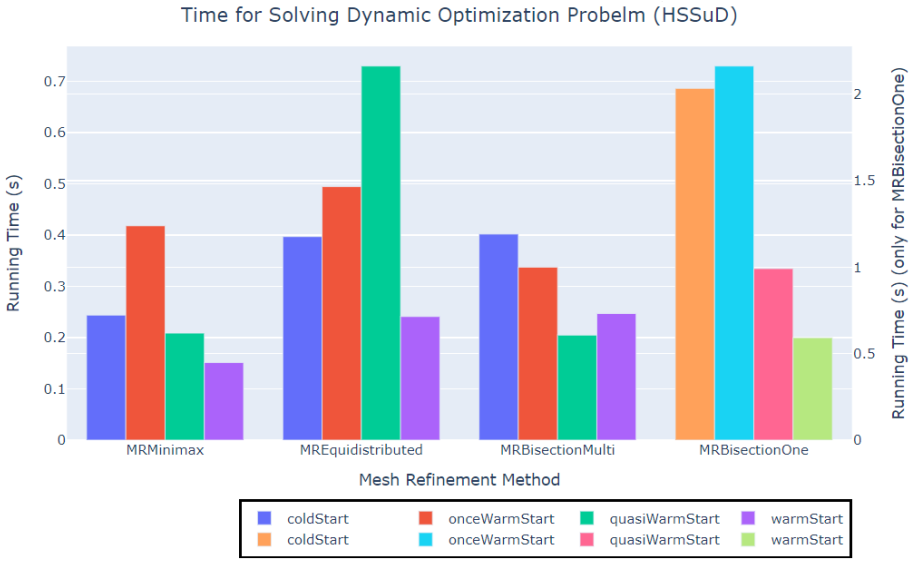

Figure 5.14: Total Running Time Cost by Various MR Methods for HSSuD

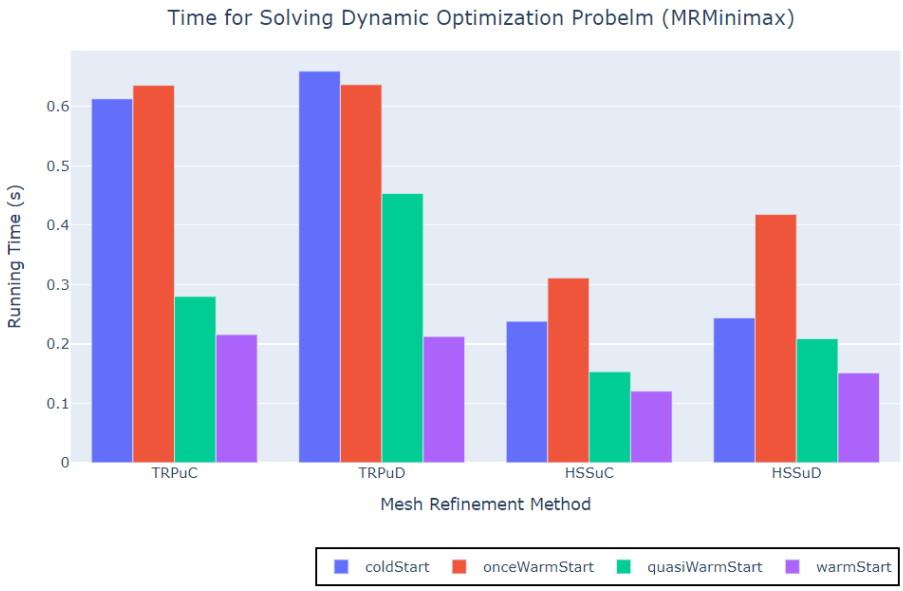

Figure 5.15: Total Running Time Cost by Various Collocation Methods with Minimax Error

	Running Time (ms)						Average ratio
	TRPuC			TRPuD			
tolerance	M	E	B	M	E	B	
default(10^{-8})	235.6	274.0	186.7	225.0	270.5	217.3	
$\max(10^{-3}N^{-1}, 10^{-8})$	221.5	255.4	**188.6**	212.3	266.6	197.0	
ratio	-6.0%	-6.8%	1.0%	-5.7%	-1.5%	-9.4%	-4.7%
$\max(4 \times 10^{-3}N^{-2}, 10^{-8})$	222.4	268.7	**189.3**	**244.1**	**275.2**	**229.3**	
ratio	-5.6%	-1.9%	1.4%	8.5%	1.7%	5.5%	1.6%
$\max(16 \times 10^{-3}N^{-3}, 10^{-8})$	223.1	269.6	**206.9**	215.8	**297.8**	194.0	
ratio	-5.3%	-1.6%	10.9%	-4.1%	10.1%	-10.7%	-0.1%
$\max(64 \times 10^{-3}N^{-4}, 10^{-8})$	226.7	**283.8**	**196.8**	208.6	**276.5**	198.3	
ratio	-3.8%	3.6%	5.4%	-7.3%	2.2%	-8.8%	-1.4%
M - Minimax Error; E - Equidistributed Error; B - Bisection(Multi)							

Table 5.1: Various Setting NLP Tolerance Functions for TRP

5.8 Solution: Termination

Early termination strategies introduced in Section 2.6 are applied during the solving process to lower the computation time further. In Tables 5.1 and 5.2, it shows that the running time is consumed in each situation.

Firstly, there are 12 combinations of collocation and mesh refinement methods where the warm start has been applied. In the default(10^{-8}) row, there are original running times with the default setting of NLP tolerance. Then the other sub-tables show the running time of the method with early termination. They are all estimated by *@benchmark* with 50 samples. There are four functions selected, $10^{-3}N^{-1}$, $4\times10^{-3}N^{-1}$, $16\times10^{-3}N^{-2}$, $64\times10^{-3}N^{-3}$. For all coarse mesh, the number of intervals is 4. To ensure fairness of contrast, we make all tolerance start from the same value. The constant α would be times 4 to eliminate the effect of $1/N$ at the beginning. Meanwhile, to fairly compare the default one, the tolerance of early termination will not be less than 10^{-8} such that the $\max$ function is used. Then, as for the ratio row, it is computed by

$$ratio = \frac{t_{ealy} - t_{defalt}}{t_{defalt}} \times 100\%. \tag{5.7}$$

where t_{early} is the running time with early termination and $t_{default}$ is the running time with default tolerance. The average ratio is also computed to evaluate the overall effects.

They are labelled boldly, meaning the running time is longer than before. Even one of the average ratios is positive, 1.6%. All of them happen in Table 5.1. The inferred reason is that the optimisation for TRP is not apparent due to the faster increase of interval number. Hence, the correct early term should be selected carefully, especially for TRP.

Comparing the average ratios, we could find that the best performance is achieved by $\max(10^{-3}N^{-1}, 10^{-8})$ in Table 5.1, where the running time is reduced by 4.7%. Differently, the best one is $\max(10^{-3}N^{-2}, 10^{-8})$ in Table 5.2 and nearly 10% computation time is reduced. Additionally, the effects of early termination are better in HSS compared with TRP overall.

	Running Time (ms)						Average ratio
	HSSuC			HSSuD			
tolerance	M	E	B	M	E	B	
default(10^{-8})	126.7	204.2	296.8	149.6	212.8	220.1	
$\max(10^{-3}N^{-1}, 10^{-8})$	115.3	187.8	288.9	141.5	192.7	215.3	
ratio	-9.0%	-8.1%	-2.7%	-5.5%	-9.4%	-2.2%	-6.1%
$\max(4 \times 10^{-3}N^{-2}, 10^{-8})$	112.8	187.1	289.1	124.6	190.6	206.1	
ratio	-10.9%	-8.4%	-2.6%	-16.8%	-10.4%	-6.3%	-9.2%
$\max(16 \times 10^{-3}N^{-3}, 10^{-8})$	115.7	187.7	268.5	141.6	196.4	202.1	
ratio	-8.6%	-8.1%	-9.5%	-5.4%	-7.7%	-8.2%	-7.9%
$\max(64 \times 10^{-3}N^{-4}, 10^{-8})$	114.9	187.0	271.1	136.4	194.8	206.5	
ratio	-9.3%	-8.4%	-8.6%	-8.9%	-8.4%	-6.2%	-8.3%
M - Minimax Error; E - Equidistributed Error; B - Bisection(Multi)							

Table 5.2: Various Setting NLP Tolerance Functions for HSS

Chapter 6

Conclusion and Future Work

IN conclusion, the direct collocation method with mesh refinement has been implemented successfully. Based on the type ecosystem, the code could be efficiently reused with type inference and multiple dispatch. The two main sub-parts of the package are direct collocation solver and mesh refinement. The former one is designed for transcribing the DOP into NLP with direct collocation methods and solving the NLP with a selected solver. The latter aims at building a new appropriate mesh according to error information given by the NLP solution. Two strategies, warm start and early termination for initial guess and NLP tolerance, are provided to lower the computation time. Comparing the running time of cold start and warm start, time could be reduced to 33% for TRP and 50% for HSS. With function $\max\left(10^{-3}N^{-1}, 10^{-8}\right)$ for TRP, the 5% running time could be reduced further. For HSS, function $\max\left(4 \times 10^{-3}N^{-2}, 10^{-8}\right)$ help decrease the computation time nearly 10%. Hence, both methods are applied efficiently.

There are three aspects worth considering for future work. The simple package for the direct collocation method with mesh refinement methods has been successfully built. However, the drawback is the lack of a complete Application Programming Interface (API). Users would struggle with bottom layer code, especially without professional knowledge of dynamic optimization. Hence, a user-friendly API should be designed. Then, there are many other collocation methods, such as orthogonal collocation. Other methods could also be implemented. There will be more options for the user and solver. Last but not

least, mesh refinement introduced above is focused on the p-method without changing the order of polynomials in each interval. Therefore, h-method could also be implemented and even a combination of them.

Bibliography

[1] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, pp. 849–904, 2017.

[2] D. J. Limebeer and A. V. Rao, "Faster, higher, and greener: Vehicular optimal control," *IEEE Control Systems*, vol. 35, pp. 36–56, 4 2015.

[3] C. L. Darby, D. Garg, and A. V. Rao, "Costate estimation using multiple-interval pseudospectral methods," *Journal of Spacecraft and Rockets*, vol. 48, pp. 856–866, 2011.

[4] 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*, vol. 41, 10 2014.

[5] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming, Third Edition*, 3rd ed. Philadelphi: Society for Industrial and Applied Mathematics, 2020.

[6] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, pp. 65–98, 2017.

[7] N. Health, "Why is programming language julia growing so fast and where is it going next?" 9 2018. [Online]. Available: https://www.techrepublic.com/article/julia-whats-next-for-pythons-fast-growing-programming-language-rival/

[8] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[9] A. V. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, 1 2010. [Online]. Available: https://www.researchgate.net/publication/268042868

[10] Y. Nie and E. C. Kerrigan, "Solving dynamic optimization problems to a specified accuracy: An alternating approach using integrated residuals," *IEEE Transactions on Automatic Control*, vol. XX, pp. 1–8, 2022.

[11] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, pp. 193–207, 1998.

[12] D. Garg, M. Patterson, W. W. Hager, A. V. Rao, D. A. Benson, and G. T. Huntington, "A unified framework for the numerical solution of optimal control problems using pseudospectral methods," *Automatica*, vol. 46, pp. 1843–1851, 11 2010.

[13] D. A. Benson, G. T. Huntington, T. P. Thorvaldsen, and A. V. Rao, "Direct trajectory optimization and costate estimation via an orthogonal collocation method," *Journal of Guidance, Control, and Dynamics*, vol. 29, pp. 1435–1440, 2006.

Appendix A

Appendix Title

A.1 Direct Collocation Method

A.1.1 Trapezoidal Collocation: Interpolation

we have an equation for system dynamics on the interval $t \in [t_k, t_{k+1}]$

$$\boldsymbol{f}(t) = \dot{\boldsymbol{x}}(t) \approx \boldsymbol{f}_k + \frac{\tau}{h_k}(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k). \tag{A.1}$$

To derive the function for state trajectories, both sides of the Equation 2.2 are integrated to achieve a quadratic expression:

$$\boldsymbol{x}(t) = \int \dot{\boldsymbol{x}}(t)\, \mathrm{d}\tau \approx \boldsymbol{c} + \boldsymbol{f}_k\tau + \frac{\tau^2}{2h_k}(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k). \tag{A.2}$$

Then, the value of $\boldsymbol{c}$ would be found by the value of $\boldsymbol{x}$ at τ, such that the final expression for the state could be obtained:

$$\boldsymbol{x}(t) \approx \boldsymbol{x}_k + \boldsymbol{f}_k\tau + \frac{\tau^2}{2h_k}(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k). \tag{A.3}$$

A.1.2 Hermite-Simpson Collocation: Integrals

The Simpson quadrature rule could be derived as follows: Considering a quadratic curve $v(t)$:

$$v(t) = A + Bt + Ct^2, \tag{A.4}$$

it would be integrated in the interval $[0, h]$ to obtain a quality x:

$$x = \int_0^h v(t)\ \mathrm{d}t, \tag{A.5}$$

$$x = Ah + \frac{1}{2}Bh^2 + \frac{1}{3}Ch^3. \tag{A.6}$$

The boundaries and midpoints are selected to confirm the exact expression of $v(t)$:

$$v(0) = v_L, \quad v(\frac{h}{2}) = v_M, \quad v(h) = v_R, \tag{A.7}$$

such that the coefficients of the quadratic curve $v(t)$ could be found:

$$A = v_L, \tag{A.8}$$

$$B = (-3V_L + 4v_M - v_R)/h, \tag{A.9}$$

$$C = (2V_L - 4v_M + 2v_R)/h^2. \tag{A.10}$$

Lastly, Equation A.6 could be simplified with the coefficients above:

$$x = \frac{h}{6}(v_L + 4v_M + v_R). \tag{A.11}$$

Hence, the approximation could be obtained for the integrals:

$$\int_{t_0}^{t_F} w(\tau)\ \mathrm{d}\tau \approx \sum_{k=1}^{N} \frac{h_k}{6}(w_k + 4w_{k+\frac{1}{2}} + w_{k+1}), \tag{A.12}$$

where the time interval k is denoted as $h_k = t_{k+1} - t_k$.

A.1.3 Hermite-Simpson Collocation: Interpolations

Firstly, each interval of control and dynamics are approximated with quadratic segments. From the collocation points, the values at both ends and middle could be known, such as $(t_L, \boldsymbol{u}_L)$, $(t_M, \boldsymbol{u}_M)$ and $(t_R, \boldsymbol{u}_R)$. The expression for $\boldsymbol{u}(t)$ passing through these points could be found:

$$\boldsymbol{u}(t) = \frac{(t-t_M)(t-t_R)}{(t_L-t_M)(t_L-t_R)}\boldsymbol{u}_L + \frac{(t-t_L)(t-t_R)}{(t_M-t_L)(t_M-t_R)}\boldsymbol{u}_M + \frac{(t-t_L)(t-t_M)}{(t_R-t_L)(t_R-t_M)}\boldsymbol{u}_R. \tag{A.13}$$

Specifically, given the condition that the second point is located at the middle of the interval k, we have further conditions, $h_k = t_{k+1} - t_k$, $t_{k+\frac{1}{2}} = \frac{1}{2}(t_k + t_{k+1})$, and $\tau = t - t_k$. Then, the equation for $\boldsymbol{u}(t)$ in the interval k could be simplified as:

$$\boldsymbol{u}(t) = \frac{2}{h_k^2}(\tau - \frac{h_k}{2})(\tau - h_k)\boldsymbol{u}_k - \frac{4}{h_k^2}(\tau)(\tau - h_k)\boldsymbol{u}_{k+\frac{1}{2}} + \frac{2}{h_k^2}(\tau)(\tau - \frac{h_k}{2})\boldsymbol{u}_{k+1}. \tag{A.14}$$

Similar to the piecewise quadratic interpolation of control trajectories, the expression for system dynamics $\dot{\boldsymbol{x}} = f(\cdot)$ in the interval k could also be achieved:

$$\boldsymbol{f}(t) = \dot{\boldsymbol{x}} = \frac{2}{h_k^2}(\tau - \frac{h_k}{2})(\tau - h_k)\boldsymbol{f}_k - \frac{4}{h_k^2}(\tau)(\tau - h_k)\boldsymbol{f}_{k+\frac{1}{2}} + \frac{2}{h_k^2}(\tau)(\tau - \frac{h_k}{2})\boldsymbol{f}_{k+1}. \tag{A.15}$$

Then, the expressions for the state trajectories could be derived by integrating the equation of $f(t)$ above:

$$\begin{aligned} x(t) &= \int \dot{\boldsymbol{x}}\mathrm{d}t \\ &= \int \left[\boldsymbol{f}_k + (-3\boldsymbol{f} + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1})(\frac{\tau}{h_k}) + (2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1})(\frac{\tau}{h_k})^2\right]\mathrm{d}t \\ &= c + (\boldsymbol{f}_k)\tau + \frac{1}{2h_k}(-3\boldsymbol{f} + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1})(\tau^2) + \frac{1}{3h_k^2}(2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1})\tau^3 \end{aligned} \tag{A.16}$$

Furthermore, considering the boundary condition $\boldsymbol{x}(t_k) = \boldsymbol{x}_k$ when $\tau = 0$, the constant c

could be computed. Finally, the expression for the state trajectory could be obtained:

$$\begin{aligned}\boldsymbol{x}(t) = x_k + (\boldsymbol{f}_k)\tau + \frac{1}{2h_k}(-3\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1})(\tau^2) \\ + \frac{1}{3h_k^2}(2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1})\tau^3.\end{aligned} \tag{A.17}$$

A.2 Solution: Error Analysis

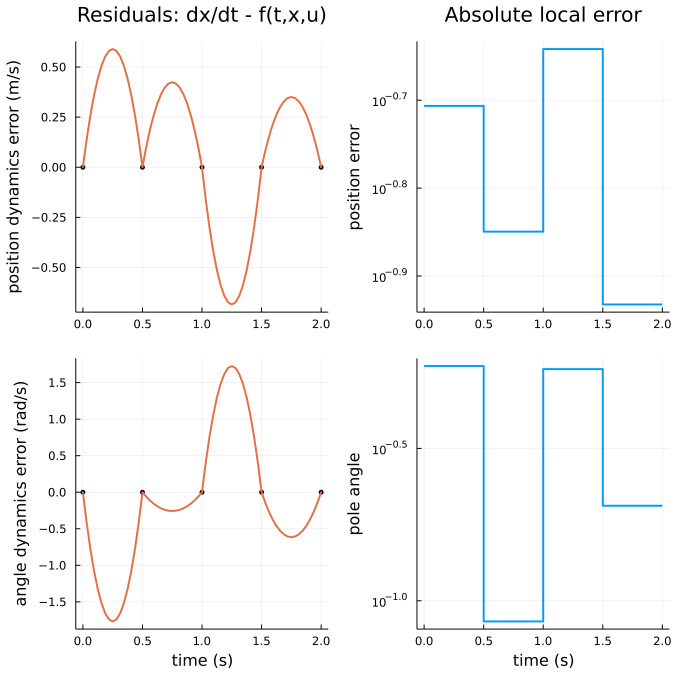

Figure A.1: Initial Error Distribution by TRPuD

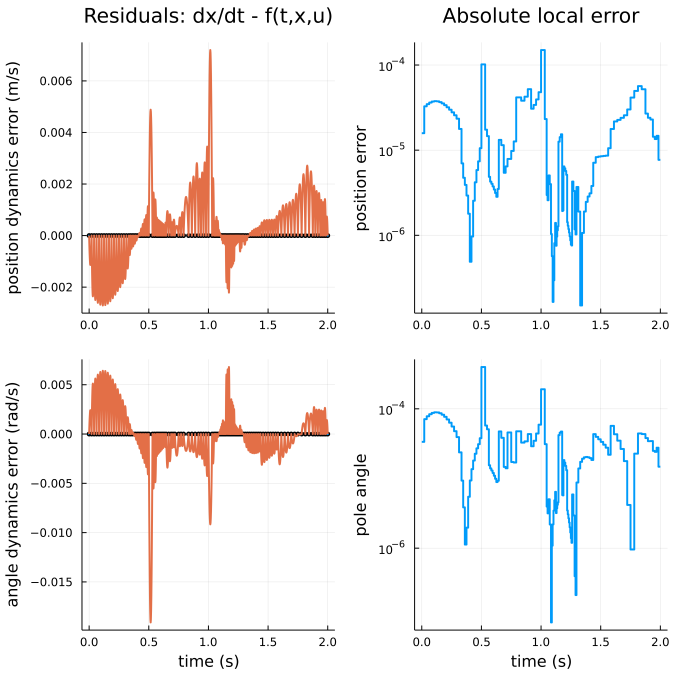

Figure A.2: Final Error Distribution by TRPuD

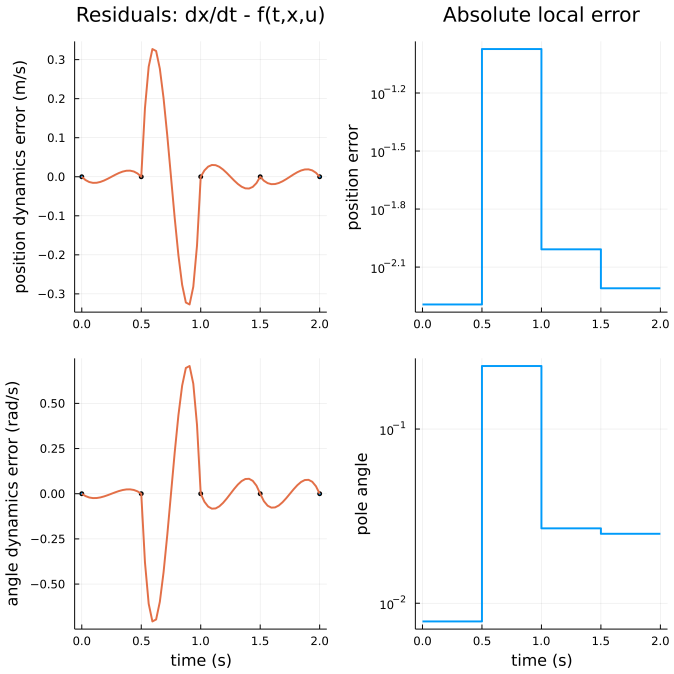

Figure A.3: Initial Error Distribution by HSSuD

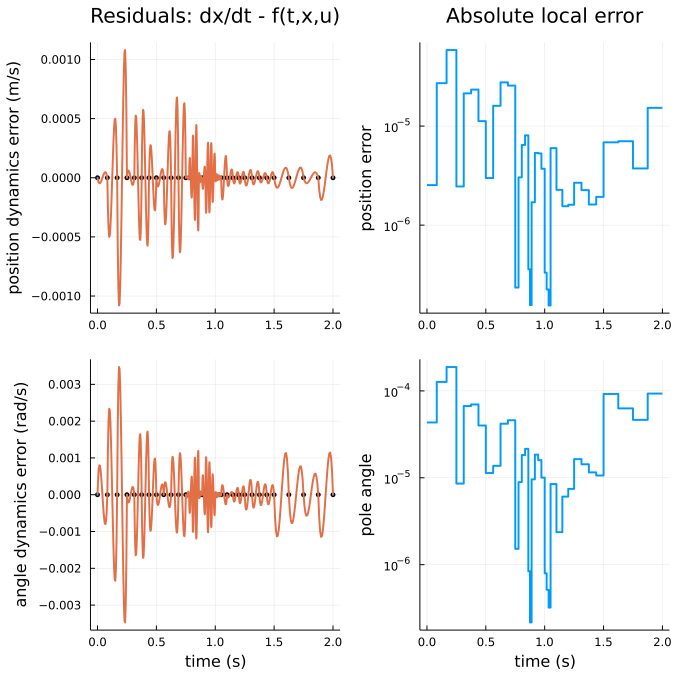

Figure A.4: Final Error Distribution by HSSuD

A.3 Solution: Warm Start

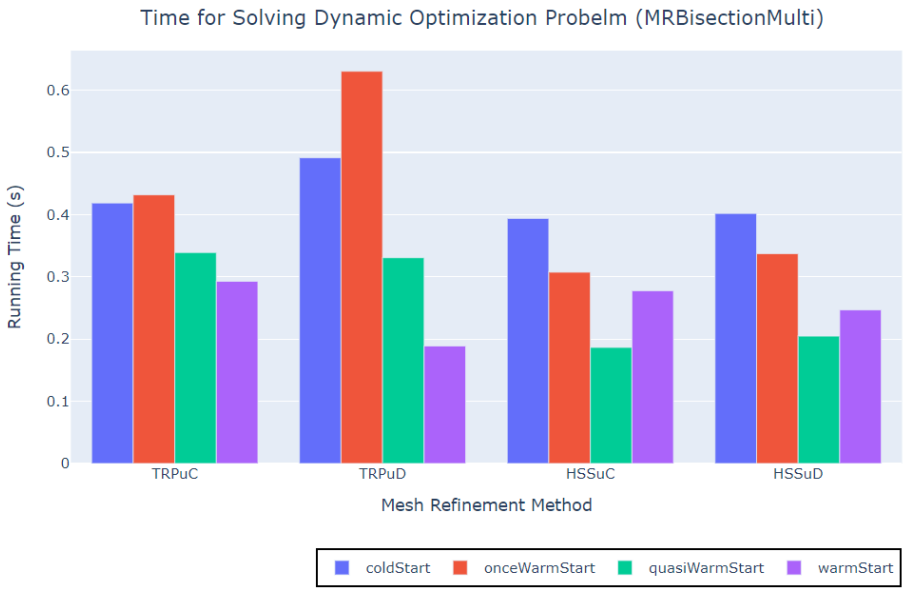

Figure A.5: Total Running Time Cost by Various Collocation Methods with Bisection(Multi)

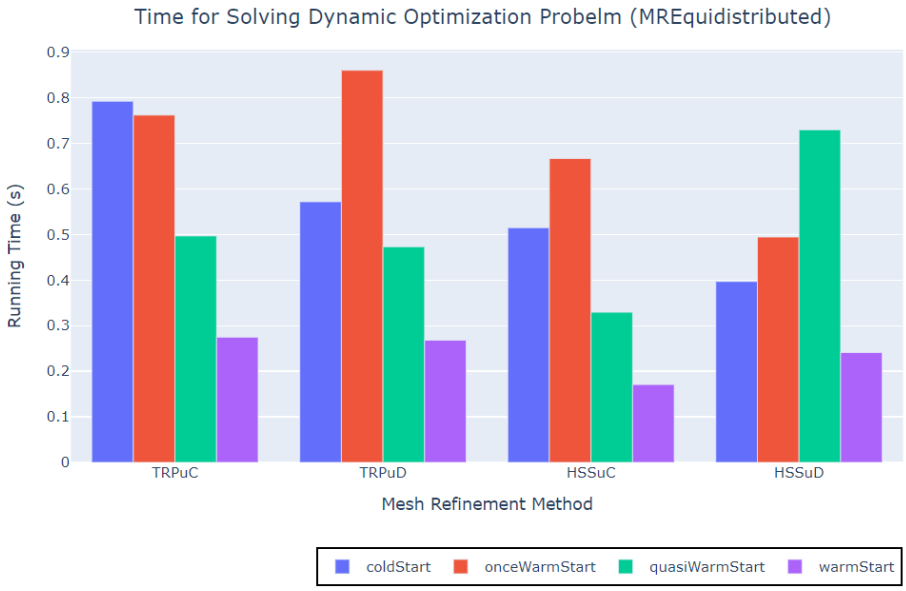

Figure A.6: Total Running Time Cost by Various Collocation Methods with Equidistributed Error

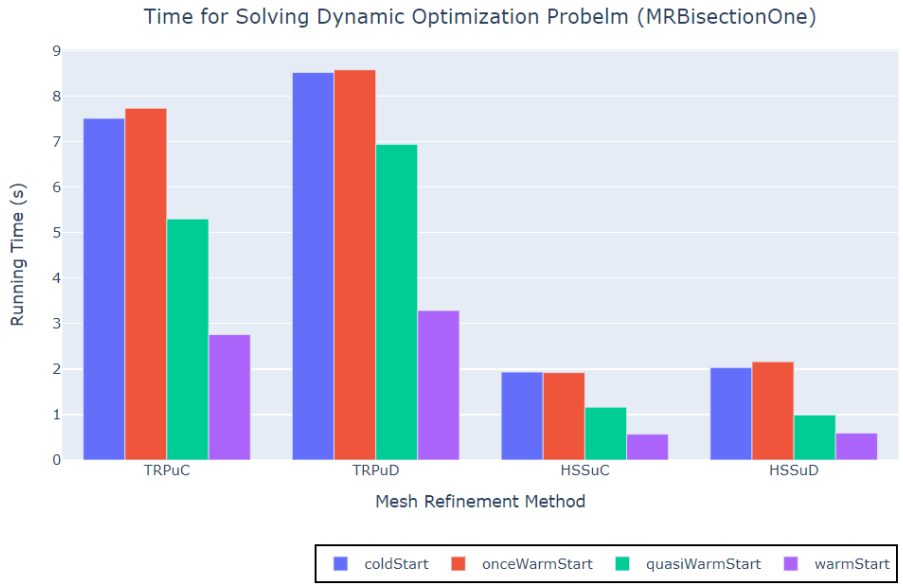

Figure A.7: Total Running Time Cost by Various Collocation Methods with Bisection (One)