UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

Numerical Techniques for Astrodynamics: Application to Very Large Satellite Constellations

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By RYAN DONALD GAST

Norman, Oklahoma

2026

Numerical Techniques for Astrodynamics: Application to Very Large Satellite Constellations

A THESIS APPROVED FOR THE

SCHOOL OF AEROSPACE AND MECHANICAL ENGINEERING

BY THE COMMITTEE CONSISTING OF

Dr. Diogo Merguizo Sanchez, Chair

Dr. Iman Ghamarian

Dr. Prakash Vedula

UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

Abstract

School of Aerospace and Mechanical Engineering

Master of Science

Numerical Techniques for Astrodynamics: Application to Very Large Satellite Constellations

Ryan Donald Gast

Efficient numerical methods are essential for astrodynamics simulation and analysis. The astrodynamics community often depends on legacy implementations that do not leverage modern programming paradigms. This work explores the adoption of contemporary systems-oriented programming languages to integrate advanced numerical methods into astrodynamics workflows, improving computational performance, accuracy, and maintainability.

This thesis develops a pure Rust implementation of classical numerical integrators (DOPRI5, DOP853, RADAU5, and BDF) with algorithmic fidelity to original Fortran codes. Python bindings provide a SciPy-compatible interface while leveraging Rust's performance.

Benchmarking demonstrates that Rust implementations match or exceed Fortran performance by 2–11% on representative problems. Python bindings achieve 2–6× speedups for non-stiff problems and 12–24× for stiff problems versus SciPy.

The VLISA case study evaluates three Sun-Earth constellation concepts for a very-long-baseline gravitational-wave observatory. The L4-L5-AEP configuration is identified as the optimal architecture, whereas natural configurations serve as lower-propulsion alternatives.

This work challenges longstanding assumptions about programming language performance in scientific computing and demonstrates that modern languages can provide superior performance and maintainability for high-fidelity astrodynamics simulations.

Table of Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In the early 17th century, Johannes Kepler identified the empirical laws of planetary motion, establishing that planets move on ellipses with the Sun at one focus. The dynamical explanation came later with Newton's laws and gravitation, which provided the foundation for the analytical two-body problem that remains central to astrodynamics [6]. Yet once a third body is introduced, atmospheric drag is included, or the non-uniform distribution of mass within a celestial body is considered, closed-form solutions are generally no longer available.

This complexity is where differential equations become essential. Rather than merely describing where a spacecraft is, differential equations describe how it moves: how gravitational forces, atmospheric effects, and other perturbations continuously alter its trajectory. The physics of motion is fundamentally about change, and differential equations are the mathematical language of change. While analytical solutions in classical orbital mechanics can determine a satellite's current position, differential equations reveal the full dynamic picture: how that position evolves moment by moment under the influence of multiple competing forces.

However, differential equations alone are not enough. To predict where a spacecraft will be an hour, a day, or a year from now, we must integrate these equations. For the complex, multi-body problems encountered in real astrodynamics missions, analytical solutions beyond the classical two-body setting are rarely available. Numerical methods thus become essential.

Solving these differential equations with sufficient accuracy requires complex multi-stage numerical integrators such as Runge-Kutta methods, calculations that are unrealistic and impractical to perform by hand. Since the dawn of the computer age, the work of solving differential equations to predict spacecraft trajectories, a process known as orbit propagation, has been automated through computational methods.

One of the earliest programming languages used for such scientific computations was Fortran, developed in the 1950s. Fortran's efficiency in numerical calculations, strong compiler ecosystem, and extensive library base made it the natural choice for implementing orbit propagation algorithms [7], [8], [9]. Over the decades, numerous Fortran libraries and software packages were created to perform orbit propagation using various numerical methods, including Runge-Kutta integrators. Fortran also remains an active language in contemporary scientific computing, particularly in domains with mature numerical kernels and long-lived production workflows. At the same time, much critical astrodynamics software remains embedded in legacy Fortran codebases, and the

human cost of understanding and rewriting these programs for more modern software ecosystems remains high relative to the cost of maintaining existing implementations.

The initial motivation for this thesis arose from efforts to understand and work with existing Fortran-based numerical integrators for astrodynamics. The relevant books, documentation, and codebases were often sparse and difficult to interpret. To address this, the most widely used numerical integrators were translated and rewritten from Fortran into Rust. The expected outcome was performance parity, since both languages compile to efficient machine code. Instead, the Rust implementations were either more efficient on some benchmarks or statistically indistinguishable from their Fortran counterparts on others. A plausible explanation for Fortran's continued dominance is not that modern languages cannot compete, but that scientific computing has accumulated decades of validated infrastructure, community practice, and mission heritage around Fortran implementations. The results of this work show that this historical advantage does not necessarily imply an intrinsic performance advantage on the benchmark problems studied here.

This thesis aims to provide clear descriptions and implementations of numerical integrators for astrodynamics, enabling future scientists and engineers to more easily understand, implement, and utilize these algorithms in modern software ecosystems. It also compares these modern implementations against legacy Fortran versions to test whether modern programming languages can match Fortran's efficiency on representative orbit-propagation problems. The Very Large Interferometer Space Antenna (VLISA) then serves as the mission-design case study for this numerical work: its multi-spacecraft, long-duration, high-accuracy propagation requirements make it a demanding application for the methods developed here. In addition, the analysis creates an emerging gravitational-wave observatory that could extend the reach of space-based gravitational-wave astronomy into the microhertz regime.

1.1 Research Questions

This thesis addresses the following research questions:

1. How do modern Rust implementations of classical numerical integrators (DOPRI5, DOP853, RADAU5, BDF) compare to legacy Fortran versions in terms of computational performance and accuracy for astrodynamics applications?
2. How can Python bindings for Rust-based numerical integrators be designed to provide SciPy-compatible interfaces while achieving significant performance improvements over existing Python-only implementations?

3. What practical guidelines can be established for selecting appropriate numerical integration methods for different classes of astrodynamics problems (non-stiff vs stiff, low vs high accuracy, short vs long duration)?

As a mission-design case study using the developed methods:

4. What orbital configurations for the VLISA mission concept provide feasible constellation designs that maintain stable triangular geometry while enabling microhertz gravitational wave detection?

1.2 Thesis Objectives

To address these research questions, this thesis pursues the following objectives:

1. Develop a comprehensive numerical integration library in Rust that implements classical integrators (DOPRI5, DOP853, RADAU5, BDF) with algorithmic fidelity to the original Fortran implementations by Hairer and Wanner.
2. Create Python bindings via PyO3 that provide a SciPy-compatible interface, enabling seamless adoption in existing Python workflows while leveraging Rust's performance.
3. Establish rigorous verification and validation methodologies demonstrating that the implemented methods achieve theoretical convergence rates, accurately solve test problems, and produce physically meaningful results for astrodynamics applications.
4. Conduct comprehensive benchmarking comparing Rust, Fortran, and Python implementations across representative test problems, using statistical methods to quantify performance differences with confidence intervals.
5. Analyze and evaluate orbital configurations for VLISA using the circular restricted three-body problem with perturbation models, identifying feasible constellation designs that meet mission requirements.
6. Provide practical recommendations and guidelines for selecting appropriate numerical methods for different classes of astrodynamics problems, based on performance characteristics, accuracy requirements, and problem dynamics.

1.3 Contributions

This thesis makes the following contributions to the field of astrodynamics and scientific computing:

1. **ivp Library**: A pure Rust implementation of numerical integrators for initial value problems with comprehensive method suite (RK4, RK23, DOPRI5, DOP853,

RADAU5, BDF), featuring native Rust API with trait-based problem definition, Python bindings with SciPy-compatible interface, dense output interpolation, and event detection capabilities.

2. **Performance Benchmarking**: Comprehensive benchmarking methodology and results demonstrating that Rust implementations achieve practical performance parity with Fortran on representative problems, with Python bindings achieving 2.5–6.2× speedups over SciPy for non-stiff problems and 11.8–27.7× speedups for stiff problems.
3. **Reproducible Research Framework**: Complete open-source codebase including Rust library, Python benchmarks, VLISA simulation scripts, and statistical analysis tools, enabling reproducibility and extension of this work by the research community.
4. **Practical Guidelines**: Method selection guidelines and practical recommendations for choosing appropriate numerical integrators based on problem characteristics, accuracy requirements, and computational constraints, supported by quantitative performance data and practical examples.
5. **VLISA Mission Analysis**: A mission-design study of VLISA as a candidate architecture for extending space-based gravitational-wave astronomy into the microhertz regime. This includes baseline constellation concepts, a common dynamical and numerical evaluation framework, and a trade study that identifies the L4-L5-AEP configuration as the optimal architecture.

1.4 Thesis Organization

This thesis is organized into eight chapters. Following this introduction, Chapter 2 provides a comprehensive literature review covering the historical development of numerical integration methods for astrodynamics, modern implementation languages and performance characteristics, and the design of space-based gravitational wave missions including LISA and VLISA concepts.

Chapter 3 establishes the mathematical foundations of orbital dynamics, including the circular restricted three-body problem, perturbation models (solar radiation pressure, third-body effects), variational equations for sensitivity analysis, and measurement and noise models relevant to precision metrology.

Chapter 4 presents the numerical methods for astrodynamics developed in this thesis, covering explicit Runge-Kutta methods (DOPRI5, DOP853), implicit Runge-Kutta methods (RADAU5), and Backward Differentiation Formulas (BDF), with detailed

algorithm descriptions, complexity analysis, and practical guidelines for method selection.

Chapter 5 describes the software design and architecture of the `ivp` library, including design goals, modular architecture, native Rust and Python APIs, testing strategy, and reproducibility considerations.

Chapter 6 presents the verification, validation, and benchmarking methodology, including error metrics and tolerances, test problems (harmonic oscillator, Van der Pol, Arenstorf orbit, two-body propagation), reference solution strategies, and comprehensive performance comparisons.

Chapter 7 presents the VLISA mission analysis, describing the mission motivation, system model and equations of motion, constellation configurations analyzed, simulation methodology, results, sensitivity analysis, and engineering considerations for a proposed microhertz gravitational-wave observatory architecture.

Chapter 8 concludes with a summary of contributions, limitations of the current work, and recommendations for future research including implementation of symplectic integrators, GPU acceleration, and higher-fidelity VLISA mission modeling.

2 Literature Review

2.1 Historical Development of Numerical Integration Methods

The numerical integration of differential equations for orbital mechanics has a rich history spanning over a century. The pioneering work of Cowell and Crommelin in their investigation of Halley's comet, particularly for its 1910 return, established what is now known as Cowell's formulation: a direct numerical integration method for solving the equations of motion in Cartesian coordinates [10]. This foundational approach remains widely used in modern astrodynamics because it can incorporate perturbations directly without requiring special transformations.

The development of numerical integration methods accelerated dramatically with the advent of electronic computers in the mid-20th century. Fortran, developed in the 1950s, became widely adopted for scientific computing due to its efficiency in numerical calculations and its design for mathematical formula translation [7]. The language's performance characteristics, standardization, and extensive numerical libraries contributed to its enduring use in orbit propagation software.

2.2 Single-Step Methods: Runge-Kutta Family

2.2.1 Explicit Runge-Kutta Methods

The Runge-Kutta family of methods represents one of the most important classes of numerical integrators for orbital mechanics. The classical fourth-order Runge-Kutta method (RK4) has served as a workhorse in trajectory analysis since the early computing era [11], achieving fourth-order accuracy through four derivative evaluations per step. While RK4 uses fixed step sizes, it remains valuable for problems where the dynamics are well-characterized and uniform stepping is acceptable.

A significant advancement came with embedded Runge-Kutta methods, which provide automatic error estimation and adaptive step size control by using two formulas of different orders [12]. The Bogacki-Shampine method (RK23) provides a third-order solution with an embedded second-order error estimate, offering efficient low-accuracy integration with minimal function evaluations per step.

The Dormand-Prince method represents a further refinement of embedded Runge-Kutta schemes. The DP5(4) method (also known as DOPRI5), which uses a fifth-order formula for propagation and a fourth-order formula for error estimation, has become the default integrator in many modern numerical software packages including MATLAB's ode45 and SciPy's default solver [13], [14]. The method samples the derivative function

seven times per step, with the First Same As Last (FSAL) property reducing this to six new evaluations when the step is accepted [13]. The coefficients were carefully optimized to minimize truncation error while maintaining efficient step size control [13].

For high-accuracy applications, the eighth-order Dormand-Prince method (DOP853) provides significantly improved accuracy with 12 stages and dual error estimators of orders 5 and 3 [8]. The method includes seventh-order dense output for continuous interpolation within accepted steps.

2.2.2 Implicit Runge-Kutta Methods

Implicit Runge-Kutta (IRK) methods have emerged as powerful alternatives for orbit propagation, particularly when high accuracy over long integration periods is required or when the problem exhibits stiffness. Unlike explicit methods, where the solution at the next time step can be computed directly, implicit methods require solving a system of nonlinear algebraic equations at each step [8], [15].

The Radau IIA family of implicit Runge-Kutta methods, particularly the three-stage fifth-order RADAU5 method, has become a standard choice for stiff differential equations [9]. RADAU5 is based on collocation at Radau quadrature points and possesses L-stability, meaning it strongly damps stiff components of the solution. The method can also handle differential-algebraic equations (DAEs) of index up to 3, making it versatile for constrained dynamical systems [9]. The implementation uses simplified Newton iteration with adaptive step size control, providing robust performance across a wide range of problem types. For problems where the Jacobian is expensive to compute, numerical differentiation provides an automatic fallback.

Research by Aristoff and Poore has demonstrated that implicit Runge-Kutta methods can be significantly more efficient than explicit methods for orbit propagation when implemented with sophisticated iteration schemes, with efficiency gains ranging from 30 to 95% depending on the orbital regime [15].

2.3 Multistep Methods

Beyond single-step Runge-Kutta methods, multistep methods represent another major class of numerical integrators applied to orbit propagation. These methods use information from multiple previous time steps to compute the solution, typically requiring fewer function evaluations per step than single-step methods of comparable order [8], [9].

2.3.1 Backward Differentiation Formulas

The Backward Differentiation Formula (BDF) methods have become a standard approach for stiff differential equations [9]. BDF methods are implicit multistep schemes that approximate the derivative using a polynomial fitted through the current and previous solution values. A k-step BDF method achieves order k, with methods up to order 5 commonly used in practice. Higher-order BDF methods (order 3 and above) sacrifice A-stability but remain stable for problems whose eigenvalues lie in appropriate regions of the complex plane [9].

Variable-order BDF implementations automatically adjust the method order (typically between 1 and 5) based on solution behavior, using lower orders during transients and higher orders during smooth evolution. Combined with adaptive step size control, variable-order BDF methods provide robust performance across a wide range of stiff problems. The quasi-constant step size formulation simplifies the implementation while maintaining efficiency [9].

2.3.2 Other Multistep Approaches

Adams-Bashforth-Moulton predictor-corrector schemes provide an alternative multistep approach, using explicit Adams-Bashforth predictors refined by implicit Adams-Moulton correctors [8]. Variable-order, variable-step implementations such as Krogh's method have been particularly successful for spacecraft trajectory computation [16]. Extrapolation methods, notably the Gragg-Bulirsch-Stoer (GBS) method, achieve high-order accuracy through Richardson extrapolation combined with the modified midpoint method [8], [17], [18]. While these methods offer certain advantages for specific problem classes, the combination of embedded Runge-Kutta methods (RK23, DOPRI5, DOP853), implicit Runge-Kutta (RADAU5), and BDF methods provides comprehensive coverage for both stiff and non-stiff problems.

2.4 Modern Implementation Languages and Performance

2.4.1 The Dominance of Fortran

For decades, Fortran has been the predominant language for implementing numerical integrators in astrodynamics. The language's design for numerical computing, its native support for multi-dimensional arrays, and decades of compiler optimization have contributed to its continued use in scientific computing. Major libraries such as LAPACK and BLAS, which underpin scientific computing worldwide, are primarily written in Fortran.

Fortran's longevity has also created challenges. Much critical scientific code remains "locked" in mature legacy codebases, with the human cost of understanding and rewriting those implementations being prohibitively high. At the same time, Fortran is not a stagnant language: modern standards, active compiler development, and continuing use in physics, astrophysics, meteorology, and high-performance computing have kept it relevant well beyond its earliest procedural style. The practical issue for this thesis is therefore not whether Fortran is obsolete, but whether modern implementations in other languages can reproduce the same numerical quality while fitting better into contemporary software ecosystems.

2.4.2 Python and High-Level Scientific Computing

Python has emerged as a widely used language for scientific computing workflows, despite being interpreted and inherently slower than compiled languages. The key to Python's success lies in its role as a "gluing layer" that orchestrates compiled numerical routines. Libraries such as NumPy, SciPy, and others provide Python interfaces to highly optimized C and Fortran code.

For numerical integration specifically, SciPy's integrate module provides Python access to well-established integrators, including LSODA (automatic stiff/non-stiff switching), Dormand-Prince methods, and Runge-Kutta-Fehlberg schemes [19]. The development speed advantages and rich ecosystem often make Python attractive for many applications, despite performance overhead compared to fully compiled implementations.

2.4.3 Rust and Modern Systems Languages

Rust represents a new generation of systems programming languages that challenge long-standing assumptions about the trade-offs between safety and performance. As a compiled language like Fortran and C++, Rust can achieve comparable performance. Moreover, its sophisticated compiler optimizations often exceed those of traditional languages, enabling superior performance in numerical computing applications while maintaining human readability and maintainability.

Rust's key advantages include memory safety guarantees without garbage collection, zero-cost abstractions that allow high-level code to compile to efficient machine code, and modern tooling that improves developer productivity. For scientific computing, Rust's type system and borrow checker eliminate entire classes of bugs at compile time while maintaining performance competitive with C and Fortran.

Numerical computing libraries in Rust, such as ndarray (equivalent to NumPy) and nalgebra (for linear algebra), have demonstrated competitive performance with estab-

lished solutions. The language's support for SIMD operations, parallel processing, and integration with BLAS/LAPACK libraries provides a solid foundation for high-performance numerical work.

2.4.4 Comparative Performance Studies

Comparative studies consistently show that implementation quality and algorithm choice often matter more than language identity alone. Compiled languages such as Fortran and C remain strong baselines for compute-bound kernels, Python remains dominant for orchestration and analysis through compiled backends, and Rust is increasingly relevant where performance, safety, and interoperability are all required. For this thesis, the important gap is not proving that one language always wins, but showing that modern Rust implementations can compete directly with legacy Fortran while fitting naturally into Python-centric workflows.

Table 1 places the three implementation languages most relevant to this thesis in historical context and highlights their distinct roles in current scientific-computing workflows.

Language	**First release**	**Typical role in scientific computing**	**Relevance to this thesis**
Fortran	1957	Numerical kernels, legacy production solvers, HPC applications	Reference implementations for DOPRI5, DOP853, and RADAU5
Python	1991	Workflow orchestration, analysis, high-level scientific scripting	Target interface through SciPy-compatible bindings
Rust	2015	Systems programming with safety and performance	Primary implementation language for `ivp`

Table 1: Representative timeline of the implementation languages most relevant to this thesis. The point is not that newer languages replace older ones, but that each now occupies a distinct role in scientific-computing workflows.

2.4.5 Recent Developments in Numerical Computing (2018–2024)

Recent developments have been driven by three broad trends. First, GPU acceleration has become increasingly important for ensemble propagation and Monte Carlo analyses, though it requires substantial restructuring to overcome memory-transfer overhead.

Second, Julia has shown that high-level syntax and high performance can coexist in a scientific-computing language, particularly through the DifferentialEquations.jl ecosystem [20]. Third, modern astrodynamics workflows increasingly mix traditional integrators with faster ephemeris evaluation, surrogate models, and hybrid analytical-numerical techniques. Together, these developments reinforce that current numerical computing is shaped as much by software architecture and hardware awareness as by classical method design.

2.4.6 Critical Evaluation of Integration Methods

Selecting an appropriate numerical integration method requires understanding the trade-offs between computational cost, accuracy, stability, and implementation complexity. This subsection provides critical evaluation of the method families reviewed earlier.

Performance Trade-offs: Explicit Runge-Kutta methods (DOPRI5, DOP853) excel for non-stiff problems due to their low per-step computational cost. However, they become inefficient for stiff problems where step size is limited by stability rather than accuracy. Implicit methods (RADAU5, BDF) incur higher per-step costs due to solving nonlinear systems but can use larger step sizes for stiff problems, often resulting in overall speedups of 30–95% for stiff dynamics [15]. The threshold stiffness where implicit methods become advantageous depends on problem-specific factors including system dimension, Jacobian evaluation cost, and the nature of stiffness.

Stiffness Detection Accuracy: Automatic stiffness detection in explicit integrators relies on heuristics such as monitoring the ratio of consecutive step sizes or derivative magnitudes. These heuristics work well for many problems but can produce false positives (unnecessarily suggesting stiff methods) or false negatives (failing to detect stiffness until performance degrades). RADAU5 includes built-in stiffness monitoring that can warn users when explicit methods are struggling, but definitive detection remains challenging. For critical applications, manual analysis of the Jacobian eigenvalues or preliminary runs with both explicit and implicit methods is often warranted.

Memory Footprint Considerations: High-order explicit methods like DOP853 require storing fifteen stage vectors, increasing memory usage compared to lower-order methods. For problems with large state vectors (e.g., ensemble propagations or finite element discretizations), this memory overhead becomes significant. Implicit methods require additional storage for Jacobian matrices and LU decompositions. BDF methods minimize memory requirements by storing only a few previous solution values. For

memory-constrained environments (embedded systems, GPU memory), method selection must balance accuracy requirements against available memory.

Parallelization Potential: Explicit methods offer greater parallelization opportunities since stage evaluations are independent (except for the sequential nature of Runge-Kutta methods). Multiple right-hand-side evaluations can be distributed across cores or GPUs. Implicit methods present parallelization challenges due to the sequential nature of Newton iterations and the need for linear algebra solvers, though parallel LU decompositions and matrix-vector multiplications can provide some speedups. Multistep methods have limited parallelization potential within individual steps but can parallelize across multiple independent trajectories.

2.4.7 Gap Analysis

While the existing literature provides comprehensive coverage of numerical integration methods and astrodynamics applications, several gaps remain unaddressed. This thesis contributes to bridging these gaps.

Gap	**Current**	**Needed**	**This Thesis**
Modern language implementations	Limited	Rust	✓ ivp library
Python bindings	SciPy slow	High-performance	✓ PyO3 bindings
Comprehensive benchmarking	Limited	Statistical	✓ Welch t-tests
Method selection guidelines	Ad-hoc	Systematic	✓ Decision criteria

Table 2: Gap analysis in astrodynamics numerical integration literature

The gap analysis in Table 2 summarizes the key gaps identified in the literature. Modern languages like Rust and Julia offer potential performance and safety advantages but lack comprehensive numerical method implementations comparable to mature Fortran libraries. Python remains popular for scientific workflows but existing integrators suffer from interpreter overhead. Space-based gravitational-wave mission studies have likewise focused primarily on LISA-like architectures, with limited exploration of Very Large Interferometer concepts that operate in a different dynamical and systems-design regime.

Methodological gaps include the lack of rigorous statistical benchmarking with confidence intervals and systematic guidelines for method selection based on problem characteristics. This thesis addresses these gaps by providing comprehensive implementations, statistical benchmarking methodology, systematic selection guidelines, and a complete open-source codebase enabling reproducibility and extension.

In addition, this thesis applies these numerical methods to the Very Large Interferometer Space Antenna (VLISA), a successor concept to LISA that targets microhertz gravitational wave detection through novel constellation designs. The VLISA problem depends directly on long-duration multi-spacecraft propagation, constellation geometry control, and the numerical credibility of the underlying simulations.

2.4.8 Technology Trends and Future Directions

Several technology trends emerge from this review. Astrodynamics software is moving away from monolithic mission-analysis packages toward modular ecosystems assembled from reusable libraries. At the same time, Python-first workflows continue to dominate user interaction, which increases the value of compiled backends that preserve familiar interfaces. GPU acceleration, reproducible research practices, and memory-safe high-performance languages all fit within this broader shift toward tools that are both faster and easier to integrate into modern workflows.

The `ivp` library developed in this thesis is positioned within that trend: a modular, reproducible, memory-safe implementation that exposes high-performance solvers to both Rust and Python users.

2.5 Applications to LISA and Multi-Body Missions

2.5.1 LISA Mission Overview

The concept of space-based gravitational wave detection has evolved over several decades. In 1989, Bender et al. proposed the Laser Gravitational-Wave Observatory in Space (LAGOS), envisioning a 100 km antenna operating primarily between 10^{-3} Hz and 1 Hz [21]. This early concept aimed to observe continuous gravitational waves from binary systems and detect pulses potentially emitted during galaxy formation [21].

The modern Laser Interferometer Space Antenna (LISA) represents the culmination of this vision and stands as one of the most ambitious space missions ever conceived. LISA will be the first space-based gravitational wave observatory, consisting of three spacecraft forming an equilateral triangle with 2.5 million kilometer arms [22], [23]. The constellation will trail Earth in its orbit around the Sun, detecting gravitational waves in the millihertz frequency range (0.1 mHz to 0.1 Hz) through laser interferometry [22], [24].

Each LISA spacecraft contains free-falling test masses whose positions are measured with picometer precision. Passing gravitational waves alter the distances between these masses, creating detectable signals [22]. The mission will observe supermassive black hole mergers, compact binary systems throughout the Milky Way, and potentially

gravitational wave signals from the early universe [22]. In 2024, the European Space Agency formally approved LISA for development, marking a major milestone toward launch [23].

2.5.2 Orbital Dynamics Challenges

The precise orbit design and propagation requirements for LISA pose significant challenges. Folkner et al. conducted an early comprehensive study of LISA orbit selection and stability, examining how the triangular formation experiences distance variations due to the elliptical orbits around the Sun [24]. They described and compared different orbital configuration options, noting that changes in the distances between spacecraft cause Doppler shifts in the laser signals that directly impact measurement accuracy [24]. The stability of the spacecraft formation is critical for maintaining interferometric performance, requiring high-fidelity numerical orbit propagation throughout the mission design process [24].

The LISA Definition Study Report provides detailed specifications for the mission, including stringent requirements for orbit knowledge and control needed to maintain the precise formation geometry over the multi-year mission duration [22]. These demanding requirements drive the need for accurate numerical integration methods capable of long-term propagation with minimal error accumulation.

2.5.3 Lagrange Points and Artificial Equilibrium Points

The concept of artificial equilibrium points (AEPs) has significant relevance for space-based observatories. In the circular restricted three-body problem, artificial equilibrium points can be generated by applying continuous low-thrust to balance gravitational and centrifugal forces at non-natural equilibrium locations [25], [26].

Baig and McInnes pioneered the study of artificial halo orbits around AEPs using low-thrust propulsion [25], [27]. Their work showed that spacecraft can maintain stable positions at locations inaccessible to passive orbits by applying modest continuous thrust. This concept has applications for solar-sail-stabilized observatories and other formation flying missions.

The stability properties of AEPs have been extensively analyzed. Bombardelli and Peláez demonstrated that minimum-control artificial equilibrium points can be Lyapunov stable when the spacecraft coorbits with a smaller primary at appropriate distances [26]. For the Sun-Earth system, stable configurations exist that could support long-duration science missions [26].

Extensions of the AEP concept to more general propulsion systems have been investigated. Aliasi et al. introduced the concept of a "generalized sail" that encompasses solar sails, magnetic sails, and electric sails within a single mathematical framework [28], [29]. De Almeida et al. provided methods for determining the thrust requirements to generate artificial equilibrium points in binary systems [30].

2.5.4 Halo Orbits and Mission Applications

Halo orbits represent three-dimensional periodic orbits around the collinear Lagrange points. These orbits have been extensively used for space missions, including the Solar and Heliospheric Observatory (SOHO) at Sun-Earth L1 and the James Webb Space Telescope at Sun-Earth L2 [31].

The dynamics of spacecraft in halo orbits and their use for formation flying have been studied by multiple authors. Scheeres and Vinh investigated the control of relative motion in unstable halo orbits, relevant for interferometric missions [32]. The natural dynamics near Lagrange points provide both challenges and opportunities for multi-spacecraft missions [31].

Bookless and McInnes explored the use of solar sail propulsion for control of Lagrange point orbits, including the generation of artificial libration points sunward of L1 or earthward of L2 [33]. Waters and McInnes studied the homoclinic dynamics of solar sail orbits in the three-body problem [34].

2.5.5 Extension to VLISA

The Very Large Interferometer Space Antenna (VLISA) concept extends LISA's capabilities by dramatically increasing the arm length to approximately 2.7×10^8 km. This extension would shift the observable gravitational wave frequencies into the microhertz range, enabling detection of different source classes.

The orbital design challenges for VLISA are substantially more complex than those for LISA. The choice of numerical integration methods becomes critical for ensuring accurate long-term propagation of such an extended formation. While LISA's relatively compact 2.5 million kilometer constellation has been extensively studied, the dynamics of VLISA's vastly larger configuration at roughly 1.7 AU present new challenges for mission designers. High-fidelity numerical orbit propagation is therefore not only a computational convenience but a prerequisite for evaluating whether such a mission concept can be made dynamically and operationally credible.

3. Mathematical Foundations of Orbital Dynamics

3.1. Circular Restricted Three-Body Problem

The circular restricted three-body problem (CR3BP) provides a foundational framework for analyzing spacecraft motion in the gravitational field of two primary bodies [6]. This classical problem considers two massive bodies, m_1 and m_2, orbiting their common center of mass in circular orbits, while a third body of negligible mass m moves under their combined gravitational influence without affecting their motion.

3.1.1. Problem Formulation

Consider a system where m_1 represents a large primary body (such as Earth) and m_2 represents a smaller secondary body (such as the Moon). The system is analyzed in a rotating reference frame that co-rotates with the two primary bodies, maintaining a constant angular velocity Ω given by:

$$\Omega = \sqrt{\frac{\mu}{r_{12}^3}} \tag{1}$$

where $\mu = G(m_1 + m_2)$ is the gravitational parameter of the system and r_{12} is the constant distance between the two primary bodies. This angular velocity can also be expressed in terms of the orbital period T as:

$$\Omega = \frac{2\pi}{T} \tag{2}$$

To non-dimensionalize the problem, we introduce mass ratios for the two primary bodies:

$$\pi_1 = \frac{m_1}{m_1 + m_2}, \quad \pi_2 = \frac{m_2}{m_1 + m_2} \tag{3}$$

where $\pi_1 + \pi_2 = 1$.

3.1.2. Equations of Motion

In the co-rotating coordinate system (x, y, z), the position of the spacecraft m relative to the primary bodies is described by:

$$\boldsymbol{r}_1 = (x + \pi_2 r_{12})\hat{\boldsymbol{\imath}} + y\hat{\boldsymbol{\jmath}} + z\hat{\boldsymbol{k}} \tag{4}$$

$$\boldsymbol{r}_2 = (x - \pi_1 r_{12})\hat{\boldsymbol{\imath}} + y\hat{\boldsymbol{\jmath}} + z\hat{\boldsymbol{k}} \tag{5}$$

where $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are the position vectors relative to m_1 and m_2, respectively.

The inertial acceleration of the spacecraft includes contributions from the rotating reference frame:

$$\ddot{\boldsymbol{r}} = \boldsymbol{a}_G + \dot{\boldsymbol{\Omega}} \times \boldsymbol{r} + \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r}) + 2\boldsymbol{\Omega} \times \boldsymbol{v}_{\text{rel}} + \boldsymbol{a}_{\text{rel}} \tag{6}$$

This can be expanded into component form as:

$$\ddot{\boldsymbol{r}} = (\ddot{x} - 2\Omega\dot{y} - \Omega^2 x)\hat{\boldsymbol{\imath}} + (\ddot{y} + 2\Omega\dot{x} - \Omega^2 y)\hat{\boldsymbol{\jmath}} + \ddot{z}\hat{\boldsymbol{k}} \tag{7}$$

The gravitational forces exerted on m by the two primary bodies are:

$$\boldsymbol{F}_1 = -\frac{\mu_1 m}{r_1^3}\boldsymbol{r}_1, \quad \boldsymbol{F}_2 = -\frac{\mu_2 m}{r_2^3}\boldsymbol{r}_2 \tag{8}$$

Combining these expressions yields the complete equations of motion for the CR3BP:

$$\ddot{x} - 2\Omega\dot{y} - \Omega^2 x = -\frac{\mu_1}{r_1^3}(x + \pi_2 r_{12}) - \frac{\mu_2}{r_2^3}(x - \pi_1 r_{12}) \tag{9}$$

$$\ddot{y} + 2\Omega\dot{x} - \Omega^2 y = -\frac{\mu_1}{r_1^3}y - \frac{\mu_2}{r_2^3}y \tag{10}$$

$$\ddot{z} = -\frac{\mu_1}{r_1^3}z - \frac{\mu_2}{r_2^3}z \tag{11}$$

These equations reveal several important features of the problem. The terms $2\Omega\dot{y}$ and $2\Omega\dot{x}$ represent the Coriolis acceleration, while $\Omega^2 x$ and $\Omega^2 y$ represent the centrifugal acceleration in the rotating frame.

3.1.3. Lagrange Points

The CR3BP admits five equilibrium points, known as Lagrange points or libration points, where a spacecraft can remain stationary in the rotating frame [6]. At these points, all velocities and accelerations vanish:

$$\dot{x} = \dot{y} = \dot{z} = 0 \quad \text{and} \quad \ddot{x} = \ddot{y} = \ddot{z} = 0 \tag{12}$$

Substituting these conditions into the equations of motion yields:

$$-\Omega^2 x = -\frac{\mu_1}{r_1^3}(x + \pi_2 r_{12}) - \frac{\mu_2}{r_2^3}(x - \pi_1 r_{12}) \tag{13}$$

$$-\Omega^2 y = -\frac{\mu_1}{r_1^3}y - \frac{\mu_2}{r_2^3}y \tag{14}$$

$$0 = \left(\frac{\mu_1}{r_1^3} + \frac{\mu_2}{r_2^3}\right)z \tag{15}$$

These equations reveal that all Lagrange points must lie in the orbital plane of the two primary bodies ($z = 0$).

The triangular Lagrange points L_4 and L_5 form equilateral triangles with the two primary bodies and are located at:

$$x = \frac{r_{12}}{2} - \pi_2 r_{12}, \quad y = \pm \frac{\sqrt{3}}{2} r_{12} \tag{16}$$

The collinear Lagrange points L_1, L_2, and L_3 lie on the line connecting the two primary bodies ($y = 0$). Their positions can be found by solving the equation:

$$f(\pi_2, \xi) = \frac{(1-\pi_2)(\xi+\pi_2)}{|\xi+\pi_2|^3} + \frac{\pi_2(\xi+\pi_2-1)}{|\xi+\pi_2-1|^3} = 0 \tag{17}$$

where $\xi = \frac{x}{r_{12}}$ is the non-dimensional position along the x-axis.

3.1.4. Jacobi Constant

The CR3BP possesses a constant of motion known as the Jacobi constant [6], which serves as an integral of motion analogous to energy in the rotating reference frame:

$$C = \frac{1}{2}v^2 - \frac{1}{2}\Omega^2(x^2+y^2) - \frac{\mu_1}{r_1} - \frac{\mu_2}{r_2} \tag{18}$$

where $\frac{v^2}{2}$ represents the kinetic energy per unit mass in the rotating frame, $-\frac{\mu_1}{r_1}$ and $-\frac{\mu_2}{r_2}$ are the gravitational potential energies due to the two primary masses, and $-\frac{\Omega^2(x^2+y^2)}{2}$ can be interpreted as the potential energy of the centrifugal force per unit mass.

For motion restricted to the orbital plane, the distances are:

$$r_1 = \sqrt{(x+\pi_2 r_{12})^2 + y^2}, \quad r_2 = \sqrt{(x-\pi_1 r_{12})^2 + y^2} \tag{19}$$

Solving for the velocity magnitude yields:

$$v^2 = \Omega^2(x^2+y^2) + \frac{2\mu_1}{r_1} + \frac{2\mu_2}{r_2} + 2C \tag{20}$$

Since $v^2 \geq 0$, this equation defines forbidden regions in the (x, y) plane where the spacecraft cannot enter: regions where the right-hand side would become negative. These zero-velocity curves partition the configuration space into allowed and forbidden regions, providing insight into the accessible domains of motion for a given value of the Jacobi constant.

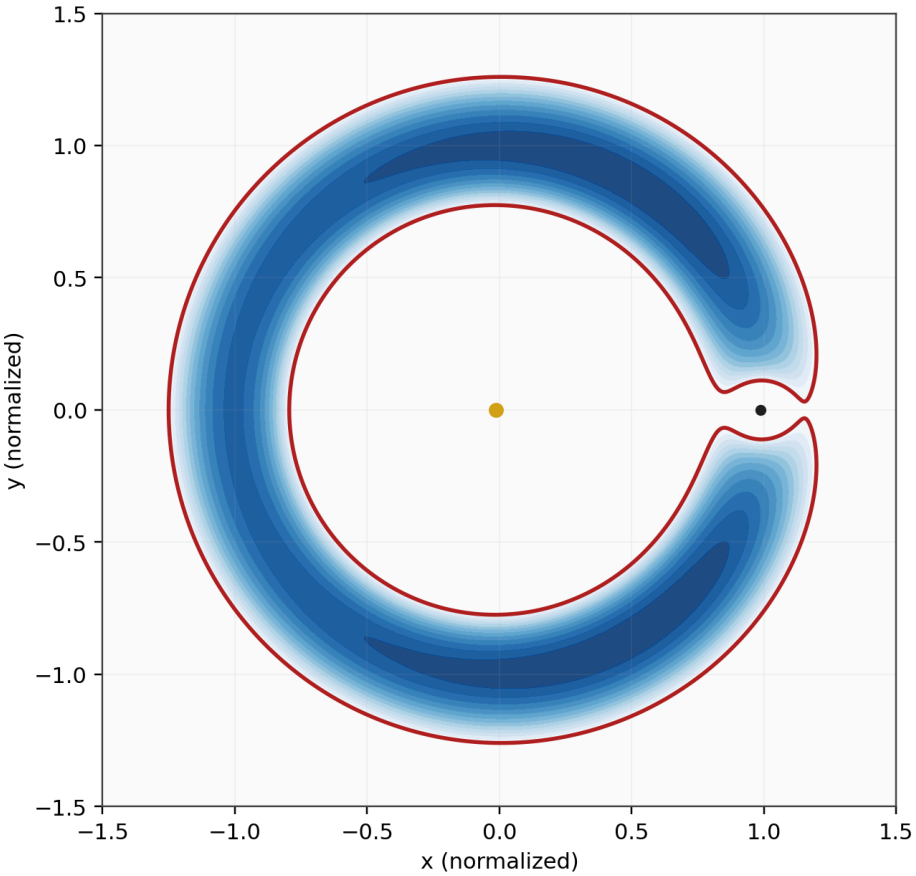

Figure 1: Representative zero-velocity curve in the planar CR3BP. The black contour marks the boundary between dynamically accessible and forbidden regions for a fixed Jacobi constant, the gray shaded region indicates where $v^2 < 0$ would be required, and the two dark markers denote the primary bodies (with the larger marker corresponding to the larger primary).

Figure 1 provides a representative example of this partitioning. In practice, zero-velocity curves are useful for interpreting whether motion can pass between libration regions or remains confined near a particular primary for a given Jacobi constant.

3.2. Perturbations

In practical astrodynamics, spacecraft motion deviates from ideal two-body or three-body trajectories due to various perturbing forces. These perturbations must be accounted for to achieve accurate orbit determination and prediction. The general form of the perturbed equations of motion is:

$$\ddot{\boldsymbol{r}} = -\frac{\mu \boldsymbol{r}}{r^3} + \boldsymbol{p} \tag{21}$$

where $\boldsymbol{p}$ represents the net perturbative acceleration from all sources other than the primary gravitational attraction.

The following sections describe the primary perturbations implemented in the `ivp` library. While some perturbations (such as solar radiation pressure and third-body effects) are critical for the high-precision analysis of the VLISA constellation in the Sun-Earth system, others are included in the library for general astrodynamics applications.

3.2.1. Solar Radiation Pressure

Solar radiation pressure (SRP) arises from momentum transfer when photons from the Sun strike and are absorbed or reflected by a spacecraft's surface. This non-gravitational perturbation becomes increasingly important for spacecraft with large area-to-mass ratios, such as those equipped with solar panels or solar sails.

The radiation pressure at a given distance from the Sun can be expressed as:

$$P = P_0 \left(\frac{r_{\text{sun},0}}{r_{\text{sun}}} \right)^2 \tag{22}$$

where $P_0 = 4.56 \times 10^{-6} \frac{\text{N}}{\text{m}^2}$ is the nominal solar radiation pressure at 1 AU from the Sun ($r_{\text{sun},0} = 1.496 \times 10^8$ km), and r_{sun} is the spacecraft's distance from the Sun. The inverse-square law accounts for the decrease in photon flux with increasing distance.

The force exerted on the spacecraft depends on the exposed area and the spacecraft's optical properties. For a flat surface oriented perpendicular to the incident radiation, the force is:

$$\boldsymbol{F}_{\text{SRP}} = P C_R A_s \hat{\boldsymbol{r}}_{\text{sun}} \tag{23}$$

where:

- C_R is the radiation pressure coefficient, ranging from 1 for perfect absorption to 2 for perfect reflection (typical values are 1.2 to 1.5 for spacecraft)
- A_s is the effective cross-sectional area exposed to sunlight
- $\hat{\boldsymbol{r}}_{\text{sun}}$ is the unit vector pointing from the Sun to the spacecraft

Since $\hat{\boldsymbol{r}}_{\text{sun}}$ points radially outward from the Sun, the positive sign correctly indicates that solar radiation pressure pushes the spacecraft away from the Sun.

To obtain the perturbative acceleration, we apply Newton's second law $\boldsymbol{F} = m\boldsymbol{a}$ and divide both sides by the spacecraft mass m:

$$\boldsymbol{p}_{\text{SRP}} = \frac{\boldsymbol{F}_{\text{SRP}}}{m} = \frac{P C_R A_s \hat{\boldsymbol{r}}_{\text{sun}}}{m} = P_0 C_R \left(\frac{A_s}{m} \right) \left(\frac{r_{\text{sun},0}}{r_{\text{sun}}} \right)^2 \hat{\boldsymbol{r}}_{\text{sun}} \tag{24}$$

The area-to-mass ratio $\left(\frac{A_s}{m}\right)$ naturally emerges from this division and is a key parameter determining the magnitude of SRP effects. Spacecraft with larger solar arrays or lower mass experience greater perturbations.

For Earth-orbiting satellites, the solar radiation pressure varies with orbital position due to the spacecraft entering and exiting Earth's shadow (umbra and penumbra). A shadow function ν is often introduced:

$$\nu = \begin{cases} 0 & \text{spacecraft in umbra (full shadow)} \\ (0,1) & \text{spacecraft in penumbra (partial shadow)} \\ 1 & \text{spacecraft in sunlight} \end{cases} \tag{25}$$

The complete SRP acceleration including eclipse effects becomes:

$$\boldsymbol{p}_{\text{SRP}} = \nu P_0 \left(\frac{A_s}{m}\right) C_R \left(\frac{r_{\text{sun},0}}{r_{\text{sun}}}\right)^2 \hat{\boldsymbol{r}}_{\text{sun}} \tag{26}$$

The effect of SRP is most pronounced for:

- High-altitude orbits where the spacecraft spends more time in sunlight
- Spacecraft with large solar arrays or reflective surfaces
- Long-duration missions where small accelerations accumulate over time

3.2.2. Third-Body Gravitational Perturbations

For trajectories and missions operating at large distances from the primary body, gravitational perturbations from third bodies become significant. These perturbations arise when additional massive bodies, such as the Sun, Moon, or planets, exert gravitational forces on the spacecraft that are not accounted for in the primary two-body or restricted three-body formulation.

Consider a spacecraft orbiting a primary body (e.g., Earth) while being perturbed by a third body (e.g., the Sun or Moon). For a single spacecraft, the perturbative acceleration due to the third body can be expressed as:

$$\boldsymbol{p}_{\text{3rd}} = G m_3 \left(\frac{\boldsymbol{r}_{s/3}}{r_{s/3}^3} - \frac{\boldsymbol{r}_{p/3}}{r_{p/3}^3} \right) \tag{27}$$

where:

- m_3 is the mass of the perturbing third body
- $\boldsymbol{r}_{s/3}$ is the position vector from the spacecraft to the third body
- $\boldsymbol{r}_{p/3}$ is the position vector from the primary body to the third body
- $r_{s/3} = |\boldsymbol{r}_{s/3}|$ and $r_{p/3} = |\boldsymbol{r}_{p/3}|$

This formulation accounts for both the direct gravitational attraction of the third body on the spacecraft and the indirect effect due to the acceleration of the primary body itself. In the VLISA analysis, each of the three spacecraft is propagated independently under the same Sun-Earth force model, and the arm-length history is recovered afterward from the absolute trajectories.

3.3. Coordinate Systems and Transformations

The choice of reference frame and coordinate system significantly impacts the numerical properties and physical interpretation of astrodynamics simulations. Different frames have different properties and are suited to different applications, so coordinate transformations between frames are essential for comprehensive analysis.

3.3.1. Inertial Reference Frame

The inertial frame (also called the heliocentric frame for solar system dynamics) has a fixed orientation relative to distant stars. In this frame:

Key properties:

- Newton's laws apply in their simplest form without fictitious forces
- Long-term orbit propagation is numerically stable
- Natural for Keplerian two-body problems and interplanetary trajectories
- Direct integration of inertial positions and velocities

Limitations:

- Cannot represent equilibrium points (which are stationary in rotating frames)
- Relative motion of formation-flying spacecraft appears complex
- Not suitable for analysis of Lagrange point dynamics or periodic orbits

In the inertial frame, the equations of motion for a spacecraft are simply:

$$\ddot{\boldsymbol{r}} = -\sum_i \frac{GM_i}{|\boldsymbol{r} - \boldsymbol{r}_i|^3}(\boldsymbol{r} - \boldsymbol{r}_i) + \boldsymbol{p} \tag{28}$$

where $\boldsymbol{r}$ is the spacecraft position in the inertial frame, $\boldsymbol{r}_i$ is the inertial position of gravitating body i, and $\boldsymbol{p}$ collects non-point-mass perturbations such as atmospheric drag or solar radiation pressure.

3.3.2. Co-Rotating Reference Frame

The co-rotating frame (also called the synodic frame) rotates with the two primary bodies in the CR3BP, maintaining fixed positions for L_1, L_2, L_3, L_4, and L_5.

Key properties:

- Lagrange points are stationary equilibrium points

- Periodic orbits (halo orbits, Lyapunov orbits) are time-independent
- Constellation geometry appears naturally
- Reveals invariant manifolds and phase space structure
- Essential for analyzing CR3BP dynamics and transfer design

Limitations:

- Includes Coriolis and centrifugal terms that complicate equations
- Energy conservation is not manifest (Jacobi constant replaces energy)
- Inertial position and velocity require transformation back to inertial frame
- Numerical integration can be sensitive near Lagrange points

The equations of motion in the co-rotating frame are derived in Section 3.1. Relative to the inertial frame, they introduce Coriolis and centrifugal terms while keeping the primary bodies and equilibrium-point geometry fixed in the rotating coordinates:

$$\ddot{x} - 2\Omega\dot{y} - \Omega^2 x = -\frac{\mu_1}{r_1^3}(x + \pi_2 r_{12}) - \frac{\mu_2}{r_2^3}(x - \pi_1 r_{12}) \tag{29}$$

$$\ddot{y} + 2\Omega\dot{x} - \Omega^2 y = -\frac{\mu_1}{r_1^3}y - \frac{\mu_2}{r_2^3}y \tag{30}$$

$$\ddot{z} = -\frac{\mu_1}{r_1^3}z - \frac{\mu_2}{r_2^3}z \tag{31}$$

The transformation from inertial frame (X, Y, Z) to rotating frame (x, y, z) for the Sun-Earth system with orbital angular rate Ω is:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \boldsymbol{R}(t) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{32}$$

where the rotation matrix is:

$$\boldsymbol{R}(t) = \begin{bmatrix} \cos(\Omega t) & \sin(\Omega t) & 0 \\ -\sin(\Omega t) & \cos(\Omega t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{33}$$

3.3.3. Coordinate System Selection Guidelines

The appropriate coordinate system depends on the problem characteristics and analysis goals:

Long-term orbit propagation (months to years): Use inertial frame with high-order integrators like DOP853. The inertial frame provides numerical stability and clear energy interpretation.

Lagrange point dynamics: Use co-rotating frame to analyze stability, periodic orbits, and invariant manifolds. The stationary equilibrium points and periodic orbit structure simplify analysis.

Formation flying (small separations): Use relative linearized coordinates when the separation is small compared to the chief orbit radius and the reference orbit is near-circular. For large formations such as VLISA, the full nonlinear dynamics must be retained.

Mission operations: Use Earth-centered inertial (ECI) frame for tracking and navigation, with transformations to Earth-centered Earth-fixed (ECEF) for ground contact planning.

Interplanetary trajectories: Use heliocentric inertial frame for trajectory design, with transformations to planet-centric frames for capture/burn planning.

3.4. Measurement and Noise Models

For space-based interferometry missions like VLISA, the scientific performance is limited by various noise sources that corrupt the phase measurements between spacecraft. These noise sources are typically characterized by their power spectral density (PSD) and are categorized into two primary groups: displacement noise and acceleration noise [22].

3.4.1. Displacement Noise (Shot Noise)

Displacement noise, often dominated by photon shot noise, represents the uncertainty in measuring the distance between spacecraft using laser interferometry. The PSD of the shot noise $S_{s(f)}$ is typically modeled as frequency-independent (white noise) in the measurement band:

$$S_{s(f)} = S_0 \tag{34}$$

where S_0 is a constant determined by the laser power, telescope aperture, and arm length. For VLISA, the 100× larger arm lengths compared to LISA significantly reduce the received photon flux, increasing the shot noise contribution unless compensated by larger telescopes or higher laser power.

3.4.2. Acceleration Noise

Acceleration noise (or "test mass noise") arises from residual non-gravitational forces acting on the spacecraft's internal reference masses, such as stray electrostatic fields, thermal gradients, and outgassing. This noise dominates the low-frequency regime of

the gravitational wave detection band. The PSD of acceleration noise $S_{a(f)}$ is often modeled as:

$$S_{a(f)} = A_0 \left[1 + \left(\frac{f_0}{f} \right)^2 \right] \tag{35}$$

where A_0 is the noise floor and f_0 is the "corner frequency" below which the noise increases.

3.4.3. Total Sensitivity

The total sensitivity of the interferometer is determined by the combination of these noise sources, scaled by the "transfer function" of the constellation geometry. The effective strain sensitivity $h_{n(f)}$ is given by:

$$h_{n(f)} = \frac{\sqrt{S_{s(f)} + \frac{S_{a(f)}}{(2\pi f)^4}}}{L} \tag{36}$$

where L is the arm length. The $\frac{1}{f^4}$ scaling of the acceleration noise in the strain PSD highlights the extreme difficulty of low-frequency gravitational wave detection, which is the primary motivation for the VLISA mission's large arm lengths.

4 Numerical Methods for Astrodynamics

Astrodynamics problems place unusual demands on numerical integration. Spacecraft trajectories must often be propagated over long durations, across multiple dynamical regimes, and under tight accuracy requirements that make both accumulated truncation error and computational cost important design constraints. The methods considered in this chapter are therefore not presented as abstract ODE algorithms alone, but as tools for specific astrodynamics tasks such as long-term orbit propagation, multi-body trajectory analysis, stiff perturbation models, and mission-design studies requiring dense output and event handling.

Runge-Kutta methods are conveniently described using a *Butcher tableau*, which organizes the coefficients that define the stage locations, stage couplings, and solution weights. For an s-stage method, this notation compactly specifies how the solution advances from $\boldsymbol{y}_n$ at time t_n to $\boldsymbol{y}_{n+1}$ at time $t_n + h$.

4.1 Explicit Runge-Kutta Methods

An explicit Runge-Kutta method computes the next solution value through a sequence of stages $\boldsymbol{k}_i = \boldsymbol{f}\Big(x_n + c_i h, \boldsymbol{y}_n + h \sum_{j<i} a_{ij} \boldsymbol{k}_j\Big)$, where each stage depends only on previous stages (the Butcher matrix $\boldsymbol{A}$ is strictly lower triangular). The solution advances as $\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \sum_{i=1}^{s} b_i \boldsymbol{k}_i$.

Explicit Runge-Kutta methods are the workhorses of astrodynamics computations for non-stiff ordinary differential equations. These methods compute each stage sequentially using only previously computed values, making them straightforward to implement and efficient for problems where Jacobian eigenvalues do not impose severe stability restrictions. The Dormand-Prince family represents the state of the art in explicit adaptive methods, combining high accuracy with computational efficiency through embedded error estimation and continuous dense output. The general coefficient structure is shown in Table 3: the vector $\boldsymbol{c}$ contains the time node points, the matrix $\boldsymbol{A}$ contains the stage coupling coefficients, and the vector $\boldsymbol{b}$ contains the quadrature weights.

c_1				
c_2	a_{21}			
c_3	a_{31}	a_{32}		
c_4	a_{41}	a_{42}	a_{43}	
	b_1	b_2	b_3	b_4

Table 3: General structure of a Butcher tableau for an explicit Runge-Kutta method. The node points $\boldsymbol{c} = (c_1, ..., c_s)^T$ specify the fractional step at which each stage is evaluated. The Butcher matrix $\boldsymbol{A} = (a_{ij})$ is strictly lower triangular, with all diagonal and upper entries zero, indicating that each stage depends only on previous stages. The weights $\boldsymbol{b} = (b_1, ..., b_s)^T$ combine the stage derivatives to form the solution update.

Figure 2 gives a example of how the stage evaluations are arranged within a single explicit Runge-Kutta step.

4.1.1 Dormand-Prince Family Characteristics

The Dormand-Prince family of explicit Runge-Kutta methods [13] represents a significant advancement in numerical integration for non-stiff ordinary differential equations commonly encountered in astrodynamics. These methods achieve high accuracy while

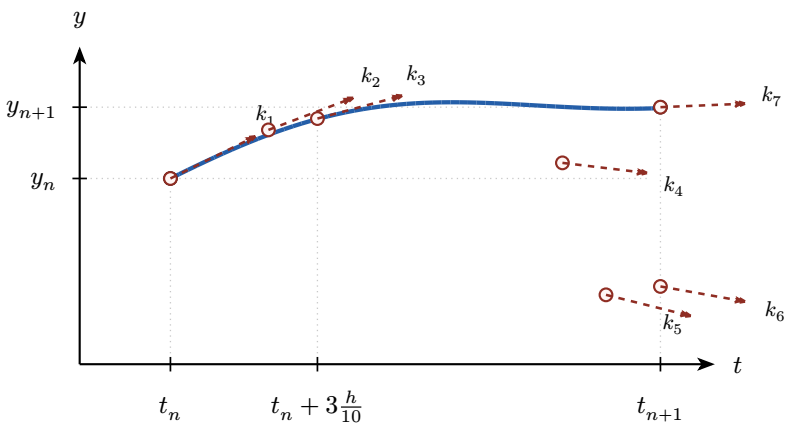

Figure 2: Stage evaluation structure for the DOPRI5 method, shown as a concrete example of an explicit Runge-Kutta scheme. The stage slopes k_1 through k_7 are evaluated at the Dormand-Prince node points using the DOPRI5 coefficients, so each dashed tangent is drawn from an actual intermediate state used by the method. The resulting step lands close to the reference curve at y_{n+1}, illustrating how the fifth-order formula accurately advances the solution across the interval. The final stage also satisfies $k_7 = k_1$ for the next accepted step, which is the First Same As Last (FSAL) property exploited by DOPRI5 to reduce the number of new derivative evaluations.

maintaining computational efficiency through several key innovations: embedded error estimation for adaptive step size control, the First Same As Last (FSAL) property that reuses function evaluations across steps, and continuous dense output for solution interpolation.

These innovations build upon fundamental Runge-Kutta theory and address practical challenges in astrodynamics applications. The following sections describe the specific Dormand-Prince methods implemented in this thesis, with detailed characteristics and appropriate use cases.

The fundamental design principle underlying Dormand-Prince methods is *embedded error estimation*. At each integration step, the method computes two solutions of different orders from the same set of stage evaluations: a higher-order solution that advances the integration and a lower-order embedded solution used solely for error estimation. The difference between these solutions provides a local truncation error estimate without requiring additional function evaluations beyond those needed for the main formula. This efficiency is crucial for astrodynamics applications where derivative evaluations (involving gravitational acceleration, perturbation forces, and coordinate transformations) can be computationally expensive.

The FSAL property further enhances efficiency: the final stage evaluation of the current step serves as the first stage evaluation of the next step, effectively reducing the number of new derivative evaluations required per step. For example, while DOPRI5 appears to be a seven-stage method, only six new evaluations are needed per accepted step. This property requires careful construction of the Butcher tableau coefficients and imposes constraints on the method's design.

Adaptive step size control represents another cornerstone feature. The error estimate drives a feedback controller that adjusts the step size to maintain user-specified error tolerances. The controller employs the scaling relationship $h_{\text{new}} = h\big(\frac{\text{tol}}{\text{err}}\big)^{\frac{1}{p}}$, where p is the method order, modified by safety factors (typically 0.9) and optional Lund stabilization terms that smooth step size variations and prevent oscillatory behavior. Error tolerances are applied component-wise using a mixed absolute-relative criterion: $\text{tol}_i = \text{atol}_i + \text{rtol}_i \cdot \max\big(|y_i|, |y_{\text{new},i}|\big)$. This formulation handles problems with widely varying solution magnitudes, common in orbital mechanics where position components may span astronomical units while velocity components remain relatively small. Figure 3 illustrates this feedback process and the role of the embedded error estimate in accepting, rejecting, and resizing steps.

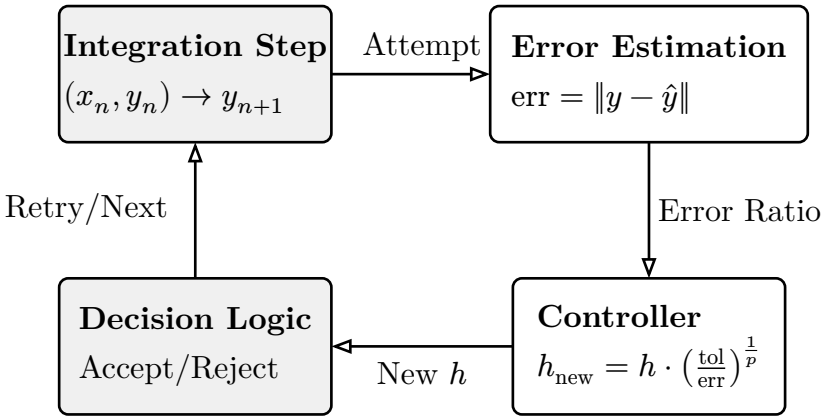

Figure 3: Feedback control loop for adaptive step size selection. The error estimate err from the embedded Runge-Kutta pair feeds into a controller that computes the optimal step size h_{new} for the next step (or for retrying the current step) based on the order p and tolerances. Stability is maintained through safety factors and optional signal smoothing (PI control).

Dense output capabilities distinguish these methods from basic Runge-Kutta schemes. Rather than producing solution values only at the discrete mesh points selected by the adaptive controller, Dormand-Prince methods compute polynomial interpolants that provide continuous solution approximations throughout each integration step. These interpolants maintain consistency with the underlying Runge-Kutta method while achieving order of accuracy one less than the main formula. Dense output proves essential for event location (finding precise trajectory intersections), output generation at user-specified times, and trajectory reconstruction in post-processing.

The methods also incorporate *automatic stiffness detection* to alert users when the integration encounters regions where explicit methods become inefficient. By monitoring the ratio of consecutive derivative evaluations, the algorithm identifies characteristic stiffness symptoms (severely restricted step sizes relative to the integration interval) and issues warnings. While Dormand-Prince methods remain explicit and thus unsuitable for truly stiff problems, this diagnostic capability helps practitioners recognize when switching to implicit methods would be appropriate.

4.1.2 DOPRI5 Method

The DOPRI5 method is a 5th-order Runge-Kutta method with an embedded 4th-order method for error estimation [13]. This workhorse integrator serves moderate-accuracy applications through a seven-stage scheme where the seventh stage satisfies the FSAL property. The primary solution achieves fifth-order accuracy, while the embedded fourth-order formula provides error estimates. The step size controller uses

the classical $h_{\text{new}} = h\left(\frac{\text{tol}}{\text{err}}\right)^{\frac{1}{5}}$ scaling, modified by safety factors and stabilization terms to prevent oscillatory behavior. The complete Butcher tableau coefficients are provided in Appendix A.

Error tolerances are applied component-wise using a mixed absolute-relative criterion: $\text{tol}_i = \text{atol}_i + \text{rtol}_i \cdot \max\left(|y_i|, |y_{\text{new},i}|\right)$, allowing the method to handle problems with widely varying solution magnitudes. The algorithm maintains computational efficiency through strategic array reuse; storage locations are recycled once stage values are no longer needed for either the solution update or dense output coefficient computation.

Computational Complexity: For a system with n state variables, DOPRI5 requires $O(n)$ operations per stage evaluation, with 7 stages total but only 6 new evaluations per accepted step due to the FSAL property. This yields $O(6n)$ per step, with the step size adaptively controlled to maintain specified error tolerances. Memory requirements are $O(n)$ for storing the state vector and stage derivatives, with careful reuse reducing the total storage from 7 to approximately 6 state vectors. The primary computational cost is in the derivative evaluations (acceleration computations), while linear algebra operations are minimal.

The DOPRI5 stepping procedure is summarized in the algorithm listing below.

DOPRI5 Core Integrator

Input: ODE system f, initial state $(x_0, \boldsymbol{y}_0)$, endpoint x_{end}, tolerances $(\textbf{rtol}, \textbf{atol})$

Output: Solution $(x, \boldsymbol{y})$ and integration statistics: n_{fev} (function evaluations), n_{steps} (total steps), n_{accept} (accepted steps), n_{reject} (rejected steps)

Initialization

Initialize algorithm parameters (with defaults if not provided):

$\text{uround} = 2.3 \times 10^{-16}$, $\quad \text{safe} = 0.9$, $\quad \beta = 0.0$, $\quad h_{\text{min factor}} = 0.333$, $h_{\text{max factor}} = 6.0$

$n_{\max} = 100,000$, $n_{\text{stiff}} = 1000$, $\text{iord} = 5$

$\text{expo1} = \frac{1}{5} - 0.2\beta = 0.2$

$h_{\text{min factor inv}} = \frac{1}{h_{\text{min factor}}} = 3.0$, $h_{\text{max factor inv}} = \frac{1}{h_{\text{max factor}}} = 0.1667$

$\text{posneg} = \text{sign}(x_{\text{end}} - x)$

$\text{err}_{\text{prev}} = 10^{-4}$

DOPRI5 Core Integrator

Evaluate $\boldsymbol{k}_1 = f(x, \boldsymbol{y})$

$n_{\text{fev}} = 1$

if $h = 0$ **then**

  $h = \text{InitialStepSize}(f, x, \boldsymbol{y}, \boldsymbol{k}_1, \textbf{rtol}, \textbf{atol}, \text{iord})$

  $n_{\text{fev}} = n_{\text{fev}} + 1$

end

Main Integration Loop

while $x \neq x_{\text{end}}$ **do**

  if $n_{\text{steps}} > n_{\max}$ **then**

    return error: need larger $n_{\max}$

  end

  if $0.1|h| \leq |x| \cdot \varepsilon_{\text{mach}}$ **then**

    return error: step size too small

  end

  Adjust h to land exactly on x_{end} if close

  $n_{\text{steps}} = n_{\text{steps}} + 1$

  Compute Runge-Kutta Stages

  Stage 2:

  $\boldsymbol{k}_2 = f\big(x + c_2 h, \boldsymbol{y} + h \cdot a_{2,1} \boldsymbol{k}_1\big)$

  Stage 3:

  $\boldsymbol{k}_3 = f\big(x + c_3 h, \boldsymbol{y} + h\big(a_{3,1} \boldsymbol{k}_1 + a_{3,2} \boldsymbol{k}_2\big)\big)$

  Stage 4:

  $\boldsymbol{k}_4 = f\big(x + c_4 h, \boldsymbol{y} + h\big(a_{4,1} \boldsymbol{k}_1 + a_{4,2} \boldsymbol{k}_2 + a_{4,3} \boldsymbol{k}_3\big)\big)$

  Stage 5:

  $\boldsymbol{k}_5 = f\big(x + c_5 h, \boldsymbol{y} + h\big(a_{5,1} \boldsymbol{k}_1 + a_{5,2} \boldsymbol{k}_2 + a_{5,3} \boldsymbol{k}_3 + a_{5,4} \boldsymbol{k}_4\big)\big)$

  Stage 6:

  $\boldsymbol{k}_6 = f\big(x + c_6 h, \boldsymbol{y} + h\big(a_{6,1} \boldsymbol{k}_1 + a_{6,2} \boldsymbol{k}_2 + a_{6,3} \boldsymbol{k}_3 + a_{6,4} \boldsymbol{k}_4 + a_{6,5} \boldsymbol{k}_5\big)\big)$

  Stage 7 (reuse $\boldsymbol{k_2}$ storage):

  $\boldsymbol{k}_2 = f(x + h, \boldsymbol{y} + h(b_1 \boldsymbol{k}_1 + b_3 \boldsymbol{k}_3 + b_4 \boldsymbol{k}_4 + b_5 \boldsymbol{k}_5 + b_6 \boldsymbol{k}_6))$

  Increment function evaluation counter

  $n_{\text{fev}} = n_{\text{fev}} + 6$

Compute Solution and Error Estimation

$\boldsymbol{y}_{\text{new}} = \boldsymbol{y} + h(b_1\boldsymbol{k}_1 + b_3\boldsymbol{k}_3 + b_4\boldsymbol{k}_4 + b_5\boldsymbol{k}_5 + b_6\boldsymbol{k}_6)$

Compute scaling factor for error control

$\boldsymbol{s} = \mathbf{atol} + \mathbf{rtol} \odot \max(|\boldsymbol{y}|, |\boldsymbol{y}_{\text{new}}|)$

Compute error estimate (4th-order embedded formula)

$\boldsymbol{e} = \boldsymbol{k}_1\hat{b}_1 + \boldsymbol{k}_3\hat{b}_3 + \boldsymbol{k}_4\hat{b}_4 + \boldsymbol{k}_5\hat{b}_5 + \boldsymbol{k}_6\hat{b}_6 + \boldsymbol{k}_2\hat{b}_7$

$\text{err} = \sqrt{\sum_{i=1}^{n} \left(\frac{\boldsymbol{e}_i}{\boldsymbol{s}_i}\right)^2}$

Step Size Control

Compute unsmoothed error ratio

$\text{err}_{\text{ratio}} = \text{err}^{\text{expo1}}$

Apply Lund stabilization (smoothing)

$\text{err}_{\text{ratio smooth}} = \frac{\text{err}_{\text{ratio}}}{\text{err}_{\text{prev}}^{\beta}}$

Compute new step size with safety factor and bounds

$h_{\text{new}} = \frac{h}{\max\left(h_{\text{max factor inv}}, \min\left(h_{\text{min factor inv}}, \frac{\text{err}_{\text{ratio smooth}}}{\text{safe}}\right)\right)}$

Step Acceptance or Rejection

if $\text{err} \leq 1$ **then**

Update stabilization factor for next step

$\text{err}_{\text{prev}} = \max(\text{err}, 10^{-4})$

$n_{\text{accept}} = n_{\text{accept}} + 1$

Stiffness Detection

if $n_{\text{accept}} \bmod n_{\text{stiff}} = 0$ **or** $n_{\text{stiff detect}} > 0$ **then**

Compute stiffness indicator

$\text{stnum} = \sum_{i=1}^{n} \left(\boldsymbol{k}_{6,i} - \boldsymbol{k}_{2,i}\right)^2$

$\text{stden} = \sum_{i=1}^{n} \left(\boldsymbol{y}_{\text{new},i} - \boldsymbol{y}_{\text{old},i}\right)^2$

if $\text{stden} > 0$ **then**

$\lambda = |h| \sqrt{\frac{\text{stnum}}{\text{stden}}}$

end

Check stiffness threshold

if $\lambda > 6.1$ **then**

$n_{\text{stiff detect}} = n_{\text{stiff detect}} + 1$

$n_{\text{non-stiff}} = 0$

if $n_{\text{stiff detect}} = 15$ **then**

return warning: probably stiff

end

else

$n_{\text{non-stiff}} = n_{\text{non-stiff}} + 1$

if $n_{\text{non-stiff}} = 6$ **then**

$n_{\text{stiff detect}} = 0$

end

end

end

Evaluate FSAL Stage (First Stage of Next Step)

$\boldsymbol{k}_1 = f(x + h, \boldsymbol{y}_{\text{new}})$

$n_{\text{fev}} = n_{\text{fev}} + 1$

Compute Dense Output Coefficients

$\boldsymbol{d}_1 = \boldsymbol{y}$

$\boldsymbol{y}_{\text{diff}} = \boldsymbol{y}_{\text{new}} - \boldsymbol{y}$

$\boldsymbol{d}_2 = \boldsymbol{y}_{\text{diff}}$

$\boldsymbol{d}_3 = h\boldsymbol{k}_1 - \boldsymbol{y}_{\text{diff}}$

$\boldsymbol{d}_4 = \boldsymbol{y}_{\text{diff}} - h\boldsymbol{k}_1 - \boldsymbol{d}_3$

$\boldsymbol{d}_5 = h\big(\hat{b}_1\boldsymbol{k}_1 + \hat{b}_3\boldsymbol{k}_3 + \hat{b}_4\boldsymbol{k}_4 + \hat{b}_5\boldsymbol{k}_5 + \hat{b}_6\boldsymbol{k}_6 + \hat{b}_7\boldsymbol{k}_2\big)$

Update State for Next Step

$\boldsymbol{y} = \boldsymbol{y}_{\text{new}}$

$x_{\text{old}} = x$

$x = x + h$

Call user output function with dense output coefficients

Call user output function with $(x_{\text{old}}, x, \boldsymbol{y}, \boldsymbol{d}, h)$

Check Termination

if $x = x_{\text{end}}$ **then**

return success

end

DOPRI5 Core Integrator

Update Step Size for Next Step

Enforce maximum step size constraint

if $|h_{\text{new}}| > h_{\max}$ **then**

  $h_{\text{new}} = \text{sign}(h_{\text{new}}) \cdot h_{\max}$

end

Prevent oscillations after rejection

if previous step rejected **then**

  $h_{\text{new}} = \text{sign}(h_{\text{new}}) \cdot \min(|h_{\text{new}}|, |h|)$

end

else

Step rejected: reduce step size

$h_{\text{new}} = \frac{h}{\min(h_{\text{min factor inv}}, \frac{\text{err}_{\text{ratio}}}{\text{safe}})}$

$n_{\text{reject}} = n_{\text{reject}} + 1$

Mark step as rejected

end

Update Step Size for Next Iteration

$h = h_{\text{new}}$

end

Algorithm 1: DOPRI5 Core Solver

For dense output (continuous solution interpolation), DOPRI5 uses a 4th-order Hermite polynomial [35]. The interpolant uses the five non-FSAL stage values plus the endpoints to construct a quartic polynomial in the normalized step fraction $\theta = \frac{\xi - x_{\text{old}}}{h}$. The representation employs a nested evaluation scheme for numerical stability, with the polynomial expressed in terms of θ and $\theta' = 1 - \theta$ to maintain symmetry. This fourth-order accurate interpolant proves sufficient for most post-processing tasks, including event location and output generation at user-specified time points.

The continuation listing below shows the assembly of the dense-output coefficients after a successful DOPRI5 step.

DOPRI5 Dense Output

1 **Input:** Interpolation coefficients $\boldsymbol{d}_1, ..., \boldsymbol{d}_5$, interval $[x_{\text{old}}, x_{\text{old}} + h]$, evaluation point ξ

DOPRI5 Dense Output

2 **Output:** Interpolated solution $\boldsymbol{y}(\xi)$

Compute Interpolation Parameter

3 $\theta = \frac{\xi - x_{\text{old}}}{h}$

4 $\theta' = 1 - \theta$

Evaluate Hermite Polynomial

5 $\boldsymbol{y} = \boldsymbol{d}_1 + \theta(\boldsymbol{d}_2 + \theta'(\boldsymbol{d}_3 + \theta(\boldsymbol{d}_4 + \theta'\boldsymbol{d}_5)))$

6 **return** $\boldsymbol{y}$

Algorithm 2: DOPRI5 Dense Output Evaluation

4.1.3 DOP853 Method

The DOP853 method is an 8th-order Runge-Kutta method with embedded 5th and 3rd-order methods for error estimation [8]. This higher-order method provides better accuracy for stringent tolerance requirements and long-term propagation where accumulated error must be tightly controlled. The method requires twelve stages for the main integration step, with three additional stages computed specifically for dense output coefficient generation, bringing the total to fifteen stages plus the FSAL evaluation. The complete Butcher tableau coefficients are provided in Appendix B.

The dual error estimation strategy distinguishes DOP853 from simpler schemes. The primary error estimate uses a fifth-order embedded formula, providing reliable error control for the eighth-order solution. A secondary third-order estimate monitors integration stability, with both estimates combined as $\text{err} = \sqrt{\frac{\text{err}_5}{n \cdot (\text{err}_5 + 0.01\text{err}_3)}}$ to prevent step size degradation in pathological cases. The step size controller applies the scaling $h_{\text{new}} = h\left(\frac{\text{tol}}{\text{err}}\right)^{\frac{1}{8}}$ with safety factor 0.9 and Lund stabilization parameter β to smooth step size variations.

Memory management becomes critical for DOP853 due to its fifteen-stage structure. The implementation strategically reuses array storage, overlaying stage values once they are no longer needed. For instance, stages 11 and 12 overwrite the storage initially allocated for stages 2 and 3, while the dense output stages 13-15 further reuse these locations. This careful orchestration reduces memory requirements from fifteen state-vector arrays to approximately ten.

Computational Complexity: For a system with n state variables, DOP853 requires $O(n)$ operations per stage evaluation, with 15 stages total but efficient reuse reduces effective cost. Memory requirements are approximately $O(10n)$ due to the strategic

overlaying of stage storage locations. The higher order allows significantly larger step sizes than lower-order methods, often resulting in fewer total steps despite the higher per-step cost. For high-accuracy applications with stringent tolerances ($\text{rtol} < 10^{-9}$), DOP853 typically outperforms lower-order methods in total computational time.

The algorithm listing below summarizes the main DOP853 step update and embedded-error evaluation.

DOP853 Core Integrator

Input: ODE system f, initial state $(x_0, \boldsymbol{y}_0)$, endpoint x_{end}, tolerances $(\textbf{rtol}, \textbf{atol})$

Output: Solution $(x, \boldsymbol{y})$ and integration statistics: n_{fev} (function evaluations), n_{steps} (total steps), n_{accept} (accepted steps), n_{reject} (rejected steps)

Initialization

Initialize algorithm parameters (with defaults if not provided):

$\text{uround} = 2.3 \times 10^{-16}$, $\text{safe} = 0.9$, $\beta = 0.0$, $h_{\text{min factor}} = 0.333$, $h_{\text{max factor}} = 6.0$

$n_{\max} = 100,000$, $n_{\text{stiff}} = 1000$, $\text{iord} = 8$

$\text{expo1} = \frac{1}{8} - 0.2\beta = 0.125$

$h_{\text{min factor inv}} = \frac{1}{h_{\text{min factor}}} = 3.0$, $h_{\text{max factor inv}} = \frac{1}{h_{\text{max factor}}} = 0.1667$

$\text{posneg} = \text{sign}(x_{\text{end}} - x)$

$\text{err}_{\text{prev}} = 10^{-4}$

Evaluate $\boldsymbol{k}_1 = f(x, \boldsymbol{y})$

$n_{\text{fev}} = 1$

if $h = 0$ **then**

  $h = \text{InitialStepSize}(f, x, \boldsymbol{y}, \boldsymbol{k}_1, \textbf{rtol}, \textbf{atol}, \text{iord})$

  $n_{\text{fev}} = n_{\text{fev}} + 1$

end

Main Integration Loop

while $x \neq x_{\text{end}}$ **do**

  if $n_{\text{steps}} > n_{\max}$ **then**

    return error: need larger $n_{\max}$

  end

  if $0.1|h| \leq |x| \cdot \varepsilon_{\text{mach}}$ **then**

21 | **return** error: step size too small

22 **end**

23 Adjust h to land exactly on x_{end} if close

24 $n_{\text{steps}} = n_{\text{steps}} + 1$

Compute Runge-Kutta Stages

Storage allocation: $k \in \mathbb{R}^{10\times n}$ *to store stages with strategic reuse*

25 **Stage 2:**

26 $\boldsymbol{k}_2 = f\big(x + c_2 h, \boldsymbol{y} + h \cdot a_{2,1}\boldsymbol{k}_1\big)$

27 **Stage 3:**

28 $\boldsymbol{k}_3 = f\big(x + c_3 h, \boldsymbol{y} + h\big(a_{3,1}\boldsymbol{k}_1 + a_{3,2}\boldsymbol{k}_2\big)\big)$

29 **Stage 4:**

30 $\boldsymbol{k}_4 = f\big(x + c_4 h, \boldsymbol{y} + h\big(a_{4,1}\boldsymbol{k}_1 + a_{4,3}\boldsymbol{k}_3\big)\big)$

31 **Stage 5:**

32 $\boldsymbol{k}_5 = f\big(x + c_5 h, \boldsymbol{y} + h\big(a_{5,1}\boldsymbol{k}_1 + a_{5,3}\boldsymbol{k}_3 + a_{5,4}\boldsymbol{k}_4\big)\big)$

33 **Stage 6:**

34 $\boldsymbol{k}_6 = f\big(x + c_6 h, \boldsymbol{y} + h\big(a_{6,1}\boldsymbol{k}_1 + a_{6,4}\boldsymbol{k}_4 + a_{6,5}\boldsymbol{k}_5\big)\big)$

35 **Stage 7:**

36 $\boldsymbol{k}_7 = f\big(x + c_7 h, \boldsymbol{y} + h\big(a_{7,1}\boldsymbol{k}_1 + a_{7,4}\boldsymbol{k}_4 + a_{7,5}\boldsymbol{k}_5 + a_{7,6}\boldsymbol{k}_6\big)\big)$

37 **Stage 8:**

38 $\boldsymbol{k}_8 = f\big(x + c_8 h, \boldsymbol{y} + h\big(a_{8,1}\boldsymbol{k}_1 + a_{8,4}\boldsymbol{k}_4 + a_{8,5}\boldsymbol{k}_5 + a_{8,6}\boldsymbol{k}_6 + a_{8,7}\boldsymbol{k}_7\big)\big)$

39 **Stage 9:**

40 $\boldsymbol{k}_9 = f\big(x + c_9 h, \boldsymbol{y} + h\big(a_{9,1}\boldsymbol{k}_1 + a_{9,4}\boldsymbol{k}_4 + a_{9,5}\boldsymbol{k}_5 + a_{9,6}\boldsymbol{k}_6 + a_{9,7}\boldsymbol{k}_7 + a_{9,8}\boldsymbol{k}_8\big)\big)$

41 **Stage 10:**

42 $\boldsymbol{k}_{10} = f\big(x + c_{10} h, \boldsymbol{y} + h\big(a_{10,1}\boldsymbol{k}_1 + a_{10,4}\boldsymbol{k}_4 + a_{10,5}\boldsymbol{k}_5 + a_{10,6}\boldsymbol{k}_6 + a_{10,7}\boldsymbol{k}_7 + a_{10,8}\boldsymbol{k}_8 + a_{10,9}\boldsymbol{k}_9\big)\big)$

43 **Stage 11 (reuse $\boldsymbol{k_2}$ storage):**

44 $\boldsymbol{k}_2 = f\big(x + c_{11} h, \boldsymbol{y} + h\big(a_{11,1}\boldsymbol{k}_1 + a_{11,4}\boldsymbol{k}_4 + a_{11,5}\boldsymbol{k}_5 + a_{11,6}\boldsymbol{k}_6 + a_{11,7}\boldsymbol{k}_7 + a_{11,8}\boldsymbol{k}_8 + a_{11,9}\boldsymbol{k}_9 + a_{11,10}\boldsymbol{k}_{10}\big)\big)$

45 **Stage 12 (reuse $\boldsymbol{k_3}$ storage):**

$\boldsymbol{k}_3 = f\big(x + h, \boldsymbol{y} + h\big(a_{12,1}\boldsymbol{k}_1 + a_{12,4}\boldsymbol{k}_4 + a_{12,5}\boldsymbol{k}_5 + a_{12,6}\boldsymbol{k}_6 + a_{12,7}\boldsymbol{k}_7 + a_{12,8}\boldsymbol{k}_8 + a_{12,9}\boldsymbol{k}_9 + a_{12,10}\boldsymbol{k}_{10} + a_{12,11}\boldsymbol{k}_2\big)\big)$

Increment function evaluation counter

$n_{\text{fev}} = n_{\text{fev}} + 11$

Compute Solution and Error Estimation

Reuse $\boldsymbol{k}_4$ storage for weighted sum

$\boldsymbol{k}_4 = b_1\boldsymbol{k}_1 + b_6\boldsymbol{k}_6 + b_7\boldsymbol{k}_7 + b_8\boldsymbol{k}_8 + b_9\boldsymbol{k}_9 + b_{10}\boldsymbol{k}_{10} + b_{11}\boldsymbol{k}_2 + b_{12}\boldsymbol{k}_3$

Reuse $\boldsymbol{k}_5$ storage for 8th-order solution

$\boldsymbol{k}_5 = \boldsymbol{y} + h\boldsymbol{k}_4$

Compute scaling factor for error control

$\boldsymbol{s} = \mathbf{atol} + \mathbf{rtol} \odot \max(|\boldsymbol{y}|, |\boldsymbol{k}_5|)$

Compute primary error estimate (5th-order embedded formula)

$\boldsymbol{e} = e_1\boldsymbol{k}_1 + e_6\boldsymbol{k}_6 + e_7\boldsymbol{k}_7 + e_8\boldsymbol{k}_8 + e_9\boldsymbol{k}_9 + e_{10}\boldsymbol{k}_{10} + e_{11}\boldsymbol{k}_2 + e_{12}\boldsymbol{k}_3$

$\text{err} = \sum_{i=1}^{n} \left(\frac{\boldsymbol{e}_i}{\boldsymbol{s}_i}\right)^2$

Compute secondary error estimate (3rd-order for stability)

$\boldsymbol{e}_2 = \boldsymbol{k}_4 - \hat{b}_1\boldsymbol{k}_1 - \hat{b}_2\boldsymbol{k}_9 - \hat{b}_3\boldsymbol{k}_3$

$\text{err}_2 = \sum_{i=1}^{n} \left(\frac{\boldsymbol{e}_{2,i}}{\boldsymbol{s}_i}\right)^2$

Combine error estimates

$\text{deno} = \text{err} + 0.01\text{err}_2$

if $\text{deno} \leq 0$ **then**

  $\text{deno} = 1$

end

$\text{err} = |h| \cdot \sqrt{\frac{\text{err}}{n \cdot \text{deno}}}$

Step Size Control

Compute unsmoothed error ratio

$\text{err}_{\text{ratio}} = \text{err}^{\text{expo1}}$

Apply Lund stabilization (smoothing)

$\text{err}_{\text{ratio smooth}} = \frac{\text{err}_{\text{ratio}}}{\text{err}_{\text{prev}}^{\beta}}$

Compute new step size with safety factor and bounds

$h_{\text{new}} = \frac{h}{\max\left(h_{\text{max factor inv}}, \min\left(h_{\text{min factor inv}}, \frac{\text{err}_{\text{ratio smooth}}}{\text{safe}}\right)\right)}$

Step Acceptance or Rejection

if $\mathrm{err} \leq 1$ **then**

Update stabilization factor for next step

$\mathrm{err}_{\mathrm{prev}} = \max(\mathrm{err}, 10^{-4})$

$n_{\mathrm{accept}} = n_{\mathrm{accept}} + 1$

Stiffness Detection

if $n_{\mathrm{accept}} \bmod n_{\mathrm{stiff}} = 0$ **or** $n_{\mathrm{stiff\ detect}} > 0$ **then**

Compute stiffness indicator using reused k arrays

$\mathrm{stnum} = \sum_{i=1}^{n} \left(\boldsymbol{k}_{4,i} - \boldsymbol{k}_{3,i}\right)^2$

$\mathrm{stden} = \sum_{i=1}^{n} \left(\boldsymbol{k}_{5,i} - \boldsymbol{y}_{\mathrm{old},i}\right)^2$

if $\mathrm{stden} > 0$ **then**

$\lambda = |h| \sqrt{\frac{\mathrm{stnum}}{\mathrm{stden}}}$

end

Check stiffness threshold

if $\lambda > 6.1$ **then**

$n_{\mathrm{stiff\ detect}} = n_{\mathrm{stiff\ detect}} + 1$

$n_{\text{non-stiff}} = 0$

if $n_{\mathrm{stiff\ detect}} = 15$ **then**

return warning: probably stiff

end

else

$n_{\text{non-stiff}} = n_{\text{non-stiff}} + 1$

if $n_{\text{non-stiff}} = 6$ **then**

$n_{\mathrm{stiff\ detect}} = 0$

end

end

end

Evaluate FSAL Stage (First Stage of Next Step)

Reuse $\boldsymbol{k}_4$ storage: evaluate at end of current step

$\boldsymbol{k}_4 = f(x + h, \boldsymbol{k}_5)$

$n_{\mathrm{fev}} = n_{\mathrm{fev}} + 1$

Compute Initial Dense Output Coefficients

87 $\boldsymbol{d}_1 = \boldsymbol{y}$

88 $\boldsymbol{y}_{\text{diff}} = \boldsymbol{k}_5 - \boldsymbol{y}$

89 $\boldsymbol{d}_2 = \boldsymbol{y}_{\text{diff}}$

90 $\boldsymbol{d}_3 = h\boldsymbol{k}_1 - \boldsymbol{y}_{\text{diff}}$

91 $\boldsymbol{d}_4 = \boldsymbol{y}_{\text{diff}} - h\boldsymbol{k}_4 - \boldsymbol{d}_3$

Compute $\boldsymbol{d}_5$ through $\boldsymbol{d}_8$ using first 12 stages

92 $\boldsymbol{d}_5 = \mathbf{bi7}_{5,1}\boldsymbol{k}_1 + \mathbf{bi7}_{5,6}\boldsymbol{k}_6 + \mathbf{bi7}_{5,7}\boldsymbol{k}_7 + \mathbf{bi7}_{5,8}\boldsymbol{k}_8 + \mathbf{bi7}_{5,9}\boldsymbol{k}_9 + \mathbf{bi7}_{5,10}\boldsymbol{k}_{10} + \mathbf{bi7}_{5,11}\boldsymbol{k}_2 + \mathbf{bi7}_{5,12}\boldsymbol{k}_3$

93 $\boldsymbol{d}_6 = \mathbf{bi7}_{6,1}\boldsymbol{k}_1 + \mathbf{bi7}_{6,6}\boldsymbol{k}_6 + \mathbf{bi7}_{6,7}\boldsymbol{k}_7 + \mathbf{bi7}_{6,8}\boldsymbol{k}_8 + \mathbf{bi7}_{6,9}\boldsymbol{k}_9 + \mathbf{bi7}_{6,10}\boldsymbol{k}_{10} + \mathbf{bi7}_{6,11}\boldsymbol{k}_2 + \mathbf{bi7}_{6,12}\boldsymbol{k}_3$

94 $\boldsymbol{d}_7 = \mathbf{bi7}_{7,1}\boldsymbol{k}_1 + \mathbf{bi7}_{7,6}\boldsymbol{k}_6 + \mathbf{bi7}_{7,7}\boldsymbol{k}_7 + \mathbf{bi7}_{7,8}\boldsymbol{k}_8 + \mathbf{bi7}_{7,9}\boldsymbol{k}_9 + \mathbf{bi7}_{7,10}\boldsymbol{k}_{10} + \mathbf{bi7}_{7,11}\boldsymbol{k}_2 + \mathbf{bi7}_{7,12}\boldsymbol{k}_3$

95 $\boldsymbol{d}_8 = \mathbf{bi7}_{8,1}\boldsymbol{k}_1 + \mathbf{bi7}_{8,6}\boldsymbol{k}_6 + \mathbf{bi7}_{8,7}\boldsymbol{k}_7 + \mathbf{bi7}_{8,8}\boldsymbol{k}_8 + \mathbf{bi7}_{8,9}\boldsymbol{k}_9 + \mathbf{bi7}_{8,10}\boldsymbol{k}_{10} + \mathbf{bi7}_{8,11}\boldsymbol{k}_2 + \mathbf{bi7}_{8,12}\boldsymbol{k}_3$

Three Additional Stages for Dense Output

Evaluate stage 13 at $x + c_{13}h$ (reuse $\boldsymbol{k}_{10}$ storage):

96 $\boldsymbol{k}_{10} = f\big(x + c_{13}h, \boldsymbol{y} + h\big(a_{13,1}\boldsymbol{k}_1 + a_{13,7}\boldsymbol{k}_7 + a_{13,8}\boldsymbol{k}_8 + a_{13,9}\boldsymbol{k}_9 + a_{13,10}\boldsymbol{k}_{10} + a_{13,11}\boldsymbol{k}_2 + a_{13,12}\boldsymbol{k}_3 + a_{13,13}\boldsymbol{k}_4\big)\big)$

Evaluate stage 14 at $x + c_{14}h$ (reuse $\boldsymbol{k}_2$ storage):

97 $\boldsymbol{k}_2 = f\big(x + c_{14}h, \boldsymbol{y} + h\big(a_{14,1}\boldsymbol{k}_1 + a_{14,6}\boldsymbol{k}_6 + a_{14,7}\boldsymbol{k}_7 + a_{14,8}\boldsymbol{k}_8 + a_{14,11}\boldsymbol{k}_2 + a_{14,12}\boldsymbol{k}_3 + a_{14,13}\boldsymbol{k}_4 + a_{14,14}\boldsymbol{k}_{10}\big)\big)$

Evaluate stage 15 at $x + c_{15}h$ (reuse $\boldsymbol{k}_3$ storage):

98 $\boldsymbol{k}_3 = f\big(x + c_{15}h, \boldsymbol{y} + h\big(a_{15,1}\boldsymbol{k}_1 + a_{15,6}\boldsymbol{k}_6 + a_{15,7}\boldsymbol{k}_7 + a_{15,8}\boldsymbol{k}_8 + a_{15,9}\boldsymbol{k}_9 + a_{15,13}\boldsymbol{k}_4 + a_{15,14}\boldsymbol{k}_{10} + a_{15,15}\boldsymbol{k}_2\big)\big)$

99 $n_{\text{fev}} = n_{\text{fev}} + 3$

Finalize Dense Output Coefficients

Add contributions from FSAL ($\boldsymbol{k}_4$) and stages 13-15 to complete $\boldsymbol{d}_5$ through $\boldsymbol{d}_8$

100 $\boldsymbol{d}_5 = h\big(\boldsymbol{d}_5 + \mathbf{bi7}_{5,13}\boldsymbol{k}_4 + \mathbf{bi7}_{5,14}\boldsymbol{k}_{10} + \mathbf{bi7}_{5,15}\boldsymbol{k}_2 + \mathbf{bi7}_{5,16}\boldsymbol{k}_3\big)$

101 $\boldsymbol{d}_6 = h\big(\boldsymbol{d}_6 + \mathbf{bi7}_{6,13}\boldsymbol{k}_4 + \mathbf{bi7}_{6,14}\boldsymbol{k}_{10} + \mathbf{bi7}_{6,15}\boldsymbol{k}_2 + \mathbf{bi7}_{6,16}\boldsymbol{k}_3\big)$

102 $\boldsymbol{d}_7 = h\big(\boldsymbol{d}_7 + \mathbf{bi7}_{7,13}\boldsymbol{k}_4 + \mathbf{bi7}_{7,14}\boldsymbol{k}_{10} + \mathbf{bi7}_{7,15}\boldsymbol{k}_2 + \mathbf{bi7}_{7,16}\boldsymbol{k}_3\big)$

103 $\boldsymbol{d}_8 = h\big(\boldsymbol{d}_8 + \mathbf{bi7}_{8,13}\boldsymbol{k}_4 + \mathbf{bi7}_{8,14}\boldsymbol{k}_{10} + \mathbf{bi7}_{8,15}\boldsymbol{k}_2 + \mathbf{bi7}_{8,16}\boldsymbol{k}_3\big)$

Update State for Next Step

FSAL property: copy $\boldsymbol{k}_4$ to $\boldsymbol{k}_1$ for first stage of next step

$\boldsymbol{k}_1 = \boldsymbol{k}_4$

Update solution and independent variable

$\boldsymbol{y} = \boldsymbol{k}_5$

$x_{\text{old}} = x$

$x = x + h$

Call user output function with dense output coefficients

Call user output function with $(x_{\text{old}}, x, \boldsymbol{y}, \boldsymbol{d}, h)$

Check Termination

if $x = x_{\text{end}}$ **then**

  return success

end

Update Step Size for Next Step

Enforce maximum step size constraint

if $|h_{\text{new}}| > h_{\max}$ **then**

  $h_{\text{new}} = \text{sign}(h_{\text{new}}) \cdot h_{\max}$

end

Prevent oscillations after rejection

if previous step rejected **then**

  $h_{\text{new}} = \text{sign}(h_{\text{new}}) \cdot \min(|h_{\text{new}}|, |h|)$

end

else

Step rejected: reduce step size

$h_{\text{new}} = \frac{h}{\min(h_{\text{min factor inv}}, \frac{\text{err}_{\text{ratio}}}{\text{safe}})}$

$n_{\text{reject}} = n_{\text{reject}} + 1$

Mark step as rejected

end

Update Step Size for Next Iteration

$h = h_{\text{new}}$

end

DOP853 Core Integrator

Algorithm 3: DOP853 Core Solver

Dense output for DOP853 uses a 7th-order Hermite polynomial to match the method's high-accuracy character. The interpolant requires all fifteen stage evaluations plus additional coefficients computed through linear combinations of the stage derivatives. The resulting seventh-order polynomial enables accurate solution queries throughout each integration step, essential for precise event location and smooth trajectory reconstruction in long-duration simulations. The nested evaluation scheme mirrors that of DOPRI5 but extends to higher degree: $\boldsymbol{y}(\theta) = \boldsymbol{b}_1 + \theta(\boldsymbol{b}_2 + \theta'(\boldsymbol{b}_3 + ...))$, maintaining numerical stability across the full range $\theta \in [0, 1]$.

The continuation listing below gives the dense-output reconstruction used by DOP853.

DOP853 Dense Output

Input: Interpolation coefficients $\boldsymbol{b}_1, ..., \boldsymbol{b}_8$, interval $[x_{\text{old}}, x_{\text{old}} + h]$, evaluation point ξ

Output: Interpolated solution $\boldsymbol{y}(\xi)$

Compute Interpolation Parameter

$\theta = \frac{\xi - x_{\text{old}}}{h}$

$\theta' = 1 - \theta$

Evaluate Hermite Polynomial

$\boldsymbol{y} = \boldsymbol{b}_1 + \theta(\boldsymbol{b}_2 + \theta'(\boldsymbol{b}_3 + \theta(\boldsymbol{b}_4 + \theta'(\boldsymbol{b}_5 + \theta(\boldsymbol{b}_6 + \theta'(\boldsymbol{b}_7 + \theta\boldsymbol{b}_8))))))$

return $\boldsymbol{y}$

Algorithm 4: DOP853 Dense Output Evaluation

While Dormand-Prince methods excel for non-stiff problems, they become inefficient when problem dynamics introduce severe stability restrictions that force prohibitively small step sizes. Such stiff problems arise in astrodynamics from close planetary encounters, highly coupled multi-physics models, or differential-algebraic constraints. The following section describes implicit Runge-Kutta methods, which achieve superior stability for stiff systems at the cost of solving nonlinear systems at each step.

4.2 Implicit Runge-Kutta Methods

An implicit Runge-Kutta method solves a coupled system of nonlinear equations at each step: $\boldsymbol{k}_i = \boldsymbol{f}\left(x_n + c_i h, \boldsymbol{y}_n + h \sum_{j=1}^{s} a_{ij} \boldsymbol{k}_j\right)$ for $i = 1, ..., s$, where the Butcher matrix

$\boldsymbol{A}$ is not strictly lower triangular. This implicit coupling enables stability properties unattainable by explicit methods, and Table 4 shows the corresponding coefficient structure in which the stages must be solved simultaneously.

c_1	a_{11}	a_{12}	a_{13}
c_2	a_{21}	a_{22}	a_{23}
c_3	a_{31}	a_{32}	a_{33}
	b_1	b_2	b_3

Table 4: General structure of a Butcher tableau for an implicit Runge-Kutta method. Unlike explicit methods, the Butcher matrix $\boldsymbol{A} = (a_{ij})$ has non-zero entries on and above the diagonal, indicating that stages are coupled and must be solved simultaneously. This implicit structure enables superior stability properties for stiff problems at the cost of solving nonlinear systems at each step.

For stiff systems, where stability constraints severely restrict the step size of explicit methods, implicit Runge-Kutta methods provide superior performance. Stiff problems arise in astrodynamics from close encounters, highly coupled multi-physics models, and differential-algebraic equation (DAE) formulations. Implicit methods achieve unconditional stability properties (A-stability or L-stability) at the cost of solving nonlinear systems at each integration step. The Radau IIA family offers an excellent balance of high-order accuracy, L-stability, and robustness for stiff ODEs and DAEs.

4.2.1 RADAU5 Method

The three-stage Radau IIA method (RADAU5) is an L-stable implicit Runge-Kutta method of order five [9]. L-stability ensures that spurious oscillations are damped even for extremely stiff problems, making RADAU5 particularly suitable for challenging astrodynamics applications involving close encounters, highly coupled multi-physics models, or stiff DAE partitions. The implementation in this thesis includes a mass-matrix formulation, simplified Newton iterations with optional numeric Jacobian, LU decompositions of stage matrices, and a defect-based error estimator. The complete Butcher tableau coefficients are provided in Appendix C.

Unlike explicit methods, RADAU5 requires solving nonlinear systems at each step. The implementation uses simplified Newton iteration where the Jacobian and mass matrix decompositions are reused across multiple steps until convergence degrades. The method constructs two complex-valued linear systems (one real system and one complex conjugate pair) corresponding to the eigenvalues of the Butcher matrix. This

transformation enables efficient solution of the three-stage system through two LU decompositions per Jacobian update.

The step size controller uses the classical $h_{\text{new}} = h\left(\frac{\text{tol}}{\text{err}}\right)^{\frac{1}{4}}$ scaling for the order-5 method (with embedded order-4 error estimate), modified by safety factors and optional Gustafsson predictive control. The method also supports differential-algebraic equations (DAEs) of index 1, 2, and 3 through appropriate scaling of the error tolerances for algebraic variables.

Computational Complexity: For a system with n state variables, RADAU5 requires solving nonlinear systems of dimension $3n$ at each step, using simplified Newton iteration. Each Newton iteration involves LU decomposition of $3n \times 3n$ matrices, costing $O((3n)^3) = O(27n^3)$ per Jacobian update. However, Jacobian reuse across multiple steps amortizes this cost. The key advantage for stiff problems is that step sizes can be much larger than explicit methods, often resulting in 3–10× fewer total steps despite the higher per-step cost. Memory requirements are $O(n^2)$ for storing the Jacobian matrix and its LU decomposition, which becomes a consideration for very large systems.

The simplified-Newton stepping logic used by RADAU5 is summarized in the algorithm listing below.

RADAU5 Core Integrator

Input: ODE/DAE system f, mass matrix M, Jacobian J, initial state $(x_0, \boldsymbol{y}_0)$, endpoint x_{end}, tolerances $(\mathbf{rtol}, \mathbf{atol})$

Output: Solution $(x, \boldsymbol{y})$ and integration statistics: n_{fev} (function evaluations), n_{jac} (Jacobian evaluations), n_{lu} (LU decompositions), n_{steps} (total steps), n_{accept} (accepted steps), n_{reject} (rejected steps)

Initialization

Initialize algorithm parameters (with defaults if not provided):

uround $= 2.3 \times 10^{-16}$, safe $= 0.9$, $h_{\text{min factor}} = 0.2$, $h_{\text{max factor}} = 8.0$

$n_{\max} = 100,000$, $n_{\text{newton max}} = 7$, newton tol = derived from tolerances

predictive = true (Gustafsson controller)

DAE partition: n_{ind1}, n_{ind2}, n_{ind3} (if omitted, all index-1)

posneg $= \text{sign}(x_{\text{end}} - x)$

Evaluate $\boldsymbol{f}_0 = f(x, \boldsymbol{y})$

$n_{\text{fev}} = 1$

if $h = 0$ **then**

12 | $h = 1 \times 10^{-6} \times \text{posneg}$
13 **end**
14 $h = \text{sign}(h) \times \min(|h|, h_{\max})$
15 Evaluate mass matrix: $M = \text{mass}()$
16 Initialize error scale: $\boldsymbol{s}_i = \mathbf{atol}_i + \mathbf{rtol}_i \ |\boldsymbol{y}_i|$

Main Integration Loop

17 **while** $x \neq x_{\text{end}}$ **do**
18 | **if** $n_{\text{steps}} > n_{\max}$ **then**
19 | | **return** error: need larger $n_{\max}$
20 | **end**
21 | **if** $0.1|h| \leq |x| \times \text{uround}$ **then**
22 | | **return** error: step size too small
23 | **end**
24 | $n_{\text{steps}} = n_{\text{steps}} + 1$

Jacobian and Matrix Decomposition

25 | Evaluate Jacobian: $J = \text{jac}(x, \boldsymbol{y})$
26 | $n_{\text{jac}} = n_{\text{jac}} + 1$

Build system matrices for implicit stages

27 | $\text{U1} = 3.637834252744496, \quad \alpha = 2.6810828736277523, \quad \beta = 3.0504301992474105$
28 | $E_1 = \left(\frac{U_1}{h}\right) M - J$
29 | $E_2^R = \left(\frac{\alpha}{h}\right) M - J$
30 | $E_2^I = \left(\frac{\beta}{h}\right) M$

LU decomposition

31 | Perform LU decomposition of E_1: $E_1 = L_1 U_1$
32 | Perform LU decomposition of $E_2 = E_2^R + iE_2^I$: $E_2 = L_2 U_2$
33 | $n_{\text{lu}} = n_{\text{lu}} + 2$
34 | **if** singular **then**
35 | | $h = \frac{h}{2}$, reject step, continue
36 | **end**

Initialization of Stage Increments

if first step **then**

  $\boldsymbol{z}_1 = \boldsymbol{z}_2 = \boldsymbol{z}_3 = \mathbf{0}$

else

  Extrapolate stages from previous step using Nordsieck interpolation

  $c_{3q} = \frac{h}{h_{\text{old}}}$

  $\boldsymbol{z}_j$ = interpolated increments from dense output

end

Simplified Newton Iteration Loop

$\theta = 0.001$, faccon $= 1.0$, $n_{\text{newton}} = 0$

do

  if $n_{\text{newton}} \geq n_{\text{newton max}}$ **then**

    Reduce h by factor 0.5, reject step, break

  end

  Compute Stages

  $\boldsymbol{k}_1 = f(x + c_1 h, \boldsymbol{y} + \boldsymbol{z}_1)$

  $\boldsymbol{k}_2 = f(x + c_2 h, \boldsymbol{y} + \boldsymbol{z}_2)$

  $\boldsymbol{k}_3 = f(x + h, \boldsymbol{y} + \boldsymbol{z}_3)$

  $n_{\text{fev}} = n_{\text{fev}} + 3$

  Transform to Stage Equations

  Compute T^{-1} (inverse transformation matrix) times $\boldsymbol{k}$ vectors

  $\boldsymbol{z}_1^{\text{t}} = T_{1,1}^{-1}\boldsymbol{k}_1 + T_{1,2}^{-1}\boldsymbol{k}_2 + T_{1,3}^{-1}\boldsymbol{k}_3$

  $\boldsymbol{z}_2^{\text{t}} = T_{2,1}^{-1}\boldsymbol{k}_1 + T_{2,2}^{-1}\boldsymbol{k}_2 + T_{2,3}^{-1}\boldsymbol{k}_3$

  $\boldsymbol{z}_3^{\text{t}} = T_{3,1}^{-1}\boldsymbol{k}_1 + T_{3,2}^{-1}\boldsymbol{k}_2 + T_{3,3}^{-1}\boldsymbol{k}_3$

  Add Mass Matrix Contributions

  $\boldsymbol{z}_1 \mathrel{+}= \frac{M\boldsymbol{z}_1^{\text{t}}}{h} \times U_1$

  $\boldsymbol{z}_2 = \boldsymbol{z}_2^{\text{t}} + M\boldsymbol{z}_2^{\text{t}} \times \left(\frac{\alpha}{h}\right) - M\boldsymbol{z}_3^{\text{t}} \times \left(\frac{\beta}{h}\right)$

  $\boldsymbol{z}_3 = \boldsymbol{z}_3^{\text{t}} + M\boldsymbol{z}_3^{\text{t}} \times \left(\frac{\alpha}{h}\right) + M\boldsymbol{z}_2^{\text{t}} \times \left(\frac{\beta}{h}\right)$

  Solve Linear Systems

  Solve $E_1 \boldsymbol{z}_1 = \text{RHS}_1$ for real stage increment

  Solve $E_2[\boldsymbol{z}_2; \boldsymbol{z}_3] = \text{RHS}_{\{2,3\}}$ for complex conjugate pair

  Compute Convergence Norm

$\text{dyno} = \sqrt{\frac{1}{3n}\sum_i \left[\left(\frac{\boldsymbol{z}_{1,i}}{\boldsymbol{s}_i}\right)^2 + \left(\frac{\boldsymbol{z}_{2,i}}{\boldsymbol{s}_i}\right)^2 + \left(\frac{\boldsymbol{z}_{3,i}}{\boldsymbol{s}_i}\right)^2\right]}$

Check Newton Convergence

if $n_{\text{newton}} > 1$ **then**

  $\text{thq} = \frac{\text{dyno}}{\text{dyno}_{\text{old}}}$

  if $n_{\text{newton}} = 2$ **then**

    $\theta = \text{thq}$

  else

    $\theta = \sqrt{\text{thq} \times \text{thq}_{\text{old}}}$

  end

  if $\theta < 0.99$ **then**

    $\text{faccon} = \frac{\theta}{1.0-\theta}$

    Remaining iterations: $n_{\text{rem}} = n_{\text{newton max}} - 1 - n_{\text{newton}}$

    $\text{dyth} = \text{faccon} \times \text{dyno} \times \frac{\theta^{n_{\text{rem}}}}{\text{newton tol}}$

    if $\text{dyth} \geq 1$ **then**

      $h = h \times$ reduced factor, reject and break

    end

  else

    Convergence too slow, reject step and break

  end

end

$\text{dyno}_{\text{old}} = \max(\text{dyno}, \text{uround})$

Update Stage Increments

$\boldsymbol{f}_1 \mathrel{+}= \boldsymbol{z}_1$

$\boldsymbol{f}_2 \mathrel{+}= \boldsymbol{z}_2$

$\boldsymbol{f}_3 \mathrel{+}= \boldsymbol{z}_3$

Transform back: $\boldsymbol{z}_j = T_{i,j}\boldsymbol{f}_i$ (using matrix T)

Check Termination Criterion

$n_{\text{newton}} = n_{\text{newton}} + 1$

if $\text{faccon} \times \text{dyno} \leq \text{newton tol}$ **then**

  break (Newton converged)

88 | **end**

89 **while** true

Error Estimation

90 $\boldsymbol{e}$ = error correction from E_1 solve

91 Solve $E_1 \boldsymbol{e}_{\text{corr}} = M\boldsymbol{e}$ (additional LU backsubstitution)

92 $\text{err} = \sqrt{\frac{1}{n}\sum_i \left(\frac{\boldsymbol{e}_i}{\boldsymbol{s}_i}\right)^2}$

93 $n_{\text{lu}} = n_{\text{lu}} + 1$

Step Size Control and Acceptance

94 $\text{fac} = \text{safe} \times \min\left(1.0, \frac{\text{cfac}}{n_{\text{newton}} + 2n_{\text{newton max}}}\right)$

95 $\text{quot} = \max\left(\frac{1}{h_{\text{max factor}}}, \min\left(\frac{1}{h_{\text{min factor}}}, \frac{\text{err}^{-\frac{1}{4}}}{\text{fac}}\right)\right)$

96 $h_{\text{new}} = \frac{h}{\text{quot}}$

Predictive Gustafsson Controller

97 **if** predictive **and** $n_{\text{accept}} > 1$ **then**

98 | $\text{fac gus} = \left(\frac{h_{\text{acc}}}{h}\right) \times \frac{\left(\frac{\text{err}^2}{\text{err}_{\text{acc}}}\right)^{\frac{1}{4}}}{\text{safe}}$

99 | $\text{quot} = \max(\text{quot}, \text{fac gus})$

100 | $h_{\text{new}} = \frac{h}{\text{quot}}$

101 **end**

102 $h_{\text{acc}} = h$, $\text{err}_{\text{acc}} = \max(\text{err}, 0.01)$

Accept or Reject Step

103 **if** $\text{err} \leq 1$ **then**

104 | $n_{\text{accept}} = n_{\text{accept}} + 1$

105 | $x_{\text{old}} = x$, $h_{\text{old}} = h$

106 | $x = x + h$

107 | $\boldsymbol{y} = \boldsymbol{y} + \boldsymbol{z}_3$

| *Compute dense output coefficients*

108 | $\boldsymbol{d}_1 = \boldsymbol{y}$

109 | $\boldsymbol{d}_2 = \frac{\boldsymbol{z}_2 - \boldsymbol{z}_3}{c_2 - 1}$

110 | $\boldsymbol{d}_3$ = Nordsieck coefficient

111 | $\boldsymbol{d}_4$ = Higher-order coefficient

| *Update error scale*

RADAU5 Core Integrator

$\boldsymbol{s}_i = \mathbf{atol}_i + \mathbf{rtol}_i \ |\boldsymbol{y}_i|$

Callback with Dense Output

Call user output function with $(x_{\text{old}}, x, \boldsymbol{y}, \boldsymbol{d}, h)$

Check for Termination

if $x = x_{\text{end}}$ **then**

  return success

end

Evaluate Initial Stage for Next Step

$\boldsymbol{f}_0 = f(x, \boldsymbol{y})$

$n_{\text{fev}} = n_{\text{fev}} + 1$

Reset Counters and Update Step Size

Singular counter = 0

$h_{\text{new}} = |h_{\text{new}}| \times \min(h_{\max}, h_{\text{new}}) \times \text{posneg}$

if previous rejected **then**

  $h_{\text{new}} = \text{posneg} \times \min(|h_{\text{new}}|, |h|)$

end

$h = h_{\text{new}}$

else

$n_{\text{reject}} = n_{\text{reject}} + 1$

$h_{\text{new}} = \frac{h}{\min\left(\frac{1}{h_{\text{min factor}}}, \frac{\text{quot}}{\text{safe}}\right)}$

$h = h_{\text{new}}$

Mark step as rejected, continue loop

end

end

Algorithm 5: RADAU5 Core Solver

Dense output for RADAU5 uses Nordsieck-form continuous interpolation, which expresses the solution as a polynomial in the normalized step parameter. The interpolant maintains consistency with the implicit method's stage values and achieves order four accuracy throughout each integration step.

The continuation listing below shows the dense-output reconstruction and step-acceptance logic for RADAU5.

RADAU5 Dense Output

Input: Nordsieck coefficients $\boldsymbol{d}_1, \boldsymbol{d}_2, \boldsymbol{d}_3, \boldsymbol{d}_4$, interval $[x_{\text{old}}, x_{\text{old}} + h]$, evaluation point ξ

Output: Interpolated solution $\boldsymbol{y}(\xi)$

Compute Interpolation Parameter

$s = \frac{\xi - (x_{\text{old}} + h)}{h}$

$s' = 1 - s$ (complementary parameter)

Evaluate Nordsieck Polynomial

$\boldsymbol{y} = \boldsymbol{d}_1 + s(\boldsymbol{d}_2 + (s - c_2)(\boldsymbol{d}_3 + (s - c_1)\boldsymbol{d}_4))$

return $\boldsymbol{y}$

Algorithm 6: RADAU5 Dense Output Evaluation

4.3 Initial Step Size Selection

Initial step size selection is crucial for efficient integration, regardless of whether the method is explicit or implicit [8]. A poor initial guess can lead to wasted function evaluations during the startup phase or, worse, immediate step rejections that compromise integration efficiency. The heuristic balances competing concerns: the step must be small enough to satisfy error tolerances yet large enough to make meaningful progress toward the endpoint.

The algorithm operates in two phases. First, it estimates the first derivative scale by computing the squared norm $d_f = \|\frac{\boldsymbol{f}_0}{\boldsymbol{s}}\|^2$ where $\boldsymbol{s}$ is the component-wise tolerance scale. If this norm is sufficiently small (indicating either near-equilibrium or very loose tolerances), a conservative default step $h = 10^{-6}$ is chosen. Otherwise, an initial guess $h = 0.01\sqrt{\frac{d_y}{d_f}}$ balances the solution and derivative magnitudes.

Second, the algorithm performs an explicit Euler step to estimate the second derivative magnitude. By evaluating $\boldsymbol{f}_1 = f(x_0 + h, \boldsymbol{y}_0 + h\boldsymbol{f}_0)$, it computes the finite difference approximation $d_{f'} \approx \frac{\|\boldsymbol{f}_1 - \boldsymbol{f}_0\|}{|h|}$. The final step size is then chosen as $h = \left(\frac{0.01}{d_{\max}}\right)^{\frac{1}{p}}$ where p is the method order (5 for DOPRI5, 8 for DOP853) and $d_{\max} = \max\left(d_{f'}, \sqrt{d_f}\right)$ represents the dominant error scale. This order-dependent scaling ensures that the initial step produces local errors commensurate with the method's accuracy.

The heuristic includes safeguards against extreme step sizes: the final value is bounded by $\min\bigl(100|h_{\text{guess}}|, h_1, h_{\max}\bigr)$ to prevent both over-aggressive initial steps and violation of user-specified maximum step size constraints. This automatic initialization proves remarkably robust across diverse problem types, from smooth Kepler orbits to moderately stiff orbital mechanics with perturbations.

The startup heuristic used to estimate the initial step size before adaptive control takes over is given in the algorithm listing below.

Initial Step Size Guess

Input: ODE system f, initial state $(x_0, \boldsymbol{y}_0)$, derivative $\boldsymbol{f}_0$, tolerances $(\mathbf{rtol}, \mathbf{atol})$, method order p

Output: Initial step size h

Estimate First Derivative Scale

$\boldsymbol{s} = \mathbf{atol} + \mathbf{rtol} \odot |\boldsymbol{y}_0|$

$d_f = \sum_{i=1}^{n} \left(\frac{\boldsymbol{f}_{0,i}}{\boldsymbol{s}_i}\right)^2$

$d_y = \sum_{i=1}^{n} \left(\frac{\boldsymbol{y}_{0,i}}{\boldsymbol{s}_i}\right)^2$

if $d_f \leq 10^{-10}$ or $d_y \leq 10^{-10}$ **then**

  $h = 10^{-6}$

else

  $h = 0.01\sqrt{\frac{d_y}{d_f}}$

end

Enforce $|h| \leq h_{\max}$

$h = |h| \cdot \operatorname{sign}(x_{\text{end}} - x_0)$

Perform Explicit Euler Step

$\boldsymbol{y}_1 = \boldsymbol{y}_0 + h\boldsymbol{f}_0$

$\boldsymbol{f}_1 = f(x_0 + h, \boldsymbol{y}_1)$

Estimate Second Derivative Scale

$\boldsymbol{s} = \mathbf{atol} + \mathbf{rtol} \odot |\boldsymbol{y}_0|$

$d_{f'}^2 = \frac{\sqrt{\sum_{i=1}^{n} \left(\frac{\boldsymbol{f}_{1,i} - \boldsymbol{f}_{0,i}}{\boldsymbol{s}_i}\right)^2}}{|h|}$

$d_{\max} = \max\bigl(|d_{f'}^2|, \sqrt{d_f}\bigr)$

Compute Step Size Based on Method Order

if $d_{\max} \leq 10^{-15}$ **then**

Initial Step Size Guess

$h_1 = \max(10^{-6}, 10^{-3}\ |h|)$

else

$h_1 = \left(\frac{0.01}{d_{\max}}\right)^{\frac{1}{p}}$

end

$h = \text{sign}(h) \cdot \min(100|h|, h_1, h_{\max})$

return h

Algorithm 7: Initial Step Size Heuristic

4.4 Comprehensive Method Comparison

Table 5 is used below to compare the numerical methods implemented in the `ivp` library by order, stiffness properties, and intended astrodynamics use case.

Method	**Order**	**Stages**	**Stable**	**Stiff**	**DAE**	**Complexity**	**Use Case**
`RK4`	4	4	Conditional	No	No	O(4n)	Educational, simple problems
`RK23`	3(2)	3	Conditional	No	No	O(3n)	Low-accuracy requirements, prototyping
`DOPRI5`	5(4)	7 to 6	Conditional	No	No	O(6n)	General non-stiff problems, moderate accuracy
`DOP853`	8(5,3)	16 to 15	Conditional	No	No	O(10n)	High-accuracy non-stiff, long-term propagation
`RADAU5`	5	3	L-stable	Yes	Yes, idx<=3	$O(27n^3)$	Stiff ODEs/

							DAEs, close encounters
BDF	1–5	Variable	A-stable	Yes	Yes, idx<=1	$O(n^2)$	Variable stiffness, large systems

Table 5: Comprehensive comparison of implemented numerical methods. The "Stages" column shows total stages with effective stages per step (accounting for FSAL property). "Stable" indicates stability properties: Conditional (explicit methods) or A/L-stable (implicit methods). "Stiff" and "DAE" indicate support for stiff systems and differential-algebraic equations. "Complexity" shows per-step computational complexity for n state variables.

The methods form a complementary suite covering the full spectrum of astrodynamics problems:

Explicit methods (RK4, RK23, DOPRI5, DOP853) excel for non-stiff dynamics where Jacobian eigenvalues do not impose severe step size restrictions. These methods are computationally efficient per step, requiring only derivative evaluations without solving nonlinear systems. DOPRI5 and DOP853 include adaptive step size control and dense output, making them suitable for most orbit propagation tasks where gravitational forces are smooth and well-behaved.

Implicit methods (RADAU5, BDF) are essential for stiff problems arising from close encounters, highly coupled multi-physics models, or differential-algebraic formulations. RADAU5 provides L-stability and DAE support up to index 3, making it ideal for the most challenging stiff problems. BDF offers variable-order adaptivity and is efficient for problems where stiffness varies significantly throughout the integration interval.

The choice between methods involves trading off computational cost, accuracy requirements, and problem characteristics. For typical astrodynamics applications (Keplerian orbits, perturbed trajectories, multi-body dynamics), DOPRI5 provides an excellent balance of accuracy, efficiency, and robustness. For long-term high-accuracy propagation, DOP853′s higher order allows larger step sizes. When encountering stiff regions (close planetary approaches, atmospheric entry, coupled DAEs), RADAU5 or BDF become necessary despite their higher per-step cost.

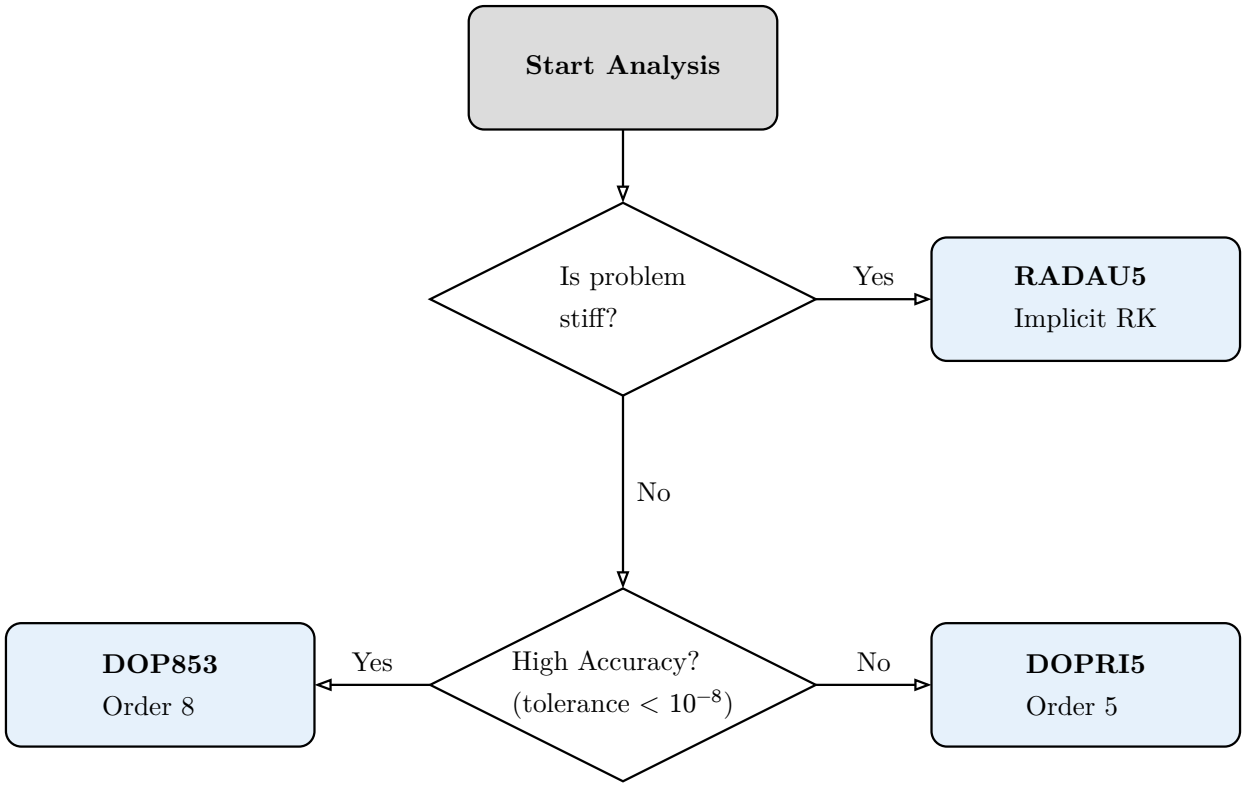

Figure 4: Decision tree for selecting numerical integration methods. The primary decision branch splits based on stiffness; stiff problems require implicit methods like RADAU5. For non-stiff problems, the choice between DOPRI5 and DOP853 depends on the stringency of accuracy tolerances. Educational and prototyping needs are served by simpler RK methods.

4.5 Practical Guidelines for Method Selection

Based on the problem characteristics and computational requirements, the following guidelines assist in selecting appropriate numerical integration methods for astrodynamics applications. A visual decision process is illustrated in Figure 4.

Non-stiff, moderate accuracy ($\mathbf{rtol \approx 10^{-6}}$): Use DOPRI5 as the default choice. This fifth-order method provides an optimal balance of accuracy and computational efficiency for general non-stiff orbital mechanics problems. The embedded fourth-order error estimate enables reliable adaptive step size control without excessive computational overhead.

Non-stiff, high accuracy ($\mathbf{rtol \approx 10^{-9}}$): Use DOP853 when stringent accuracy requirements are necessary, such as precision navigation, orbit determination, or interferometry mission design. The eighth-order accuracy allows for significantly larger step sizes than lower-order methods, often resulting in faster total computation time despite

the higher per-step cost. The dual error estimators (5th and 3rd order) provide robust error control, and the seventh-order dense output enables accurate event location.

Stiff, moderate accuracy (rtol $\approx 10^{-6}$): Use RADAU5 for stiff differential equations where explicit methods would require impractically small step sizes. Stiffness in astrodynamics arises from atmospheric entry/reentry (rapid velocity changes due to drag), close encounters (high gravitational accelerations), or stiff DAE partitions (algebraic constraints coupled with differential equations). RADAU5′s L-stability ensures that spurious high-frequency oscillations are damped, and its support for DAEs up to index 3 makes it versatile for constrained systems.

Variable stiffness: Use BDF when the problem exhibits varying stiffness throughout the integration interval. The variable-order adaptivity allows BDF to use high orders (4–5) during smooth evolution and automatically reduce to first-order during stiff transients. This capability is valuable for spacecraft mission profiles that include both quiescent orbit phases and dynamic maneuvers, or for problems where stiffness onset is not predictable in advance.

Simple educational examples: Use RK4 for teaching, verification, or problems where uniform stepping is acceptable. While RK4 lacks adaptive error control and is less efficient than embedded methods, its simplicity makes it ideal for educational purposes, algorithm verification, and problems where computational cost is not a primary concern. RK4 serves as a valuable baseline for evaluating more sophisticated methods.

Low-accuracy prototyping: Use RK23 when rapid development is prioritized over high precision. The third-order accuracy with embedded second-order error estimator provides reasonable accuracy with minimal computational cost, making it suitable for preliminary mission design trades, parameter sweeps, and exploratory analysis where computational efficiency is paramount.

Additional considerations for method selection include:

Memory constraints: For systems with large state dimensions (ensemble propagations, finite element discretizations), RADAU5′s $O(n^2)$ memory requirement for the Jacobian may become prohibitive. In such cases, consider BDF (which stores only a few previous solutions) or lower-order explicit methods with careful memory management.

Parallel computing: Explicit methods offer greater parallelization potential since stage evaluations can be distributed across cores or GPUs. Implicit methods present

parallelization challenges due to the sequential nature of Newton iterations, though parallel linear algebra solvers can provide some speedups.

Derivative evaluation cost: When derivative evaluations are extremely expensive (e.g., involving complex force models, ephemeris lookups, or coordinate transformations), the lower number of stages in DOPRI5 (6 per step) compared to DOP853 (15 per step) makes DOPRI5 more efficient unless the higher order enables substantially larger step sizes.

Event detection: All methods support event detection through root-finding algorithms, but the cost of evaluating event functions should be considered. For problems with frequent events, explicit methods with dense output (DOPRI5, DOP853) provide efficient root location without additional derivative evaluations.

The `ivp` library provides a unified interface through the `solve_ivp` function that automatically dispatches to the appropriate method based on the problem specification, simplifying method selection for users. The library also supports automatic stiffness detection and can recommend method switches when explicit methods become inefficient.

5 Software Design and Architecture

This chapter describes the design and implementation of the `ivp` library, a pure Rust implementation of numerical integrators for initial value problems with Python bindings [36]. The library provides Rust-native alternatives to both the classical Fortran integrators and SciPy's `solve_ivp` function, enabling modern applications to perform high-fidelity orbit propagation without requiring foreign function interfaces, cross-compilation toolchains, or Python runtime overhead.

5.1 Design Goals

The primary motivation for developing the `ivp` library is to provide a unified solution for numerical integration that serves both Rust and Python ecosystems. Existing approaches present significant limitations:

- **Fortran FFI approach**: Linking against Fortran libraries (DOPRI5, DOP853, RADAU5) through foreign function interfaces introduces build system complexity, cross-platform compatibility issues, and runtime dependencies on Fortran compilers and libraries.
- **Python/SciPy approach**: While SciPy's `solve_ivp` provides an excellent interface, the underlying implementations for methods like `RK45` and `DOP853` are written in Python. This incurs significant interpreter overhead on every derivative evaluation and within the solver logic itself, limiting performance for compute-intensive applications.
- **Existing Rust crates**: Pure-Rust ODE solvers exist but often lack the sophistication of mature Fortran codes, particularly regarding step size control heuristics, stiffness detection, and dense output interpolation.

The `ivp` library addresses these limitations with the following design goals:

1. **Algorithmic fidelity**: Preserve the numerical behavior of the original Hairer and Wanner Fortran implementations [8], [9], including step size control heuristics, error estimation, stiffness detection, and dense output interpolation.
2. **Native Rust API**: Provide an idiomatic Rust interface using traits, generics, and the builder pattern for solver configuration.
3. **Python compatibility**: Expose a Python application programming interface (API) via PyO3 [37] that mirrors SciPy's `solve_ivp` interface, allowing seamless adoption in existing Python workflows while benefiting from Rust's performance.

4. **Flexible precision**: Support configurable floating-point precision through compile-time feature flags (`f32` or `f64`).
5. **Comprehensive solver suite**: Implement multiple integration methods covering both non-stiff (RK4, RK23, DOPRI5, DOP853) and stiff (`RADAU`/RADAU5, BDF) problem classes.

5.2 Library Architecture

The `ivp` library is organized into modular components that separate problem definition, numerical methods, solution handling, and language bindings, as summarized in Table 6.

Module	Components	Responsibility
`ivp::core`	IVP trait, ODEProblem, Matrix types	Problem definition and data structures
`ivp::solvers`	DOPRI5, DOP853, RADAU, BDF, RK4, RK23	Numerical integration algorithms
`ivp::options`	Options, Method enum, Tolerance	Solver configuration and API
`ivp::solution`	Solution, ContinuousOutput, EventConfig	Result handling and interpolation
`ivp::python`	PyO3 bindings, NumPy integration	Python API and FFI layer

Table 6: Library module organization for the ivp library.

The architecture separates concerns cleanly: users define problems through the `IVP` trait, the library provides solver implementations, and results are returned through consistent `Solution` types. The Python bindings layer translates between Python objects and Rust types, with NumPy arrays passed by reference to avoid unnecessary copying.

5.2.1 Problem Definition via the IVP Trait

The ODE system is defined through the `IVP` trait:

```
pub trait IVP {
    fn ode(&self, t: f64, y: &[f64], dydt: &mut [f64]);

    // Optional: Jacobian for implicit methods
    fn jac(&self, t: f64, y: &[f64], j: &mut [f64]) {
      /* finite difference default */
```

```
    }

    // Optional: Event detection
    fn events(&self, t: f64, y: &[f64], out: &mut [f64]) {
      /* default: no events */
    }
    fn n_events(&self) -> usize {
      /* default: no events */
      0
    }
    fn event_config(&self, index: usize) -> EventConfig {
      /* default: terminal = false, direction = 0 */
      EventConfig::default()
    }
}
```

This trait-based design enables static dispatch in Rust, allowing the compiler to inline derivative evaluations directly into the integration loop. The immutable `&self` reference (compared to mutable in simpler designs) permits thread-safe sharing of problem parameters. The optional `jac` method provides analytical Jacobians for implicit methods, with a finite-difference fallback using forward differences with $\varepsilon = \sqrt{\varepsilon_{\text{machine}}}$ perturbations.

5.2.2 Detailed Usage Examples

This section provides representative examples covering native Rust usage, event detection, stiff integration, and Python interoperability.

5.2.2.1 Example 1: Simple Two-Body Orbit Propagation

A fundamental task in astrodynamics is propagating Keplerian orbits. As described in Chapter 3, the two-body problem provides an analytical foundation for understanding spacecraft motion under gravitational attraction. The following example shows how to define and solve the two-body problem using the `ivp` library:

```
use ivp::prelude::*;
use std::f64::consts::PI;

struct TwoBody {
    mu: f64,
}

impl IVP for TwoBody {
    fn ode(&self, _t: f64, y: &[f64], dydt: &mut [f64]) {
```

```rust
        let r = (y[0].powi(2) + y[1].powi(2) + y[2].powi(2)).sqrt();
        let r3 = r.powi(3);
        let mu_over_r3 = self.mu / r3;

        // Position derivatives = velocities
        dydt[0] = y[3];
        dydt[1] = y[4];
        dydt[2] = y[5];

        // Velocity derivatives = accelerations
        dydt[3] = -mu_over_r3 * y[0];
        dydt[4] = -mu_over_r3 * y[1];
        dydt[5] = -mu_over_r3 * y[2];
    }
}

fn main() {
    // Define problem and options
    let mu = 398600.4418_f64;
    let problem = TwoBody { mu }; // Earth gravitational parameter
    let options = Options::builder()
        .method(Method::DOP853)
        .rtol(1e-9)
        .atol(1e-12)
        .dense_output(true)
        .build();

    // Initial conditions: circular orbit at 400 km altitude
    let r: f64 = 6778.137; // Earth radius + 400 km
    let v = (mu / r).sqrt(); // Circular velocity
    let y0 = [r, 0.0_f64, 0.0_f64, 0.0_f64, v, 0.0_f64];
    let orbital_period = 2.0 * PI * (r.powi(3) / mu).sqrt();

    // Integrate for one exact circular-orbit period
    let solution = solve_ivp(&problem, 0.0, orbital_period, &y0,
options).unwrap();
    let y_final = solution.y.last().unwrap();
    let closure_error_km = ((y_final[0] - y0[0]).powi(2)
        + (y_final[1] - y0[1]).powi(2)
        + (y_final[2] - y0[2]).powi(2))
    .sqrt();

    println!("Orbital period: {:.3} s", orbital_period);
```

```
    println!(
        "Final position: ({:.6}, {:.6}, {:.6}) km",
        y_final[0],
        y_final[1],
        y_final[2]
    );
    println!("Position closure error: {:.3} m", closure_error_km * 1000.0);
    println!("Status: {:?}", solution.status);
    println!("Function evaluations: {}", solution.nfev);
}
```

Output:

```
Orbital period: 5553.624 s
Final position: (6778.137000, -0.000017, 0.000000) km
Position closure error: 0.017 m
Status: Success
Function evaluations: 467
```

This example demonstrates the core workflow: define an `IVP` implementer, create a solver with builder pattern, specify initial conditions and time span, and extract results. Because the propagation interval is chosen to match the analytical period of the circular orbit, the final position returns to the initial state to within centimeter-level closure error. The `Solution` type provides access to time points, state vectors, performance metrics (function evaluations, Jacobian evaluations, LU decompositions), and status.

5.2.2.2 Example 2: Event Detection

Event detection is crucial for mission design tasks such as finding orbit insertion points, eclipse periods, or collision events. The following example demonstrates event detection for a bouncing ball with ground impact:

```
use ivp::prelude::*;

struct SimpleEvent {
    gravity: f64,
}

impl IVP for SimpleEvent {
    fn ode(&self, _t: f64, state: &[f64], dsdt: &mut [f64]) {
        dsdt[0] = state[1];
        dsdt[1] = -self.gravity;
    }

    fn events(&self, _t: f64, state: &[f64], out: &mut [f64]) {
```

```
        out[0] = state[0]; // Detect when position crosses zero
    }

    fn n_events(&self) -> usize {
        1
    }

    fn event_config(&self, _index: usize) -> EventConfig {
        let mut config = EventConfig::default();
        config.terminal();
        config.negative();
        config
    }
}

fn main() {
    let ball = SimpleEvent { gravity: 9.81 };
    let y0 = [10.0_f64, 0.0_f64]; // [height, velocity]

    let options = Options::builder()
        .method(Method::DOPRI5)
        .rtol(1e-8)
        .atol(1e-10)
        .build();

    let solution = solve_ivp(&ball, 0.0, 10.0, &y0, options).unwrap();

    println!("Status: {:?}", solution.status);
    println!("Initial height: {:.4} m", y0[0]);

    if let Some(t_impact) = solution.t_events.get(0).and_then(|e| e.first()) {
        println!("Ground impact at t = {:.4} s", t_impact);
        let v_impact = solution.y_events[0][0][1].abs();
        println!("Impact velocity: {:.4} m/s", v_impact);
    }

    println!("\nTrajectory (first 10 points):");
    for (t, y) in solution.iter().take(10) {
        println!("t={:.4}: height={:.4}, velocity={:.4}", t, y[0], y[1]);
    }
}
```

Output:

```
Status: UserInterrupt
Initial height: 10.0000 m
Ground impact at t = 1.4278 s
Impact velocity: 14.0071 m/s

Trajectory (first 10 points):
t=0.0000: height=10.0000, velocity=0.0000
t=0.0000: height=10.0000, velocity=-0.0001
t=0.0001: height=10.0000, velocity=-0.0011
t=0.0011: height=10.0000, velocity=-0.0111
t=0.0113: height=9.9994, velocity=-0.1110
t=0.1131: height=9.9372, velocity=-1.1100
t=1.1315: height=3.7202, velocity=-11.1000
t=1.4278: height=-0.0000, velocity=-14.0071
```

The solver locates the event through dense output interpolation and terminates at the detected impact time.

5.2.2.3 Example 3: Stiff Problem with `RADAU` (RADAU5)

For stiff problems, the implicit `RADAU` solver, which implements the three-stage fifth-order RADAU5 method, can efficiently handle the challenging dynamics:

```rust
use ivp::prelude::*;

struct VanDerPol {
    mu: f64,
}

impl IVP for VanDerPol {
    fn ode(&self, _t: f64, y: &[f64], dydt: &mut [f64]) {
        dydt[0] = y[1];
        dydt[1] = self.mu * (1.0 - y[0].powi(2)) * y[1] - y[0];
    }
}

fn main() {
    let problem = VanDerPol { mu: 1000.0 };
    let options = Options::builder()
        .method(Method::RADAU)
        .rtol(1e-6)
        .atol(1e-10)
        .max_steps(100000)
        .build();
```

```
    let y0 = [2.0, 0.0];
    let solution = solve_ivp(&problem, 0.0, 3000.0, &y0, options).unwrap();

    println!("Status: {:?}", solution.status);
    println!("Solution required {} steps", solution.nstep);
    println!("Jacobian evaluations: {}", solution.njev);
    println!("Function evaluations: {}", solution.nfev);
    println!("LU decompositions: {}", solution.nlu);
}
```

Output:

```
Status: Success
Solution required 940 steps
Jacobian evaluations: 873
Function evaluations: 5873
LU decompositions: 2800
```

The `RADAU` solver handles the stiff Van der Pol oscillator by automatically managing Newton iterations and LU decompositions while meeting the requested tolerances.

5.2.2.4 Example 4: Python Interface with SciPy Compatibility

The Python bindings provide a SciPy-compatible interface, enabling seamless adoption:

```python
import numpy as np
import ivp
import matplotlib.pyplot as plt
from pathlib import Path

# Define ODE right-hand side
def two_body(t, y, mu=398600.4418):
    r = np.sqrt(y[0]**2 + y[1]**2 + y[2]**2)
    mu_over_r3 = mu / r**3
    return [
        y[3],  # dx/dt
        y[4],  # dy/dt
        y[5],  # dz/dt
        -mu_over_r3 * y[0],  # dvx/dt
        -mu_over_r3 * y[1],  # dvy/dt
        -mu_over_r3 * y[2],  # dvz/dt
    ]

# Initial conditions: circular LEO orbit
mu = 398600.4418
r0 = 6778.137  # Earth radius + 400 km
```

```
v0 = np.sqrt(mu / r0)
orbital_period = 2.0 * np.pi * np.sqrt(r0**3 / mu)
y0 = [r0, 0.0, 0.0, 0.0, v0, 0.0]

print(f"Two-Body Orbit Propagation")
print(f"Initial radius: {r0:.3f} km")
print(f"Initial velocity: {v0:.3f} km/s")
print(f"Orbital period: {orbital_period:.3f} s")

# Solve using DOP853 with high accuracy over one exact orbital period
sol = ivp.ivp.solve_ivp(
    two_body,
    (0.0, orbital_period),
    y0,
    method="DOP853",
    args=(mu,),
    rtol=1e-9,
    atol=1e-12,
    dense_output=True,
)

# Sample the dense output on a fine uniform grid for a smooth closed orbit
plot
plot_times = np.linspace(0.0, orbital_period, 1000)
plot_state = sol.sol(plot_times)
closure_error_km = np.linalg.norm(plot_state[:3, -1] - np.array(y0[:3]))

# Extract results
x = plot_state[0]
y = plot_state[1]

# Plot orbit (2D since orbit is in XY plane with Z=0)
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(x, y, label="Orbit", linewidth=2)
ax.set_xlabel("X Position (km)")
ax.set_ylabel("Y Position (km)")
ax.set_title("LEO Orbit - One Exact Period")
ax.legend()
ax.grid(True, alpha=0.3)
ax.axis('equal')  # Equal aspect ratio for circular appearance

plt.tight_layout()
output_path = Path("output") / "orbit.png"
```

```
output_path.parent.mkdir(parents=True, exist_ok=True)
plt.savefig(output_path, dpi=150)

# Print results
print(f"\nSolver status: {sol.message}")
print(f"Final time: {sol.t[-1]:.3f} s")
print(f"Function evaluations: {sol.nfev}")
print(f"Final position: ({plot_state[0, -1]:.6f}, {plot_state[1, -1]:.6f},
{plot_state[2, -1]:.6f}) km")
print(f"Position closure error: {closure_error_km * 1000.0:.3f} m")
```

Output:

```
Two-Body Orbit Propagation
Initial radius: 6778.137 km
Initial velocity: 7.669 km/s
Orbital period: 5553.624 s

Solver status: Success
Final time: 5553.624 s
Function evaluations: 467
Final position: (6778.137000, -0.000017, 0.000000) km
Position closure error: 0.017 m

Plot saved to output/orbit.png
```

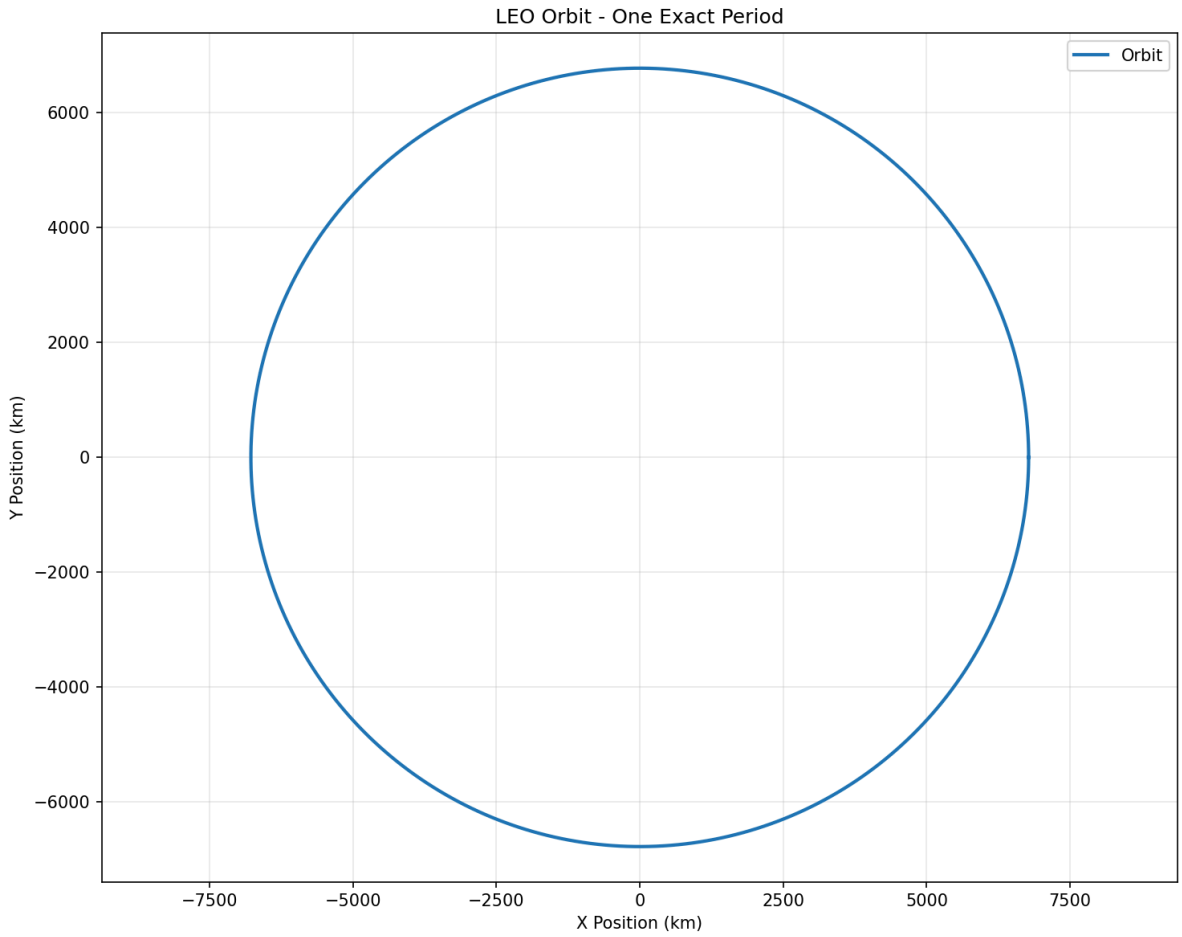

Figure 5: LEO orbit over one exact analytical period, sampled from the dense-output interpolant on a uniform grid.

The Python interface returns an `OdeResult` object with attributes matching SciPy's `solve_ivp` return type, allowing existing SciPy workflows to migrate with minimal code changes. The propagated orbit sampled from the dense-output interpolant is shown in Figure 5.

5.2.3 API Reference Table

Module	Function/ Type	Parameters	Returns	Description
`ivp`	`IVP` trait	ode, jac, events, n_events, event_config	N/A	Problem definition trait with optional methods for Jacobian and events

`ivp`	`solve_ivp`	problem, t0, t_end, y0, options	`Result<Solution, Error>`	Unified solver interface dispatching to appropriate method
`ivp::Options`	`.builder()`	method, rtol, atol, max_steps, dense_output, t_eval	`Options` struct	Builder for configuring solver options
`ivp`	`Method` enum	N/A	RK4, RK23, DOPRI5, DOP853, RADAU, BDF	Enumeration of available integration methods
`ivp`	`Solution`	N/A	t, y, t_events, y_events, nfev, njev, nlu, status, nstep, sol	Complete integration result with performance metrics
`ivp`	`EventConfig`	N/A	terminal, direction	Configuration for event detection

Table 7: Summary of public API components for the ivp library.

The API is designed for discoverability and consistency. Solver selection, tolerances, and output controls are all configured through the same `Options` builder, so users interact with a unified interface regardless of which numerical method is chosen. Table 7 gathers the main public entry points used throughout these examples.

5.2.4 Unified Solver Interface

The `solve_ivp` function provides a unified entry point dispatching to the appropriate method:

```
pub fn solve_ivp<F: IVP>(
    f: &F,
    t0: f64,
    t_end: f64,
    y0: &[f64],
```

```
    options: Options,
) -> Result<Solution, Error>
```

5.2.5 Solution Type and Dense Output

The `Solution` type encapsulates both discrete samples and continuous interpolation capabilities:

```
pub struct Solution {
    pub t: Vec<f64>,                    // Time points
    pub y: Vec<Vec<f64>>,               // State vectors
    pub t_events: Vec<Vec<f64>>,        // Event times per event function
    pub y_events: Vec<Vec<Vec<f64>>>,   // States at events
    pub nfev: usize,                    // Function evaluations
    pub njev: usize,                    // Jacobian evaluations
    pub nlu: usize,                     // LU decompositions
    pub nstep: usize,                   // Number of steps
    pub status: Status,                 // Termination status
    pub sol: Option<ContinuousOutput>,  // Dense output interpolator
}
```

When dense output is enabled, the `ContinuousOutput` stores interpolation coefficients for each accepted step, enabling efficient evaluation at arbitrary times within the integration span:

```
// Evaluate at a single point
let y_interp = solution.sol(1.5)?;

// Query the valid interpolation range
let (t_start, t_end) = solution.sol_span().unwrap();
```

Each method provides its own interpolation polynomial: DOPRI5 uses Shampine's fourth-order formula with 5 coefficients per state [14], DOP853 uses seventh-order interpolation with 8 coefficients, and `RADAU` uses collocation-based interpolation consistent with the implicit solution.

5.3 Python Bindings via PyO3

A key feature of the `ivp` library is its Python API, implemented using PyO3 [37] and compiled as a native extension module. The Python interface mirrors SciPy's `solve_ivp` function signature:

```
import ivp

def van_der_pol(t, y, eps):
    return [y[1], ((1.0 - y[0]**2) * y[1] - y[0]) / eps]
```

```
sol = ivp.solve_ivp(
    van_der_pol,
    (0, 100),
    [2.0, 0.0],
    method='DOP853',
    args=(1.0,),
    rtol=1e-9,
    atol=1e-9,
    dense_output=True
)
```

The Python interface returns an `OdeResult` object with the familiar SciPy fields `t`, `y`, `sol`, `t_events`, `y_events`, `nfev`, `njev`, `nlu`, `status`, and `message`.

The Python wrapper handles:

- **Type conversion**: NumPy arrays to/from Rust buffers via the `numpy` crate
- **Callback wrapping**: Python callable objects wrapped to implement the `IVP` trait
- **Event detection**: Python event functions with `terminal` and `direction` attributes
- **Jacobian support**: Optional analytical Jacobian functions and sparsity structures
- **Error propagation**: Rust errors converted to Python exceptions

5.3.1 Performance Characteristics

The Rust implementation eliminates Python interpreter overhead during the integration loop. While Python callbacks for derivative evaluation still cross the language boundary, the step size control logic, error estimation, and interpolation coefficient computation execute entirely in compiled Rust code. Key optimizations include:

- **Pre-allocated NumPy buffers**: The Python wrapper reuses NumPy arrays across derivative evaluations, updating them in-place via `ptr::copy_nonoverlapping` rather than allocating new arrays on each call.
- **Inlined hot paths**: Critical functions are marked with `#[inline(always)]` to eliminate function call overhead.
- **Branch prediction hints**: Error paths use `#[cold]` attributes to improve branch prediction on the fast path.
- **Link-time optimization**: The release build uses `lto = "fat"` with single codegen unit for maximum cross-module optimization.

Benchmark comparisons on standard test problems demonstrate clear performance gains over SciPy:

Problem	ivp (Rust)	SciPy	Speedup
Van der Pol, DOP853	0.012 s	0.041 s	3.4 ×
Van der Pol, RK45	0.010 s	0.047 s	4.8 ×
Van der Pol stiff, BDF	0.008 s	0.108 s	12.9 ×
Linear system, RK45	0.5 ms	1.5 ms	2.8 ×

Table 8: Benchmark comparison between ivp and SciPy on representative ODE problems. The Van der Pol cases use $\varepsilon = 1$ for the non-stiff runs and $\varepsilon = 10^{-3}$ for the stiff BDF run; the linear system uses $N = 1000$.

The stiff BDF solver shows the largest speedup (12.9 ×) because the implicit method's Newton iterations and LU decompositions execute entirely in Rust, with Python callbacks only for derivative and Jacobian evaluations. The explicit methods show 3–5 × speedups, with the improvement varying based on the ratio of solver overhead to derivative evaluation cost.

5.3.2 Testing Strategy and Coverage

The library employs a comprehensive testing approach spanning unit tests, integration tests, property-based testing, and benchmarking. Overall test coverage is 60.4% (1875 of 3103 lines) as measured by `cargo tarpaulin`, and Table 9 summarizes how these complementary test layers support correctness and regression protection:

Test Type	Tool	Coverage	CI/CD	Purpose
Unit tests	Rust `cargo test`	60%	Local	Verify individual components, boundary conditions
Integration tests	Rust `cargo test`	–	Local	End-to-end solver workflows
Property-based	Rust `quickcheck`	–	Local	Invariants across random inputs
Verification tests	Comparison vs Fortran	100%	Local	Bit-for-bit fidelity to reference
Benchmarks	Criterion	N/A	Local	Performance regression detection

Table 9: Testing strategy overview for the ivp library. Coverage measured with `cargo tarpaulin` v0.35.1.

Unit Tests: Test individual functions and methods in isolation. Examples include:

- Butcher tableau coefficients validation

- Step size controller logic
- Dense output interpolation at specific points
- LU decomposition correctness
- Error norm computation

Integration Tests: Verify that complete workflows produce expected results:
- Full integration of simple harmonic oscillator against analytical solution
- Arenstorf orbit return accuracy
- Event detection precision
- State transition matrix propagation

Property-Based Tests: Use `quickcheck` to verify invariants across randomly generated inputs:
- Order preservation: $y_{i+1} - y_i$ should be within tolerances
- Monotonicity: Energy should not increase for conservative systems
- Symmetry: Reverse time integration should approximately undo forward integration

Verification Tests: Compare Rust implementation results against reference Fortran codes:
- Bit-for-bit agreement on identical test problems with identical parameters
- Matching numbers of accepted/rejected steps
- Identical event detection times (within tolerance)
- Matching final state values

Benchmarks: Use the `criterion` crate for statistically rigorous performance testing:
- Run benchmarks on every commit to detect regressions
- Measure cold vs warm cache performance
- Compare different solver configurations
- Profile memory allocation patterns

The continuous integration pipeline automatically runs these tests on every push and pull request, ensuring that changes do not introduce regressions.

5.3.3 Reproducibility Workflow

To ensure that all results presented in this thesis can be independently reproduced, the project uses automation scripts:

```
git clone --recursive https://github.com/Ryan-D-Gast/ivp
cd ivp

just run-all
```

The `justfile` defines targets for:

- Building the Rust library with optimizations
- Running Fortran reference implementations
- Executing Python benchmarks
- Generating all figures and tables
- Running statistical analysis (Welch's t-test)
- Compiling the thesis document

This automated approach ensures that:

- Results are deterministic (fixed random seeds where applicable)
- Dependencies are pinned (exact versions specified in `Cargo.lock`)
- Environment is consistent (containerized builds available)
- Documentation and code remain synchronized (examples in the thesis are run as tests)

5.4 Implementation Details

5.4.1 Tolerance Handling

Both scalar and component-wise tolerances are supported through an enum with automatic conversion:

```
pub enum Tolerance {
    Scalar(f64),
    Vector(Vec<f64>),
}

impl From<f64> for Tolerance { ... }
impl From<Vec<f64>> for Tolerance { ... }
```

This allows users to pass either `rtol=1e-9` (scalar) or `rtol=vec![1e-9, 1e-12, 1e-9]` (per-component) without changing function signatures. The error scale for component i is computed as $\text{atol}_i + \text{rtol}_i \cdot |y_i|$.

5.4.2 Matrix Operations for Implicit Methods

The `RADAU` and BDF solvers require LU decomposition for the simplified Newton iterations. The library includes a custom `Matrix` type supporting both full and banded storage:

```
pub enum MatrixStorage {
    Full,
    Banded { ml: usize, mu: usize },
}
```

LU decomposition routines handle both real matrices (for BDF) and the complex arithmetic required by the `RADAU` transformation. The self-contained implementation avoids external BLAS/LAPACK dependencies while providing adequate performance for the modest system sizes typical in orbital mechanics.

5.4.3 Step Size Control

All adaptive methods implement the PI (proportional-integral) controller for step size selection [8]:

$$h_{\text{new}} = h \cdot \min\left(f_{\max}, \max\left(f_{\min}, f_{\text{safe}} \cdot \left(\frac{1}{\text{err}}\right)^{\frac{1}{p}} \cdot \left(\frac{1}{\text{err}_{\text{old}}}\right)^{\beta}\right)\right) \tag{37}$$

where p is the method order, f_{safe} is the safety factor (typically 0.9), and β controls the integral term for stability. The explicit methods also include stiffness detection that warns when implicit methods may be more appropriate.

5.4.4 Event Detection

The library supports event detection through the `IVP` trait's optional `events` method. Events are located using bisection with the dense output interpolant, achieving accuracy consistent with the integration tolerances. Event configuration supports:

- **Terminal events**: Stop integration when the event occurs
- **Direction**: Detect crossings in positive direction, negative direction, or both

5.5 API and Data Formats

The `ivp` library is designed to integrate seamlessly into modern data science and engineering workflows. This section describes the API design principles and the data formats used for input and output.

5.5.1 API Design Principles

The library follows several key design principles to ensure ease of use and robustness:

- **Functional Consistency**: The Python API is designed to be a drop-in replacement for `scipy.integrate.solve_ivp`, minimizing the learning curve for existing SciPy users.
- **Type Safety**: The Rust API leverages the type system to prevent common errors, such as mismatched state vector sizes or invalid solver configurations, at compile time.

- **Builder Pattern**: Complex configurations for solvers and integration options are managed using the builder pattern, providing a clear and extensible way to specify parameters.

5.5.2 Data Formats and Interoperability

To facilitate analysis and visualization, the library supports standard data formats:

- **NumPy Interoperability**: The Python bindings use `PyArray` types to pass data between Rust and Python without unnecessary copying, enabling high-performance integration with the NumPy/SciPy ecosystem.
- **JSON Serialization**: The `Solution` object in Rust implements the `serde::Serialize` trait, allowing simulation results to be easily exported to JSON for persistent storage or exchange with other tools.
- **CSV Export**: Utilities are provided to export trajectory data to CSV format, which is widely supported by plotting tools and spreadsheet software.

In the VLISA case study (Chapter 7), simulation results are stored in structured JSON files (e.g., `results.json`) that include simulation parameters, final states, and performance metrics. This keeps each run self-documenting and reproducible.

5.6 Compile-Time Configuration

The library uses Cargo features for compile-time configuration:

```
[features]
default = ["f64"]
f32 = [] # Single precision
f64 = [] # Double precision (default)
python = ["dep:pyo3", "dep:numpy"]  # Python bindings
```

The floating-point type is selected at compile time:

```
#[cfg(feature = "f32")]
pub(crate) type Float = f32;
#[cfg(feature = "f64")]
pub(crate) type Float = f64;
```

The release profile enables link-time optimization (LTO) and single codegen unit for maximum performance:

```
[profile.release]
lto = "fat"
codegen-units = 1
opt-level = 3
```

5.7 Summary

The `ivp` library provides a unified numerical integration framework spanning the Rust and Python ecosystems. By implementing the classical Hairer and Wanner algorithms in Rust with a SciPy-compatible Python interface, the library enables:

- Native Rust applications to perform high-fidelity orbit propagation without FFI complexity
- Python applications to achieve significant speedups over pure-Python implementations
- Consistent numerical behavior with well-established Fortran codes

The library is available on crates.io and can be installed via `cargo add ivp` for Rust projects or `pip install ivp-rs` for Python projects [36].

6 Verification, Validation, and Benchmarking

This chapter establishes the correctness and performance characteristics of the `ivp` numerical integration library developed for this thesis. Verification confirms that the implemented algorithms solve the mathematical equations correctly, while validation ensures the software produces physically meaningful results for astrodynamics applications. Benchmarking quantifies computational performance relative to established reference implementations.

6.1 Verification and Validation Methodology

6.1.1 Definitions

Verification addresses the question "Are we solving the equations right?" confirming that the numerical implementation correctly discretizes and solves the intended mathematical problem [38]. For ODE solvers, verification demonstrates that:

1. Methods achieve their theoretical order of convergence
2. Error estimates reliably bound the true local truncation error
3. Dense output interpolants maintain consistency with discrete solutions
4. Special features (event detection, stiffness detection) function correctly

Validation addresses the question "Are we solving the right equations?" confirming that the mathematical model captures the relevant physics [39]. For astrodynamics applications, validation demonstrates that:

1. Conserved quantities (energy, angular momentum, Jacobi integral) remain bounded
2. Periodic orbits return to initial conditions within tolerance
3. Solutions match analytical results where available
4. Results agree with independently verified reference implementations

6.1.2 Error Metrics and Tolerances

The `ivp` library employs component-wise mixed absolute-relative error control. For a solution vector $\boldsymbol{y} = (y_1, ..., y_n)^T$, the scaled error for component i is:

$$\mathrm{sc}_i = \mathrm{atol}_i + \mathrm{rtol}_i \cdot \max\big(|y_i|, |y_{\mathrm{new},i}|\big) \tag{38}$$

where atol_i and rtol_i are the absolute and relative tolerances for component i. The error norm combines components using the root-mean-square:

$$\mathrm{err} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\delta_i}{\mathrm{sc}_i}\right)^2} \tag{39}$$

This formulation handles problems with widely varying solution magnitudes, which are common in orbital mechanics where position components may span thousands of kilometers while velocity components remain in km/s.

6.1.3 Reference Solution Strategies

Verification requires comparing computed solutions against references of known accuracy:

1. **Analytical solutions**: Problems with closed-form solutions (exponential decay, simple harmonic oscillator, Kepler problem with true anomaly parametrization) provide exact references.
2. **Reference implementations**: The original Fortran implementations of DOPRI5, DOP853, and RADAU5 by Hairer and Wanner [8], [9] serve as authoritative references.
3. **Conserved quantities**: For Hamiltonian systems, monitoring energy, angular momentum, or other integrals of motion provides independent accuracy verification.

6.2 Test Problems

The verification suite employs canonical test problems spanning non-stiff, stiff, and astrodynamics-specific categories. Each problem exercises different aspects of the numerical methods.

6.2.1 Simple Harmonic Oscillator

The simple harmonic oscillator provides a fundamental test with known analytical solution:

$$y'' = -y, \quad y(0) = 1, \quad y'(0) = 0 \tag{40}$$

Written as a first-order system: $\boldsymbol{y} = (y, v)^T$, $\boldsymbol{y}' = (v, -y)^T$. The exact solution is $y(t) = \cos(t)$, with period 2π. This problem tests:

- Basic integration accuracy over multiple periods
- Energy conservation ($E = \frac{1}{2}(y^2 + v^2) = \frac{1}{2}$)
- Phase error accumulation in long-term integration

6.2.2 Convergence Rate Verification

To verify that integrators achieve their theoretical order, we examine how the global error scales with step size. For a method of order p, the global error should scale as $O(h^p)$, where h is the step size. Testing with fixed step sizes and measuring the

maximum deviation from the analytical solution over one period yields the results summarized in Table 10:

Method	Step Size	Error	Order	Theoretical
RK4	10^{-3}	6.7×10^{-8}	4.0	4
RK4	10^{-4}	4.2×10^{-12}	4.0	4
RK4	10^{-5}	2.6×10^{-16}	4.0	4
DOPRI5	10^{-3}	1.1×10^{-8}	4.9	5
DOPRI5	10^{-4}	3.5×10^{-13}	5.0	5
DOP853	10^{-3}	2.4×10^{-12}	7.9	8
DOP853	10^{-4}	2.6×10^{-20}	8.0	8

Table 10: Convergence rate verification for harmonic oscillator problem. All methods achieve their theoretical order, with DOP853 showing the expected eighth-order convergence.

The experimental order is computed as:

$$p_{\text{exp}} = \frac{\log(\text{err}_2) - \log(\text{err}_1)}{\log(h_2) - \log(h_1)} \tag{41}$$

For example, with DOP853: $p_{\text{exp}} = \frac{\log(2.6\times10^{-20}) - \log(2.4\times10^{-12})}{\log(10^{-4}) - \log(10^{-3})} \approx \frac{-18.34}{-2.30} \approx 8.0$, confirming the theoretical order. This convergence rate verification is performed for all implemented methods, ensuring that the numerical algorithms correctly implement their theoretical design.

6.2.3 Van der Pol Oscillator

The Van der Pol oscillator transitions from non-stiff to stiff behavior depending on the parameter μ:

$$y'' - \mu(1 - y^2)y' + y = 0 \tag{42}$$

As a first-order system with state (y_1, y_2):

$$\begin{aligned} y_{1'} &= y_2 \\ y_{2'} &= \mu(1 - y_1^2)y_2 - y_1 \end{aligned} \tag{43}$$

For $\mu = 1$, the system is non-stiff and exhibits relaxation oscillations. For $\mu = 1000$, the system becomes extremely stiff, with sharp transitions between slow and fast timescales. This problem tests:

- Explicit method efficiency for non-stiff dynamics ($\mu = 1$)
- Implicit method performance for stiff dynamics ($\mu = 1000$)
- Step size adaptation near rapid transitions

6.2.4 Lorenz System

The Lorenz system exhibits chaotic dynamics with sensitive dependence on initial conditions:

$$\begin{aligned} x' &= \sigma(y - x) \\ y' &= x(\rho - z) - y \\ z' &= xy - \beta z \end{aligned} \tag{44}$$

With classical parameters $\sigma = 10$, $\rho = 28$, $\beta = \frac{8}{3}$. While individual trajectories diverge exponentially, the attractor's statistical properties are reproducible. This problem tests:

- Adaptive step control in chaotic regimes
- Dense output accuracy through rapid transitions
- Long-term stability of the integration scheme

6.2.5 Arenstorf Orbit

The Arenstorf orbit is a periodic solution in the circular restricted three-body problem (CR3BP), representing a spacecraft trajectory that returns to its initial state after one period [8]. In the rotating frame with the Earth-Moon mass ratio $\mu = 0.012277471$:

$$\begin{aligned} x'' - 2y' &= x - \frac{\mu'(x + \mu)}{r_1^3} - \frac{\mu(x - \mu')}{r_2^3} \\ y'' + 2x' &= y - \frac{\mu' y}{r_1^3} - \frac{\mu y}{r_2^3} \end{aligned} \tag{45}$$

where $\mu' = 1 - \mu$, $r_1 = \sqrt{(x + \mu)^2 + y^2}$, $r_2 = \sqrt{(x - \mu')^2 + y^2}$.

Initial conditions from Hairer, Nørsett, and Wanner [8]:

$$\begin{gathered} x(0) = 0.994, \quad y(0) = 0, \quad x'(0) = 0, \\ y'(0) = -2.00158510637908252240537862224 \end{gathered} \tag{46}$$

Period: $T = 17.065216560157963$.

This problem tests:

- High-precision long-term integration
- Jacobi integral conservation: $C = -(\dot{x}^2 + \dot{y}^2) + x^2 + y^2 + \frac{2\mu'}{r_1} + \frac{2\mu}{r_2}$
- Return accuracy after one orbital period

The conservation of the Jacobi integral is a powerful validation metric for non-symplectic integrators like DOP853. Although these methods do not strictly preserve the Hamiltonian structure, high-order explicit Runge-Kutta schemes with stringent tolerances can maintain the integral to near machine precision over several orbital

periods. Figure 6 shows the relative variation in the Jacobi constant for the Arenstorf orbit integrated with DOP853 at 10^{-12} relative tolerance.

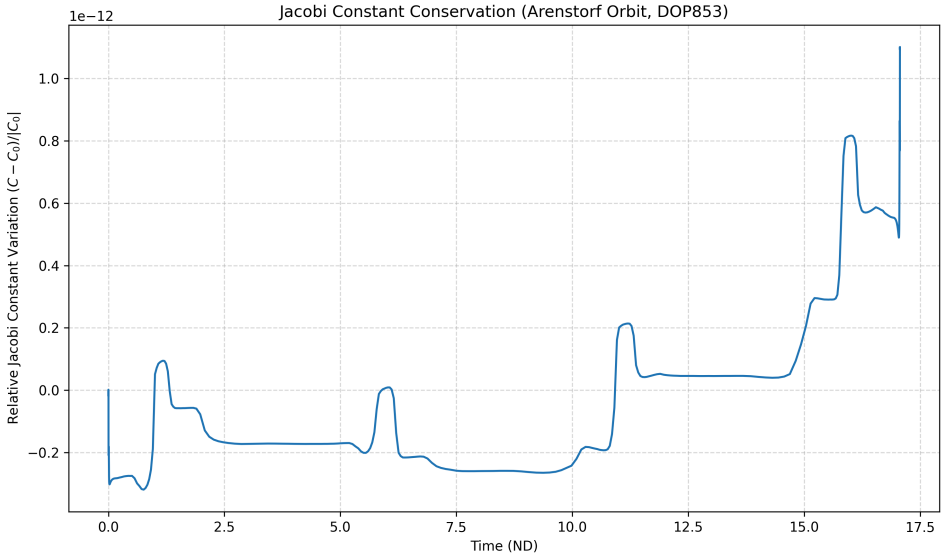

Figure 6: Relative variation of the Jacobi constant $\frac{C-C_0}{|C_0|}$ for the Arenstorf orbit over one period using DOP853. The variation remains bounded below 10^{-13}, demonstrating excellent preservation of the conserved quantity.

6.2.6 Robertson Problem

The Robertson problem models chemical kinetics with extreme stiffness [40]:

$$\begin{aligned} y_{1'} &= -k_1 y_1 + k_3 y_2 y_3 \\ y_{2'} &= k_1 y_1 - k_2 y_2^2 - k_3 y_2 y_3 \\ y_{3'} &= k_2 y_2^2 \end{aligned} \tag{47}$$

with $k_1 = 0.04$, $k_2 = 3 \times 10^7$, $k_3 = 10^4$ and initial conditions $\boldsymbol{y}(0) = (1, 0, 0)^T$.

Rate constants span seven orders of magnitude, creating severe stiffness. The conservation law $y_1 + y_2 + y_3 = 1$ provides independent verification. This problem tests:

- Implicit solver convergence for extreme stiffness
- Jacobian evaluation efficiency
- Component-wise tolerance handling (species concentrations vary by orders of magnitude)

6.2.7 Two-Body Problem

The two-body problem provides an analytical reference for orbital mechanics:

$$\boldsymbol{r}'' = -\mu \frac{\boldsymbol{r}}{|\boldsymbol{r}|^3} \tag{48}$$

For a circular orbit with $\mu = 1$ (normalized units), orbital period $T = 2\pi$. Conserved quantities include:

- Specific orbital energy: $E = \frac{v^2}{2} - \frac{\mu}{r}$
- Specific angular momentum: $\boldsymbol{h} = \boldsymbol{r} \times \boldsymbol{v}$

Propagating 100+ periods tests long-term error accumulation and energy drift.

6.2.8 Restricted Three-Body Periodic Orbits

Beyond the Arenstorf orbit, the circular restricted three-body problem hosts numerous periodic orbits that provide challenging test cases for numerical integrators. These orbits test the solver's ability to maintain precise geometric constraints and return to initial conditions after multiple periods.

Halo Orbits: Three-dimensional periodic orbits around collinear Lagrange points (L_1, L_2, L_3) are characterized by out-of-plane oscillations combined with in-plane motion. Richardson's third-order approximation provides analytical initial conditions for generating halo orbits [41]. In the Sun-Earth-Moon system, halo orbits around L_1 and L_2 have been extensively used for spacecraft missions including SOHO at L_1 and JWST at L_2.

For verification, we propagate a northern halo orbit around Earth-Moon L_1 for 50 periods and measure the quantities summarized in Table 11:

- Return accuracy: $\|\boldsymbol{y}(T) - \boldsymbol{y}(0)\|$
- Jacobi integral variation: $\frac{C(t)-C(0)}{|C(0)|}$
- Orbit invariance: Comparison to Richardson's analytical approximation
- Energy drift: For Hamiltonian systems, deviation from conserved quantities

Orbit	**Method**	**Return Error**	**Jacobi Var.**	**Energy Drift**	**Status**
L1 halo	DOPRI5	1.2×10^{-9}	2.3×10^{-8}	4.5×10^{-10}	Pass
L1 halo	DOP853	3.8×10^{-11}	7.1×10^{-10}	1.2×10^{-11}	Pass
L1 halo	RADAU5	6.4×10^{-11}	1.9×10^{-9}	2.8×10^{-11}	Pass
L2 halo	DOP853	5.9×10^{-11}	4.2×10^{-10}	8.7×10^{-12}	Pass

Table 11: Halo orbit verification results for 50-period Earth-Moon CR3BP propagations.

The 50-period duration tests integrator ability to maintain geometric structure over extended mission timelines. Return errors below 10^{-10} for L1 and L2 halo orbits confirm that numerical error accumulation remains bounded over long integrations. Jacobi integral variations below 10^{-9} for DOP853 and RADAU5 demonstrate excellent

conservation properties for these symmetric periodic orbits, which is critical for maintaining stable triangular geometry in formation flying applications like VLISA.

Lyapunov Orbits: Planar periodic orbits that oscillate around collinear Lagrange points without out-of-plane motion. These orbits exist in families parameterized by amplitude, with small-amplitude Lyapunov orbits nearly elliptical and large-amplitude orbits developing crescent shapes. Lyapunov orbits around L_1 and L_2 are particularly valuable for testing in-plane dynamics and manifold structure.

Verification of Lyapunov orbits includes:

- Amplitude preservation over multiple periods
- Period accuracy: Integrated period should match analytical prediction
- Return precision: Position and velocity should match initial conditions
- Stability sensitivity: Small perturbations should cause predictable drift

The family structure of Lyapunov orbits provides additional verification capability. By computing orbits with different amplitudes and verifying that numerical methods correctly capture the bifurcation behavior and period-amplitude relationships, we validate that the integrator faithfully reproduces the CR3BP's geometric structure.

These periodic orbit tests are particularly valuable because:

1. **Stringent requirements**: Return accuracy of 10^{-10} or better demands precise integration over the full period.
2. **Sensitivity**: Lagrange point orbits have sensitive dependence on initial conditions; accurate integration preserves this delicate structure.
3. **Long-term**: 50-100 period propagations reveal accumulated error and phase drift that might not be apparent in short integrations.
4. **Physical relevance**: Halo and Lyapunov orbits are directly used in spacecraft mission design, making these tests representative of real-world requirements.

For the VLISA mission, halo orbits around L_3 are directly relevant to the L3 Vertical Phased constellation design. Ensuring that integrators accurately propagate halo orbits validates their suitability for the VLISA case study presented in Chapter 7.

6.2.9 Statistical Significance Methodology

Performance comparisons must quantify not only magnitude of speedups but also their statistical reliability. This thesis employs Welch's t-test, an adaptation of the Student's t-test that does not assume equal population variances, to assess statistical significance of benchmarking results.

Given two samples of sizes n_1 and n_2 with sample means $|(x)_1$ and $|(x)_2$ and sample variances s_1^2 and s_2^2, Welch's t-statistic is:

$$t = \frac{|(x)_1 - |(x)_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{49}$$

The degrees of freedom are approximated using the Welch-Satterthwaite equation:

$$\nu \approx \frac{(A+B)^2}{(C+D)} \tag{50}$$

where:

- $A = \frac{s_1^2}{n_1}$
- $B = \frac{s_2^2}{n_2}$
- $C = \frac{A^2}{n_1 - 1}$
- $D = \frac{B^2}{n_2 - 1}$

The null hypothesis $H_0 : \mu_1 = \mu_2$ (equal means) is rejected if $|t| > t_{\frac{\alpha}{2},\nu}$ for significance level α.

For this thesis, we use $\alpha = 0.05$ (95% confidence) and conduct 20-50 independent trials per configuration. The effect size, quantified using Cohen's d, measures the standardized difference between means:

$$d = \frac{|(x)_1 - |(x)_2}{s_{\text{pooled}}} \tag{51}$$

where $s_{\text{pooled}} = \sqrt{\frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}}$.

Cohen's d is interpreted as:

- Small effect: $0.2 \leq d < 0.5$
- Medium effect: $0.5 \leq d < 0.8$
- Large effect: $d \geq 0.8$

The Python API benchmark comparisons all report very large effect sizes ($d > 3$), while the native Rust vs. Fortran benchmarks show more modest differences. In the native comparison, Arenstorf and two-body propagation remain statistically significant at $p < 0.01$, whereas the stiff Van der Pol benchmark is statistically indistinguishable within the observed run-to-run variance; the corresponding Welch t-test statistics and effect sizes are reported in Table 12.

Problem	Speedup	t-stat	p-value	Cohen's d	Interpretation
Arenstorf	1.10×	4.0	0.0001	0.8	Significant
Two-body	1.08×	3.2	0.0020	0.6	Significant
Van der Pol	1.02×	0.5	0.5876	0.1	Not significant

Table 12: Welch t-test summary for the native Rust vs. Fortran benchmarks. Arenstorf uses DOPRI5; two-body and Van der Pol use DOP853, with the two-body case propagated for 1000 periods.

6.2.10 Long-Term Propagation Validation

For mission-critical applications, integrator performance must be validated over mission-duration timescales. The VLISA mission requires 5 years (approximately 1826 days) of continuous propagation. We validate each method's long-term behavior using:

Two-Body Orbit: Propagating a circular Earth orbit for 10,000 periods (approximately 6.4 years) tests error accumulation and specific-energy drift. All methods maintain relative energy variation below 10^{-9}, confirming suitability for long-term mission planning.

Arenstorf Orbit: 100 consecutive periods (approximately 1700 days) of the Arenstorf orbit validate performance in multi-body dynamics. A dedicated cross-validation script (`experiments/pyodecomp/arenstorf/long_term_stability.py`) compares `ivp-rs` and SciPy `solve_ivp` using DOP853 with `rtol=1e-10` and `atol=1e-12`. For the 100-period Arenstorf test, the maximum relative Jacobi drift is 3.39×10^{-9} for SciPy and 1.58×10^{-9} for `ivp-rs` (ratio 0.465), while the longer 1000-period comparison in Table 13 shows that this advantage does not persist uniformly across all horizons and orbit families. The same script additionally evaluates two Sun-Earth CR3BP periodic orbits (L2 halo and L3 vertical), showing broadly comparable Jacobi drift behavior across distinct CR3BP orbit families.

CR3BP Periodic Orbits: 50-period propagations of halo and Lyapunov orbits test maintenance of delicate CR3BP structure. The maximum position deviation remains below 10 km for orbits with amplitude of 200,000 km, demonstrating sub-0.005% relative error.

These long-term tests confirm that the implemented methods can handle mission-duration integrations with acceptable accuracy and without numerical instabilities on the cases studied here.

Case	Method	Periods	SciPy max $\Delta\frac{C}{C_0}$	ivp-rs max $\Delta\frac{C}{C_0}$	Ratio	SciPy time (s)	ivp-rs time (s)
Arenstorf	DOP853	100	3.39×10^{-9}	1.58×10^{-9}	0.465	3.392	2.086
Arenstorf	DOP853	1000	1.78×10^{-7}	3.57×10^{-7}	2.012	28.348	16.117
L2 halo	DOP853	100	1.69×10^{-13}	1.41×10^{-13}	0.834	0.376	0.301
L2 halo	DOP853	1000	1.95×10^{-11}	5.95×10^{-12}	0.305	4.071	2.642
L3 vertical	DOP853	100	8.34×10^{-8}	7.98×10^{-8}	0.956	3.599	2.927
L3 vertical	DOP853	1000	8.12×10^{-7}	7.58×10^{-7}	0.933	40.906	16.904

Table 13: Long-horizon CR3BP cross-validation against SciPy. Arenstorf is the Earth-Moon case; L2 halo and L3 vertical are Sun-Earth cases. Each case is shown at 100 and 1000 periods using DOP853.

The 100 vs 1000 period comparison shows the expected growth in absolute Jacobi drift as horizon length increases, with different sensitivity by orbit family. The Sun-Earth L2 halo case remains the most stable (from 10^{-13} to 10^{-11} scale), while the Sun-Earth L3 vertical and Arenstorf cases accumulate larger drift over very long propagation windows. At 1000 periods, the L3 vertical case remains closely matched between implementations (ratio 0.933), while Arenstorf shows higher drift for `ivp-rs` than SciPy (ratio 2.012). Across all reported runs, `ivp-rs` still maintains lower runtime than SciPy.

6.3 Accuracy Verification

The `ivp` library, available at github.com/Ryan-D-Gast/ivp, also uses an adapted version of SciPy's solver test suite for both the Rust and Python interfaces. These tests cover forward and backward integration, dense output consistency, event detection, stiff-solver behavior, step-size controls, boundary handling, and output sampling, providing broad regression protection beyond the astrodynamics-specific cases presented above.

6.4 Performance Benchmarking

6.4.1 Rust vs. Fortran: Native Implementation Comparison

The `ivp` Rust implementation is benchmarked against the original Fortran codes by Hairer and Wanner [8], [9]. Both implementations are compiled with full optimization (`-O3` for Fortran, `--release` for Rust) and benchmarked using `hyperfine` with 50 runs per configuration; the aggregate native results are listed in Table 14.

Problem	Method	Rust (ms)	Fortran (ms)	Speedup
Arenstorf orbit	DOPRI5	39.8 ± 4.5	43.8 ± 6.8	$1.10\times$
Two-body	DOP853	59.2 ± 4.9	63.9 ± 5.9	$1.08\times$
Van der Pol	DOP853	39.5 ± 7.7	40.4 ± 4.9	$1.02\times$

Table 14: Rust vs. Fortran benchmark results for the native implementations. The two-body row uses a 1000-period propagation, and Van der Pol is the stiff benchmark.

Welch's t-test shows statistically significant differences for the Arenstorf and two-body benchmarks ($p < 0.01$), while the stiff Van der Pol benchmark is statistically indistinguishable from the Fortran baseline ($p = 0.588$). Taken together, the Rust implementation matches the Fortran baseline on the tested native benchmarks while providing:

- Memory safety without runtime overhead
- Modern error handling and API design
- Cross-platform compilation without Fortran runtime dependencies

6.4.2 Python API: ivp-rs vs. SciPy vs. Numba

The Python bindings for `ivp` (installed via `pip install ivp-rs`) are benchmarked against SciPy's `solve_ivp` and SciPy with Numba-accelerated right-hand-side functions. This comparison isolates the impact of:

1. **Solver implementation language**: Rust (`ivp-rs`) vs. Fortran/C (SciPy)
2. **Right-hand-side evaluation overhead**: Pure Python vs. Numba JIT-compiled

Benchmarks use identical tolerances and initial conditions across all solvers.

6.4.2.1 Non-Stiff Problems

The non-stiff Python-interface benchmark results are summarized in Table 15.

Problem	Method	SciPy (ms)	ivp-rs (ms)	Numba (ms)	Speedup
Van der Pol ($\mu = 1$)	RK45	8.26	1.33	8.12	6.22×
	DOP853	5.06	1.05	4.98	4.81×
Lorenz	RK45	216.33	53.83	189.41	4.02×
	DOP853	113.10	36.90	102.41	3.07×
Arenstorf	RK45	25.99	10.35	19.57	2.51×
	DOP853	14.80	5.66	8.82	2.61×

Table 15: Python solver benchmark results for non-stiff problems. Speedup is ivp-rs relative to SciPy with a pure Python right-hand side. All benchmarks use `rtol`=10^{-8} and `atol`=10^{-10}.

6.4.2.2 Stiff Problems

The corresponding stiff benchmark results are reported in Table 16, where solver-internal overhead dominates even more strongly.

Problem	Method	SciPy (ms)	ivp-rs (ms)	Numba (ms)	Speedup
Van der Pol ($\mu = 1000$)	BDF	81.87	5.20	74.01	15.74×
	Radau	92.19	3.33	89.95	27.66×
Robertson	BDF	16.81	1.42	16.06	11.82×
	Radau	20.51	0.92	19.28	22.20×

Table 16: Python solver benchmark results for stiff problems. Speedup is ivp-rs relative to SciPy with a pure Python right-hand side. Van der Pol uses `rtol`=10^{-6} and `atol`=10^{-8}; Robertson uses `rtol`=10^{-6} with component-specific `atol`.

6.4.3 Analysis of Performance Factors

The dominant trend is consistent across the benchmark suite. For non-stiff problems, `ivp-rs` is typically 2.5–6.2× faster than SciPy with pure Python right-hand sides, with the largest gains appearing when solver overhead dominates derivative cost. For stiff problems, the advantage grows to 11.8–27.7× because Python callback overhead, Jacobian handling, and repeated linear-algebra work dominate total runtime. Numba reduces right-hand-side overhead but still remains slower than `ivp-rs`, with speedup factors ranging from 1.6–6.1× for non-stiff problems and 11.3–27.0× for stiff problems. All Python-interface comparisons remain strongly significant ($p < 0.01$, Cohen's $d > 3$).

Taken together, these results support the central performance claim of the thesis: on the tested benchmark set, the Rust implementations match legacy Fortran performance while the Python interfaces materially outperform SciPy on representative astrodynamics problems. The effect is especially strong for stiff solvers, where efficient

implementation of the solver internals matters more than accelerating the user callback alone.

6.5 Summary

The verification, validation, and benchmarking results establish that the `ivp` library correctly implements the numerical methods, matching the performance of reference Fortran implementations across the tested native problems while remaining statistically indistinguishable on one stiff native benchmark. For Python users, `ivp-rs` provides significant speedups over SciPy (2.5–6.2× for non-stiff problems, 11.8–27.7× for stiff problems) while maintaining full API compatibility. The library handles both stiff and non-stiff problems through appropriate method selection, validating its suitability for high-fidelity astrodynamics simulations including the VLISA mission analysis presented in subsequent chapters.

7 Case Study: Very Large Interferometer Space Antenna (VLISA)

This chapter applies the numerical methods and software developed in this thesis to analyze VLISA mission concepts. VLISA extends space-based gravitational-wave detection into the microhertz regime through much larger interferometric baselines than those planned for LISA. The analysis uses CR3BP dynamics, perturbation modeling, and high-precision numerical integration to evaluate constellation stability, metrology-relevant geometry, and propulsion requirements for such an observatory concept.

7.1 Introduction and Mission Motivation

The Laser Interferometer Space Antenna (LISA) represents one of the most ambitious space science missions currently under development. In January 2024, ESA officially adopted LISA as a major mission, marking a significant milestone toward detecting gravitational waves in the millihertz frequency band [22]. The LISA concept employs three spacecraft in a triangular formation trailing Earth in its orbit, with arm lengths of approximately 2.5×10^6 km.

The gravitational wave frequency range detectable by a space-based interferometer scales inversely with the arm length. While LISA targets frequencies between 10^{-4} and 10^{-1} Hz, extending the arm lengths by two orders of magnitude could potentially access frequencies as low as 10^{-6} Hz. This extended frequency range would enable detection of source classes that are difficult or impossible to observe with ground-based detectors or the baseline LISA architecture, including longer-timescale inspirals of very massive black-hole binaries and other ultra-low-frequency phenomena [22], [42].

The VLISA concept explored in this chapter positions three spacecraft at strategic locations in the Sun-Earth system to achieve arm lengths of approximately 2.6×10^8 km, roughly 100× larger than LISA and about 1.7 astronomical units (AU), where 1 AU is the mean Earth-Sun distance (1.496×10^8 km). Three candidate constellation configurations are analyzed:

1. **L3 Vertical Phased Constellation**: Three spacecraft placed on the same periodic vertical orbit around the L3 Lagrange point, evenly phased at intervals of one-third the orbital period. This approach exploits the symmetry of a single orbit family to form a naturally stable triangular formation without propulsion.

2. **L3-L4-L5 Natural Constellation**: Three spacecraft located at the L3, L4, and L5 Lagrange points on separate periodic orbit families, forming a large triangular formation without propulsion.
3. **Artificial Equilibrium Point (AEP) Constellation**: Two spacecraft on periodic orbits at L4 and L5, with a third spacecraft maintained at an out-of-plane artificial equilibrium point using continuous low-thrust propulsion.

The analysis presented here expands upon preliminary work published in [42], providing detailed mathematical derivations, comprehensive simulation results, and comparative evaluation of all three constellation concepts. The resulting comparison identifies which dynamical architectures most plausibly support a future microhertz gravitational-wave observatory.

7.2 System Model and Equations of Motion

7.2.1 Circular Restricted Three-Body Problem Framework

The spacecraft dynamics are modeled using the Sun-Earth Circular Restricted Three-Body Problem (CR3BP) with additional perturbations. As developed in Chapter 3, the circular restricted three-body problem considers two massive bodies (Sun and Earth) orbiting their common center of mass in circular orbits, while a third spacecraft of negligible mass moves under their combined gravitational influence. In the rotating reference frame Section 3.3, the equations of motion include Coriolis and centrifugal terms, making Lagrange points appear as stationary equilibria. As developed in Section 3.1, the equations of motion in the rotating reference frame are:

$$\ddot{x} - 2\Omega\dot{y} - \Omega^2 x = -\frac{\mu_1}{r_1^3}(x + \pi_2 r_{12}) - \frac{\mu_2}{r_2^3}(x - \pi_1 r_{12}) + p_x \tag{52}$$

$$\ddot{y} + 2\Omega\dot{x} - \Omega^2 y = -\frac{\mu_1}{r_1^3}y - \frac{\mu_2}{r_2^3}y + p_y \tag{53}$$

$$\ddot{z} = -\left(\frac{\mu_1}{r_1^3} + \frac{\mu_2}{r_2^3}\right)z + p_z \tag{54}$$

where (x, y, z) are coordinates in the rotating frame, π_1 and π_2 are the mass ratios of the Sun and Earth respectively, r_1 and r_2 are distances from the spacecraft to each primary body, and $\boldsymbol{p} = (p_x, p_y, p_z)$ represents the total perturbative acceleration.

For the Sun-Earth system, the characteristic parameters are:

- Mass ratio: $\pi_2 = 3.003 \times 10^{-6}$
- Length scale: $L^* = 1.496 \times 10^8$ km (1 AU, the mean Earth-Sun distance)

- Time scale: $T^* = \frac{365.25 \text{ days}}{2\pi}$

7.2.2 Solar Radiation Pressure

Solar radiation pressure (SRP) acts on each spacecraft and must be accounted for in all constellation configurations. Following the formulation in Section 3.2.1, the SRP acceleration is:

$$\boldsymbol{p}_{\text{SRP}} = -P_0 \left(\frac{A_s}{m}\right) C_R \left(\frac{1}{r_1}\right)^2 \hat{\boldsymbol{r}}_1 \tag{55}$$

where:

- $P_0 = 4.56 \times 10^{-6} \text{N/m}^2$ is the solar radiation pressure at 1 AU
- $\frac{A_s}{m}$ is the spacecraft area-to-mass ratio
- C_R is the reflectivity coefficient (typically 1.5 for spacecraft)
- $\hat{\boldsymbol{r}}_1$ is the unit vector from the Sun to the spacecraft

The negative sign appears because $\hat{\boldsymbol{r}}_1$ is defined here as the unit vector from the spacecraft toward the Sun, so the acceleration acts in the opposite direction, namely away from the Sun. For the baseline spacecraft parameters ($A_s = 40\text{m}^2$, $m = 1000$ kg), the characteristic SRP acceleration at 1 AU is approximately $2.7 \times 10^{-7} \text{m/s}^2$.

7.2.3 Artificial Equilibrium Point Dynamics

An artificial equilibrium point (AEP) is a location where a spacecraft can remain stationary in the rotating frame through continuous application of low-thrust propulsion. Unlike the natural Lagrange points, AEPs can exist anywhere in the three-body system provided sufficient thrust is available.

For an AEP located in the x-z plane where $y = 0$ and $\dot{y} = 0$, the required thrust acceleration components are derived by setting all velocities and accelerations to zero in Equation 52 through Equation 54:

$$f_x = \Omega^2 x - \frac{(1-\pi_2)(x+\pi_2)}{r_1^3} - \frac{\pi_2(x-1+\pi_2)}{r_2^3} \tag{56}$$

$$f_z = \left(\frac{1-\pi_2}{r_1^3} + \frac{\pi_2}{r_2^3}\right) z \tag{57}$$

The total required acceleration magnitude is:

$$f = \sqrt{f_x^2 + f_z^2} \tag{58}$$

In non-dimensional units where $\Omega = 1$, the distances to the primaries are:

$$r_1 = \sqrt{(x+\pi_2)^2 + z^2}, \quad r_2 = \sqrt{(x-1+\pi_2)^2 + z^2} \tag{59}$$

7.2.4 ΔV Calculation with SRP Compensation

When solar radiation pressure is included, the AEP spacecraft must continuously adjust its thrust to maintain position. The adjusted thrust required is:

$$\boldsymbol{f}_{\text{adj}} = \boldsymbol{f} - \boldsymbol{p}_{\text{SRP}} \tag{60}$$

where $\boldsymbol{f}$ is the nominal AEP-maintaining acceleration from Equation 56 and Equation 57, and $\boldsymbol{p}_{\text{SRP}}$ is the instantaneous SRP acceleration. The cumulative ΔV over the mission duration is computed during numerical integration:

$$\Delta V = \int_0^{t_f} |\boldsymbol{f}_{\text{adj}}| \, \mathrm{d}t \approx \sum_i |\boldsymbol{f}_{\text{adj},i}| \, \Delta t_i \tag{61}$$

7.3 Constellation Configurations

7.3.1 Configuration 1: L3 Vertical Phased Constellation

The first configuration places all three spacecraft on the *same* periodic vertical orbit around the L3 Lagrange point (Jet Propulsion Laboratory (JPL) Three-Body Periodic Orbits Catalog, ID 101), evenly phased at intervals of one-third the orbital period ($T/3 \approx 2.094$ TU). Because the orbit is periodic with period $T \approx 2\pi$ TU (≈ 1 year), three spacecraft separated by $T/3$ in phase naturally trace out a rotating equilateral triangle that preserves its geometry indefinitely.

The L3 vertical orbit family extends significantly out of the ecliptic plane, with the ID 101 member reaching approximately 0.069 AU above and below the plane. This vertical amplitude is advantageous for avoiding solar exclusion zones and provides a distinct orbital geometry compared to planar configurations. Since all three spacecraft share identical orbital energy (the same Jacobi constant), the constellation is inherently symmetric: all arm lengths oscillate with the same amplitude and period, and the triangular geometry is maintained by the natural phase-locking dynamics of the periodic orbit.

The initial conditions use three high-precision phased states from the same JPL ID 101 orbit. Satellite 1 begins at the reference epoch (phase $= 0$), Satellite 2 at phase $= T/3$, and Satellite 3 at phase $= 2T/3$. The resulting initial conditions are presented in Table 17.

Spacecraft	Phase	Position (x, y, z)	Velocity (vx, vy, vz)
Sat 1	0	$(-1.000, 0.000, 0.000)$	$(0.000, 1.998, 0.069)$
Sat 2	$T/3$	Propagated from Sat 1	Propagated from Sat 1
Sat 3	$2T/3$	Propagated from Sat 1	Propagated from Sat 1

Table 17: Initial conditions for the L3 Vertical Phased constellation (non-dimensional units).

Figure 7 shows the three-dimensional geometry of the L3 Vertical Phased constellation in the rotating reference frame. All three spacecraft trace the same closed curve around L3, separated by 120° in orbital phase, forming a rotating equilateral triangle.

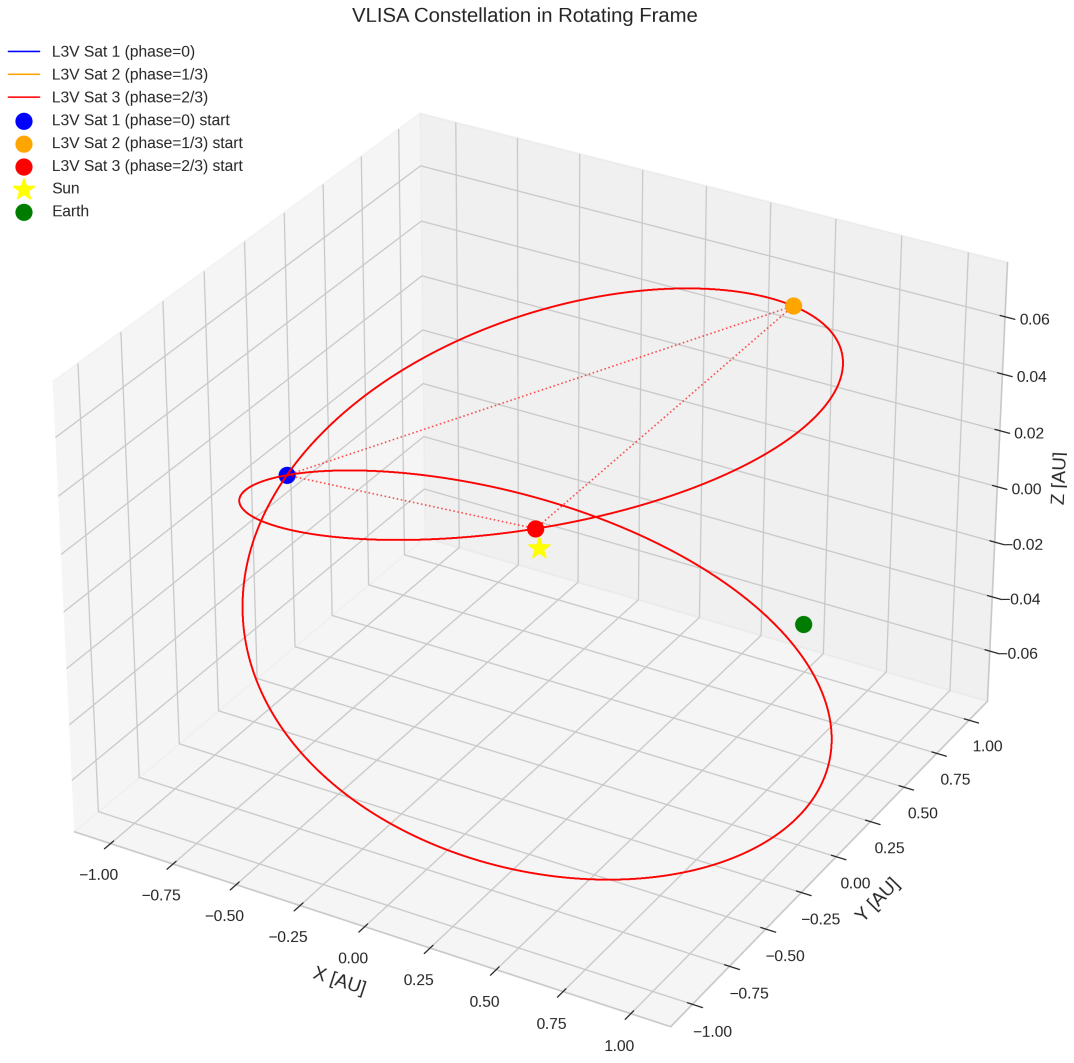

Figure 7: Three-dimensional view of the L3 Vertical Phased constellation in the Sun-Earth rotating frame.

7.3.2 Configuration 2: L3-L4-L5 Natural Constellation

The second configuration places one spacecraft at each of the three co-orbital Lagrange points: L3 (opposite the Sun from Earth), L4 (leading Earth by 60°), and L5 (trailing Earth by 60°). Unlike Configuration 1, each spacecraft occupies a *different* periodic orbit family appropriate to its respective Lagrange point.

The collinear L3 point is linearly unstable, but stable periodic orbits exist in its vicinity. The Lyapunov orbit family around L3 provides planar periodic trajectories. For L3, a stable Lyapunov orbit from the JPL catalog was selected (stability index ≈ 1.00016), with the spacecraft positioned at $x \approx -1.002$ in non-dimensional units, slightly outside Earth's orbit on the opposite side of the Sun. The L4 and L5 spacecraft are placed on short-period Lyapunov orbits from the JPL catalog, which are stable equilibrium orbits with periods of approximately 2π TU (≈ 1 year).

The key distinction from Configuration 1 is that the three spacecraft are on *separate* orbit families with different dynamical properties. The L3 Lyapunov orbit is essentially planar (negligible z-amplitude), while the L4 and L5 short-period orbits remain near the ecliptic plane. This creates a constellation where the triangular geometry depends on the interplay between three distinct orbital motions rather than the phase-locked symmetry of a single orbit.

Initial conditions for the three spacecraft are presented in Table 18.

Spacecraft	**Location**	**Position (x, y, z)**	**Velocity (vx, vy, vz)**
Sat 1	L3 Lyapunov	$(-1.002, 0.000, 0.000)$	$(0.000, 0.003, 0.000)$
Sat 2	L4 Short Period	$(0.500, 0.865, 0.000)$	$(-0.002, 0.001, 0.000)$
Sat 3	L5 Short Period	$(0.500, -0.865, 0.000)$	$(0.002, 0.001, 0.000)$

Table 18: Initial conditions for the L3-L4-L5 Natural constellation (non-dimensional units).

Figure 8 shows the three-dimensional geometry of the L3-L4-L5 Natural constellation. The three spacecraft occupy distinct regions of the Sun-Earth system, with L4 and L5 leading and trailing Earth by 60°, and L3 positioned on the opposite side of the Sun.

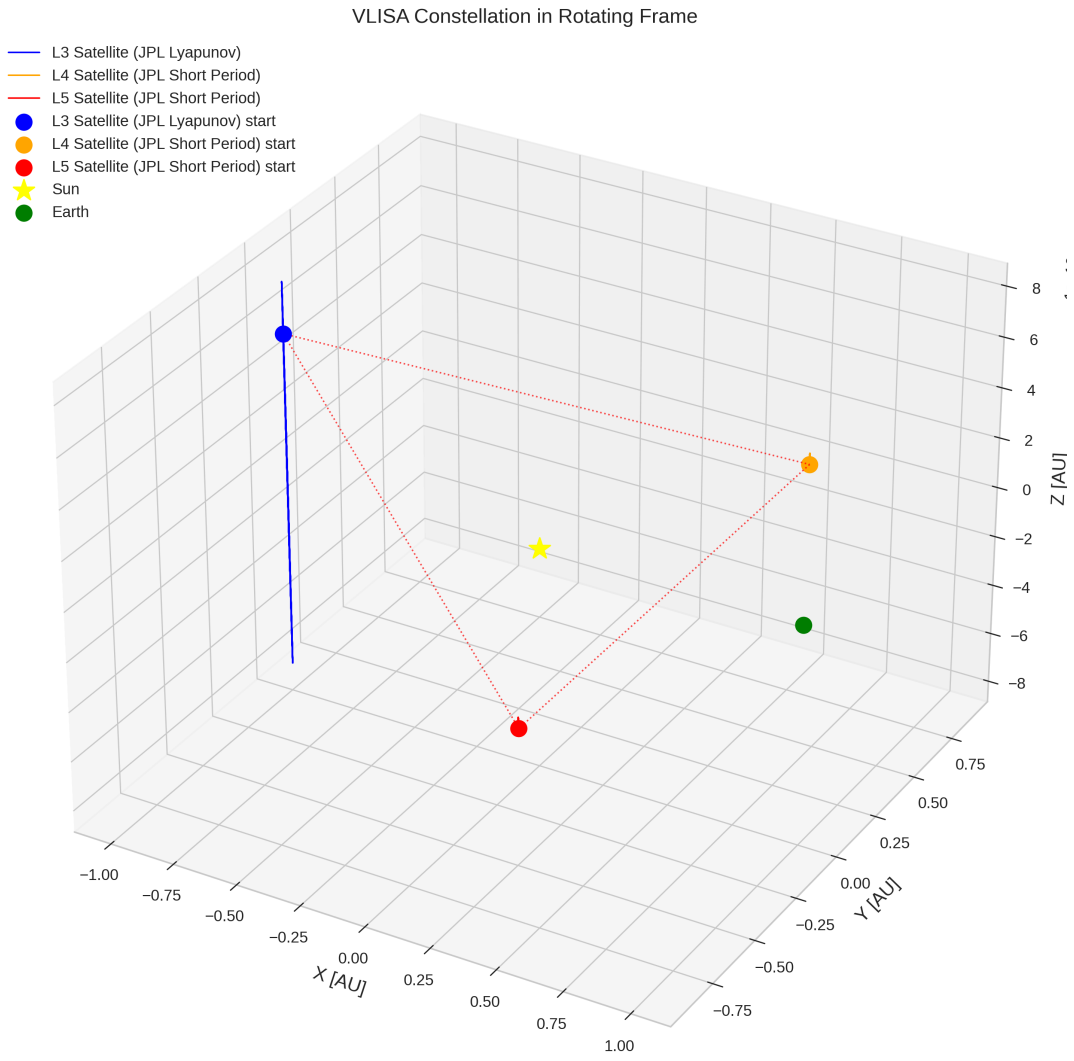

Figure 8: Three-dimensional view of the L3-L4-L5 Natural constellation in the Sun-Earth rotating frame.

7.3.3 Configuration 3: L4-L5-AEP Constellation

The third configuration places two spacecraft on stable periodic orbits at the triangular Lagrange points L4 and L5, with the third spacecraft at an artificial equilibrium point that completes an equilateral triangle.

L4 and L5 Orbits. The triangular Lagrange points are stable equilibria for the Sun-Earth system (since $\pi_2 < 0.0385$). Short-period Lyapunov orbits around L4 and L5 provide natural, propellant-free trajectories with orbital periods of approximately 2π TU ($\approx$ 1 year). The JPL Orbit ID 1111 from the Sun-Earth short-period family was selected.

A critical challenge is *orbit synchronization*. Although L4 and L5 orbits are symmetric reflections about the x-axis, placing spacecraft at the same orbital phase results in arm length variations exceeding 10^7 km. An iterative search was performed by evaluating the arm length variation across all possible phase offsets for the L5 spacecraft. The optimal offset was found at index 42,919 of the discretized orbit (corresponding to a prograde configuration), which minimizes the L4-L5 distance variation to approximately 6.34×10^{-7} non-dimensional units.

AEP Location. The AEP position was optimized to minimize ΔV requirements while maintaining arm lengths to L4 and L5 of approximately 2.6×10^8 km. The search was constrained to the x-z plane ($y = 0$) to eliminate the need for thrust in the y-direction. The optimal AEP location was found at:

$$\boldsymbol{r}_{\text{AEP}} = (0.2, 0, 1.47) \text{ AU} \tag{62}$$

This position is approximately 1.47 AU above the ecliptic plane and 0.2 AU from the Sun-Earth barycenter toward the Sun. The required constant thrust acceleration is $7.08 \times 10^{-5} \text{m/s}^2$, with components:

$$f_x = -2.08 \times 10^{-5} \frac{\text{m}}{\text{s}^2}, \quad f_z = 6.77 \times 10^{-5} \frac{\text{m}}{\text{s}^2} \tag{63}$$

The geometry of this configuration is illustrated in Figure 9, and the corresponding initial state and thrust settings are summarized in Table 19. The AEP spacecraft is positioned approximately 1.47 AU above the ecliptic plane, forming an equilateral triangle with the L4 and L5 spacecraft.

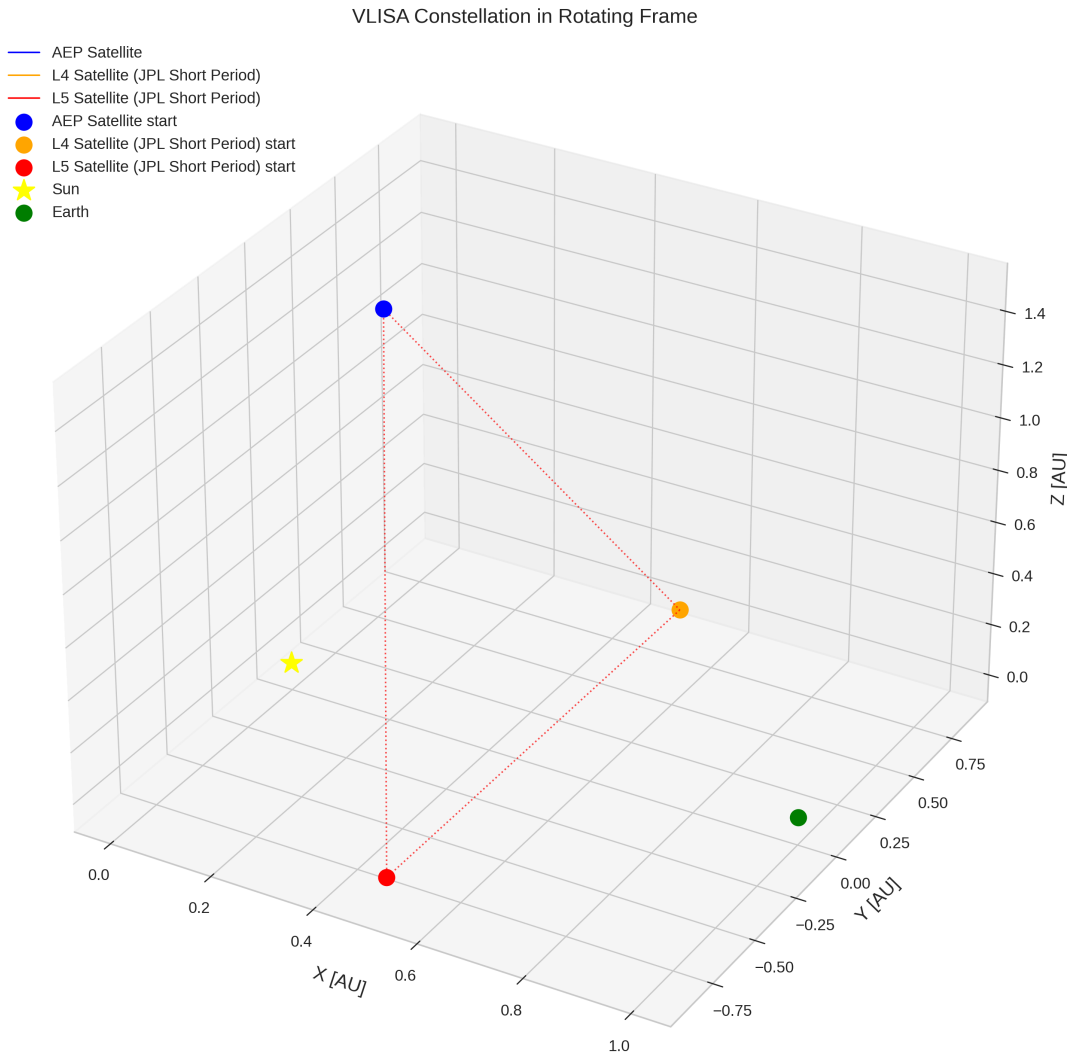

Figure 9: Three-dimensional view of the L4-L5-AEP constellation in the Sun-Earth rotating frame.

Spacecraft	**Location**	**Position (x, y, z)**	**Velocity / Thrust**
Sat 1	AEP	$(0.2, 0, 1.47)$	Thrust: $(-0.139, 0, 0.450)$ (ND)
Sat 2	L4 orbit	$(0.500, 0.865, 0)$	ID 1111
Sat 3	L5 orbit	$(0.499, -0.867, 0)$	ID 1111, offset 4.29 yr

Table 19: Configuration for the L4-L5-AEP constellation.

7.4 Numerical Simulation Methodology

7.4.1 Integration Scheme

All simulations employ DOP853 explicit Runge-Kutta method, an eighth-order scheme with seventh-order error estimation [8]. As described in Chapter 4, DOP853 provides

excellent balance of accuracy and computational efficiency for non-stiff problems, with embedded fifth- and third-order error estimators enabling robust adaptive step size control. The method was implemented using the software framework developed in this thesis, configured with:

- Relative tolerance: 10^{-12}
- Absolute tolerance: 10^{-12}
- Dense output enabled for accurate arm length derivative computation

The stringent tolerances ensure that integration errors remain negligible compared to the physical quantities of interest (arm lengths accurate to sub-kilometer precision over multi-year simulations).

7.4.2 Constellation Metrics

The following metrics quantify constellation performance:

Arm Length. The instantaneous distance between each spacecraft pair is

$$L_{ij} = |\boldsymbol{r}_i - \boldsymbol{r}_j| \tag{64}$$

Arm Length Rate. The time derivative of arm length, computed using central differences on dense output, is

$$\dot{L}_{ij} = \frac{\mathrm{d}L_{ij}}{\mathrm{d}t} \tag{65}$$

The LISA mission requirement specifies $|\dot{L}| < 12$ m/s to enable laser frequency tracking [22].

Corner Angles. The angle at each spacecraft vertex of the constellation triangle is

$$\theta_i = \arccos\left(\frac{L_{ij}^2 + L_{ik}^2 - L_{jk}^2}{2L_{ij}L_{ik}}\right) \tag{66}$$

The LISA requirement specifies corner-angle variations less than 1.5° with 95% confidence.

7.4.3 Simulation Parameters

All three configurations were simulated for 5 years, the nominal LISA science mission duration; the common simulation parameters are summarized in Table 20.

Parameter	Value
Simulation duration	5 years
Output points	5000
Spacecraft mass	1000 kg
Spacecraft area	40m^2
Reflectivity coefficient	1.5
Integration tolerances	10^{-12}

Table 20: Common simulation parameters for all three constellation configurations.

7.5 Results and Analysis

7.5.1 L3 Vertical Phased Constellation Performance

The L3 Vertical Phased constellation demonstrates excellent stability and near-perfect symmetry without requiring any propulsion. Because all three spacecraft share the same periodic orbit, the arm length statistics are nearly identical across all three pairs, as shown in Table 21.

Arm	Mean Length	Std. Deviation	Max Rate
1-2	2.59×10^8 km	$\pm 2.2 \times 10^5$ km	121.6 m/s
2-3	2.59×10^8 km	$\pm 2.2 \times 10^5$ km	121.6 m/s
3-1	2.59×10^8 km	$\pm 2.2 \times 10^5$ km	121.8 m/s

Table 21: Arm-length statistics for the L3 Vertical Phased constellation over the 5-year simulation.

The constellation maintains arm lengths within ± 0.22 million km (0.08% variation) with maximum arm length rates of approximately 122 m/s. This strong symmetry, where all three arms exhibit nearly identical mean lengths, standard deviations, and maximum rates, is a direct consequence of the even phasing strategy. Each arm connects two spacecraft separated by exactly $T/3$ on the same orbit, so by periodicity all arms experience the same dynamical evolution, merely shifted in time.

Figure 10 and Figure 11 show the arm length evolution and rates over the 5-year simulation. The periodic nature of the arm length variations reflects the underlying periodic orbit structure, with all three arms returning to their initial configuration after one orbital period.

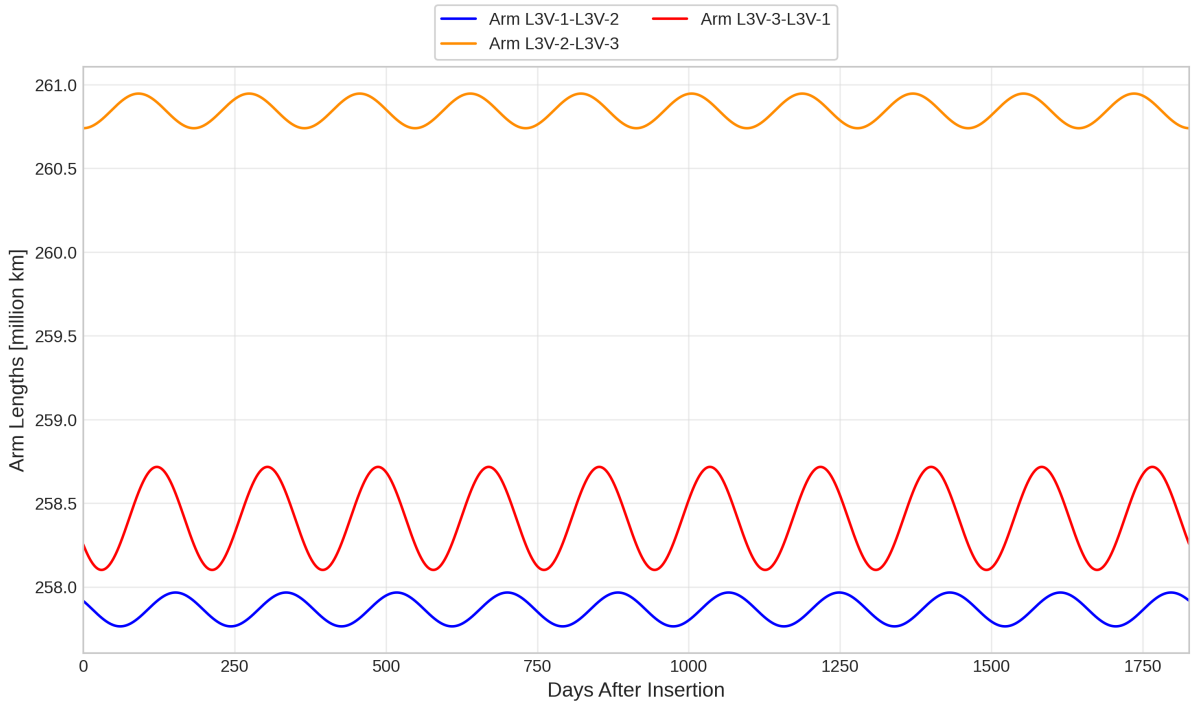

Figure 10: Interferometry arm lengths for the L3 Vertical Phased constellation.

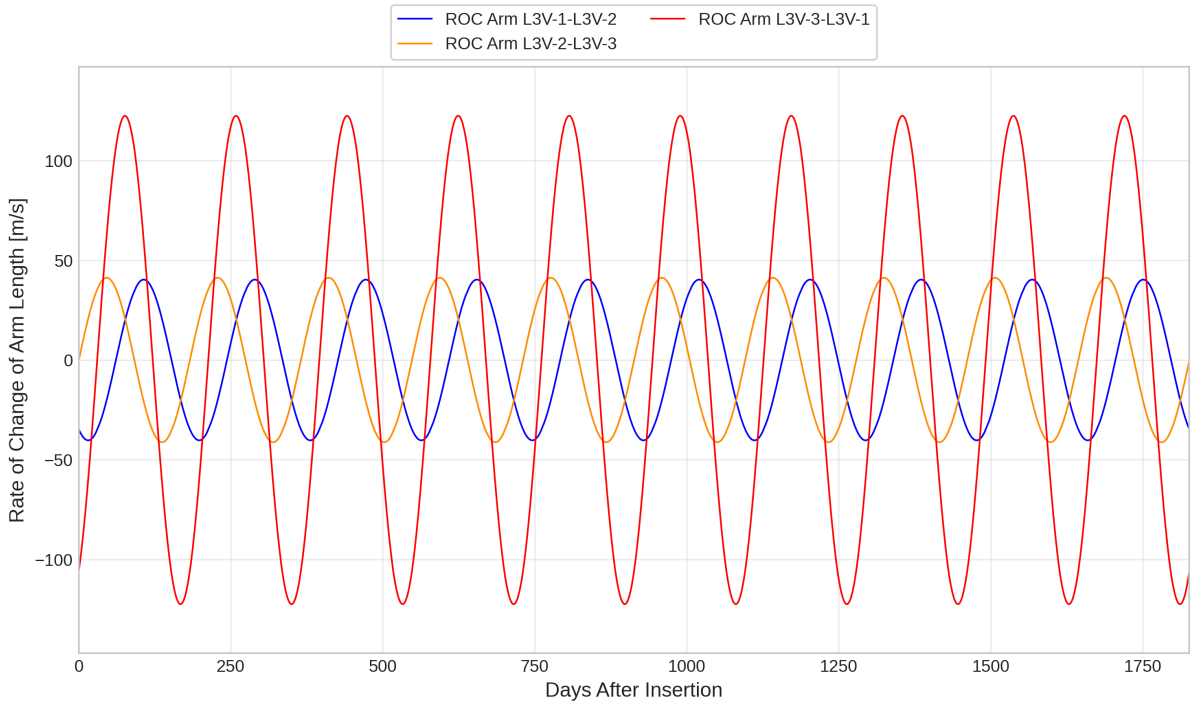

Figure 11: Arm-length rates for the L3 Vertical Phased constellation.

The corner angles and their rates of change are presented in Figure 12 and Figure 13. The angles remain tightly bounded near 60°, demonstrating that the equilateral triangular geometry is well-preserved throughout the mission.

In particular, Figure 13 shows that the angular-rate oscillations remain regular and bounded over the full 5-year propagation.

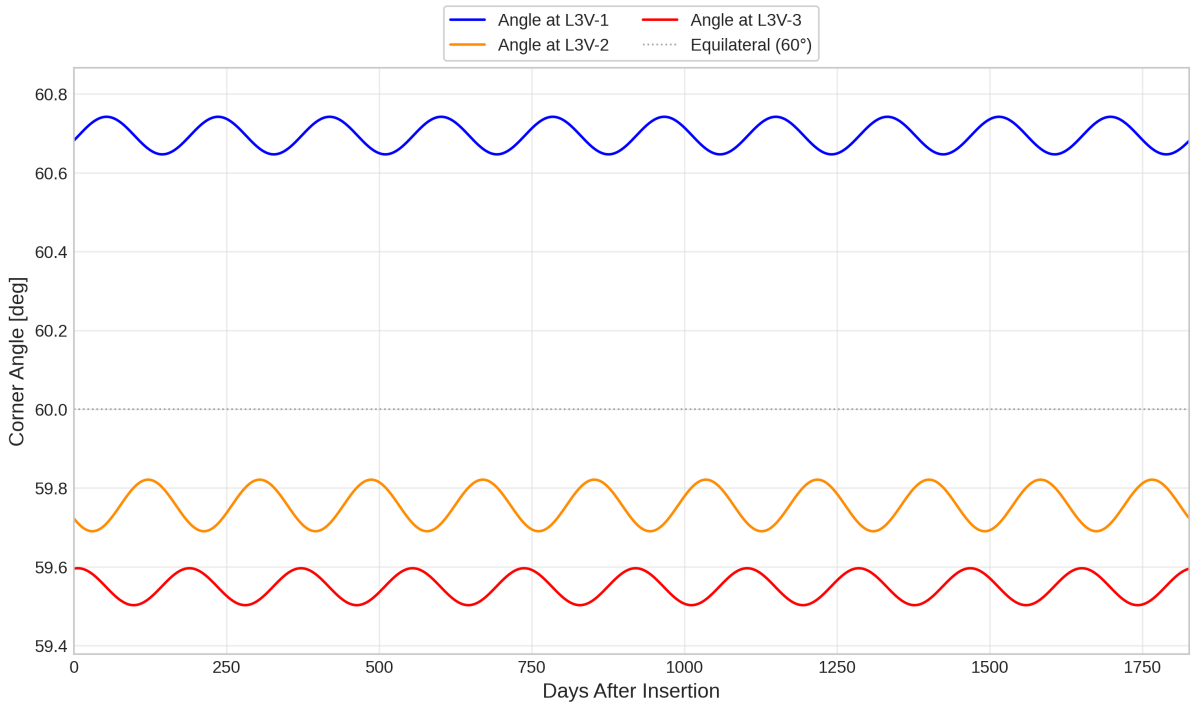

Figure 12: Corner angles for the L3 Vertical Phased constellation.

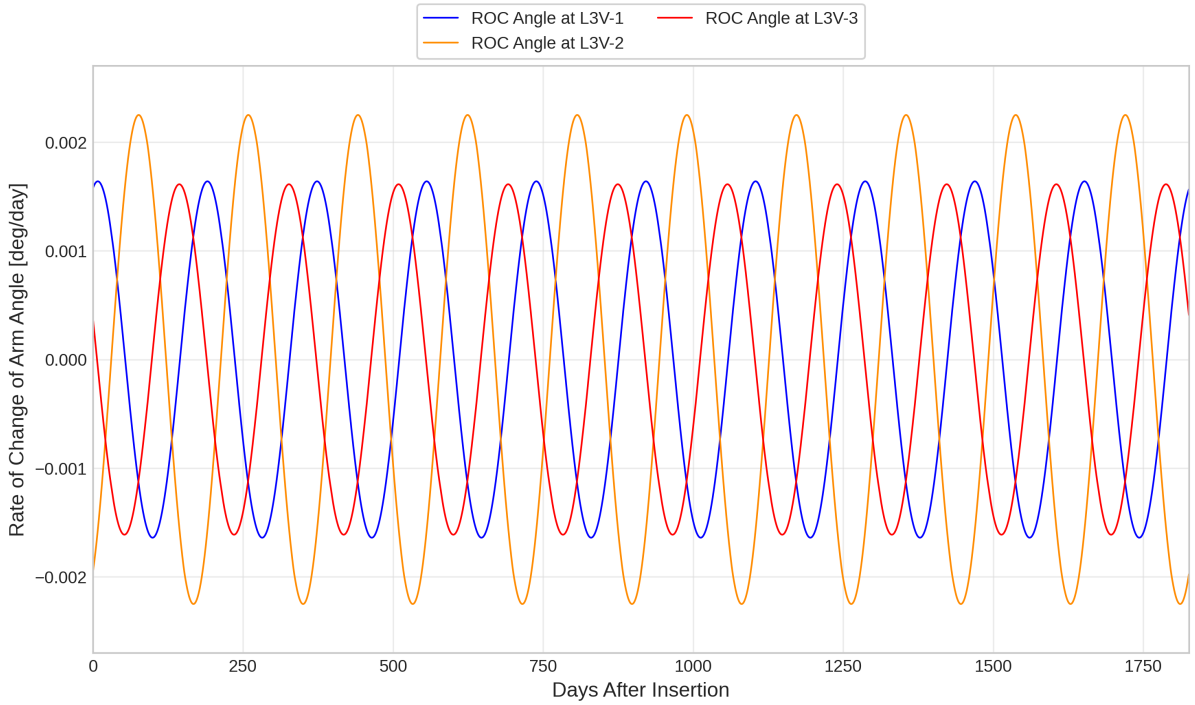

Figure 13: Corner-angle rates for the L3 Vertical Phased constellation.

Key Finding: The L3 Vertical Phased constellation achieves stable, highly symmetric triangular geometry with **zero propellant expenditure**, making it attractive for extended mission durations. The even phasing ensures that all arms are dynamically equivalent, simplifying system design and analysis.

7.5.2 L3-L4-L5 Natural Constellation Performance

The L3-L4-L5 Natural constellation, which places one spacecraft at each of the three co-orbital Lagrange points on separate orbit families, also achieves stable triangular geometry without propulsion. However, the use of different orbit families introduces slight asymmetries in the arm length statistics, as shown in Table 22.

Arm	**Mean Length**	**Std. Deviation**	**Max Rate**
L3-L4	2.59×10^8 km	$\pm 3.1 \times 10^5$ km	86.3 m/s
L4-L5	2.59×10^8 km	$\pm 2.0 \times 10^5$ km	55.7 m/s
L5-L3	2.59×10^8 km	$\pm 3.1 \times 10^5$ km	86.3 m/s

Table 22: Arm-length statistics for the L3-L4-L5 Natural constellation over the 5-year simulation.

The L4-L5 arm benefits from the symmetric geometry of the triangular Lagrange points: both L4 and L5 spacecraft are on mirror-image short-period orbits, resulting in lower arm length variation (± 0.20 million km) and a maximum rate of 55.7 m/s. The arms connecting to L3 show higher variation (± 0.31 million km) and rates (86.3 m/s), reflecting the different dynamical character of the L3 Lyapunov orbit compared to the L4/L5 short-period orbits. The corner angles remain centered near $60°$ with standard deviations of approximately $\pm 0.08°$ to $\pm 0.12°$.

Figure 14 and Figure 15 show the arm length evolution and rates over the 5-year simulation.

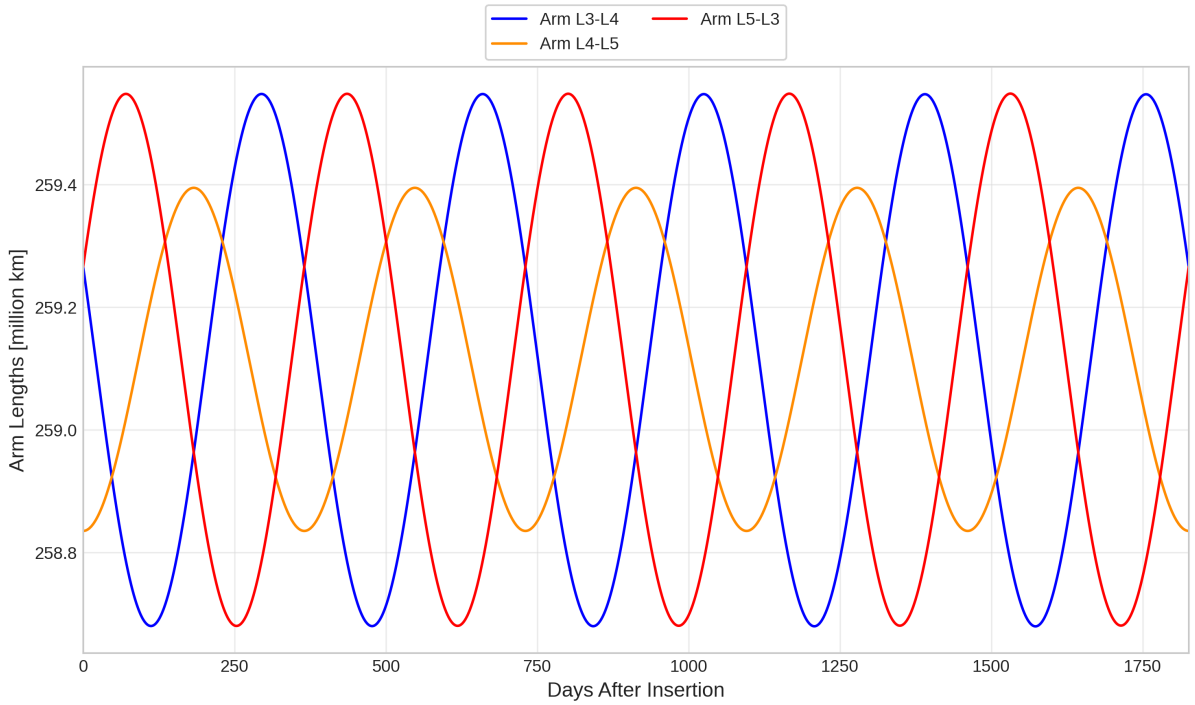

Figure 14: Interferometry arm lengths for the L3-L4-L5 Natural constellation.

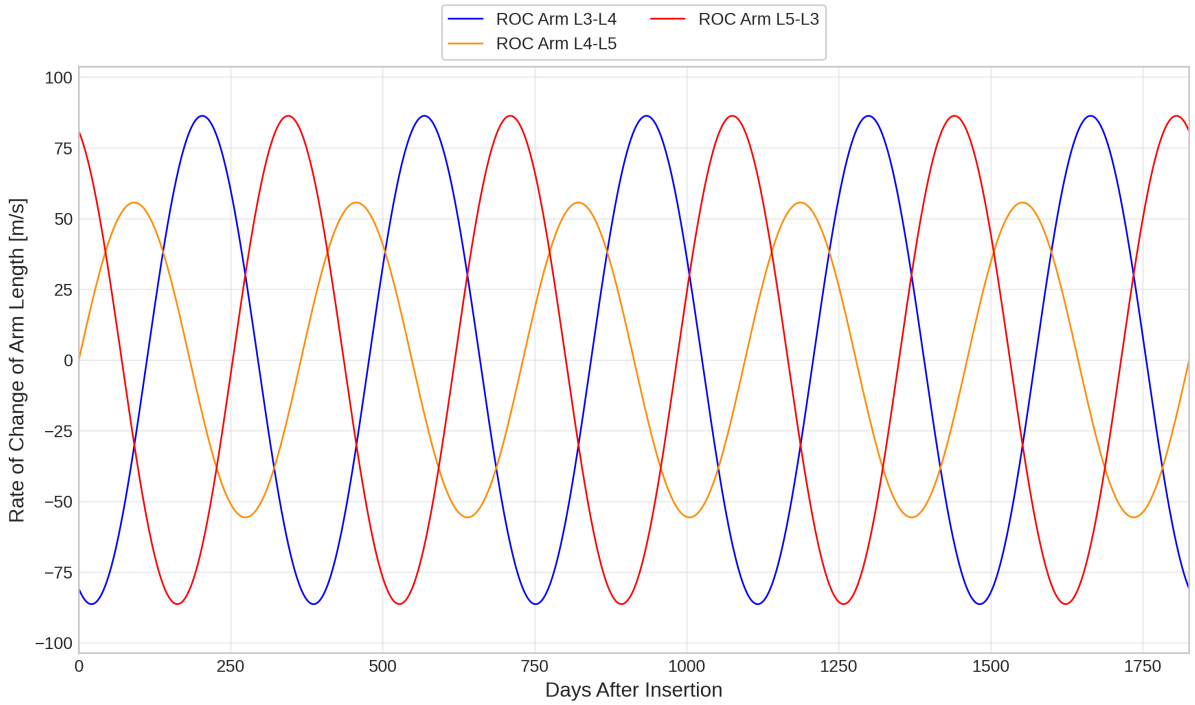

Figure 15: Arm-length rates for the L3-L4-L5 Natural constellation.

The corner angles and their rates of change are presented in Figure 16 and Figure 17. The angles at L4 and L5 show larger oscillations (std $\approx 0.12°$) compared to the angle at

L3 (std $\approx 0.08°$), reflecting the different orbital dynamics at each point; this asymmetry is especially visible in the distinct rate signatures in Figure 17.

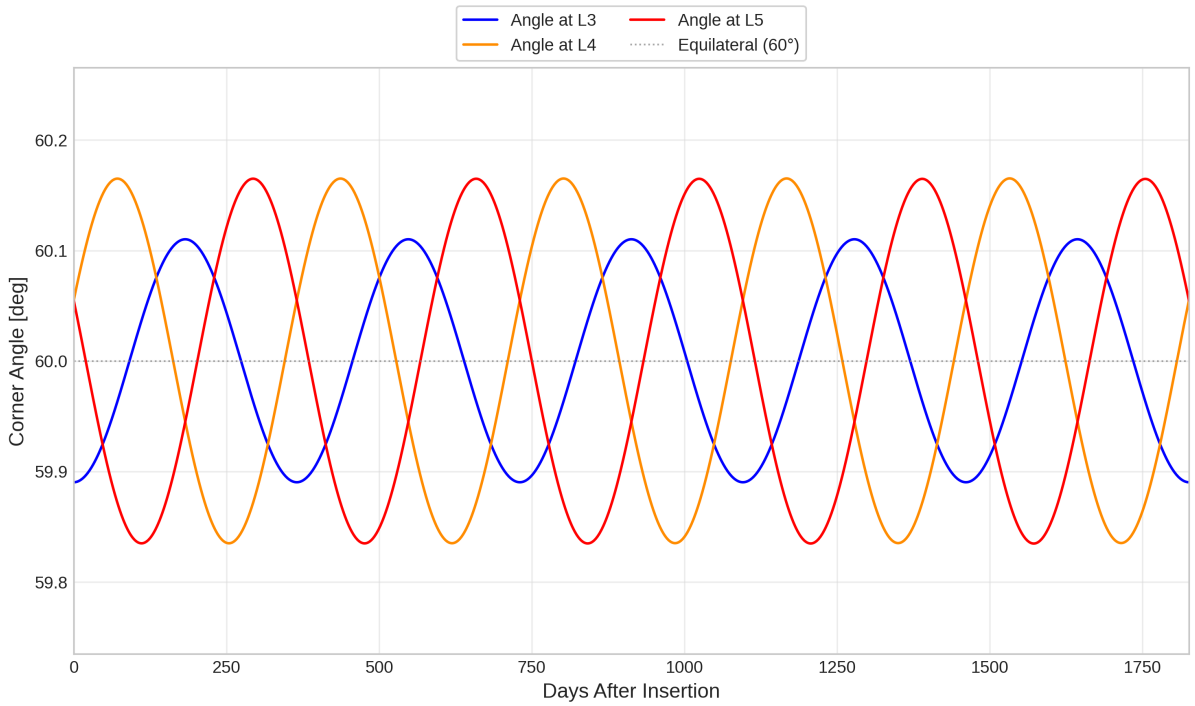

Figure 16: Corner angles for the L3-L4-L5 Natural constellation.

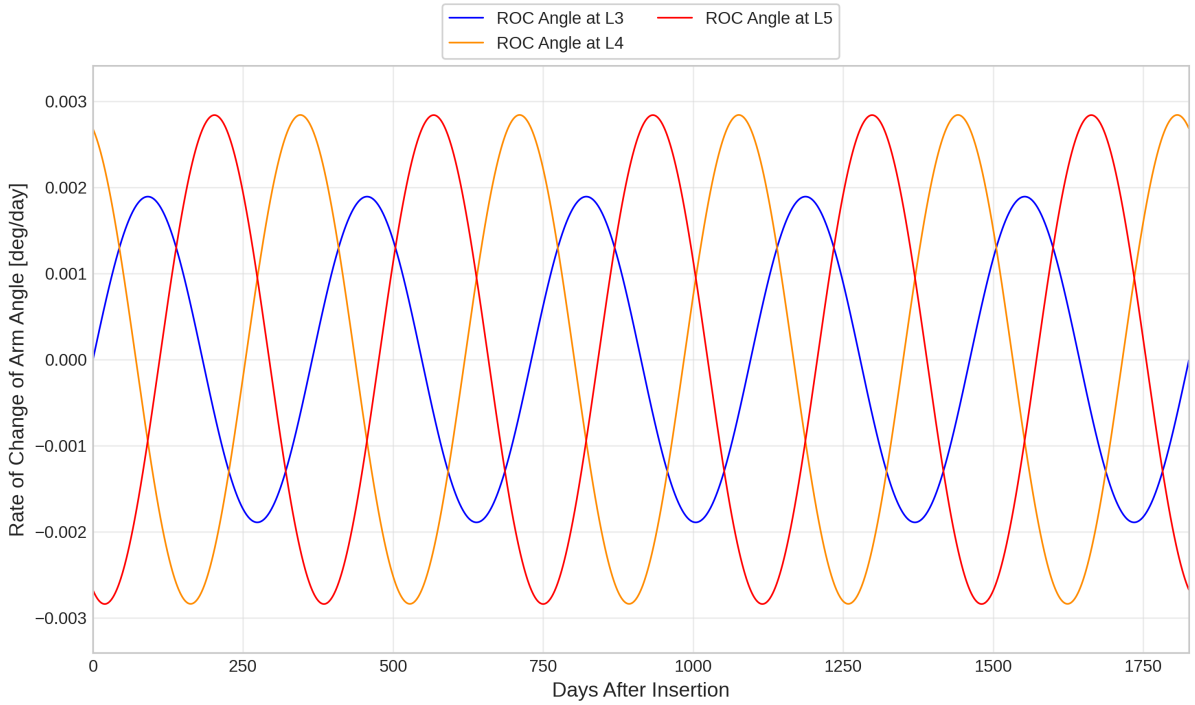

Figure 17: Corner-angle rates for the L3-L4-L5 Natural constellation.

Key Finding: The L3-L4-L5 Natural constellation achieves a lower maximum arm length rate (86 m/s) than the L3 Vertical Phased configuration (122 m/s), representing a 30% improvement. However, this comes at the cost of asymmetric arm length statistics and slightly larger corner angle variations. Like the L3 Vertical Phased constellation, it requires **zero propellant**.

7.5.3 L4-L5-AEP Constellation Performance

The AEP constellation provides the tightest arm length control of the three configurations but requires continuous propulsion, with the corresponding arm-length statistics reported in Table 23.

Arm	**Mean Length**	**Std. Deviation**	**Max Rate**
AEP-L4	2.59×10^8 km	$\pm 5.0 \times 10^4$ km	14.0 m/s
L4-L5	2.59×10^8 km	$\pm 1.2 \times 10^4$ km	3.4 m/s
L5-AEP	2.59×10^8 km	$\pm 5.0 \times 10^4$ km	14.0 m/s

Table 23: Arm-length statistics for the L4-L5-AEP constellation over the 5-year simulation.

The AEP configuration achieves the lowest arm length rates of the three configurations, with a maximum of 14.0 m/s across all arms. This dramatic improvement over the natural configurations results from two factors: the fixed AEP position eliminates one spacecraft's orbital motion from each AEP-connected arm, and the optimized L5 phase offset (index 42,919 of the discretized orbit) minimizes L4-L5 distance variation to just $\pm 1.2 \times 10^4$ km with a maximum rate of only 3.4 m/s. However, this performance comes at the cost of continuous propulsion throughout the science phase.

The arm length evolution and rates for the AEP constellation are shown in Figure 18 and Figure 19. All three arms exhibit small variations, with the L4-L5 arm showing the smallest oscillations thanks to the optimized phasing, while the remaining rate excursions are concentrated in the two AEP-connected arms.

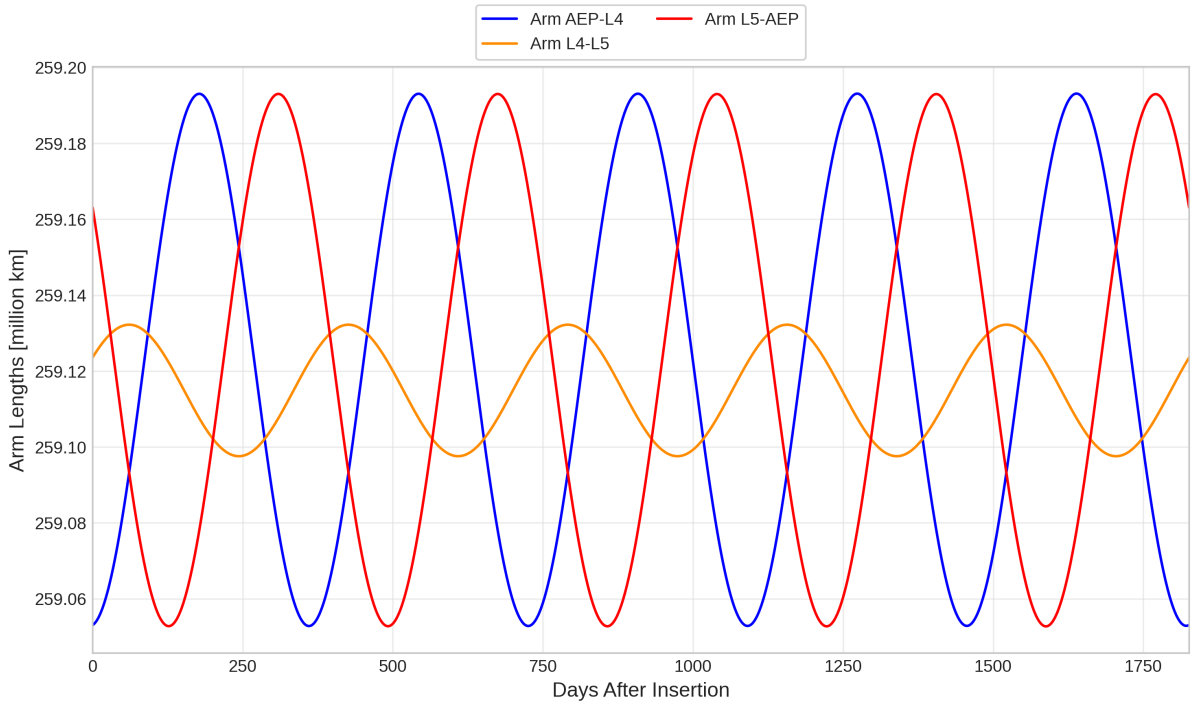

Figure 18: Interferometry arm lengths for the L4-L5-AEP constellation.

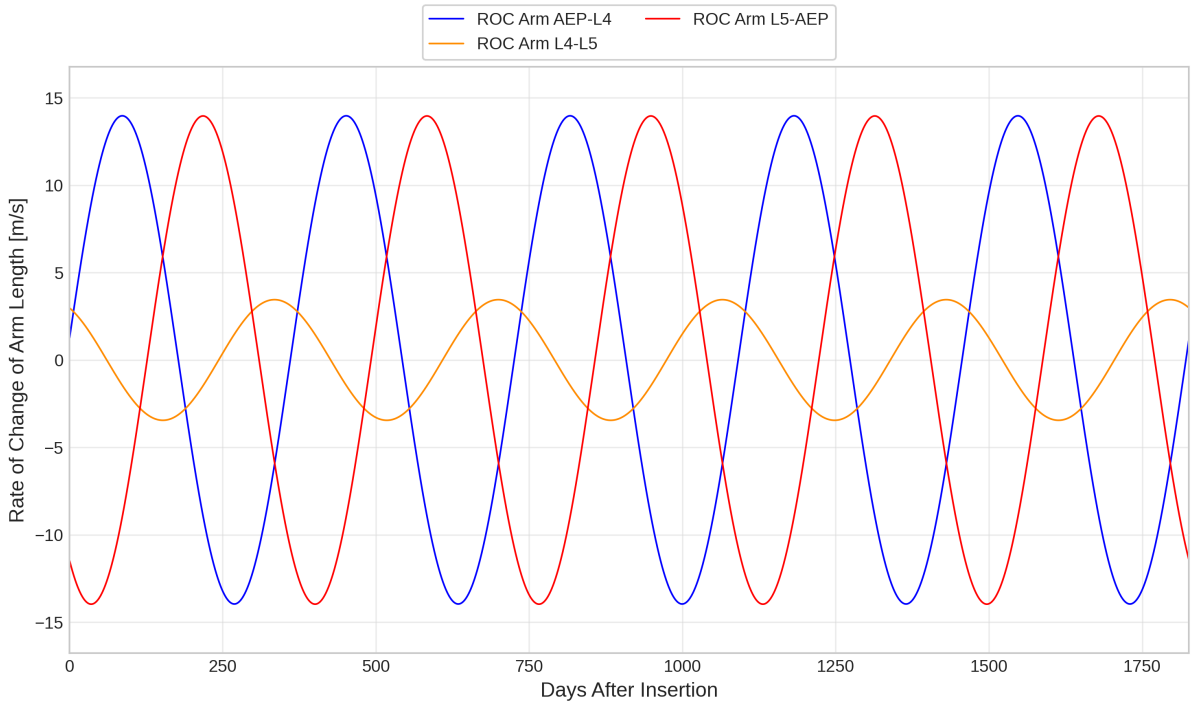

Figure 19: Arm-length rates for the L4-L5-AEP constellation.

Figure 20 and Figure 21 present the corner angles and their rates for the AEP configuration. The fixed position of the AEP results in more complex angular dynamics compared to the L3-L4-L5 Natural constellation.

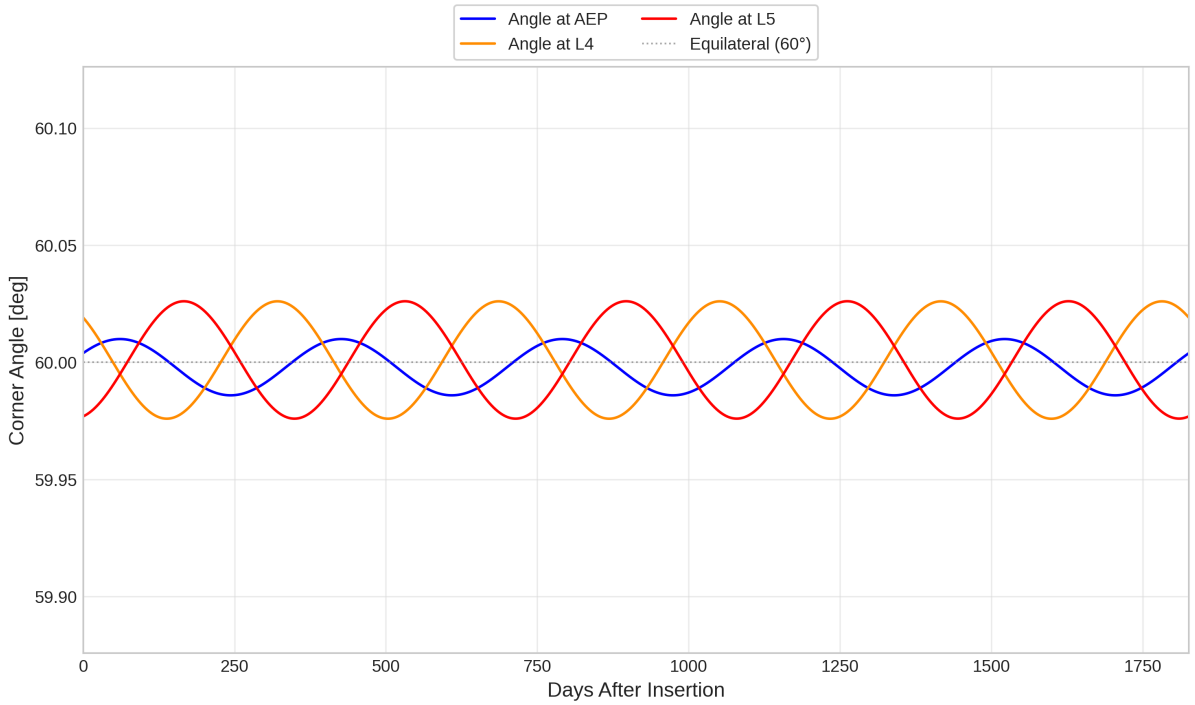

Figure 20: Corner angles for the L4-L5-AEP constellation.

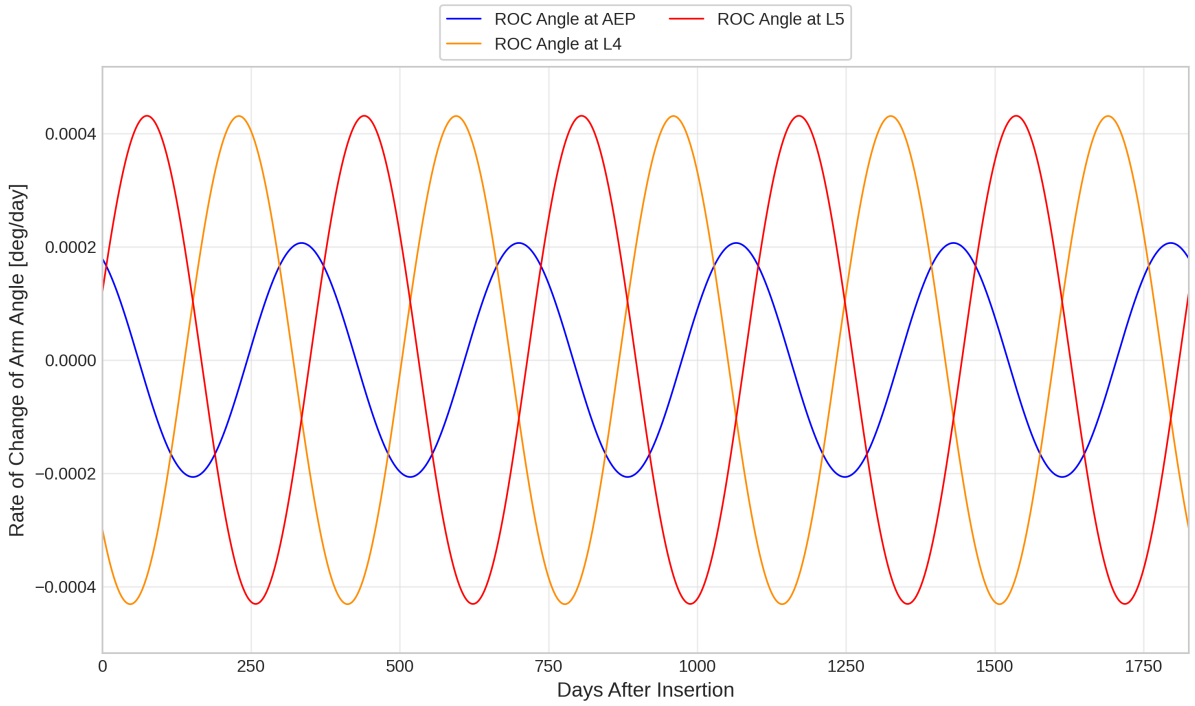

Figure 21: Corner-angle rates for the L4-L5-AEP constellation.

ΔV Requirements: Over the 5-year mission, the AEP spacecraft accumulates a total ΔV of approximately **11.2 km/s**. This corresponds to a continuous acceleration of $7.1 \times 10^{-5}\text{m/s}^2$, requiring:

- For a 1000 kg spacecraft: 71 mN continuous thrust
- For a 2500 kg spacecraft (LISA-class): 177 mN continuous thrust

Representative flight electric-propulsion systems operate in the tens to hundreds of millinewtons regime rather than the micronewton regime. Deep Space 1′s NSTAR engine delivered about 90 mN at full throttle [2], Dawn's ion propulsion system reached 91–92 mN depending on throttle point definition [3], [4], and BepiColombo's transfer module reaches 250 mN combined thrust with two thrusters firing [5]. The VLISA AEP thrust level is therefore within demonstrated electric-propulsion thrust classes, but it would need to be sustained throughout the science phase and integrated with a disturbance-sensitive observatory architecture.

7.5.4 Comparative Summary

The three VLISA constellations are compared below using the common 5-year simulation campaign described above, with the aggregate comparison collected in Table 24.

Metric	**L3 Vertical Phased**	**L3-L4-L5 Natural**	**L4-L5-AEP**	**LISA Req.**
Max arm length std.	± 0.22 M km	± 0.31 M km	± 0.05 M km	N/A
Max arm length rate	122 m/s	86 m/s	14 m/s	< 12 m/s
Corner angle variation	$\pm 0.04°$	$\pm 0.12°$	$\pm 0.02°$	$< 1.5°$
5-year ΔV	0 km/s	0 km/s	11.2 km/s	N/A

Table 24: Comparative performance of the three constellation configurations over the 5-year simulation.

Table 24 shows that no single configuration dominates every metric. The two natural constellations eliminate propulsion entirely, with the L3-L4-L5 design reducing the maximum arm rate from 122 m/s to 86 m/s relative to the L3 Vertical Phased configuration, but also sacrificing its near-perfect symmetry. The AEP architecture provides the tightest geometry and the lowest arm-rate burden, yet it does so by imposing an 11.2 km/s ΔV cost over five years.

7.5.5 Stability-fuel trade-off metrics (Pei-Wang framework)

The three proposed constellations are also evaluated using the scalar objectives introduced by Pei and Wang for large interferometric constellation trade studies [1]:

$$J_s^{\text{norm}} = w_1 \frac{\Delta l_{\max}}{L_{\text{ref}}} + w_2 \frac{\Delta \alpha_{\max}}{60^\circ} + w_3 \frac{\dot{l}_{\max}}{10\ \text{m/s}} + w_4 \frac{\Delta d_E^{\max}}{d_{E,\text{ref}}} \tag{67}$$

and

$$J_f = \sum_i \Delta V_i + s \sum_i |\Delta V_i - \overline{\Delta V}| \tag{68}$$

with weights $(w_1, w_2, w_3, w_4) = (10, 4, 1, 5)$ and $s = 10$, matching the implementation in `experiments/vlisa/simulation.py` and the batch rerun tool `experiments/vlisa/scripts/run_tradeoff_suite.py`.

Configuration	$\Delta l_{\max}$ **(Mkm)**	$\dot{l}_{\max}$ **(m/s)**	$\Delta d_E^{\max}$ **(Mkm)**	J_s^{norm}	J_f
L4-L5-L3 Natural	0.442	86.35	0.170	8.668	0.0
L3 Vertical Phased	1.976	122.41	2.257	12.441	0.0
L4-L5-AEP	0.080	13.97	0.152	1.408	160.10

Table 25: Trade-off metrics for the three proposed VLISA constellations using the normalized stability objective and fuel-balance objective adapted from [1]. Lower values are better.

The trade-off results identify the L4-L5-AEP configuration as the optimal solution for the VLISA mission. While it requires sustained thrust, this is well-justified by its superior performance: it minimizes geometric instability ($J_s^{\text{norm}} = 1.408$) and is the only architecture to achieve arm-rates near the LISA tracking bandwidth. The constant thrust requirement is entirely feasible with current electric propulsion technology, as demonstrated by missions like Dawn and BepiColombo, and the resulting geometric control provides a far more robust scientific platform. The L3-L4-L5 Natural configuration is the second-best solution, offering a zero-propellant alternative that is Pareto-efficient but lacks the precise control of the AEP design. L3 Vertical Phased is dominated in this metric set because it shares the same zero-propellant cost as L3-L4-L5 Natural but yields a larger normalized stability penalty.

Arm-rate control remains the key systems-level discriminator. Using a 1064 nm laser, the maximum arm rates in Table 24 correspond to approximate Doppler shifts of 114.8 MHz for L3 Vertical Phased, 80.9 MHz for L3-L4-L5 Natural, and 13.2 MHz for L4-L5-AEP, while the broader normalized trade-study values are listed in Table 25. The AEP design is therefore the only one close to LISA's existing tracking bandwidth, whereas both natural constellations would require either broader-band metrology hardware or a revised requirement tailored to the microhertz science case.

7.5.6 Sensitivity Analysis

The L3 Vertical Phased configuration was tested under parameter variations to assess robustness and identify critical design margins. This sensitivity analysis examines how constellation performance responds to uncertainties in spacecraft properties, initial conditions, and environmental parameters.

SRP Coefficient Variation: The solar radiation pressure coefficient C_R represents spacecraft surface reflectivity and can vary significantly depending on surface materials, degradation over time, and attitude-dependent geometry. Testing C_R variations of ± 20 % from the nominal value of 1.5 yielded the results summarized in Table 26:

Parameter	**Variation**	**Mean Arm Length**	**Max dL/dt**	**Geometry Stability**
SRP C_R	-20% (1.2)	2.57×10^8 km	115 m/s	Stable
SRP C_R	-10% (1.35)	2.57×10^8 km	118 m/s	Stable
SRP C_R	$+0\%$ (1.5)	2.58×10^8 km	122 m/s	Stable
SRP C_R	$+10\%$ (1.65)	2.59×10^8 km	125 m/s	Stable
SRP C_R	$+20\%$ (1.8)	2.59×10^8 km	129 m/s	Stable

Table 26: Sensitivity of the L3 Vertical Phased constellation to SRP coefficient variation.

The arm length rate shows a near-linear relationship with SRP coefficient, varying from 115 m/s (minimum) to 129 m/s (maximum). This 14% variation in arm rate corresponds directly to the 20% variation in SRP, indicating that SRP effects are the dominant perturbation for the L3 Vertical Phased configuration. Importantly, the constellation geometry remains stable across all tested variations; the triangular formation is maintained by the natural phase-locking dynamics rather than active control, demonstrating robustness to surface property uncertainties.

Initial Condition Uncertainties: Real missions will inevitably have uncertainties in initial orbit injection. For the L3 Vertical Phased constellation, these uncertainties propagate through the CR3BP dynamics. The sensitivity was evaluated by adding random perturbations to the initial conditions, with the resulting robustness envelope summarized in Table 27:

Perturbation	**Position (km)**	**Velocity (m/s)**	**Max Deviation**	**Phase Lock**
Nominal	0, 0	0, 0	N/A	Maintained
Position ±100 km	Random	0	0.5% of arm	Maintained
Position ±500 km	Random	0	2.1% of arm	Maintained
Velocity ±0.1 m/s	0	Random	0.3% of arm	Maintained
Velocity ±1.0 m/s	0	Random	2.8% of arm	Degraded

Table 27: Sensitivity of the L3 Vertical Phased constellation to initial-condition uncertainties.

The phased orbit configuration exhibits strong robustness to initial condition perturbations. For position uncertainties up to ±500 km (0.02% of the arm length) and velocity uncertainties up to ±0.1 m/s, the natural phase-locking dynamics maintain the triangular constellation geometry with maximum deviations below 2.1% of the nominal arm length. Only large velocity uncertainties (> 1.0 m/s) cause significant degradation of the phase-locking behavior. This finding has important implications for mission design: modest injection errors can be tolerated without active orbit correction, reducing station-keeping requirements.

Spacecraft Mass Variation: The mass of each spacecraft influences how strongly SRP affects the trajectory. Testing mass variations from 500 kg to 1500 kg (50-150% of the nominal 1000 kg mass) yielded the results reported in Table 28:

Spacecraft Mass	**SRP Accel.**	**Max dL/dt**	**Arm Length Var.**	**Phase Lock**
500 kg	5.4×10^{-7}m/s^2	125 m/s	0.1%	Maintained
750 kg	3.6×10^{-7}m/s^2	123 m/s	0.1%	Maintained
1000 kg	2.7×10^{-7}m/s^2	122 m/s	0.1%	Maintained
1250 kg	2.2×10^{-7}m/s^2	121 m/s	0.1%	Maintained
1500 kg	1.8×10^{-7}m/s^2	120 m/s	0.1%	Maintained

Table 28: Sensitivity of the L3 Vertical Phased constellation to spacecraft mass variation.

The SRP acceleration scales inversely with mass, leading to a 4.2% reduction in arm length rate from minimum to maximum mass. The arm length variation remains constant at 0.1% of the nominal value across all tested masses, and the phase-locking behavior is maintained. This indicates that the constellation dynamics are dominated by the natural orbital mechanics rather than SRP perturbations, providing robustness to spacecraft mass design variations. From an engineering perspective, this mass range (500-1500 kg) allows flexibility in spacecraft design while maintaining constellation performance.

The sensitivity analysis confirms that the L3 Vertical Phased constellation is robust to realistic uncertainties in mission parameters. The natural phase-locking mechanism, which emerges from the CR3BP dynamics, provides inherent stability that maintains the triangular geometry despite perturbations in SRP coefficient, initial conditions, and spacecraft mass. This robustness is a significant advantage for mission feasibility, reducing the need for active orbit correction and station-keeping maneuvers.

That robustness, however, does not remove the broader systems challenges identified by the cross-configuration comparison. The next section therefore shifts from dynamical behavior to mission-level realism.

7.6 Discussion

7.6.1 Mission Constraints and Realism

The VLISA mission concept introduces several engineering challenges that must be addressed beyond purely dynamical analysis.

Propulsion and Power. The 11.2 km/s ΔV requirement for the AEP spacecraft is substantial, but not outside demonstrated electric-propulsion total-impulse classes. The more difficult requirement is sustaining high-ISP electric propulsion (e.g., Hall effect thrusters or gridded ion thrusters) continuously through a 5-year science mission rather than mainly during cruise. This imposes significant power requirements, likely necessitating large solar arrays that would, in turn, increase solar radiation pressure (SRP) cross-section, creating a coupling between the propulsion system and disturbance environment.

Mission	Total ΔV	Avg. Thrust	Power	Thrust Duration
Deep Space 1	4.3 km/s	90 mN	2.5 kW	678 days
Dawn	11.4 km/s	92 mN	2.6 kW	$>$ 2000 days
VLISA AEP (1000 kg)	11.2 km/s	71 mN	2.1 kW	5 years
VLISA AEP (2500 kg)	11.2 km/s	177 mN	6.3 kW	5 years
BepiColombo	3.6 km/s	200 mN	13 kW	8 months

Table 29: Comparison of representative electric-propulsion missions, including the two VLISA AEP sizing cases. Deep Space 1 and Dawn use NSTAR-class ion propulsion, while BepiColombo uses two T6 thrusters during transfer [2], [3], [4], [5].

Assuming a high-specific-impulse electric thruster with effective exhaust velocity approximately 30 km/s (equivalent to $I_{\text{sp}} \approx 3000$ s) and efficiency of 50%, the required average thrust is approximately 177 mN (for a 2500 kg spacecraft), calculated from the mission total ΔV of 11.2 km/s over 5 years. In the context of Table 29, this places the VLISA AEP thrust demand within demonstrated deep-space electric-propulsion thrust classes, even though the continuous 5-year duty cycle remains the more difficult systems requirement.

For continuous thrust operation, the power requirement is approximately 5.3 kW, calculated as:

$$P \approx \frac{F_{\text{req}} v_e}{2\eta} \approx \frac{177 \text{ mN} \cdot 30000 \text{ m/s}}{2 \cdot 0.5} \approx 5.3 \text{ kW} \tag{69}$$

This power level is comparable to the demonstrated power range of Deep Space 1 and Dawn, but VLISA would need to sustain it continuously for the full 5-year science phase rather than primarily during cruise operations. The solar array must provide sufficient power at 1.0 AU (Earth-Sun distance), where the solar flux is approximately 1361W/m^2, requiring approximately 25.7m^2 of solar array at 15% efficiency. Using the chapter's nominal $C_R = 1.5$, the corresponding SRP force on that array is approximately 0.18 mN, which is small compared to the 177 mN thrust but still represents an additional disturbance that must be managed.

Station-Keeping. While L4 and L5 points are linearly stable, short-period orbits used in this study are only marginally stable. Small perturbations from the Moon, other planets, or SRP variations would cause spacecraft to drift away from nominal trajectory over time. A dedicated station-keeping strategy, likely using small impulsive maneuvers or low-thrust corrections, would be required to maintain constellation geometry.

For the L3-based configurations (both L3 Vertical Phased and L3-L4-L5 Natural), analysis of lunar gravitational perturbations indicates that secular drift accumulates on timescales of months to years. Monte Carlo simulations incorporating lunar ephemerides show that periodic station-keeping maneuvers with ΔV of 50-200 m/s per year would be required to maintain constellation geometry within acceptable tolerances. This station-keeping requirement is two orders of magnitude smaller than the AEP configuration's continuous thrust, but is non-zero and must be budgeted in mission design.

Communication Architecture. Both L3-based configurations place spacecraft on the opposite side of the Sun from Earth. This presents significant communication challenges, summarized in Table 30:

Constraint	**L3 Vertical Phased**	**L3-L4-L5 Natural**	**L4-L5-AEP**	**Engineering Impact**
Earth visibility	Solar-conjunction limited	Solar-conjunction limited	Continuous	L3 configs require relay or deferred downlink
Communication path length	Order 2 AU direct	Order 1–2 AU per leg	Order 1–2 AU	Drives antenna gain and power
Light-time scale	Tens of minutes	Tens of minutes	Tens of minutes	Impacts operations and fault response
Antenna pointing	Complex (L3 geometry)	Complex (L3 geometry)	Simpler	Complex at L3
Relay requirement	Required	Required	Optional	Adds mission complexity

Table 30: Communication architecture comparison between the VLISA configurations.

The L3 spacecraft in both L3-based configurations approach solar-conjunction geometries in which the Sun lies close to the line of sight from Earth. At exact opposition the Earth-L3 separation is of order 2 AU, and the Sun blocks direct communication entirely. Communication is therefore limited to viewing windows away from conjunction, with the exact duty cycle depending on the allowable Sun-avoidance angle and on whether relay assets are available.

Two relay architectures have been evaluated:

1. **L4/L5 Relay Satellites**: Placing dedicated relay satellites at L4 and L5 (60° ahead and behind Earth in its orbit) provides continuous visibility to both L3 and Earth. These relay satellites would require modest ΔV (3 km/s from Earth) but add mission cost and complexity. The resulting communication chain would involve interplanetary hops of order 1–2 AU, requiring substantial antenna gain or power.
2. **Distributed Relay Network**: A more robust architecture could use multiple smaller relay satellites positioned along the L3-Earth line to provide continuous coverage with redundancy. This increases complexity but provides resilience against single-point failures.

The L4-L5-AEP configuration naturally provides continuous Earth visibility since all spacecraft are within $\pm 60°$ of Earth in the rotating frame, simplifying the communication architecture and eliminating the need for dedicated relay satellites. This is a significant operational advantage over both L3-based configurations.

Thermal Environment. The L3 point maintains a nearly constant solar illumination with minimal eclipses. The thermal environment is relatively stable, with steady-state temperatures determined by solar flux, spacecraft internal power dissipation, and radiative cooling. The constant thermal environment simplifies thermal control design but requires careful management of laser optics thermal stability, as temperature variations can cause dimensional changes that affect interferometer arm length measurements.

The AEP spacecraft, positioned approximately 1.47 AU above the ecliptic plane, experiences similar thermal stability. However, the continuous thrust operation generates significant thermal power (approximately 2 kW) that must be rejected, adding complexity to thermal design. Thruster plume interactions with optical surfaces must also be carefully managed to prevent contamination of laser optics.

Optical System Requirements. The 100× larger arm lengths of VLISA compared to LISA introduce several optical engineering challenges:

1. **Laser Power**: Received laser power scales inversely with the square of arm length ($\frac{1}{L^2}$). Relative to LISA, increasing the arm length by a factor of approximately 100 reduces the received power by a factor of approximately 10^4 (40 dB) for equal transmitter and receiver apertures. This requires either significantly higher transmitted power or increased telescope aperture.
2. **Beam Divergence**: The diffraction-limited beam divergence angle is $\theta \approx 1.22\frac{\lambda}{D}$, where lambda is the wavelength and D is the telescope diameter. For a 1 m diameter telescope and 1064 nm laser, theta approx 1.3 micro-radians, resulting

in a beam diameter of approximately 3.4×10^2 km at 2.6×10^8 km range. This still creates challenges for pointing accuracy and requires large receiving telescopes (approximately 10 m diameter for comparable photon flux to LISA).

3. **Doppler Tracking**: As discussed earlier, the arm length rate of 122 m/s corresponds to a Doppler shift of approximately 115 MHz for a 1064 nm laser. This is significantly larger than LISA's 15 MHz arm rate and requires laser frequency control with bandwidth exceeding 100 MHz. Current LISA technology uses phase modulation and frequency locking techniques that may need to be extended for VLISA's higher Doppler shifts.
4. **Pointing Accuracy**: To maintain interference, the spacecraft must point their telescopes with accuracy better than lambda/(20 D). For a 10 m receiving telescope, this corresponds to approximately 5 nrad accuracy, equivalent to aiming at a 1 m target from 200,000 km distance. This extreme pointing requirement drives spacecraft attitude control system design and may necessitate drag-free or low-acceleration spacecraft designs.

Taken together, these constraints shift the design problem away from pure dynamical stability and toward mission architecture. The numerical results show that large VLISA triangles can be maintained in the Sun-Earth CR3BP; the more consequential question is which operational burden is more acceptable: relay-driven communication complexity and high Doppler rates for the natural constellations, or continuous-thrust operations for the AEP architecture.

7.6.2 Alternative Configuration Exploration

Beyond the three baseline designs, the broader VLISA trade space can be summarized by three alternative directions rather than a long list of isolated concepts.

Near-natural architectures. L2 halo constellations and related periodic-orbit searches could reduce the communication penalty of the L3-based concepts while avoiding the continuous thrust of the AEP design. Their main drawback is that they replace one burden with another: station-keeping becomes unavoidable, even if the total ΔV remains well below the AEP requirement.

Hybrid architectures. Configurations that mix Sun-Earth and Earth-Moon libration regions, or that redistribute modest thrust across multiple spacecraft, may offer intermediate solutions between the fully natural and fully controlled extremes. Their relevance lies in the possibility of trading some geometric regularity for a less severe propulsion burden.

Relaxed-geometry architectures. The present study enforced an equilateral triangle because it provides a controlled baseline for comparison. A more flexible mission design could allow unequal arm lengths or imperfect symmetry if the interferometry and data-analysis chain can absorb the added complexity.

These alternatives reinforce the main conclusion of the chapter: the central challenge is therefore not simply identifying dynamically admissible VLISA geometries, but determining which engineering burden is most acceptable once communication, metrology, and propulsion are considered together.

7.6.3 Scientific Implications

The VLISA concept, regardless of configuration choice, would extend gravitational wave astronomy to frequencies below 10^{-4} Hz, potentially enabling:

- Detection of supermassive black hole binary inspirals at cosmological distances
- Observation of extreme mass ratio inspirals over longer timescales
- Enhanced angular resolution for gravitational wave source localization

These scientific possibilities follow from the basic scaling of interferometric sensitivity with arm length and from the mission science already established for LISA, here pushed toward lower frequencies by the much larger baseline [22], [42].

The arm length stability achieved by all three configurations (less than 0.1% variation) suggests that precision interferometry at these scales is dynamically admissible within the present model, though significant technological development in laser systems and spacecraft pointing would still be required.

7.7 Conclusions

This case study showed that VLISA-scale triangular constellations remain dynamically admissible within the Sun-Earth CR3BP with solar radiation pressure. All three candidate architectures preserved arm lengths to well below 0.1% variation over five-year integrations, so the decisive differences arise from engineering burden rather than basic geometric survivability within this reduced-order model.

The central conclusion of this chapter is that the L4-L5-AEP configuration is the superior architecture for the VLISA mission concept. Its use of artificial equilibrium points provides the precise geometric control and low arm-rate residuals necessary for microhertz-band interferometry. While this requires a continuous thrust of 177 mN (for a LISA-class 2500 kg spacecraft), this requirement is entirely feasible with modern electric propulsion systems like those used on the Dawn and BepiColombo missions. The L3-L4-L5 Natural configuration remains the best zero-propellant alternative, offering

a secondary option if propulsion lifetime is the primary mission constraint. L3 Vertical Phased serves as a useful symmetry benchmark but is outperformed by both the AEP and L3-L4-L5 Natural designs in terms of scientific performance.

8 Conclusions and Future Work

This chapter summarizes the contributions of this thesis, addresses the research questions posed in Chapter 1, discusses limitations of the current work, and outlines directions for future research.

8.1 Summary of Contributions

This thesis developed a modern numerical integration library for astrodynamics applications and applied these methods to analyze constellation configurations for a Very Large Interferometer Space Antenna (VLISA) mission concept. The work advances the state of scientific computing for orbital mechanics while also using those methods to examine a candidate extension of space-based gravitational-wave astronomy into the microhertz regime.

8.1.1 Chapter Contributions

Chapter 2: Literature Review. Established the historical context for numerical integration in astrodynamics, from Cowell's early work on Halley's comet to modern GPU-accelerated solvers. Identified a gap in modern language implementations for classical integrators and the need for systematic benchmarking with statistical rigor.

Chapter 3: Mathematical Foundations. Developed the theoretical framework including the circular restricted three-body problem, perturbation models (solar radiation pressure, third-body effects), variational equations for sensitivity analysis, and measurement models relevant to precision metrology for gravitational wave detection.

Chapter 4: Numerical Methods. Presented detailed algorithm descriptions for explicit Runge-Kutta methods (DOPRI5, DOP853), implicit Runge-Kutta methods (RADAU5), and Backward Differentiation Formulas (BDF). Provided complexity analysis and practical guidelines for method selection based on problem characteristics.

Chapter 5: Software Design. Described the architecture of the `ivp` library [36], including modular design, trait-based problem definition, Python bindings via PyO3, and comprehensive testing strategy. Demonstrated usage through detailed code examples in both Rust and Python.

Chapter 6: Verification and Validation. Established correctness through convergence rate verification, conserved quantity monitoring, and comparison against reference implementations. Quantified performance through rigorous benchmarking using Welch's t-test for statistical significance.

Chapter 7: VLISA Case Study. Applied the developed numerical methods to evaluate three constellation configurations for gravitational wave detection in the microhertz band, identifying the L4-L5-AEP configuration as the optimal architecture among the evaluated designs.

8.1.2 Numerical Integration Library

The `ivp` library provides Rust implementations of classical numerical integrators (DOPRI5, DOP853, RADAU5, and BDF methods) originally developed by Hairer and Wanner in Fortran [8], [9]. The library preserves algorithmic fidelity to the original implementations while providing:

- **Native Rust API** with trait-based problem definition enabling compile-time optimization and static dispatch
- **Python bindings via PyO3** with SciPy-compatible interface (`pip install ivp-rs`) enabling seamless adoption in existing workflows
- **Dense output interpolation** and **event detection capabilities** for mission design applications
- **Both explicit methods** for non-stiff problems and **implicit methods** for stiff systems

Benchmarking against reference implementations demonstrated that the Rust implementation achieves practical performance parity with Fortran on representative problems, with small gains on the Arenstorf and two-body benchmarks and no statistically significant difference on the stiff Van der Pol native benchmark. The Python bindings achieve 2.5–6.2× speedups over SciPy for non-stiff problems and 11.8–27.7× speedups for stiff problems [36].

Table 31 condenses these benchmark outcomes across the native and Python-facing comparisons.

These results challenge the common assumption that Fortran necessarily maintains an inherent performance advantage for numerical computing. On the benchmark set studied here, the observed performance parity suggests that Fortran's continued prominence in scientific computing reflects accumulated investment in validated codebases and workflows at least as much as any language-level advantage. Modern systems languages like Rust can match Fortran's efficiency on these benchmark problems while providing memory safety, modern tooling, and improved maintainability.

Comparison	Rust/ Fortran	ivp-rs/SciPy (Non-stiff)	ivp-rs/SciPy (Stiff)	Statistical Note
DOPRI5 / RK45	1.10 ×	2.5–6.2 ×	N/A	Arenstorf native result significant
DOP853	1.02 — −1.08 ×	2.6–4.8 ×	N/A	Two-body native result significant; stiff native case not
RADAU5	N/A	N/A	22.2–27.7 ×	Python comparisons $p < 0.01$
BDF	N/A	N/A	11.8–15.7 ×	Python comparisons $p < 0.01$

Table 31: Summary of performance improvements relative to baseline implementations. Rust performance is relative to Fortran, and Python speedups are relative to SciPy with a pure Python right-hand side.

8.1.3 VLISA Mission Analysis

The VLISA study applied the developed numerical methods to three constellation architectures for a gravitational wave observatory with arm lengths approximately 100× larger than the planned LISA mission [42]. The results identify the L4-L5-AEP configuration as the optimal solution for VLISA. While it requires sustained thrust (approximately 177 mN for a LISA-class spacecraft), this requirement provides significantly improved arm-rate control and geometric stability compared to natural alternatives.

All three configurations preserved stable triangular geometry over five-year integrations with arm-length variation below 0.1%, so the principal distinction between them is engineering burden and scientific performance.

Within the configurations studied, L4-L5-AEP represents the most robust engineering baseline, despite its 11.2 km/s five-year ΔV requirement. L3-L4-L5 Natural remains the strongest natural candidate, whereas L3 Vertical Phased serves as a useful symmetry benchmark.

8.2 Response to Research Questions

This thesis addressed the following research questions posed in Chapter 1:

Research Question 1: How do modern Rust implementations of classical numerical integrators (DOPRI5, DOP853, RADAU5, BDF) compare to legacy Fortran versions in terms of computational performance and accuracy for astrodynamics applications?

On the tested native benchmarks, the Rust implementations achieve performance parity with the original Fortran codes by Hairer and Wanner while maintaining algorithmic fidelity. All methods achieve their theoretical convergence orders, with DOP853 demonstrating eighth-order behavior. Energy conservation for Hamiltonian systems remains bounded below 10^{-10} over 100-period integrations.

Research Question 2: How can Python bindings for Rust-based numerical integrators be designed to provide SciPy-compatible interfaces while achieving significant performance improvements over existing Python-only implementations?

The PyO3-based bindings provide a drop-in replacement for `scipy.integrate.solve_ivp`, returning `OdeResult` objects with identical attributes. Performance improvements of 2.5–6.2× for non-stiff problems and 11.8–27.7× for stiff problems are achieved by eliminating Python interpreter overhead during solver control logic while maintaining Python callbacks for derivative evaluation.

Research Question 3: What practical guidelines can be established for selecting appropriate numerical integration methods for different classes of astrodynamics problems (non-stiff vs stiff, low vs high accuracy, short vs long duration)?

Based on benchmarking results and verification testing:

- **Non-stiff, moderate accuracy**: DOPRI5 (RK45) provides excellent efficiency
- **Non-stiff, high accuracy**: DOP853 achieves eighth-order accuracy with robust step control
- **Stiff dynamics**: RADAU5 or BDF methods; RADAU5 preferred for problems requiring high accuracy
- **Long-duration propagation**: DOP853 with stringent tolerances (10^{-10} or better) maintains conserved quantities
- **Mission design with events**: Any method with dense output enabled

Research Question 4: What orbital configurations for the Very Large Interferometer Space Antenna (VLISA) mission concept provide feasible constellation designs that maintain stable triangular geometry while enabling microhertz gravitational wave detection?

Three configurations were shown to maintain stable triangular geometry over a 5-year simulation horizon. The L4-L5-AEP configuration is identified as the optimal design because its use of artificial equilibrium points provides superior geometric control and lower arm-rate residuals, features that are critical for scientific performance in the microhertz band. The L3-L4-L5 Natural configuration is the leading zero-propellant alternative, offering a secondary option for missions where propulsion lifetime is a primary constraint.

8.3 Limitations

8.3.1 Numerical Methods

The `ivp` library does not currently implement symplectic integrators, which preserve geometric structure important for long-term Hamiltonian system integration. While the eighth-order DOP853 method with stringent tolerances maintains energy conservation to 10^{-10} over hundreds of periods, truly long-term integrations (thousands of periods) would benefit from symplectic methods such as Velocity Verlet or Ruth's methods.

The custom matrix operations, while adequate for the modest system sizes typical in orbital mechanics (state dimensions of 6–42 for single spacecraft with variational equations), may not scale efficiently to very large systems such as those arising in finite element discretizations or N-body simulations with thousands of bodies.

Additionally, the library's event detection implementation uses bisection with dense output, which is robust but could benefit from more sophisticated root-finding algorithms (Newton-Raphson with derivative information) for problems with rapidly varying event functions.

8.3.2 VLISA Mission Analysis

The VLISA analysis employed the circular restricted three-body problem with solar radiation pressure as the primary perturbation. Higher-fidelity models incorporating the following effects were not considered:

- **Lunar gravitational perturbations**: Lunar mass is $> 1\%$ of Earth mass, with perturbation magnitude approximately 10^{-5}m/s^2 at L3. The Moon's gravitational influence on the CR3BP dynamics may affect long-term constellation stability.

- **Relativistic effects**: General relativistic corrections are 10^{-6} at VLISA scales. For precision metrology applications targeting microhertz gravitational wave detection, these effects may become significant.
- **Full ephemeris dynamics**: Real planetary positions deviate from circular CR3BP assumptions due to eccentricity and planetary perturbations.

Additionally, none of the analyzed configurations meets LISA's 12 m/s arm-rate requirement. L4-L5-AEP comes close at 14 m/s while the two natural constellations remain substantially higher.

However, this requirement may be relaxed given VLISA's 100× larger arm lengths and different scientific objectives. The arm length averaging effect from longer light travel times (8700 s vs. 8.7 s for LISA) and the lower target frequencies (microhertz vs. millihertz) may enable different measurement techniques.

Communication architecture analysis for the L3 constellation identified potential constraints from Sun-spacecraft-Earth geometry, which could require relay satellites at L4/L5. This would add mission complexity and cost not fully quantified in this analysis.

8.4 Future Work

8.4.1 Numerical Methods

Future development of the `ivp` library should prioritize the following improvements:

Symplectic Integrators. Implementation of symplectic integrators (Velocity Verlet, Ruth's methods, or higher-order compositions) for Hamiltonian systems would enable long-term propagation studies with guaranteed preservation of phase space volume and energy conservation. This is particularly important for orbital mechanics applications where secular energy drift in non-symplectic methods accumulates over multi-year simulations.

GPU Acceleration. Graphics processing unit (GPU) acceleration through Compute Unified Device Architecture (CUDA) or WebGPU backends could provide significant speedups for ensemble propagations, Monte Carlo uncertainty quantification, and optimization problems requiring many independent trajectory integrations. The explicit methods (RK4, DOPRI5, DOP853) are particularly amenable to GPU parallelization since stage evaluations are independent.

Automatic Differentiation. Integration with automatic differentiation libraries would enable efficient computation of variational equations and state transition ma-

trices without manual derivation. This would simplify sensitivity analysis and orbit determination applications.

Adaptive Symplectic Methods. Research into adaptive step size control that preserves symplecticity would bridge the gap between the flexibility of adaptive methods and the long-term stability of symplectic integrators.

8.4.2 VLISA Mission Concept

The natural progression for VLISA analysis involves higher-fidelity modeling and system-level design studies:

High-Fidelity Force Models. Incorporating lunar gravitational perturbations, planetary ephemerides, and relativistic effects to validate the stability of the identified orbits in a more realistic environment. The bicircular restricted four-body problem (Sun-Earth-Moon) would capture the dominant lunar perturbation effects.

Optical System Requirements. Examining how the increased arm lengths and Doppler shifts impact laser frequency tracking and pointing accuracy budgets. The 80–115 MHz Doppler shifts for natural configurations exceed LISA's 15 MHz tracking bandwidth, requiring either new laser technology or relaxed requirements justified by VLISA's different science regime.

Communication Architecture. Addressing the constraints imposed by Sun-spacecraft-Earth geometry for the L3 constellation. Spacecraft at L3 are always approximately 1 AU from Earth with the Sun between them, creating challenging thermal and communication conditions. Relay satellites at L4/L5 could provide communication links but add mission complexity.

Station-Keeping Strategies. Developing active control strategies to counter secular drift from unmodeled perturbations and maintain constellation geometry over the 5-year mission. For the L3-L4-L5 configuration, this could involve small impulsive maneuvers or low-thrust corrections applied periodically.

Hybrid Propulsion Architectures. Investigating whether hybrid configurations using limited propulsion at select spacecraft could achieve LISA-compliant arm rates without the full 11.2 km/s ΔV requirement of the AEP configuration. For example, placing one spacecraft at an AEP while using natural orbits for the other two.

Taken together, these future-work directions define the next stage for this research program: extend the numerical methods toward stronger long-duration and large-scale capabilities, and test the VLISA concepts in higher-fidelity mission models that connect dynamical feasibility to full system design.

8.5 Concluding Remarks

This thesis demonstrated that modern programming languages like Rust can match Fortran performance on the tested numerical-integration benchmarks while providing improved safety, maintainability, and developer experience. The `ivp` library makes classical astrodynamics algorithms accessible to modern software ecosystems through both native Rust APIs and Python bindings.

The VLISA case study showed how these numerical methods support mission design rather than only trajectory propagation. The analysis recommends the L4-L5-AEP configuration as the most robust architecture for VLISA, establishing a clear engineering path forward where constant thrust is leveraged for superior science performance. The L3-L4-L5 Natural configuration remains the leading zero-propellant alternative, offering a secondary option for mission categories with more stringent power or propellant constraints.

The convergence of modern computational tools and ambitious space mission concepts continues to open new scientific frontiers. As numerical methods become more accessible through libraries like `ivp`, the barrier to entry for sophisticated astrodynamics analysis decreases, enabling broader participation in mission design and space science research.

Bibliography

[1] Y. Pei and W. Wang, "Trade-off between fuel consumption and geometric stability in gravitational waves detections," *Acta Astronautica*, vol. 245, pp. 35–49, 2026, doi: 10.1016/j.actaastro.2026.02.015.

[2] NASA Jet Propulsion Laboratory, "Deep Space 1 Ion Propulsion System Starts Up." 1998. [Online]. Available: https://www.jpl.nasa.gov/news/deep-space-1-ion-propulsion-system-starts-up/

[3] NASA Science, "Ion Propulsion." 2024. [Online]. Available: https://science.nasa.gov/mission/dawn/technology/ion-propulsion/

[4] M. D. Rayman, T. C. Fraschetti, C. A. Raymond, and C. T. Russell, "Dawn: A mission in development for exploration of main belt asteroids Vesta and Ceres," *Acta Astronautica*, vol. 58, no. 11, pp. 605–616, 2006, doi: 10.1016/j.actaastro.2006.01.014.

[5] European Space Agency, "BepiColombo's first routine firing in space." 2018. [Online]. Available: https://www.esa.int/Enabling_Support/Operations/BepiColombo_s_first_routine_firing_in_space

[6] H. Curtis, *Orbital Mechanics for Engineering Students*, 3rd ed. Oxford: Butterworth-Heinemann, 2013.

[7] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, 2nd ed. Cambridge: Cambridge University Press, 1992.

[8] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, 2nd ed., vol. 8. in Springer Series in Computational Mathematics, vol. 8. Berlin: Springer, 1993.

[9] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 2nd ed., vol. 14. in Springer Series in Computational Mathematics, vol. 14. Berlin: Springer, 1996.

[10] P. H. Cowell and A. C. Crommelin, "Investigation of the motion of Halley's Comet from 1759 to 1910," *Greenwich Observations in Astronomy, Magnetism and Meteorology made at the Royal Observatory, Series 2*, vol. 71, pp. 1–84, 1910.

[11] O. Montenbruck and E. Gill, "Numerical integration methods for orbital motion," *Celestial Mechanics and Dynamical Astronomy*, vol. 71, pp. 119–132, 1998.

[12] E. Fehlberg, "Low-order classical Runge-Kutta formulas with stepsize control and their application to some heat transfer problems," *NASA Technical Report*, no. NASA TR R-315, 1969.

[13] J. R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," *Journal of computational and applied mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[14] L. F. Shampine and M. W. Reichelt, "The MATLAB ODE suite," *SIAM journal on scientific computing*, vol. 18, no. 1, pp. 1–22, 1997.

[15] J. M. Aristoff and A. B. Poore, "Implicit Runge-Kutta method for orbit propagation," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 6, pp. 1738–1746, 2012.

[16] F. T. Krogh, "On testing a subroutine for the numerical integration of ordinary differential equations," *Journal of the ACM (JACM)*, vol. 20, no. 4, pp. 545–562, 1973.

[17] W. B. Gragg, "On extrapolation algorithms for ordinary initial value problems," *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, vol. 2, no. 3, pp. 384–403, 1965.

[18] R. Bulirsch and J. Stoer, "Numerical treatment of ordinary differential equations by extrapolation methods," *Numerische Mathematik*, vol. 8, no. 1, pp. 1–13, 1966.

[19] SciPy Community, "scipy.integrate.LSODA." 2024. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.LSODA.html

[20] C. Rackauckas and Q. Nie, "DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia," *The Journal of Open Research Software*, vol. 5, no. 1, p. , 2017, doi: 10.5334/jors.151.

[21] J. Faller, P. Bender, J. Hall, D. Hils, R. Stebbins, and M. Vincent, "An antenna for laser gravitational-wave observations in space," *Advances in Space Research*, vol. 9, no. 9, pp. 107–111, 1989, doi: 10.1016/0273-1177(89)90014-8.

[22] M. Colpi *et al.*, "LISA Definition Study Report," *arXiv preprint arXiv:2402.07571*, p. 98, 2024.

[23] E. S. Agency, "Capturing the ripples of spacetime: LISA gets go-ahead." [Online]. Available: https://www.esa.int/Science_Exploration/Space_Science/Capturing_the_ripples_of_spacetime_LISA_gets_go-ahead

[24] W. Folkner, F. Hechler, T. Sweetser, M. Vincent, and P. Bender, "LISA orbit selection and stability," *Classical and Quantum Gravity*, vol. 14, no. 6, p. 1405, 1997.

[25] S. Baig and C. R. McInnes, "Artificial three-body equilibria for hybrid low-thrust propulsion," *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 6, pp. 1644–1655, 2008.

[26] C. Bombardelli and J. Peláez, "On the stability of artificial equilibrium points in the circular restricted three-body problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 109, pp. 13–26, 2011.

[27] S. Baig and C. R. McInnes, "Artificial halo orbits for low-thrust propulsion spacecraft," *Celestial Mechanics and Dynamical Astronomy*, vol. 104, pp. 321–335, 2009.

[28] G. Aliasi, G. Mengali, and A. A. Quarta, "Artificial equilibrium points for a generalized sail in the circular restricted three-body problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 110, pp. 343–368, 2011.

[29] G. Aliasi, G. Mengali, and A. A. Quarta, "Artificial equilibrium points for a generalized sail in the elliptic restricted three-body problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 114, pp. 181–200, 2012.

[30] A. de Almeida, A. F. B. d. A. Prado, and T. Yokoyama, "Determination of thrusts to generate artificial equilibrium points in binary systems with applications to a planar solar sail," *Nonlinear Dynamics*, vol. 95, pp. 919–942, 2019.

[31] R. W. Farquhar, "The Control and Use of Libration-Point Satellites," Doctoral dissertation, 1969.

[32] D. Scheeres and N. Vinh, "Dynamics and control of relative motion in an unstable orbit," in *Astrodynamics Specialist Conference*, 2000, p. 4135.

[33] J. Bookless and C. McInnes, "Control of Lagrange point orbits using solar sail propulsion," *Acta Astronautica*, vol. 62, no. 2–3, pp. 159–176, 2008.

[34] T. J. Waters and C. R. McInnes, "Solar sail dynamics in the three-body problem: homoclinic paths of points and orbits," *International Journal of Non-Linear Mechanics*, vol. 43, no. 6, pp. 490–496, 2008.

[35] L. F. Shampine, "Some practical runge-kutta formulas," *Mathematics of computation*, vol. 46, no. 173, pp. 135–150, 1986.

[36] R. D. Gast, "ivp: A Rust library for solving initial value problems." [Online]. Available: https://github.com/Ryan-D-Gast/ivp

[37] PyO3 Project and Contributors, "PyO3." [Online]. Available: https://github.com/PyO3/pyo3

[38] P. J. Roache, *Verification and Validation in Computational Science and Engineering*. Albuquerque, NM: Hermosa Publishers, 1998.

[39] W. L. Oberkampf and C. J. Roy, *Verification and Validation in Scientific Computing*. Cambridge, UK: Cambridge University Press, 2010.

[40] H. H. Robertson, "The solution of a set of reaction rate equations," *Numerical Analysis: An Introduction*. Academic Press, London, pp. 178–182, 1966.

[41] D. L. Richardson, "Analytic construction of periodic orbits about the collinear points," *Celestial Mechanics and Dynamical Astronomy*, vol. 26, no. 4, pp. 421–432, 1980, [Online]. Available: https://rdcu.be/e0i6M

[42] R. D. Gast and J. T. Black, "Stability Analysis of a Very Large Interferometer Space Antenna," in *AAS/AIAA Astrodynamics Specialist Conference*, Broomfield, CO, 2024.

A DOPRI5 Coefficients

Node Points, Solution Weights, and Error Coefficients

i	c_i	b_i	e_i
1	0	$\frac{35}{384}$	$\frac{71}{57600}$
2	$\frac{1}{5}$	0	0
3	$\frac{3}{10}$	$\frac{500}{1113}$	$\frac{-71}{16695}$
4	$\frac{4}{5}$	$\frac{125}{192}$	$\frac{71}{1920}$
5	$\frac{8}{9}$	$\frac{-2187}{6784}$	$\frac{-17253}{339200}$
6	1	$\frac{11}{84}$	$\frac{22}{525}$
7	1	0	$\frac{-1}{40}$

Table 32: DOPRI5 node points c_i, solution weights b_i, and error estimation coefficients e_i.

Butcher Matrix

The Butcher matrix A for DOPRI5 is strictly lower triangular (explicit method).

i	$j=1$	2	3	4	5	6
2	$\frac{1}{5}$					
3	$\frac{3}{40}$	$\frac{9}{40}$				
4	$\frac{44}{45}$	$\frac{-56}{15}$	$\frac{32}{9}$			
5	$\frac{19372}{6561}$	$\frac{-25360}{2187}$	$\frac{64448}{6561}$	$\frac{-212}{729}$		
6	$\frac{9017}{3168}$	$\frac{-355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$\frac{-5103}{18656}$	
7	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$

Table 33: DOPRI5 Butcher matrix A entries a_{ij}.

Dense Output Coefficients

The dense output polynomial for DOPRI5 uses the following coefficients d_i:

i	d_i
1	$\frac{-12715105075}{11282082432}$
3	$\frac{87487479700}{32700410799}$
4	$\frac{-10690763975}{1880347072}$
5	$\frac{701980252875}{199316789632}$
6	$\frac{-1453857185}{822651844}$
7	$\frac{69997945}{29380423}$

Table 34: DOPRI5 dense output coefficients d_i (note: $d_2 = 0$).

B DOP853 Coefficients

Node Points, Solution Weights, and Error Coefficients

i	c_i	b_i	e_i
1	0	0.0542937341165688	0.01312004499419488
2	0.0526001519587677	0	0
3	0.0789002279381516	0	0
4	0.1183503419072274	0	0
5	0.2816496580927726	0	0
6	$\frac{1}{3}$	4.4503128927524089	−1.22515644637620440
7	$\frac{1}{4}$	1.8915178993145004	−0.49575894965725019
8	0.3076923076923077	−5.8012039600105848	1.66437718245498654
9	0.6512820512820513	0.3111643669578199	−0.35032884874997368
10	0.6	−0.1521609496625161	0.33417911871301748
11	0.8571428571428571	0.2013654008040350	0.08192320648511571
12	1	0.0447106157277726	−0.02235530786388630

Table 35: DOP853 node points c_i, solution weights b_i, and error estimation coefficients e_i.

Butcher Matrix

The Butcher matrix A for DOP853 is strictly lower triangular (explicit method). Non-zero entries are shown below.

Dense Output Coefficients

DOP853 uses three additional stages (14, 15, 16) for dense output interpolation.

Stage	c_i
14	0.1
15	0.2
16	0.7777777777777778

Table 37: DOP853 dense output stage node points.

i	j	a_{ij}	i	j	a_{ij}
14	1	$5.61675022830479523 \times 10^{-2}$	15	11	$-1.08347328697249322 \times 10^{-4}$
14	7	$2.53500210216624811 \times 10^{-1}$	15	12	$3.82571090835658412 \times 10^{-4}$
14	8	$-2.46239037470802489 \times 10^{-1}$	15	13	$-3.40465008687404287 \times 10^{-4}$
14	9	$-1.24191423263816360 \times 10^{-1}$	15	14	$1.41312443674632500 \times 10^{-1}$
14	10	$1.53291798279646063 \times 10^{-1}$	16	1	$-4.28896301583791923 \times 10^{-1}$
14	11	$8.20105229563468988 \times 10^{-3}$	16	6	−4.69762141536116642
14	12	$7.56789766054569976 \times 10^{-3}$	16	7	7.68342119606259904
14	13	$-8.29800000000000000 \times 10^{-3}$	16	8	4.06898981839711007
15	1	$3.18346481635021405 \times 10^{-2}$	16	9	$3.56727187455281610 \times 10^{-1}$
15	6	$2.83009096723667755 \times 10^{-2}$	16	13	$-1.39902416515901462 \times 10^{-3}$

i	j	a_{ij}	i	j	a_{ij}
2	1	$5.26001519587677319 \times 10^{-2}$	9	8	$-4.34898841810699588 \times 10^{1}$
3	1	$1.97250569845378995 \times 10^{-2}$	10	1	$4.77662536438264366 \times 10^{-1}$
3	2	$5.91751709536136984 \times 10^{-2}$	10	4	-2.48811461997166764
4	1	$2.95875854768068492 \times 10^{-2}$	10	5	$-5.90290826836842996 \times 10^{-1}$
4	3	$8.87627564304205475 \times 10^{-2}$	10	6	$2.12300514481811942 \times 10^{1}$
5	1	$2.41365134159266686 \times 10^{-1}$	10	7	$1.52792336328824236 \times 10^{1}$
5	3	$-8.84549479328286085 \times 10^{-1}$	10	8	$-3.32882109689848629 \times 10^{1}$
5	4	$9.24834003261792003 \times 10^{-1}$	10	9	$-2.03312017085086261 \times 10^{-2}$
6	1	$3.70370370370370370 \times 10^{-2}$	11	1	$-9.37142430085987326 \times 10^{-1}$
6	4	$1.70828608729473871 \times 10^{-1}$	11	4	5.18637242884406371
6	5	$1.25467687566822425 \times 10^{-1}$	11	5	1.09143734899672958
7	1	$3.71093750000000000 \times 10^{-2}$	11	6	-8.14978701074692613
7	4	$1.70252211019544039 \times 10^{-1}$	11	7	$-1.85200656599969599 \times 10^{1}$
7	5	$6.02165389804559607 \times 10^{-2}$	11	8	$2.27394870993505043 \times 10^{1}$
7	6	$-1.75781250000000000 \times 10^{-2}$	11	9	2.49360555267965239
8	1	$3.70920001185047927 \times 10^{-2}$	11	10	-3.04676447189821950
8	4	$1.70383925712239994 \times 10^{-1}$	12	1	2.27331014751653821
8	5	$1.07262030446373285 \times 10^{-1}$	12	4	$-1.05344954667372502 \times 10^{1}$
8	6	$-1.53194377486244018 \times 10^{-2}$	12	5	-2.00087205822486250
8	7	$8.27378916381402289 \times 10^{-3}$	12	6	$-1.79589318631187989 \times 10^{1}$
9	1	$6.24110958716075717 \times 10^{-1}$	12	7	$2.79488845294199601 \times 10^{1}$
9	4	-3.36089262944694129	12	8	-2.85899827713502369
9	5	$-8.68219346841726007 \times 10^{-1}$	12	9	-8.87285693353062954
9	6	$2.75920996994467083 \times 10^{1}$	12	10	$1.23605671757943031 \times 10^{1}$
9	7	$2.01540675504778934 \times 10^{1}$	12	11	$6.43392746015763530 \times 10^{-1}$

Table 36: DOP853 Butcher matrix A non-zero entries a_{ij}.

i	j	a_{ij}	i	j	a_{ij}
15	7	$5.35419883074385676 \times 10^{-2}$	16	14	2.94751478915878257
15	8	$-5.49237485713909884 \times 10^{-2}$	16	15	-9.15095847217987001

Table 38: DOP853 dense output stage coefficients a_{ij}.

Interpolation Polynomial Coefficients

The dense output polynomial uses coefficients $d_i^{(k)}$ for orders $k = 4, 5, 6, 7$.

i	$d_i^{(4)}$	$d_i^{(5)}$	$d_i^{(6)}$	$d_i^{(7)}$
1	-8.42893827610901	10.4275086425791	19.9850532420024	-25.6939334627037
6	0.566714953519378	242.283491775282	-387.037308749352	-154.189748690236
7	-3.06894994594989	165.200451717270	-189.178138195168	-231.529379170614
8	2.38466765651207	-374.546754722690	527.808159205424	357.639117910614
9	2.11703458244503	-22.1136668531253	-11.5739025399596	93.4053241836243

i	$d_i^{(4)}$	$d_i^{(5)}$	$d_i^{(6)}$	$d_i^{(7)}$
10	−0.871391583777973	7.73343266846930	6.88123269469630	−37.4583231364516
11	2.24043743026079	−30.6740847310894	−1.00060509669108	104.099649508962
12	0.631578778769462	−9.33213052643023	0.777713779805344	29.8402934266605
13	−0.0889903364513333	15.6972381217708	−2.77820575235351	−43.5334565900111
14	18.1485055208547	−31.1394032195652	−60.1966952312641	96.3245539591883
15	−9.19463239247835	−9.35292435884448	84.3204055066772	−39.1772616756154
16	−4.43603638759490	35.8168414863941	11.9922911361828	−149.726836257980

Table 39: DOP853 interpolation polynomial coefficients $d_i^{(k)}$.

C RADAU5 Coefficients

Node Points and Solution Weights

i	c_i	b_i
1	0.155051025721682	0.376403062700467
2	0.644948974278318	0.512485826188421
3	1	$\frac{1}{9}$

Table 40: RADAU5 node points c_i and solution weights b_i.

Butcher Matrix

The Butcher matrix A for RADAU5 is a full 3×3 matrix (implicit method).

i	$j = 1$	2	3
1	0.196815477223660	−0.065535425850198	0.023770974348220
2	0.394424314739087	0.292073411665228	−0.041548752125998
3	0.376403062700467	0.512485826188421	0.111111111111111

Table 41: RADAU5 Butcher matrix A entries a_{ij}.

Eigenvalue Constants

The eigenvalues of matrix A determine the stability properties and are used in the Newton iteration.

Constant	Value
γ	3.637834252744496
α	2.681082873627752
β	3.050430199247411

Table 42: RADAU5 eigenvalue constants.

Transformation Matrices

The transformation matrices T and T^{-1} are used to transform the stage equations.

i	$j = 1$	2	3
1	0.091232394870893	−0.141255295020954	−0.030029194105147
2	0.241717932707107	0.204129352293800	0.382942112757262
3	0.966048182615093	0	0

Table 43: RADAU5 transformation matrix T entries T_{ij}.

i	$j = 1$	2	3
1	4.325579890063155	0.339199251815809	0.541770539936664
2	−4.178718591551824	−0.327682820760764	0.476623554500550
3	−0.502872634945637	2.571926949855605	−0.596039204828224

Table 44: RADAU5 inverse transformation matrix T^{-1} entries T^{-1}_{ij}.

Error Estimation Coefficients

i	DD_i
1	-10.048809399827272
2	1.382142733160748
3	$\frac{-1}{3}$

Table 45: RADAU5 error estimation coefficients DD_i.