Modeling and Simulation with Interval Uncertainties in Julia

Framework Development and Robotic Applications

Rasmus Kovács

LUND
UNIVERSITY

Department of Automatic Control

MSc Thesis TFRT-9999
ISSN 0280–5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2025

Abstract

In robotic systems, uncertainty in model parameters, such as joint stiffness, link lengths, or mass distributions, can significantly affect performance and precision. This thesis investigates the quantification and propagation of such uncertainties using the programming language Julia, with a focus on developing a systematic framework for uncertainty-aware modeling and simulation. It leverages the `ModelingToolkit.jl` package to build symbolic, component-based, acausal models and integrates several uncertainty quantification methods, including a Scanning method, Monte Carlo simulations, Polynomial Chaos Expansion, and Taylor Models, to analyze how uncertain parameters and initial conditions impact robotic accuracy.

A novel package, `IntervalSimulations.jl`, was developed to enable seamless uncertainty propagation within the symbolic modeling framework. It allows uncertain parameters to be defined directly within the models and automatically generates bounded trajectories and plots. The thesis compares the methods in terms of computational efficiency, accuracy, and ease of integration.

Case studies, including a three-degree-of-freedom robot model, a double pendulum, and a pair of scissors, are used to evaluate method performance and demonstrate how uncertainty affects robotic behavior. Results show that while Monte Carlo and scanning methods provide reliable estimates, Polynomial Chaos Expansion often offers improved computational efficiency. Taylor Models, while theoretically rigorous, showed limited robustness in practice.

The results also highlight how uncertainty analysis can enhance robust robot design by pinpointing the parameters that most significantly influence performance. This insight is valuable both during the design phase and when programming robots for tasks requiring high precision. By quantifying how accurately a robot can follow a desired trajectory under uncertainty, the developed tools in this thesis support more informed decisions in robotic planning and control.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Björn Olofsson from Lund University and Philip Olhager from Cognibotics, for their invaluable support and guidance throughout this thesis. Their consistent feedback and willingness to discuss and explore ideas have played a crucial role in helping me make progress and keep the project moving forward.

I would also like to thank my partner for her patience and for tolerating countless hours of me talking about these topics over the past few months. Her insightful yet seemingly ”stupid” questions have often sparked new ideas and directions for exploration.

Lastly, I would like to thank Cognibotics for giving me the opportunity to carry out my thesis in a dynamic and technically inspiring environment. Being part of the company provided valuable perspectives, from hallway discussions to fika conversations, that helped shape my understanding of the field. I am grateful for the trust placed in me and for the opportunity to contribute to work that closely aligns with my interests. I hope this thesis can be of use in return.

Contents

1

Introduction

1.1 Introduction

Modeling and simulation are essential tools in robotics and many other engineering domains. By formulating a system of equations that describe a robot's behavior, and then solving these equations under different conditions, engineers can predict motion, evaluate performance, and design control strategies without physical prototypes. However, all models are simplifications of reality. Parameters such as link lengths, joint stiffnesses, and damping coefficients are rarely known exactly, they are estimated, measured with uncertainty, or subject to manufacturing tolerances and environmental variations.

These uncertainties are not trivial. Even small deviations in model parameters can result in noticeable differences in predicted motion, applied forces, or the position of the end-effector. In applications where precision is important, such as manufacturing, medical robotics, or quality assurance, these discrepancies can reduce the usefulness of simulation results. Understanding how uncertainty affects system behavior is therefore essential for validating models, designing robust control strategies, and improving the reliability of robotic systems.

In robotic manipulators, uncertainties in joint compliance, link flexibility, and actuation can significantly affect dynamic behavior. Parameters such as joint stiffness are often uncertain as a result of estimation errors or structural variation, leading to deviations in predicted motion and forces [Nazari and Notash, 2015; Busch et al., 2022]. These effects are especially important in precision applications, motivating the need for tools that can propagate such uncertainties through simulation.

Traditional modeling and simulation tools typically assume deterministic values for all parameters. In Julia, the `ModelingToolkit.jl` [Ma et al., 2021] package provides a powerful symbolic modeling language that can generate differential-algebraic equations and interface seamlessly with other components of the SciML ecosystem. However, it is primarily designed for deterministic simulations.

External tools for uncertainty quantification, such as Dakota [Adams et al., 2025] and UQ-PyL [Wang et al., 2016], provide established workflows for propagating parameter uncertainty through numerical models. Dakota is language-agnostic

and can interface with models written in Julia via wrapper scripts or file-based communication, but this typically requires additional setup and lacks deep integration with symbolic modeling frameworks. UQ-PyL, by contrast, is a Python-native package and more closely resembles the embedded workflow used in this thesis. However, it operates on numerical models and does not integrate with symbolic modeling libraries in Python. In both cases, users are responsible for managing model execution, parameter sampling, and result extraction manually. These workflows, while flexible, offer limited composability with symbolic modeling tools like `ModelingToolkit.jl`, motivating the development of a native uncertainty-aware simulation framework within Julia.

In response, this thesis introduces `IntervalSimulations.jl`, a Julia package that extends the `ModelingToolkit.jl` framework to support simulation and uncertainty propagation in models with interval-valued parameters. With this toolkit, users can define uncertain parameters directly in their model and initiate uncertainty analysis with a single function call. Although the underlying methods treat the model as a black box, the user interface is fully embedded in the symbolic modeling workflow, enabling uncertainty propagation to be applied with minimal effort and without additional configuration.

The package supports four commonly used methods for uncertainty analysis: scanning, based on brute-force evaluation over a parameter grid [Wang and Yang, 2021]; Monte Carlo sampling, which estimates statistics over randomly drawn inputs [Wang and Yang, 2021]; Polynomial Chaos Expansion (PCE), which uses orthogonal polynomials to approximate uncertainty propagation [Wang and Yang, 2021]; and Taylor Models, implemented via `ReachabilityAnalysis.jl`, which compute validated enclosures using high-order polynomial expansions with remainder bounds [Bogomolov et al., 2019].

Their inclusion was initially motivated by the comparative study of Wang and Yang [Wang and Yang, 2021], where all four methods were applied in a robotics context. While scanning is less commonly used outside that study, it provides a consistent baseline for comparison. PCE and Monte Carlo methods have been applied across domains such as energy systems [Xu et al., 2018], mechanical simulations [Manfredi, 2025; Ranftl and Linden, 2021], and safety-critical applications [Halder and Bhattacharya, 2010; Carney et al., 2020; Srikanthakumar and Chen, 2015]. Taylor models, in turn, are widely used in validated simulation tools and reachability analysis [Chen, 2015; Berz and Makino, 2017; Althoff, 2015].

In this thesis, these four methods are evaluated not only for their theoretical properties, but also for their practical behavior when applied within a symbolic modeling workflow in Julia. This includes benchmarking their computational performance, comparing the tightness of the resulting uncertainty bounds, and assessing their ease of use and integration in typical robotic modeling scenarios. The goal is to provide insight into both the quantitative trade-offs and the practical utility of each method, with a particular focus on how well they support uncertainty-aware simulation in robotic systems.

1.2 Aim of the Thesis

In the perspective of the previous work and available tools described in Section 1.1, this thesis aims to explore how uncertainty in model parameters affects the accuracy and reliability of robotic simulations performed in Julia. The focus is on understanding how symbolic modeling and numerical methods can be combined to propagate uncertainty efficiently and meaningfully through robotic systems.

This work is guided by two research questions:

1. How can the programming language Julia be used to systematically analyze how uncertainties in model parameters affect robotic accuracy?
2. What insights into robotic performance and precision can be drawn from uncertainty analyses conducted in Julia?

These questions are addressed through simulation-based experiments using the custom-developed package `IntervalSimulations.jl`, applied across several representative robotic systems.

1.3 Introductory Example

To illustrate the concept of uncertainty propagation in robotic systems, consider the scissors shown in Figure 1.1. While simple, this mechanical system is structurally similar to tools used in robot-assisted surgery, where precision is critical. Uncertainties in spring constants, link lengths, and joint properties can make it difficult to predict the exact behavior of the mechanism. Instead of a single deterministic trajectory, the Tool Center Point (TCP), the cutting edge of the scissors, occupies a range of possible positions over time.

This red shaded region reflects the effect of uncertain parameters on a single snapshot in time. Even a simple mechanism like a pair of scissors can exhibit considerable variability when these uncertainties are taken into account.

Figure 1.2 shows how this uncertainty evolves as the scissors close. The simulation begins with the blades fully open, and the TCP progressively moves leftward toward a closed configuration. Each red rectangle marks the reachable set of TCP positions at a specific time. The black outlines mark selected time points to aid interpretation. As time advances, the envelopes widen, illustrating how uncertainty accumulates over the course of the motion. This highlights why understanding and quantifying parameter uncertainty is vital in applications requiring precise control, such as surgical robotics.

1.4 Outline

The remainder of the thesis is structured as follows:

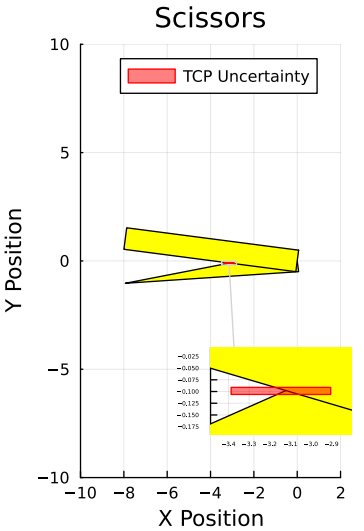


Figure 1.1 Illustration of uncertainty for a pair of scissors, with scaled axes to reflect the geometry. The red region shows the range of possible TCP positions at one time step.

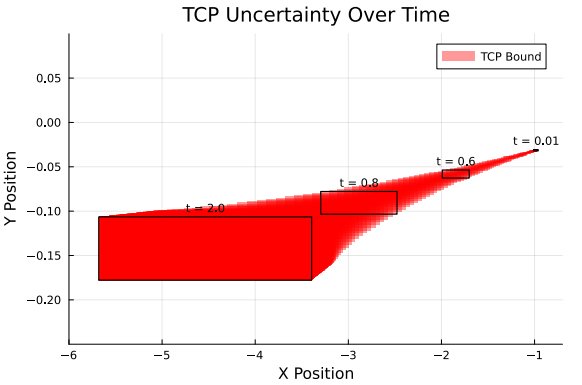


Figure 1.2 TCP position uncertainty during closing motion. Each red region shows the reachable TCP position at a given time step. Time progresses from right to left. Selected rectangles are labeled to highlight key moments.

- **Chapter 2 – Background and Theory:** Introduces core concepts such as ordinary and differential-algebraic equations, uncertainty propagation techniques, and their theoretical foundations.
- **Chapter 3 – Implementation:** Describes the design of my package for interval simulations, the modeling interface, and integration with Julia's SciML ecosystem.
- **Chapter 4 – Robotic Models:** Defines the dynamical systems used in the study, including a first-order test case, a double pendulum, a pair of scissors, and a 3-DOF robotic arm.
- **Chapter 5 – Methods:** Outlines the evaluation strategy used to assess the accuracy, performance, and usability of each uncertainty quantification method.
- **Chapter 6 – Results:** Presents simulation results, compares methods, and visualizes how uncertainties affect system trajectories and derived quantities.
- **Chapter 7 – Discussion:** Interprets the results, highlights method-specific strengths and limitations, and provides guidance on practical use cases.
- **Chapter 8 – Conclusion:** Summarizes the contributions, answers the research questions, and outlines opportunities for future work.

2

Theoretical Background

2.1 Scope and Structure of the Theoretical Background

The previous chapter introduced the motivation and objectives of this thesis, emphasizing the need for rigorous uncertainty propagation methods compatible with the Julia programming language and the SciML ecosystem. To provide the necessary context, this chapter introduces the key theoretical concepts and methods used throughout this work.

It includes:

- A brief motivation for using Julia and the MTK package for robotic simulations.
- An introduction to interval arithmetic and the challenges posed by the wrapping effect.
- Four primary methods for uncertainty propagation: the scanning method, Taylor models, Polynomial Chaos Expansions (PCE), and Monte Carlo (MC) simulations.

These components provide the conceptual framework for the implementations and evaluations presented in later chapters.

2.2 Motivation for Using Julia

Julia is a high-level programming language designed for numerical computing. It enables users to write generic and abstract code that closely resembles mathematical notation, while achieving performance comparable to low-level languages like C and Fortran [Bezanson et al., 2017]. This makes it particularly well suited for applications in scientific computing and control.

Julia includes a well-developed set of packages for scientific computing, several of which are central to this thesis. The most important ones include:

- ModelingToolkit.jl [Ma et al., 2021] for modeling,
- DifferentialEquations.jl [Rackauckas and Nie, 2017] for solving differential equations,
- ReachabilityAnalysis.jl [Bogomolov et al., 2019] for interval-based uncertainty propagation using Taylor models.
- IntervalArithmetic.jl [Sanders and Benet, 2014] for rigorous interval computations.

2.3 Modeling Dynamical Systems with ModelingToolkit.jl

ModelingToolkit.jl is a high-level symbolic modeling framework designed for constructing and analyzing dynamical systems. It supports acausal modeling, allowing users to define systems in terms of parameters, variables, and equations, without prescribing a computational order. The framework automatically restructures the model for simulation, handling tasks such as index reduction for differential-algebraic equations (DAEs), introduction of auxiliary variables, and insertion of algebraic constraints to improve numerical stability.

ModelingToolkitStandardLibrary.jl [SciML contributors, 2025] extends the framework with a library of modular and reusable components, enabling users to build large-scale models through component-based composition. While symbolic preprocessing is managed by ModelingToolkit.jl, numerical simulation is performed by DifferentialEquations.jl [Rackauckas and Nie, 2017].

Although MTK offers a convenient and powerful interface for deterministic modeling, its current integration with validated or interval-based uncertainty propagation is limited. This gap motivates the exploration of additional methods described in later sections.

A key feature of MTK is how it distinguishes between equations that must be solved and those that can be observed. For example, consider the system:

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = x(t), \quad x(0) = 0, \tag{2.1}$$

$$y(t) = 2x(t). \tag{2.2}$$

Here, the first equation defines the dynamics of the system and is treated as a differential constraint. The second equation, although not needed to compute the system state, defines an additional output and is recorded as an *observed equation*. When solving the system, the solver only returns values for $x(t)$, but the framework retains the relationship for $y(t)$ and allows it to be computed post hoc.

`ModelingToolkit.jl` can also be extended with domain-specific libraries such as `Multibody.jl` [JuliaSim contributors, 2025], which is particularly useful for modeling mechanical systems like robots. It provides reusable components including joints, rigid bodies, and translations, allowing users to assemble complex multibody systems in a modular and physically meaningful way. This makes it well-suited for constructing realistic robotic models with minimal manual effort [JuliaSim contributors, 2025].

2.4 DAEs and ODEs, Numerical Stability

Many dynamic systems can be formulated either as Ordinary Differential Equations (ODEs) or as Differential-Algebraic Equations (DAEs). ODEs have the general form

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f(\mathbf{x}, \mathbf{u}, t),$$

where $\mathbf{x}$ denotes the vector of state variables, and $\mathbf{u}$ represents external inputs or control signals. All equations in this form involve only time derivatives of the state variables.

In contrast, DAEs include algebraic constraints in addition to differential equations. A general semi-explicit DAE system can be written as

$$\begin{aligned} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} &= f(\mathbf{x}, \mathbf{z}, \mathbf{u}, t), \\ 0 &= g(\mathbf{x}, \mathbf{z}, \mathbf{u}, t), \end{aligned}$$

where $\mathbf{z}$ are algebraic variables that do not have explicit time derivatives. DAEs arise naturally in mechanical systems with constraints, such as a pendulum or a robotic arm.

An important property of DAEs is their *index*, which informally measures the number of times the algebraic constraints must be differentiated to reduce the system to an ODE [Brenan et al., 1996]. Index-1 DAEs require minimal manipulation and can typically be solved directly, while higher-index systems often require index reduction techniques. The Pantelides algorithm [Pantelides, 1988], widely used for symbolic index reduction, systematically identifies which algebraic equations in a DAE system must be differentiated to reduce the system's index. This algorithm is implemented in `ModelingToolkit.jl` as part of its structural simplification workflow.

Although index reduction enables numerical integration, it can also introduce numerical drift, especially in cases where the original algebraic constraints are no longer explicitly enforced, as seen in pendulum models [Safdarnejad et al., 2015].

2.5 Interval Arithmetic

Interval arithmetic provides a mathematically rigorous framework for representing and manipulating uncertain numerical quantities. This section offers a brief overview of the essential concepts, based primarily on the presentation in *Introduction to Interval Analysis* [Moore et al., 2009].

2.5.1 Definition

A closed interval $[a,b]$ is defined as

$$[a,b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}. \tag{2.3}$$

2.5.2 Basic Operations

Let X and Y be two intervals. Interval arithmetic is defined by:

$$X + Y = \{x + y \mid x \in X, y \in Y\}, \tag{2.4}$$
$$X - Y = \{x - y \mid x \in X, y \in Y\}, \tag{2.5}$$
$$X \cdot Y = \{xy \mid x \in X, y \in Y\}, \tag{2.6}$$
$$X / Y = \{x/y \mid x \in X, y \in Y\}, \quad \text{if } 0 \notin Y. \tag{2.7}$$

2.5.3 Properties and Applications

A straightforward strategy for simulating systems with uncertain or varying inputs is to apply interval arithmetic directly within a numerical solver. This ensures that the computed solutions rigorously enclose all possible trajectories, given the initial interval uncertainties.

However, repeated interval computations can lead to significant overestimation. This issue arises because of dependencies between variables being ignored during arithmetic operations, resulting in overly conservative bounds, a phenomenon known as the *wrapping effect*.

2.5.4 The Wrapping Effect

One of the fundamental challenges in interval arithmetic is the *wrapping effect*. This phenomenon arises when dependencies between variables are lost during repeated interval operations. In particular, when the same variable appears multiple times in a computation, interval arithmetic treats each occurrence as independent. This leads to overly conservative enclosures that grow rapidly with expression complexity and reuse.

To illustrate this, consider evaluating the function

$$f(x) = x^3 - 2x^2 + x$$

over the interval $x \in [0,1]$. The function can be decomposed as

$$f(x) = f_1(x) + f_2(x) + f_3(x), \quad \text{where} \quad f_1(x) = x^3, \quad f_2(x) = -2x^2, \quad f_3(x) = x.$$

Applying interval arithmetic to each subfunction over the same domain $x^I = [0,1]$ and summing the results yields

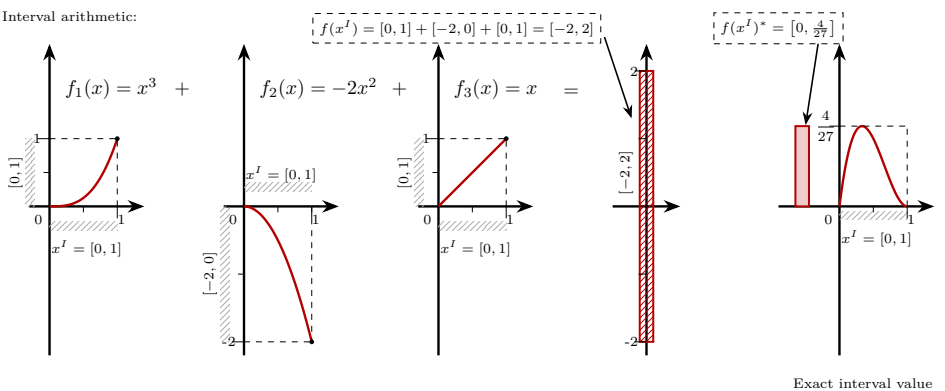
$$f(x^I) = [0,1] + [-2,0] + [0,1] = [-2,2],$$

which is significantly wider than the actual range of $f(x)$ over the same interval. In fact, evaluating the composite function directly and computing its minimum and maximum yields the much tighter bound

$$f(x^I)^* = \left[0, \frac{4}{27}\right] \approx [0, 0.148].$$

Figure 2.1 illustrates this effect. Subfigure 2.1(a) shows the interval-based evaluation applied independently to each term, resulting in a severe overapproximation due to the wrapping effect. Subfigure 2.1(b) shows the true image of the function over the interval. The discrepancy highlights how variable dependency loss across multiple operations leads to excessive wrapping.

This example and figure are inspired by Wang et al. [Wang and Yang, 2021], who use a similar construction to highlight the limitations of naive interval evaluation methods.



(a) Evaluating using interval arithmetic (b) Evaluating directly

Figure 2.1 Illustration of the wrapping effect using interval arithmetic (left) and the exact range (right). Inspired by Wang et al. [Wang and Yang, 2021].

2.5.5 Integration of interval arithmetic in Julia

The Julia package `IntervalArithmetic.jl` implements interval arithmetic, adhering to the IEEE standard for interval arithmetic. This package enables users to work intuitively with interval numbers directly in Julia, facilitating rigorous computations and uncertainty propagation.

2.6 Scanning Method

One of the most straightforward approaches to uncertainty propagation is the *Scanning method* [Wang and Yang, 2021]. In this approach, each uncertain parameter is discretized over a finite set of values, typically using a uniform grid. A deterministic simulation is then performed for every combination of these grid points. The resulting trajectories are collected to estimate upper and lower bounds on the system's behavior. This approach is non-intrusive, as it relies on repeated calls to the existing numerical solver without modifying the model equations.

Figure 2.2 illustrates the scanning method for a simple first-order system with uncertainty in its initial condition. Each thin gray curve represents a simulation run from a different initial value. The red curves trace the pointwise minimum and maximum over time, forming an outer envelope that bounds the reachable trajectories.

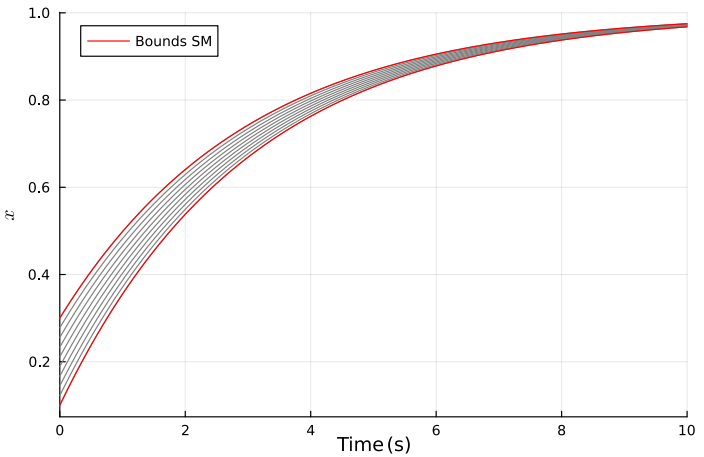


Figure 2.2 Visualization of the scanning method for a first-order system with uncertainty in the initial condition of x. A grid of initial values is simulated, and the red curves represent the envelope of minimum and maximum responses over time.

The main advantage of this method is that it computes tight bounds over the discrete grid without over-approximation. Since all grid points are explicitly enu-

merated, the method avoids the wrapping effects that typically arise in interval or set-based approaches. However, this guarantee only applies to the finite set of sampled points and does not extend to the full continuous parameter space.

A major limitation is the exponential growth in the number of required simulations. For d uncertain parameters, each divided into n grid points, the method requires n^d simulations. This quickly becomes intractable for even moderately high dimensions or fine resolutions.

Evaluating the system at the boundary values of the uncertain parameters, corresponding to a grid with two points per dimension, is the minimal form of the scanning method. While this strategy captures the corners of the parameter space, it is generally insufficient for nonlinear systems, where intermediate parameter values may lead to more extreme system responses. As a result, low-resolution grids may underestimate the true reachable set, even though all boundary combinations are included. Accurate bounding in nonlinear settings typically requires finer sampling of the interior parameter space.

Despite its simplicity and computational cost, the scanning method is often used as a reference solution [Wang and Yang, 2021]. In low-dimensional problems, it provides a valuable benchmark for evaluating more advanced uncertainty quantification techniques.

2.7 Monte Carlo

A widely used method for uncertainty propagation is the *Monte Carlo* (MC) method, which involves sampling the uncertain parameter space and simulating the system dynamics for each sample. Its generality and ease of implementation make it a common benchmark for comparing more advanced techniques [Wang and Yang, 2021; Carney et al., 2020; Halder and Bhattacharya, 2010]. In this thesis, where the uncertain parameters are defined over uniformly distributed intervals, each Monte Carlo trial corresponds to a random draw within these intervals, forming a unique parameter combination. As with scanning, Monte Carlo is a non-intrusive method: it treats the model as a black box and requires no changes to the governing equations or solver.

Traditionally, Monte Carlo is employed to compute statistical moments such as the mean and variance of system outputs. However, in this thesis, the focus is on estimating bounding envelopes for the system trajectories. Specifically, for each state variable x and time t, the minimum and maximum values across all simulation realizations are recorded to approximate the lower and upper bounds. These empirical bounds provide an intuitive estimate of the system's behavior under uncertainty, but they will underestimate the true reachable set since only a finite number of samples are used. An illustration of the method is shown in Figure 2.3, where 10 samples are used to estimate the bounds of the first angle of a double pendulum, described in more detail in Section 4.3.

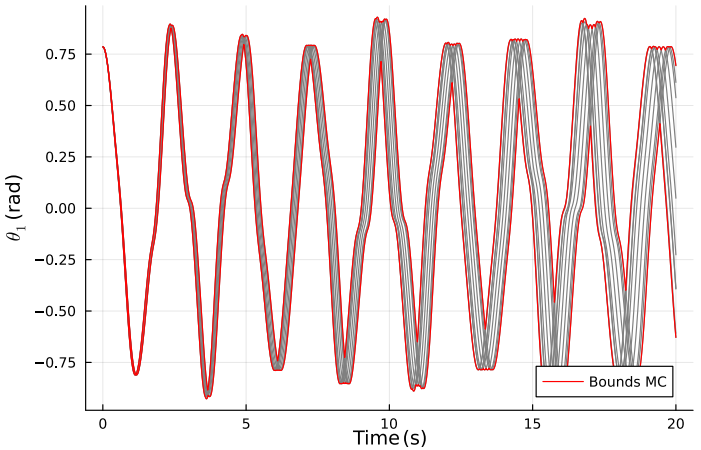


Figure 2.3 Illustration of the Monte Carlo method. Ten random samples are used to estimate the bounds of the first angle, θ_1, of a double pendulum. The individual trajectories are shown in grey, while the estimated bounds are highlighted in red. As seen, the estimated bounds depend on the spread of the sampled trajectories and may miss the true extremes when using few samples.

Since Monte Carlo sampling is inherently stochastic, it provides a statistical approximation of the system's behavior under uncertainty. The resulting bounds are typically an underapproximation of the true reachable set, as the method only explores a finite number of trajectories within the parameter space. While the probability of capturing near-extreme values increases with the number of samples, there is no guarantee that the true maximum or minimum values will be observed. Without additional assumptions about the system dynamics, it is generally difficult to quantify how close the observed bounds are to the true worst-case outcomes.

In specific applications, methods based on safe and unsafe region classification can provide probabilistic estimates of failure likelihood [Carney et al., 2020], but in a general setting without a predefined critical threshold, such classifications are not directly applicable.

Despite these limitations, Monte Carlo remains a valuable baseline method for uncertainty propagation, against which more advanced techniques can be compared.

2.8 Taylor Models

Taylor models provide a method for numerically integrating nonlinear dynamical systems under uncertainty by combining high-order polynomial approximations with rigorously bounded error terms [Berz and Makino, 2017]. They form the basis of one of the validated reachability algorithms used in this thesis. Un-

like scanning, Monte Carlo, and Polynomial Chaos Expansion (PCE), Taylor model methods are intrusive, they require symbolic transformations and dedicated integration schemes that differ from standard numerical solvers. This section briefly outlines the key ideas behind Taylor model-based integration, as implemented in the `ReachabilityAnalysis.jl` (`RA.jl`) package, with a focus on how such models enable mathematically sound uncertainty propagation.

Taylor Series and Automatic Differentiation

At the core of the Taylor model approach lies the classical Taylor series. For a sufficiently smooth function $f(t)$, its expansion around a point t_0 takes the form

$$f(t) = f_0 + f_1 \cdot (t - t_0) + f_2 \cdot (t - t_0)^2 + \cdots, \tag{2.8}$$

where $f_k = \frac{1}{k!} \frac{\mathrm{d}^k f}{\mathrm{d}t^k}(t_0)$ denotes the normalized k-th derivative.

In practice, these coefficients can be computed efficiently using automatic differentiation [Tucker, 2011]. The `TaylorSeries.jl` package provides tools for constructing such expansions in both one and several variables [Benet and Sanders, 2019]. As described in its documentation [Benet and Sanders, n.d.], multivariate expansions are represented using homogeneous polynomials indexed by multi-indices, making the approach well suited for high-order approximations of ODE solutions. These expansions form the polynomial part of the Taylor models used in the reachability analysis.

Taylor Models: Polynomial and Remainder

A Taylor model augments a standard Taylor polynomial with a rigorous bound on the approximation error. For a function f defined on an interval $[a,b]$, a Taylor model of order n consists of a pair (P,Δ), where

$$f(x) \in P(x) + \Delta \quad \text{for all } x \in [a,b],$$

with $P(x)$ being a polynomial of degree n approximating f, and Δ an interval that captures all errors because of truncation and numerical rounding. Figure 2.4 illustrates this concept for the nonlinear function

$$f(x) = x(x-1.1)(x+2)(x+2.2)(x+2.5)(x+3)\sin(1.7x+0.5),$$

showing 6th- and 7th-order Taylor models over the domain $[-0.5, 1.0]$. Each model computes a polynomial enclosure that tightly bounds the true function within a validated remainder interval.

The code used to construct the Taylor models in Figure 2.4 is adapted from the official `TaylorModels.jl` documentation [Benet and Sanders, 2025], while the figure itself was generated independently.

This construction enables validated computations on uncertain or nonlinear functions, ensuring that all possible values of f over the domain are rigorously

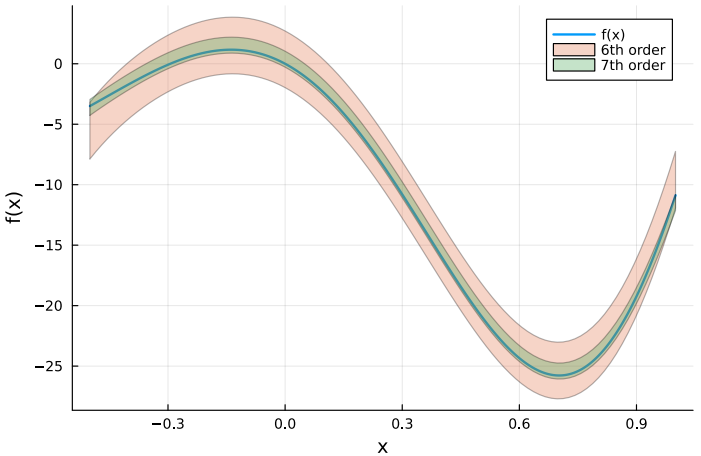


Figure 2.4 Taylor models of 6th and 7th order approximating a nonlinear function on the interval $[-0.5, 1.0]$. The polynomial approximation is shown together with the interval remainder, forming a validated enclosure of the function. The bounds differ notably between the two cases because the remainder is computed as a worst-case deviation across the entire domain.

enclosed. In Julia, the `TaylorModels.jl` package [Benet and Sanders, 2025] implements Taylor models by combining high-order polynomial expansions from `TaylorSeries.jl` [Benet and Sanders, 2019] with rigorous interval bounds from `IntervalArithmetic.jl` [Sanders and Benet, 2014]. This makes the data structure suitable for applications in reachability analysis and validated ODE integration.

Validated Integration Using TMJets21b

This thesis uses the TMJets21b algorithm from the `RA.jl` package to compute rigorous enclosures of trajectories for nonlinear systems under uncertainty. The method performs validated numerical integration using high-order Taylor expansions, implemented via the `TaylorIntegration.jl` package [Pérez-Hernández and Benet, 2019].

The integration is based on the approach by Bünger [Bünger, 2020], which uses Taylor models to propagate enclosures of the solution over time, ensuring rigorous bounds on the reachable set. The algorithm automatically handles remainder bounding, validation, and convergence checks internally. The user specifies the Taylor expansion orders via the parameters `orderQ` (for state variables) and `orderT` (for time), which together define the structure and precision of the computed Taylor models.

This validated integration guarantees existence and uniqueness of the solution

within the computed bounds.

Error Control and Adaptive Tolerance

The TMJets21b integrator supports adaptive error control, which can be enabled by setting the `adaptive` flag to `true`. When a validation step fails to produce a sufficiently tight enclosure, the algorithm attempts to recover by adjusting two key parameters:

- The integration step size δt is reduced to improve the accuracy of the Taylor expansion
$$\delta t \leftarrow \delta t \cdot 0.1^{1/\texttt{orderT}}.$$
- The absolute tolerance `abstol`, which controls the allowable size of the validated remainder, is tightened
$$\texttt{abstol}_{\text{new}} = \frac{\texttt{abstol}}{10}.$$

These adjustments are applied iteratively until a valid enclosure is obtained or a user-defined minimum tolerance (`minabstol`) is reached. After a successful step, the integrator may increase `abstol` again, up to its original value, to improve efficiency in later steps.

Illustrative Example

To illustrate the core idea of Taylor model propagation, consider the scalar ODE

$$\frac{d}{dt}x(t) = \sin(x), \quad x(0) \in 1 \pm 0.1. \tag{2.9}$$

To approximate the solution for small t, we take a first-order forward step

$$x(t) \approx x_0 + \sin(x_0)t, \tag{2.10}$$

where $x_0 = P_0 + \Delta_0$, with $P_0 = 1$, $\Delta_0 \in [-0.1, 0.1]$. Since $\sin(x_0)$ is nonlinear in Δ_0, we approximate it using a first-order Taylor expansion around P_0

$$\sin(x_0) \approx \sin(P_0) + \cos(P_0)\Delta_0. \tag{2.11}$$

Substituting into (2.11) expression gives the enclosure

$$x(t) \in P_0 + \sin(P_0)t + (1 + \cos(P_0)t)\Delta_0. \tag{2.12}$$

Evaluating at $t = 0.1$, using $\sin(1) \approx 0.8415$, $\cos(1) \approx 0.5403$, we find

$$x(0.1) \in [0.9788, 1.1896]. \tag{2.13}$$

This example shows how Taylor models combine polynomial approximation and remainder bounding to enclose the solution over time, even in the presence of initial uncertainty.

2.9 Polynomial Chaos Expansion

Polynomial Chaos Expansion (PCE) is a non-intrusive method for uncertainty propagation [Wang and Yang, 2021], meaning it does not require modifications to the underlying governing equations or numerical solvers. Instead of reformulating the model, multiple deterministic simulations are performed at carefully selected points in the uncertain parameter space. The resulting data are used to construct a polynomial approximation of the system output. This black-box property allows PCE to be integrated seamlessly with modeling frameworks such as MTK, where model definitions remain unchanged and standard numerical solvers for DAEs can be directly employed.

This functionality is not currently available in any existing Julia package and is therefore implemented as part of this thesis.

Mathematical Theory

Following the presentation in Wang and Yang [Wang and Yang, 2021], with minor notational adjustments, consider a system output function

$$y(\bar{\xi}, t_k),$$

where $\bar{\xi} \in [-1,1]^d$ denotes a vector of d uncertain parameters, mapped to the standard interval $[-1,1]$ by an affine transformation if necessary, and t_k represents a fixed time instant.

In the PCE framework, the system output $y(\bar{\xi},t_k)$ is represented as an infinite series expansion in terms of orthogonal polynomial basis functions. In practical implementations, the series is truncated at a maximum total polynomial order p, yielding the approximation

$$y(\bar{\xi}, t_k) \approx a_0(t_k) + \sum_{i=1}^{S-1} a_i(t_k)\, \psi_{\boldsymbol{\alpha}_i}(\bar{\xi}), \quad \bar{\xi} \in [-1,1]^d, \tag{2.14}$$

where $\psi_{\boldsymbol{\alpha}_i}(\bar{\xi}) = \psi_i(\bar{\xi})$ are multivariate orthogonal polynomial basis functions, and $a_i(t_k)$ are the corresponding expansion coefficients at time t_k. The coefficient $a_0(t_k)$ represents the expected value of $y(\bar{\xi},t_k)$ under the assumption of a uniform distribution of the uncertain parameters, whereas the remaining terms capture the effects of parameter variations.

The total number of terms S in the truncated expansion depends on both the polynomial order p and the number of uncertanties d, and is given by

$$S = \frac{(d+p)!}{d!\,p!}, \tag{2.15}$$

which corresponds to the number of multivariate polynomials whose total degree does not exceed p.

Polynomial Basis: Legendre Polynomials

For uncertainties defined over bounded intervals, Legendre polynomials are a natural choice of basis because of their orthogonality on the domain $[-1,1]$. The univariate Legendre polynomials $\phi_n(\xi)$ satisfy the recurrence relation [Wang and Yang, 2021]

$$\begin{aligned}\phi_0(\xi) &= 1, \quad \phi_1(\xi) = \xi, \\ \phi_{n+1}(\xi) &= \frac{2n+1}{n+1}\,\xi\,\phi_n(\xi) - \frac{n}{n+1}\,\phi_{n-1}(\xi), \quad n \geq 1,\end{aligned}$$

and are orthogonal with respect to the L^2 inner product [Lima, 2022]

$$\langle \phi_m, \phi_n \rangle = \int_{-1}^{1} \phi_m(\xi)\,\phi_n(\xi)\,d\xi = \frac{2}{2n+1}\,\delta_{mn} \tag{2.16}$$

where δ_{mn} denotes the Kronecker delta.

In the multivariate case, the basis functions $\psi_i(\bar{\xi})$ are constructed as tensor products of univariate Legendre polynomials

$$\psi_i(\bar{\xi}) = \prod_{k=1}^{d} \phi_{\boldsymbol{\alpha}_k}(\bar{\xi}_k), \tag{2.17}$$

where $\boldsymbol{\alpha}_i = (\alpha_1, \ldots, \alpha_d)$ is a multi-index specifying the degree of the univariate polynomial $\phi_{\boldsymbol{\alpha}_k}$ associated with the k-th uncertain parameter ξ_k. The total degree of the multivariate polynomial is given by $\sum_{k=1}^{d} \boldsymbol{\alpha}_k$, and only indices satisfying $\sum_{k=1}^{d} \boldsymbol{\alpha}_k \leq p$ are included in the truncated basis.

As a concrete example, consider the case $d = 2$ and $p = 2$, where the total-degree truncation includes all multi-indices $\boldsymbol{\alpha}_i = (\alpha_1, \alpha_2)$ such that $\alpha_1 + \alpha_2 \leq 2$. The resulting multivariate basis functions are

$$\begin{aligned}\psi_{(0,0)}(\xi_1,\xi_2) &= \phi_0(\xi_1)\phi_0(\xi_2), \\ \psi_{(1,0)}(\xi_1,\xi_2) &= \phi_1(\xi_1)\phi_0(\xi_2), \\ \psi_{(0,1)}(\xi_1,\xi_2) &= \phi_0(\xi_1)\phi_1(\xi_2), \\ \psi_{(2,0)}(\xi_1,\xi_2) &= \phi_2(\xi_1)\phi_0(\xi_2), \\ \psi_{(1,1)}(\xi_1,\xi_2) &= \phi_1(\xi_1)\phi_1(\xi_2), \\ \psi_{(0,2)}(\xi_1,\xi_2) &= \phi_0(\xi_1)\phi_2(\xi_2).\end{aligned} \tag{2.18}$$

These correspond to all multivariate basis functions with total degree at most 2, and the total number of such terms is given by

$$S = \frac{(d+p)!}{d!\,p!} = \frac{(2+2)!}{2!\,2!} = 6.$$

For reference, the first few univariate Legendre polynomials and their exact value bounds over $\xi \in [-1,1]$ are:

$$\begin{aligned}
&\phi_0(\xi) = 1, && \phi_0(\xi) \in [1,1],\\
&\phi_1(\xi) = \xi, && \phi_1(\xi) \in [-1,1],\\
&\phi_2(\xi) = \frac{1}{2}(3\xi^2 - 1), && \phi_2(\xi) \in [-0.5,1],\\
&\phi_3(\xi) = \frac{1}{2}(5\xi^3 - 3\xi), && \phi_3(\xi) \in [-1,1],\\
&\phi_4(\xi) = \frac{1}{8}(35\xi^4 - 30\xi^2 + 3), && \phi_4(\xi) \in [-0.4286,1],\\
&\phi_5(\xi) = \frac{1}{8}(63\xi^5 - 70\xi^3 + 15\xi), && \phi_5(\xi) \in [-1,1].
\end{aligned} \tag{2.19}$$

These bounds are useful for later estimation of the range of the multivariate basis functions when evaluating or bounding the polynomial chaos expansion.

Computation of Expansion Coefficients

The expansion coefficients $a_i(t_k)$ in (2.14) are computed using *Galerkin projection* [Wang and Yang, 2021], which involves projecting the system output $y(\bar{\xi},t_k)$ onto each basis function $\psi_i(\bar{\xi})$ in the orthogonal polynomial basis

$$a_i(t_k) = \frac{\langle y(\bar{\xi},t_k), \psi_i(\bar{\xi})\rangle}{\langle \psi_i(\bar{\xi}), \psi_i(\bar{\xi})\rangle}, \tag{2.20}$$

where $\langle f,g\rangle$ denotes the inner product

$$\langle f,g\rangle = \int_{[-1,1]^d} f(\bar{\xi})\, g(\bar{\xi})\, d\bar{\xi}.$$

The orthogonality of the basis functions allows the denominator of (2.20) to be calculated analytically, as shown in (2.16).

In most practical cases, the function $y(\bar{\xi},t_k)$ is not available in closed form, but its values can be obtained through deterministic simulations at a finite set of collocation points $\{\bar{\xi}_j\}_{j=1}^N$. Following the approach in [Wang and Yang, 2021], this thesis uses $N = (p+1)^d$ collocation nodes, constructed as a full tensor product of $p+1$ Gauss–Legendre points in each of the d uncertain dimensions.

To approximate the integrals required for projection, Gauss-Legendre quadrature over $[-1,1]^d$ is employed, leading to the numerical approximation:

$$\int_{[-1,1]^d} f(\bar{\xi})\, g(\bar{\xi})\, d\bar{\xi} \approx \sum_{j=1}^{N} w_j\, f(\bar{\xi}_j)\, g(\bar{\xi}_j), \tag{2.21}$$

where w_j are the associated quadrature weights. Accordingly, the expansion coefficients are approximated as

$$a_i(t_k) \approx \frac{\sum_{j=1}^{N} w_j \, y(\bar{\xi}_j, t_k) \, \psi_i(\bar{\xi}_j)}{\langle \psi_i(\bar{\xi}), \psi_i(\bar{\xi}) \rangle}. \tag{2.22}$$

Since the values $y(\bar{\xi}_j, t_k)$ are obtained through independent deterministic simulations at the collocation nodes $\bar{\xi}_j$, the procedure remains non-intrusive and readily applicable to existing computational models. This property makes PCE particularly suitable for black-box models or simulation pipelines, as it requires no reformulation of the governing equations or their numerical solvers.

Interval Bounds from the Expansion

Once the polynomial chaos expansion has been constructed, interval bounds for the system output $y(\bar{\xi}, t_k)$ can be estimated by evaluating or bounding the truncated series

$$a_0(t_k) + \sum_{i=1}^{S-1} a_i(t_k) \, \psi_i(\bar{\xi}), \quad \text{for} \quad \bar{\xi} \in [-1, 1]^d.$$

A straightforward approach to bounding the expansion is to apply interval arithmetic using the known value ranges of the univariate basis functions listed in (2.19), thereby constructing an inclusion function over $[-1, 1]^d$. However, such an evaluation can suffer from overestimation, particularly in higher dimensions, because of the wrapping effect.

Standard techniques to improve enclosure tightness include subdividing the parameter domain into smaller subdomains and evaluating the expansion over each piece individually, as described in [Moore et al., 2009, Chapter 6]. Splitting each of d uncertain parameters into n parts results in n^d independent evaluations.

3

Implementation of the IntervalSimulations.jl Framework

3.1 Overview of `IntervalSimulations.jl`

This chapter presents the Julia package `IntervalSimulations.jl` (`IS.jl`), developed as part of this thesis to support simulation of dynamical systems under interval-valued uncertainty. It extends `ModelingToolkit.jl` (`MTK.jl`) by enabling the representation of uncertain parameters and initial conditions as intervals, and automates the application of uncertainty propagation techniques.

The package implements the following non-intrusive methods (introduced in Chapter 2):

- **Scanning Method** (SM)
- **Polynomial Chaos Expansion** (PCE)
- **Monte Carlo Simulation** (MC)

For each method, the simulation timespan and relevant configuration parameters are provided as input. The package then automatically generates the required sampling or quadrature points and invokes an appropriate solver from the `DifferentialEquations.jl` (`DE.jl`) ecosystem. A central design feature is the decoupling of trajectory computation and bound extraction: simulation results are stored in full and analyzed separately, allowing flexible reuse across multiple uncertainty analyses.

In addition, `IS.jl` serves as a bridge between symbolic models defined in `MTK.jl` and reachability solvers in `ReachabilityAnalysis.jl` (`RA.jl`), specifically those based on Taylor models. Since `RA.jl` requires systems in strict ODE form, `IS.jl` includes functionality to transform differential-algebraic equation (DAE) models into compatible initial value problems (IVPs) automatically.

3.2 Scanning Method Implementation

The scanning method in `IS.jl` constructs a parameter grid over the uncertain variables and simulates the system for each grid point combination, as explained in Section 2.6. The results are stored in full, and bounds are computed in a separate post-processing step, consistent with the general architecture of the package.

The implementation is non-intrusive and solver-agnostic, relying on solvers from `DifferentialEquations.jl`. To support efficient post-processing, the code allows selective extraction of bounds for a specified subset of variables, avoiding unnecessary computations for untracked outputs.

The function is designed for flexibility, with options for grid resolution, time step, and variable selection. This general structure is shared with the Monte Carlo implementation, enabling consistent bound computation across different uncertainty propagation methods.

3.3 Monte Carlo Method Implementation

The Monte Carlo method in `IS.jl` performs uncertainty propagation by randomly sampling parameter values from the specified intervals and simulating the system for each sampled configuration, as outlined in Section 2.7. Each simulation corresponds to a unique realization of the uncertain parameters.

To support flexible analysis workflows, the simulation and bounding phases are separated. All raw simulation results are stored, and bounds are computed independently as a post-processing step. This design enables repeated evaluation of different variables or aggregation strategies without requiring re-simulation.

As with the Scanning Method, the implementation is non-intrusive and solver-agnostic, relying on solvers from `DifferentialEquations.jl`. Configuration options include the number of samples, output time resolution, and variable selection for bound computation.

3.4 Polynomial Chaos Expansion Implementation

The Polynomial Chaos Expansion (PCE) implementation in `IS.jl` follows the collocation-based approach described in Section 2.9. When parameters or initial conditions are specified as intervals in an `MTK.jl` model, `IS.jl` automatically constructs a collocation grid and propagates uncertainty using PCE. The total polynomial order p controls both the number of terms in the truncated expansion and the resolution of the collocation grid. By default, the number of collocation nodes is set to $(p+1)^d$, where d is the number of uncertain parameters. The number of collocation nodes can be increased to improve integration accuracy.

Collocation nodes and their associated quadrature weights are computed using the `FastGaussQuadrature.jl` [FastGaussQuadrature.jl contributors, 2025]

package. Each node in the standard domain $[-1,1]^d$ is mapped to the corresponding parameter interval, and the system is simulated independently at each transformed point using a solver from the `DifferentialEquations.jl` ecosystem. The resulting trajectories are stored and used to compute the expansion coefficients using Galerkin projection, numerically approximated via Gauss-Legendre quadrature as explained in Section 2.9.

To extract interval bounds from the resulting expansion, the package supports both interval arithmetic (using known bounds on the basis functions) and grid-based evaluation over $[-1,1]^d$. The latter reduces overestimation at the cost of additional computations. A simple heuristic, developed as part of this thesis, is used to automatically switch to grid-based evaluation when high-order terms contribute significantly to the expansion, helping to maintain tighter enclosures without manual intervention. Domain splitting is also supported: the parameter domain is divided into axis-aligned subregions, and the expansion is evaluated separately within each. The final bounds are then computed by taking the pointwise extremum across all subregions. This approach improves enclosure accuracy and can be applied regardless of the problem dimension.

All coefficients, basis evaluations, and integration metadata are returned to support post hoc analysis, error checking, and reuse across simulations. The PCE functionality in `IS.jl` is fully non-intrusive and integrates naturally into the symbolic modeling workflow established by `ModelingToolkit.jl`.

3.5 Taylor Model Reachability

Reachability using Taylor models is supported in `IS.jl` via integration with `ReachabilityAnalysis.jl` (`RA.jl`). The core challenge is compatibility: while `RA.jl` expects initial value problems (IVPs) formulated as strict ODEs, models created in `ModelingToolkit.jl` (`MTK.jl`) are often DAEs and may include algebraic constraints, auxiliary variables, or symbolic constructs.

To address this, `IS.jl` provides a preprocessing pipeline, exposed through the `createIVP` function. It performs symbolic transformations to produce an ODE-only system with interval-valued initial conditions, compatible with `RA.jl` solvers. The pipeline includes

- **Structural simplification** via `structural_simplify`.
- **Substitution of observed variables** using MTK's internal mappings.
- **Elimination of algebraic equations** using `symbolic_linear_solve`.
- **Code generation** using `build_function` to produce callable right-hand side functions from symbolic expressions.

- **Assembly of an IVP** in the form `@ivp(...)`, where uncertain parameters are promoted to states with zero dynamics, i.e., $\dot{p} = 0$, as required by `RA.jl`.

Once an ODE has been generated, further simplification may still be required. The symbolic expressions produced by `Symbolics.jl` [Gowda et al., 2021] can contain redundant terms or have suboptimal structure, which may hinder performance. To assist with this, `IS.jl` supports exporting expressions in MATLAB-compatible syntax [The MathWorks, Inc., 2024a], enabling refinement using the Symbolic Math Toolbox [The MathWorks, Inc., 2024b]. The simplified expressions can then be reintroduced into the Julia workflow for use in reachability analysis.

The final output is an IVP object with interval initial conditions, fully compatible with RA.jl's solvers. This workflow automates what would otherwise be a manual and error-prone translation step, and enables symbolic MTK models to be used directly in formal reachability analysis.

Once a compatible ODE-based initial value problem has been constructed, it is passed to `RA.jl` for simulation using Taylor model-based solvers provided by `TaylorModels.jl`(`TM.jl`). Enclosures for the system states are computed at each time step, and bounds are computed using built-in postprocessing routines.

The implementation in `IS.jl` includes support for interval splitting, which divides the initial condition space into axis-aligned subregions and runs reachability analysis independently for each. Since the resulting trajectories may differ in time resolution, a linear interpolation step aligns the outputs to a common time grid before combining them.

To enable finer control over solver behavior, the local installation of `TM.jl` was extended with support for fixed step sizes and configurable maximum step lengths. This is currently not available in the upstream version of the package.

3.6 Multibody.jl Support

Although `IntervalSimulations.jl` was not initially designed for compatibility with `Multibody.jl`, its uncertainty quantification methods can be applied to such models with minimal modification. `Multibody.jl` provides a high-level interface for constructing mechanical systems using interconnected rigid-body components and generates symbolic models using `ModelingToolkit.jl`.

However, limitations arise when attempting to use Taylor model-based reachability analysis. Models constructed with `Multibody.jl` often involve complex symbolic structures and branching logic (e.g., `ifelse` expressions) that hinder symbolic simplification. As a result, they cannot reliably be transformed into strict initial value problems, which prevents the use of Taylor model-based solvers in `ReachabilityAnalysis.jl`.

Additionally, direct interval-valued parameters are generally not supported inside `Multibody.jl` component definitions due to incompatibility between inter-

val arithmetic and conditional expressions. uncertainty can instead be introduced externally by passing parameter dictionaries to the solver during simulation.

Despite these constraints, models generated with `Multibody.jl` remain fully compatible with the non-intrusive uncertainty propagation methods implemented in `IS.jl`, including Scanning, Monte Carlo, and Polynomial Chaos Expansion.

4

Models

This thesis uses a series of models with increasing complexity to demonstrate and evaluate uncertainty quantification methods. The examples range from a simple first-order linear system to a more advanced three-degree-of-freedom (3-DOF) robotic arm. All models are implemented in Julia, using the `ModelingToolkit.jl` with components from `ModelingToolkitStandardLibrary.jl` and `Multibody.jl`.

4.1 First-Order Linear System

The simplest model considered is a first-order linear control system described by

$$\dot{x}(t) = Ax(t) + Bu(t), \quad x(0) = 0, \tag{4.1}$$

with constant input $u(t) = 1$, $A = -\frac{1}{\tau}$, and $B = \frac{1}{\tau}$, where $\tau \in [2.9, 3.1]$. This formulation is equivalent to the affine system

$$\dot{x}(t) = \frac{1 - x(t)}{\tau}, \quad x(0) \in [0.1, 0.3] \tag{4.2}$$

and serves as a minimal example to illustrate fundamental issues with naive interval arithmetic approaches.

4.2 Scissors

The scissors used in this thesis was originally intended to represent a surgical instrument for robotic surgery. In such applications, high precision is essential, and the effects of model uncertainty cannot be overlooked. The model captures key physical features of a planar scissors design while remaining tractable for symbolic manipulation. A 3D scan of the surgical scissors used as inspiration is shown in Figure 4.1.

The system was constructed using `ModelingToolkit.jl` and components from `ModelingToolkitStandardLibrary.jl`. It models a single rotational degree of freedom around an axis orthogonal to the plane of motion, driven

Figure 4.1 3D scan of surgical scissors used as inspiration for the modeled system.

by a torque source with time-varying amplitude $\tau(t)$. The structure includes a fixed base, a rigid body with rotational inertia, a nonlinear friction element (combining Coulomb and breakaway friction), and a spring-damper with pretension about a rest angle. The resulting rotational angle $\phi(t)$ governs the mechanical state of the system and is used to compute the position of the tool center point (TCP) through geometric projection.

Model parameters were selected through trial and error to produce behavior qualitatively consistent with closing scissors. Friction parameters were tuned to avoid stiffness that caused convergence problems during Taylor model-based reachability analysis. Uncertainty was introduced by assigning intervals to parameters such as stiffness, damping, and rest angle.

The symbolic model was automatically transformed into an initial value problem (IVP) using the tools described in Chapter 3. The full ModelingToolkit definition is listed in Appendix 9.1.1, and the manually simplified final implementation used in simulations is provided in Appendix 9.1.2.

4.3 Double Pendulum

The double pendulum, shown in Figure 4.2, serves as a benchmark system for evaluating uncertainty quantification methods. The model is based on the formulation in [Wang and Yang, 2021], which reports results for Polynomial Chaos Expansion (PCE), Monte Carlo simulation, parameter scanning, and Taylor model-based reachability analysis. This makes it well suited for validating the present implementations and ensuring consistent conditions for performance comparison. The complete system equations are provided in the original reference.

The system was originally defined as a DAE using `ModelingToolkit.jl` and then transformed into an ODE to support Taylor model-based reachability analysis. To investigate how expression structure influences performance and numerical

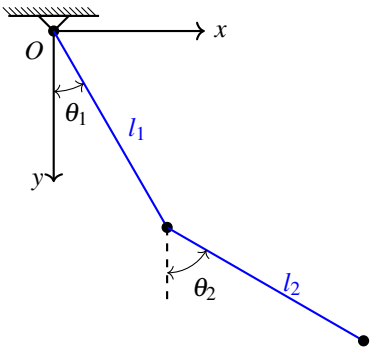


Figure 4.2 Schematic of the double pendulum used for benchmarking, showing joint angles θ_1 and θ_2 as well as link lengths l_1 and l_2. Image adapted from [Wang and Yang, 2021].

stability, four variants of the model were created:

- **dp**: Baseline model with automatically generated ODEs from `MTK.jl`.
- **dp2**: A manual transcription of the equations for improved readability and control.
- **dp_simplified**: Symbolically simplified using MATLAB to reduce expression size and complexity.
- **dp_structured**: Further optimized by extracting common subexpressions and rearranging terms to improve runtime efficiency.

This setup allowed for controlled evaluation of execution time, error growth, and solver robustness under different uncertainty quantification methods.

Simulation setup. Unless otherwise stated, all simulations use the same initial conditions and uncertainties: $\theta_1 = \frac{\pi}{4}$, $\theta_2 = \frac{\pi}{3}$, $\dot{\theta}_1 = 0$, $\dot{\theta}_2 = 0$, with uncertain parameters $l_1 = 1 \pm 0.05$ and $l_2 = 1 \pm 0.05$. These values are chosen to match the setup in [Wang and Yang, 2021] and are used consistently throughout all experiments involving the double pendulum.

4.4 3-DOF Robotic Arm

As a final example, a realistic robotic system was modeled: a planar robotic arm with three degrees of freedom (3-DOF). The system was constructed using `Multibody.jl` and `ModelingToolkit.jl`, which enable the definition of

mechanical systems by connecting modular components such as rigid bodies, joints, gears, and actuators.

The robot includes two actuated rotational joints, each driven through gear-reduced motors with spring-damper elements to model joint compliance and friction. A third, passive rotational joint is located at the base to represent compliance in the attachment to the ground. Each link is assigned realistic mass and inertia properties. The reference motion is defined using an analytical inverse kinematics trajectory that describes circular end-effector motion.

The multibody structure was assembled using standard components such as `Revolute`, `Body`, `FixedTranslation`, `SpringDamper`, and `Inertia`. This modular, block-based modeling approach allows system equations to be generated automatically and then structurally simplified for simulation and reachability analysis. The resulting model captures the realistic interactions between links, joints, and actuators, making it a suitable test case for evaluating uncertainty quantification methods. The complete model definition is provided in the Appendix 9.1.3.

An illustration of the robot configuration at a selected time point, rendered using `Multibody.jl` and `GLMakie.jl` [Danisch and Krumbiegel, 2021], is shown in Figure 4.3.

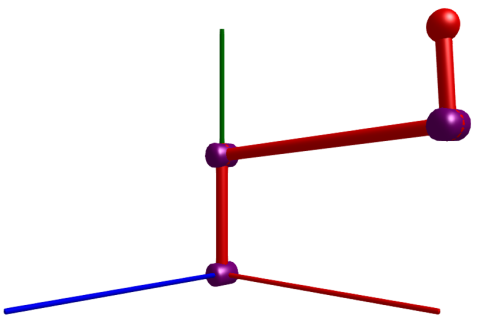


Figure 4.3 3D rendering of the 3-DOF robotic arm configuration, generated in Julia using `Multibody.jl` and `GLMakie.jl`.

5

Methods

This chapter outlines the methodology used to investigate the performance, robustness, and practical utility of various uncertainty quantification methods in simulating robotic systems. The study is grounded in computational experiments, guided by prior theoretical knowledge and practical challenges observed during implementation and testing of `IntervalSimulations.jl`.

All simulations were executed on a single workstation. Full hardware and software specifications, including CPU, GPU, operating system, and Julia version, are provided in Appendix 9.2 (Table 9.1).

5.1 Methodological Overview

The goal of this thesis is to assess how well different uncertainty propagation methods perform in practice, both in terms of accuracy and computational efficiency. The following methods were evaluated:

- **Naive Interval Arithmetic**, used to demonstrate overestimation from dependency issues.
- **Scanning Method**, serving as a benchmark by evaluating system behavior at interval endpoints.
- **Monte Carlo Sampling**, using thousands of samples to approximate empirical bounds.
- **Polynomial Chaos Expansion (PCE)**, evaluated for convergence, coefficient behavior, and reconstruction strategies.
- **Taylor Models**, assessed for bounding accuracy, sensitivity to equation structure, and robustness under different simulation settings.

5.2 Reference Bounds via Scanning and Monte Carlo

To evaluate the tightness of bounds produced by each method, a hybrid reference was created by combining scanning over extreme parameter values with Monte Carlo simulations using up to 50,000 samples. This combination served as an empirical approximation of the true reachable set.

5.3 Naive Interval Arithmetic

Naive interval arithmetic refers to directly propagating interval-valued parameters through a numerical integration scheme, without restructuring the equations or addressing dependency effects. In this thesis, it is implemented using an explicit Euler solver in combination with the `IntervalArithmetic.jl` package.

All computations are carried out in interval form, with no symbolic simplification, constraint propagation, or postprocessing. The purpose of this method is not to produce useful bounds, but rather to demonstrate how standard numerical solvers can quickly lead to divergence or severe overestimation when used with interval initial conditions, even on simple systems.

5.4 Taylor Models: Structure and Robustness

Taylor model-based reachability analysis was tested using various representations of the system equations, ranging from automatically generated ODEs to manually simplified forms. These tests assessed how expression structure affects simulation time and accuracy. Taylor model robustness was further evaluated by varying parameters such as `orderT`, `orderQ`, and `abstol`, and by identifying conditions under which simulations fail.

5.5 Domain Splitting

Domain splitting was incorporated into the experimental design for both Taylor and PCE methods. In the case of Taylor models, the technique was used to examine its impact on solver reliability and bounding behavior. For PCE, domain splitting was applied to study how subdividing the parameter space affects reconstruction strategies and bound tightness.

5.6 Validation of PCE Implementation

To validate the correctness of the PCE method, simulation results for the double pendulum were visually compared against the solution plots presented in [Wang and

Yang, 2021]. This comparison served to confirm that the implementation produced qualitatively consistent results.

5.7 Heuristic Bounding Strategy and Coefficient Analysis

Coefficient-based metrics such as entropy and the magnitude of high-order terms were used to guide post-processing strategies and to assess when interval-based reconstruction becomes unreliable. These diagnostics were also analyzed in relation to domain splitting and reconstruction error.

5.8 Symbolic Modeling with `ModelingToolkit.jl`

To investigate how uncertainties propagate through robotic systems, most models in this thesis were defined symbolically using `ModelingToolkit.jl`. This framework enables uncertainty to be introduced by replacing parameter values with intervals or samples, without requiring structural changes to the model.

The symbolic formulation also supports observed variables, quantities derived from the system state, such as the TCP position, which can be evaluated automatically after simulation. This setup allows consistent and automated propagation of uncertainty to relevant outputs and is used throughout the thesis to assess the effectiveness of Julia's symbolic modeling tools in this context.

5.9 Convergence with PCE Order and Monte Carlo Samples

Experiments were conducted to investigate the effect of increasing PCE order and the number of Monte Carlo samples on the accuracy of the resulting bounds. These simulations aimed to determine whether sample sizes of 10,000 or 50,000 were necessary and to assess the trade-offs between computational cost and bounding precision.

6

Results

6.1 Results and Comparative Analysis

This section compares the results obtained using four uncertainty propagation methods: naive interval arithmetic, Polynomial Chaos Expansion (PCE), Taylor models, and Monte Carlo simulations. In addition, a hybrid reference method—Monte Carlo combined with scanning over parameter extremes (MC+SM)—is used as a benchmark for accuracy.

The methods are evaluated on several systems described in Chapter 4. Key metrics include bound tightness, runtime, and sensitivity to uncertainty structure.

Visualizations, error comparisons, and runtime statistics are provided for each method, with a focus on practical trade-offs between accuracy and computational cost.

6.2 Monte Carlo

Monte Carlo (MC) simulations are used as both a standalone method and as part of the hybrid MC+SM benchmark to evaluate the accuracy of other uncertainty quantification techniques. In this section, we assess the convergence properties and computational cost of Monte Carlo sampling for the double pendulum system introduced in Section 4.3.

Figure 6.1 shows how the norm of the error between results obtained using different numbers of samples and a reference solution with 50,000 samples decreases as the number of simulations increases. The plot is shown on a log-log scale, revealing the expected approximate exponential convergence.

Table 6.1 compares the computational time required for Polynomial Chaos Expansion (PCE), Monte Carlo (MC), and the hybrid MC+SM method across increasing PCE orders. For each PCE order, the corresponding number of MC samples needed to match the accuracy is estimated by aligning the error. As seen in the table, higher-order PCEs provide accurate bounds with significantly less computational time than pure MC sampling.

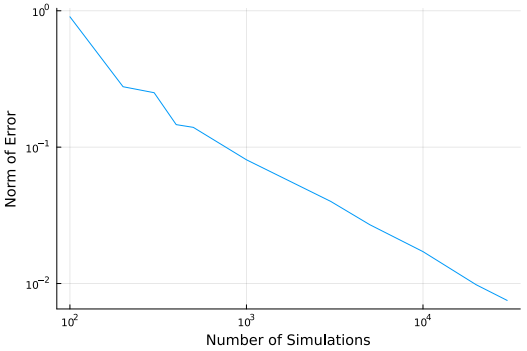


Figure 6.1 Convergence of the Monte Carlo method for the double pendulum. The plot shows the norm of the error compared to a reference solution with 50,000 samples. Axes are shown on a log-log scale.

Table 6.1 Estimated number of Monte Carlo (MC) samples and runtime required to match the bounding accuracy of PCE at increasing polynomial orders, based on simulations of the double pendulum. MC+SM times refer to the hybrid method combining MC with parameter scanning.

PCE Order	PCE (time) [s]	MC+SM (time) [s]	MC (time) [s]	MC+SM Samples	MC Samples
2	1.6	0.4	49.6	0.0	302.0
3	2.9	1.3	520.4	5.7	3213.0
4	4.1	2.4	1395.0	19.1	8752.0
5	5.9	5.2	4320.0	33.5	26337.0
6	9.3	11.8	-	75.7	-
7	10.8	30.2	-	197.3	-
8	14.3	94.3	-	627.4	-
9	15.8	158.0	-	1093.1	-
10	20.5	323.4	-	1557.7	-

6.3 Interval Arithmetic

A simulation was performed using naive interval arithmetic on the first-order system introduced in Section 4.1. Figure 6.2 shows the evolution of the state interval over time. The plot compares the bounds produced by four uncertainty quantification methods

- **Bounds Euler**: naive forward Euler integration with interval arithmetic,
- **Bounds SM**: scanning method evaluating the system at parameter extremes,
- **Bounds PCE**: bounds reconstructed from a Polynomial Chaos Expansion,
- **Bounds TM**: bounds obtained using Taylor model integration.

The bounds computed using SM, PCE, and TM overlap closely for this simple system and are therefore visually indistinguishable in the figure. Only the black bounds from naive interval arithmetic are clearly visible, illustrating its tendency to overestimate due to the wrapping effect.

Table 6.2 summarizes the simulation and bounding times for each method. The Taylor Model simulation used a solver with order 4 in time, order 2 in space, and a fixed step size of 0.01 seconds.

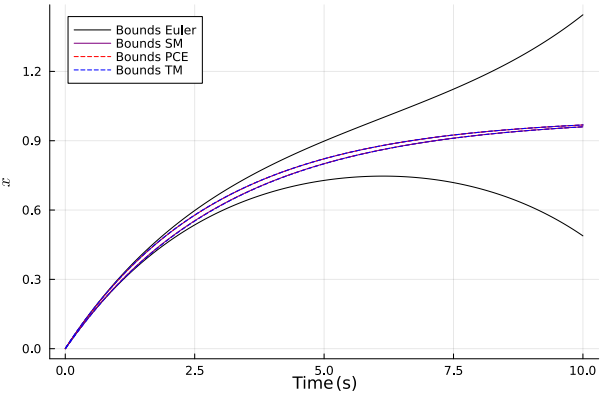


Figure 6.2 Uncertainty propagation over time using naive interval arithmetic (Euler) and other bounding methods. Bounds from SM, PCE, and TM overlap and are nearly indistinguishable in the plot.

Table 6.2 Computation times (in seconds) for simulation and bounding across different methods.

Method	Simulation Time (s)	Bounding Time (s)
Scanning Method (SM)	0.012	0.004
Interval Arithmetic	0.0009	0.024
Polynomial Chaos (PCE)	0.019	0.033
Taylor Models (TM)	2.080	0.034

6.4 Taylor Models

6.4.1 Comparing Different Taylor Model Formulations

To evaluate the impact of equation structure on Taylor model performance, four variants of the double pendulum system defined in Section 4.3 were simulated over

the time interval $[0, 1.0]$. All simulations use the same initial conditions and parameter uncertainties specified in the model section. The solver `TMJets21b` was configured with order 4 in time, order 2 in space and adaptive steps (abstol = 10^{-6}).

Simulation runtimes varied significantly. The unoptimized models `dp` and `dp2` required over 225 seconds to complete, while the simplified and structured variants completed in under 22 and 15 seconds, respectively. Applying the `@taylorize` macro to the reduced models had negligible impact on runtime but introduced substantial parsing overhead. When applied to `dp2`, compilation stalled for over 10 minutes and was aborted.

The propagated uncertainty for the first angle, θ_1, is shown in Figure 6.3. All model variants produced visually identical results for θ_1. The runtime for each variant is shown in Table 6.3.

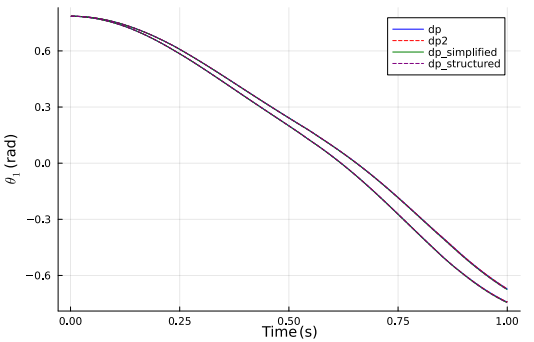


Figure 6.3 Uncertainty propagation over time for θ_1 across four differently structured double pendulum models.

Table 6.3 Simulation runtime for different Taylor model formulations of the double pendulum.

Model	Time (s)
dp	225.7
dp2	229.0
dp_simplified	21.6
dp_structured	14.7

6.4.2 Domain Splitting

When simulating the `dp_structured` double pendulum model over a 10-second interval, the solver failed after 7.67 seconds due to unsuccessful steps. This occurred both with and without the use of the `@taylorize` macro.

Applying domain splitting to the uncertain parameters l_1 and l_2 (2 splits per parameter) allowed the simulation to complete successfully over the full interval. The simulation result is included in the overall method comparison in Figure 6.14.

6.4.3 Sensitivity to Uncertainty in Initial Conditions

In the scissors simulation, the solver completed successfully when small uncertainties were included in two state variables of the initial condition.

```
X0 =IntervalBox([1.0..1.0, 0..0, 0.10 ±0.01, 0.5 ±0.01, 0.0 ±
    0.0])
@time sol =solve_reachability(prob1, prob[2], (0.0, 2.0),
   solver =TMJets21b(abstol=1e-13, orderT=8, orderQ=2,
                  maxsteps=100000, adaptive=true))
```

With the same values set as exact points, the solver failed after approximately 1.14 seconds.

```
X0 =IntervalBox([1.0..1.0, 0..0, 0.10 ±0.0, 0.5 ±0.0, 0.0 ±0
    .0])
@time sol =solve_reachability(prob1, prob[2], (0.0, 2.0),
   solver =TMJets21b(abstol=1e-13, orderT=8, orderQ=2,
                  maxsteps=100000, adaptive=true))
```

Adding a maximum step size of 10^{-3} enabled the simulation to complete successfully with deterministic initial conditions.

6.5 Polynomial Chaos Expansion (PCE)

Polynomial Chaos Expansion (PCE) is used to propagate parametric uncertainty in the double pendulum, scissors, and robotic arm models. The method approximates the system response using orthogonal Legendre polynomial expansions, capturing the uncertainty in a compact, structured form. This section evaluates PCE under different orders, domain splitting strategies, and bounding techniques.

All simulations use the initial conditions and uncertainties defined in their respective model sections. The PCE coefficients are computed via Gauss–Legendre collocation using a custom Julia implementation.

6.5.1 Implementation Verification

The setup for the double pendulum model was reproduced from [Wang and Yang, 2021], using the parameter uncertainties $l_1, l_2 = 1 \pm 0.05$. The resulting trajectories matched those shown in the original reference.

6.5.2 Effect of Domain Splitting

A 10-second simulation of the double pendulum was performed using a tenth-order PCE expansion, with bounds reconstructed using interval arithmetic. Results were compared to a high-resolution Monte Carlo simulation combined with scanning over parameter extremes (MC+SM), used as the reference.

Figure 6.4 shows the resulting bounds for different domain splitting configurations. Solve and bounding times are summarized in Table 6.4. Solving and bounding were performed concurrently using threads, and reported times correspond to the initial execution.

Table 6.4 Solve and bounding times (in seconds) for tenth-order PCE on the double pendulum with different domain splitting configurations.

Split Configuration	Solve Time (s)	Bounding Time (s)
No splits	9.70	2.27
2 splits	12.48	3.32
3 splits	21.86	7.49

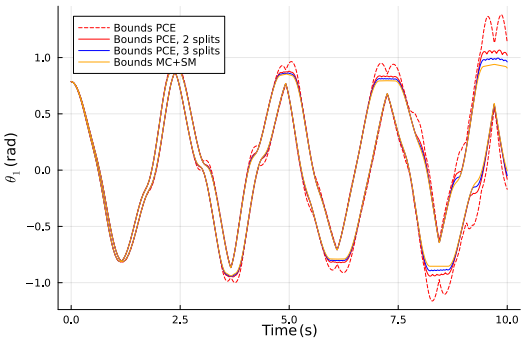


Figure 6.4 Uncertainty propagation using tenth-order PCE with different domain splitting strategies. Bounds are reconstructed using IntervalArithmetic. MC+SM serves as the reference.

6.5.3 Coefficient Comparison With and Without Domain Splitting (3-DOF Robot)

A fifth-order PCE expansion was applied to the 3-DOF robotic arm model to approximate the TCP position under uncertainty in spring and damper parameters. Table 6.5 lists the eight largest coefficients at $t = 130$, with the left column show-

ing results without domain splitting and the right column showing results with 10× splitting on `springdamper2.c`.

In the unsplit case, the top four coefficients all have indices of the form $(0,0,i)$, corresponding to variation along a single parameter. In the split case, the largest coefficients are distributed across mixed terms involving multiple indices. The average deviation between the reconstructed bounds and those obtained from the MC+SM reference decreased from 0.1211 (no splitting) to 0.0502 (with splitting).

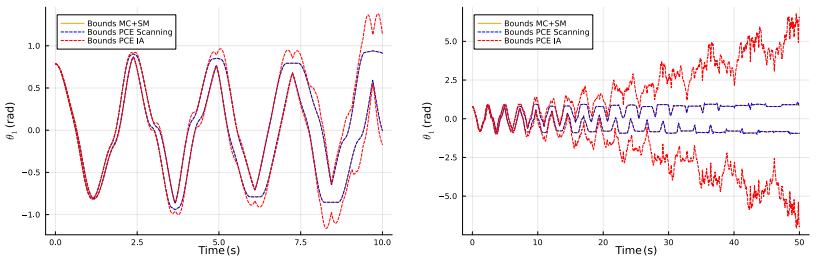
Table 6.5 Top 7 PCE coefficients at $t = 130$. Left: no domain splitting. Right: 10× splitting on `springdamper2.c`.

Index	Coeff.
(0, 0, 0)	0.8542
(0, 0, 4)	-2.49e-3
(0, 0, 2)	1.42e-3
(0, 0, 3)	7.43e-4
(0, 0, 5)	3.44e-4
(1, 0, 4)	-4.10e-5
(0, 1, 4)	3.86e-5

Index	Coeff.
(0, 0, 0)	0.8553
(0, 0, 1)	9.15e-4
(0, 0, 2)	-3.74e-4
(2, 2, 1)	1.19e-4
(1, 0, 3)	-1.11e-4
(2, 1, 1)	1.02e-4
(2, 0, 2)	9.76e-5

6.5.4 Comparison of Bounding Strategies

Bounding was performed using two reconstruction methods: interval arithmetic and grid evaluation. Figure 6.5 compares the resulting bounds using a fourth-order PCE expansion over a short simulation horizon (left) and a tenth-order expansion over 50 seconds (right).



(a) Fourth-order PCE

(b) Tenth-order PCE over 50 seconds

Figure 6.5 Comparison of bounds reconstructed from PCE using interval arithmetic (IA) and grid evaluation (Scanning). The reference bounds from MC+SM are visually indistinguishable from those obtained using PCE with grid evaluation.

Figure 6.6 shows the reconstruction error for both interval arithmetic and grid evaluation methods, measured relative to MC+SM. The x-axis shows the sum of Legendre coefficients of order ≥ 2.

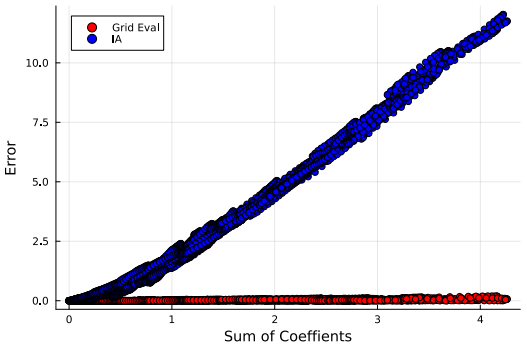


Figure 6.6 Reconstruction error vs. sum of Legendre coefficients of order ≥ 2 for interval arithmetic (IA) and grid evaluation.

Figure 6.7 shows reconstruction error as a function of total coefficient sum (horizontal axis) and entropy (vertical axis), with darker regions indicating lower error. The left panel shows results using grid evaluation; the right panel shows results using IA.

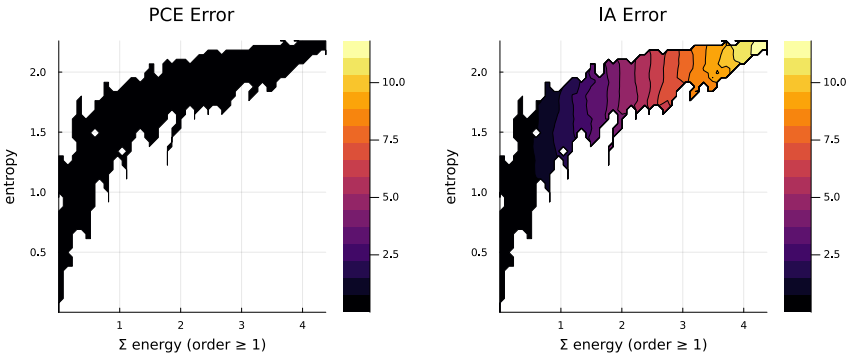


Figure 6.7 Heatmap of reconstruction error as a function of coefficient sum and entropy. Left: grid evaluation. Right: interval arithmetic.

To compare bounding accuracy and runtime under domain splitting, a fifth-order PCE expansion was applied to the 3-DOF robotic arm. The resulting bounds on the x and y TCP positions were extracted using both bounding methods. Table 6.6

reports the average reconstruction error and runtime. Values in parentheses show times from repeated execution.

Table 6.6 Bounding error and runtime using PCE order 5 for various splitting and reconstruction strategies.

Split Strategy	Bounding Method	Error	Runtime (s)
None	IA	0.036	26.2 (3.2)
springdamper2.c (×5)	IA	0.007	135.5 (17.7)
springdamper3.c (×5)	IA	0.052	140.8 (17.3)
None	Grid Eval	0.004	92.5 (66.8)

6.5.5 Heuristic Bounding Strategy

A hybrid bounding strategy was evaluated using a threshold on the sum of Legendre coefficients of order 2 or higher. Grid evaluation was applied when the sum exceeded 0.1; otherwise, interval arithmetic was used. Figure 6.8 shows the resulting bounds. Table 6.7 reports the runtime for each bounding strategy.

Table 6.7 Runtime (in seconds) for different bounding strategies.

Strategy	Runtime (s)
Interval arithmetic only	0.32
Grid evaluation only	0.65
Heuristic strategy	0.38

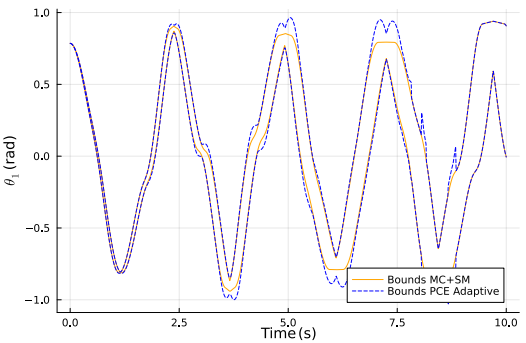


Figure 6.8 Bounds obtained using a hybrid strategy that selects between interval arithmetic and grid evaluation based on the sum of higher-order coefficients.

6.6 Solver Selection and Symbolic Modeling Features

6.6.1 Inverse Problem: Required Torque from Prescribed Trajectory

A second-order PCE expansion was used to compute the torque required to follow a prescribed blade angle trajectory in the scissors model. Figure 6.9 shows the resulting torque profile under parameter uncertainty. Bounds were reconstructed using interval arithmetic. Table 6.8 lists the top 10 Legendre coefficients at $t = 1$, sorted by absolute value.

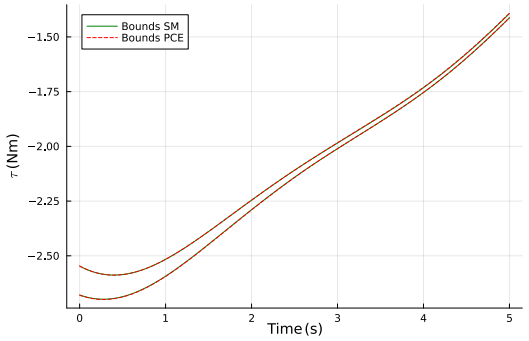


Figure 6.9 Torque required to follow a prescribed blade trajectory in the scissors under parameter uncertainty. Bounds were reconstructed using a second-order PCE expansion. The results from the Scanning Method (SM) and PCE are visually indistinguishable.

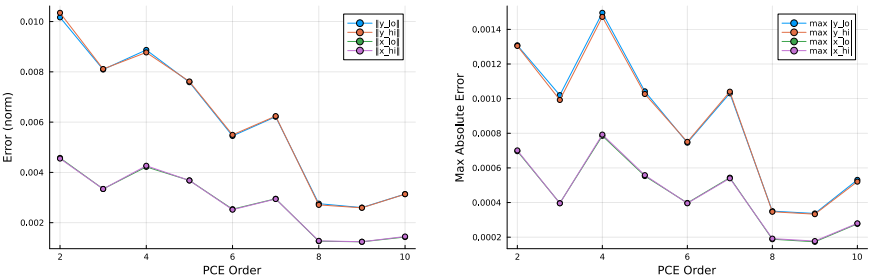
Table 6.8 Top 10 Legendre PCE coefficients at $t = 1$ for the torque inversion problem.

Multi-index	Coefficient
(0, 0, 0, 0)	-2.56
(0, 0, 0, 1)	-0.024
(1, 0, 0, 0)	0.0086
(0, 0, 1, 0)	0.0037
(0, 1, 0, 0)	-0.0020
(1, 1, 0, 0)	-6.0×10^{-4}
(0, 0, 0, 2)	-2.83×10^{-15}
(0, 0, 2, 0)	-1.58×10^{-15}
(2, 0, 0, 0)	-1.28×10^{-15}
(0, 2, 0, 0)	-1.24×10^{-15}

6.6.2 Multibody.jl Simulation of a 3-DOF Robotic Arm

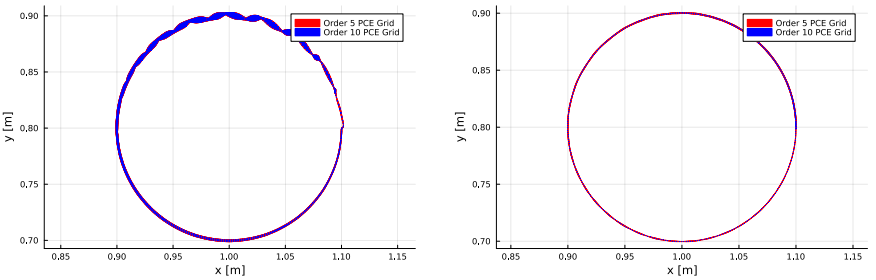
Polynomial Chaos Expansion was applied to a 3-DOF robotic arm model to estimate the reachable TCP positions under parameter uncertainty in spring stiffness and damping. Simulations were performed using PCE orders 5 and 10.

Figure 6.10 shows the error in the reachable bounds, measured relative to a Monte Carlo + Scanning (MC+SM) reference with 24,000 samples. Figure 6.11 compares the TCP trajectories produced by both expansion orders.



(a) Norm of error (b) Maximum absolute error

Figure 6.10 Error metrics for the 3-DOF robot using PCE of increasing order, relative to a Monte Carlo + scanning (MC+SM) reference.



(a) First half of trajectory (b) Second half of trajectory

Figure 6.11 Reachable TCP positions for a 3-DOF robotic arm under uncertainty. PCE of order 5 was evaluated at higher temporal resolution; PCE of order 10 used a coarser time step.

6.7 Method Comparison Across Systems

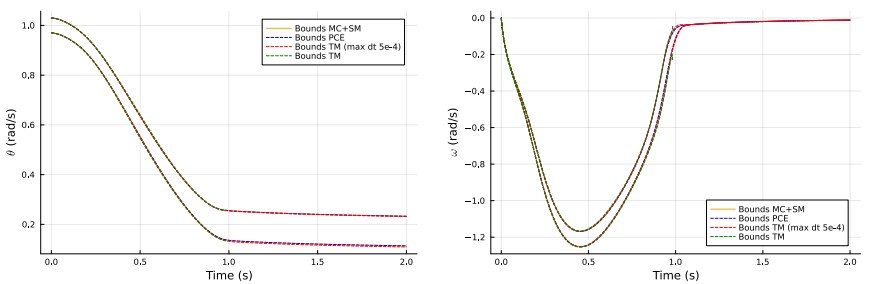
This section provides a side-by-side comparison of the different uncertainty quantification methods applied to two representative systems: the scissors and the double pendulum. By comparing Polynomial Chaos Expansion (PCE), Taylor Models (TM), and Monte Carlo combined with Scanning (MC+SM), the figures highlight the strengths and limitations of each method.

Scissors Bounds

Figure 6.12 compares the bounds on angular position θ and angular velocity ω during the closing motion of the scissors, computed using MC+SM (10,000 samples), Polynomial Chaos Expansion (PCE), and Taylor Models (TM). For TM, simulations were performed both with and without a manually imposed maximum step size.

Without this constraint, the TM-based integration failed after approximately one second due to the growing complexity of the Taylor representation. As a result, the corresponding TM bounds (green) disappear from the plots beyond this point. In contrast, when the step size was limited, the simulation completed successfully, and the bounds remain visible throughout the interval.

In 6 out of 100 batches of MC re-simulation, minor expansions of the reachable set were observed. These deviations were visually negligible and did not affect the overall conclusions.



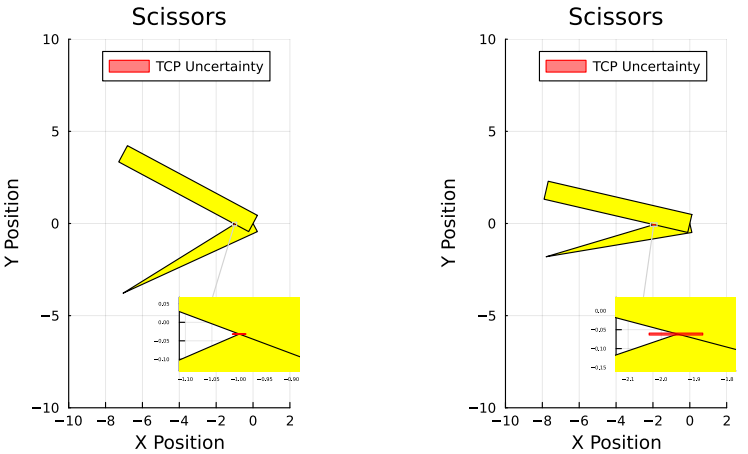
(a) θ for closing scissors

(b) ω for closing scissors

Figure 6.12 Comparison of bounds for closing scissors using MC+SM (10,000 samples), Polynomial Chaos Expansion (PCE), and Taylor Models (TM), both with and without a maximum time step. The simulation using TM without a time step constraint fails around $t = 1\,\text{s}$, after which the corresponding bounds (green) are no longer visible in the plots.

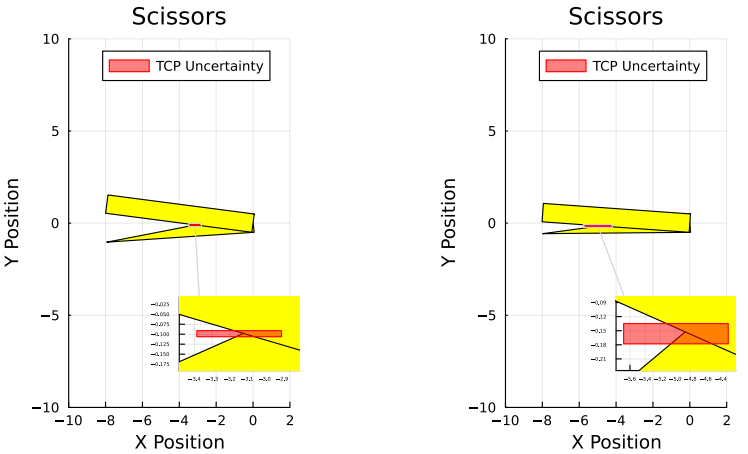
Snapshots of the scissors configuration at selected time points are shown in

Figure 6.13. These frames visualize how uncertainty in the initial conditions and parameters affects the TCP position over time.



(a) Frame at $t = 0.01$

(b) Frame at $t = 0.61$



(c) Frame at $t = 0.81$

(d) Frame at $t = 2.01$

Figure 6.13 Snapshots of the scissors at four time points, illustrating the propagation of uncertainty in the system's dynamics and the TCP position.

Double Pendulum

Figure 6.14 compares the bounds on the first angle θ_1 of the double pendulum computed using four different methods: naive interval arithmetic, fourth-order Polynomial Chaos Expansion (PCE), Taylor Models (TM), and Monte Carlo combined with Scanning (MC+SM). All simulations use the same parameter uncertainties and initial conditions defined in the model.

The plot shows the resulting trajectory enclosures over a 10-second simulation interval. The bounds produced by MC+SM and PCE with grid-based reconstruction are visually indistinguishable, indicating that the PCE approach provides a highly accurate approximation of the true reachable set in this case.

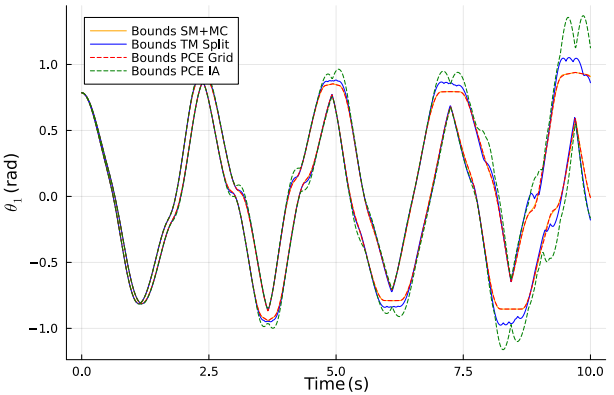


Figure 6.14 Bounds on θ_1 for the double pendulum using interval arithmetic, Polynomial Chaos Expansion (PCE), Taylor Models (TM), and Monte Carlo + Scanning (MC+SM). The bounds from MC+SM and PCE with grid-based reconstruction are visually identical.

7

Discussion

7.1 Interval Arithmetic

The results in Figure 6.2 highlight the limitations of naive interval arithmetic for uncertainty propagation. Even in a simple linear first-order system, the method exhibits rapid overestimation because of dependency loss, a well-known issue where repeated variables are treated independently. For example, the expression $x - x$, with $x = [-1, 1]$, evaluates to $[-2, 2]$ instead of the expected 0.

For these simulations, interval-valued parameters were propagated using the `IntervalArithmetic.jl` package together with an explicit Euler method. No symbolic simplification or restructuring was applied. As expected, the resulting bounds became overly conservative over time, despite the simplicity of the system. This behavior illustrates the wrapping effect in practice and underscores the limitations of using naive interval arithmetic for dynamic systems, even in low dimensions.

In terms of computational cost, interval arithmetic is extremely fast, faster than all other methods evaluated, as shown in Table 6.2. However, this speed comes at the expense of tightness and usefulness of the resulting enclosures.

In summary, naive interval arithmetic serves primarily as a baseline for illustrating dependency-driven overestimation. While fast and simple to implement, it is not suitable for practical uncertainty quantification when accurate bounding is required.

7.2 ModelingToolkit

The results in this thesis show that symbolic modeling using `MTK.jl` plays a central role in enabling systematic uncertainty analysis of robotic systems in Julia. Once a system is defined symbolically, uncertainty can be introduced by replacing fixed parameter values with intervals or samples. This allows the same model structure to be reused across multiple propagation methods without modification, thereby streamlining experimentation and implementation.

A key advantage of symbolic modeling in `ModelingToolkit.jl` is the automated handling of observed variables, quantities derived from the system state but not required to solve the differential equations. This capability is particularly relevant in robotics, where the internal state may consist of joint angles and velocities, while the outputs of interest typically involve the position of the tool center point (TCP). These relationships are retained symbolically during simulation and can be evaluated post hoc without additional model manipulation. For instance, Figure 6.11 shows reachable TCP positions under parameter uncertainty, extracted directly from the symbolic representation.

This symbolic flexibility also extends to inverse problem formulations. In the case of the scissors, only minor modifications were needed to simulate the torque required to follow a prescribed blade trajectory instead of predicting motion from a given torque input. The resulting torque profile under uncertainty is shown in Figure 6.9, demonstrating how symbolic expressions support both forward and inverse analyses within a unified modeling framework.

Some computational overhead was observed when extracting bounds for observed variables, particularly when domain splitting was applied. As shown in Table 6.6, computing bounds on the TCP position of a 3-DOF robotic arm using interval arithmetic became significantly more expensive in split scenarios. While subsequent runs exhibited reduced overhead, possibly due to internal caching, this cost remains relevant when processing large numbers of subdomains. In contrast, bounds on state variables were obtained more efficiently and with greater consistency.

Symbolic modeling also facilitates the preservation of structural parameter dependencies. In the robotic arm model, link masses were defined as proportional to link lengths under the assumption of homogeneous materials and constant density. Encoding this relationship symbolically (e.g., `m1 = l1`) ensured consistent variation of dependent parameters under uncertainty. When such dependencies were not preserved, the resulting bounds were unnecessarily conservative. This highlights the role of symbolic modeling in maintaining physical coherence in parametric uncertainty analysis.

The modeling capabilities of `MTK.jl` are further enhanced by libraries such as `ModelingToolkitStandardLibrary.jl` and `Multibody.jl`, which provide modular components for constructing physically structured systems. These libraries support the composition of models from reusable elements, such as rigid bodies, joints, and actuators, according to physical topology and system architecture. In most cases, models built using these libraries were compatible with the non-intrusive uncertainty propagation methods applied in this thesis, including PCE, MC, and Scanning.

By contrast, compatibility with Taylor model-based reachability analysis was more limited. These solvers require smooth ordinary differential equation formulations that can be evaluated under interval arithmetic. In several instances, particularly with models built using `Multibody.jl`, symbolic expressions included branching logic such as `ifelse` constructs, which are not well defined over inter-

vals spanning multiple branches. This incompatibility reflects a structural mismatch rather than a flaw in either `MTK.jl` or the Taylor model approach. While `RA.jl` supports hybrid systems with discrete transitions, translating symbolic models into a hybrid formalism was not attempted here. This limitation is discussed further in the section on future work.

Overall, these findings support the first research question of this thesis: how Julia can be used to systematically analyze how parametric uncertainty affects robotic accuracy. The symbolic modeling capabilities of `MTK.jl`, including equation reuse, dependency tracking, observed variable support, and modular system construction, were essential to enabling efficient and flexible uncertainty-aware simulation.

7.3 Scanning Method

The scanning method, introduced in Section 2.6, is used in this thesis primarily as a reference technique for evaluating the accuracy and efficiency of more advanced uncertainty propagation methods. While conceptually simple and non-intrusive, its computational cost scales exponentially with the number of uncertain parameters. For instance, a uniform grid with 20 points per parameter and three uncertainties requires $20^3 = 8000$ simulations, compared to approximately 1300 simulations for a tenth-order polynomial chaos expansion (PCE) with the same dimensionality. Moreover, the method provides no guarantees beyond the sampled points, as it does not model the structure of the system between them.

Despite these limitations, scanning performs well in low-dimensional problems. With only one uncertain parameter, a 100-point grid yields accurate bounds at modest computational cost, making the use of more sophisticated techniques unnecessary in such cases. The method was also applied successfully in an inverse problem setting. As shown in Figure 6.9, it was used to estimate the torque required to close the scissors under parameter uncertainty. Even with 10,000 Monte Carlo samples, the bounds did not extend beyond those computed by scanning.

In addition to its direct use, scanning serves as a reliable benchmark for validating other methods. Table 6.1 illustrates how scanning can reduce the number of Monte Carlo simulations needed to reach a target accuracy. Accuracy here refers both to the overall error norm and to the maximum deviation over the entire time horizon. In many cases, scanning effectively captures the dominant dynamics of the system.

Nevertheless, the method's scalability limits its applicability. Increasing the number of uncertain parameters from one to three raises the required number of simulations from 100 to 1,000,000 for a uniform 100-point grid. Although selective refinement along the most sensitive dimensions could reduce the computational burden, such structural information is not assumed in the general case considered here.

7.4 Monte Carlo

The Monte Carlo (MC) method remains a standard approach for uncertainty propagation and is included in this thesis primarily as a benchmark. While conceptually simple and straightforward to implement, it does not provide rigorous guarantees and can be computationally demanding. Figure 6.1 illustrates its convergence behavior for the double pendulum system, using a hybrid setup where MC is combined with scanning (MC+SM) to form a high-sample reference solution. The error, computed as the norm between trajectories over a 10-second interval, decreases approximately exponentially with increasing sample size. However, this convergence pattern is problem-dependent and partially influenced by the underlying scanning component, which improves the baseline accuracy relative to standalone Monte Carlo.

In practical applications, memory constraints limited the number of simulations that could be performed in a single batch. In some cases, executing more than 10,000 simulations simultaneously resulted in excessive memory usage or crashes. This issue was partially mitigated by performing simulations in smaller batches (typically 500 at a time) and merging bounds after each batch. Although this strategy appeared to reduce memory pressure, it did not fully resolve the problem, and the underlying cause was not further investigated.

While higher sample counts could improve accuracy, this raises a broader question regarding diminishing returns. In many applications, uncertain parameters are estimated or assumed rather than precisely known. Beyond a certain point, increasing the number of samples may yield negligible improvements relative to the inherent uncertainty in the inputs. This consideration applies to all methods discussed in this thesis and highlights a general limitation in pushing for excessive precision when input distributions are themselves approximations.

Monte Carlo also lacks built-in guidance on when to terminate sampling. Stopping criteria based on output bound stabilization, minimal incremental change, or risk thresholds could provide more efficient convergence but were not implemented here. Incorporating such heuristics into the simulation framework would be straightforward and may be useful in future work.

Unlike other methods considered in this thesis, MC enables the estimation of statistical quantities such as violation probabilities and empirical distributions of peak responses. These outputs are not central to the current analysis but may be valuable in risk-based or probabilistic performance evaluations.

Table 6.1 confirms the method's computational cost. Achieving bounds comparable to a fifth-order PCE (runtime: 5.9 seconds) required 26,337 MC samples and approximately 72 minutes. Although MC performed competitively for lower PCE orders when combined with scanning, this benefit was primarily due to the deterministic coverage provided by scanning, not the stochastic sampling.

In summary, Monte Carlo offers value as a benchmark and for probabilistic insight, but its high computational cost and slow convergence make it less attractive

as a general-purpose method for uncertainty propagation in dynamical systems.

7.5 Monte Carlo and Scanning as Reference Benchmark

A combination of Monte Carlo (MC) sampling and the Scanning Method (SM) is used in this thesis as the reference benchmark for evaluating other uncertainty propagation techniques. This hybrid approach combines stochastic variability with explicit coverage of boundary cases and is motivated by both precedent and practical considerations. For example, Wang et al. [Wang and Yang, 2021] use 10,000 MC samples in their evaluations, which informs the choice of sample counts adopted here. To improve robustness, this thesis employs 50,000 MC samples in combination with SM.

The effectiveness of this configuration is supported by empirical results. As shown in Figure 6.1, the error norm decreases rapidly with sample count, with diminishing returns beyond approximately 30,000 samples. In the double pendulum example, the error norm between the bounds computed with 30,000 and 50,000 samples is below 0.01 over a 10-second simulation, indicating that the chosen configuration provides a sufficiently accurate approximation of the reachable set.

Because a full simulation with 50,000 samples requires approximately six hours, a reduced count of 10,000 samples was used for cases such as the scissors mechanism, where the results section shows that additional samples had negligible impact on the bounds. These settings are consistent with prior work and reflect a balance between accuracy and computational feasibility.

The hybrid approach also offers significant performance advantages over pure MC. As shown in Table 6.1, comparable accuracy can be achieved with far fewer samples and shorter runtimes. For instance, achieving the accuracy of a fifth-order PCE required over 26,000 MC samples and 70 minutes of compute time, while MC+SM attained similar performance in just over 5 seconds.

Overall, MC+SM serves as a practical and high-resolution benchmark, combining dense interior sampling with explicit coverage of parameter space boundaries. It is used throughout this thesis to approximate the reachable set and to assess the accuracy of more structured uncertainty propagation methods such as PCE and Taylor models.

7.6 Taylor Models

7.6.1 Impact of Model Structure and Simplification

As shown in Figure 6.3, different but mathematically equivalent formulations of the double pendulum model produced identical enclosures, confirming the correctness of the symbolic transformations used to convert `ModelingToolkit.jl` models into ODEs.

However, simulation time varied significantly depending on symbolic simplification. As reported in Table 6.3, simplified and structurally optimized models were simulated an order of magnitude faster than their unsimplified counterparts. Manual simplification using MATLAB's symbolic toolbox reduced runtimes substantially.

The `@taylorize` macro from `TaylorModels.jl` had no measurable effect on runtime, and Julia's native simplifier (`Symbolics.jl`) proved insufficient for complex systems. External tools such as `SymPy.jl` may offer improved simplification capabilities, but integration with `ModelingToolkit.jl` is nontrivial due to differences in symbolic backends.

7.6.2 Numerical Robustness and Domain Splitting

Taylor model performance was sensitive to parameter settings such as `orderT`, `orderQ`, and `abstol`, as well as to the size of the uncertainty domain. For larger domains or strongly nonlinear dynamics, simulations frequently failed. This was addressed using domain splitting, where the parameter space was subdivided and each region simulated independently.

In the double pendulum simulation (Figure 6.14), domain splitting enabled a full 10-second reachability analysis. The resulting enclosures closely matched the MC+SM benchmark, with slight overestimation as expected from the interval remainder terms. However, the bounds temporarily failed to enclose the reference trajectory near $t = 8$ and $t = 10$ seconds. This behavior may be attributed to accumulated numerical error or sensitivity introduced by converting the model from a DAE to an ODE formulation, as discussed in Section 2.4.

7.6.3 Time Step Control and Performance Limitations

Although the `TMJets21b` algorithm adaptively computes valid time steps based on local polynomial expansions, performance improved when small steps were enforced manually. By modifying the solver source code, either fixed or maximum step sizes could be imposed.

For example, the double pendulum simulation with a time step size of 4×10^{-5}, `abstol` = 10^{-13}, `orderT` = 8, and `orderQ` = 4 completed successfully, but began diverging from the reference after approximately 5 seconds. These results indicate that even under strict solver tolerances, overestimation and numerical drift may accumulate over time. Enforcing small time steps improves robustness but substantially increases computational cost, limiting practicality for long or stiff simulations.

7.6.4 Sensitivity to Initial Conditions

As shown in Section 6.4.3, simulations of the scissors mechanism failed when initial conditions were specified exactly but succeeded when small uncertainties were introduced. This suggests that the adaptive step-size controller may select steps that are too large when no interval width is present, leading to instability as the system enters more nonlinear regions.

Constraining the maximum time step (e.g., to 5×10^{-4}) restored stability. This behavior was confirmed in Figure 6.12(b), where limiting the step size produced bounds consistent with MC+SM and PCE. The result highlights the importance of step-size control in Taylor model simulations, particularly for near-deterministic initial conditions.

7.6.5 Observability and Postprocessing Limitations

Taylor model simulations in `ReachabilityAnalysis.jl` return validated enclosures only for state variables. In contrast to `ModelingToolkit.jl`, which supports symbolic observation of derived quantities, no postprocessing functionality exists for computing bounds on non-state variables.

While it is theoretically possible to propagate expressions for derived outputs through the polynomial enclosures, this requires explicit arithmetic on Taylor models and is not supported out-of-the-box. A possible workaround is to include derived quantities directly as state variables with associated dynamics, but this adds complexity and requires manual derivation, which may be error-prone in systems with geometric or symbolic dependencies.

7.7 Polynomial Chaos Expansion (PCE)

The Polynomial Chaos Expansion (PCE) implementation used in this thesis follows the collocation-based formulation described in [Wang and Yang, 2021], and the theoretical background outlined in Section 2.9. The method was validated by reproducing published benchmarks, including the double pendulum system shown in Figure 6.14. In addition to replicating reference results, several extensions were developed to improve bounding accuracy and computational efficiency, including diagnostic tools, domain splitting, and heuristic evaluation strategies.

7.7.1 Bounding with Interval Arithmetic vs. Grid Evaluation

After computing the PCE coefficients, bounds on the system response can be extracted using either interval arithmetic or grid evaluation. Interval-based bounding applies known value ranges of the polynomial basis functions to the expansion. While this approach is computationally efficient, it tends to overestimate when many higher-order terms are nonzero. Figures 6.5 and 6.6 demonstrate that this overestimation correlates with the cumulative contribution of higher-order coefficients, primarily due to dependency loss and interval wrapping.

To reduce wrapping, a deterministic grid evaluation strategy was implemented. This involves evaluating the PCE expansion across a uniform grid in the stochastic domain. The resulting bounds are significantly tighter, particularly in nonlinear systems such as the double pendulum (Figure 6.5). However, as shown in Table 6.6, the cost of grid evaluation increases rapidly with the number of uncertain parameters.

Since PCE separates simulation from bounding, this trade-off can be addressed by applying grid evaluation selectively, e.g., at key time points.

7.7.2 Coefficient Diagnostics and Domain Splitting

To understand when wrapping becomes significant, two diagnostics were introduced: coefficient energy (the sum of squared values for terms of order ≥ 2) and entropy of the multi-index distribution. As shown in Figure 6.7, overestimation is most severe when both metrics are high. In contrast, when a few dominant first-order terms capture most of the variation, as in Figure 6.9, interval-based bounds remain tight. This is further supported by Table 6.8, which shows that nearly all significant coefficients are of order one, and higher-order terms are negligible.

To mitigate wrapping without incurring the full cost of grid evaluation, domain splitting was applied. The stochastic domain was divided into axis-aligned subregions, and a separate PCE expansion was computed for each. This reduced local polynomial complexity and improved bounding performance. Table 6.5 shows how dominant coefficients became more concentrated along single parameter directions after splitting.

Empirical results for the double pendulum illustrate the trade-off between accuracy and runtime. As shown in Figure 6.4, increasing the number of splits improves bounding accuracy by reducing overestimation relative to the MC+SM reference. However, Table 6.4 shows that this comes at the cost of increased solve and bounding time. These results confirm that domain splitting improves reliability and tightness of bounds, but must be balanced against the added computational overhead.

This approach also revealed physically meaningful sensitivities. For example, the spring-damper coefficient of joint 2, associated with the robot's longest link, was found to have a particularly strong influence on the system response. Splitting on that parameter improved bounds and highlighted its structural importance. In contrast, splitting along a less influential dimension degraded accuracy under interval bounding, underscoring the need for informed selection of the splitting direction.

7.7.3 Heuristic Evaluation Strategies

To balance accuracy and cost, a hybrid strategy was implemented that switches between interval and grid evaluation based on the sum of Legendre coefficients of order two or higher. Grid evaluation was used when this sum exceeded 0.1; otherwise, interval arithmetic was applied.

Figure 6.8 shows that the heuristic strategy closely approximates the bounds from full grid evaluation. As shown in Table 6.7, it also reduced runtime compared to grid evaluation (0.38 s vs. 0.65 s), while offering a tighter enclosure than interval arithmetic alone.

The threshold was chosen heuristically and not systematically optimized. Although Figures 6.6 and 6.7 suggest potential diagnostics for guiding such decisions, further study is needed to develop more general or predictive strategies.

7.7.4 Time and Order Comparison Strategies

This strategy is particularly suitable for PCE because computing the expansion coefficients is relatively fast, while the main computational cost, especially when using grid evaluation, lies in the bounding phase. By evaluating bounds only at a sparse set of key time points, a large amount of computation can be saved without sacrificing accuracy.

In practice, high-order expansions can be computed at a few selected times and compared against lower-order versions. If the difference is small, the lower-order expansion can be used to simulate the full trajectory at higher temporal resolution. This approach was applied to the circular TCP trajectory in Figure 6.11, where the results from an order-5 expansion closely matched those of a more expensive order-10 simulation.

As shown in Figure 6.10, both the norm and maximum absolute error decrease with increasing PCE order, but the improvements are relatively modest. While the error drops by nearly an order of magnitude over the full range, the absolute change—from around 10^{-2} to 10^{-3}—is small in practical terms. This suggests that lower-order expansions already capture most of the relevant behavior, and higher orders may not be justified unless particularly tight bounds are required.

In addition to bounding costs, this strategy reduces symbolic overhead during postprocessing, such as recomputing observed variables like end-effector position or torque. Moreover, as reported in Table 6.1, a tenth-order PCE can take approximately four times longer than a fifth-order one in grid-based evaluation. Using a sparse grid for the high-order case and deferring full-resolution evaluation to the cheaper model offers a practical compromise between cost and accuracy.

7.8 Comparison of Methods

The scanning and Monte Carlo methods are conceptually simple and broadly applicable. Their main drawback lies in computational cost: both require a large number of simulations to achieve accurate bounds, particularly as the number of uncertain parameters increases. Nonetheless, they are fully non-intrusive and integrate seamlessly with the SciML ecosystem. They require no reformulation of model equations and generalize readily across different systems and model structures. This makes them applicable to high-dimensional systems such as 6-DOF robots, although the number of required simulations can become a limiting factor. As shown throughout this thesis, thousands of samples are often needed for convergence. When model evaluations are expensive, this becomes a serious bottleneck.

PCE shares the non-intrusiveness and general applicability of scanning and Monte Carlo while offering a more favorable trade-off between accuracy and runtime. It was found to be the most robust and scalable method among those evaluated. The number of required simulations grows as $(p+1)^d$, where p is the polynomial order and d the number of uncertainties. For 6-DOF robots, this scaling is man-

ageable when the number of uncertain parameters is kept modest, for example by focusing on a few dominant sources of variation. In such cases, PCE is particularly attractive, offering interpretable results, support for sensitivity diagnostics, and flexible postprocessing. Moreover, low-order expansions—such as order 2 or 3—can provide meaningful insight with relatively low computational cost. This makes PCE well suited for integration into robot programming workflows, where fast feedback and black-box compatibility are important.

Taylor model-based reachability analysis offers mathematically rigorous enclosures and is appealing in applications requiring formal guarantees. However, it proved fragile in practice, with simulations often failing even for moderate systems like the double pendulum. The method is also less compatible with symbolic workflows in `ModelingToolkit.jl`, requiring explicit ODE reformulations and manual simplification. Observed variables and automatic DAE handling are not natively supported. While Taylor models scale well with the number of uncertain parameters—since these are treated as state variables—they are not practically suitable for more complex robotic systems due to their numerical sensitivity and high configuration overhead.

Ultimately, the choice of method should reflect not only theoretical guarantees but also the intended use of the results. In many robotics applications, uncertainties stem from approximated tolerances or rough estimates. A slightly conservative or underapproximated bound may still be useful for design, validation, or calibration. From a practical perspective, low-order PCE or a hybrid approach combining scanning and PCE may offer the best balance between speed, interpretability, and bounding accuracy. This is particularly true in real-world robot programming contexts, where responsiveness and simplicity are essential.

7.9 Research Questions

The research questions guiding this thesis were introduced in Chapter 1.2. They concern how Julia can be used to systematically analyze parametric uncertainty in robotic systems, and what kinds of insights such analyses can yield about system performance and precision.

The results presented in this thesis demonstrate that Julia, together with the symbolic modeling capabilities of `ModelingToolkit.jl`, provides a powerful foundation for uncertainty-aware simulation. Uncertain parameters can be introduced directly into symbolic models as intervals or sample sets, enabling seamless integration with propagation methods such as scanning, Monte Carlo simulation, Polynomial Chaos Expansion (PCE), and Taylor model-based reachability.

A custom package, `IntervalSimulations.jl`, was developed to streamline this workflow. It supports automatic generation of simulations and extraction of output bounds without requiring manual reformulation of the model equations. This structure makes it possible to analyze how uncertainty propagates through a robotic

system and impacts quantities such as joint trajectories, torques, or end-effector positions.

Beyond producing bounded outputs, the methods evaluated here, particularly PCE, enable more detailed diagnostic insight. By analyzing the structure of the PCE coefficients, it becomes possible to identify which uncertain parameters contribute most to output variability. This supports sensitivity analysis and helps guide model refinement, tolerance prioritization, and calibration strategies. Such insights can be used not only for verification, but also for design decision-making in robotic applications where precision is critical.

In summary, Julia's symbolic and numerical toolchain, extended with targeted uncertainty propagation techniques, enables both systematic analysis and interpretive insight into how parametric uncertainty affects robotic system behavior.

8

Conclusions

8.1 Recap and Summary of Findings

This thesis explored how uncertainty in model parameters affects robotic systems and how such uncertainty can be systematically analyzed using Julia. Five methods were evaluated, naive interval arithmetic, scanning, Monte Carlo (MC), Taylor models, and Polynomial Chaos Expansion (PCE), across several case studies, including a double pendulum, a pair of scissors, and a 3-DOF robotic arm.

Key contributions include:

- The development of `IntervalSimulations.jl`, a Julia package for automating uncertainty propagation in `ModelingToolkit.jl` models.
- Implementation and extension of PCE, including heuristic bounding and domain splitting to reduce overestimation.
- An empirical comparison of method performance across varying uncertainty levels, time horizons, and model complexities.
- Insight into dominant uncertainties to guide modeling and design decisions.

8.2 Key Reflections

No single method emerged as universally best. The appropriate choice depends on the system's dimensionality, nonlinearity, and the acceptable level of conservatism. PCE offered the most favorable balance between computational cost, accuracy, and interpretability in most scenarios. While Taylor models provide formal guarantees in theory, they were less robust in practice and sensitive to configuration. Scanning proved reliable and easy to apply in low-dimensional settings, but it does not scale well with increasing uncertainty. Monte Carlo, though conceptually simple and broadly applicable, showed slow convergence and high computational cost in high-resolution use cases.

It is also important to note that input intervals are often based on engineering tolerances or rough estimates. In such cases, approximate bounds that capture the likely range of system behavior may be more useful than exact enclosures, especially if achieving those tight bounds requires substantially more computational effort.

8.3 Future Work

Several directions remain for future development. A key challenge is extending `IntervalSimulations.jl` to support hybrid systems with mode-dependent dynamics. While `ReachabilityAnalysis.jl` supports such models, integration with symbolic workflows from `ModelingToolkit.jl` remains unresolved. Another limitation is the lack of postprocessing for Taylor models, adding support for observed quantities such as torques or end-effector positions would increase practical usability.

Adaptive domain splitting could improve robustness by subdividing the uncertainty domain during simulation based on error indicators. The structure of PCE coefficients may also support automated diagnostics to identify dominant uncertainties and guide refinement or calibration. Benchmarking across growing uncertainty dimensions and model complexity would help characterize performance and failure modes.

Performance could be improved through Julia-specific optimizations. Using `remake` in `MTK.jl` reduces overhead when solving similar problems repeatedly. Ensemble solving via `DifferentialEquations.jl`, with multithreading or GPU support, could accelerate Monte Carlo simulations. Perturbation-based workflows, as demonstrated in the MTK documentation,[1] may also offer a lightweight way to simulate nearby parameter configurations with minimal overhead.

8.4 Final Remarks

This thesis has demonstrated that Julia, supported by its symbolic and numerical libraries, provides a strong foundation for uncertainty-aware simulation. The methods and tools developed, particularly `IntervalSimulations.jl` and the extended PCE framework, enable systematic and reproducible analyses of how parameter uncertainty affects model behavior. While the main focus has been on robotic systems, the same techniques apply broadly to any domain where robust simulation under uncertainty is essential. These contributions support more transparent, flexible, and scalable modeling workflows across science and engineering.

[1] `https://docs.sciml.ai/ModelingToolkit/stable/examples/perturbation/` (accessed May 25, 2025)

Bibliography

Adams, B. M., W. J. Bohnhoff, K. R. Dalbey, M. S. Ebeida, J. P. Eddy, M. S. Eldred, R. W. Hooper, P. D. Hough, K. T. Hu, J. D. Jakeman, M. Khalil, K. A. Maupin, J. A. Monschke, E. E. Prudencio, E. M. Ridgway, F. Rizzi, P. Robbe, A. A. Rushdi, D. T. Seidl, J. A. Stephens, L. P. Swiler, and J. G. Winokur (2025). *Dakota 6.22.0 documentation*. Tech. rep. SAND2025-05563O. Sandia National Laboratories, Albuquerque, NM. URL: http://snl-dakota.github.io (visited on 2025-05-20).

Althoff, M. (2015). "An Introduction to CORA 2015". In: *Proc. of the 1st and 2nd Workshop on Applied Verification for Continuous and Hybrid Systems*. EasyChair, pp. 120–151. DOI: 10.29007/zbkv.

Benet, L. and D. P. Sanders (2019). "TaylorSeries.jl: Taylor expansions in one and several variables in Julia". *Journal of Open Source Software* **4**:36, p. 1043. DOI: 10.21105/joss.01043.

Benet, L. and D. P. Sanders (n.d.). *JuliaDiff/TaylorSeries.jl*. DOI: 10.5281/zenodo.2601941.

Benet, L. and D. P. Sanders (2025). *TaylorModels.jl Documentation*. URL: https://juliaintervals.github.io/TaylorModels.jl/dev/ (visited on 2025-05-02).

Berz, M. and K. Makino (2017). "Rigorous Reachability Analysis and Domain Decomposition of Taylor Models". In: Abate, A. et al. (Eds.). *Numerical Software Verification*. Springer International Publishing, Cham, pp. 90–97. ISBN: 978-3-319-63501-9.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah (2017). "Julia: A Fresh Approach to Numerical Computing". *SIAM Review* **59**:1, pp. 65–98. DOI: 10.1137/141000671.

Bogomolov, S., M. Forets, G. Frehse, K. Potomkin, and C. Schilling (2019). "JuliaReach: a toolbox for set-based reachability". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '19. Association for Computing Machinery, Montreal, Quebec, Canada, pp. 39–44. ISBN: 9781450362825. DOI: 10.1145/3302504.3311804.

Brenan, K. E., S. L. Campbell, and L. R. Petzold (1996). "Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations". In: Society for Industrial and Applied Mathematics. Chap. 2, pp. 15–39. ISBN: 9780898713534. DOI: 10.1137/1.9781611971224.ch2.

Bünger, F. (2020). "A Taylor model toolbox for solving ODEs implemented in MATLAB/INTLAB". *Journal of Computational and Applied Mathematics* **368**, p. 112511. ISSN: 0377-0427. DOI: 10.1016/j.cam.2019.112511.

Busch, M., F. Schnoes, A. Elsharkawy, and M. F. Zaeh (2022). "Methodology for model-based uncertainty quantification of the vibrational properties of machining robots". *Robotics and Computer-Integrated Manufacturing* **73**, p. 102243. ISSN: 0736-5845. DOI: 10.1016/j.rcim.2021.102243.

Carney, M., H. Kantz, and M. Nicol (2020). "Analysis and Simulation of Extremes and Rare Events in Complex Systems". In: Junge, O. et al. (Eds.). *Advances in Dynamics, Optimization and Computation*. Springer International Publishing, Cham, pp. 151–182. ISBN: 978-3-030-51264-4.

Chen, X. (2015). *Reachability analysis of non-linear hybrid systems using Taylor Models*. Doctoral Dissertation. RWTH Aachen University.

Danisch, S. and J. Krumbiegel (2021). "Makie.jl: Flexible high-performance data visualization for Julia". *Journal of Open Source Software* **6**:65, p. 3349. DOI: 10.21105/joss.03349.

FastGaussQuadrature.jl contributors (2025). *FastGaussQuadrature.jl*. URL: https://github.com/JuliaApproximation/FastGaussQuadrature.jl (visited on 2025-05-22).

Gowda, S., Y. Ma, A. Cheli, M. Gwóźzdź, V. Shah, A. Edelman, and C. Rackauckas (2021). "High-performance symbolic-numerics via multiple dispatch". *ACM Communications in Computer Algebra* **55**, pp. 92–96. DOI: 10.1145/3511528.3511535.

Halder, A. and R. Bhattacharya (2010). "Beyond Monte Carlo: A Computational Framework for Uncertainty Propagation in Planetary Entry, Descent and Landing". DOI: 10.2514/6.2010-8029.

JuliaSim contributors (2025). *Multibody.jl – A ModelingToolkit-based library for multibody systems*. https://github.com/JuliaSim/Multibody.jl. (Visited on 2025-05-02).

Lima, F. M. S. (2022). "Lecture notes on Legendre polynomials: their origin and main properties". DOI: 10.48550/arXiv.2210.10942.

Ma, Y., S. Gowda, R. Anantharaman, C. Laughman, V. Shah, and C. Rackauckas (2021). *ModelingToolkit: A Composable Graph Transformation System For Equation-Based Modeling*. arXiv: 2103.05244 [cs.MS].

Manfredi, P. (2025). "A hybrid polynomial chaos expansion – Gaussian process regression method for Bayesian uncertainty quantification and sensitivity analysis". *Computer Methods in Applied Mechanics and Engineering* **436**, p. 117693. ISSN: 0045-7825. DOI: 10.1016/j.cma.2024.117693.

Moore, R. E., R. B. Kearfott, and M. J. Cloud (2009). *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9780898717716.

Nazari, V. and L. Notash (2015). "Motion Analysis of Manipulators With Uncertainty in Kinematic Parameters". *Journal of Mechanisms and Robotics* **8**:2, p. 021014. ISSN: 1942-4302. DOI: 10.1115/1.4031657.

Pantelides, C. C. (1988). "The Consistent Initialization of Differential-Algebraic Systems". *SIAM Journal on Scientific and Statistical Computing* **9**:2, pp. 213–231. DOI: 10.1137/0909014.

Pérez-Hernández, J. A. and L. Benet (2019). *PerezHz/TaylorIntegration.jl: TaylorIntegration v0.4.1*. Version v0.4.1. DOI: 10.5281/zenodo.2562353.

Rackauckas, C. and Q. Nie (2017). "Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia". *Journal of Open Research Software* **5**. DOI: 10.5334/jors.151.

Ranftl, S. and W. von der Linden (2021). "Bayesian Surrogate Analysis and Uncertainty Propagation". *Physical Sciences Forum* **3**:1. ISSN: 2673-9984. DOI: 10.3390/psf2021003006.

Safdarnejad, S. M., J. D. Hedengren, N. R. Lewis, and E. L. Haseltine (2015). "Initialization strategies for optimization of dynamic systems". *Computers & Chemical Engineering* **78**, pp. 39–50. ISSN: 0098-1354. DOI: 10.1016/j.compchemeng.2015.04.016.

Sanders, D. P. and L. Benet (2014). *IntervalArithmetic.jl*. DOI: 10.5281/zenodo.3336308. URL: https://github.com/JuliaIntervals/IntervalArithmetic.jl (visited on 2025-05-02).

SciML contributors (2025). *ModelingToolkitStandardLibrary.jl*. URL: https://github.com/SciML/ModelingToolkitStandardLibrary.jl (visited on 2025-05-21).

Srikanthakumar, S. and W.-H. Chen (2015). "Worst-case analysis of moving obstacle avoidance systems for unmanned vehicles". *Robotica* **33**:4, pp. 807–827. DOI: 10.1017/S0263574714000642.

The MathWorks, Inc. (2024a). *MATLAB, version 24.2.0.2740171 (R2024b)*. The MathWorks, Inc., Natick, Massachusetts.

The MathWorks, Inc. (2024b). *Symbolic Math Toolbox*. Natick, Massachusetts.

Tucker, W. (2011). "Validated numerics. a short introduction to rigorous computations". *Validated Numerics: A Short Introduction to Rigorous Computations*.

Wang, C., Q. Duan, C. H. Tong, Z. Di, and W. Gong (2016). "A GUI platform for uncertainty quantification of complex dynamical models". *Environmental Modelling & Software* **76**, pp. 1–12. ISSN: 1364-8152. DOI: 10.1016/j.envsoft.2015.11.004.

Wang, L. and G. Yang (2021). "An interval uncertainty propagation method using polynomial chaos expansion and its application in complicated multibody dynamic systems". *Nonlinear Dynamics* **105**:1, pp. 837–858. ISSN: 1573-269X. DOI: 10.1007/s11071-021-06512-1.

Xu, Y., L. Mili, A. Sandu, J. Zhao, and M. von Spakovsky (2018). "Propagating uncertainty in power system dynamic simulations using polynomial chaos". *IEEE Transactions on Power Systems* **34**, pp. 338–348. DOI: 10.1109/TPWRS.2018.2865548.

9

Appendix

9.1 Models

9.1.1 Scissors ModelingToolkit

```
@mtkmodel DynamicScissor begin
    @parameters begin
        f      = 0.5 # Viscous friction coefficient
        tau_c  = 1.5 # Coulomb friction torque
        w_brk  = 0.3 # Breakaway friction velocity
        tau_brk =2.4 # Breakaway friction torque
        g =9.82
    end
    @variables begin
        tau(t)=0
        amplitude(t)
    end

    @components begin
        fixed =ModelingToolkitStandardLibrary.Mechanical.
            Rotational.Fixed()
        torque =Torque(use_support =false)
        inertia =Inertia(J =0.2, phi =1.0±0.03, w =0)
        friction =RotationalFriction(f, tau_c, w_brk, tau_brk)
        spring =SpringDamper(d =0.5±0.03, c =0.10±0.02,phi_rel0
            =0.5)
        constant =Constant(k =amplitude)
    end

    @equations begin
        D(tau) ~1
        amplitude ~-3.25/(tau^2+1)
        torque.tau.u ~amplitude
```

```
        connect(fixed.flange, friction.flange_a)
        connect(fixed.flange,spring.flange_a)
        connect(friction.flange_b, inertia.flange_a)
        connect(spring.flange_b, inertia.flange_a)
        connect(torque.flange, inertia.flange_b)
    end
end

@mtkmodel Scissor begin
    @parameters begin
        h =8
        b =1
    end
    @variables begin
        θ(t)
        x_tcp(t)
        y_tcp(t)
    end
    @equations begin
        x_tcp ~(sin(θ/2)*b^2 -4*h*cos(θ/2)*b)/(2*b*cos(θ) +4*h*
            sin(θ)) # Simplified in Matlab
        y_tcp ~(-(1//16)*(b^2)*(sin((1//2)*θ)^2)*cos((1//2)*θ)
            -(1//16)*(b^2)*(cos((1//2)*θ)^3)) /((1//4)*(-(1//2)*
            b*(sin((1//2)*θ)^2) +(1//2)*b*(cos((1//2)*θ)^2) +(2/
            /1)*h*sin((1//2)*θ)*cos((1//2)*θ)))
    end
end

@mtkmodel DynamicScissorSystem begin
    @components begin
        scissor =Scissor()
        dynamics =DynamicScissor()
    end
    @equations begin
        dynamics.inertia.phi ~scissor.θ
    end
end
t_end=5.0

@named sys =DynamicScissor()
@named sys2 =Scissor()

@named sys =DynamicScissorSystem()
fol =structural_simplify(sys)
```

9.1.2 Scissors Final Model

```
function scissor2(du, u, p, t)
    phi, w, c, d, tau =u

    tau2 =tau^2
    cphi =c *phi
    tanhw =tanh(33.33333333 *w)
    ew2 =exp(-5.555555556 *w^2)

    du[1] =w

    du[2] =(
        2.5 *c
        - 2.5 *w
        - 7.5 *tanhw
        - 5.0 *cphi
        - 5.0 *d *w
        + 2.5 *c *tau2
        - 2.5 *tau2 *w
        - 7.5 *tau2 *tanhw
        - 24.73081906 *w *ew2
        - 5.0 *cphi *tau2
        - 5.0 *d *tau2 *w
        - 24.73081906 *tau2 *w *ew2
        - 16.25
    ) / (1 +tau2)

    du[3] =zero(c)
    du[4] =zero(d)
    du[5] =one(tau)
    return du
end
```

9.1.3 3-DOF Multibody.jl

```
t =Multibody.t
world =Multibody.world

l1= 0.5
l2=1.0
l3=0.3
r0 =0.1
a1=l2
a2=l3
```

```
t1=0
x=l2 +r0*cos(t1)
y=l3 +r0*sin(t1)
D =(x^2 +y^2-a1^2-a2^2)/(2*a1*a2)
θ23 =atan(-sqrt(1-D^2),D)
θ13=atan(x,y)-atan(a2*sin(θ23),(a1+a2*cos(θ23)))


θ1 =-θ13
θ2 =-θ23


@mtkmodel Circle2 begin
    @parameters begin
        r=r0
        a1=l2
        a2=l3
    end

    # all symbolic vars (including your intermediates) must be
        declared here:
    @variables begin
        x(t)
        y(t)
        D(t)
        th1(t)
        th2(t)
    end

    @components begin
      θ1 =Blocks.RealOutput()
      θ2 =Blocks.RealOutput()
    end

    @equations begin
      # Cartesian goal
      x   ~ a1 +r*cos(2t)
      y   ~ a2 +r*sin(2t)

      # 2link IK, step by step, each eqn uses `~`
      D   ~ (x^2 +y^2 -a1^2 -a2^2)/(2*a1*a2)
      th2 ~atan(-sqrt(1 -D^2), D)                  # twoarg atan!
      th1 ~atan(x, y) -
            atan(a2*sin(th2) ,(a1 +a2*cos(th2)))
```

```
      # drive outputs
      θ1.u ~-th1*145
      θ2.u ~-th2*145
    end
  end


systems =@named begin
    joint =Revolute(n =[0, 0, 1],phi0=pi/2, axisflange=true)
    joint2 =Revolute(n =[0, 0, 1],phi0=θ1, axisflange=true)
    joint3 =Revolute(n =[0, 0, 1], phi0=θ2,axisflange=true)
    linkmass =Body(; m =200.0, r_cm =[l1, 0.0, 0.0], I_33 =1.5)
    linkmass2 =Body(; m =50.0, r_cm =[l2, 0.0, 0.0], I_33 =1.5)
    linkmass3 =Body(; m =50.0, r_cm =[l3, 0.0, 0.0], I_33 =1.5)
    bar =FixedTranslation(r =[l1, 0, 0])
    bar2 =FixedTranslation(r =[l2, 0, 0])
    bar3 =FixedTranslation(r =[l3, 0, 0])
    motor =Rotational.Inertia(J =1e-4)
    motor2 =Rotational.Inertia(J =1e-4)
    gearbox =Rotational.IdealGear(ratio =145)
    gearbox2 =Rotational.IdealGear(ratio =145)
    springdamper =Rotational.SpringDamper(; c =1e5, d =1000) #d
    springdamper2 =Rotational.SpringDamper(; c =10.0, d =0.01)
        #c
    springdamper3 =Rotational.SpringDamper(; c =10.0, d =0.01)
        #c
    positiondriver =Rotational.Position()
    positiondriver2 =Rotational.Position()
    positiondriver3 =Rotational.Position()
    input =Blocks.Constant(;k=pi/2)
    circle_input= Circle2()
end
push!(systems, world)


eqs =[
    # base →joint 1
    connect(world.frame_b, joint.frame_a),
    connect(joint.frame_b, linkmass.frame_a),
    connect(joint.frame_b, bar.frame_a),

    connect(input.output, positiondriver.phi_ref),
    connect(positiondriver.flange, springdamper.flange_a),
    connect(springdamper.flange_b, joint.axis),

    # link 1 geometry
```

```
    connect(bar.frame_b, joint2.frame_a),
    connect(joint2.frame_b, linkmass2.frame_a, bar2.frame_a),

    ### C O R R E C T E D ###
    connect(circle_input.θ1, positiondriver2.phi_ref),
    connect(positiondriver2.flange, motor.flange_a),
    connect(motor.flange_b, springdamper2.flange_a),
    connect(springdamper2.flange_b, gearbox.flange_a),
    connect(gearbox.flange_b, joint2.axis),

    # link 2 geometry
    connect(bar2.frame_b, joint3.frame_a),
    connect(joint3.frame_b, linkmass3.frame_a, bar3.frame_a),

    #### C O R R E C T E D ###
    connect(circle_input.θ2, positiondriver3.phi_ref),
    connect(positiondriver3.flange, motor2.flange_a),
    connect(motor2.flange_b, springdamper3.flange_a),
    connect(springdamper3.flange_b, gearbox2.flange_a),
    connect(gearbox2.flange_b, joint3.axis),
]

@named model =ODESystem(eqs, t; systems)
model =complete(model)

ssys =structural_simplify(IRSystem(model))

defs =[
    world.g =>0,
    positiondriver.w =>0.0,
    positiondriver.phi =>pi/2,
    positiondriver3.w =>0.0,
    positiondriver2.phi =>θ1*145,
    positiondriver2.w =>0.0,
    positiondriver3.phi =>θ2*145
]
prob =ODEProblem(ssys, defs, (0, 10.0))
```

9.2 Hardware and Software Specifications

All simulations were performed on a personal computer with the following specifications:

Component	Specification
CPU	Intel Core i7-12700F (20 cores, ∼2.1GHz)
GPU	NVIDIA GeForce RTX 3060 Ti (8 GB VRAM)
RAM	32 GB DDR4
Operating System	Windows 11 Pro, running WSL2
WSL Environment	Ubuntu 22.04.5 LTS
Julia Version	1.11.4 (1.10.9 for Multibody)

Table 9.1 System specifications used for all numerical simulations and benchmarks.