



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

T R A B A J O T E R M I N A L

“Modelo para representar comportamientos gravitacionales con dos cuerpos”

PRESENTAN:

**José Emiliano Carrillo Barreiro
José Ángel Robles Otero**

DIRECTORES:

**Dr. Cesar Hernández Vasquez
Dr. Mauricio Olguín Carbajal**



No. de TT: 2025-B065

14 de mayo de 2025

Documento técnico

“Modelo para representar comportamientos gravitacionales con dos cuerpos”

Presentan:

José Emiliano Carrillo Barreiro.¹
José Ángel Robles Otero.²

DIRECTORES:

Dr. Cesar Hernández Vasquez
Dr. Mauricio Olguín Carbajal

Resumen:

En los últimos años, la elaboración de entornos virtuales complejos, como *REBOUND*, *Universe sandbox*, *Stellarium* y *Celestia*, han experimentado un crecimiento exponencial en sus capacidades. Más, cuando se habla de implementar mejoras en la precisión y escalabilidad dentro de sistemas de n -cuerpos celestes, por ejemplo, la simulación de nacimientos de galaxias, todavía existen dificultades, ya que, actualmente, los modelos que los simulan no permiten introducir cambios en los parámetros de los cuerpos durante su ejecución, parámetros como la velocidad, posición y masa del cuerpo, lo que impide contar con escenarios más fidedignos a los comportamientos físicos dentro de sistemas de n -cuerpos atípicos. En este proyecto, se propone solucionar este problema mediante la construcción de un modelo que permita realizar cambios al parámetro más significativo del sistema, compuesto por dos cuerpos, en tiempo de ejecución, implementando técnicas como el método multipolar rápido (FMM, por sus siglas en inglés, *Fast Multipole Method*), el *algoritmo de Barnes-Hut* y algoritmos bioinspirados. La solución que se desarrolle podría usarse para simular comportamientos físicos dentro de entornos virtuales.

Palabras clave: Modelado y Simulación de sistemas, problema de los n -cuerpos, Mecánica Celeste, y Algoritmos Bioinspirados.

Fecha: 14 de mayo de 2025

¹carrillobarjosee@gmail.com

²robles.oto.rose.angel@gmail.com

Advertencia

“Este documento contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan.”

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en:

La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

Resumen.

En los últimos años, la elaboración de entornos virtuales complejos, como *REBOUND*, *Universe sandbox*, *Stellarium* y *Celestia*, han experimentado un crecimiento exponencial en sus capacidades. Más, cuando se habla de implementar mejoras en la precisión y escalabilidad dentro de sistemas de n -cuerpos celestes, por ejemplo, la simulación de nacimientos de galaxias, todavía existen dificultades, ya que, actualmente, los modelos que los simulan no permiten introducir cambios en los parámetros de los cuerpos durante su ejecución, parámetros como la velocidad, posición y masa del cuerpo, lo que impide contar con escenarios más fidedignos a los comportamientos físicos dentro de sistemas de n -cuerpos atípicos. En este proyecto, se propone solucionar este problema mediante la construcción de un modelo que permita realizar cambios al parámetro más significativo del sistema, compuesto por dos cuerpos, en tiempo de ejecución, implementando técnicas como el método multipolar rápido (FMM, por sus siglas en inglés, *Fast Multipole Method*), el *algoritmo de Barnes-Hut* y algoritmos bioinspirados. La solución que se desarrolle podría usarse para simular comportamientos físicos dentro de entornos virtuales.

Abstract.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellen-tesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Agradecimientos.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Índice general

Resumen.	iii
Abstract.	iv
Agradecimientos.	v
1. Introducción	1
1.1. Antecedentes	1
1.2. Planteamiento del problema	2
1.3. Propuesta de solución	2
1.4. Objetivo general	3
1.4.1. Objetivos específicos	3
1.5. Justificación	4
2. Estado del Arte	5
2.1. <i>Framework</i> <code>ode_num_int</code>	5
2.1.1. Descripción y Arquitectura Técnica	5
2.1.2. Características y Aplicaciones Prácticas	6
2.1.3. Ventajas y Desventajas	6
2.1.4. Relevancia para el Proyecto de Simulación Gravitacional .	6
2.2. Representación de Objetos	7
2.2.1. Descripción y Arquitectura Técnica	7
2.2.2. Características y Aplicaciones Prácticas	7
2.2.3. Ventajas y Desventajas	8
2.2.4. Relevancia para el Proyecto de Simulación Gravitacional .	8
2.2.5. Experiencia Personal y Opinión	8
2.3. Método n-NNN	8
2.3.1. Descripción y Arquitectura Técnica	9
2.3.2. Características y Aplicaciones Prácticas	9
2.3.3. Ventajas y Desventajas	9
2.3.4. Relevancia para el Proyecto de Simulación Gravitacional .	10
2.3.5. Experiencia Personal y Opinión	10
2.4. Métodos Hidrodinámicos Sin Malla	10
2.4.1. Descripción y Arquitectura Técnica	11
2.4.2. Características y Aplicaciones Prácticas	11
2.4.3. Ventajas y Desventajas	11
2.4.4. Relevancia para el Proyecto de Simulación Gravitacional .	11

2.4.5. Experiencia Personal y Opinión	12
2.5. Método híbrido <i>SPH/N</i> -cuerpos	12
2.5.1. Descripción y Arquitectura Técnica	12
2.5.2. Características y Aplicaciones Prácticas	12
2.5.3. Ventajas y Desventajas	13
2.5.4. Relevancia para el Proyecto de Simulación Gravitacional .	13
2.6. Integrador simpléctico para problemas gravitacionales	14
2.6.1. Descripción y Arquitectura Técnica	14
2.6.2. Características y Aplicaciones Prácticas	14
2.6.3. Ventajas y Desventajas	15
2.6.4. Relevancia para el Proyecto de Simulación Gravitacional .	15
2.7. Solver gravitacional híbrido TPM	15
2.7.1. Descripción y Arquitectura Técnica	15
2.7.2. Aplicaciones Prácticas	16
2.7.3. Ventajas y Desventajas	16
2.7.4. Relevancia para el Proyecto de Simulación Gravitacional .	16
2.8. Algoritmo TPM para simulaciones N-cuerpos	17
2.8.1. Descripción y Arquitectura Técnica	17
2.8.2. Aplicaciones Prácticas	17
2.8.3. Ventajas y Desventajas	18
2.8.4. Relevancia para el Proyecto de Simulación Gravitacional .	18
2.9. REBOUND para dinámica N-cuerpos	18
2.9.1. Descripción y Arquitectura Técnica	19
2.9.2. Aplicaciones Prácticas	19
2.9.3. Ventajas y Desventajas	19
2.9.4. Relevancia para el Proyecto de Simulación Gravitacional .	20
2.10. Modelo de Estabilidad Planetaria	20
2.10.1. Características Principales	20
2.10.2. Ventajas	21
2.10.3. Desventajas y Limitaciones	21
2.10.4. Ámbito de Uso	22
2.10.5. Resultados Clave	22
2.10.6. Relevancia para el Proyecto de Simulación Gravitacional .	23
2.11. Tabla Comparativa del Estado del Arte	23
3. Marco Teórico	26
3.1. Problema de los <i>n</i> cuerpos	26
3.1.1. Puntos clave	26
3.1.2. Principios y funcionamiento	26
3.1.3. Aplicaciones inesperadas	27
3.1.4. Definición y Enfoques	27
3.1.5. Contexto Histórico	27
3.1.6. Principios Fundamentales	27
3.1.7. Variantes y Enfoques Principales	27
3.1.8. Aplicaciones Típicas	28
3.2. Problema de dos cuerpos	28
3.2.1. Definición y Enfoques Conceptuales	28
3.2.2. Principios Fundamentales y Mecanismo de Funcionamiento	28

3.2.3.	Contexto Histórico y Evolución del Problema	29
3.2.4.	Variantes y Enfoques Metodológicos	29
3.2.5.	Aplicaciones Típicas	29
3.2.6.	Ventajas y Limitaciones	30
3.3.	Exponente de Lyapunov	30
3.3.1.	Definiciones de los Exponentes de Lyapunov	30
3.3.2.	Principios Fundamentales y Funcionamiento	31
3.3.3.	Contexto Histórico	31
3.3.4.	Variantes y Conceptos Relacionados	32
3.3.5.	Aplicaciones (basadas en los artículos)	32
3.3.6.	Ventajas y Limitaciones	33
3.4.	Integradores Simplécticos	34
3.4.1.	Definición y Principios Fundamentales	34
3.4.2.	Contexto Histórico y Desarrollo	35
3.4.3.	Variantes y Enfoques Principales	35
3.4.4.	Propiedades Clave	35
3.4.5.	Aplicaciones Típicas	36
3.4.6.	Ventajas	36
3.4.7.	Desventajas y Limitaciones	36
3.4.8.	Desarrollos Recientes y Direcciones Futuras	37
3.5.	Integrador Simpléctico WHFast	37
3.5.1.	Principios Fundamentales y Funcionamiento	38
3.5.2.	Contexto Histórico y Origen	39
3.5.3.	Variantes y Opciones de Configuración	39
3.5.4.	Aplicaciones Típicas	40
3.5.5.	Ventajas y Desventajas/Limitaciones	40
3.6.	Método Multipolar Rápido	41
3.6.1.	Principios Fundamentales y Funcionamiento	41
3.6.2.	Contexto Histórico/Origen	43
3.6.3.	Variantes o Enfoques Principales	44
3.6.4.	Aplicaciones Típicas	44
3.6.5.	Ventajas y Desventajas/Limitaciones	45
3.6.6.	Simulación Barnes-Hut	45
3.7.	REBOUND	49
3.7.1.	Principios Fundamentales y Funcionamiento	49
3.7.2.	Contexto Histórico y Origen	51
3.7.3.	Variantes y Enfoques Principales	51
3.7.4.	Aplicaciones Típicas	51
3.7.5.	Ventajas y Limitaciones	52
3.8.	Programación Matemática	53
3.8.1.	Variantes Principales de la Programación Matemática	54
3.8.2.	Aplicaciones Típicas	56
3.8.3.	Ventajas y Desventajas	57
3.9.	Problemas de Factibilidad y Optimización	58
3.9.1.	Problemas de Optimización	58
3.9.2.	Problemas de Factibilidad	60
3.9.3.	Relación y Contención	62
3.10.	AGs Simples mediante pymoo	63

3.10.1. Introducción a los Algoritmos Genéticos (AGs) Simples	63
3.10.2. Descripción General de la Biblioteca Pymoo	64
3.10.3. El Algoritmo Genético Canónico en Pymoo	66
3.10.4. Análisis Detallado de la Clase <code>pymoo.algorithms.soo.nonconvex.ga.GA</code>	67
3.10.5. Componentes y Operadores Clave	68
3.10.6. Implementación y Uso Práctico	72
3.10.7. Consideraciones Adicionales	74
3.11. FrameworkQt & PyQt	76
3.11.1. Introducción a Qt	76
3.11.2. Principios Fundamentales de Qt	77
3.11.3. Variantes y Licencias de Qt	78
3.11.4. Aplicaciones Típicas de Qt	78
3.11.5. Ventajas y Desventajas de Qt	79
3.11.6. Introducción a PyQt	80
3.11.7. Principios Fundamentales de PyQt	80
3.11.8. Aplicaciones Típicas de PyQt	81
3.11.9. Ventajas y Desventajas de PyQt	81
3.12. Matplotlib	82
3.12.1. Introducción: El Papel de Matplotlib en la Visualización de Datos Científicos	82
3.12.2. Definición, Contexto Histórico y Orígenes	82
3.12.3. Arquitectura y Conceptos Fundamentales	83
3.12.4. Estilos de API: Pyplot vs. Orientada a Objetos (OO)	84
3.12.5. Características Clave y Capacidades de Trazado	84
3.12.6. Aplicaciones Típicas en Computación Científica y Campos Relacionados	84
3.12.7. Ventajas de Matplotlib	85
3.12.8. Limitaciones de Matplotlib	86
3.12.9. Personalización para Figuras con Calidad de Publicación .	86
4. Planeación	87
4.1. Validez por Juicio de Expertos	87
4.1.1. Consulta con experto núm. 01	87
4.1.2. Consulta con experto núm. 02	90
4.2. Análisis de Requerimientos	93
4.2.1. Requerimientos Funcionales (RF)	94
4.2.2. Requerimientos No Funcionales (RNF)	95
4.3. Planteamiento del Problema de Factibilidad	95
4.3.1. Definiciones Preliminares: Variables, Parámetros y Función de Evaluación	96
4.3.2. Definición Formal Matemática del Problema de Factibilidad	97
4.3.3. Abordaje del Problema de Factibilidad mediante Formulación de Optimización	98
4.3.4. Consideraciones Adicionales	99
5. Análisis	100
5.1. Análisis de Procesos Internos del Programa	100

5.1.1. Proceso Interno 01: Captura Parámetros	100
5.1.2. Proceso Interno 02: Inicializar Población	104
5.1.3. Proceso Interno 03: Generar Nueva Generación	107
5.1.4. Proceso Interno 04: Crear Nueva Simulación	111
5.1.5. Proceso Interno 05: Agregar Cuerpos	114
5.1.6. Proceso Interno 06: Definir Condiciones	117
5.1.7. Proceso Interno 07: Iniciar Simulación	120
5.1.8. Proceso Interno 08: Resolver Ecuaciones	123
5.1.9. Proceso Interno 09: Actualizar Estado	127
5.1.10. Proceso Interno 10: Recolectar Datos	132
5.1.11. Proceso Interno 11: Procesar Datos	137
5.1.12. Proceso Interno 12: Generar Gráficos	144
5.1.13. Proceso Interno 13: Evaluar Fitness	149
5.1.14. Proceso Interno 14: Formar Siguiente Población	152
5.1.15. Proceso Interno 15: Mostrar Resultados	155
5.1.16. Matriz de Procesos	159
5.2. Diagrama de Actividades	165
5.2.1. Diagrama de Actividades	166
5.2.2. Descripción del Diagrama de Actividades	167
5.2.3. Aspectos Clave del Diseño Reflejados en el Diagrama	169
5.3. diccionario de datos	169
5.3.1. Relacion de datos	171
6. Diseño	173
6.1. Diseño Arquitectónico	173
6.1.1. Introducción y Objetivos de Diseño	173
6.1.2. Arquitectura en Capas y Componentes Principales	174
6.1.3. Interacciones Principales y Flujos de Datos y Control	178
6.1.4. Estilo y Paradigmas Arquitectónicos	182
6.1.5. Consideraciones Adicionales de Diseño	183
Bibliografía	185

Índice de figuras

3.1. Esquema de la estructura jerárquica de celdas (e.g., quadtree) y la distinción entre interacciones de campo cercano (directas) y campo lejano (aproximadas mediante expansiones), incluyendo la lista de interacción. Adaptado de [46, 51].	43
3.2. Ilustración del flujo de información en el FMM: fuentes originales (\mathbf{q}^σ) → expansión saliente ($\hat{\mathbf{q}}^\sigma$) → expansión entrante ($\hat{\mathbf{v}}^\tau$) → potenciales (\mathbf{v}^τ), mostrando los operadores T^{ofs} , T^{ifo} , y T^{tfi} . Adaptado de [47].	43
3.3. Ilustración del algoritmo de barrido plano. El plano interseca las trayectorias de las partículas 5 y 7. Véanse los detalles en el texto adaptado de [8].	51
3.4. Representación bidimensional de la región factible de un problema de programación lineal, caracterizada por un polígono convexo definido por restricciones lineales. Se incluyen líneas de nivel de la función objetivo y el punto óptimo ubicado en un vértice del conjunto factible, ilustrando la solución típica en problemas lineales. Adaptado de [79, p. 6-7]	55
3.5. Visualización de una función no lineal con múltiples extremos locales y un único óptimo global, empleada para ilustrar las diferencias conceptuales entre mínimos (o máximos) locales y globales en problemas de programación no lineal. Esta distinción es clave en el análisis de algoritmos de optimización y su convergencia. Adaptado de [81, p. 419]	56
3.6. Gráfica 2D que muestre contornos de una función objetivo $f_0(x)$, una región factible sombreada definida por restricciones, y el punto óptimo x^*	59
3.7. Representación bidimensional de un conjunto de restricciones lineales (líneas rectas) que definen una región poliédrica sombreada correspondiente al conjunto factible de un problema de programación lineal (LP). Se ilustran además los contornos paralelos de la función objetivo lineal y la dirección de optimización. Adaptada de [88, p. 101]	61
3.8. Representación bidimensional del conjunto factible, el punto óptimo y la trayectoria central $x^*(t)$ seguida por los métodos de barrera conforme el parámetro t incrementa. Se incluyen curvas de nivel que ilustran la función de barrera o, alternativamente, la función objetivo. Adaptada de [88, p. 361]	62

5.1.	Diagrama de Proceso Interno 01: Captura de Parámetros	103
5.2.	Diagrama de Proceso Interno 02: Inicializar Población	106
5.3.	Diagrama de Proceso Interno 03: Generar Nueva Generación	110
5.4.	Diagrama de Proceso Interno 04: Crear Simulación	113
5.5.	Diagrama de Proceso Interno 05: Agregar Cuerpos	116
5.6.	Diagrama de Proceso Interno 06: Definir Condiciones	119
5.7.	Diagrama de Proceso Interno 07: Iniciar Simulación	122
5.8.	Diagrama de Proceso Interno 08: Resolver Ecuaciones	126
5.9.	Diagrama de Proceso Interno 09: Actualizar Estado	131
5.10.	Diagrama de Proceso Interno 10: Recolectar Datos	136
5.11.	Diagrama de Proceso Interno 12: Generar Gráfico	143
5.12.	Diagrama de Proceso Interno 05: Agregar Cuerpos	148
5.13.	Diagrama de Proceso Interno 13: Calcular Fitness	151
5.14.	Diagrama de Proceso Interno 14: Selección de Individuos	154
5.15.	Diagrama de Proceso Interno 15: Mostrar Resultados	158
5.16.	Diagrama de Actividades UML del proceso de simulación y optimización.	166
6.1.	Diagrama de Arquitectura.	173

Índice de cuadros

2.1. Resumen comparativo de proyectos y artículos relacionados con simulaciones N-cuerpos analizados en el Estado del Arte.	24
3.1. Enfoques en n cuerpos	27
3.2. Principales variantes de integradores simplécticos para problemas de N-cuerpos.	35
3.3. Opciones de Configuración Principales de WHFast en REBOUND [41]	40
3.4. Comparación de complejidad computacional para problemas de N-cuerpos.	45
3.5. Parámetros Clave de la Clase <code>pymoo.algorithms.soo.nonconvex.ga.GA</code>	75
3.6. Estilos de API: Pyplot vs. Orientada a Objetos (OO)	84
3.7. Tipos de Gráficos Comunes en Matplotlib y Aplicaciones Científicas	85
4.1. Requerimientos Funcionales	94
4.2. Requerimientos No Funcionales	95
5.1. Matriz de Procesos	160
5.2. Cuerpo celeste.	170
5.3. Simulación.	171
5.4. Métricas.	171

List of Algorithms

CAPÍTULO 1

Introducción

1.1. Antecedentes

El desarrollo de simulaciones de n -cuerpos ha sido ampliamente investigado con el objetivo de mejorar su capacidad para modelar fenómenos complejos en entornos como la colisión de galaxias y la evolución de sistemas planetarios. Sin embargo, una limitación crítica es la incapacidad de ajustar los parámetros de los cuerpos durante la ejecución de la simulación, lo que afecta la representación precisa de eventos masivos como colisiones en las simulaciones disponibles.

El problema de los n -cuerpos¹ ha sido uno de los desafíos más complejos en la mecánica celeste, desde los trabajos pioneros de Newton en el siglo XVII. Si bien el problema de los dos cuerpos tiene una solución analítica exacta bajo ciertas condiciones, cuando se introducen factores adicionales, se requieren métodos numéricos avanzados para mejorar la precisión de las simulaciones.

Avances como el método multipolar rápido (FMM, por sus siglas en inglés, *Fast Multipole Method*)² y el algoritmo de Barnes-Hut³ han optimizado las simulaciones de n -cuerpos al reducir la complejidad computacional en el cálculo de las interacciones gravitacionales. Sin embargo, la falta de capacidad para ajustar dinámicamente las propiedades de los cuerpos durante la ejecución sigue siendo una limitación en la representación realista de eventos astronómicos.

Estudios recientes han introducido herramientas y enfoques que podrían mejorar las simulaciones de n -cuerpos, como el uso de árboles cuádruples y octales para la representación geométrica y la aplicación de inteligencia artificial en las simulaciones moleculares que podrían inspirar nuevas técnicas en simulaciones celestes. Además, se ha demostrado la eficacia de integradores simplécticos⁴ para

¹El problema de n -cuerpos se refiere al estudio del movimiento e interacción gravitacional de varios cuerpos bajo sus influencias mutuas. En la sección 3.1 se profundiza a fondo el término.

²El FMM es un algoritmo que optimiza el cálculo de interacciones entre partículas distantes en sistemas físicos, como los gravitacionales, agrupando partículas y aproximando su influencia colectiva, lo que reduce la complejidad computacional de $O(n^2)$ a $O(n)$ o $O(n \log n)$.

³El algoritmo de Barnes-Hut es una técnica de simulación de n -cuerpos que aproxima las fuerzas gravitacionales al agrupar cuerpos distantes en una misma región del espacio, representándolos como una única masa, lo que reduce la complejidad computacional de $O(n^2)$ a $O(n \log n)$.

⁴Los integradores simplécticos son métodos numéricos utilizados para resolver ecuaciones

garantizar la estabilidad de las simulaciones durante colisiones y la paralelización para mejorar la eficiencia en simulaciones masivas.

1.2. Planteamiento del problema

Las simulaciones de sistemas de n -cuerpos celestes han sido una herramienta fundamental en la astronomía y la mecánica celeste para estudiar la evolución de sistemas planetarios, la dinámica estelar y la interacción gravitacional a gran escala. Sin embargo, una limitación crítica en los modelos actuales es la imposibilidad de modificar dinámicamente los parámetros de los cuerpos durante la ejecución de la simulación. En la mayoría de los simuladores, estos parámetros, como la masa, la energía del sistema, la posición y el tiempo de existencia de los cuerpos se definen antes de la ejecución, impidiendo la representación precisa de eventos atípicos o transitorios, como colisiones de galaxias, acreción de materia en discos protoplanetarios o cambios de masa en estrellas variables.

Uno de los problemas más importantes derivados de esta limitación es la incapacidad de ajustar la masa de los cuerpos celestes en tiempo real. En eventos como fusiones de agujeros negros o estrellas en procesos de acreción, la masa de los cuerpos cambia significativamente a lo largo del tiempo, alterando la evolución del sistema. Sin la posibilidad de modificar este parámetro de manera dinámica, los modelos actuales ofrecen solo aproximaciones estáticas que no reflejan con precisión la naturaleza de estos fenómenos.

Además, el rendimiento computacional también representa un desafío. La simulación de n -cuerpos es un problema computacionalmente costoso, ya que la cantidad de interacciones a calcular crece cuadráticamente con el número de cuerpos ($O(n^2)$), lo que hace que las simulaciones a gran escala sean prohibitivas en términos de tiempo de ejecución y recursos computacionales. Métodos como el algoritmo de Barnes-Hut y el método multipolar rápido han sido desarrollados para mitigar este problema reduciendo la cantidad de cálculos directos, pero no abordan la falta de flexibilidad en la modificación de parámetros en tiempo real.

En este contexto, la incapacidad de modificar parámetros dinámicamente en simulaciones de n -cuerpos afecta no solo la precisión de la representación de fenómenos astronómicos, sino también la capacidad de realizar simulaciones interactivas en tiempo real, algo que podría tener aplicaciones en la enseñanza, la exploración espacial y la industria del entretenimiento. La ausencia de modelos que permitan esta flexibilidad limita el alcance de las simulaciones actuales y plantea la necesidad de desarrollar enfoques más adaptativos y eficientes.

1.3. Propuesta de solución

La solución propuesta se basa en el desarrollo de un modelo que permite la modificación dinámica de parámetros durante la simulación, lo que representa una mejora significativa con respecto a los simuladores tradicionales, que requieren

diferenciales en sistemas dinámicos conservando las propiedades geométricas del sistema, como la conservación de la energía a largo plazo, lo que los hace especialmente adecuados para simular interacciones físicas, como las gravitacionales, en sistemas de n -cuerpos.

reconfiguraciones antes de cada ejecución. Este modelo integrará técnicas avanzadas para optimizar el cálculo de interacciones gravitacionales y garantizar la estabilidad de la simulación.

Para lograrlo, se empleará el método multipolar rápido (FMM) y el algoritmo de Barnes-Hut, ambos ampliamente utilizados para reducir la complejidad computacional en simulaciones de n -cuerpos sin comprometer la precisión. Estas técnicas permiten agrupar cuerpos en estructuras jerárquicas, disminuyendo el número de cálculos directos y mejorando la eficiencia del sistema.

Además, la implementación de algoritmos bioinspirados, como la optimización por enjambre de partículas (PSO) y algoritmos genéticos, permitirá el ajuste dinámico de los parámetros de los cuerpos celestes, garantizando un comportamiento estable incluso en escenarios de colisión o interacciones complejas. Estas estrategias facilitarán la identificación y ajuste de valores óptimos sin necesidad de intervención manual, lo que hará que la simulación sea más adaptable a eventos inesperados.

El modelo se diseñará para ser escalable, permitiendo la modificación en tiempo real de los parámetros de hasta dos cuerpos celestes, con la posibilidad de extenderse a más cuerpos en implementaciones futuras. Su integración con entornos virtuales como Unreal Engine o motores de simulación especializados permitirá su aplicación en campos como la educación, los videojuegos y la industria aeroespacial.

Para garantizar un rendimiento eficiente, el modelo se desarrollará optimizado para ejecutarse en hardware de gama media, como procesadores multinúcleo con al menos 16 GB de RAM. Esto asegurará que la solución sea accesible sin requerir infraestructuras de alto rendimiento, ampliando su potencial uso en distintos ámbitos.

Esta combinación de técnicas avanzadas y adaptabilidad en tiempo real establece una base innovadora para el desarrollo de simulaciones gravitacionales más dinámicas, eficientes y versátiles, superando las limitaciones de los modelos pre-configurados actuales.

1.4. Objetivo general

Desarrollar un modelo teórico para la simulación del problema de dos cuerpos que permita la modificación dinámica de la masa, mejorando la precisión en la representación de sus interacciones gravitacionales y eventos asociados.

1.4.1. Objetivos específicos

- Desarrollar el módulo de simulación para la integración del método multipolar rápido (FMM) y el algoritmo de Barnes-Hut.
- Desarrollar el módulo de optimización para ajustar dinámicamente la configuración del sistema con algoritmos bioinspirados.
- Desarrollar la implementación del modelo de simulación dinámica de interacciones gravitatorias entre dos cuerpos.

- Desarrollar el módulo de visualización para representar gráficamente la evolución del sistema en un número limitado de iteraciones.
- Desarrollar una interfaz básica para la modificación de parámetros y la visualización de resultados.

1.5. Justificación

Este proyecto es relevante porque introduce un enfoque innovador en la simulación de sistemas gravitacionales al permitir la modificación dinámica de parámetros, superando las limitaciones de los modelos tradicionales. La elección de algoritmos avanzados, como el método multipolar rápido (FMM) y el algoritmo de Barnes-Hut, garantiza un equilibrio entre precisión y eficiencia computacional, lo que permite su aplicación en escenarios más complejos sin un costo computacional excesivo. Además, la implementación de técnicas de optimización bioinspiradas ofrece un método adaptable para ajustar el sistema en tiempo real, mejorando la fidelidad de las simulaciones.

El uso de estas tecnologías no solo optimiza el rendimiento de la simulación, sino que también sienta las bases para su posible escalabilidad a problemas de mayor complejidad, como la simulación de múltiples cuerpos. Aunque el presente trabajo se centra en el problema de dos cuerpos, su enfoque y metodología podrían aplicarse en otros ámbitos, como el modelado de interacciones gravitacionales en videojuegos y simulaciones interactivas.

Los principales beneficiarios de este proyecto serán investigadores y académicos en los campos de la astronomía, astrofísica y mecánica celeste, quienes podrán utilizar el modelo para mejorar el análisis y la comprensión de interacciones gravitacionales. Además, su posible aplicación en videojuegos y simulaciones interactivas podría impactar significativamente la industria del entretenimiento y la educación, proporcionando herramientas más flexibles y precisas para la enseñanza y la creación de simulaciones realistas.

Asimismo, la industria aeroespacial y los organismos encargados de planificar maniobras espaciales podrían beneficiarse del modelo, ya que permitiría simular con mayor precisión eventos como colisiones orbitales o ajustes en la trayectoria de satélites. Al ofrecer la posibilidad de modificar dinámicamente los parámetros de los cuerpos en simulación, esta herramienta facilitaría la planificación y optimización de maniobras espaciales, mejorando la toma de decisiones en misiones reales.

CAPÍTULO 2

Estado del Arte

2.1. Producto 01: *Framework ode_num_int para Integración Numérica*

En el ámbito de la simulación de sistemas gravitacionales, la integración numérica de ecuaciones diferenciales ordinarias (EDOs) desempeña un papel fundamental al describir el movimiento de cuerpos bajo influencias gravitacionales. La precisión y eficiencia de los métodos de integración utilizados impactan directamente la capacidad de los modelos para representar fenómenos físicos complejos, como los comportamientos dinámicos de sistemas de dos cuerpos. En este contexto, el *framework ode_num_int*, presentado en el artículo “C++ Playground for Numerical Integration Method Developers” [1], emerge como una herramienta relevante para el desarrollo y evaluación de métodos de integración numérica personalizados, con potencial aplicabilidad al proyecto descrito en este documento.

2.1.1. Descripción y Arquitectura Técnica

El *framework ode_num_int* es una biblioteca de software desarrollada en C++11, diseñada para proporcionar a investigadores y desarrolladores un entorno flexible y extensible para la creación y prueba de métodos de integración numérica destinados a resolver EDOs. A diferencia de bibliotecas tradicionales como GSL o Boost.Odeint, que priorizan soluciones numéricas predefinidas, *ode_num_int* se centra en el proceso investigativo, ofreciendo una arquitectura modular basada en clases *template*. Esta estructura permite a los usuarios personalizar componentes individuales según las necesidades específicas de sus simulaciones.

La arquitectura del *framework* se organiza en tres pilares principales:

1. Una infraestructura común que incluye el patrón observador para la gestión dinámica de *callbacks*, *holders* de propiedades para el manejo flexible de variables, factorías para la creación de instancias, parámetros opcionales y utilidades de temporización.
2. Componentes de álgebra lineal con *templates* para vectores y matrices dispersas, soporte para factorización *LU* y almacenamiento optimizado de datos.

3. Capacidades de resolución numérica que abarcan *solvers* para sistemas algebraicos no lineales, métodos de iteración tipo Newton, *solvers* explícitos e implícitos para EDOs, manejo de eventos y controladores de tamaño de paso.

Esta modularidad permite combinar diferentes elementos de manera sencilla, facilitando la experimentación con métodos diversos.

2.1.2. Características y Aplicaciones Prácticas

Entre las características distintivas de `ode_num_int` se encuentra su flexibilidad para integrar componentes de *solver*, su monitoreo de rendimiento integrado y su soporte para múltiples tipos de métodos numéricos, incluyendo explícitos, implícitos y basados en extrapolación. Estas capacidades lo hacen adecuado para simulaciones de sistemas físicos complejos. Un ejemplo concreto de su aplicación es la simulación de la dinámica de transmisiones continuamente variables (CVT), un sistema con 3,600 variables de estado que involucra modelos de cuerpos elásticos deformables e interacciones de fricción. En este caso, se observó que el método del trapecio con un tamaño de paso de 10^{-5} alcanzó una precisión comparable a la del método Runge-Kutta de cuarto orden (RK4) con un tamaño de paso de 10^{-8} , lo que indica un potencial de mejora en la eficiencia computacional.

El *framework* se utiliza principalmente en campos como la computación científica, simulaciones de ingeniería, modelado de sistemas mecánicos y análisis de dinámica en aeronáutica y robótica. Su diseño lo posiciona como una herramienta valiosa para investigadores que requieren un alto grado de control sobre los métodos de integración.

2.1.3. Ventajas y Desventajas

El `ode_num_int` ofrece varias ventajas significativas. Su alta extensibilidad permite a los usuarios incorporar nuevos componentes y métodos, mientras que su enfoque en el rendimiento, respaldado por herramientas de monitoreo, facilita la optimización de simulaciones. Además, su carácter *open-source* bajo la licencia GNU GPL fomenta la colaboración y el desarrollo comunitario. Sin embargo, presenta limitaciones notables: está diseñado principalmente para sistemas de escala media, lo que podría restringir su uso en simulaciones de gran escala como las de *n*-cuerpos masivos, y requiere un conocimiento avanzado de C++, lo que puede limitar su accesibilidad. Actualmente, se enfoca en métodos de un solo paso, aunque se planea ampliar esta capacidad en futuras versiones.

2.1.4. Relevancia para el Proyecto de Simulación Gravitacional

En el contexto del trabajo terminal abordado en este reporte, el cual busca desarrollar un modelo para simular comportamientos gravitacionales de dos cuerpos con modificación dinámica de parámetros de posición y velocidad inicial, `ode_num_int` podría desempeñar un papel complementario. La simulación de sistemas gravitacionales requiere integrar las ecuaciones de movimiento, que son EDOs, y combinarlas con técnicas como el Método Multipolo Rápido (FMM) y el algoritmo de Barnes-Hut para calcular fuerzas gravitacionales de manera

eficiente. La flexibilidad de `ode_num_int` para adaptar métodos de integración y su capacidad de monitoreo de rendimiento podrían facilitar la implementación de integradores personalizados que soporten cambios dinámicos en los parámetros durante la ejecución, un objetivo central del proyecto.

2.2. Producto 02: Representación de Objetos mediante *Quadtrees* y *Octrees* de División No Minimal

Dentro del contexto que nos concierne, la simulación de sistemas gravitacionales, la representación eficiente de estructuras geométricas resulta esencial para modelar y visualizar cuerpos celestes y sus interacciones. La precisión en dicha representación influye directamente en la capacidad de los modelos para simular fenómenos físicos complejos, como trayectorias y colisiones en sistemas de dos cuerpos. En este contexto, el artículo “Object Representation by Means of Nonminimal Division *Quadtrees* and *Octrees*” [2] propone una técnica avanzada para la representación de objetos geométricos en dos y tres dimensiones mediante estructuras jerárquicas de datos, con potencial relevancia para el proyecto descrito en este reporte técnico.

2.2.1. Descripción y Arquitectura Técnica

El artículo desarrolla un método innovador para representar objetos poligonales (en 2D) y polihédricos (en 3D) utilizando *quadtrees* y *octrees* de división no minimal. Estas estructuras jerárquicas subdividen recursivamente el espacio en cuadrantes (*quadtree*) u octantes (*octree*), optimizando la representación de geometrías complejas. A diferencia de los métodos tradicionales, este enfoque introduce nodos especializados:

- WHITE: áreas fuera del objeto.
- BLACK: áreas dentro del objeto.
- GRAY: áreas que requieren subdivisión adicional.
- EDGE: nodos con segmentos de borde.
- VERTEX: nodos con vértices.

Esta clasificación permite una representación detallada de los contornos y vértices, facilitando operaciones como la intersección, unión y diferencia, así como la conversión precisa entre representaciones arbóreas y de bordes.

2.2.2. Características y Aplicaciones Prácticas

La técnica destaca por su capacidad para minimizar el uso de memoria mediante una subdivisión no minimal, que evita divisiones innecesarias en regiones uniformes. Los algoritmos propuestos exhiben una complejidad lineal, lo que los hace escalables para objetos en 2D y 3D. Estas propiedades son valiosas en aplicaciones como el modelado geométrico por computadora, sistemas CAD y simulaciones gráficas. En el proyecto de simulación gravitacional de dos cuerpos, este método podría integrarse con técnicas como el Método Multipolar Rápido (FMM) y el

algoritmo de Barnes-Hut, optimizando la representación geométrica de los cuerpos celestes y complementando el cálculo de fuerzas gravitacionales.

2.2.3. Ventajas y Desventajas

Entre las ventajas del método se encuentran la reducción significativa del espacio de memoria, la eficiencia en operaciones booleanas y la precisión en la representación de bordes y vértices gracias a los nodos EDGE y VERTEX. Sin embargo, presenta desafíos, como una mayor complejidad de implementación debido a la necesidad de algoritmos especializados y una posible pérdida de precisión en objetos con detalles extremadamente pequeños, derivada de la naturaleza discreta de la subdivisión.

2.2.4. Relevancia para el Proyecto de Simulación Gravitacional

Aunque el artículo no aborda directamente la simulación de n -cuerpos, su enfoque en la representación geométrica eficiente tiene implicaciones para el proyecto. La modelización precisa de la geometría de dos cuerpos celestes podría mejorar la simulación de interacciones cercanas o colisiones, aspectos clave en el objetivo de ajustar dinámicamente parámetros como masa, velocidad y posición. La eficiencia en memoria y la rapidez en operaciones geométricas podrían integrarse con los algoritmos FMM y Barnes-Hut, potenciando el rendimiento del modelo propuesto.

2.2.5. Experiencia Personal y Opinión

Tras revisar el artículo y explorar conceptualmente sus planteamientos, consideramos que la introducción de nodos especializados representa un avance significativo sobre los *quadtrees* y *octrees* tradicionales, ofreciendo una solución elegante y eficiente para problemas de representación geométrica. Aunque no se implementó directamente el método, la lógica descrita sugiere una alta viabilidad para su uso en simulaciones que demandan precisión geométrica. La complejidad de implementación podría ser un obstáculo, pero su integración con técnicas de simulación gravitacional parece prometedora, especialmente para optimizar el modelado de cuerpos celestes.

2.3. Método de Red de n-Vecinos Más Cercanos (n-NNN) para Simulaciones Moleculares

En la disciplina de las simulaciones de sistemas dinámicos, como los comportamientos gravitacionales de cuerpos celestes descritos en este proyecto, la eficiencia computacional y la precisión física son pilares fundamentales. El proyecto aquí presentado busca modelar interacciones gravitacionales entre dos cuerpos con la capacidad de modificar dinámicamente parámetros como masa, posición y velocidad durante la ejecución, un desafío que requiere métodos innovadores para optimizar recursos sin comprometer la fidelidad física. En este contexto, el artículo “Artificial Intelligent Molecular Dynamics and Hamiltonian Surgery” [3] introduce el método de Red de n-Vecinos Más Cercanos (n-NNN), un enfoque inicialmente diseñado para simulaciones moleculares que ofrece perspectivas valiosas y adaptables al modelado de sistemas gravitacionales.

2.3.1. Descripción y Arquitectura Técnica

El método n-NNN surge como una alternativa a las simulaciones tradicionales que dependen de Hamiltonianos aditivos por pares, los cuales limitan la representación de interacciones complejas de n-cuerpos debido a su alta demanda computacional. En lugar de calcular las fuerzas entre todos los elementos del sistema en cada iteración, n-NNN emplea matrices multidimensionales para almacenar las fuerzas de cada sitio en función de su vecindario local, asemejándose a una red neuronal. Este diseño permite capturar distribuciones espaciales de múltiples cuerpos sin necesidad de incluir el Hamiltoniano completo, reduciendo significativamente la complejidad computacional. Además, la “cirugía Hamiltoniana” propuesta por los autores facilita la selección de términos de orden superior relevantes, ajustando el modelo para mantener la precisión estructural con un número reducido de vecinos considerados.

2.3.2. Características y Aplicaciones Prácticas

El método n-NNN se caracteriza por su capacidad para simular sistemas complejos, como líquidos de Lennard-Jones¹ o sistemas iónicos², con una precisión notable incluso al limitar el número de vecinos analizados. Su independencia respecto al grado de no aditividad permite incorporar términos de múltiples cuerpos sin un incremento exponencial en el tiempo de cálculo, una ventaja clave para sistemas dinámicos. En el contexto de este proyecto, que busca simular interacciones gravitacionales con parámetros modificables, el enfoque n-NNN podría inspirar estrategias para optimizar el cálculo de fuerzas gravitacionales, especialmente en escenarios donde la masa o la posición de los cuerpos cambian en tiempo real. Aunque su aplicación original se centra en dinámicas moleculares, la lógica subyacente es trasladable a sistemas gravitacionales, donde la eficiencia y la adaptabilidad son igualmente críticas.

2.3.3. Ventajas y Desventajas

Entre las ventajas del método n-NNN destacan su escalabilidad a sistemas complejos, la preservación de la precisión estructural con un número reducido de vecinos y la posibilidad de “entrenar” redes aplicables a diversos estados del sistema. Estas características lo convierten en una herramienta prometedora para simulaciones que requieren flexibilidad computacional. Empero, presenta desventajas notables: su validación se ha restringido a sistemas relativamente simples, como líquidos de Lennard-Jones y cloruro de sodio, lo que plantea dudas sobre su generalización a configuraciones más intrincadas. Por otra parte, la precisión depende de una selección cuidadosa del número de vecinos y de la función generadora, lo que podría complicar su implementación en sistemas gravitacionales altamente dinámicos sin un ajuste riguroso.

¹Los líquidos de Lennard-Jones son sistemas modelados en los que las interacciones entre las partículas (átomos o moléculas neutras) se describen mediante el potencial de Lennard-Jones. Para una definición a profundidad, consulte: [4]

²Los sistemas iónicos son aquellos en los que las interacciones predominantes se deben a la atracción electrostática entre iones de carga opuesta. En aras de una definición comprehensiva, consúltese: [5]

2.3.4. Relevancia para el Proyecto de Simulación Gravitacional

El trabajo terminal descrito en este reporte técnico tiene como objetivo desarrollar un modelo que permita ajustar dinámicamente parámetros de dos cuerpos celestes, utilizando técnicas como el Método Multipolar Rápido (FMM) y el algoritmo de Barnes-Hut. Aunque el n-NNN se diseñó para simulaciones moleculares, su enfoque en reducir la complejidad computacional sin sacrificar precisión física resulta altamente relevante. La capacidad de ajustar el modelo mediante “cirugía Hamiltoniana” podría adaptarse para optimizar el cálculo de interacciones gravitacionales, complementando las técnicas propuestas en el proyecto. Por ejemplo, combinar n-NNN con FMM y Barnes-Hut podría mejorar la eficiencia al manejar cambios en la masa o posición de los cuerpos, permitiendo simulaciones más rápidas y adaptativas.

2.3.5. Experiencia Personal y Opinión

Al evaluar conceptualmente el método n-NNN en el marco de este proyecto, no se planea realizar una implementación directa, pero se analizaron sus principios aplicados a simulaciones gravitacionales. Consideramos que el enfoque de Maguire y Woodcock representa un avance significativo en la optimización de simulaciones de sistemas de múltiples cuerpos. Su énfasis en la eficiencia y la flexibilidad sugiere un gran potencial para nuestro modelo, especialmente en la gestión de parámetros dinámicos. La idea de reducir el número de interacciones consideradas sin perder precisión estructural es particularmente atractiva para simulaciones en tiempo real. No obstante, la necesidad de calibrar cuidadosamente los parámetros del método podría ser un obstáculo en sistemas gravitacionales con comportamientos altamente variables, aunque la “cirugía Hamiltoniana” ofrece una vía prometedora para superar esta limitación.

2.4. Implementación de Métodos Hidrodinámicos Sin Malla en PKDGRAV3 para Simulaciones Cosmológicas

Desde la perspectiva de las simulaciones de sistemas dinámicos, como los comportamientos gravitacionales de cuerpos celestes descritos en este proyecto, la eficiencia computacional y la precisión física son pilares fundamentales. El proyecto aquí presentado busca modelar interacciones gravitacionales entre dos cuerpos con la capacidad de modificar dinámicamente parámetros como masa, posición y velocidad durante la ejecución, un desafío que requiere métodos innovadores para optimizar recursos sin comprometer la fidelidad física. En el marco de lo anterior, el artículo “Mesh-free hydrodynamics methods for astrophysical simulations: the PKDGRAV3 code” de I. Alonso Asensio et al [6] presenta la implementación de métodos hidrodinámicos sin malla (*mesh-free*) en PKDGRAV3, un enfoque inicialmente diseñado para simulaciones cosmológicas que ofrece perspectivas valiosas y adaptables al modelado de sistemas gravitacionales.

2.4.1. Descripción y Arquitectura Técnica

El método presentado en el artículo surge como una alternativa a las simulaciones tradicionales que dependen de mallas estructuradas, las cuales limitan la representación de interacciones complejas en sistemas dinámicos debido a su alta demanda computacional. En lugar de calcular las fuerzas entre todos los elementos del sistema en cada iteración mediante una malla global, los métodos sin malla, como *Meshless Finite Mass* (MFM) y *Meshless Finite Volume** (MFV), emplean partículas para discretizar el fluido, resolviendo las ecuaciones hidrodinámicas con un enfoque adaptativo basado en los vecinos más cercanos. Este diseño permite capturar fenómenos complejos, como choques y discontinuidades, sin necesidad de una malla fija, reduciendo significativamente la complejidad computacional. Además, la optimización del código mediante algoritmos de búsqueda de vecinos y paralelización avanzada ajusta el modelo para mantener la precisión física con un manejo eficiente de recursos.

2.4.2. Características y Aplicaciones Prácticas

El método sin malla en PKDGRAV3 se caracteriza por su capacidad para simular sistemas complejos, como la formación de estructuras cosmológicas, con una precisión notable incluso al manejar grandes contrastes de densidad. Su independencia respecto a una malla fija permite incorporar fenómenos dinámicos sin un incremento exponencial en el tiempo de cálculo, una ventaja clave para sistemas variables. En el contexto de este proyecto, que busca simular interacciones gravitacionales con parámetros modificables, el enfoque sin malla podría inspirar estrategias para optimizar el cálculo de fuerzas gravitacionales, especialmente en escenarios donde la masa o la posición de los cuerpos cambian en tiempo real. Aunque su aplicación original se centra en hidrodinámica, la lógica subyacente es trasladable a sistemas gravitacionales, donde la eficiencia y la adaptabilidad son igualmente críticas.

2.4.3. Ventajas y Desventajas

Entre las ventajas del método sin malla destacan su escalabilidad a sistemas complejos, la preservación de la precisión física con una resolución adaptativa y la posibilidad de manejar simulaciones a gran escala mediante optimizaciones paralelas. Estas características lo convierten en una herramienta prometedora para simulaciones que requieren flexibilidad computacional. Sin embargo, presenta desventajas notables: su validación se ha restringido a sistemas cosmológicos específicos, lo que plantea dudas sobre su generalización a configuraciones más intrincadas. Adicionalmente, la precisión depende de una selección cuidadosa de parámetros computacionales, lo que podría complicar su implementación en sistemas gravitacionales altamente dinámicos sin un ajuste riguroso.

2.4.4. Relevancia para el Proyecto de Simulación Gravitacional

El proyecto descrito en este reporte técnico tiene como objetivo desarrollar un modelo que permita ajustar dinámicamente parámetros de dos cuerpos celestes, utilizando técnicas como el Método Multipolar Rápido (FMM) y el algoritmo de Barnes-Hut. Aunque los métodos sin malla se diseñaron para simulaciones hidrodinámicas, su enfoque en reducir la complejidad computacional sin sacrificar

precisión física resulta altamente relevante. La capacidad de ajustar el modelo mediante una discretización flexible podría adaptarse para optimizar el cálculo de interacciones gravitacionales, complementando las técnicas propuestas en el proyecto. Por ejemplo, combinar MFM y MFV con FMM y Barnes-Hut podría mejorar la eficiencia al manejar cambios en la masa o posición de los cuerpos, permitiendo simulaciones más rápidas y adaptativas.

2.4.5. Experiencia Personal y Opinión

Al evaluar conceptualmente el método sin malla en el marco de este proyecto, no se realizó una implementación directa, pero se analizaron sus principios aplicados a simulaciones gravitacionales. Inferimos que el enfoque de Alonso Asensio et al. representa un avance significativo en la optimización de simulaciones de sistemas de múltiples cuerpos. Su énfasis en la eficiencia y la flexibilidad sugiere un gran potencial para nuestro modelo, especialmente en la gestión de parámetros dinámicos. La idea de reducir la dependencia de estructuras fijas sin perder precisión física es particularmente atractiva para simulaciones en tiempo real. No obstante, la necesidad de calibrar cuidadosamente los parámetros del método podría ser un obstáculo en sistemas gravitacionales con comportamientos altamente variables, aunque las optimizaciones presentadas ofrecen una vía prometedora para superar esta limitación.

2.5. Método híbrido *SPH/N*-cuerpos para simulaciones de cúmulos estelares

2.5.1. Descripción y Arquitectura Técnica

El artículo presenta un enfoque híbrido que combina *Smoothed Particle Hydrodynamics* (SPH) con simulaciones *n*-cuerpos para modelar la evolución de cúmulos estelares jóvenes inmersos en un medio gaseoso. Tradicionalmente, las simulaciones astrofísicas han separado estos dos componentes: las simulaciones SPH permiten modelar la evolución del gas, mientras que las simulaciones *n*-cuerpos se enfocan en la interacción gravitacional entre estrellas. Sin embargo, esta separación limita la capacidad de capturar la retroalimentación entre el gas y las estrellas.

Este nuevo esquema híbrido busca cerrar esta brecha al combinar ambos enfoques dentro del código *SEREN*, un software de simulación hidrodinámica basado en partículas. Mediante la formulación de las ecuaciones de movimiento desde una perspectiva Lagrangiana, se garantiza la conservación de energía y momento, algo esencial para la precisión de las simulaciones.

2.5.2. Características y Aplicaciones Prácticas

Una de las claves del modelo es el uso del árbol de gravedad Barnes-Hut para calcular las fuerzas gravitacionales de todas las partículas gaseosas autogravitantes. Esta estructura de datos permite resolver la dinámica gravitacional de manera eficiente, reduciendo la complejidad computacional en comparación con cálculos directos de fuerza.

Para mejorar la precisión en el cálculo de la gravedad, se emplea el criterio de

apertura multipolar (MAC), en lugar del estándar basado en el ángulo de apertura geométrica. Este criterio minimiza los errores en el cálculo de la fuerza gravitatoria, lo que a su vez reduce los errores en la conservación de energía.

Otro aspecto relevante es la incorporación del cálculo del jerk gravitacional, es decir, la derivada temporal de la aceleración. Mientras que para las partículas SPH solo se calcula la aceleración, para las estrellas se computa tanto la aceleración como el jerk, permitiendo una integración más precisa de sus órbitas.

Además, el código usa un esquema de integración por bloques de tiempo (block timestepping), que ajusta dinámicamente los pasos de integración de cada partícula según su estado dinámico. Las partículas con interacciones más fuertes o aceleraciones altas pueden usar pasos de tiempo más cortos, mientras que aquellas en regiones más estables pueden emplear pasos más largos, optimizando así el rendimiento computacional.

2.5.3. Ventajas y Desventajas

El método híbrido SPH/N-body presenta varias ventajas clave. En primer lugar, proporciona una mayor precisión en la interacción entre el gas y las estrellas, al combinar ambos métodos en una estructura conservativa, eliminando la separación artificial entre estos componentes. Su eficiencia computacional se ve optimizada por el uso del árbol de gravedad Barnes-Hut y el esquema de integración por block timestepping, lo que permite realizar simulaciones más rápidas sin sacrificar la precisión. Además, el criterio MAC mejora significativamente la estimación de la fuerza gravitacional, reduciendo errores numéricos acumulativos y garantizando una mejor conservación de la energía. También ofrece flexibilidad en la resolución, al establecer criterios que equilibran la fidelidad física con los costos computacionales.

Sin embargo, el método también tiene algunas desventajas y limitaciones. Aunque más eficiente que las simulaciones full-SPH, sigue siendo más costoso que una simulación N-body pura. Además, no incluye un tratamiento avanzado de sistemas binarios, lo cual es fundamental para la evolución de cúmulos estelares jóvenes. Finalmente, la precisión de la simulación depende en gran medida de la correcta elección de los parámetros numéricos, como los criterios de resolución y de apertura del árbol de gravedad, lo que puede afectar la exactitud de los resultados si no se configuran adecuadamente.

2.5.4. Relevancia para el Proyecto de Simulación Gravitacional

El método híbrido SPH/N-body presentado en el artículo tiene una gran relevancia para proyectos de simulación gravitacional que buscan ajustar dinámicamente los parámetros de interacción entre dos cuerpos celestes. En particular, su uso del árbol de gravedad Barnes-Hut y la optimización mediante el criterio de apertura multipolar (MAC) proporcionan un marco eficiente para la resolución de fuerzas gravitacionales en sistemas con múltiples escalas de interacción. Además, la integración de estos métodos con enfoques más avanzados, como el Método Multipolar Rápido (FMM), permitiría reducir aún más la complejidad computacional al calcular interacciones gravitacionales con precisión ajustable.

Por otro lado, la combinación de estas técnicas con algoritmos bioinspirados, como algoritmos evolutivos o enjambre de partículas, abre la posibilidad de optimizar parámetros clave en la simulación, como la distribución de masa. Estos algoritmos pueden explorar de manera adaptativa diferentes configuraciones iniciales y estrategias de integración temporal, maximizando la eficiencia y precisión de la simulación. En este sentido, la metodología híbrida discutida en el artículo proporciona un punto de partida sólido para desarrollar simulaciones dinámicas en las que los parámetros de interacción entre cuerpos celestes evolucionan de manera autónoma en función de condiciones físicas y computacionales óptimas.

2.6. Integrador simpléctico para problemas gravitacionales N-cuerpos colisionables

En el artículo se muestra un nuevo integrador simpléctico diseñado específicamente para problemas gravitacionales N-cuerpos con colisiones. El integrador se inspira en el método no simpléctico y no reversible SAKURA, desarrollado por Gonçalves Ferrari, pero introduce modificaciones clave que permiten conservar las propiedades simplécticas y de reversibilidad temporal. El enfoque que se tiene descompone el problema N-cuerpos en múltiples problemas de dos cuerpos utilizando solucionadores de Kepler.

2.6.1. Descripción y Arquitectura Técnica

El integrador propuesto es de segundo orden, reversible en el tiempo y conserva nueve integrales del movimiento del problema N-cuerpos con precisión de máquina. Está basado en una estructura simpléctica, lo que garantiza una evolución temporal coherente con la física del sistema, preservando la estructura de fase a largo plazo.

Aunque el método base es de segundo orden, su precisión puede incrementarse mediante el esquema de composición de Yoshida, sin comprometer las propiedades simplécticas del integrador.

Durante todas las pruebas numéricas presentadas en el artículo, se utilizó un paso de tiempo fijo para mantener exactamente la simplécticidad. Esta elección evita los problemas típicos de los pasos adaptativos, como la pérdida de reversibilidad temporal y la degradación de la conservación de cantidades físicas.

2.6.2. Características y Aplicaciones Prácticas

El integrador fue evaluado en escenarios con bajo número de cuerpos y colisiones frecuentes, situaciones desafiantes para los métodos tradicionales. En estas pruebas, el nuevo integrador mostró un rendimiento comparable o superior al del método Hermite de cuarto orden, ampliamente utilizado en dinámica estelar. También se realizaron comparaciones con integradores simplécticos de segundo orden, superándolos en precisión y estabilidad, especialmente en contextos colisionables.

Estas propiedades lo convierten en una herramienta eficaz para el estudio de cúmulos estelares densos, núcleos galácticos activos, discos protoplanetarios y otros sistemas donde las interacciones gravitacionales colisionables juegan un papel central. Su estructura conservativa lo hace especialmente adecuado para

simulaciones a largo plazo donde se requiere estabilidad numérica y precisión en la conservación de cantidades físicas.

2.6.3. Ventajas y Desventajas

Entre las ventajas del nuevo integrador destacan su capacidad para preservar la estructura simpléctica incluso en presencia de colisiones, conservar integrales del movimiento con alta precisión, y su posibilidad de ser extendido a órdenes superiores. Además, ha mostrado un rendimiento superior frente a integradores ampliamente usados, especialmente en escenarios colisionables. Como desventaja principal, el integrador requiere el uso de un paso de tiempo fijo para mantener su estructura simpléctica exacta, lo que puede limitar su flexibilidad en contextos donde el uso de pasos adaptativos resulta beneficioso. Asimismo, su complejidad estructural podría representar un mayor costo computacional frente a métodos más simples, especialmente en simulaciones de gran escala sin optimización.

2.6.4. Relevancia para el Proyecto de Simulación Gravitacional

Este integrador simpléctico proporciona un marco robusto para estudios donde las colisiones no pueden ser ignoradas. Su estructura conservativa y reversible lo hace especialmente apto para simulaciones a largo plazo, donde es esencial preservar las propiedades físicas del sistema.

Además, su rendimiento superior frente a integradores estándar lo posiciona como una alternativa eficaz y precisa para estudios computacionales intensivos.

2.7. Solver gravitacional híbrido TPM para simulaciones de N-cuerpos en arquitecturas paralelas

Se presenta un nuevo solucionador gravitacional híbrido, denominado TPM (*Tree-Particle-Mesh*), que combina las ventajas del método de partículas en malla (*Particle-Mesh*, PM) y de técnicas jerárquicas basadas en árboles (*TREE methods*). Este enfoque está especialmente diseñado para arquitecturas computacionales paralelas y permite realizar simulaciones de dinámica gravitacional con decenas de millones de partículas, manteniendo un balance óptimo entre precisión y eficiencia.

2.7.1. Descripción y Arquitectura Técnica

El método TPM asigna dinámicamente el tipo de tratamiento a cada partícula dependiendo de la densidad local del entorno: las partículas ubicadas en regiones de baja densidad son tratadas como partículas PM, mientras que las partículas en zonas sobredensas son tratadas como partículas TREE. Las fuerzas de largo alcance se calculan utilizando el método PM, eficiente para todo el volumen simulado, mientras que las fuerzas de corto alcance —específicas de las regiones sobredensas— se calculan utilizando estructuras jerárquicas tipo TREE.

En este esquema, la fuerza sobre una partícula TREE es la suma de dos contribu-

ciones: una fuerza externa, proveniente de partículas fuera del árbol, calculada con el método PM; y una fuerza interna, debida a las partículas dentro del árbol, calculada con el método TREE. Esta separación permite una alta resolución espacial en las regiones más dinámicamente complejas, sin sacrificar eficiencia en el resto del dominio.

La implementación aprovecha arquitecturas paralelas modernas como IBM SP1, SGI Challenge y clústeres de estaciones de trabajo. El código está diseñado para distribuir tanto la carga computacional como la memoria, logrando una escalabilidad casi lineal. En una IBM SP2 con 32 procesadores, se alcanzó una eficiencia del 80%

2.7.2. Aplicaciones Prácticas

El solver TPM es especialmente útil en simulaciones de formación de estructuras cósmicas, evolución de cúmulos de galaxias y formación de galaxias, donde coexisten regiones de muy diferentes densidades y escalas dinámicas. Su capacidad para diferenciar el tratamiento de partículas según su entorno lo convierte en una herramienta adecuada para estudios donde se requiere alta resolución local sin comprometer el rendimiento global. También es aplicable en estudios de materia oscura, colisiones de halos, y evolución de subestructuras dentro de grandes volúmenes cosmológicos.

2.7.3. Ventajas y Desventajas

El principal beneficio del método TPM es su capacidad para combinar la eficiencia del método PM con la resolución espacial del método TREE, lo que permite realizar simulaciones masivas con un uso óptimo de los recursos computacionales. La integración temporal con múltiples escalas añade precisión sin afectar negativamente el rendimiento global. Además, su diseño orientado a arquitecturas paralelas modernas permite su uso en supercomputadoras y clústeres de alto rendimiento.

Entre las desventajas, se encuentra la complejidad adicional en la implementación y mantenimiento del código, especialmente en lo relativo a la sincronización de tiempos y la gestión de zonas frontera entre regiones PM y TREE. Además, aunque el balance entre eficiencia y precisión es muy favorable, en ciertos escenarios altamente dinámicos puede requerirse una cuidadosa sintonización de parámetros para evitar pérdidas de precisión o sobrecostos computacionales.

2.7.4. Relevancia para el Proyecto de Simulación Gravitacional

El enfoque híbrido TPM es especialmente relevante para proyectos que requieren simulaciones multiescala de sistemas gravitacionales. Al permitir un tratamiento diferenciado de las regiones según su densidad, el método ofrece una solución eficaz para abordar la evolución tanto de estructuras globales como de subestructuras internas con alto nivel de detalle.

Además, su compatibilidad con plataformas de alto rendimiento lo hace ideal para futuras generaciones de simulaciones astrofísicas. Su arquitectura modular también

permite incorporar técnicas avanzadas como algoritmos bioinspirados o esquemas adaptativos para optimizar la asignación dinámica de recursos computacionales.

2.8. Algoritmo Tree Particle-Mesh (TPM) para simulaciones N-cuerpos cosmológicas

El algoritmo *Tree Particle-Mesh* (TPM) combina dos enfoques clásicos en dinámica gravitacional: el método de árbol (*Tree*) para el cálculo directo de fuerzas en agrupaciones jerárquicas de partículas, y el método de partículas en malla (*Particle-Mesh, PM*), altamente eficiente para computar fuerzas gravitacionales en grandes volúmenes espaciales. Esta integración permite un balance efectivo entre precisión y eficiencia computacional, siendo especialmente útil en simulaciones cosmológicas con millones o miles de millones de partículas.

2.8.1. Descripción y Arquitectura Técnica

El algoritmo TPM aprovecha la linealidad de las fuerzas gravitacionales respecto al campo de densidad para aplicar una descomposición de dominio basada en la distribución de densidad. De esta manera, el campo se divide en múltiples regiones de alta densidad espacialmente localizadas, que contienen una fracción significativa de la masa total pero ocupan un volumen reducido.

En estas regiones densas, la fuerza gravitacional se calcula mediante el algoritmo de árbol, complementado por fuerzas de marea provenientes del resto del dominio. Para el resto del volumen, donde la densidad es baja, se emplea el algoritmo PM con pasos de tiempo más amplios, optimizando así el uso de recursos.

El uso de diferentes escalas de tiempo —pasos grandes para el componente PM y pasos adaptativos más cortos para las regiones árbol— permite una integración eficiente y precisa de la dinámica de partículas. Además, debido a la independencia de las regiones árbol, el algoritmo puede ser fácilmente paralelizado usando técnicas de paso de mensajes (*message passing*), haciéndolo ideal para entornos de cómputo distribuido.

2.8.2. Aplicaciones Prácticas

El algoritmo TPM ha sido ampliamente utilizado en simulaciones cosmológicas centradas en la formación de estructuras a gran escala, como cúmulos de galaxias, halos de materia oscura y la red cósmica de filamentos y vacíos. Su diseño lo hace especialmente útil para estudios que requieren alta resolución en regiones localizadas de alta densidad, sin comprometer la eficiencia global del cómputo.

Además, el enfoque del TPM también puede extenderse a otros dominios fuera de la cosmología, siempre que la distribución de densidad permita segmentar el dominio en grupos relativamente aislados de partículas. Esto lo hace aplicable en problemas de dinámica de sistemas granulares, plasma, o simulaciones de fluidos donde existan regiones con alta concentración de materia rodeadas de volúmenes menos densos.

2.8.3. Ventajas y Desventajas

Entre las principales ventajas del algoritmo TPM se encuentra su capacidad de balancear de manera eficiente la resolución espacial con el uso de recursos computacionales. La segmentación del espacio según la densidad permite asignar mayor detalle a regiones físicamente relevantes, mientras que el uso del método PM en zonas menos densas conserva la eficiencia. Asimismo, la posibilidad de paralelizar de forma natural las regiones árbol mediante paso de mensajes lo convierte en una herramienta escalable para supercomputación.

Sin embargo, también existen algunas desventajas. La implementación del TPM puede ser más compleja que la de otros algoritmos más homogéneos, como P3M o solo PM, debido al manejo de múltiples escalas temporales y dominios dinámicamente segmentados. Además, en situaciones donde la densidad no presenta fuertes contrastes o donde las regiones densas no están bien aisladas, el beneficio del algoritmo puede verse reducido, y otros métodos podrían ser más apropiados.

2.8.4. Relevancia para el Proyecto de Simulación Gravitacional

La estrategia híbrida del algoritmo TPM es altamente relevante para proyectos que requieren combinar escalabilidad con fidelidad física. Su capacidad de segmentar dinámicamente el dominio espacial según la densidad permite aplicar distintos niveles de resolución según la necesidad, lo cual es ideal para simulaciones adaptativas.

Además, su compatibilidad con computación paralela y su menor tiempo de ejecución respecto a algoritmos tradicionales como P3M lo hacen especialmente útil para integrar con métodos de optimización, análisis masivo de datos simulados o incluso para alimentar sistemas de aprendizaje automático que requieran datos de alta resolución espacial en regiones seleccionadas del espacio simulado.

2.9. Código REBOUND para dinámica N-cuerpos colisionante y no colisionante

REBOUND es un código N-cuerpos de propósito general, de código abierto, diseñado inicialmente para estudiar dinámica colisionante en contextos como anillos planetarios, pero con capacidades completas para resolver también el problema clásico de dinámica N-cuerpos. Su arquitectura altamente modular permite adaptar el código fácilmente a distintos problemas en astrofísica, incluyendo dinámica planetaria, formación de sistemas estelares y simulaciones cosmológicas a pequeña escala.

Este código y sus precursores han sido ya utilizados en una amplia variedad de publicaciones científicas, y continúa en desarrollo activo. A la fecha, no existe otro código de dinámica colisionante de acceso público que ofrezca la misma versatilidad para los tipos de problemas abordados, motivo por el cual ha sido liberado bajo la licencia de código abierto GPLv3.

2.9.1. Descripción y Arquitectura Técnica

El diseño de REBOUND se basa en una estructura modular que facilita la combinación de diferentes métodos numéricos para integración temporal, cálculo gravitacional, detección de colisiones y manejo de condiciones de frontera. Incluye tres integradores simplécticos: *leap-frog*, el integrador simpléctico epicíclico (SEI), y el método de mapeo de Wisdom-Holman (WH), permitiendo así cubrir una amplia gama de escalas temporales y niveles de precisión.

El código soporta condiciones de frontera abiertas, periódicas y del tipo *shearing-sheet*, lo que lo hace ideal para simular sistemas que requieren modelado de geometrías específicas o entornos en rotación diferencial. Esta última condición es particularmente útil en simulaciones de anillos planetarios, que suelen modelarse con esta aproximación, y donde la gravedad propia desempeña un papel clave, especialmente en regiones densas como los anillos de Saturno.

Para el cálculo de la auto-gravedad y las colisiones, REBOUND implementa el algoritmo de Barnes-Hut basado en árboles, con soporte completo para paralelización mediante MPI (paso de mensajes entre nodos) y OpenMP (multi-hilo compartido). La paralelización se logra a través de una descomposición estática del dominio y el uso de árboles distribuidos esenciales.

2.9.2. Aplicaciones Prácticas

REBOUND ha sido ampliamente utilizado en el estudio de anillos planetarios, donde los tiempos de colisión pueden ser comparables o incluso menores que los tiempos orbitales. También se emplea en simulaciones de formación de planetesimales, donde las colisiones entre cuerpos actúan como mecanismo disipativo que facilita el colapso gravitacional. Otra aplicación importante es en discos protoplanetarios, de transición y de escombros, donde se puede utilizar una versión estadística con partículas representativas (*super-particles*) para estimar frecuencias de colisión sin simular cada evento individual.

Además, el código permite estudiar sistemas completamente no colisionantes, como cúmulos estelares o sistemas planetarios en configuración estable, gracias a su capacidad para utilizar integradores simplécticos y seguir con precisión tanto partículas de prueba como cuerpos masivos. La combinación de estos enfoques lo hace adecuado también para estudios híbridos o multiescala.

2.9.3. Ventajas y Desventajas

Entre sus principales ventajas, REBOUND destaca por su arquitectura modular y extensible, la disponibilidad de múltiples integradores adecuados para diferentes escalas de tiempo, y su capacidad para manejar tanto colisiones físicas como dinámicas puramente gravitacionales. La paralelización mediante MPI y OpenMP permite escalar el código desde computadoras personales hasta supercomputadoras. Además, sus algoritmos de detección de colisiones tipo *plane-sweep* ofrecen un rendimiento sobresaliente en geometrías cuasi-bidimensionales.

Sin embargo, el código también presenta algunas limitaciones. Aunque su diseño modular permite gran flexibilidad, esto puede aumentar la complejidad inicial para usuarios novatos. Además, en simulaciones con geometrías tridimensionales

muy complejas o con más de varios millones de partículas, el rendimiento puede verse limitado por la sobrecarga de comunicación entre nodos. Asimismo, ciertas aplicaciones especializadas pueden requerir la implementación de módulos propios, lo que implica un conocimiento más profundo del código fuente.

2.9.4. Relevancia para el Proyecto de Simulación Gravitacional

El código REBOUND representa una herramienta versátil y poderosa para proyectos de simulación gravitacional que requieren modelar tanto interacciones suaves como encuentros cercanos o colisiones físicas entre cuerpos. Su diseño modular permite incorporar algoritmos adicionales, como optimizadores bioinspirados o métodos adaptativos de integración, facilitando la exploración de configuraciones dinámicas complejas.

Además, su capacidad para ejecutarse eficientemente en una variedad de entornos computacionales, junto con su soporte para distintos esquemas de frontera, lo hacen ideal para experimentar con nuevos esquemas híbridos o sistemas autoajustables donde los parámetros de interacción se modifiquen en función del entorno dinámico.

2.10. Modelo de Simulación de Estabilidad Planetaria Basado en Uniformidad de Masas

Dentro de los trabajos recientes que abordan la estabilidad dinámica en sistemas planetarios y su relación con las características intrínsecas del sistema, destaca el enfoque presentado por Dong-Hong Wu, Sheng Jin y Jason H. Steffen en su artículo “Enhanced Stability in Planetary Systems with Similar Masses”, publicado en *The Astronomical Journal* [7]. Este trabajo se centra en investigar cuantitativamente la conexión entre la estabilidad dinámica a largo plazo de sistemas multiplanetarios y la uniformidad de las masas de los planetas que los componen. Su propósito principal es explorar, mediante simulaciones numéricas de N-cuerpos, si los sistemas planetarios con masas más similares (mayor uniformidad) exhiben intrínsecamente una mayor estabilidad, ofreciendo así una posible explicación basada en sesgos de supervivencia para la tendencia observada de “guisantes en una vaina” (*peas-in-a-pod*) en los sistemas detectados por la misión Kepler.

2.10.1. Características Principales

El modelo computacional descrito en [7] se basa fundamentalmente en el código N-cuerpos *REBOUND* [8], utilizando específicamente el integrador híbrido *Mercurius* [9], adecuado para simulaciones de larga duración que involucran posibles encuentros cercanos. Las simulaciones parten de sistemas idealizados compuestos por ocho planetas orbitando una estrella de masa solar. Las condiciones iniciales establecen órbitas circulares y coplanares, una configuración común en muchos sistemas Kepler. La masa total de los planetas se mantiene constante en 30 masas terrestres (M_{\oplus}) en la mayoría de las simulaciones, explorando también casos con 10 M_{\oplus} y 100 M_{\oplus} .

Un parámetro clave es la separación mutua entre planetas adyacentes, K , medida

en unidades del radio de Hill mutuo (Ecuaciones 1 y 2 en [7]), que se mantiene constante dentro de cada sistema simulado y se varía entre 3 y 10 en el conjunto de experimentos. Para cuantificar la uniformidad de masas planetarias, los autores emplean el índice de Gini ajustado (\mathcal{G}_m , Ecuación 3 en [7]), donde $\mathcal{G}_m = 0$ representa masas idénticas y \mathcal{G}_m cercano a 1 indica que una sola masa domina el sistema. Se generan conjuntos de sistemas con valores de \mathcal{G}_m distribuidos uniformemente entre 0 y aproximadamente 0.97 para cada valor de K estudiado.

Las simulaciones se integran hasta que ocurre un “encuentro cercano”, definido como una separación entre dos planetas menor que el radio de Hill del planeta más interno, o hasta alcanzar un tiempo máximo de integración de 10^8 años. El tiempo hasta este evento se registra como el “tiempo de inestabilidad dinámica” (t), que luego se escala por el período orbital del planeta más interno (t_0). El análisis principal se enfoca en la relación entre t/t_0 y \mathcal{G}_m para diferentes valores de K , comparando sistemas con masas desiguales frente a sistemas de control con masas iguales. Investigaciones adicionales consideran sistemas “bien ordenados”, donde las masas planetarias aumentan monotónicamente hacia el exterior. **Es fundamental señalar que, según la descripción metodológica en [7], el modelo no contempla la modificación de parámetros de los cuerpos (masa, posición, velocidad) durante el tiempo de ejecución de la simulación; la evolución se sigue pasivamente desde las condiciones iniciales hasta el punto de inestabilidad.**

2.10.2. Ventajas

El estudio [7] establece una correlación estadísticamente significativa entre la uniformidad de masas y la estabilidad del sistema para configuraciones no resonantes (específicamente para $K > 4$), proporcionando una fuerte evidencia cuantitativa para la hipótesis de que los sistemas más uniformes son inherentemente más estables. Esta es una contribución importante para explicar las arquitecturas observadas en exoplanetas. El uso del índice de Gini ajustado (\mathcal{G}_m) como métrica de uniformidad es una elección metodológica clara y efectiva para comparar diversas distribuciones de masa. La implementación sobre *REBOUND* con el integrador *Mercurius* sugiere una base computacionalmente eficiente, capaz de manejar las escalas de tiempo prolongadas requeridas para estos estudios de estabilidad dinámica. Además, el trabajo aborda explícitamente el papel de las resonancias de movimiento medio (MMRs), identificando cómo estas pueden alterar la tendencia general de estabilidad, como se visualiza en la Figura 1 de [7]. Los autores también proponen un mecanismo físico plausible para la menor estabilidad de sistemas no uniformes: la equipartición de la energía aleatoria, que tiende a excitar las excentricidades de los planetas de menor masa, conduciendo a encuentros cercanos más tempranos.

2.10.3. Desventajas y Limitaciones

Una limitación explícita reconocida por los autores [7] es que la simulación se detiene en el *primer* encuentro cercano detectado. Esto impide el estudio de la evolución dinámica posterior, que podría incluir colisiones planetarias, eyecciones, o incluso la reconfiguración hacia un estado estable diferente. Este criterio de parada simplifica la dinámica post-inestabilidad, que puede ser crucial en la

formación final de la arquitectura del sistema.

La limitación más relevante desde la perspectiva de nuestro proyecto principal, inferida directamente de la metodología descrita en [7], es la **ausencia total de capacidad para modificar los parámetros de los cuerpos celestes (masa, velocidad, posición)** durante la ejecución de la simulación. El modelo opera bajo un paradigma de “condiciones iniciales fijas”, evolucionando el sistema pasivamente. Esto lo hace fundamentalmente inadecuado para escenarios que requieran simular eventos dinámicos interactivos o cambios paramétricos en tiempo real, como la acreción de masa, efectos de fuerzas no conservativas variables, o la introducción/eliminación de cuerpos, que son objetivos centrales de nuestro trabajo.

Otras limitaciones incluyen una menor sensibilidad del tiempo de inestabilidad al índice G_m para valores bajos de K ($K \leq 4$), donde la alta densidad de MMRs domina la dinámica (como se observa en la Figura 1 y 2 de [7]). Además, aunque se exploran sistemas “bien ordenados”, el enfoque principal en distribuciones aleatorias o perfectamente iguales podría no capturar toda la diversidad de configuraciones iniciales posibles post-formación. Finalmente, los propios autores señalan en su discusión [7] que, si bien sus resultados coinciden parcialmente con las observaciones de pares resonantes ($G_m < 0.38$), no pueden explicar completamente por qué estos pares observados tienden a ser *más* uniformes que los pares no resonantes, sugiriendo que la dinámica simulada (posiblemente debido a la detención temprana o la falta de modelado de colisiones) podría estar incompleta.

2.10.4. Ámbito de Uso

El modelo y las simulaciones presentadas en [7] están diseñados específicamente para la investigación teórica de la estabilidad dinámica a largo plazo de sistemas planetarios multi-cuerpo, con un enfoque particular en arquitecturas compactas y coplanares similares a las descubiertas por Kepler. Su utilidad reside en estudios estadísticos para entender cómo las propiedades intrínsecas del sistema (separación, uniformidad de masa) influyen en las tasas de supervivencia y, por ende, en las características observables de la población de exoplanetas. Es una herramienta para probar hipótesis sobre la evolución dinámica pasiva y los sesgos de observación.

2.10.5. Resultados Clave

Los hallazgos cuantitativos principales incluyen:

- Para sistemas non-resonantes y $K > 4$, existe una fuerte anticorrelación entre el logaritmo del tiempo de inestabilidad ($\log(t/t_0)$) y el índice de Gini (G_m). Sistemas con mayor uniformidad (menor G_m) son significativamente más estables. Por ejemplo, para $K=6.5$, el coeficiente de correlación de Spearman es de -0.69 con $p < 10^{-4}$ (Figura 2 en [7]).
- Los sistemas con masas desiguales son generalmente menos estables que sus contrapartes de masas iguales para el mismo K (siempre que no estén dominados por MMRs específicas). La diferencia en tiempo de estabilidad puede ser de varios órdenes de magnitud (Figura 1 en [7]).

- Cerca de MMRs de primer orden (ej., 4:3, 5:4), la relación entre estabilidad y \mathcal{G}_m es más compleja. En algunos casos (como cerca de 5:4, $K=7.5$), la anticorrelación casi desaparece, mientras que en otros (como cerca de 4:3, $K=9.73$) persiste pero es más débil (Figura 3 en [7]).
- Para sistemas “bien ordenados” cerca de la MMR 5:4 ($K \approx 7.5$), la máxima estabilidad se alcanza para un rango intermedio de \mathcal{G}_m ($0.2 < G_m < 0.4$), lo cual difiere del comportamiento en sistemas con masas distribuidas aleatoriamente (Figura 5 en [7]). Este resultado se alinea mejor con las observaciones de pares resonantes ($G_m < 0.38$).

2.10.6. Relevancia para el Proyecto de Simulación Gravitacional

El trabajo de Wu *et al.* [7] es relevante para nuestro proyecto en varios aspectos contextuales, aunque sus objetivos y metodología difieren fundamentalmente. Proporciona un ejemplo contemporáneo de cómo se utilizan herramientas avanzadas de simulación N-cuerpos (*REBOUND*, *Mercurius*) y métricas específicas (\mathcal{G}_m) para abordar cuestiones astrofísicas complejas, como la estabilidad dinámica de sistemas exoplanetarios. El estudio demuestra la importancia de las condiciones iniciales y las propiedades intrínsecas del sistema (como la distribución de masas) en la determinación de su evolución a largo plazo.

Sin embargo, la **relevancia directa** para la implementación técnica de nuestro proyecto es limitada debido a una diferencia conceptual clave: el modelo de Wu *et al.* [7] opera bajo la premisa de parámetros de cuerpo (masa, posición, velocidad) **fijos durante la ejecución**, excepto por su evolución natural bajo la gravedad mutua. Su objetivo es estudiar la estabilidad *inherente* de configuraciones iniciales dadas, no simular sistemas donde los parámetros cambian debido a procesos físicos adicionales (como acreción, colisiones con modificación de masa) o intervenciones externas simuladas.

Nuestro proyecto, en cambio, se enfoca precisamente en superar esta limitación, buscando desarrollar un modelo que **permite la modificación de parámetros clave, especialmente la masa, en tiempo de ejecución**. El análisis del trabajo de Wu [7] subraya la necesidad de arquitecturas de simulación diferentes cuando el objetivo es modelar sistemas N-cuerpos “atípicos” o procesos dinámicos que involucran cambios paramétricos activos. Mientras que [7] ilustra el estado del arte en simulaciones de estabilidad pasiva a largo plazo, nuestro proyecto se orienta hacia la flexibilidad y la capacidad de interacción dinámica durante la simulación, un área donde el modelo analizado no ofrece una solución aplicable. Por lo tanto, sirve como un punto de referencia importante sobre las capacidades (y limitaciones) de los enfoques estándar para un tipo diferente de problema de simulación N-cuerpos.

2.11. Tabla Comparativa del Estado del Arte

Tabla 2.1: Resumen comparativo de proyectos y artículos relacionados con simulaciones N-cuerpos analizados en el Estado del Arte.

Producto/Méto- do	Enfoque Principal y Características Distintivas	Escalabilidad	Usa IA	Cambios Dinámicos*
Framework ode_num_int ^a	Framework C++11 modular (template-based) para desarrollo y prueba de integradores numéricos (EDOs); monitoreo de rendimiento. Orientado a sistemas de escala media.	Media/Limitada	NO	NO
Representación Geométrica ^b	Representación eficiente de geometría 2D/3D usando Quadtrees/Octrees no minimales (nodos EDGE/VERTEX); optimiza memoria y operaciones booleanas. No es un simulador dinámico.	Alta (Geom.)**	NO	NO
Método n-NNN ^c	Simulación molecular usando Red de n-Vecinos Cercanos (matrices multidimensionales) y cirugía Hamiltoniana; inspirado en IA para evitar cálculo completo de interacciones.	Alta	SÍ	NO
PKDGRAV3 (Hi- drodinámica) ^d	Métodos hidrodinámicos sin malla (MFM/MFV) en PKDGRAV3; enfoque adaptativo basado en vecinos, bueno para altos contrastes de densidad (cosmología).	Alta	NO	NO
Método Híbrido SPH/N-body ^e	Combina SPH (gas) y N-body (estrellas) usando árbol Barnes-Hut con criterio MAC y block timestepping para interacciones gas-estrella.	Alta	NO	NO
Integrador Simpléctico ^f	Integrador simpléctico (orden 2+, reversible) basado en descomposición Kepler para problemas N-cuerpos colisionables; preserva estructura de fase. Requiere paso fijo.	Media	NO	NO
Solver Híbrido TPM ^g	Combina Particle-Mesh (largo alcance) y Tree (corto alcance), asignando tratamiento dinámicamente según densidad local; optimizado para arquitecturas paralelas.	Alta	NO	NO
Algoritmo TPM ^h	Tree-Particle-Mesh basado en descomposición de dominio por densidad; usa árbol para regiones densas y PM para el resto; integración multi-escala temporal.	Alta	NO	NO
REBOUND ⁱ	Código N-cuerpos modular y abierto; incluye integradores simplécticos (WHFast, SEI), Barnes-Hut, manejo de colisiones y condiciones de frontera diversas. Paralelizable (MPI/OpenMP).	Alta	NO	NO
Modelo Estabili- dad Planetaria ^j	Estudio teórico de estabilidad N-cuerpos (usando REBOUND) enfocado en correlación entre uniformidad de masa (índice Gini) y tiempo de inestabilidad. No simula eventos dinámicos internos.	Alta (Estudio)	NO	NO
Solución Pro- puesta	Combina FMM/Barnes-Hut para cálculo gravitacional eficiente con Algoritmos Bioinspirados para la optimización y ajuste dinámico de parámetros (masa) durante la simulación.	Alta	SÍ	SÍ

* **Cambios Dinámicos:** Capacidad de modificar parámetros clave durante la ejecución

^a Sección 2.1: Framework `ode_num_int` para Integración Numérica

^b Sección 2.2: Representación de Objetos mediante *Quadtrees* y *Octrees* de División No Minimal

^c Sección 2.3: Método de Red de n-Vecinos Más Cercanos (n-NNN) para Simulaciones Moleculares

^d Sección 2.4: Implementación de Métodos Hidrodinámicos Sin Malla en PKDGRAV3 para Simulaciones Cosmológicas

^e Sección 2.5: Método híbrido SPH/N-cuerpos para simulaciones de cúmulos estelares

^f Sección 2.6: Integrador simpléctico para problemas gravitacionales N-cuerpos colisionables

^g Sección 2.7: Solver gravitacional híbrido TPM para simulaciones de N-cuerpos en arquitecturas paralelas

^h Sección 2.8: Algoritmo Tree Particle-Mesh (TPM) para simulaciones N-cuerpos cosmológicas

ⁱ Sección 2.9: Código REBOUND para dinámica N-cuerpos colisionante y no colisionante

^j Sección 2.10: Algoritmo TPM (Bode & Ostriker 2000)

** Escalabilidad: “Alta (Geom.)” = específica para operaciones geométricas

CAPÍTULO 3

Marco Teórico

3.1. Problema de los n cuerpos

El problema de n cuerpos estudia cómo n objetos, como planetas o estrellas, se mueven bajo la atracción gravitacional mutua, siguiendo las leyes de Newton. Para dos cuerpos, como la Tierra y el Sol, se puede calcular exactamente sus órbitas, que suelen ser elípticas. Sin embargo, cuando hay tres o más cuerpos, como en el sistema Tierra-Luna-Sol, predecir el movimiento se complica mucho, y a menudo se necesita usar computadoras para simulaciones.

Como se puede intuir, el problema de n cuerpos es un tema fundamental en mecánica celeste, astrofísica y física computacional, con implicaciones significativas en la simulación de sistemas dinámicos. A continuación, se presenta un análisis exhaustivo basado en fuentes académicas.

3.1.1. Puntos clave

- El problema de n cuerpos es un desafío central en física para predecir el movimiento de múltiples objetos que interactúan gravitacionalmente, como planetas o estrellas.
- La investigación indica que para dos cuerpos, hay una solución exacta, pero para tres o más, no hay solución general analítica y puede ser caótico.
- Se utiliza en simulaciones de sistemas planetarios, cúmulos estelares y galaxias, con métodos numéricos como el algoritmo de Barnes-Hut para grandes números de cuerpos.

3.1.2. Principios y funcionamiento

Cada cuerpo atrae a los demás con una fuerza que depende de sus masas y la distancia, según la fórmula:

$$\vec{a}_i = \sum_{j \neq i} \frac{Gm_j(\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3} \quad (3.1)$$

Esto crea un sistema de ecuaciones que, para muchos cuerpos, es difícil de resolver

a mano y requiere métodos numéricos, como dividir el espacio en celdas para calcular fuerzas más rápido.

3.1.3. Aplicaciones inesperadas

Además de la astronomía, el problema de n cuerpos se usa en simulaciones de dinámica molecular, como el movimiento de proteínas, lo que podría sorprender a quienes solo lo asocian con el espacio.

3.1.4. Definición y Enfoques

El problema de n cuerpos se define como el desafío de predecir los movimientos individuales de un grupo de n partículas materiales que interactúan entre sí mediante la ley de gravitación universal de Newton, dada por:

$$F = \frac{Gm_1m_2}{r^2} \quad (3.2)$$

donde G es la constante gravitacional y r es la distancia entre los cuerpos.

3.1.5. Contexto Histórico

El origen del problema se remonta a Isaac Newton (1687), quien formuló la ley de gravitación universal. Henri Poincaré (1889) descubrió el caos determinista al estudiar el problema de tres cuerpos. Karl Fritiof Sundman (siglo XX) proporcionó una solución teórica para $n = 3$ usando series convergentes.

3.1.6. Principios Fundamentales

La aceleración de cada cuerpo i está dada por:

$$\vec{a}_i = \sum_{j \neq i} \frac{Gm_j(\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3} \quad (3.3)$$

Resultando en un sistema de $6n$ ecuaciones diferenciales.

3.1.7. Variantes y Enfoques Principales

Tabla 3.1: Tabla de enfoques principales a la hora de resolver problemas de n cuerpos

Enfoque	Descripción	Complejidad	Referencia
Solución Analítica ($n=2$)	Método que resuelve el problema de Kepler para dos cuerpos. Proporciona soluciones exactas para sistemas de dos objetos.	$O(1)$	[1]
Integración Numérica	Utiliza métodos como Runge-Kutta para simular movimientos. Permite aproximaciones numéricas para sistemas complejos.	$O(n^2)$	[5]
Algoritmo de Barnes-Hut	Método de agrupamiento jerárquico para reducir complejidad computacional. Mejora la eficiencia en simulaciones de múltiples cuerpos.	$O(n \log n)$	[6]

3.1.8. Aplicaciones Típicas

- Dinámica de sistemas planetarios
- Evolución de cúmulos estelares ($n \sim 10^6$)
- Formación de galaxias ($n \sim 10^9$)

3.2. Problema de dos cuerpos

El **problema de dos cuerpos** constituye uno de los fundamentos de la mecánica clásica, la astrofísica y la física computacional. Se trata del estudio del movimiento de dos masas puntuales que interactúan mediante una fuerza central, usualmente la gravitación newtoniana, lo cual permite obtener soluciones exactas que describen las trayectorias orbitales. Diversos autores han abordado este tema desde distintas perspectivas, resaltando su importancia tanto en el análisis teórico como en aplicaciones prácticas, como la predicción de órbitas planetarias y el estudio de sistemas binarios.

3.2.1. Definición y Enfoques Conceptuales

El problema de dos cuerpos se define formalmente como el estudio dinámico de dos objetos (por ejemplo, planetas, satélites o estrellas) que interactúan a través de una fuerza central, típicamente descrita por la ley de gravitación universal:

$$F = \frac{Gm_1m_2}{r^2} \quad (3.4)$$

donde G es la constante gravitacional y r representa la distancia entre las dos masas. Según la literatura, esta interacción se traduce en movimientos cuyas trayectorias son secciones cónicas (elipses, paráolas o hipérbolas), lo que implica que el sistema puede ser resuelto de forma analítica mediante la reducción a un problema de un solo cuerpo utilizando el concepto de centro de masa y la masa reducida. En este marco, la masa reducida se expresa como:

$$\mu = \frac{m_1m_2}{m_1 + m_2} \quad (3.5)$$

lo que permite transformar el movimiento relativo de ambos cuerpos en el movimiento de un único cuerpo bajo la acción de una fuerza central.

3.2.2. Principios Fundamentales y Mecanismo de Funcionamiento

El procedimiento para abordar el problema de dos cuerpos se fundamenta en la transformación al sistema de referencia del centro de masa. En ausencia de fuerzas externas, el centro de masa se mueve a velocidad constante, lo que facilita separar el movimiento global del sistema del movimiento relativo entre las masas. La ecuación diferencial que rige la dinámica radial se puede expresar de manera general como:

$$\frac{d^2\mathbf{r}}{dt^2} = -\frac{G(m_1 + m_2)}{r^3}\mathbf{r} + \frac{l^2}{r^3}\mathbf{r} \quad (3.6)$$

donde l representa el momento angular por unidad de masa. La resolución de esta ecuación se basa en la aplicación de las leyes de conservación de la energía y del momento angular, lo cual conduce a las formulaciones de las leyes de Kepler: la primera ley establece que las órbitas son elipses, la segunda que áreas iguales son barridas en tiempos iguales, y la tercera relaciona el período orbital con el semieje mayor.

3.2.3. Contexto Histórico y Evolución del Problema

El origen teórico del problema de dos cuerpos se remonta a los trabajos de Isaac Newton, quien en 1687, a través de su obra *Philosophiae Naturalis Principia Mathematica* [10], sentó las bases de la gravitación universal y demostró que las trayectorias planetarias son elípticas. Posteriormente, durante el siglo XVIII, Joseph Louis Lagrange y otros matemáticos expandieron estos conceptos, estableciendo métodos analíticos que permitieron una comprensión más profunda del movimiento relativo en sistemas binarios y facilitando la transición hacia el estudio de problemas con más de dos cuerpos mediante técnicas de perturbación.

3.2.4. Variantes y Enfoques Metodológicos

En la literatura se han desarrollado diversos enfoques para abordar el problema de dos cuerpos:

- **Elementos Orbitales:** Este método utiliza parámetros como el semieje mayor, la excentricidad, la inclinación y otros elementos que definen la órbita. Es especialmente valorado en astrofísica por su interpretación directa de la geometría orbital, aunque su aplicación puede volverse compleja en presencia de perturbaciones.
- **Enfoque Vectorial:** Emplea la representación de la posición y la velocidad mediante vectores, lo cual resulta intuitivo para el análisis dinámico y permite incorporar de manera natural las variaciones en la dirección y magnitud de los movimientos.
- **Formulación Hamiltoniana:** Utilizando las ecuaciones de Hamilton, este enfoque es adecuado para estudios perturbativos y para la integración numérica de sistemas dinámicos, aunque requiere una base matemática avanzada.

3.2.5. Aplicaciones Típicas

El problema de dos cuerpos se aplica de forma extendida en áreas tales como:

- **Predicción de órbitas planetarias y trayectorias de satélites:** Permite modelar de manera precisa la dinámica orbital de sistemas planetarios y misiones espaciales.
- **Análisis de sistemas binarios:** Es crucial en la astrofísica para estudiar la interacción entre estrellas en sistemas dobles.
- **Cálculo de trayectorias de cometas y asteroides:** Facilita la determinación de órbitas y la evaluación de posibles perturbaciones en el sistema solar.

Estas aplicaciones han sido fundamentales en el desarrollo de simulaciones numéricas y en la validación de modelos teóricos en astrofísica.

3.2.6. Ventajas y Limitaciones

Entre las principales ventajas del problema de dos cuerpos se destacan:

- **Solución analítica exacta:** Permite una predicción precisa del movimiento de dos cuerpos sin recurrir a aproximaciones numéricas.
- **Base para métodos de perturbación:** Sirve como punto de partida para el análisis de sistemas más complejos en los que se introducen perturbaciones adicionales.

No obstante, sus limitaciones también son evidentes:

- **Restricción a dos cuerpos:** La formulación clásica ignora la influencia de cuerpos adicionales, lo que puede resultar inadecuado en escenarios reales de sistemas múltiples.
- **Suposición de cuerpos puntuales y fuerzas centrales:** La aproximación se basa en modelos ideales que no consideran efectos de extensión finita o fuerzas no centrales, lo que puede introducir discrepancias al modelar situaciones con alta complejidad dinámica.

3.3. Exponente de Lyapunov

El exponente de Lyapunov cuantifica la tasa promedio de separación (divergencia) o acercamiento (convergencia) exponencial de trayectorias infinitamente cercanas en el espacio de fases de un sistema dinámico. Es una medida fundamental de la sensibilidad a las condiciones iniciales. Durante esta sección del reporte abarcaremos lo necesario para el marco de este trabajo terminal.

3.3.1. Definiciones de los Exponentes de Lyapunov

El concepto de exponente de Lyapunov surge de la necesidad de cuantificar la estabilidad y la sensibilidad a las condiciones iniciales en sistemas dinámicos. Existen principalmente dos definiciones formales que, aunque relacionadas, no son idénticas y tienen interpretaciones geométricas distintas:

1. **Exponentes Característicos de Lyapunov (LCEs):** Introducidos originalmente por A.M. Lyapunov [11], se definen como los límites superiores de las tasas de crecimiento exponencial de las *normas* de las soluciones $x_i(t, x_0)$ del sistema linealizado $\dot{x} = J(t, x_0)x$ a lo largo de una trayectoria $z(t, x_0)$ del sistema no lineal original $\dot{z} = F(z)$. Formalmente: [12, 13]:

$$LCE_i(x_0) = \limsup_{t \rightarrow +\infty} \frac{1}{t} \ln |x_i(t, x_0)| \quad (3.7)$$

Estos miden cómo crecen o decrecen las longitudes de los vectores solución individuales. Geométricamente, se relacionan con la evolución de las aristas de un hipercubo infinitesimal bajo el mapeo linealizado [13].

2. Exponentes de Lyapunov (LEs): Definidos más tarde en el contexto de la teoría ergódica [14], se basan en las tasas de crecimiento exponencial de los *valores singulares* $\sigma_i(t, x_0)$ de la matriz fundamental $X(t, x_0)$ del sistema linealizado. Formalmente [13]:

$$LE_i(x_0) = \limsup_{t \rightarrow +\infty} \frac{1}{t} \ln \sigma_i(X(t, x_0)) \quad (3.8)$$

Estos miden cómo crecen o decrecen las longitudes de los semiejes principales de un elipsoide infinitesimal transformado por el sistema linealizado. Los LEs son cruciales para la teoría de la dimensión (p.ej., dimensión de Lyapunov) y son los que usualmente se calculan en estudios de caos [13].

Nota Importante: Aunque el LCE máximo siempre coincide con el LE máximo ($LCE_1 = LE_1$), los demás exponentes LCE_i y LE_i (para $i > 1$) generalmente no coinciden. Además, la suma de los LEs está relacionada con la traza de la Jacobiana (tasa de cambio de volumen en el espacio de fases), mientras que la suma de los LCEs para una base general no tiene esta interpretación directa [13]. La confusión entre LCEs y LEs puede llevar a errores si no se especifica claramente cuál se está usando [15].

3.3.2. Principios Fundamentales y Funcionamiento

El cálculo y la interpretación de los exponentes de Lyapunov se basan en:

- **Linealización:** Se estudia la dinámica local alrededor de una trayectoria $z(t, x_0)$ mediante la ecuación variacional $\dot{x} = J(t, x_0)x$, donde $J(t, x_0) = \frac{\partial F}{\partial z}|_{z=z(t, x_0)}$ es la matriz Jacobiana evaluada a lo largo de la trayectoria [13].
- **Matriz Fundamental:** La evolución de las perturbaciones está gobernada por la matriz fundamental $X(t, x_0)$, solución de $\dot{X} = J(t, x_0)X$ con $X(0, x_0) = I_n$.
- **Teorema Ergódico Multiplicativo (Oseledec):** Garantiza la existencia de los límites (no solo \limsup) para los LEs para casi todo punto x_0 respecto a una medida ergódica invariantes [14, 16]. Sin embargo, verificar la ergodicidad es difícil en la práctica [13].
- **Cálculo Numérico:** Se usan algoritmos basados en la descomposición QR o SVD de la matriz fundamental, a menudo involucrando la reortogonalización periódica de un conjunto de vectores ortonormales evolucionados bajo el sistema linealizado (método de Benettin) [17]; Wolf et al. [18, 19].
- **Espectro de Lyapunov:** Para un sistema n-dimensional, hay n exponentes $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$. λ_1 (el exponente máximo) determina la predictibilidad. Si $\lambda_1 > 0$, el sistema es caótico. La suma $\sum \lambda_i$ indica si el sistema es disipativo (< 0) o conservativo ($= 0$).

3.3.3. Contexto Histórico

- **A.M. Lyapunov (1892):** Introdujo los “exponentes característicos” (LCEs) para estudiar la estabilidad de soluciones de ecuaciones diferenciales *lineales*

con coeficientes variables en el tiempo [11]. Definió sistemas “regulares” para los cuales el signo del LCE máximo determina la estabilidad de la solución cero [20].

- **O. Perron (1930):** Construyó un contraejemplo (un sistema lineal 2D) mostrando que para sistemas “irregulares”, tener todos los LCEs negativos no implica estabilidad (inestabilidad de Lyapunov de la solución cero). Además, mostró que en la vecindad de la solución cero pueden existir soluciones con LCEs positivos. Este fenómeno se conoce como *efecto Perron* [12, 20].
- **V.I. Oseledec (1968):** Probó el Teorema Ergódico Multiplicativo, estableciendo rigurosamente la existencia de los exponentes (LEs) y sus propiedades fundamentales en el marco de la teoría ergódica [14].
- **Desarrollos Posteriores:** Conexión con el caos, cálculo de la dimensión fractal Kaplan-Yorke [21], métodos numéricos robustos Benettin et al. [17], Wolf et al. [18].

3.3.4. Variantes y Conceptos Relacionados

- **LCEs vs. LEs:** Discutido arriba. Los LEs (basados en valores singulares) son los relevantes para la dimensión de Lyapunov y la tasa de cambio de volumen.
- **Exponentes de Tiempo Finito (FTLEs):** Valores calculados sobre intervalos finitos T , $LE_i(T, x_0)$. Convergen a los LEs asintóticos cuando $T \rightarrow \infty$ (si el límite existe). Son útiles numéricamente pero pueden fluctuar y mostrar “saltos” [13, 22].
- **Exponentes Condicionales:** Usados en sistemas acoplados (drive-response) para medir la estabilidad de la sincronización. Miden la divergencia/convergencia en las direcciones transversales al manifold de sincronización [23].
- **Exponentes k-dimensionales:** Miden la tasa de expansión/contracción de volúmenes k-dimensionales en el espacio tangente, dados por la suma de los k mayores LEs ($\sum_{i=1}^k \lambda_i$) [24].
- **Regularidad vs. Irregularidad:** Un sistema linealizado $\dot{x} = J(t)x$ es *regular* si la suma de sus LCEs (calculados a partir de una base normal) es igual al exponente de la traza ($\int Tr(J)dt$). Si no, es *irregular*. Solo para sistemas regulares, $\lambda_1 < 0$ implica estabilidad asintótica [20], [13]. El efecto Perron es una característica de los sistemas irregulares.
- **Invariancia:** Los LEs son invariantes bajo diffeomorfismos del espacio de fases, lo que los hace adecuados para caracterizar atractores. Los LCEs no comparten esta propiedad general, pero son invariantes bajo transformaciones de Lyapunov específicas [13].

3.3.5. Aplicaciones (basadas en los artículos)

- **Detección y Caracterización del Caos:** El signo del LE máximo (λ_1) se usa como indicador primario de caos [19].
- **Estabilidad Orbital:** En el CR3BP (*Problema Restringido Circular de Tres Cuerpos*), se usa λ_1 para determinar los límites de estabilidad de órbitas planetarias [19].
- **Dimensión de Atractores:** La dimensión de Lyapunov D_{KY} , calculada a partir del espectro de LEs usando la fórmula de Kaplan-Yorke, proporciona una cota superior para la dimensión de Hausdorff del atractor [13, 22]. El método de Leonov ofrece una vía analítica para estimar D_{KY} usando funciones de Lyapunov [22].
- **Sincronización:** Los exponentes condicionales determinan si el estado sincronizado es estable en sistemas acoplados [23].
- **Estabilidad por Primera Aproximación:** El análisis de estabilidad de puntos fijos o trayectorias periódicas se basa en los LCEs/LEs del sistema linealizado. Sin embargo, el efecto Perron limita su aplicabilidad general a sistemas irregulares [12, 20].

3.3.6. Ventajas y Limitaciones

- **Ventajas:**
 - Proporciona una medida cuantitativa y objetiva del caos.
 - Relaciona la dinámica local (expansión/contracción) con el comportamiento global.
 - Fundamental para calcular la dimensión de atractores.
 - Invariante bajo cambios suaves de coordenadas (para LEs).
- **Limitaciones:**
 - **Efecto Perron:** La principal limitación teórica. El signo del LCE/LE máximo *no* siempre determina la estabilidad/inestabilidad en sistemas no lineales generales, solo en los regulares [13, 20].
 - **Cálculo Numérico:** La convergencia puede ser lenta; los FTLEs pueden fluctuar; problemas de precisión si los exponentes están cerca o son cero; la elección del tiempo de integración y reortogonalización es crucial [13, 22].
 - **Requerimientos Teóricos:** La existencia rigurosa de LEs (como límites) depende de la teoría ergódica (Teorema de Oseledec), cuyas hipótesis (existencia de medida ergódica) son difíciles de verificar en la práctica [13].
 - **Estimación a partir de Series Temporales:** Muy sensible al ruido, longitud de la serie, dimensión de embedding y otros parámetros; puede dar resultados espurios [13], citando a Sander & York.

- **Confusión Terminológica:** La existencia de LCEs y LEs (y a veces la falta de distinción clara en la literatura) puede generar confusión [13].

3.4. Integradores Simplicéticos Gravitacionales

Los integradores simplicéticos gravitacionales representan una clase especializada de métodos numéricos diseñados para la simulación de sistemas dinámicos descritos por la mecánica hamiltoniana, con una aplicación primordial en el problema gravitacional de N-cuerpos [25, 26]. Estos métodos son cruciales en la mecánica celeste y la astrofísica computacional para modelar con alta fidelidad la evolución a largo plazo de sistemas como planetas orbitando estrellas, satélites, asteroides, cúmulos estelares y galaxias [25, 27, 28]. Su característica distintiva y fundamental es la preservación de la estructura geométrica del espacio de fases, conocida como la forma simplicética, lo que conlleva excelentes propiedades de conservación a largo plazo para cantidades como la energía y el momento angular, aunque esta conservación de energía no sea exacta, sino acotada [26, 29, 30].

3.4.1. Definición y Principios Fundamentales

Un sistema dinámico descrito por un Hamiltoniano $H(\mathbf{q}, \mathbf{p}, t)$ evoluciona en el tiempo según las ecuaciones de Hamilton. Un integrador numérico approxima este flujo continuo mediante un mapa discreto:

$$\Phi_h : (\mathbf{q}_n, \mathbf{p}_n) \mapsto (\mathbf{q}_{n+1}, \mathbf{p}_{n+1}) \quad (3.9)$$

que avanza el sistema en un paso de tiempo h .

El integrador Φ_h se denomina simplicético si preserva la 2-forma simplicética:

$$\omega = \sum_i d\mathbf{p}_i \wedge d\mathbf{q}_i \quad [26, 29]. \quad (3.10)$$

Matemáticamente, esto significa que la matriz Jacobiana del mapa:

$$J = \frac{\partial(\mathbf{q}_{n+1}, \mathbf{p}_{n+1})}{\partial(\mathbf{q}_n, \mathbf{p}_n)} \quad (3.11)$$

debe satisfacer la condición:

$$J^T \Omega J = \Omega \quad (3.12)$$

donde Ω es la matriz estándar simplicética [29, 30]. Esta propiedad garantiza que el mapa es una transformación canónica y preserva el volumen en el espacio de fases (Teorema de Liouville) [25, 26].

Para el problema gravitacional de N-cuerpos, el Hamiltoniano es separable, $H = T(\mathbf{p}) + V(\mathbf{q})$, donde T es la energía cinética (dependiente solo de los momentos) y V es la energía potencial (dependiente solo de las posiciones) [27, 31]. Esta separabilidad permite la construcción de integradores simplicéticos mediante métodos de división (*splitting methods*) [26, 27]. Estos métodos descomponen el Hamiltoniano en partes integrables analíticamente (o de forma muy precisa) y componen sus flujos para aproximar el flujo completo. El ejemplo más básico es el integrador de Leapfrog (o Verlet), que es de segundo orden y simplicético [26, 29, 31].

3.4.2. Contexto Histórico y Desarrollo

La necesidad de simulaciones estables a largo plazo en mecánica celeste impulsó el desarrollo de integradores simplécticos. Aunque la idea de conservar propiedades geométricas no es nueva, su aplicación sistemática a la integración numérica cobró fuerza a partir de los años 80. Trabajos como los de Ruth (1983) [32] y posteriormente Forest & Ruth (1990) [32], Candy & Rozmus (1991) [30], y Yoshida (1990) [29, 30] establecieron métodos para construir integradores simplécticos de orden superior mediante composición simétrica de métodos de orden inferior. Un hito fundamental fue el método de mapeo simpléctico de Wisdom y Holman (1991) [25], diseñado específicamente para la dinámica planetaria, que explotaba la dominancia del término Kepleriano. Paralelamente, los integradores variacionales, basados en la discretización del principio de acción de Hamilton, emergieron como otro enfoque robusto para derivar algoritmos simplécticos y conservadores de momento [29, 31].

3.4.3. Variantes y Enfoques Principales

La literatura describe una diversidad de integradores simplécticos aplicados al problema de N-cuerpos:

Tabla 3.2: Principales variantes de integradores simplécticos para problemas de N-cuerpos.

Variante	Descripción	Referencias Clave
Método de Leapfrog/Verlet	Integrador simpléctico de segundo orden, base para muchos otros métodos. Eficiente para $H = T(p) + V(q)$.	[26, 29, 31]
Método de Wisdom-Holman (WH)	Separa H en $H_{\text{Kepler}} + H_{\text{Interaction}}$. Muy eficiente para órbitas casi Keplerianas (e.g., sistemas planetarios). Preserva implícitamente la integral de Jacobi modificada.	[25, 32, 33]
Integradores de Orden Superior	Construidos mediante composición (e.g., Yoshida). Aumentan la precisión pero requieren más evaluaciones de fuerza y pueden tener pasos de tiempo negativos.	[29, 30, 32]
Integradores Variacionales (GGL)	Derivados de la discretización del Lagrangiano (e.g., Galerkin Gauss-Lobatto). Conservan exactamente el momento (lineal y angular si hay simetría) y son simplécticos.	[29, 31]
Integradores Simplécticos Híbridos	Combinan un integrador simpléctico (como WHFast) para la evolución general con otro método (simpléctico o no, como IAS15 o BS) para manejar eventos específicos como encuentros cercanos. Ejemplos: MERCURIUS, TRACE, EnckeHH.	[9, 28, 33, 34]
Integradores Simplécticos Adelante (FSI)	Clase de integradores (e.g., 4A, 4B, 4C) que solo usan pasos de tiempo positivos, a costa de requerir el gradiente de la fuerza. Eficientes para problemas con fuerzas dependientes del tiempo.	[27]
Métodos Basados en Encke	Resuelven las ecuaciones para las <i>perturbaciones</i> respecto a una órbita Kepleriana de referencia. Muy precisos si las perturbaciones son pequeñas. Ejemplo: EnckeHH.	[28]
Métodos Gauge-Compatible (GCSM)	Para problemas con simetrías de gauge (como PIC en electromagnetismo), preservan explícitamente el mapa de momento asociado (e.g., ley de Gauss discreta).	[35]
Integradores para Coordenadas Corrotantes	Adapta métodos simplécticos a sistemas no canónicos que surgen en marcos de referencia en rotación, preservando una estructura K-simpléctica.	[36]

3.4.4. Propiedades Clave

Las propiedades más importantes de estos integradores son:

1. **Simplécticidad:** Conservación de la 2-forma simpléctica, lo que implica la preservación del volumen en el espacio de fases [26, 29].
2. **Conservación de Energía:** Aunque no conservan exactamente la energía

H , el error energético para un integrador simpléctico de orden r aplicado a un sistema hamiltoniano autónomo permanece acotado a largo plazo, oscilando alrededor del valor inicial, en contraste con el error secular (lineal o peor) de métodos no simplécticos [26, 28, 30]. Existe un Hamiltoniano “sombra” \tilde{H} , cercano a H , que es conservado casi exactamente por el integrador [30, 34].

3. **Conservación de Momento:** Los integradores variacionales conservan exactamente los mapas de momento asociados a las simetrías del Lagrangiano discreto (e.g., momento lineal y angular si el sistema es invariante bajo traslaciones y rotaciones) [29, 31]. Otros métodos simplécticos también suelen tener excelente conservación del momento, aunque puede haber errores de orden superior, especialmente con pasos de tiempo adaptativos [31].
4. **Estabilidad a Largo Plazo:** La preservación de la estructura simpléctica evita la deriva secular en los elementos orbitales y garantiza una excelente estabilidad numérica para integraciones muy largas [25, 26, 30].

3.4.5. Aplicaciones Típicas

Los integradores simplécticos son la herramienta estándar en:

- **Dinámica del Sistema Solar:** Estudio de la evolución orbital de planetas, asteroides y cometas a escalas de tiempo de millones o miles de millones de años [25, 32].
- **Dinámica de Exoplanetas:** Análisis de la estabilidad de sistemas planetarios extrasolares [33].
- **Dinámica Estelar:** Simulación de cúmulos globulares y centros galácticos, aunque aquí los encuentros cercanos son un desafío [28, 31].
- **Simulaciones Cosmológicas:** El método Leapfrog es ampliamente usado en simulaciones de N-cuerpos para materia oscura debido a su simplicidad, eficiencia y naturaleza simpléctica [26].
- **Física de Plasmas (PIC):** Métodos geométricos y simplécticos, incluyendo los gauge-compatibles, se usan para preservar estructuras fundamentales [35].

3.4.6. Ventajas

- Excelente estabilidad y precisión a largo plazo [25, 30].
- Error de energía acotado, sin deriva secular [26, 29].
- Buena conservación (a menudo exacta en métodos variacionales) de otros integrales de movimiento como el momento lineal y angular [29, 31].
- Eficiencia computacional para Hamiltoniano separables [25].

3.4.7. Desventajas y Limitaciones

- **Pasos de Tiempo:** Los métodos simplécticos puros construidos por composición o métodos explícitos estándar requieren pasos de tiempo fijos para mantener la simplécticidad [26, 29]. Esto es ineficiente para sistemas

multi-escala o con encuentros cercanos donde se necesitarían pasos muy pequeños [28, 31].

- **Adaptatividad:** Introducir pasos de tiempo adaptativos (que varían con el tiempo o por partícula) generalmente rompe la estructura simpléctica exacta [29, 31]. Sin embargo, se ha demostrado que usar pasos de tiempo adaptativos discretizados en potencias de dos (*block power-of-two*) en integradores variacionales preserva la simplécticidad “casi siempre” (excepto en un conjunto de medida cero) [31], ofreciendo un compromiso viable. Hernandez (2019) [34] enfatiza que romper la simplécticidad, incluso en pocos puntos, puede ser problemático si no se mantiene la continuidad Lipschitz.
- **Encuentros Cercanos:** La singularidad del potencial gravitacional $1/r$ causa problemas. Los integradores simplécticos puros pueden fallar o requerir pasos de tiempo prohibitivamente pequeños. Técnicas como la regularización o el cambio a integradores no simplécticos (híbridos) son necesarias [28, 33].
- **Fuerzas No Conservativas:** Los integradores simplécticos están diseñados para sistemas hamiltonianos. Incorporar fuerzas disipativas (como arrastre o radiación) requiere extensiones que pueden comprometer las propiedades de conservación [34].
- **Complejidad:** Los métodos de orden superior o los que requieren gradientes de fuerza (FSIs) pueden ser más costosos por paso [27, 29]. Los métodos implícitos requieren resolver ecuaciones no lineales [32].

3.4.8. Desarrollos Recientes y Direcciones Futuras

La investigación activa se centra en superar las limitaciones:

- **Integradores Híbridos:** Combinan la eficiencia y estabilidad simpléctica para la mayor parte de la evolución con la robustez de métodos adaptativos (como IAS15 o BS) durante fases críticas (encuentros cercanos) [9, 28, 33].
- **Métodos Adaptativos Simplécticos:** Desarrollo de esquemas que permitan variar el paso de tiempo mientras se preserva (exacta o aproximadamente) la simplécticidad, como los basados en potencia de dos [31] o correctores simplécticos [32].
- **Integradores Geométricos Más Allá de Simplécticos:** Exploración de métodos que preserven otras estructuras geométricas relevantes para sistemas específicos (e.g., simetrías de gauge [35], integradores para coordenadas corrotantes [36]).
- **Optimización de Errores:** Técnicas para minimizar errores de redondeo en integraciones de alta precisión, como la suma compensada y el manejo cuidadoso de constantes [33].

3.5. Integrador Simpléctico WHFast

El integrador WHFast (Wisdom-Holman Fast) es un método numérico perteneciente a la clase de integradores simplécticos, diseñado específicamente para la

simulación gravitacional a largo plazo de sistemas de N-cuerpos, particularmente en el contexto de la dinámica planetaria [37]. Los integradores simplécticos, en general, son algoritmos numéricos para resolver ecuaciones de Hamilton que tienen la propiedad fundamental de preservar la estructura simpléctica del espacio de fases del sistema [38]. Esta preservación implica que, aunque la energía total calculada numéricamente puede oscilar alrededor del valor verdadero, no exhibe una deriva secular sistemática a largo plazo, una característica crucial para la estabilidad y precisión en simulaciones extendidas [37, 38].

3.5.1. Principios Fundamentales y Funcionamiento

WHFast se basa en el algoritmo de Wisdom-Holman [25], que explota la estructura de muchos problemas astrofísicos donde existe un cuerpo central masivo (e.g., una estrella) y cuerpos de masa mucho menor (e.g., planetas) cuyas interacciones mutuas son perturbaciones sobre sus órbitas Keplerianas casi independientes alrededor del cuerpo central. El método de Wisdom-Holman logra esto mediante una técnica de *splitting* (partición o descomposición) del Hamiltoniano del sistema H . El Hamiltoniano se divide típicamente en dos o más partes que son integrables analíticamente o más fáciles de aproximar numéricamente. Para el problema planetario, la partición usual es:

$$H = H_{\text{Kepler}} + H_{\text{Interaction}} \quad (3.13)$$

donde H_{Kepler} describe la suma de los problemas de dos cuerpos independientes (planeta-estrella), cuya solución son las órbitas Keplerianas, y $H_{\text{Interaction}}$ contiene las interacciones gravitatorias entre los planetas [25, 37].

El integrador avanza el sistema alternando pasos de integración correspondientes a cada parte del Hamiltoniano. Un esquema común es el de Leapfrog, también conocido como *Drift-Kick-Drift* (DKD):

1. **Drift (Deriva):** Se avanzan las posiciones de los cuerpos bajo H_{Kepler} durante medio paso de tiempo ($\Delta t/2$). Esto equivale a mover los cuerpos a lo largo de sus órbitas Keplerianas no perturbadas.
2. **Kick (Impulso):** Se actualizan los momentos (velocidades) de los cuerpos debido a $H_{\text{Interaction}}$ durante un paso de tiempo completo (Δt). Esto aplica las aceleraciones debidas a las interacciones mutuas entre los planetas.
3. **Drift (Deriva):** Se avanzan nuevamente las posiciones bajo H_{Kepler} durante el segundo medio paso de tiempo ($\Delta t/2$).

WHFast implementa mejoras significativas sobre el esquema básico de Wisdom-Holman. Rein y Tamayo [37] destacan dos áreas principales de optimización:

1. **Solucionador de Kepler Mejorado:** La evolución bajo H_{Kepler} requiere resolver la ecuación de Kepler, que es trascendental. WHFast utiliza implementaciones optimizadas del método de Newton y otros (como Laguerre-Conway para altas excentricidades) con criterios de convergencia robustos y eficientes en aritmética de punto flotante, además de una cuidadosa gestión de las funciones especiales involucradas (funciones c y G de Stumpff) [8, 37].

2. **Transformaciones de Coordenadas Estables:** A menudo es computacionalmente ventajoso trabajar en coordenadas de Jacobi en lugar de heliocéntricas o baricéntricas, especialmente para la parte Kepleriana. Las coordenadas de Jacobi referencian la posición de un cuerpo respecto al centro de masas de todos los cuerpos interiores a él. WHFast implementa transformaciones numéricamente estables y sin sesgos entre las coordenadas iniciales (necesarias para calcular $H_{\text{Interaction}}$) y las coordenadas de Jacobi, minimizando la propagación de errores de redondeo [8, 37].

Gracias a estas mejoras, WHFast logra eliminar la deriva secular lineal en el error de energía que afectaba a implementaciones anteriores y, para pasos de tiempo suficientemente pequeños, alcanza la Ley de Brouwer, donde el error energético está dominado por un camino aleatorio debido a errores de redondeo, escalando como \sqrt{t} en lugar de t [8, 37].

3.5.2. Contexto Histórico y Origen

El método fundamental fue propuesto por Wisdom y Holman en 1991 [25] (e independientemente por Kinoshita et al. [39]), basándose en ideas previas de Wisdom sobre mapeos simplécticos [40]. WHFast, como implementación específica y optimizada, fue presentado por Hanno Rein y Daniel Tamayo en 2015 [37], con el objetivo de proporcionar una herramienta rápida, precisa y numéricamente robusta, disponible como parte del paquete de simulación REBOUND [8].

3.5.3. Variantes y Opciones de Configuración

La implementación de WHFast, particularmente dentro del marco de REBOUND, ofrece varias configuraciones para ajustar su comportamiento y precisión [8, 41]:

1. **Correctores Simplécticos:** Para reducir aún más el error energético (específicamente, las oscilaciones a corto plazo), se pueden aplicar correctores simplécticos de alto orden (3, 5, 7, 11, e incluso 17) [37, 42]. Estos correctores modifican las transformaciones al inicio y al final de la integración (o antes de las salidas de datos), con un costo computacional adicional mínimo si las salidas son poco frecuentes. La efectividad de los correctores de orden superior depende de la relación de masas en el sistema [37]. También existe la opción de correctores de segundo orden (`corrector2`) [41].
2. **Sistemas de Coordenadas:** Aunque las coordenadas de Jacobi son a menudo las más eficientes para la parte Kepleriana, WHFast puede configurarse para usar internamente otros sistemas como el democrático heliocéntrico, WHDS o baricéntrico, lo que puede ser ventajoso en ciertos escenarios [41, 43]. Sin embargo, los correctores simplécticos suelen ser compatibles solo con Jacobi o baricéntricas [41, 43].
3. **Núcleo (Kernel):** Además del núcleo estándar de Wisdom-Holman, existen variantes como `modifiedkick` (patada modificada para gravedad Newtoniana exacta en Jacobi), `composition` (método de composición de Wisdom) y `lazy` (método del implementador perezoso), que pueden ofrecer ventajas en casos específicos pero a menudo requieren coordenadas de Jacobi [41].
4. **Modo Seguro (`safe_mode`):** Por defecto (`safe_mode=1`), WHFast realiza

comprobaciones y sincronizaciones (transformaciones de coordenadas y aplicación de pasos de deriva fraccionados para asegurar que las posiciones y velocidades reportadas sean físicamente significativas en el tiempo solicitado) en cada paso, lo que garantiza robustez pero reduce el rendimiento. Desactivarlo (`safe_mode=0`) ofrece un aumento sustancial de velocidad, pero requiere que el usuario gestione manualmente la sincronización y la recálculo de coordenadas si modifica partículas entre pasos [41, 44].

5. **Ecuaciones Variacionales:** WHFast puede integrar simultáneamente las ecuaciones variacionales, lo que permite calcular indicadores de caos como el Número Característico de Lyapunov (LCN) y el MEGNO (Mean Exponential Growth factor of Nearby Orbits) con un costo adicional muy bajo [8, 37].

Tabla 3.3: Opciones de Configuración Principales de WHFast en REBOUND [41]

Opción	Descripción
<code>corrector</code>	Orden del corrector simpléctico C1 (0: apagado, valores impares 3-17).
<code>corrector2</code>	Activa/desactiva corrector C2 (0: apagado, 1: activado).
<code>kernel</code>	Selección del núcleo de integración (default, modifiedkick, composition, lazy).
<code>coordinates</code>	Sistema de coordenadas interno (jacobi, democratichelio-centric, whds, barycentric).
<code>safe_mode</code>	Activa/desactiva sincronización y comprobaciones automáticas (1: activado, 0: desactivado).

3.5.4. Aplicaciones Típicas

WHFast es ampliamente utilizado en la simulación a largo plazo de la dinámica de sistemas planetarios, tanto dentro del Sistema Solar como en sistemas exoplanetarios, donde se requiere alta precisión en la conservación de energía y momento angular durante miles o millones de órbitas [8, 37]. También es una herramienta estándar para estudios de estabilidad orbital, resonancias y caracterización del caos en dichos sistemas mediante el cálculo de indicadores como LCN y MEGNO [37].

3.5.5. Ventajas y Desventajas/Limitaciones

Ventajas:

- **Velocidad:** Es significativamente más rápido (1.5 a 5 veces) que otras implementaciones de Wisdom-Holman debido a sus optimizaciones [37].
- **Precisión y Conservación:** Conserva la energía y otras integrales del movimiento a largo plazo mucho mejor que los integradores no simplécticos y que implementaciones anteriores de Wisdom-Holman [8, 37]. La opción de correctores simplécticos mejora aún más la precisión [37].
- **Estabilidad Numérica:** Es robusto frente a errores de redondeo, logrando un comportamiento de error no sesgado (ley de Brouwer) [37].
- **Flexibilidad:** Permite el uso de diferentes coordenadas, correctores, y la integración de ecuaciones variacionales [37, 41]. Puede operar en cualquier

marco inercial [37].

Desventajas/Limitaciones:

- **Supuesto de Dominancia Kepleriana:** Al igual que el método de Wisdom-Holman original, su eficiencia y precisión se basan en que el movimiento esté dominado por órbitas Keplerianas. No es adecuado para sistemas con encuentros cercanos frecuentes, colisiones, o donde no hay un cuerpo central claramente dominante [8, 45].
- **Fuerzas No Conservativas o Dependientes de la Velocidad:** Como integrador simpléctico diseñado para sistemas Hamiltonianos conservativos, no maneja de forma nativa e inherentemente simpléctica fuerzas no conservativas (como el arrastre atmosférico) o fuerzas dependientes de la velocidad (como las fuerzas de marea o la relatividad general, aunque estas pueden añadirse como perturbaciones externas gestionadas por otros módulos como REBOUNDx) [38, 44]. La adición de tales fuerzas rompe la estructura simpléctica.
- **Complejidad de Configuración Avanzada:** Para obtener el máximo rendimiento (`safe_mode=0`), el usuario debe comprender los detalles del funcionamiento interno del integrador, como la sincronización y la necesidad de recalcular coordenadas [41, 44].

3.6. Método Multipolar Rápido (Fast Multipole Method - FMM)

El Método Multipolar Rápido (FMM), es una técnica computacional fundamental desarrollada para acelerar significativamente la evaluación de interacciones de largo alcance en sistemas de N partículas o elementos [46, 47]. Reconocido como uno de los algoritmos más influyentes del siglo XX [48], el FMM reduce drásticamente la complejidad computacional inherente a los problemas de N -cuerpos. Mientras que un cálculo directo de todas las interacciones par a par requiere $O(N^2)$ operaciones, el FMM logra esta tarea típicamente en $O(N)$ o $O(N \log N)$ operaciones, dependiendo de la distribución de las partículas y la variante del método empleada [46, 49]. De manera más precisa, en d dimensiones, la complejidad puede estimarse como $O(N \log^{(d-1)}(1/\epsilon))$ a medida que la precisión deseada ϵ tiende a cero [47]. Conceptualmente, el FMM puede interpretarse como un método para acelerar la multiplicación de un vector por una matriz densa específica que surge de la discretización de operadores integrales o sumas de interacciones par a par [47, 50].

3.6.1. Principios Fundamentales y Funcionamiento

El núcleo del FMM reside en una combinación de descomposición jerárquica del dominio computacional y el uso de expansiones matemáticas para aproximar interacciones [46]. El dominio se subdivide recursivamente mediante una estructura de árbol, como un *quadtree* (2D) o un *octree* (3D), agrupando las partículas o fuentes en celdas (o “cajas”) a diferentes niveles de refinamiento [47, 49].

El método distingue entre interacciones de “campo cercano” y “campo lejano”. Las interacciones entre partículas en celdas adyacentes (campo cercano) se calculan directamente, ya que las aproximaciones no son precisas a corta distancia [50, 51]. Para las interacciones entre celdas bien separadas (campo lejano), el FMM emplea dos tipos principales de expansiones [46, 47]:

- **Expansión Multipolo (Saliente):** Representa el campo potencial generado por todas las fuentes *dentro* de una celda (la “fuente”) como una única serie convergente fuera de la celda. Esta expansión es válida a una distancia suficiente de la celda origen. Funciona como una representación compacta de las fuentes.
- **Expansión Local (Entrante):** Representa el potencial generado por fuentes *distantes* (campo lejano) como una serie convergente *dentro* de una celda (la “objetivo”). Funciona como una representación compacta del campo externo que actúa sobre la celda.

El algoritmo clásico opera en dos fases principales sobre la estructura de árbol [46, 47, 49]:

- **Paso Ascendente (Upward Pass):** Se calculan las expansiones multipolares para todas las celdas. Para las celdas hoja (el nivel más fino), se calcula directamente a partir de las fuentes contenidas. Para celdas padre, las expansiones se obtienen combinando (trasladando y sumando) las expansiones de sus celdas hijas (operador *Multipole-to-Multipole* o M2M).
- **Paso Descendente (Downward Pass):** Se calculan las expansiones locales para todas las celdas. Para cada celda, se convierten las expansiones multipolares de las celdas en su “lista de interacción” (celdas bien separadas en el mismo nivel, cuyos padres son vecinos pero ellas no) en contribuciones a su propia expansión local (operador *Multipole-to-Local* o M2L). Luego, las expansiones locales de una celda padre se trasladan y se añaden a las expansiones locales de sus celdas hijas (operador *Local-to-Local* o L2L).

Finalmente, en las celdas hoja, el potencial total en cada punto se calcula sumando la contribución directa de las fuentes en celdas vecinas y evaluando la expansión local acumulada (que representa todo el campo lejano). La clave de la eficiencia $O(N)$ radica en que el número de celdas en la lista de interacción es acotado (independiente de N), y las operaciones de traslación (M2M, M2L, L2L) tienen un coste que depende del orden de la expansión p (relacionado con la precisión ϵ) pero no de N [46].

Desde una perspectiva matricial, el FMM explota la propiedad de que los bloques fuera de la diagonal de la matriz de interacción (que representan interacciones entre grupos de puntos separados espacialmente) tienen bajo rango numérico y pueden aproximarse eficientemente mediante factorizaciones de bajo rango, como las implícitas en las expansiones multipolares y locales [47, 50].

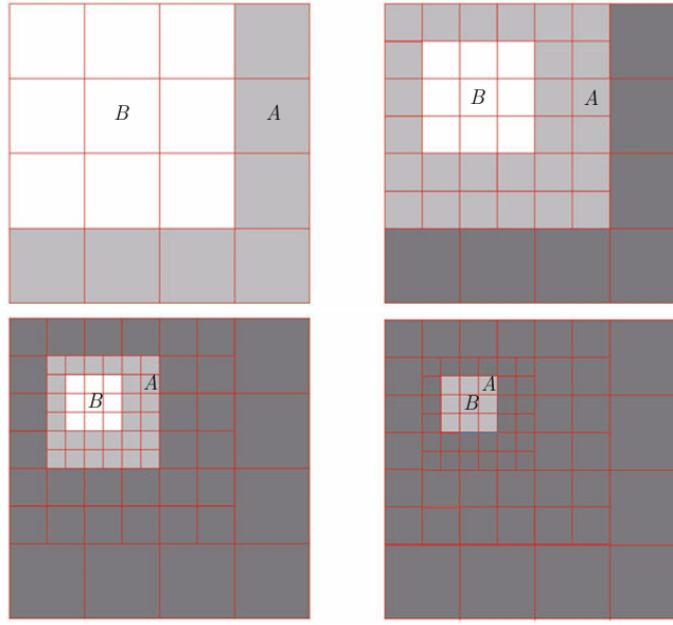


Figura 3.1: Esquema de la estructura jerárquica de celdas (e.g., quadtree) y la distinción entre interacciones de campo cercano (directas) y campo lejano (aproximadas mediante expansiones), incluyendo la lista de interacción. Adaptado de [46, 51].

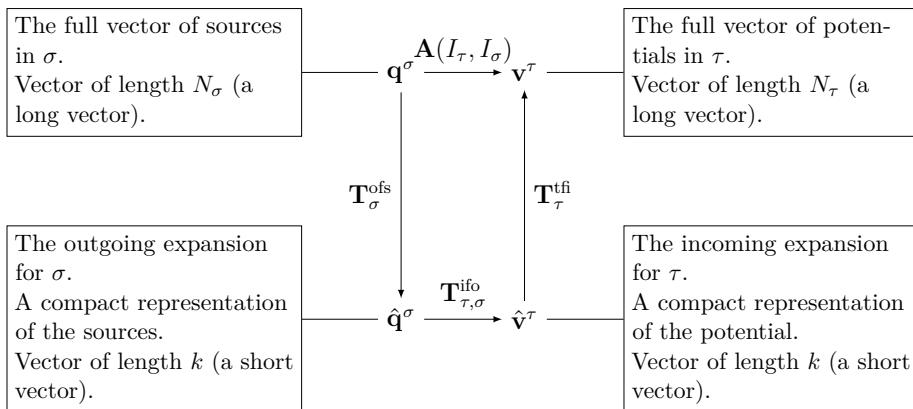


Figura 3.2: Ilustración del flujo de información en el FMM: fuentes originales (\mathbf{q}^σ) \rightarrow expansión saliente ($\hat{\mathbf{q}}^\sigma$) \rightarrow expansión entrante ($\hat{\mathbf{v}}^\tau$) \rightarrow potenciales (\mathbf{v}^τ), mostrando los operadores T^{ofs} , T^{ifo} , y T^{tfi} . Adaptado de [47].

3.6.2. Contexto Histórico/Origen

El FMM fue introducido formalmente por Leslie Greengard y Vladimir Rokhlin en su seminal artículo de 1987 [46], aunque ideas relacionadas con expansiones multipolares y métodos jerárquicos existían previamente (e.g., [52, 53]). El trabajo de Greengard y Rokhlin proporcionó un marco riguroso y un algoritmo con complejidad demostrada de $O(N)$ para el problema de Laplace, revolucionando la simulación de N -cuerpos y la solución de ecuaciones integrales [46]. Su impacto fue tal que fue incluido en la lista de los 10 algoritmos más importantes del siglo XX [48].

3.6.3. Variantes o Enfoques Principales

El FMM ha evolucionado considerablemente desde su concepción original:

- **Extensiones a otros Kernels:** Aunque inicialmente formulado para el kernel de Laplace ($1/r$), se ha extendido a otros kernels importantes como el de Helmholtz (ondas), Stokes (fluidos), Yukawa (física de partículas) y elasticidad [47, 54, 55]. Para kernels oscilatorios como el de Helmholtz, se requieren enfoques más sofisticados (e.g., *FMM direccional*) para mantener la eficiencia, especialmente a altas frecuencias [55, 56].
- **Métodos Adaptativos:** Para manejar distribuciones de partículas altamente no uniformes, se desarrollaron versiones adaptativas del FMM que refinan el árbol jerárquico solo donde es necesario, manteniendo la eficiencia [54, 57].
- **FMM Independiente del Kernel (KIFMM):** Estos métodos buscan separar la maquinaria del FMM (árbol, traslaciones) de las expansiones específicas del kernel, utilizando representaciones intermedias como cargas equivalentes/ficticias y potenciales de chequeo, lo que facilita su aplicación a nuevos kernels [47, 58].
- **Optimizaciones para Baja Precisión:** En campos como la astrofísica, donde no siempre se requiere alta precisión, se han desarrollado métodos FMM-like o variantes de códigos de árbol optimizados, que pueden lograr complejidades $O(N)$ o incluso sublineales empíricamente [59]. El método de Dehnen [59], por ejemplo, utiliza interacciones celda-celda y expansiones de Taylor, conservando el momento lineal.
- **Versiones Paralelas y para GPU:** Se han desarrollado implementaciones altamente optimizadas para arquitecturas paralelas modernas, incluyendo CPUs multi-núcleo y GPUs, lo que permite abordar problemas de escala masiva [47, 60].

3.6.4. Aplicaciones Típicas

El FMM es una herramienta esencial en:

- **Astrofísica:** Simulación de la evolución dinámica de galaxias y cúmulos estelares (interacciones gravitacionales) [48, 59].
- **Dinámica Molecular:** Cálculo de fuerzas electrostáticas y de van der Waals en simulaciones de proteínas y otras biomoléculas [49].
- **Electromagnetismo:** Solución de ecuaciones integrales para el cálculo de capacitancia, dispersión de ondas electromagnéticas y diseño de antenas [47, 54].
- **Mecánica de Fluidos:** Simulación de flujos utilizando métodos de vórtices o resolviendo ecuaciones integrales para flujos de Stokes [49].
- **Método de Elementos de Contorno (BEM):** Aceleración de la solución de sistemas lineales densos que surgen de la discretización de ecuaciones integrales de contorno [47, 50].

3.6.5. Ventajas y Desventajas/Limitaciones

Ventajas:

- **Eficiencia Asintótica:** Su complejidad $O(N)$ o $O(N \log N)$ lo hace significativamente más rápido que los métodos directos $O(N^2)$ para N grande [46].
- **Precisión Controlable:** La exactitud de la aproximación se puede ajustar sistemáticamente aumentando el orden p de las expansiones [49].
- **Escalabilidad:** Las versiones modernas han demostrado una excelente escalabilidad en arquitecturas paralelas de alto rendimiento [60].
- **Amplia Aplicabilidad:** Existen variantes para una gran diversidad de kernels de interacción física [47].

Desventajas/Limitaciones:

- **Complejidad de Implementación:** El algoritmo es considerablemente más complejo de implementar y depurar que los métodos directos o códigos de árbol más simples [47, 49].
- **Constante Oculta:** Aunque asintóticamente eficiente, la constante multiplicativa en la complejidad $O(N)$ puede ser grande, haciendo que para valores pequeños o intermedios de N , métodos más simples (como Barnes-Hut o incluso directos) puedan ser competitivos [59].
- **Consumo de Memoria:** Requiere almacenar las expansiones multipolares y locales en cada celda del árbol, lo que puede incrementar el uso de memoria en comparación con métodos directos [58].
- **Rendimiento en Distribuciones No Uniformes:** Las versiones no adaptativas pueden perder eficiencia si la distribución de partículas es muy irregular, aunque las variantes adaptativas mitigan este problema [54, 57].

Tabla 3.4: Comparación de complejidad computacional para problemas de N-cuerpos.

Método	Complejidad Computacional
Cálculo Directo	$O(N^2)$
Algoritmo Barnes-Hut	$O(N \log N)$
Método Multipolar Rápido (FMM)	$O(N)$ o $O(N \log N)$

Fuente: Basado en [46, 47, 49].

3.6.6. Simulación Barnes-Hut

La simulación Barnes-Hut es un algoritmo de aproximación ampliamente utilizado en el campo de la física computacional, especialmente en astrofísica, para simular la evolución dinámica de sistemas de N-cuerpos (*N-body systems*) bajo la influencia de fuerzas de largo alcance, como la gravedad o las interacciones electrostáticas [61, 62]. Fue propuesto originalmente por Josh Barnes y Piet Hut en 1986 [61] como una solución eficiente al problema computacionalmente intensivo del cálculo directo de todas las interacciones por pares en un sistema grande, cuya complejidad escala como $O(N^2)$. El algoritmo Barnes-Hut reduce esta complejidad a $O(N \log N)$,

permitiendo la simulación de sistemas con un número significativamente mayor de partículas [61, 63].

Principios y Funcionamiento

El principio fundamental del algoritmo Barnes-Hut radica en la aproximación de la fuerza ejercida por un grupo distante de partículas tratándolo como una única pseudo-partícula (o un conjunto de momentos multipolares de bajo orden) ubicada en el centro de masa (CoM) del grupo [61, 64]. Esta aproximación es válida porque el campo gravitatorio (o electrostático) de un grupo de partículas a gran distancia se asemeja al de una sola partícula puntual con la masa total del grupo ubicada en su centro de masa [65].

Para implementar esta idea, el algoritmo emplea una estructura de datos jerárquica basada en la subdivisión recursiva del espacio:

1. **Construcción del Árbol (*Tree Construction*):** El espacio que contiene las N partículas se subdivide recursivamente en celdas. En 3D, se utiliza un *octree* (árbol óctuple), donde el cubo que engloba todo el sistema se divide en ocho subcubos (octantes) iguales. Este proceso se repite para cada subcubo que contenga más de una partícula, hasta que cada celda hoja (nodo terminal del árbol) contenga como máximo una partícula [61, 62]. En 2D, se utiliza una estructura análoga llamada *quadtree* (árbol cuádruple) [66, 67]. Cada nodo interno del árbol almacena información agregada sobre las partículas contenidas en su celda correspondiente, como la masa total y la posición del centro de masa [61, 63].
2. **Cálculo de Fuerzas (*Force Calculation*):** Para calcular la fuerza neta sobre una partícula específica i , se recorre el árbol desde la raíz. Para cada nodo (celda) c encontrado durante el recorrido, se aplica el *criterio de apertura de Barnes-Hut (Barnes-Hut opening criterion)*. Este criterio compara el tamaño de la celda s (usualmente su anchura o diagonal) con la distancia d entre la partícula i y el centro de masa de la celda c . Si la relación s/d es menor que un parámetro de precisión umbral θ (theta), es decir:

$$s/d < \theta$$

la celda c se considera “suficientemente lejana”. En este caso, la contribución a la fuerza sobre la partícula i debida a todas las partículas dentro de la celda c se aproxima utilizando la masa total y el centro de masa (o una expansión multipolar de bajo orden) almacenados en el nodo c [61, 63, 64]. Si la condición no se cumple ($s/d \geq \theta$), la celda está demasiado cerca o es demasiado grande para ser aproximada con precisión, por lo que el algoritmo desciende recursivamente a los hijos de ese nodo (si es un nodo interno) y repite el proceso [61, 64]. Si el nodo es una hoja que contiene una partícula j (distinta de i), la fuerza entre i y j se calcula directamente [65]. El valor de θ (típicamente entre 0.5 y 1.2) controla el equilibrio entre precisión y velocidad: valores más pequeños de θ resultan en cálculos más precisos pero más lentos (más interacciones directas y nodos visitados), mientras que valores más grandes aceleran el cálculo a costa de una menor precisión [64, 66].

Contexto Histórico y Origen

El algoritmo fue presentado por Josh Barnes y Piet Hut en su artículo de 1986 en la revista *Nature*, titulado “A hierarchical $O(N \log N)$ force-calculation algorithm” [61]. Su desarrollo fue motivado por la necesidad de superar las limitaciones computacionales de las simulaciones N-cuerpos directas en astrofísica, que impedían estudiar sistemas a gran escala como la formación y dinámica de galaxias o cúmulos estelares [61, 62]. El algoritmo Barnes-Hut representó un avance significativo al hacer factibles simulaciones con cientos de miles o millones de partículas en la época.

Variantes y Enfoques Principales

Desde su concepción, el algoritmo Barnes-Hut ha sido objeto de diversas optimizaciones y variantes:

- **Implementaciones Paralelas:** Dada la naturaleza jerárquica y divisible del problema, el algoritmo se adapta bien a la parallelización. Se han desarrollado diversas estrategias para distribuir la construcción del árbol y el cálculo de fuerzas entre múltiples procesadores o nodos de cómputo, utilizando técnicas como la descomposición de dominio (ej. *Bisección Recursiva Ortogonal ORB* [63]) o la distribución de partículas basada en carga computacional [62, 68, 69].
- **Aceleración por GPU:** Las unidades de procesamiento gráfico (GPUs), con su arquitectura masivamente paralela, han demostrado ser muy eficaces para acelerar las simulaciones Barnes-Hut [70, 71]. Existen implementaciones donde partes significativas o la totalidad del algoritmo se ejecutan en la GPU [70].
- **Expansiones Multipolares:** Para mejorar la precisión de la aproximación de fuerzas de grupos distantes, en lugar de usar solo el monopolo (masa total y CoM), se pueden incluir términos de orden superior como el cuadrupolo [64, 72]. Esto es especialmente relevante cuando se requiere una alta fidelidad en la simulación.
- **Combinación con otros Métodos:** En algunas simulaciones cosmológicas a gran escala, el algoritmo Barnes-Hut puede combinarse con métodos de malla de partículas (*Particle-Mesh*, PM) para manejar las fuerzas de largo alcance de manera eficiente, mientras que Barnes-Hut se encarga de las interacciones a corta y mediana escala [73].
- **Optimización de Árboles:** Se han explorado variantes en la construcción y recorrido del árbol, como los árboles $k\text{-}d$ o el uso de árboles duales (*dual-tree*) que recorren simultáneamente dos árboles para optimizar el cálculo de interacciones celda-celda en lugar de partícula-celda [74, 75].

Aplicaciones Típicas

El algoritmo Barnes-Hut y sus variantes se aplican en diversos campos científicos y técnicos:

- **Astrofísica:** Es una herramienta estándar para simulaciones de formación y evolución de galaxias, interacciones galácticas, dinámica de cúmulos estelares

y globulares, y formación de estructuras a gran escala en cosmología [62, 63, 73].

- **Dinámica Molecular:** Se utiliza para calcular eficientemente las interacciones electrostáticas y de Van der Waals (que también son de largo alcance) en simulaciones de sistemas moleculares grandes como proteínas, ADN o fluidos iónicos [65, 76].
- **Física de Plasmas:** Para simular la interacción entre partículas cargadas en plasmas [77].
- **Visualización de Datos y Aprendizaje Automático:** Se ha adaptado para acelerar algoritmos de reducción de dimensionalidad y visualización, como t-SNE (*t-distributed Stochastic Neighbor Embedding*), donde se modelan “fuerzas” atractivas y repulsivas entre puntos de datos en un espacio de baja dimensión [74, 75].
- **Graficación por Computadora y Diseño de Grafos:** Para algoritmos de diseño dirigido por fuerzas (*force-directed layout*), donde los nodos de un grafo se repelen y las aristas actúan como resortes, buscando una disposición espacial estéticamente agradable o funcional [78].

Ventajas y Limitaciones

Ventajas

- **Eficiencia Computacional:** Su complejidad $O(N \log N)$ permite simular sistemas mucho más grandes que los métodos directos $O(N^2)$ [61].
- **Versatilidad:** Aplicable a diferentes tipos de fuerzas de largo alcance (gravitatoria, electrostática) [65, 76].
- **Adaptabilidad:** La estructura de árbol se adapta naturalmente a distribuciones de partículas no uniformes (clusterizadas) [62, 68].
- **Paralelizable:** Se presta bien a la implementación en arquitecturas de cómputo paralelo (multi-núcleo, GPU, clusters) [62, 70].

Limitaciones

- **Aproximación:** Introduce errores en el cálculo de fuerzas, cuya magnitud depende del parámetro θ y de la distribución de partículas [64, 66]. No es adecuado para simulaciones que requieran una precisión extremadamente alta en las interacciones individuales (ej. dinámica de sistemas planetarios precisos a largo plazo).
- **Complejidad de Implementación:** Aunque conceptualmente más simple que otros métodos rápidos como el Método Multipolo Rápido (FMM), su implementación correcta, especialmente las versiones paralelas u optimizadas, requiere un esfuerzo considerable [62, 63].

- **Coste de Construcción del Árbol:** La construcción (o reconstrucción) del árbol en cada paso de tiempo representa una sobrecarga computacional que no existe en los métodos directos [63].
- **Condiciones de Contorno:** La implementación de condiciones de contorno periódicas, comunes en cosmología o dinámica molecular, es más compleja que en métodos basados en mallas [73].

3.7. REBOUND: Código N-cuerpos multipropósito de código abierto para dinámica colisional

REBOUND (Rein & Liu, 2012) [8] se define como un código numérico de N-cuerpos, multipropósito y de código abierto, distribuido bajo la licencia GPLv3 [8]. Fue diseñado primordialmente para abordar problemas de dinámica colisional, como los encontrados en anillos planetarios, pero es igualmente capaz de resolver el problema clásico de N-cuerpos en sistemas puramente gravitacionales [8] [Abstract, Sec. 1]. Su desarrollo buscó llenar un vacío existente en cuanto a códigos públicos disponibles para la simulación de dinámica colisional compleja [8] [Sec. 1].

3.7.1. Principios Fundamentales y Funcionamiento

La filosofía central de REBOUND es su alta modularidad [8] [Abstract, Sec. 2.2]. Está escrito enteramente en C (conforme al estándar ISO C99) y diseñado para compilar y ejecutarse en plataformas modernas compatibles con POSIX (como Linux, Unix y Mac OSX) [8] [Sec. 2, Sec. 2.1]. La estructura modular permite al usuario seleccionar y combinar diferentes componentes mediante el uso de enlaces simbólicos, sin necesidad de modificar el código fuente principal. Esto facilita la adaptación del código a una amplia variedad de problemas astrofísicos y de otras disciplinas [8] [Sec. 2.2]. Los módulos intercambiables incluyen:

1. **Integradores Numéricicos:** REBOUND implementa varios integradores, mayormente basados en el esquema simpléctico de segundo orden Drift-Kick-Drift (DKD). Es importante notar que la naturaleza simpléctica se pierde formalmente si se introducen aproximaciones en el cálculo de la gravedad, colisiones o fuerzas dependientes de la velocidad [8] [Sec. 3]. Los integradores principales son:
 - **Leap-frog:** Un integrador simpléctico estándar de segundo orden para marcos no rotacionales [8] [Sec. 3.1].
 - **Wisdom-Holman (WH):** Un mapeo simpléctico de variables mixtas que integra el movimiento kepleriano de forma exacta durante el paso de deriva (*drift*). Es muy preciso para problemas dominados por un potencial central $1/r$, pero computacionalmente más costoso que Leap-frog debido a la resolución iterativa de la ecuación de Kepler [8] [Sec. 3.2].
 - **Symplectic Epicycle Integrator (SEI):** Diseñado para la aproximación de Hill (utilizada en láminas de cizallamiento o *shearing sheets*),

opera en un marco rotacional y es computacionalmente rápido [8] [Sec. 3.3].

REBOUND utiliza un paso de tiempo (*timestep*) fijo para todas las partículas por defecto, lo cual es crucial para mantener las propiedades simplécticas de los integradores correspondientes. Es responsabilidad del usuario asegurar que el paso de tiempo sea suficientemente pequeño para la convergencia numérica [8] [Sec. 3].

2. **Cálculo de Gravedad:** Se proporcionan dos módulos principales para calcular las fuerzas gravitacionales [8] [Sec. 4]:

- **Suma Directa:** Calcula la interacción entre todos los pares de partículas activas (N_{active}) de forma exacta (ecuación 1 en [8]). Su costo computacional escala como $O(N \cdot N_{\text{active}})$, siendo eficiente solo para un número bajo de partículas masivas ($N_{\text{active}} \leq 10^2$) [8] [Sec. 4.1]. Permite incluir un parámetro de suavizado gravitacional (b) para evitar aceleraciones extremas en encuentros cercanos [8] [Eq. 1].
- **Octree (Árbol Octal):** Implementa el algoritmo de Barnes & Hut (1986) para aproximar las fuerzas de largo alcance, reduciendo la complejidad computacional a $O(N \log N)$ [8] [Sec. 4.2]. Agrupa jerárquicamente las partículas distantes y utiliza su centro de masa (y opcionalmente su tensor cuadrupolar, siguiendo a Hernquist, 1987) para calcular la fuerza, basándose en un criterio de ángulo de apertura (θ_{crit}). Este módulo está completamente paralelizado usando MPI (con descomposición estática de dominio y árboles esenciales distribuidos) y OpenMP [8] [Sec. 4.2, Sec. 7].

3. **Detección de Colisiones:** REBOUND incluye varios módulos para detectar colisiones, modeladas como interacciones entre esferas duras con un coeficiente de restitución ϵ especificado por el usuario [8] [Sec. 5]. Las colisiones detectadas en un paso de tiempo se resuelven en orden aleatorio para evitar sesgos [8] [Sec. 5]. Los métodos son:

- **Búsqueda Directa del Vecino Más Cercano:** Comprueba el solapamiento de todas las parejas de partículas al final del paso de tiempo. Escala como $O(N^2)$ [8] [Sec. 5.1].
- **Octree:** Utiliza la estructura del árbol octal para buscar vecinos cercanos y detectar solapamientos, con una complejidad promedio de $O(N \log N)$. Puede usar la misma estructura del árbol de gravedad y está paralelizado (MPI/OpenMP) [8] [Sec. 5.2].
- **Algoritmo de Barrido Plano (Plane-Sweep):** Dos variantes (cartesiana en x y angular en ϕ) basadas en el algoritmo de Bentley & Ottmann (1979), pero simplificadas. Aproxima las trayectorias de las partículas como líneas rectas durante un paso de tiempo. Mueve un plano conceptual a través del dominio, manteniendo una lista de partículas activas intersectadas por el plano y comprobando colisiones solo dentro de esa lista. Su complejidad es $O(N \cdot N_{\text{SWEPL}})$, donde N_{SWEPL} es el número promedio de partículas en la lista activa. Es sig-

nificativamente más eficiente que los métodos basados en árboles para simulaciones quasi-bidimensionales o muy elongadas (donde N_{SWEEP} es pequeño), como anillos planetarios densos o simulaciones en láminas de cizallamiento [8] [Sec. 5.3].

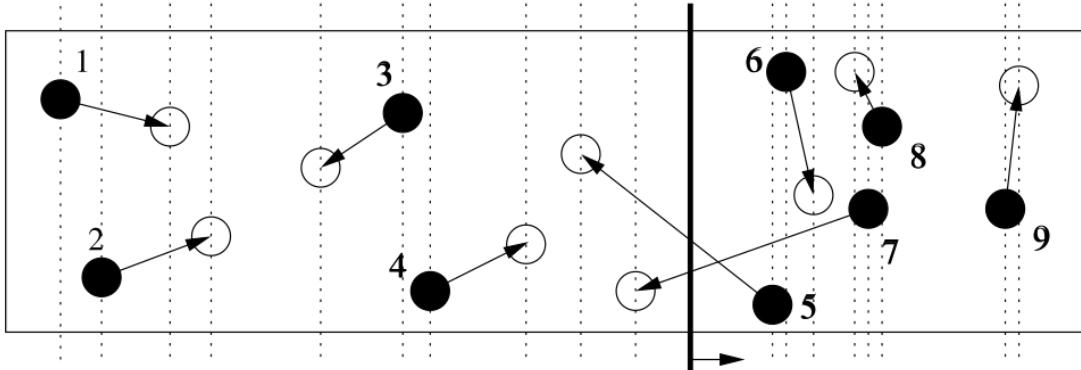


Figura 3.3: Ilustración del algoritmo de barrido plano. El plano interseca las trayectorias de las partículas 5 y 7. Véanse los detalles en el texto adaptado de [8].

4. **Condiciones de Contorno:** Soporta condiciones de contorno abiertas (las partículas que salen son eliminadas), periódicas (implementadas con cajas fantasma) y de lámina de cizallamiento (*shear-periodic*) [8] [Sec. 2.3].

3.7.2. Contexto Histórico y Origen

El código fue presentado por H. Rein y S.-F. Liu en 2012 [8], aunque versiones precursoras ya se habían utilizado en publicaciones anteriores (e.g., Rein & Papaloizou, 2010; Crida et al., 2010; Rein et al., 2010) [8] [Sec. 1]. Nació de la necesidad de contar con una herramienta pública, flexible y eficiente para estudiar la dinámica de sistemas astrofísicos donde las colisiones juegan un papel crucial, como los anillos de Saturno o los discos protoplanetarios [8] [Sec. 1].

3.7.3. Variantes y Enfoques Principales

Más que variantes distintas del código, REBOUND ofrece diferentes enfoques de simulación a través de la combinación de sus módulos. El usuario puede configurar la simulación eligiendo el integrador más adecuado (e.g., WH para alta precisión a largo plazo en sistemas planetarios, SEI para láminas de cizallamiento, Leap-frog como opción general), el método de cálculo de gravedad (directo para pocos cuerpos, árbol para muchos) y el algoritmo de detección de colisiones (directo, árbol, o barrido plano según la geometría y densidad del problema) [8] [Sec. 2.2, Sec. 3, Sec. 4, Sec. 5].

3.7.4. Aplicaciones Típicas

La literatura documenta el uso de REBOUND en una variedad de problemas astrofísicos:

- Simulación de anillos planetarios, incluyendo el estudio de su viscosidad y estructuras inducidas por satélites [8] [Sec. 1, Sec. 6.4].

- Formación de planetesimales a partir de enjambres de partículas en discos protoplanetarios [8] [Sec. 1].
- Estudio de discos de transición y de escombros (utilizando superpartículas) [8] [Sec. 1].
- Problemas clásicos de N-cuerpos, como la integración a largo plazo de la dinámica del Sistema Solar [8] [Sec. 1, Sec. 6.3].

Aunque diseñado para astrofísica, su estructura modular lo hace potencialmente aplicable a otros campos que involucren dinámica de partículas, como la dinámica molecular o flujos granulares [8] [Abstract].

3.7.5. Ventajas y Limitaciones

Las principales **ventajas** de REBOUND, según se desprenden de [8], son:

- **Código Abierto y Gratuito:** Accesible para la comunidad científica bajo licencia GPLv3 [8] [Sec. 1].
- **Modularidad y Flexibilidad:** Altamente personalizable para diferentes problemas [8] [Abstract, Sec. 2.2].
- **Eficiencia y Escalabilidad:** Demuestra buen escalado en paralelo (tanto fuerte como débil) usando MPI y OpenMP, permitiendo simulaciones con un gran número de partículas en clústeres de cómputo [8] [Abstract, Sec. 7].
- **Variedad de Métodos:** Ofrece múltiples opciones de integradores, cálculo de gravedad y detección de colisiones [8] [Sec. 3, Sec. 4, Sec. 5].
- **Precisión:** Los integradores simplécticos son adecuados para integraciones a largo plazo, y se conserva la energía y el momento hasta la precisión de la máquina en colisiones elásticas [8] [Sec. 3, Sec. 6.2].

Las **limitaciones** inherentes o aspectos a considerar son:

- **Paso de Tiempo Fijo:** Generalmente requerido por los integradores simplécticos, lo que puede ser ineficiente si ocurren escalas de tiempo muy diferentes en el sistema [8] [Sec. 3]. El usuario debe elegirlo cuidadosamente.
- **Pérdida de Simplicidad:** Las propiedades simplécticas se pierden si se usan aproximaciones (árbol de gravedad), colisiones o fuerzas no conservativas [8] [Sec. 3].
- **Aproximaciones:** Los métodos de árbol para gravedad y colisiones son aproximados, con una precisión que depende de parámetros como θ_{crit} [8] [Sec. 4.2, Sec. 6.1]. El método de barrido plano aproxima las trayectorias como líneas rectas entre pasos [8] [Sec. 5.3, Fig. 2].
- **Rendimiento:** El rendimiento del árbol puede degradarse si el número de partículas por nodo MPI es muy bajo debido al costo de comunicación [8] [Sec. 7.1]. La eficiencia del barrido plano depende crucialmente de la dimensionalidad efectiva del problema [8] [Sec. 5.3, Sec. 9].

3.8. Programación Matemática

El concepto de Programación Matemática¹ constituye una disciplina fundamental y contemporánea dentro del campo de las Matemáticas Aplicadas. Se define como el área dedicada al estudio y desarrollo de la teoría, los algoritmos y las aplicaciones para la resolución de problemas de optimización, donde se busca seleccionar la mejor alternativa entre un conjunto de opciones disponibles, sujeta a un conjunto de restricciones [79, p. 1], [80, p. 1]. En esencia, la Programación Matemática se enfoca en la identificación y determinación de la solución más ventajosa o eficiente para un problema específico, considerando un conjunto predefinido de restricciones que delinean el espacio de alternativas factibles [79, p. 1]. La optimización, por lo tanto, se erige como el núcleo central de la Programación Matemática, buscando la asignación más efectiva de recursos escasos para alcanzar objetivos concretos. *Management Science*² (Ciencia de la Gestión), una disciplina caracterizada por un enfoque científico para la toma de decisiones gerenciales, utiliza la Programación Matemática como una de sus herramientas más poderosas, aplicando métodos matemáticos y las capacidades de las computadoras modernas a los problemas complejos y no estructurados que enfrentan los gestores [81, p. 1].

Desde una perspectiva formal, la Programación Matemática implica la formulación de una representación abstracta del problema a través de un modelo matemático, seguido por la aplicación de algoritmos de optimización para la determinación de los valores óptimos que deben adoptar las variables de decisión dentro de dicho modelo [82, p. 175]. Un problema de optimización general se puede plantear como la selección de variables de decisión x_1, x_2, \dots, x_n de una región factible dada, de tal manera que se optimice (minimice o maximice) una función objetivo $f(x_1, x_2, \dots, x_n)$ [83, p. 410].

Los principios fundamentales de la Programación Matemática se estructuran en torno a los componentes esenciales que definen cualquier problema de optimización. Estos incluyen:

1. **Variables de Decisión:** Cantidades que el tomador de decisiones controla y cuyos valores determinan la solución del modelo [81, p. 181].
2. **Función Objetivo:** Una medida de rendimiento o efectividad, expresada como una función matemática de las variables de decisión, que se busca maximizar o minimizar [81, p. 181].
3. **Restricciones:** Un conjunto de relaciones (ecuaciones o inecuaciones) que las variables de decisión deben satisfacer, reflejando las limitaciones impuestas por la naturaleza del problema (disponibilidad de recursos, requisitos tecnológicos, etc.) [81, p. 181].

El objetivo primordial es, por lo tanto, determinar los valores óptimos de las variables de decisión que no solo satisfacen todas las restricciones impuestas al

¹**Programación Matemática:** Rama de las matemáticas aplicadas que se ocupa de la teoría y los métodos para resolver problemas de optimización, es decir, encontrar el mejor valor (máximo o mínimo) de una función objetivo sujeta a un conjunto de restricciones.

²**Management Science (Ciencia de la Gestión):** Disciplina que utiliza un enfoque científico, a menudo cuantitativo, para la toma de decisiones gerenciales y la resolución de problemas en organizaciones.

sistema, sino que también logran el mejor valor posible para la función objetivo.

El desarrollo formal de la Programación Matemática, especialmente la Programación Lineal, se consolidó a mediados del siglo XX. George B. Dantzig es ampliamente reconocido por el desarrollo del *algoritmo simplex* alrededor de 1947, que proporcionó un método sistemático y eficiente para resolver problemas de Programación Lineal [79, p. 2], [81, p. 1]. Este avance fue un punto de inflexión, impulsado en gran medida por la necesidad de optimizar la asignación de recursos durante la Segunda Guerra Mundial en el contexto de la *Investigación Operativa*³ (Operations Research) [79, p. 1]. Sin embargo, los fundamentos de la optimización se pueden rastrear a trabajos mucho anteriores de matemáticos como Lagrange y Fourier [79, p. 1]. En la década de 1950, surgieron avances significativos en Programación No Lineal con las contribuciones de Kuhn y Tucker sobre condiciones de optimalidad, y en Programación Dinámica gracias a Richard Bellman [79, p. 9]. Ralph Gomory realizó trabajos pioneros en Programación Entera durante el mismo período [79, p. 9].

3.8.1. Variantes Principales de la Programación Matemática

La Programación Matemática abarca diversas variantes, cada una diseñada para tipos específicos de problemas:

1. **Programación Lineal (PL):** Se caracteriza porque tanto la función objetivo como todas las restricciones son funciones lineales de las variables de decisión [79, p. 2], [81, p. 9]. Los principios fundamentales de la PL incluyen la proporcionalidad (la contribución de cada variable es proporcional a su valor), la aditividad (la contribución total es la suma de las contribuciones individuales), la divisibilidad (las variables pueden tomar valores fraccionarios) y la certeza (todos los parámetros del modelo son conocidos con exactitud) [81, p. 9]. La forma general de un problema de PL es:

$$\text{Optimizar } z = \sum_{j=1}^n c_j x_j$$

Sujeto a:

$$\begin{aligned} \sum_{j=1}^n a_{ij} x_j (\leq, =, \geq) b_i, & \quad \text{para } i = 1, \dots, m_1 + m_2 + m_3 \\ x_j \geq 0 & \quad (\text{para algunas o todas las } j) \\ x_j \text{ irrestricta} & \quad (\text{para algunas o todas las } j) \end{aligned}$$

[79, p. 117].

2. **Programación No Lineal (PNL):** Se aplica cuando la función objetivo o al menos una de las restricciones (o ambas) es una función no lineal de las variables de decisión [79, p. 9], [83, p. 410]. A diferencia de la PL, donde el óptimo (si existe) se encuentra en un vértice de la región factible, en PNL el óptimo puede estar en el interior de la región, en su frontera (no

³**Investigación Operativa (Operations Research):** Disciplina que aplica métodos analíticos avanzados para ayudar a tomar mejores decisiones. Surgió durante la Segunda Guerra Mundial para resolver problemas militares complejos.

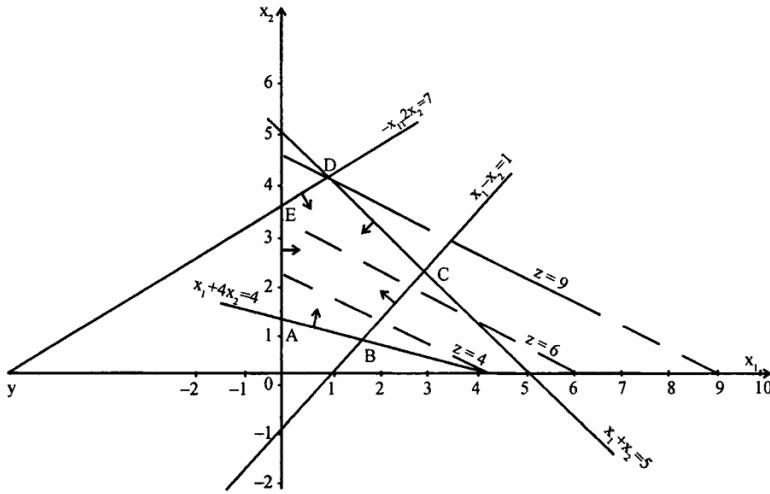


Figura 3.4: Representación bidimensional de la región factible de un problema de programación lineal, caracterizada por un polígono convexo definido por restricciones lineales. Se incluyen líneas de nivel de la función objetivo y el punto óptimo ubicado en un vértice del conjunto factible, ilustrando la solución típica en problemas lineales. Adaptado de [79, p. 6-7]

necesariamente un vértice), o en un vértice [83, p. 413, Figura 13.1]. Una distinción crucial en PNL es entre óptimos locales y globales. Un óptimo local es una solución mejor que todas las soluciones “cercanas”, mientras que un óptimo global es la mejor solución en toda la región factible [83, p. 413].

Dentro de la PNL, la *Programación Convexa* es de particular importancia. Si un problema de minimización involucra minimizar una función convexa sobre un conjunto convexo, o un problema de maximización implica maximizar una función cóncava sobre un conjunto convexo, entonces cualquier óptimo local es también un óptimo global [83, p. 418], [84, p. 1].

- **Programación Cuadrática:** Un caso especial de PNL donde la función objetivo es cuadrática y las restricciones son lineales [83, p. 419]. Su forma general es:

$$\begin{aligned} \text{Maximizar } f(x) &= \sum_{j=1}^n c_j x_j + \frac{1}{2} \sum_{j=1}^n \sum_{k=1}^n q_{jk} x_j x_k \\ \text{Sujeto a:} \end{aligned}$$

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad \text{para } i = 1, \dots, m$$

$$x_j \geq 0, \quad \text{para } j = 1, \dots, n$$

[83, p. 433].

3. **Programación Entera (PE):** Se caracteriza porque algunas o todas las variables de decisión están restringidas a tomar valores enteros [79, p. 9], [85, p. 272]. Si todas las variables son enteras, se denomina *Programación*

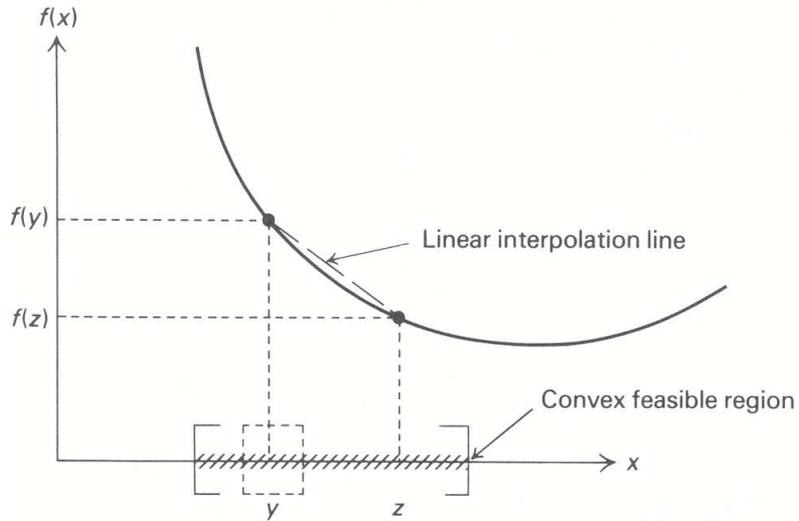


Figura 3.5: Visualización de una función no lineal con múltiples extremos locales y un único óptimo global, empleada para ilustrar las diferencias conceptuales entre mínimos (o máximos) locales y globales en problemas de programación no lineal. Esta distinción es clave en el análisis de algoritmos de optimización y su convergencia. Adaptado de [81, p. 419]

Entera Pura; si solo algunas lo son, es *Programación Entera Mixta*. Un caso especial importante es la *Programación Binaria* o *Programación 01*, donde las variables solo pueden tomar los valores 0 o 1, útil para modelar decisiones de tipo sí/no (go/no-go) [85, p. 273].

4. **Programación Dinámica (PD):** Es un enfoque que transforma un problema complejo en una secuencia de subproblemas más simples, interrelacionados. Su característica esencial es la naturaleza multietapa del procedimiento de optimización [86, p. 320]. Se basa en el *Principio de Optimalidad de Bellman*: una política óptima tiene la propiedad de que, cualesquiera que sean el estado y la decisión iniciales, las decisiones restantes deben constituir una política óptima con respecto al estado resultante de la primera decisión [86, p. 323]. La PD utiliza ecuaciones recursivas para relacionar la solución óptima de un problema de n etapas con la de un problema de $(n - 1)$ etapas [86, p. 325].
5. **Programación Estocástica:** Aborda problemas de optimización donde algunos de los parámetros del modelo (coeficientes de la función objetivo, lado derecho de las restricciones o coeficientes tecnológicos) son inciertos y se representan como variables aleatorias con distribuciones de probabilidad conocidas [79, p. 432]. Los enfoques incluyen la optimización del valor esperado o la incorporación de restricciones probabilísticas (chance constraints).

3.8.2. Aplicaciones Típicas

La Programación Matemática se aplica extensamente en diversos campos:

- **Investigación Operativa:** Planificación de la producción, gestión de in-

ventarios, problemas de transporte y asignación, secuenciación de tareas, optimización de rutas [79, p. 3-5], [81, p. 1-2].

- **Economía y Finanzas:** Optimización de portafolios de inversión, asignación de capital, modelos de equilibrio económico, planificación del desarrollo económico [81, p. 100-102], [83, p. 411].
- **Ingeniería:** Diseño óptimo de estructuras, planificación de recursos hídricos, optimización de procesos químicos, problemas de carga de redes eléctricas [83, p. 412].
- **Logística:** Localización de almacenes, diseño de redes de distribución, gestión de cadenas de suministro [85, p. 274].

En el ámbito de la física computacional y la simulación, aunque el término no siempre se use explícitamente, la optimización es fundamental. Por ejemplo, en la minimización de energía en sistemas moleculares, la optimización de parámetros en modelos físicos, o la búsqueda de configuraciones óptimas en simulaciones de N-cuerpos. Los métodos de Programación Matemática, como la Programación Cuadrática, son relevantes para problemas de ajuste de curvas (regresión restringida) en el análisis de datos experimentales [83, p. 412].

3.8.3. Ventajas y Desventajas

- **Ventajas:**

- Proporciona un marco estructurado para la toma de decisiones y la optimización de recursos limitados [81, p. 182].
- Permite modelar sistemas complejos y encontrar soluciones óptimas o cercanas a la óptima que pueden no ser intuitivas [81, p. 183].
- La teoría de la dualidad y el análisis de sensibilidad ofrecen información valiosa sobre el valor de los recursos (precios sombra) y la robustez de la solución ante cambios en los parámetros del modelo [81, p. 78-91].
- La disponibilidad de software eficiente permite resolver problemas de gran escala, especialmente en Programación Lineal [81, p. 182].

- **Desventajas/Limitaciones:**

- La formulación de un modelo matemático preciso puede ser compleja y consumir mucho tiempo, requiriendo una comprensión profunda del problema y de las técnicas de modelado [81, p. 183].
- Para problemas no lineales no convexos, los algoritmos pueden converger a óptimos locales en lugar de globales [83, p. 413].
- La Programación Entera, especialmente para problemas grandes, puede ser computacionalmente muy demandante (“NP-hard”) [85, p. 287].
- La suposición de certeza en los parámetros (en modelos determinísticos) puede no reflejar la realidad; aunque la Programación Estocástica aborda esto, incrementa la complejidad del modelo [79, p. 432].

- La calidad de la solución depende críticamente de la calidad y disponibilidad de los datos [81, p. 183].

3.9. Problemas de Factibilidad y Optimización

Los problemas de factibilidad y optimización son pilares fundamentales en la modelización y resolución de una vasta gama de desafíos en la ingeniería, las ciencias computacionales, la economía y la investigación operativa. Un problema de optimización, en su forma más general, busca seleccionar la “mejor” alternativa de un conjunto de opciones posibles, sujeta a ciertas limitaciones [87, p. 1], [88, p. 4]. Por otro lado, un problema de factibilidad se centra en determinar si existe al menos una solución que satisfaga un conjunto dado de restricciones, sin considerar necesariamente un criterio de optimalidad [87, p. 130], [89, p. 1]. Esta sección definirá formalmente ambos tipos de problemas, explorará su interrelación y cómo los problemas de factibilidad pueden ser vistos como un subconjunto de los problemas de optimización.

3.9.1. Problemas de Optimización

Un **problema de optimización matemática**⁴ se puede expresar en su forma estándar como [87, p. 127], [88, p. 89]:

$$\text{minimizar} \quad f_0(x) \quad (3.14)$$

$$\text{sujepto a} \quad g_i(x) \leq 0, \quad i = 1, \dots, m \quad (3.15)$$

$$h_j(x) = 0, \quad j = 1, \dots, p \quad (3.16)$$

donde:

- $x \in \mathbb{R}^n$ es el **vector de variables de optimización**⁵ (o variables de decisión).
- $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$ es la **función objetivo**⁶, la cual se desea minimizar (o maximizar, lo que es equivalente a minimizar $-f_0$).
- $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$, para $i = 1, \dots, m$, son las **funciones de restricción de desigualdad**.
- $h_j : \mathbb{R}^n \rightarrow \mathbb{R}$, para $j = 1, \dots, p$, son las **funciones de restricción de igualdad** (usualmente deben ser afines en problemas de optimización convexa).

⁴**Problema de optimización matemática:** También conocido como problema de programación matemática. Consiste en seleccionar la mejor solución (según algún criterio cuantificado por la función objetivo) de un conjunto de alternativas factibles (definidas por las restricciones) [87, p. 1].

⁵**Vector de variables de optimización:** El vector $x = (x_1, \dots, x_n)$ contiene las cantidades que se pueden elegir o variar para encontrar la solución óptima. La dimensión n es el número de variables escalares [87, p. 1].

⁶**Función objetivo:** La función $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$ cuyo valor se busca minimizar (o maximizar). Cuantifica el rendimiento, costo, utilidad o mérito asociado a una elección particular de las variables de optimización [87, p. 1].

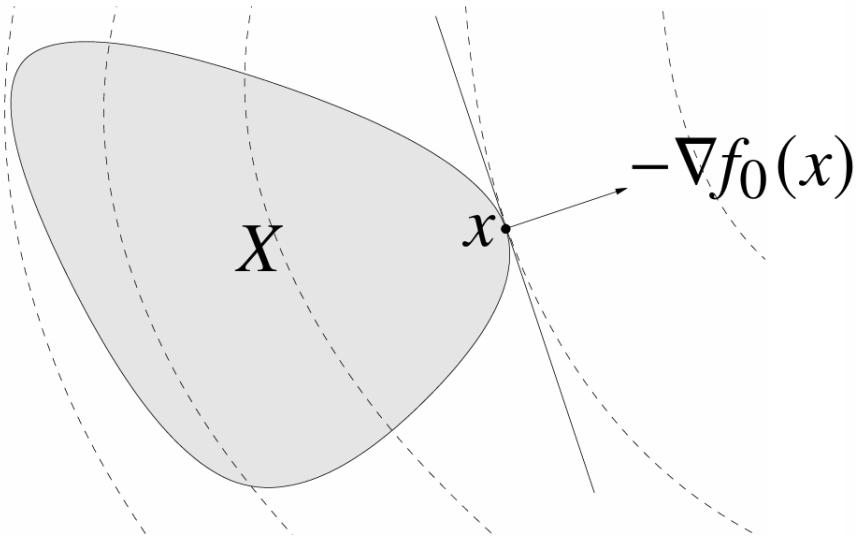


Figura 3.6: Gráfica 2D que muestre contornos de una función objetivo $f_0(x)$, una región factible sombreada definida por restricciones, y el punto óptimo x^* .

El conjunto $\mathcal{D} = (\bigcap_{i=0}^m \text{dom}(f_i)) \cap (\bigcap_{j=1}^p \text{dom}(h_j))$ es el **dominio** del problema. Un punto $x \in \mathcal{D}$ es **factible** si satisface todas las restricciones. El conjunto de todos los puntos factibles se denomina **conjunto factible** o **región factible** [87, p. 128], [88, p. 90]. Si el conjunto factible es vacío, el problema es **infactible**. El valor $p^* = \inf\{f_0(x) \mid x \text{ es factible}\}$ es el **valor óptimo**. Si existe un x^* tal que $f_0(x^*) = p^*$, entonces x^* es un **punto óptimo** (global) [87, p. 128], [88, p. 90].

Los problemas de optimización se clasifican según la naturaleza de sus funciones objetivo y de restricción. Los **problemas de programación lineal (LP)**⁷ tienen funciones objetivo y de restricción afines [87, p. 4], [88, p. 101]. Los **problemas de programación cuadrática (QP)**⁸ implican minimizar una función objetivo cuadrática convexa sobre un poliedro (definido por restricciones afines) [87, p. 152], [88, p. 105].

Un caso particularmente importante es la **optimización convexa**, donde la función objetivo f_0 y las funciones de restricción de desigualdad f_1, \dots, f_m son funciones convexas⁹, y las funciones de restricción de igualdad h_j son afines (es

⁷**Programación Lineal (LP):** Un problema de optimización donde tanto la función objetivo f_0 como todas las funciones de restricción f_i y h_j son afines (lineales más una constante) [87, p. 4], [88, p. 101].

⁸**Programación Cuadrática (QP):** Un problema de optimización donde la función objetivo f_0 es cuadrática (y usualmente convexa para garantizar tratabilidad) y las funciones de restricción f_i y h_j son afines [87, p. 152], [88, p. 105].

⁹**Función convexa:** Una función $f : \mathbb{R}^n \rightarrow \mathbb{R}$ es convexa si su dominio $\text{dom}(f)$ es un conjunto convexo y para todo $x, y \in \text{dom}(f)$ y todo θ con $0 \leq \theta \leq 1$, se cumple la desigualdad de Jensen: $f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$ [87, p. 67], [88, p. 46]. Geométricamente, el segmento de línea que une dos puntos cualesquiera $(x, f(x))$ e $(y, f(y))$ en la gráfica de la función se encuentra por encima o sobre la gráfica de f .

decir, de la forma $Ax = b$) [87, p. 7], [88, pp. 13, 95]. La optimización convexa es fundamental porque, a diferencia de los problemas de optimización no lineal generales, cualquier óptimo local es también un óptimo global, y existen algoritmos eficientes y confiables para resolverlos [87, pp. 8-9], [88, pp. 10-11, 97]. Otros tipos relevantes incluyen la optimización no lineal (general), la optimización discreta (o entera), la optimización estocástica y la optimización robusta [87, pp. 9-10].

Las aplicaciones típicas de la optimización matemática son vastas e incluyen el diseño de ingeniería (e.g., diseño de dispositivos, estructuras), el análisis y ajuste de datos (e.g., regresión, clasificación), la economía (e.g., modelos de equilibrio), y la gestión y finanzas (e.g., optimización de carteras, planificación de producción) [87, pp. 2-3]. Por ejemplo, en el ajuste de datos, x puede representar los parámetros de un modelo y $f_0(x)$ una medida del error de predicción, a menudo con un término de regularización para penalizar la complejidad del modelo y evitar el sobreajuste [88, p. 6]. En la optimización de carteras, x podría representar las asignaciones de inversión en diferentes activos, las restricciones podrían incluir el presupuesto total y los requisitos mínimos de retorno, y $f_0(x)$ podría ser el riesgo (e.g., la varianza del retorno) que se busca minimizar [87, p. 2].

3.9.2. Problemas de Factibilidad

Un **problema de factibilidad** consiste simplemente en encontrar un punto x que satisfaga un conjunto de restricciones, sin una función objetivo explícita que optimizar [87, p. 130], [89, p. 1]:

$$\text{encontrar } x \tag{3.17}$$

$$\text{sujeto a } g_i(x) \leq 0, \quad i = 1, \dots, m \tag{3.18}$$

$$h_j(x) = 0, \quad j = 1, \dots, p \tag{3.19}$$

La pregunta fundamental es si el conjunto factible (el conjunto de todos los x que satisfacen las restricciones) es no vacío. Si existe al menos un x que cumple todas las restricciones, el problema es **factible** y cualquier x de este tipo es una solución. De lo contrario, si el conjunto factible es vacío, el problema es **infactible** [87, p. 129], [88, p. 94].

Sidford [89, p. 2] presenta una definición más algorítmica para el problema de factibilidad, particularmente en el contexto de los **métodos de planos cortantes**¹⁰. Para $0 < r < R$ y $n \geq 1$, el problema de factibilidad (r, R, n) se define de la siguiente manera: se dispone de un **oráculo de separación**¹¹ que, al

¹⁰**Métodos de planos cortantes (Cutting Plane Methods):** Una clase de algoritmos utilizados para resolver problemas de optimización (a menudo convexos o problemas de optimización entera). Funcionan refinando iterativamente una aproximación poliédrica del conjunto factible o del epígrafe de la función objetivo, añadiendo “planos cortantes” (restricciones de hiperplanos) que eliminan regiones del espacio de búsqueda que se garantiza que no contienen la solución óptima, sin eliminar la propia solución [89, p. 1].

¹¹**Oráculo de separación (Separation Oracle):** Para un conjunto convexo $C \subseteq \mathbb{R}^n$, un oráculo de separación es un procedimiento que, dado un punto $x \in \mathbb{R}^n$, o bien determina que $x \in C$, o bien devuelve un hiperplano que separa x de C . Es decir, encuentra un vector $a \in \mathbb{R}^n$ y un escalar $b \in \mathbb{R}$ tales que $a^T x > b$ mientras que $a^T z \leq b$ para todo $z \in C$ [89, p. 1]. En el contexto de la definición de Sidford, g_x es el vector normal a tal hiperplano separador.

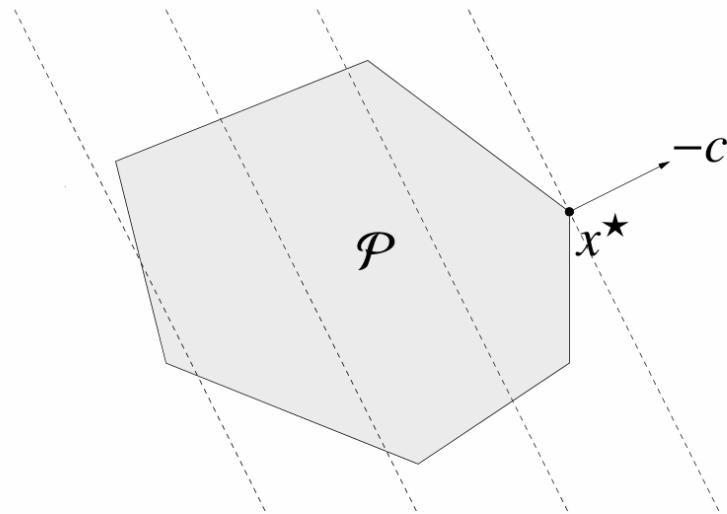


Figura 3.7: Representación bidimensional de un conjunto de restricciones lineales (líneas rectas) que definen una región poliédrica sombreada correspondiente al conjunto factible de un problema de programación lineal (LP). Se ilustran además los contornos paralelos de la función objetivo lineal y la dirección de optimización. Adaptada de [88, p. 101]

ser consultado con un punto $x \in B_\infty(R)$ (una bola en la norma infinito de radio R centrada en el origen), produce un vector $g_x \in \mathbb{R}^n$. El objetivo es uno de los siguientes:

1. Encontrar algún $x \in B_\infty(R)$ tal que $g_x = 0$ (este x se considera un “punto bueno” que satisface la condición de factibilidad implícita del oráculo).
2. Encontrar un conjunto de puntos $x_1, \dots, x_k \in B_\infty(R)$ tales que la región $S = B_\infty(R) \cap_{i \in [k]} \text{half}(g_{x_i}, g_{x_i}^T x_i)$ (la intersección de la bola inicial con los semiespacios definidos por los vectores g_{x_i}) no contenga ninguna bola de radio r .

Este enfoque es fundamental para algoritmos que iterativamente refinan una región de búsqueda. Utilizan los hiperplanos generados por el oráculo para “cortar” o eliminar porciones del espacio de búsqueda que se sabe que no contienen soluciones factibles, hasta que se encuentra una solución o se demuestra que la región de soluciones “buenas” (si existen) es demasiado pequeña para ser relevante (de radio menor que r) [89, pp. 1, 5].

Los problemas de factibilidad son cruciales en diversas áreas. Por ejemplo, en la minimización de funciones, muchos algoritmos de optimización, especialmente los métodos de punto interior, requieren un punto inicial factible (o incluso estrictamente factible) para comenzar [87, p. 579], [88, p. 372]. Determinar la existencia de tal punto es en sí mismo un problema de factibilidad. También surgen intrínsecamente en el estudio de oráculos de separación para conjuntos y funciones convexas, que son componentes básicos de muchos algoritmos de optimización eficientes [89, p. 1].

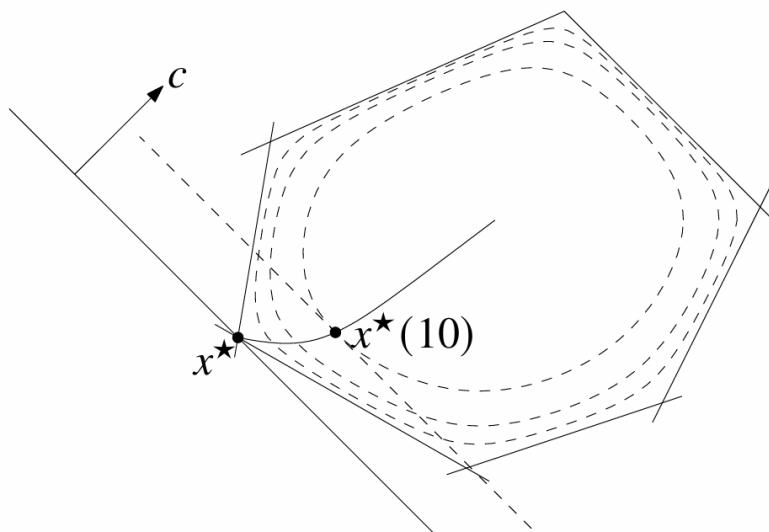


Figura 3.8: Representación bidimensional del conjunto factible, el punto óptimo y la trayectoria central $x^*(t)$ seguida por los métodos de barrera conforme el parámetro t incrementa. Se incluyen curvas de nivel que ilustran la función de barrera o, alternativamente, la función objetivo. Adaptada de [88, p. 361]

3.9.3. Relación y Contención

La relación entre los problemas de factibilidad y optimización es intrínseca y fundamental. Todo problema de optimización inherentemente contiene un problema de factibilidad subyacente: antes de poder encontrar la “mejor” solución (óptima), primero se debe determinar si existe *alguna* solución que satisfaga todas las restricciones impuestas. Si el conjunto factible es vacío, el problema de optimización es infactible y, por lo tanto, no tiene solución [87, p. 128], [88, p. 90].

Más formalmente, un **problema de factibilidad puede ser considerado un caso particular de un problema de optimización** [87, p. 130], [88, p. 94]. Esto se logra simplemente definiendo una función objetivo constante, por ejemplo, $f_0(x) = 0$ para todo x (o cualquier otra constante). En este caso, el objetivo de “minimizar” $f_0(x)$ se satisface trivialmente para cualquier x . Por lo tanto, cualquier punto factible x es también un punto óptimo, y el valor óptimo p^* será 0 si el problema es factible (es decir, si el conjunto factible no es vacío). Si el problema es infactible, el valor óptimo p^* será ∞ (asumiendo la convención de que el ínfimo de una función sobre un conjunto vacío es ∞) [87, p. 129], [88, p. 94].

Esta contención es a menudo explotada en la práctica algorítmica. Por ejemplo, muchos métodos eficientes para resolver problemas de optimización con restricciones de desigualdad, como los **métodos de barrera** (una clase importante de **métodos de punto interior**¹²), requieren un punto inicial estrictamente factible

¹²**Métodos de punto interior (Interior-point methods):** Una clase de algoritmos para resolver problemas de optimización lineal y no lineal convexa. A diferencia de los métodos como el simplex que se mueven a lo largo de los bordes del conjunto factible, los métodos de punto interior generan una secuencia de puntos estrictamente factibles en el *interior* de la región factible que convergen al óptimo. A menudo siguen una “trayectoria central” teórica, que es el conjunto de soluciones a una familia parametrizada de problemas de optimización

(un punto x tal que $f_i(x) < 0$ para todas las restricciones de desigualdad) [87, pp. 561, 568], [88, pp. 355, 367].

Si tal punto no se conoce de antemano, se emplea comúnmente un procedimiento de dos fases. La **fase I** consiste en formular y resolver un problema de optimización auxiliar cuyo objetivo principal es encontrar un punto estrictamente factible para el problema original. Si la fase I tiene éxito (es decir, encuentra un punto estrictamente factible), su solución se utiliza como punto de partida para la **fase II**, que consiste en resolver el problema de optimización original [87, p. 579], [88, p. 372]. Una formulación común del problema de fase I implica introducir una variable artificial s y minimizar s sujeto a las restricciones originales modificadas, como $f_i(x) \leq s$. Este problema de fase I es en sí mismo un problema de optimización (a menudo convexo si el original lo es) que busca determinar la factibilidad del problema original y encontrar un punto factible si existe [87, p. 579], [88, p. 373]. *[Aquí podría insertarse una figura conceptual del método de barrera y la trayectoria central, como la Figura 11.2 de [87, p. 565] o la Figura 9.1 de [88, p. 361], para ilustrar cómo, una vez asegurada la factibilidad (a través de la fase I o por conocimiento previo), el algoritmo procede hacia la optimalidad manteniéndose dentro de la región factible.]*

En resumen, la determinación de la factibilidad es un prerequisito lógico y a menudo algorítmico para la optimización. Un problema de optimización busca el “mejor” punto dentro del conjunto factible, mientras que un problema de factibilidad simplemente busca determinar si dicho conjunto no está vacío y, en caso afirmativo, encontrar cualquier punto en él. La capacidad de enmarcar problemas de factibilidad como problemas de optimización (con una función objetivo constante) permite aplicar el amplio espectro de teorías y herramientas algorítmicas desarrolladas para la optimización también a la resolución de problemas de factibilidad.

3.10. Algoritmos Genéticos (AGs) Simples Mono-objetivo mediante pymoo

3.10.1. Introducción a los Algoritmos Genéticos (AGs) Simples

La optimización, como disciplina fundamental, se dedica a la búsqueda de la mejor solución posible para un problema dado, considerando las restricciones existentes. Su importancia se manifiesta en una amplia gama de campos, desde la ingeniería y la ciencia hasta la economía y la inteligencia artificial, donde la necesidad de encontrar soluciones eficientes y efectivas es primordial. Dentro del vasto panorama de las técnicas de optimización, la computación evolutiva emerge como un paradigma de resolución de problemas inspirado en los mecanismos de la evolución natural [90]. Los Algoritmos Genéticos (AGs) constituyen una clase prominente de algoritmos evolutivos que han demostrado su eficacia en la

que incorporan una función de barrera (típicamente logarítmica) para las restricciones de desigualdad [87, Cap. 11], [88, Cap. 9 y 11].

resolución de problemas de optimización complejos [91].

La idea central detrás de los AGs radica en la simulación de los principios de la selección natural, la recombinación genética (cruce) y la mutación para mejorar iterativamente una población de soluciones candidatas [91]. De manera análoga a la evolución biológica, donde los individuos mejor adaptados tienen una mayor probabilidad de sobrevivir y reproducirse, los AGs evalúan la “aptitud” de cada solución candidata mediante una función objetivo. Las soluciones más aptas tienen una mayor probabilidad de ser seleccionadas para producir la siguiente generación, combinando su información genética a través del cruce y sufriendo mutaciones aleatorias. Este proceso iterativo, que se repite a lo largo de varias generaciones, busca converger hacia una solución óptima para el problema planteado. El presente informe se centra específicamente en la implementación canónica de AGs simples diseñados para abordar problemas de optimización con un único objetivo dentro del contexto de la biblioteca Pymoo [92].

Los AGs resultan particularmente útiles para problemas de optimización monoobjetivo que presentan características desafiantes para los métodos tradicionales basados en gradientes. Su capacidad para explorar espacios de búsqueda complejos, no lineales y no diferenciables los convierte en una herramienta valiosa en escenarios donde la función objetivo exhibe múltiples óptimos locales o una superficie de búsqueda irregular. A diferencia de los métodos deterministas que siguen una trayectoria específica guiada por el gradiente, los AGs operan sobre una población de soluciones y emplean operadores estocásticos, lo que les permite explorar una gama más amplia de posibles soluciones y ser más robustos frente a quedar atrapados en óptimos locales. Esta característica fundamental motiva su aplicación en una variedad de problemas del mundo real donde la complejidad de la función objetivo dificulta el uso de técnicas convencionales.

El término “canónico” alude a una forma estándar o ampliamente aceptada del AG simple [91]. Esta forma fundamental típicamente involucra la representación binaria de las soluciones candidatas, un mecanismo de selección proporcional a la aptitud (como la ruleta), un operador de cruce de un solo punto y un operador de mutación de inversión de bits. Si bien la biblioteca Pymoo puede ofrecer una mayor flexibilidad y una variedad de operadores más avanzados [92], comprender esta estructura básica es esencial para contextualizar la implementación específica que se analizará en este informe. Establecer esta base desde el principio permitirá al lector comprender mejor cómo la implementación de Pymoo se alinea con los principios fundamentales de un AG canónico y cómo potencialmente los extiende o adapta.

3.10.2. Descripción General de la Biblioteca Pymoo

Pymoo se presenta como una biblioteca moderna y rica en funcionalidades, escrita en Python y diseñada principalmente para abordar problemas de optimización multiobjetivo [92]. No obstante, su arquitectura versátil y su diseño modular la convierten también en una herramienta poderosa para la resolución de problemas de optimización con un único objetivo. Una de sus principales fortalezas reside en su capacidad para separar claramente los componentes de un problema de optimización (la función objetivo, las variables de decisión y las restricciones) de

los algoritmos de optimización propiamente dichos y de los operadores genéticos utilizados. Esta separación promueve la flexibilidad y la reutilización del código, permitiendo a los usuarios experimentar fácilmente con diferentes combinaciones de algoritmos y operadores para un mismo problema.

La biblioteca Pymoo destaca por su amplio soporte para una variedad de algoritmos de optimización, que abarcan tanto técnicas evolutivas, como los Algoritmos Genéticos, la Optimización por Enjambre de Partículas y la Evolución Diferencial, como también métodos clásicos de optimización [92]. Además, proporciona un conjunto completo de herramientas para la definición de problemas de optimización, la implementación de operadores genéticos personalizados y la evaluación del rendimiento de los algoritmos. Su extensibilidad es otra característica clave, ya que permite a los usuarios implementar fácilmente sus propios algoritmos, operadores o criterios de terminación si las opciones predefinidas no se ajustan a sus necesidades específicas.

Aunque el enfoque principal de Pymoo radica en la optimización multiobjetivo [92, 93], su marco de trabajo subyacente y muchos de sus operadores pueden aplicarse directamente o tienen contrapartidas específicas para el caso de la optimización con un solo objetivo. Esto significa que los usuarios interesados en resolver problemas con una única función objetivo pueden beneficiarse igualmente de la robustez y la flexibilidad que ofrece la biblioteca. La arquitectura de Pymoo se organiza en varios módulos clave que interactúan para definir y ejecutar los procesos de optimización. Estos módulos incluyen:

- **pymoo.core.problem**: Este módulo proporciona las clases base para definir problemas de optimización, especificando la función objetivo, el número y los límites de las variables de decisión, así como cualquier restricción que deba cumplirse.
- **pymoo.algorithms**: Contiene implementaciones de una amplia gama de algoritmos de optimización, incluyendo la clase GA para problemas monoobjetivo que será el foco de este informe.
- **pymoooperators**: Ofrece una colección diversa de operadores genéticos esenciales para los algoritmos evolutivos, como operadores de selección (para elegir los padres), operadores de cruce (para combinar la información genética) y operadores de mutación (para introducir variaciones aleatorias).
- **pymoo.termination**: Define diferentes criterios que pueden utilizarse para detener el proceso de optimización, como alcanzar un número máximo de generaciones o un cierto nivel de convergencia.
- **pymoo.optimize**: Proporciona funciones de alto nivel para facilitar la ejecución de los algoritmos de optimización una vez que se han definido el problema y el algoritmo deseado.

La filosofía de diseño de Pymoo, que separa claramente la definición del problema, la implementación del algoritmo y la aplicación de los operadores, promueve una flexibilidad y una capacidad de reutilización significativas [92]. Esto implica que un mismo problema de optimización podría potencialmente resolverse utilizando diferentes algoritmos mono-objetivo disponibles en Pymoo, lo que facilita la realiza-

ción de estudios comparativos de rendimiento. Esta capacidad de desacoplamiento permite a los usuarios experimentar con diversas combinaciones de algoritmos y operadores sin necesidad de reescribir la definición del problema cada vez, lo que representa una ventaja considerable tanto para la investigación como para las aplicaciones prácticas.

3.10.3. El Algoritmo Genético Canónico en Pymoo

Pymoo proporciona una implementación fácilmente accesible de un Algoritmo Genético mono-objetivo que se alinea de cerca con la forma canónica descrita anteriormente [91]. Dentro de la biblioteca, la clase `pymoo.algorithms.soo.nonconvex.ga.GA`, ubicada en el módulo `pymoo.algorithms.soo.nonconvex`, constituye el foco principal para la discusión de la implementación canónica de un AG mono-objetivo [92]. La designación `soo` en el nombre del módulo indica que se trata de un algoritmo diseñado para la Optimización con un Solo Objetivo, mientras que `nonconvex` sugiere que es adecuado para problemas donde la función objetivo puede no ser convexa, una característica común en los escenarios donde los AGs suelen ser efectivos.

Si bien Pymoo ofrece una variedad de otros algoritmos de optimización mono-objetivo, la clase `GA` se destaca como una representación directa de un AG estándar, incorporando los componentes fundamentales y el flujo de trabajo típico de este tipo de algoritmo evolutivo. El proceso general para utilizar la clase `GA` en Pymoo sigue una serie de pasos bien definidos:

- **Definición del Problema de Optimización:** El primer paso crucial consiste en definir el problema específico que se desea resolver. Esto se realiza mediante la creación de una instancia de la clase `Problem` de Pymoo (o una de sus subclases), donde se especifica la función objetivo que se va a minimizar o maximizar, el número de variables de decisión involucradas y los límites inferior y superior para cada una de estas variables. Adicionalmente, si el problema incluye alguna restricción que las soluciones deben satisfacer, estas también se definen en este paso.
- **Creación de una Instancia del Algoritmo Genético:** Una vez definido el problema, se procede a crear una instancia de la clase `GA`. Durante esta instanciación, se deben especificar varios parámetros que controlan el comportamiento del algoritmo, como el tamaño de la población (`pop_size`), el método de muestreo para generar la población inicial (`sampling`), el operador de cruce (`crossover`) que se utilizará para combinar la información genética de los padres y el operador de mutación (`mutation`) que se aplicará para introducir variaciones aleatorias en la descendencia.
- **Ejecución del Algoritmo:** Con el problema definido y el algoritmo instanciado y configurado, el siguiente paso es ejecutar el proceso de optimización. Esto se logra llamando al método `solve()` de la instancia de la clase `GA`, pasando como argumento el objeto que representa el problema de optimización definido en el paso anterior.
- **Análisis de los Resultados:** Una vez que el algoritmo ha terminado de ejecutarse (ya sea por alcanzar un criterio de terminación predefinido o

por completarse el número máximo de generaciones), el método `solve()` devuelve un objeto de tipo `Result` que contiene información relevante sobre la ejecución de la optimización. Esta información típicamente incluye la mejor solución encontrada (los valores de las variables de decisión que producen el mejor valor de la función objetivo) y el valor correspondiente de la función objetivo.

Es importante señalar que, si bien la clase `pymoo.algorithms.soo.nonconvex.ga.GA` es una representación fundamental de un AG canónico dentro de Pymoo [92], la flexibilidad de la biblioteca podría permitir la existencia de otras implementaciones o variantes de AGs, ya sea dentro de la propia biblioteca o aportadas por la comunidad de usuarios. No obstante, para los fines de este informe y con el objetivo de proporcionar una comprensión clara de la forma “canónica”, nos centraremos en el análisis detallado de esta clase específica debido a su estrecha correspondencia con la estructura tradicional de un AG [91].

3.10.4. Análisis Detallado de la Clase `pymoo.algorithms.soo.nonconvex.ga.GA`

La clase `pymoo.algorithms.soo.nonconvex.ga.GA` se configura principalmente a través de su constructor, que acepta varios parámetros clave que definen el comportamiento del algoritmo genético. Estos parámetros permiten a los usuarios adaptar el algoritmo a las características específicas del problema que se está abordando. A continuación, se presenta un análisis detallado de los parámetros más importantes del constructor:

- **`pop_size`:** Este parámetro de tipo entero especifica el número de individuos que conformarán la población en cada generación del algoritmo. El tamaño de la población influye significativamente en el rendimiento del AG. Una población más grande permite una mayor exploración del espacio de búsqueda, lo que puede aumentar la probabilidad de encontrar la solución óptima, pero también incrementa el costo computacional. Por otro lado, una población más pequeña puede converger más rápidamente, pero corre el riesgo de quedar atrapada en óptimos locales. La elección del tamaño de la población suele ser un compromiso que depende de la complejidad del problema y los recursos computacionales disponibles.
- **`sampling`:** Este parámetro define el método que se utilizará para crear la población inicial de individuos. Acepta un objeto de una clase de muestreo proporcionada por Pymoo (generalmente del módulo `pymoo.operators.samplig`). Una opción común es `RandomSampling`, que genera individuos con valores de variables de decisión distribuidos uniformemente dentro de sus respectivos límites. La estrategia de muestreo inicial puede tener un impacto considerable en la diversidad inicial de la población, lo que a su vez puede afectar la capacidad del algoritmo para explorar eficazmente el espacio de búsqueda.
- **`crossover`:** Este parámetro especifica el operador de cruce que se aplicará durante el proceso de reproducción. Acepta un objeto de una clase de operador de cruce del módulo `pymoo.operators.crossover`. Pymoo ofrece

una variedad de operadores de cruce diseñados para diferentes tipos de representación de variables de decisión. Por ejemplo, para problemas con variables continuas, un operador común es `SBX` (Simulated Binary Crossover) [94], mientras que para problemas con variables binarias se podrían utilizar operadores como `SinglePointCrossover` o `TwoPointCrossover`. La elección del operador de cruce influye en cómo se combina la información genética de los padres para generar nuevos descendientes.

- **`mutation`:** De manera similar al parámetro `crossover`, este parámetro define el operador de mutación que se utilizará para introducir variaciones aleatorias en la población. Acepta un objeto de una clase de operador de mutación del módulo `pymoo.operators.mutation`. Al igual que con el cruce, Pymoo proporciona varios operadores de mutación adecuados para diferentes tipos de variables. Por ejemplo, `PolynomialMutation` se utiliza a menudo para variables continuas [95], mientras que `BitflipMutation` es común para variables binarias. El operador de mutación juega un papel crucial en el mantenimiento de la diversidad genética y en la prevención de la convergencia prematura del algoritmo.
- **`eliminate_duplicates`:** Este parámetro booleano indica si se deben eliminar los individuos duplicados de la población después de cada generación. Si se establece en `True`, el algoritmo verificará si existen individuos idénticos en la nueva población y los eliminará, manteniendo así una mayor diversidad genética. Si se establece en `False`, se permitirán los duplicados. La eliminación de duplicados puede ayudar a prevenir la convergencia prematura, pero también puede añadir una sobrecarga computacional al proceso.

La principal funcionalidad de la clase `GA` se expone a través de sus métodos, siendo el más importante de ellos el método `solve()`. Este método es el encargado de ejecutar el algoritmo genético y llevar a cabo el proceso iterativo de evolución de la población hasta que se cumple algún criterio de terminación. El método `solve()` requiere como entrada principal un objeto que represente el problema de optimización que se desea resolver. Este objeto debe ser una instancia de una subclase de `pymoo.core.problem.Problem` y debe definir la función objetivo, el número y los límites de las variables de decisión, y cualquier restricción que sea aplicable. La separación entre la definición del problema y la implementación del algoritmo es un aspecto fundamental del diseño de Pymoo, ya que permite una mayor flexibilidad y modularidad en el proceso de optimización [92].

Al finalizar su ejecución, el método `solve()` devuelve un objeto de tipo `pymoo.optimize.Result`. Este objeto contiene información valiosa sobre la ejecución del algoritmo, incluyendo el mejor individuo encontrado durante todo el proceso de optimización (accesible a través del atributo `X`) y el valor correspondiente de la función objetivo para ese individuo (accesible a través del atributo `F`). Además de la mejor solución, el objeto `Result` puede contener otra información útil, como el historial de la evolución de la población o métricas de rendimiento del algoritmo.

3.10.5. Componentes y Operadores Clave

- **Inicialización:** La primera etapa en la ejecución de un Algoritmo Genético es la creación de la población inicial de soluciones candidatas. En Pymoo, este proceso está determinado por el parámetro `sampling` que se especifica al instanciar la clase `GA`. El método de muestreo elegido define cómo se generan los individuos iniciales en el espacio de búsqueda del problema. Una estrategia común es `RandomSampling`, que genera soluciones distribuidas uniformemente dentro de los límites definidos para cada variable de decisión. Otras estrategias de muestreo, como el muestreo de hipercubo latino (LHS), también podrían estar disponibles en Pymoo y ofrecer diferentes formas de distribuir inicialmente las soluciones en el espacio de búsqueda. La diversidad de la población inicial es un factor crítico que puede influir significativamente en la capacidad del AG para encontrar el óptimo global. Una población inicial bien diversificada aumenta las posibilidades de explorar una región más amplia del espacio de búsqueda y de evitar la convergencia prematura hacia óptimos locales. La elección de una estrategia de muestreo apropiada debe basarse en las características del problema específico que se está abordando.
- **Evaluación:** Una vez que se ha generado la población inicial (y en cada generación subsiguiente), es necesario evaluar la calidad de cada individuo, es decir, qué tan bien resuelve el problema de optimización. En Pymoo, esta evaluación se realiza llamando a la función objetivo definida dentro del objeto `Problem` para cada individuo de la población. El valor resultante de la función objetivo (o valores, en el caso de optimización multiobjetivo, aunque aquí nos centramos en el caso mono-objetivo) representa la aptitud o “fitness” de ese individuo. Cuanto mejor sea el valor de la función objetivo (dependiendo de si se busca minimizar o maximizar), mayor será la aptitud del individuo.
- **Selección:** El operador de selección juega un papel fundamental en la evolución de la población al determinar qué individuos de la generación actual se elegirán como padres para producir la siguiente generación. El objetivo de la selección es favorecer a los individuos con mayor aptitud, aumentando así la probabilidad de que sus “genes” (es decir, sus características como soluciones) se transmitan a la siguiente generación. Pymoo implementa varios métodos de selección comunes [90, 91]:
 - *Selección por Torneo:* En este método, se selecciona aleatoriamente un pequeño grupo de individuos (el tamaño del torneo es un parámetro configurable). El individuo con la mejor aptitud dentro de este grupo se elige como parent. Este proceso se repite hasta que se ha seleccionado el número deseado de padres. El tamaño del torneo influye en la presión de selección: un tamaño mayor aumenta la probabilidad de que los individuos más aptos sean seleccionados.
 - *Selección por Ruleta (Selección Proporcional a la Aptitud):* En este método, a cada individuo se le asigna una probabilidad de ser seleccionado proporcional a su aptitud relativa en la población. Los individuos con mayor aptitud tienen una mayor probabilidad de ser elegidos. Sin embargo, este método puede presentar problemas si hay un individuo excepcionalmente apto en las primeras generaciones, ya que podría dominar el proceso de selección y llevar a una convergencia prematura.

- Pymoo podría ofrecer otros métodos de selección, como la selección basada en el rango, donde la probabilidad de selección se basa en el rango de aptitud del individuo en lugar de su valor absoluto. La elección del método de selección y sus parámetros asociados tiene un impacto significativo en la velocidad de convergencia y en el riesgo de convergencia prematura del algoritmo. Una mayor presión de selección puede acelerar la convergencia, pero también puede aumentar el riesgo de quedar atrapado en óptimos locales.
- **Cruce (Recombinación):** El operador de cruce, también conocido como recombinación, es un mecanismo clave en los Algoritmos Genéticos que permite combinar la información genética de dos individuos padres seleccionados para crear nuevos individuos descendientes [91]. El objetivo del cruce es explorar nuevas regiones del espacio de búsqueda mediante la creación de combinaciones de características que podrían no existir en los padres individualmente. Pymoo proporciona una variedad de operadores de cruce, algunos de los cuales son:
 - *Cruce de un Punto:* Para representaciones binarias o de enteros, se elige aleatoriamente un único punto de cruce. Los segmentos de los cromosomas de los dos padres se intercambian en este punto para producir dos nuevos descendientes.
 - *Cruce de Dos Puntos:* Similar al cruce de un punto, pero se eligen dos puntos de cruce. El segmento del cromosoma entre estos dos puntos se intercambia entre los dos padres.
 - *Cruce Binario Simulado (SBX):* Este operador se utiliza comúnmente para problemas con variables continuas [94]. Simula el comportamiento del cruce de un punto en el espacio de las variables reales, generando descendientes que se encuentran en la vecindad de los padres en el espacio de búsqueda. El SBX tiene un parámetro importante, el factor de dispersión, que controla qué tan lejos de los padres pueden estar los descendientes.
 - Pymoo puede incluir otros operadores de cruce, como el cruce uniforme, donde cada gen (variable) del descendiente se selecciona aleatoriamente de uno de los dos padres. La elección del operador de cruce debe ser compatible con la representación de las soluciones (por ejemplo, binaria, continua) y puede tener un impacto significativo en la capacidad del algoritmo para explorar y explotar el espacio de búsqueda. El operador de cruce y sus parámetros se especifican en Pymoo a través del parámetro `crossover` en el constructor de la clase `GA`.
- **Mutación:** El operador de mutación introduce pequeños cambios aleatorios en los individuos descendientes después del cruce. El propósito de la mutación es mantener la diversidad genética en la población y evitar la convergencia prematura hacia un óptimo local al introducir nuevo material genético que no estaba presente en la población de padres [90, 91]. Pymoo ofrece varios operadores de mutación:
 - *Mutación por Inversión de Bits:* Este operador se aplica típicamente a representaciones binarias. Para cada bit del cromosoma de un individuo,

existe una pequeña probabilidad (la tasa de mutación) de que su valor se invierta (de 0 a 1 o de 1 a 0).

- *Mutación Polinomial:* Este operador se utiliza a menudo para problemas con variables continuas [95]. Perturba el valor de cada variable de decisión en un individuo en función de una distribución polinomial. La magnitud de la perturbación está controlada por parámetros como la tasa de mutación y el índice de distribución.
- Pymoo puede proporcionar otros operadores de mutación, como la mutación gaussiana para variables continuas, donde se añade un pequeño valor aleatorio extraído de una distribución gaussiana al valor de la variable. La tasa de mutación es un parámetro crítico. Una tasa demasiado baja podría no introducir suficiente diversidad, mientras que una tasa demasiado alta podría perturbar las buenas soluciones y hacer que la búsqueda sea más aleatoria. El operador de mutación y sus parámetros se especifican en Pymoo mediante el parámetro `mutation` en el constructor de la clase GA.
- **Reemplazo:** Después de que se han generado los nuevos individuos descendientes a través de los procesos de selección, cruce y mutación, es necesario determinar cómo se formará la siguiente generación de la población. En un Algoritmo Genético simple, la estrategia de reemplazo más común es reemplazar toda la población de padres por la población de hijos. Sin embargo, Pymoo podría ofrecer opciones para implementar el elitismo, una estrategia donde uno o varios de los mejores individuos de la generación anterior se conservan y se pasan directamente a la siguiente generación, asegurando que las mejores soluciones encontradas hasta el momento no se pierdan. La implementación del elitismo puede mejorar el rendimiento y la fiabilidad del AG al garantizar que el progreso realizado durante la optimización se mantenga.
- **Criterios de Terminación:** Es fundamental definir cuándo debe detenerse el proceso iterativo del Algoritmo Genético. Sin criterios de terminación adecuados, el algoritmo podría ejecutarse indefinidamente. Pymoo proporciona varios criterios de terminación comunes que se pueden utilizar:
 - *Número Máximo de Generaciones:* El algoritmo se detiene después de que se ha alcanzado un número predefinido de generaciones. Este es un criterio de terminación simple y comúnmente utilizado.
 - *Número Máximo de Evaluaciones de la Función Objetivo:* El algoritmo se detiene después de que la función objetivo ha sido evaluada un número máximo de veces. Este criterio es útil en problemas donde la evaluación de la función objetivo es computacionalmente costosa.
 - *Tolerancia en la Convergencia:* El algoritmo se detiene cuando la mejora en el valor de la función objetivo entre generaciones sucesivas es menor que un umbral predefinido durante un cierto número de generaciones. Esto indica que el algoritmo ha convergido a una solución (posiblemente local).
 - Pymoo puede ofrecer otros criterios de terminación, como un tiempo máximo de ejecución. Los criterios de terminación se suelen especificar al llamar al método `solve()` de la clase GA o mediante el uso de objetos específicos de

criterios de terminación del módulo `pymoo.termination`. La elección de los criterios de terminación adecuados es crucial para asegurar que el algoritmo se ejecute durante el tiempo suficiente para encontrar una buena solución sin desperdiciar recursos computacionales.

3.10.6. Implementación y Uso Práctico

A continuación, se presenta un ejemplo de código conciso que ilustra cómo utilizar la clase `pymoo.algorithms.soo.nonconvex.ga.GA` de Pymoo para resolver un problema simple de optimización mono-objetivo. Este ejemplo demostrará los pasos esenciales para definir el problema, configurar el algoritmo genético, ejecutar la optimización y analizar los resultados.

Listing 1 Ejemplo de código Pymoo para optimización mono-objetivo con AG

```

1  import numpy as np
2  from pymoo.core.problem import Problem
3  from pymoo.algorithms.soo.nonconvex.ga import GA
4  from pymoo.operators.sampling.rnd import RandomSampling
5  from pymoo.operators.crossover.sbx import SBX
6  from pymoo.operators.mutation.pm import PolynomialMutation
7  from pymoo.optimize import minimize
8
9  # 1. Definir el problema de optimización
10 class MyProblem(Problem):
11     def __init__(self):
12         super().__init__(n_var=2, n_obj=1, n_constr=0, xl=-5,
13                         xu=5)
14
15     def _evaluate(self, x, out, *args, **kwargs):
16         out["F"] = np.sum(x**2, axis=1)
17
18 # 2. Crear una instancia del problema
19 problem = MyProblem()
20
21 # 3. Crear una instancia del algoritmo genético
22 algorithm = GA(pop_size=20, sampling=RandomSampling(),
23                 crossover=SBX(prob=0.9, eta=15),
24                 mutation=PolynomialMutation(prob=0.1,
25                                             eta=20), eliminate_duplicates=True)
26
27 # 4. Ejecutar el algoritmo de optimización
28 res = minimize(problem, algorithm, ("n_gen", 100), verbose=False)
29
30 # 5. Analizar los resultados
31 print("Mejor solución encontrada:", res.X)
32 print("Valor de la función objetivo para la mejor solución:",
33       res.F)
```

En este ejemplo, primero se define un problema de optimización simple (`MyProblem`) donde el objetivo es minimizar la suma de los cuadrados de dos variables de decisión, cada una con límites entre -5 y 5. Luego, se crea una instancia de la clase `GA`, especificando un tamaño de población de 20 individuos, utilizando un muestreo aleatorio para la población inicial, el operador de cruce `SBX` [94] con una probabilidad de 0.9 y un factor de distribución de 15, el operador de mutación polinomial [95] con una probabilidad de 0.1 y un factor de distribución de 20, y habilitando la eliminación de duplicados. A continuación, se llama a la función `minimize` de Pymoo, que toma el problema, el algoritmo y un criterio de terminación (en este caso, un máximo de 100 generaciones) como argumentos. Finalmente, se imprimen la mejor solución encontrada (`res.X`) y el valor correspondiente de la función objetivo (`res.F`). Este ejemplo ilustra el flujo de trabajo básico para utilizar el AG canónico en Pymoo [92].

Configuración y Ajuste de Parámetros

La elección adecuada de los valores para los parámetros del Algoritmo Genético, como el tamaño de la población, la probabilidad de cruce y la probabilidad de mutación, es de suma importancia para lograr un buen rendimiento del algoritmo [90]. Estos parámetros influyen directamente en la capacidad del algoritmo para explorar el espacio de búsqueda, explotar las buenas soluciones encontradas y converger hacia un óptimo. No existe un conjunto de parámetros universalmente óptimo que funcione bien para todos los problemas; los mejores valores a menudo dependen de las características específicas del problema que se está resolviendo.

El tamaño de la población (`pop_size`) determina el número de soluciones candidatas que se mantienen y evolucionan en cada generación. Un tamaño de población mayor permite una exploración más exhaustiva del espacio de búsqueda, lo que puede aumentar la probabilidad de encontrar el óptimo global, especialmente en problemas complejos con muchos óptimos locales. Sin embargo, una población más grande también implica un mayor costo computacional por generación. Por otro lado, un tamaño de población más pequeño puede resultar en una convergencia más rápida, pero corre el riesgo de estancarse en óptimos locales debido a una exploración limitada. Los rangos típicos para el tamaño de la población pueden variar ampliamente, desde unas pocas decenas hasta varios cientos, dependiendo de la complejidad del problema.

La probabilidad de cruce (a menudo un parámetro del operador de cruce, como `prob` en el ejemplo del operador `SBX`) controla la frecuencia con la que se aplica el operador de cruce a los padres seleccionados. Una probabilidad de cruce alta favorece la exploración al generar nuevas soluciones mediante la combinación de la información genética de los padres. Una probabilidad de cruce baja, por otro lado, permite que las soluciones se transmitan a la siguiente generación con menos cambios, lo que puede ser beneficioso en las etapas posteriores de la búsqueda cuando se están afinando las soluciones cercanas al óptimo. Los valores típicos para la probabilidad de cruce suelen estar en el rango de 0.8 a 0.95 [90].

La probabilidad o tasa de mutación (a menudo un parámetro del operador de mutación, como `prob` en el ejemplo del operador de mutación polinomial) determina la probabilidad de que cada gen (o variable) en un individuo descendiente

se modifique aleatoriamente. La mutación juega un papel crucial en el mantenimiento de la diversidad genética y en la prevención de la convergencia prematura al introducir nuevas características en la población que no estaban presentes en los padres. Una tasa de mutación demasiado baja puede resultar en una falta de exploración de nuevas áreas del espacio de búsqueda, mientras que una tasa demasiado alta puede perturbar las buenas soluciones y hacer que la búsqueda sea esencialmente aleatoria. Los valores típicos para la tasa de mutación suelen ser bajos, a menudo en el rango de 0.01 a 0.1, dependiendo del problema y la representación [90].

Dado que no existen reglas fijas para determinar los valores óptimos de estos parámetros, a menudo es necesario recurrir a la experimentación para encontrar una configuración que funcione bien para un problema específico. Esto puede implicar ejecutar el algoritmo con diferentes combinaciones de parámetros y comparar su rendimiento en términos de la calidad de la solución encontrada y la velocidad de convergencia. Algunas técnicas más avanzadas para el ajuste de parámetros incluyen la búsqueda en cuadrícula (grid search), la búsqueda aleatoria (random search) y métodos de control de parámetros adaptativos, donde los parámetros del algoritmo se ajustan dinámicamente durante el proceso de optimización en función de su rendimiento.

La siguiente tabla (Tabla 3.5) resume los parámetros clave de la clase `pymoo.algorithms.soo.nonconvex.ga.GA` y su impacto en el rendimiento del algoritmo:

3.10.7. Consideraciones Adicionales

La modularidad de Pymoo facilita la experimentación con diferentes componentes del AG, como los operadores de selección, cruce y mutación, lo que permite a los investigadores y profesionales explorar diversas estrategias evolutivas para encontrar soluciones óptimas. Si bien esta sección del marco teórico se ha centrado en la implementación canónica del AG, es importante destacar que Pymoo ofrece una amplia gama de otros algoritmos de optimización mono-objetivo, como la Evolución Diferencial y la Optimización por Enjambre de Partículas (PSO), que podrían ser explorados como alternativas o complementos al AG. Además, para problemas que involucran múltiples objetivos, Pymoo es especialmente potente gracias a su enfoque principal en la optimización multiobjetivo [92, 93].

La extensibilidad de Pymoo permite a los usuarios avanzados implementar sus propios operadores genéticos o modificar el comportamiento del algoritmo `GA` existente para satisfacer necesidades específicas. Para aplicaciones del mundo real, es crucial considerar las posibles restricciones que pueden existir en el problema de optimización. Pymoo proporciona mecanismos para manejar restricciones en problemas mono-objetivo utilizando AGs, lo que amplía su aplicabilidad a una gama más amplia de escenarios. Finalmente, es fundamental evaluar el rendimiento del AG utilizando métricas apropiadas, como la velocidad de convergencia, la calidad de la solución obtenida y la robustez del algoritmo frente a diferentes ejecuciones.

Se anima a los usuarios a experimentar con la clase `GA` en Pymoo y a explorar la extensa documentación de la biblioteca [92] para descubrir características más avanzadas y aplicaciones en diversos dominios de la optimización. La comprensión

Tabla 3.5: Parámetros Clave de la Clase `pymoo.algorithms.soo.nonconvex.ga.GA`

Parámetro	Descripción	Rango/Valores Típicos	Impacto en el Rendimiento
<code>pop_size</code>	Número de individuos en la población.	50500 (según la complejidad)	Valores más altos aumentan la exploración pero también el costo computacional. Valores más bajos pueden llevar a convergencia prematura.
<code>sampling</code>	Método para crear la población inicial.	<code>RandomSampling</code> , <code>LHS</code>	Influye en la diversidad inicial de la población y la cobertura del espacio de búsqueda.
<code>crossover</code>	Operador de cruce para generar descendencia (ej., SBX [94], PMX).	Depende del problema	Determina cómo se combina la información genética. Diferentes operadores tienen diferentes características de búsqueda.
<code>mutation</code>	Operador de mutación para mantener diversidad (ej., <code>PolynomialMutation</code> [95]).	Depende de la codificación	Introduce cambios aleatorios para mantener la diversidad genética y evitar óptimos locales.
<code>eliminate_duplicates</code>	Indicador para eliminar individuos duplicados.	True, False	Ayuda a prevenir convergencia prematura manteniendo diversidad, pero añade costo computacional.

de los fundamentos del AG canónico proporcionados en este informe sirve como una base sólida para explorar las capacidades más amplias de Pymoo y para abordar una variedad de problemas de optimización complejos.

3.11. Framework de interfaz: Qt & PyQt

3.11.1. Introducción a Qt

El framework Qt se define formalmente como un entorno de desarrollo de aplicaciones multiplataforma, diseñado para la creación de interfaces gráficas de usuario (GUI) y aplicaciones que pueden ejecutarse en diversos sistemas operativos y plataformas de hardware, incluyendo Linux, Windows, macOS, Android y sistemas embebidos, con mínimas o nulas modificaciones en el código subyacente [96]. Constituye un conjunto integral de clases de biblioteca C++ altamente intuitivas y modularizadas, enriquecidas con APIs que simplifican el proceso de desarrollo de aplicaciones [97]. Esta capacidad permite a los programadores escribir código una sola vez y distribuirlo a través de múltiples plataformas, abarcando desde aplicaciones de escritorio hasta sistemas embebidos [98]. La cualidad de ser multiplataforma, a menudo resumida en el principio de “escribir una vez, ejecutar en cualquier lugar” (WORA, por sus siglas en inglés), representa una ventaja fundamental de Qt, ya que reduce significativamente los tiempos y costos de desarrollo al mismo tiempo que permite alcanzar una audiencia más amplia [99]. Además, el hecho de que C++ sea el lenguaje subyacente [96] sugiere un enfoque en el rendimiento y el acceso a funcionalidades de bajo nivel del sistema.

La historia de Qt se remonta a 1990, cuando los programadores noruegos Eirik Chambe-Eng y Haavard Nord concibieron la idea, con el primer lanzamiento público en 1995 para el sistema operativo Linux [100]. Inicialmente fue desarrollado por la empresa Trolltech, que posteriormente fue adquirida por Nokia, luego por Digia y actualmente es conocida como The Qt Company [100]. Su popularidad experimentó un crecimiento significativo tras ser utilizado en la creación del entorno de escritorio KDE [97]. A lo largo de los años, Qt ha evolucionado desde una simple biblioteca de clases hasta convertirse en un framework extenso, experimentando cambios en sus modelos de licenciamiento [97]. Esta trayectoria de larga data y evolución sugiere un framework maduro y confiable, refinado a través de un uso extensivo y contribuciones continuas. El enfoque inicial en el soporte multiplataforma para Unix, Macintosh y Windows [100] subraya el principio fundacional de la independencia de la plataforma.

La arquitectura de Qt se basa en el lenguaje de programación C++, el cual es extendido por un preprocesador denominado Meta-Object Compiler (moc) [96]. El moc interpreta ciertas macros presentes en el código C++ como anotaciones y genera código C++ adicional que contiene meta-information sobre las clases utilizadas [96]. Esta meta-information habilita características como *signals* y *slots*, introspección y llamadas a funciones asíncronas, las cuales no están disponibles de forma nativa en C++ [96]. El moc se erige como un componente arquitectónico clave que distingue a Qt al facilitar sus funcionalidades únicas para la comunicación entre objetos y el comportamiento dinámico. El hecho de que Qt extienda C++

en lugar de ser un lenguaje independiente [96] permite a los desarrolladores aprovechar los beneficios de rendimiento de C++ al mismo tiempo que utilizan las abstracciones proporcionadas por Qt.

3.11.2. Principios Fundamentales de Qt

Qt presenta una arquitectura modular, compuesta por módulos esenciales y módulos complementarios [96]. Los *Qt Essentials* constituyen la base de Qt en todas las plataformas compatibles e incluyen módulos como **Qt Core** (funcionalidades no gráficas), **Qt GUI** (interfaz gráfica), **Qt Multimedia**, **Qt Network**, **Qt Quick** (interfaz de usuario declarativa) y **Qt SQL** (integración de bases de datos) [97]. Por otro lado, los *Qt Add-Ons* ofrecen módulos para propósitos específicos que podrían no estar disponibles en todas las plataformas, como **Qt OpenGL**, **Qt Wayland Compositor**, **Qt Sensors**, **Qt WebView**, **Qt Safe Renderer** y **Qt SCXML** [97]. Esta modularidad permite a los desarrolladores incluir únicamente los componentes necesarios, lo que potencialmente reduce el tamaño de la aplicación [97]. La separación en “Essentials” y “Add-Ons” sugiere un diseño que atiende tanto a las necesidades de desarrollo comunes como a las especializadas.

Signals y Slots

Un principio fundamental de Qt es el mecanismo de *signals* y *slots*, una construcción del lenguaje introducida para la comunicación entre objetos. Este mecanismo facilita la implementación del patrón observador, evitando la necesidad de escribir código repetitivo. Un *signal* es emitida por un objeto cuando su estado interno cambia de una manera que podría ser de interés para otros objetos [101]. Un *slot* es una función que se llama en respuesta a una *signal* particular [101]. Las firmas de una *signal* deben coincidir con la firma del *slot* receptor, garantizando la seguridad de tipos [101]. Un *slot* puede tener una firma más corta al ignorar argumentos adicionales [101]. Los *signals* y *slots* están débilmente acoplados; el objeto emisor no necesita conocer qué *slots* recibirán la *signal* [101]. Las conexiones entre *signals* y *slots* se establecen mediante la función `connect ()` [101]. Se pueden conectar múltiples *signals* a un solo *slot*, y una sola *signal* se puede conectar a múltiples *slots* [101]. Este mecanismo de *signals* y *slots* representa una innovación central de Qt que simplifica el manejo de eventos y la comunicación entre componentes de una manera segura en cuanto a tipos y desacoplada. Resulta particularmente adecuado para la programación de GUIs, donde las interacciones del usuario desencadenan comportamientos específicos [98].

Widgets y Layouts

Para la construcción de interfaces gráficas, Qt introduce los conceptos de *widgets* y *layouts*. Los programas Qt se crean utilizando partes denominadas *widgets*, que son los bloques de construcción básicos de cualquier interfaz de usuario, abarcando desde entradas de texto y etiquetas hasta ventanas y botones [98]. Qt utiliza *layouts* para posicionar estos *widgets* en la pantalla, controlando automáticamente su tamaño y ubicación para adaptarse a diferentes tamaños y resoluciones de pantalla [98]. Los administradores de *layouts* comunes incluyen **QVBoxLayout**, **QHBoxLayout** y **QGridLayout** [102]. El sistema de *widgets* y *layouts* proporciona una manera estructurada y consistente en todas las plataformas para diseñar

interfaces de usuario. Los *layouts* son cruciales para crear aplicaciones responsivas que se adaptan a diversas dimensiones de pantalla. Qt también ofrece **Qt Quick**, un framework declarativo para construir aplicaciones altamente dinámicas con interfaces de usuario personalizadas [96]. Las GUIs que utilizan **Qt Quick** se escriben en QML (Qt Meta Language o Qt Modeling Language), un lenguaje de descripción de objetos declarativo que integra JavaScript para la programación procedural [96]. **Qt Quick** facilita el desarrollo rápido de aplicaciones, especialmente para dispositivos móviles, con la posibilidad de escribir lógica en código nativo (C++) para las partes que requieren un rendimiento crítico [96]. **Qt Quick** y QML ofrecen un enfoque más moderno y a menudo más eficiente en términos de rendimiento para el desarrollo de interfaces de usuario, particularmente para interfaces táctiles y animaciones [103]. La separación del diseño de la interfaz de usuario (QML) y la lógica de backend (C++ o Python con PyQt) promueve una mejor organización del código y colaboración entre diseñadores y desarrolladores [97].

3.11.3. Variantes y Licencias de Qt

Qt está disponible bajo licencias tanto comerciales como de código abierto. Las licencias de código abierto incluyen diversas versiones de la GPL (GNU General Public License) y la LGPL (GNU Lesser General Public License) [96]. Las licencias comerciales son ofrecidas por The Qt Company y proporcionan características adicionales, soporte y opciones de licenciamiento adecuadas para el desarrollo de software propietario [96]. Este modelo de doble licencia permite que Qt se utilice en una amplia gama de proyectos, desde código abierto hasta aplicaciones comerciales. Comprender los términos de la licencia es crucial para los desarrolladores para garantizar el cumplimiento de los requisitos de su proyecto [96]. Las versiones principales de Qt incluyen Qt 5 (lanzado en 2012) y Qt 6 (lanzado en 2020) [97]. Qt 5 introdujo cambios significativos como una nueva base de código modularizada, la consolidación de QPA (Qt Platform Abstraction) y Qt Quick 2 [100]. Qt 6 representa una evolución adicional con actualizaciones y mejoras continuas [97]. Las versiones de soporte a largo plazo (LTS) están disponibles para usuarios comerciales, proporcionando estabilidad y períodos de soporte extendidos [96]. La existencia de versiones principales indica un desarrollo y evolución continuos del framework, con cada versión introduciendo nuevas funcionalidades y mejoras. La disponibilidad de versiones LTS es importante para proyectos que requieren estabilidad y soporte a largo plazo, como aplicaciones industriales o críticas para la seguridad [99].

3.11.4. Aplicaciones Típicas de Qt

Qt se utiliza en el desarrollo de aplicaciones que se ejecutan en las principales plataformas de escritorio (Windows, macOS, Linux), plataformas móviles (Android, iOS) y plataformas embebidas [96]. Es una opción popular para crear aplicaciones multiplataforma con interfaces de aspecto nativo [96]. Qt también se utiliza ampliamente en el mercado de sistemas embebidos debido a su soporte para diversos sistemas operativos embebidos como Linux y QNX [99]. La amplia compatibilidad con plataformas convierte a Qt en una opción versátil para diversos dominios de aplicación. Su idoneidad para sistemas embebidos resalta su

eficiencia y adaptabilidad a entornos con recursos limitados [97]. Qt se emplea en diversas industrias, incluyendo la automotriz (tableros digitales, sistemas de infoentretenimiento), la salud (instrumentos médicos), la aeroespacial y defensa, la automatización industrial y la electrónica de consumo [99]. Ejemplos de aplicaciones conocidas construidas con Qt incluyen Google Earth, Skype, VirtualBox, Autodesk Maya, Telegram y partes de la interfaz KDE [104]. En el ámbito de la computación científica, PyQt (el binding de Python para Qt) se utiliza para construir herramientas de análisis y visualización de datos, así como interfaces gráficas de usuario para equipos de laboratorio y sistemas de adquisición de datos [105]. Qt Quick 3D Physics proporciona una API de alto nivel para la simulación física, soportando cuerpos rígidos y mallas estáticas [106]. Herramientas como QTCAD® están construidas con Qt para simular hardware de tecnología cuántica [105]. La amplia gama de aplicaciones demuestra la versatilidad y potencia del framework Qt en diferentes sectores, incluyendo aquellos relevantes para la investigación científica y la simulación. La mención específica de Qt en la simulación física y la visualización científica subraya su potencial utilidad en el contexto del proyecto principal.

3.11.5. Ventajas y Desventajas de Qt

- **Ventajas:** Entre las fortalezas inherentes a Qt se destaca su compatibilidad multiplataforma, que permite a los desarrolladores escribir código una vez y desplegarlo en múltiples sistemas operativos, ahorrando tiempo y recursos [96]. Su rico conjunto de herramientas y bibliotecas ofrece una amplia gama de funcionalidades, desde elementos de interfaz de usuario hasta redes, multimedia y gráficos 3D [97]. Al estar basado en C++, Qt proporciona un alto rendimiento nativo y un tamaño reducido [97]. Además, cuenta con una gran y activa comunidad que ofrece tutoriales, foros y bibliotecas para soporte y aprendizaje [98]. Su documentación exhaustiva incluye tutoriales detallados, ejemplos y referencias de la API [99]. Estas ventajas convierten a Qt en un framework potente y eficiente para desarrollar aplicaciones complejas en diversas plataformas.
- **Desventajas:** Sin embargo, Qt también presenta ciertas limitaciones. Su curva de aprendizaje puede ser pronunciada para principiantes, especialmente para aquellos que no están familiarizados con C++ o la programación de interfaces gráficas, particularmente en la comprensión de conceptos como *signals* y *slots* [107]. Los costos de las licencias comerciales pueden ser elevados, lo que podría ser una preocupación para startups o proyectos más pequeños [96]. No todas las partes de Qt están bajo la licencia LGPL [99]. Para aplicaciones pequeñas o simples, Qt podría percibirse como demasiado pesado en comparación con frameworks más sencillos [107]. La capa de abstracción en Qt podría introducir cierta sobrecarga de rendimiento, lo que podría ser una preocupación para sistemas embebidos con recursos muy limitados [99]. Si bien Qt ofrece ventajas significativas, los desarrolladores deben ser conscientes de su curva de aprendizaje y las implicaciones de la licencia, especialmente para proyectos comerciales. La percepción de que es demasiado pesado para tareas simples podría llevar a algunos a elegir frameworks alternativos.

3.11.6. Introducción a PyQt

PyQt se define como un conjunto de bindings de Python para el framework de aplicaciones Qt de The Qt Company [[pyqt·wiki](#)]. Estos bindings se implementan como un conjunto de módulos de Python y contienen un gran número de clases, superando las 1000 [[pyqt·wiki](#)]. PyQt actúa como un puente, integrando de manera fluida el robusto framework multiplataforma Qt C++ con el lenguaje de programación Python, conocido por su flexibilidad [103]. Esto permite a los desarrolladores de Python aprovechar la potencia y las características del framework Qt utilizando la sintaxis de Python. El extenso número de clases disponibles en PyQt indica un acceso integral a las funcionalidades de Qt. PyQt fue lanzado por primera vez en 1998 por la firma británica Riverbank Computing [[pyqt·wiki](#)]. Fue desarrollado por Phil Thompson utilizando la herramienta SIP para generar automáticamente los bindings [[pyqt·wiki](#)]. Posteriormente, Nokia (entonces propietaria de Qt) lanzó sus propios bindings de Python denominados PySide (ahora Qt for Python) debido a desacuerdos en las licencias [[pyqt·wiki](#)]. El desarrollo temprano de PyQt resalta el deseo de llevar las capacidades de Qt al ecosistema de Python. La existencia de PySide como un binding alternativo proporciona a los desarrolladores más opciones de licencia [102]. Qt for Python es el conjunto oficial de bindings de Python (PySide6) respaldado por The Qt Company [103]. PyQt y PySide ofrecen funcionalidades similares ya que ambos se construyen sobre Qt [102]. Una diferencia clave radica a menudo en la licencia, siendo PySide típicamente disponible bajo la LGPL, que es más permisiva para proyectos comerciales en comparación con las licencias GPL y comerciales de PyQt [102]. Si bien PyQt fue el primer binding de Python para Qt ampliamente adoptado, Qt for Python (PySide) es ahora la oferta oficial de los mantenedores de Qt. La elección entre PyQt y PySide a menudo depende de las necesidades específicas de licencia del proyecto.

3.11.7. Principios Fundamentales de PyQt

PyQt integra la funcionalidad de Qt en el entorno de programación Python proporcionando acceso a la funcionalidad de Qt a través de módulos de extensión de Python [102]. Los desarrolladores pueden importar estos módulos (por ejemplo, `QtCore`, `QtGui`, `QtWidgets`) para utilizar las clases y funciones de Qt dentro de su código Python [102]. PyQt conserva gran parte de la sintaxis de Qt, lo que facilita a los desarrolladores familiarizados con Qt/C++ la transición a Python [103]. PyQt actúa como un envoltorio alrededor de la biblioteca Qt C++, exponiendo sus características a Python. La estrecha semejanza de sintaxis entre PyQt y Qt/C++ puede facilitar el intercambio y la comprensión del código entre lenguajes. PyQt refleja la estructura modular de Qt, proporcionando módulos de Python que corresponden a los módulos C++ de Qt (por ejemplo, `QtCore` para funcionalidades centrales no gráficas, `QtGui` para componentes de GUI, `QtNetwork` para redes, `QtSql` para acceso a bases de datos) [102]. Cada módulo contiene clases de Python que proporcionan acceso a las clases C++ de Qt correspondientes y sus funcionalidades [102]. Esta modularidad permite a los desarrolladores de Python importar y utilizar selectivamente solo las partes necesarias del framework Qt. PyQt implementa el mecanismo de *signals* y *slots* de Qt en Python, permitiendo que los objetos de Python se comuniquen de una manera segura en cuanto a tipos y desacoplada [98]. Los métodos de Python se pueden definir como *slots* y

conectarse a *signals* de Qt, habilitando la programación orientada a eventos [98]. Las *signals* son emitidas por los *widgets* de Qt y se pueden conectar a *slots* de Python para desencadenar acciones en respuesta a interacciones del usuario u otros eventos [98]. La integración fluida de *signals* y *slots* en Python permite construir aplicaciones GUI interactivas y responsivas. PyQt se integra con Qt Designer, una herramienta de construcción de GUI visual proporcionada por Qt [98]. Los desarrolladores pueden diseñar interfaces de usuario utilizando una interfaz de arrastrar y soltar en Qt Designer y luego utilizar PyQt para cargar estos diseños en sus aplicaciones Python [99]. PyQt puede generar código Python a partir de archivos .ui de Qt Designer utilizando herramientas como pyuic [102]. La integración con Qt Designer facilita la creación rápida de prototipos y el desarrollo visual de GUIs en Python.

3.11.8. Aplicaciones Típicas de PyQt

PyQt se utiliza principalmente para crear programas con interfaz gráfica de usuario en Python, siendo reconocido por su facilidad de uso, flexibilidad y la apariencia nativa de sus aplicaciones GUI [102]. Es una opción popular para construir aplicaciones de escritorio multiplataforma, aplicaciones móviles y sistemas embebidos [105]. Su versatilidad y naturaleza multiplataforma lo convierten en una herramienta valiosa para el desarrollo de interfaces gráficas en Python. En el ámbito de la computación científica, PyQt se utiliza para construir herramientas de análisis y visualización de datos, como interfaces gráficas para equipos de laboratorio y sistemas de adquisición de datos [105]. También se aplica en la creación de aplicaciones para el análisis y visualización de imágenes médicas, como visores de resonancias magnéticas y tomografías computarizadas [105]. Además, se utiliza en herramientas para la gestión de proyectos, la visión por computador y el análisis de datos [105]. Su integración con bibliotecas como PyQtGraph y Matplotlib permite realizar gráficos y visualizaciones avanzadas [103]. Dada su aplicación en la computación científica y la visualización de datos, PyQt podría ser una herramienta valiosa para la creación de interfaces de usuario para simulaciones físicas, permitiendo el control interactivo y la visualización de los parámetros y resultados de la simulación. El framework Qt subyacente también cuenta con módulos como Qt Quick 3D Physics, lo que sugiere un potencial para capacidades de simulación física más integradas, aunque PyQt principalmente proporciona la capa de interfaz en Python.

3.11.9. Ventajas y Desventajas de PyQt

- **Ventajas:** Entre las ventajas de utilizar PyQt se encuentra la facilidad de uso y flexibilidad que ofrece la combinación de la simplicidad de Python con las potentes características de Qt, lo que lo hace relativamente fácil de aprender y utilizar para el desarrollo de interfaces gráficas [105]. Su compatibilidad multiplataforma permite que las aplicaciones PyQt se ejecuten en Windows, macOS, Linux y otras plataformas sin modificaciones significativas en el código [99]. Además, PyQt proporciona extensas bibliotecas y herramientas para la construcción de aplicaciones GUI complejas, incluyendo *widgets*, gráficos, soporte de red, soporte de bases de datos y multimedia [102]. La integración con Qt Designer permite un desarrollo rápido de interfaces de

usuario a través de un diseño visual [99]. Finalmente, cuenta con una gran y activa comunidad que proporciona recursos, soporte y facilita la resolución de problemas [99].

- **Desventajas:** Sin embargo, también presenta desventajas. Su licencia es dual, bajo GPL v3 y una licencia comercial. Para aplicaciones comerciales donde no se desea una licencia compatible con GPL, se debe adquirir una licencia comercial [102]. A diferencia de Qt, PyQt no está disponible bajo la LGPL [[pyqt.wiki](#)]. Al ser un wrapper alrededor de C++, las aplicaciones PyQt podrían ser susceptibles a errores críticos que pueden provocar el cierre inesperado del intérprete de Python [108]. Su curva de aprendizaje puede ser más pronunciada en comparación con frameworks GUI puramente basados en Python como Tkinter [107]. La documentación podría estar más centrada en el framework Qt en C++, lo que requeriría cierta interpretación para los desarrolladores de Python [108]. En algunos casos, los desarrolladores podrían necesitar gestionar explícitamente las actualizaciones de la interfaz de usuario utilizando *signals* y *slots*, lo que puede volverse complejo en aplicaciones grandes [108].

3.12. Matplotlib

3.12.1. Introducción: El Papel de Matplotlib en la Visualización de Datos Científicos

En la investigación científica, especialmente en disciplinas como la Física Computacional, Astrofísica y el análisis de algoritmos complejos, la visualización de datos es un componente fundamental [109]. La capacidad de transformar datos numéricos abstractos en representaciones visuales claras facilita la identificación de patrones, la interpretación de resultados y la comunicación efectiva de hallazgos [110]. En este contexto, Matplotlib, una biblioteca de Python ampliamente adoptada, se ha consolidado como una piedra angular para la visualización dentro del ecosistema científico de Python [111]. Su diseño busca emular la usabilidad de MATLAB, aprovechando la potencia y flexibilidad de Python, siendo además de código abierto y gratuito [112]. Su integración con bibliotecas esenciales como NumPy, utilizada para la manipulación de arrays, subraya su idoneidad para aplicaciones numéricas y científicas [109, 113]. La prevalencia de Matplotlib en la comunidad científica sugiere un amplio respaldo y una continua evolución para satisfacer las necesidades de los investigadores [111].

3.12.2. Definición, Contexto Histórico y Orígenes

Matplotlib se define como una biblioteca fundamental para la generación de gráficos 2D (y con capacidades básicas 3D) en Python, utilizada para el desarrollo de aplicaciones, la creación de scripts interactivos y la generación de imágenes con calidad de publicación [110, 112]. Su propósito primordial es ofrecer un entorno flexible para crear visualizaciones estáticas, animadas e interactivas [112].

El origen de Matplotlib se remonta al trabajo de John D. Hunter en 2003 [111, 114]. Hunter, un neurobiólogo, la desarrolló inicialmente por la necesidad de visualizar

datos de electrocorticografía (ECOG) de pacientes con epilepsia, buscando una alternativa de código abierto al software propietario existente [115]. En un entorno académico con acceso limitado a licencias costosas, se hizo evidente la necesidad de una herramienta gratuita y extensible [115]. Inspirado en MATLAB, Hunter buscó replicar sus capacidades de trazado en Python, facilitando la transición para científicos ya familiarizados con ese entorno [112, 114]. Sin embargo, limitaciones en la gestión de grandes conjuntos de datos biomédicos en MATLAB impulsaron el desarrollo en Python, con el objetivo de lograr gráficos con calidad de publicación y un entorno más flexible [114]. La introducción formal a la comunidad científica se produjo con el artículo “Matplotlib: A 2D Graphics Environment” en *Computing in Science & Engineering* en 2007 [110], un trabajo seminal para la biblioteca. Su desarrollo recibió apoyo temprano de instituciones como el Space Telescope Science Institute (STScI) [116], destacando su valor en proyectos científicos de alto impacto.

3.12.3. Arquitectura y Conceptos Fundamentales

La arquitectura de Matplotlib se basa conceptualmente en tres capas distintas, fomentando la modularidad y permitiendo su uso en diversos entornos [115, 117]:

1. **Capa Backend:** Se encarga de la interfaz con los dispositivos de salida (renderizadores) y los kits de herramientas de interfaz de usuario (GUI). Es la capa que dibuja en la pantalla o guarda en un archivo (ej. PNG, PDF, SVG) [115]. Incluye backends interactivos (para Tkinter, wxPython, Qt, GTK) y no interactivos (para generar archivos de imagen como Agg, PDF, SVG) [118]. Esta separación permite la portabilidad entre diferentes sistemas operativos y entornos [115].
2. **Capa Artist (Núcleo/API):** Es el corazón de la biblioteca, donde reside la lógica de trazado. Contiene un conjunto de clases (los “Artistas”) que representan y gestionan todos los elementos de un gráfico (figuras, ejes, líneas, texto, etc.) [112, 118]. Proporciona un control granular sobre cada elemento visual [112]. Los componentes clave en esta capa son:
 - **Figure:** El contenedor de nivel superior para todos los elementos del gráfico; representa la ventana o página completa [112, 118].
 - **Axes:** El área de trazado dentro de la Figure donde se dibujan los datos; incluye los ejes (x, y, y opcionalmente z), etiquetas, títulos y leyendas [112, 118]. Es importante distinguir **Axes** (el área) de **Axis** (los objetos que representan los ejes de coordenadas) [118].
 - **Artist:** Cualquier objeto visible en la Figure es un Artist (incluyendo Figure, Axes, líneas, texto, colecciones, parches, etc.) [112, 118]. Comprender esta jerarquía es esencial para la personalización avanzada [112].
3. **Capa Scripting (pyplot):** Proporciona una interfaz de alto nivel, principalmente a través del módulo `matplotlib.pyplot`. Ofrece una colección de funciones que emulan los comandos de MATLAB, permitiendo crear gráficos rápidamente con comandos simples [112, 119]. Mantiene un estado interno

(la figura y ejes actuales), lo que facilita la creación rápida de gráficos pero ofrece menos control explícito que la API orientada a objetos [119].

3.12.4. Estilos de API: Pyplot vs. Orientada a Objetos (OO)

Matplotlib ofrece dos interfaces principales para crear gráficos [112]:

Tabla 3.6: Estilos de API: Pyplot vs. Orientada a Objetos (OO)

Característica	Pyplot API	Object-Oriented (OO) API
Descripción	Interfaz con estado, similar a MATLAB	Creación y manipulación explícita de objetos Figure/Axes
Facilidad de Uso	Generalmente más fácil para gráficos simples	Curva de aprendizaje más pronunciada para principiantes
Control	Control limitado sobre elementos individuales	Control total sobre todos los aspectos del gráfico
Estructura Código	Estilo procedural, a menudo más corto para básicos	Más detallado, mejor para gráficos complejos/reutilizables

La interfaz pyplot es conveniente para exploración rápida y scripts sencillos, mientras que la API OO es preferida para funciones complejas, desarrollo de aplicaciones y cuando se necesita un control total sobre la figura [112, 119]. La coexistencia de ambas puede ser confusa para principiantes [120].

3.12.5. Características Clave y Capacidades de Trazado

Matplotlib admite una amplia gama de tipos de gráficos 2D, incluyendo gráficos de líneas, dispersión, barras, histogramas, circulares, diagramas de caja y mapas de calor [109]. También ofrece capacidades básicas de trazado 3D a través del módulo `mpl_toolkits.mplot3d` [109]. Proporciona extensas opciones de personalización para controlar casi cada aspecto de un gráfico: colores, marcadores, estilos de línea, fuentes, etiquetas, anotaciones, etc. [121, 122]. Admite diversos formatos de salida, incluyendo formatos rasterizados (PNG) y vectoriales (PDF, SVG, EPS), siendo estos últimos cruciales para mantener la calidad en publicaciones académicas [115, 123]. Matplotlib también ofrece capacidades de animación para visualizar datos cambiantes en el tiempo o en respuesta a parámetros [124].

3.12.6. Aplicaciones Típicas en Computación Científica y Campos Relacionados

Matplotlib es ampliamente utilizada en física, astronomía, ingeniería, biología y ciencia de datos para visualizar datos y crear gráficos de calidad [109]. Su integración con NumPy, SciPy y pandas facilita flujos de trabajo de análisis y visualización [109, 125]. Es la base para bibliotecas de visualización de más alto nivel como Seaborn [126].

Ejemplos notables incluyen su uso por la colaboración del Telescopio de Horizonte de Eventos (EHT) para visualizar los datos que llevaron a la primera imagen de

Tabla 3.7: Tipos de Gráficos Comunes en Matplotlib y Aplicaciones Científicas

Tipo de Gráfico	Descripción	Aplicaciones (Computación Científica)	Típicas Científicas
Gráfico de Líneas	Muestra datos como serie de puntos conectados por líneas rectas.	Series temporales, trayectorias de partículas, espectros.	
Gráfico de Dispersion	Muestra puntos de datos individuales como marcadores.	Espacio de fases, condiciones iniciales, correlación entre variables.	
Gráfico de Barras	Usa barras rectangulares para representar categorías de datos.	Comparación de métricas de rendimiento, distribuciones categóricas.	
Histograma	Muestra la distribución de frecuencia de un conjunto de datos.	Distribución de energías, errores, tamaños de partículas.	
Gráfico de Contorno	Muestra líneas de valor constante de una superficie 3D en un plano 2D.	Paisajes de energía potencial, campos escalares (temperatura, presión).	
Mapa de Calor (Heatmap)	Muestra datos matriciales donde los valores se representan por color.	Matrices de correlación, patrones espaciales, expresión génica.	

un agujero negro [127] y por la NASA para visualizar datos de la misión Phoenix en Marte [118]. En Física Computacional, se usa para visualizar simulaciones de sistemas dinámicos (ej. péndulo accionado), densidades de probabilidad cuántica (ej. átomo de hidrógeno) [128], análisis de radiación de cuerpo negro y diagramas de Hertzsprung-Russell en Astrofísica [129]. También se utiliza en simulaciones de N-cuerpos para visualizar trayectorias e interacciones [130]. En educación, se integra frecuentemente con Jupyter Notebooks para enseñar programación y visualización de datos de forma interactiva [109].

3.12.7. Ventajas de Matplotlib

- **Versatilidad y Flexibilidad:** Admite una gran variedad de gráficos 2D y ofrece control detallado sobre casi todos los elementos [121, 122].
- **Calidad de Publicación:** Capaz de producir figuras de alta calidad en diversos formatos, incluyendo vectoriales (PDF, SVG), adecuados para artículos académicos [110, 123]. Existen herramientas como SciencePlots para simplificar el formateo para revistas específicas [131].
- **Compatibilidad Multiplataforma:** Funciona en diversos sistemas operativos y admite múltiples backends gráficos [115].
- **Integración con el Ecosistema Científico:** Se integra estrechamente con NumPy, SciPy, pandas y otras bibliotecas científicas [109, 125].
- **Comunidad Grande y Documentación Extensa:** Cuenta con una comunidad activa y una documentación detallada con tutoriales y ejemplos [111, 132].

- **Código Abierto y Gratuito:** Accesible para todos sin costos de licencia, fomentando su adopción y colaboración [112].

3.12.8. Limitaciones de Matplotlib

- **Verbosidad para Gráficos Complejos:** Crear gráficos altamente personalizados puede requerir una sintaxis más detallada en comparación con bibliotecas de nivel superior [122].
- **Estilos Predeterminados:** Los estilos por defecto a menudo necesitan personalización significativa para lograr una apariencia profesional o de publicación [116, 127].
- **API Menos Intuitiva (Percibida):** La API, especialmente la diferencia entre `pyplot` y la OO, puede tener una curva de aprendizaje pronunciada y ser percibida como menos intuitiva que otras bibliotecas [120, 133].
- **Rendimiento con Datos Muy Grandes:** Puede enfrentar limitaciones de rendimiento con conjuntos de datos extremadamente grandes (millones de puntos), pudiendo requerir optimización o bibliotecas alternativas [134].
- **Interactividad Limitada:** Aunque soporta cierta interactividad, es menos avanzada en comparación con bibliotecas diseñadas específicamente para visualizaciones web interactivas (ej., Plotly, Bokeh) [121].

3.12.9. Personalización para Figuras con Calidad de Publicación

Lograr figuras de calidad de publicación requiere atención al detalle. Matplotlib ofrece un control exhaustivo sobre la estética [122]:

- **Ajustes de Elementos:** Permite ajustar fuentes (tamaño, tipo), estilos de línea, colores y marcadores para asegurar legibilidad y distinción [123].
- **Etiquetas y Leyendas:** La adición de etiquetas de ejes claras, títulos descriptivos y leyendas informativas es fundamental [135].
- **Métodos de Configuración:** Se pueden modificar parámetros globalmente o localmente mediante `rcParams`, aplicar estilos predefinidos con hojas de estilo (*style sheets*), o configurar permanentemente a través del archivo `matplotlibrc` [136].
- **Paquetes Externos:** Herramientas como SciencePlots proporcionan estilos preconfigurados para cumplir con los formatos de revistas académicas [131].
- **Formatos Vectoriales:** Guardar en PDF o SVG asegura que las figuras mantengan la claridad al redimensionarlas [123].

CAPÍTULO 4

Planeación

4.1. Validez por Juicio de Expertos

La validación de un proyecto técnico mediante el juicio de expertos constituye una etapa fundamental para asegurar su rigor, pertinencia y alineación con el conocimiento y las prácticas actuales en los campos de estudio relevantes. Esta sección, “4.1 Validez por Juicio de Expertos”, se dedica a documentar y analizar este proceso crucial, buscando corroborar la pertinencia del problema abordado, la solidez de la metodología empleada y la viabilidad de la solución propuesta para el “Modelo para representar comportamientos gravitacionales con dos cuerpos”.

A continuación, se detallarán las consultas realizadas a profesionales con experiencia en áreas directamente relacionadas con los objetivos y aplicaciones de este Trabajo Terminal. Se presentará el perfil de los expertos consultados, el método empleado para la recolección de sus aportaciones, un resumen de sus observaciones y valoraciones, y un análisis de cómo estas perspectivas externas contribuyen a la robustez y credibilidad general del proyecto. La incorporación de una visión cualificada es esencial no solo para refinar el enfoque del trabajo, sino también para asegurar que sus contribuciones sean significativas y estén bien fundamentadas dentro del estado del arte y sus potenciales campos de aplicación, como la simulación científica y el desarrollo de videojuegos.

4.1.1. Consulta con Experto en Optimización Evolutiva e Investigador de agentes en Videojuegos

Con el objetivo de validar la pertinencia del problema abordado y la adecuación de la solución propuesta en este Trabajo Terminal, se realizó una consulta especializada. Esta validación externa es crucial para asegurar que el proyecto se alinea con las necesidades y desafíos actuales en los campos de la simulación y sus potenciales aplicaciones.

Perfil del Experto Consultado

Se consultó al Maestro en Ciencias Computacionales (M.CC.) José Alberto Torres León, egresado del Centro de Investigación en Cómputo (CIC) del Instituto Politécnico Nacional, y actualmente estudiante de último año del Doctorado en

Ciencias de la Computación en la misma institución. Sus áreas de especialización incluyen la generación procedimental de contenido, agentes inteligentes para videojuegos (jugadores y dinámicos), aprendizaje por refuerzo, optimización evolutiva y *Deep Learning*. Su experiencia en optimización y su profundo conocimiento del desarrollo y las limitaciones en el ámbito de los videojuegos lo convierten en un validador idóneo para el presente proyecto.

Método de Consulta

La consulta se llevó a cabo mediante una reunión formal el martes 06 de mayo de 2025, entre las 12:35 y las 13:10 horas, en las instalaciones de investigación del CIC. Previo a la discusión, el M.CC. Torres León revisó el borrador del presente reporte técnico. Durante la reunión, se le expuso verbalmente el propósito, la visión y la evolución del proyecto, desde su concepción inicial como un “videojuego que permitiera jugar con las cualidades físicas gravitacionales de objetos para atraer, repeler, o que se mantengan en un equilibrio modificando características de los objetos en tiempo de ejecución”, hasta su iteración final como un “modelo de simulación de dos cuerpos celestes que permita el cambio de características en los cuerpos para encontrar los valores idóneos que mantengan al sistema en estabilidad”.

Resumen de Opiniones y Observaciones Obtenidas

Tras la revisión del material y la discusión, el M.CC. Torres León aportó las siguientes observaciones y valoraciones:

- **Comprensión del Problema y Enfoque:**

- Inicialmente, el experto señaló que la afirmación de que el estado del arte actual en simuladores no permite la definición propia de parámetros no era del todo precisa. Se clarificó que la limitación principal que el proyecto busca abordar es la *modificación de estos parámetros en tiempo de ejecución*.
- Identificó correctamente que el problema subyacente que se trata de resolver es de naturaleza combinatoria y subrayó la necesidad de una definición formal y clara de dicho problema combinatorio dentro del reporte.
- Enfatizó la importancia de establecer un objetivo específico para la evaluación de resultados, que permita corroborar que el comportamiento simulado sea el adecuado y esperado.

- **Aplicabilidad y Relevancia en Videojuegos:**

- Confirmó que las herramientas actuales para motores físicos en el desarrollo de videojuegos “no poseen flexibilidad en el entorno de desarrollo” para el tipo de interacciones dinámicas que este proyecto plantea, limitando la implementación de mecánicas como las concebidas originalmente para el Trabajo Terminal.
- Señaló que la industria de los videojuegos busca realizar simulaciones apegadas a comportamientos físicos reales, pero explorando “casos y

resultados que no se han captado o no pasan en la realidad”, lo cual se alinea con la capacidad del modelo propuesto.

- Consideró plausible que modelos similares ya existan en videojuegos comerciales, pero como parte de “patentes o secretos industriales”, citando como ejemplo el jefe final del videojuego *Bayonetta 1*, donde cuerpos celestes son dirigidos dinámicamente. Esto resalta la dificultad de acceso a este conocimiento y la relevancia de una propuesta abierta.
- Opinó que la solución propuesta puede “dar una ilusión más realista de comportamientos gravitacionales en videojuegos”, con un “mucho más valor en juegos de tipo plataformero”.
- Prevé una “influencia positiva en la experiencia del usuario y del desarrollador” al contar con modelos de esta naturaleza.
- Destacó un “sesgo en la investigación” del área debido a que muchos avances no se documentan públicamente o están protegidos, limitando el acceso a información pública. Mencionó que la investigación en generación procedural de contenido y videojuegos ha ganado seriedad recientemente (desde aproximadamente 2016), aunque persiste cierto escepticismo en algunos círculos académicos.
- **Planeación y Delimitación del Proyecto:**
 - El experto consideró que el proyecto presenta una “buena dirección” y una “buena delimitación”, especialmente tras las sucesivas adaptaciones del alcance.
- **Notas Adicionales:**
 - Mencionó la existencia de optimizadores basados en fenómenos gravitacionales, aunque no especificó nombres.
 - Observó que el enfoque del reporte técnico parecía más orientado hacia la Ingeniería en Sistemas Computacionales (ISC) que hacia la Ingeniería en Inteligencia Artificial (IIA), una apreciación relevante para la presentación del trabajo.

Análisis de la Validación Experta y Vinculación con el Proyecto

Las observaciones del M.CC. José Alberto Torres León proporcionan una validación significativa tanto para la relevancia del problema como para la viabilidad y el impacto potencial de la solución propuesta.

- **Validación de la Pertinencia del Problema:**
 - La confirmación de que las herramientas actuales en motores de videojuegos carecen de la flexibilidad necesaria para modificar parámetros físicos gravitacionales en tiempo de ejecución (como se propone) valida directamente la necesidad que este proyecto busca satisfacer.
 - Su comentario sobre la existencia de soluciones propietarias o no publicadas refuerza la justificación de desarrollar un modelo abierto y documentado que pueda servir como base para futuras investigaciones

o desarrollos, especialmente considerando el sesgo de investigación existente.

- La identificación del problema como combinatorio y la necesidad de su definición clara son aportes que refinan la presentación del desafío técnico que se aborda.

- **Respaldo a la Solución Propuesta:**

- La opinión de que el modelo puede ofrecer una “ilusión más realista” y tener un “mayor valor” en ciertos géneros de videojuegos respalda el potencial de aplicación de la solución.
- La valoración positiva sobre la “buena dirección” y “delimitación” del proyecto, tras conocer su evolución, sugiere que el enfoque actual es pertinente y manejable.
- La necesidad de un objetivo específico de evaluación de resultados, sugerida por el experto, se alinea con la metodología de investigación y se considerará fundamental para la etapa de pruebas y validación del modelo.

Gracias a la consulta con el M.CC. Torres León validamos que el problema de la modificación dinámica de parámetros gravitacionales en simulaciones es relevante, especialmente con miras a aplicaciones innovadoras como en videojuegos, y que la solución propuesta, aunque delimitada a dos cuerpos, aborda una brecha existente y tiene una dirección metodológica sólida. Las sugerencias sobre la definición del problema combinatorio y los criterios de evaluación serán integradas para fortalecer el rigor del proyecto.

4.1.2. Consulta con Experto en Optimización Evolutiva y Mecatrónica Aplicada

Para corroborar la relevancia del problema de investigación y la solidez del enfoque metodológico propuesto en este Trabajo Terminal, se llevó a cabo una consulta especializada con un experto en el campo de la optimización evolutiva y sus aplicaciones en ingeniería.

Perfil del Experto Consultado

Se consultó al Dr. Daniel Molina Pérez, Investigador titular en el área de Mecatrónica del Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC) del Instituto Politécnico Nacional (IPN). El Dr. Molina ostenta un Doctorado en Ingeniería de Sistemas Robóticos y Mecatrónica por el mismo centro. Sus líneas de investigación y especialización abarcan la algoritmia, algoritmos bioinspirados, optimización, optimización aplicada a problemas de ingeniería y optimización evolutiva. Su trayectoria y conocimiento en estas áreas lo constituyen como un validador idóneo para los aspectos centrales de este proyecto.

Método de Consulta

La consulta se efectuó mediante una reunión formal el lunes 12 de mayo de 2025, de 14:40 a 15:15 horas, en las instalaciones del departamento de investigación

de Mecatrónica del CIDETEC-IPN. Previo a la discusión, se proporcionó al Dr. Molina un borrador del presente reporte técnico para su revisión. Durante la reunión, se expuso verbalmente el propósito, los objetivos y la evolución conceptual del proyecto, desde su concepción inicial hasta la formulación actual del modelo de simulación y optimización.

Resumen de Opiniones y Observaciones Obtenidas

Tras la presentación y discusión, el Dr. Molina Pérez aportó las siguientes valoraciones y observaciones clave:

- **Naturaleza del Problema de Optimización:**

- Confirmó que el problema, tal como está planteado (encontrar masas que conduzcan a un exponente de Lyapunov¹ indicativo de estabilidad local), puede interpretarse como un **problema de factibilidad**² si el único objetivo es encontrar *cualquier* conjunto de masas que cumpla la restricción de estabilidad. No obstante, si se buscara el *mejor* conjunto (por ejemplo, el que minimice alguna otra métrica mientras mantiene la estabilidad, o el exponente de Lyapunov más cercano a cero sin ser positivo), se trataría de un problema de optimización con una función objetivo definida.
- Subrayó la importancia de definir claramente si el objetivo es la mera factibilidad o la optimización de un criterio específico.

- **Variables de Decisión y Aplicabilidad del Modelo:**

- Identificó correctamente que, al ser las masas las variables de decisión, el modelo se orienta a la estabilidad de sistemas de cuerpos celestes (limitado a dos en esta etapa).
- Hipotetizó que considerar variables adicionales como la densidad o la relación masa/volumen (kg/m^3) podría enriquecer el modelo y la resolución del problema, sugiriendo una investigación en astrofísica y aerofísica.

- **Dinámica del Sistema y la Optimización:**

- Inicialmente, dedujo que las masas serían fijas *después* del proceso de optimización para la simulación. Se clarificó que, si bien las masas son el resultado de la optimización y se usan para *una* simulación, el concepto más amplio del proyecto implica que estas podrían cambiar dinámicamente en la simulación general (aunque no durante un ciclo de optimización individual para encontrar *un* conjunto de masas estables).
- Respecto a la simulación con REBOUND (específicamente con WH-Fast), reconoció su idoneidad para determinar trayectorias.

¹El **exponente de Lyapunov** (LE) caracteriza la tasa de separación de trayectorias cercanas en sistemas dinámicos; un LE máximo positivo indica caos, mientras que uno no positivo sugiere estabilidad [13, 19]. Es crucial en el *estudio del caos* y la *teoría de la dimensión* [13, 21, 22].

²Un **problema de factibilidad** busca determinar si existe al menos una solución que satisface un conjunto de restricciones, sin optimizar una función objetivo particular. Cf. [137].

- Consideró que el problema de optimización, tal como se describió (encontrar un conjunto de masas), no es inherentemente dinámico en el sentido de que la función de aptitud (exponente de Lyapunov para un conjunto dado de masas iniciales) no varía entre ejecuciones idénticas. Sin embargo, reconoció el aspecto dinámico *general* del sistema si las masas se modifican en función del tiempo t durante la simulación. Esta distinción es crucial: la optimización busca un estado inicial, la simulación general podría explorar cambios en ese estado.
- Sugirió la necesidad de un **activador** (*trigger*)³ que inicie una nueva búsqueda (optimización/factibilidad) si la aptitud del sistema cambia durante una simulación extendida, implicando un chequeo continuo del fitness.

- **Justificación Metodológica y Relevancia:**

- Consideró el modelo como “super interesante”, con aplicaciones potenciales en la simulación científica para el desarrollo de motores físicos en videojuegos, aunque de interés prioritario en el ámbito aeroespacial y astrofísico.
- Afirmó que el uso de **algoritmos bioinspirados**⁴ se justifica en problemas de simulación donde esta actúa como una “caja negra”, desconociéndose a priori la relación explícita entre las variables de entrada y la métrica de salida.
- Mencionó que los **Algoritmos Genéticos**⁵ son particularmente adecuados para resolver problemas con características dinámicas (refiriéndose a la capacidad de adaptar la búsqueda si el entorno o los objetivos cambian, lo cual se alinea con la idea del *trigger*).

Análisis de la Validación Experta y Vinculación con el Proyecto

Las observaciones del Dr. Daniel Molina Pérez ofrecen una validación sustancial y directrices valiosas para el proyecto:

- **Validación de la Pertinencia del Problema:**

- La distinción entre problema de factibilidad y optimización es fundamental y será clarificada en la formulación del problema dentro del reporte, enfocándose en la búsqueda de conjuntos de parámetros que satisfagan el criterio de estabilidad del exponente de Lyapunov.
- Su énfasis en las aplicaciones aeroespaciales y astrofísicas refuerza la relevancia del dominio elegido, aunque el modelo pueda tener implicaciones secundarias en otras áreas como la simulación para videojuegos.

³Un **activador** (*trigger*) es una condición o evento que, al cumplirse, inicia un proceso subsecuente, como la re-evaluación o un nuevo ciclo de optimización.

⁴Los **algoritmos bioinspirados** son métodos de optimización inspirados en procesos naturales como la evolución o el comportamiento de enjambres [90, 91].

⁵Los **Algoritmos Genéticos** (AGs) son algoritmos bioinspirados basados en la selección natural y la genética, que operan sobre una población de soluciones para evolucionar hacia mejores resultados [91]. Ver Sección 3.8.

- La confirmación de que los algoritmos bioinspirados son una elección justificada para problemas donde la simulación actúa como evaluador de “caja negra” respalda la metodología central del proyecto.
- **Respaldo a la Solución Propuesta y Metodología:**
 - El reconocimiento de REBOUND y WHFast como herramientas adecuadas para la simulación de trayectorias apoya la elección tecnológica.
 - La sugerencia de un *trigger* para la re-optimización basada en el chequeo continuo del fitness se alinea con la visión de un sistema adaptable y será considerada para futuras extensiones del modelo, aunque la implementación actual se centre en encontrar un conjunto inicial estable.
 - Su comentario sobre los Algoritmos Genéticos siendo aptos para problemas dinámicos valida su elección como el paradigma bioinspirado principal, especialmente si se considera la adaptación del sistema a largo plazo.
- **Consideraciones para el Desarrollo y Reporte:**
 - La necesidad de un modelo matemático claro para la trayectoria (satisficha por REBOUND) es un punto ya cubierto.
 - La hipótesis sobre la inclusión de densidad y kg/m^3 se tomará como una posible línea de investigación futura para enriquecer el modelo, aunque excede el alcance actual de dos cuerpos puntuales con masa como única variable de decisión.
 - Se refinará la descripción del “dinamismo” del problema, distinguiendo claramente entre la naturaleza estática de una evaluación de fitness individual (para un conjunto de masas) y el comportamiento dinámico de la simulación gravitacional en sí misma a lo largo del tiempo.

4.2. Análisis de Requerimientos

La sección de requerimientos constituye el pilar fundamental para el desarrollo exitoso de cualquier modelo o software, ya que establece de manera clara y estructurada las expectativas sobre lo que el modelo debe hacer y cómo debe comportarse. En este sentido, los requerimientos se clasifican en dos grandes categorías: **funcionales** y **no funcionales**. Los requerimientos funcionales (RF) detallan las acciones específicas que el modelo debe ejecutar, como la modificación dinámica de parámetros o la simulación de interacciones gravitacionales. Por su parte, los requerimientos no funcionales (RNF) especifican las cualidades y limitaciones del modelo, incluyendo aspectos como el rendimiento, la escalabilidad, la usabilidad y la compatibilidad con hardware determinado.

En este reporte técnico, se presenta un análisis detallado del contenido textual para identificar tanto los requerimientos explícitos como los implícitos, garantizando que el modelo de simulación gravitacional cumpla con los objetivos establecidos. Entre estos objetivos se incluye la capacidad de ajustar dinámicamente la masa

de los cuerpos celestes y asegurar el cumplimiento de las leyes de la física y la mecánica celeste. Además, se han tomado en cuenta factores esenciales como la eficiencia computacional, la estabilidad de las simulaciones y la accesibilidad para usuarios sin experiencia técnica avanzada.

La definición precisa y la documentación adecuada de estos requerimientos no solo orientarán el proceso de desarrollo, sino que también facilitarán la validación y verificación del modelo una vez implementado. Esto asegura que el producto final esté alineado con las necesidades y expectativas de los usuarios, ya sean investigadores, académicos o profesionales del sector.

4.2.1. Requerimientos Funcionales (RF)

Los requerimientos funcionales describen las capacidades que el modelo debe tener:

Tabla 4.1: Requerimientos Funcionales

ID	Descripción	Prioridad	Actor	Criterios de Aceptación
RF-001	El sistema debe permitir la modificación dinámica de la masa de los cuerpos celestes durante la simulación.	Alta	Usuario/Modelo	El usuario puede modificar la masa de los cuerpos celestes en tiempo real y la simulación responde adecuadamente a estos cambios.
RF-002	El sistema debe simular las interacciones gravitacionales entre dos cuerpos celestes.	Alta	Modelo	La simulación calcula y representa correctamente las fuerzas gravitacionales entre los cuerpos según las leyes físicas establecidas.
RF-003	El sistema debe usar el método multipolar rápido (FMM) y el algoritmo de Barnes-Hut para optimizar los cálculos gravitacionales.	Media	Modelo	Los algoritmos se implementan correctamente y mejoran el rendimiento de los cálculos en comparación con métodos directos.
RF-004	El sistema debe implementar algoritmos bioinspirados para ajustar dinámicamente los parámetros y encontrar valores idóneos que garanticen la estabilidad del sistema.	Alta	Modelo	Los algoritmos bioinspirados funcionan correctamente y logran mantener la estabilidad del sistema durante los cambios de masa.
RF-005	El sistema debe incluir un módulo para visualizar gráficamente la evolución de los dos cuerpos en iteraciones limitadas.	Media	Usuario	La visualización muestra claramente las trayectorias y estados de los cuerpos celestes durante la simulación.
RF-006	El sistema debe ofrecer una interfaz básica para modificar parámetros y ver resultados.	Media	Usuario	El usuario puede interactuar con la interfaz para ajustar parámetros y visualizar los cambios en la simulación.

4.2.2. Requerimientos No Funcionales (RNF)

Los requerimientos no funcionales especifican las cualidades y restricciones del modelo, incluyendo los dos implícitos que me pediste añadir:

Tabla 4.2: Requerimientos No Funcionales

ID	Descripción	Prioridad	Actor	Criterios de Aceptación
RNF-001	El sistema debe optimizar la complejidad de las simulaciones de n-cuerpos para ser eficiente.	Alta	Modelo	La simulación se ejecuta en tiempos razonables sin consumir excesivos recursos computacionales.
RNF-002	El sistema debe permitir la extensión futura a más de dos cuerpos.	Media	Modelo	La arquitectura del sistema es escalable y puede adaptarse para simular más cuerpos sin cambios estructurales mayores.
RNF-003	El sistema debe respetar las leyes de la física y la mecánica celeste en las interacciones gravitacionales.	Alta	Modelo	Las simulaciones producen resultados coherentes con las leyes físicas establecidas.
RNF-004	El sistema debe mantener simulaciones estables al modificar parámetros dinámicamente.	Alta	Modelo	La simulación no colapsa ni presenta comportamientos erráticos cuando se modifican los parámetros durante la ejecución.
RNF-005	La interfaz debe ser intuitiva y accesible sin necesidad de conocimientos avanzados.	Media	Usuario	Usuarios sin experiencia técnica pueden utilizar el sistema sin dificultades significativas.
RNF-006	El sistema debe ejecutarse en hardware de gama media (procesadores multinúcleo, 16 GB de RAM).	Media	Modelo	El sistema funciona correctamente en equipos con las especificaciones mínimas establecidas.
RNF-007	El sistema debe integrarse con entornos virtuales como Unreal Engine.	Baja	Modelo	La integración con Unreal Engine u otros entornos similares es posible y funcional.
RNF-008	El sistema debe ser modular para facilitar actualizaciones y correcciones.	Media	Desarrollador	Los componentes del sistema están claramente separados y pueden modificarse independientemente.
RNF-009	El sistema debe incluir documentación clara para usuarios y desarrolladores.	Media	Desarrollador	La documentación es completa, comprensible y cubre todos los aspectos relevantes del sistema.

4.3. Planteamiento del Problema Principal: Factibilidad de Estabilidad del Modelo

El objetivo fundamental y principal de este proyecto es determinar si existe al menos una configuración de masas (m_1, m_2) para un sistema de dos cuerpos celestes que produzca un comportamiento gravitacional localmente estable. Esta

estabilidad se cuantifica principalmente a través del Exponente de Lyapunov (LE), donde valores más bajos (idealmente negativos, $LE < 0$) indican mayor estabilidad. Este es, en esencia, un *problema de factibilidad*: buscamos probar la existencia de al menos una solución que satisfaga un conjunto de criterios.

A continuación, se presenta la formulación matemática de este problema de factibilidad. Posteriormente, se detallará cómo este problema puede ser abordado mediante una formulación de optimización, una estrategia común para resolver problemas de factibilidad complejos, especialmente cuando se utilizan algoritmos bioinspirados.

4.3.1. Definiciones Preliminares: Variables, Parámetros y Función de Evaluación

Para formular el problema, primero definimos las variables de decisión, los parámetros del sistema, los conjuntos relevantes y la función de evaluación.

Variables de Decisión

Las decisiones fundamentales giran en torno a las masas de los dos cuerpos celestes:

- m_1 : Masa del primer cuerpo celeste.
 - Naturaleza: Continua.
 - Unidades: Kilogramos (kg).
- m_2 : Masa del segundo cuerpo celeste.
 - Naturaleza: Continua.
 - Unidades: Kilogramos (kg).

Parámetros del Modelo

Estos son los valores fijos o predefinidos que contextualizan el problema:

- \mathcal{C}_{ini} : Conjunto de condiciones iniciales del sistema. Incluye:
 - $\mathbf{r}_{1,0}, \mathbf{r}_{2,0}$: Vectores de posición inicial para el cuerpo 1 y el cuerpo 2, respectivamente (e.g., (x_0, y_0, z_0) para cada uno). Unidades: metros (m).
 - $\mathbf{v}_{1,0}, \mathbf{v}_{2,0}$: Vectores de velocidad inicial para el cuerpo 1 y el cuerpo 2, respectivamente (e.g., $(v_{x,0}, v_{y,0}, v_{z,0})$ para cada uno). Unidades: metros por segundo (m/s).
- \mathcal{P}_{sim} : Conjunto de parámetros de la simulación gravitacional. Incluye:
 - G : Constante de gravitación universal. Unidades: $\text{N}\cdot\text{m}^2/\text{kg}^2$. (Valor aproximado: $6.674 \times 10^{-11} \text{ N}\cdot\text{m}^2/\text{kg}^2$)
 - Δt : Paso de tiempo de integración numérica. Unidades: segundos (s).
 - T_{max} : Tiempo máximo de simulación para cada evaluación del Exponente de Lyapunov. Unidades: segundos (s).

- I_{type} : Especificación del tipo de integrador numérico utilizado (e.g., WHFast).
- Otros parámetros relevantes para los algoritmos de cálculo de interacciones (e.g., parámetros específicos de FMM, Barnes-Hut si se usan).
- α : Constante real positiva que define la cota superior para la relación de masas (m_2/m_1). Adimensional.
- β : Constante real positiva que define la cota inferior para la relación de masas (m_2/m_1). Adimensional.
- S_{target} : Umbral de estabilidad objetivo para el Exponente de Lyapunov. Para que una configuración sea considerada factiblemente estable, se requiere $LE \leq S_{\text{target}}$.
 - Este valor es crucial para el problema de factibilidad. Puede ser un L_{target} específico (si $L_{\text{target}} < 0$ está definido como el objetivo mínimo de estabilidad).
 - Alternativamente, S_{target} puede ser $-\delta_{\text{stability}}$, donde $\delta_{\text{stability}}$ es una constante positiva pequeña (e.g., 10^{-5}), asegurando que LE sea significativamente negativo.
 - Unidades: Adimensional o s^{-1} .
- ε_m : Un valor positivo pequeño para asegurar que las masas sean estrictamente positivas (e.g., 10^{-6} kg). Unidades: kg.
- $\varepsilon_{\text{ineq}}$: Un valor positivo pequeño para manejar desigualdades estrictas en restricciones numéricas (e.g., 10^{-9}). Adimensional o unidades consistentes con la restricción.
- $m_{1,\min}, m_{1,\max}$: Límites absolutos inferior y superior para la masa m_1 (opcional). Unidades: kg.
- $m_{2,\min}, m_{2,\max}$: Límites absolutos inferior y superior para la masa m_2 (opcional). Unidades: kg.

Función de Evaluación (Exponente de Lyapunov)

El Exponente de Lyapunov, LE , es una función que depende de las masas de los cuerpos y del conjunto de parámetros de simulación y condiciones iniciales. No es una forma cerrada simple, sino el resultado de una simulación numérica:

$$LE(m_1, m_2; \mathcal{C}_{\text{ini}}, \mathcal{P}_{\text{sim}}) : \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R} \quad (4.1)$$

Este valor, $LE(m_1, m_2; \mathcal{C}_{\text{ini}}, \mathcal{P}_{\text{sim}})$, es el Exponente de Lyapunov calculado para una configuración dada de masas m_1, m_2 , con condiciones iniciales \mathcal{C}_{ini} y parámetros de simulación \mathcal{P}_{sim} . Sus unidades son adimensionales o s^{-1} .

4.3.2. Definición Formal Matemática del Problema de Factibilidad

Sea el espacio de decisión $\mathcal{M} = \{(m_1, m_2) \in \mathbb{R}^2 : m_1, m_2 \geq 0\}$. Definimos el conjunto factible:

$$\mathcal{F} = \left\{ (m_1, m_2) \in \mathcal{M} \mid \begin{array}{l} m_1 \geq \varepsilon_m, \\ m_2 \geq \varepsilon_m, \\ m_2 - \beta m_1 \geq 0, \\ \alpha m_1 - m_2 \geq 0, \\ 10 m_1 - m_2 \geq \varepsilon_{\text{ineq}}, \\ \text{LE}(m_1, m_2; \mathcal{C}_{\text{ini}}, \mathcal{P}_{\text{sim}}) \leq S_{\text{target}}, \\ m_{1,\min} \leq m_1 \leq m_{1,\max}, \\ m_{2,\min} \leq m_2 \leq m_{2,\max} \end{array} \right\}. \quad (4.2)$$

El problema de factibilidad consiste en:

$$\text{Encontrar } (m_1, m_2) \in \mathcal{F}.$$

Si $\mathcal{F} \neq \emptyset$, existe al menos una configuración de masas que satisface todas las restricciones y, por tanto, el sistema puede alcanzar estabilidad bajo el criterio S_{target} .

4.3.3. Abordaje del Problema de Factibilidad mediante Formulación de Optimización

Para resolver el problema de factibilidad definido en la sección 4.3.2, especialmente cuando la función LE es compleja y el espacio de búsqueda es extenso, es ventajoso reformularlo como un problema de optimización. Esta aproximación es particularmente adecuada para la aplicación de algoritmos bioinspirados.

La estrategia consiste en buscar la minimización del Exponente de Lyapunov (LE) sujeto a las restricciones del dominio. Si el proceso de optimización encuentra una solución (m_1^*, m_2^*) cuyo valor $LE(m_1^*, m_2^*)$ satisface $LE \leq S_{\text{target}}$ y todas las demás restricciones, entonces se ha encontrado una solución al problema de factibilidad.

Descripción Verbal del Objetivo y Restricciones de la Formulación de Optimización

- **Objetivo de la Formulación de Optimización:** Se busca minimizar el valor del Exponente de Lyapunov $Z = LE(m_1, m_2; \mathcal{C}_{\text{ini}}, \mathcal{P}_{\text{sim}})$. El propósito principal de esta minimización es encontrar *alguna* configuración de masas (m_1, m_2) para la cual el valor de LE resultante sea igual o inferior al umbral de estabilidad S_{target} , satisfaciendo a su vez todas las demás restricciones (positividad, relaciones de masa, rangos).
- **Restricciones del Dominio de Búsqueda (para la optimización):** Las mismas restricciones que definen el espacio de soluciones factibles (sección 4.3.2, puntos 1, 2, 3, 5 y 6), excluyendo la condición de $LE \leq S_{\text{target}}$ (punto 4 de la sección 4.3.2), que se convierte en el criterio de éxito de la búsqueda.

Definición Formal Matemática de la Formulación de Optimización

El problema se formula matemáticamente como:

$$\begin{aligned}
 & \underset{m_1, m_2}{\text{Minimizar}} \quad Z(m_1, m_2) = LE(m_1, m_2; \mathcal{C}_{\text{ini}}, \mathcal{P}_{\text{sim}}) \\
 & \text{sujeto a} \\
 & \quad m_1 \geq \varepsilon_m \\
 & \quad m_2 \geq \varepsilon_m \\
 & \quad m_2 - \beta \cdot m_1 \geq 0 \\
 & \quad \alpha \cdot m_1 - m_2 \geq 0 \\
 & \quad 10 \cdot m_1 - m_2 \geq \varepsilon_{\text{ineq}} \\
 & \quad m_{1,\min} \leq m_1 \leq m_{1,\max} \quad (\text{Opcional}) \\
 & \quad m_{2,\min} \leq m_2 \leq m_{2,\max} \quad (\text{Opcional}) \\
 & \quad m_1, m_2 \in \mathbb{R}
 \end{aligned} \tag{4.3}$$

Criterio de Éxito para Resolver el Problema de Factibilidad mediante esta Optimización

Se considera que se ha encontrado una solución al problema de factibilidad original (sección 4.3.2) si el proceso de optimización produce un par de masas (m_1^*, m_2^*) que:

1. Satisface todas las restricciones del problema de optimización (4.3).
2. Y para el cual el valor de la función objetivo evaluada $Z^* = LE(m_1^*, m_2^*; \mathcal{C}_{\text{ini}}, \mathcal{P}_{\text{sim}})$ cumple con la condición de estabilidad original:

$$Z^* \leq S_{\text{target}} \tag{4.4}$$

Si, tras ejecutar el algoritmo de optimización, el valor mínimo de Z encontrado (Z_{\min}) es persistentemente mayor que S_{target} (o si no se puede encontrar ninguna solución que satisfaga las restricciones de la ecuación (4.3)), se concluiría que, dentro del alcance y las capacidades del método de optimización utilizado, no se ha podido encontrar una configuración de masas que resuelva el problema de factibilidad.

4.3.4. Consideraciones Adicionales

- **Optimización Pura de la Estabilidad (Paso Secundario Opcional):** Si el objetivo secundario fuera, además de encontrar *una* configuración estable, encontrar la configuración *más* estable (el LE más negativo posible), la formulación de optimización (ecuación (4.3)) sigue siendo válida. El interés no se detendría al alcanzar S_{target} , sino que el algoritmo continuaría buscando minimizar LE tanto como sea posible.
- **Interpretación Física de Restricciones:** La relación $\beta \leq \frac{m_2}{m_1} \leq \alpha$ y la condición $m_2 < 10 \cdot m_1$ son ejemplos de restricciones que aseguran proporciones de masa físicamente significativas o relevantes para el estudio específico.

CAPÍTULO 5

Análisis

5.1. Análisis de Procesos Internos del Programa

El análisis detallado de los procesos internos del software constituye un pilar esencial para el diseño, implementación y validación del modelo de simulación gravitacional propuesto en este proyecto. Comprender la secuencia de operaciones, el flujo de datos y la interacción entre los componentes clave (como la biblioteca REBOUND, la lógica de modificación y la interfaz de usuario) es fundamental para abordar el desafío central del proyecto: habilitar la modificación dinámica de parámetros, como la masa, en un sistema de N-cuerpos (initialmente abordado con dos cuerpos) durante la ejecución de la simulación.

Esta sección establece la base operativa sobre la cual se construirá la solución. Proporciona el mapa necesario para implementar las innovaciones clave del proyecto –la modificación dinámica de parámetros– de manera estructurada, eficiente y verificable, asegurando que el modelo final cumpla con el objetivo de superar las limitaciones de las herramientas de simulación actuales.

5.1.1. Proceso Interno 01: Captura Parámetros

Objetivo del Proceso

El propósito principal de la actividad “**Captura Parámetros**” es recopilar, validar y almacenar todos los parámetros de configuración necesarios proporcionados por el usuario a través de la interfaz gráfica (UI). Estos parámetros son esenciales para definir cómo se ejecutará tanto la simulación gravitacional como el proceso de optimización bioinspirado, asegurando que la configuración sea correcta y coherente antes de iniciar la ejecución del sistema.

Entradas Principales

- **Evento de inicio:** Interacción inicial del usuario con la UI para comenzar la configuración de parámetros.
- **Interacciones del usuario:** Acciones como clicks, selecciones e ingreso de texto o números en los campos específicos de la UI.

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 15.1.

1. Presentar Formulario/Interfaz UI

- La UI muestra una pantalla o conjunto de controles para ingresar la configuración
- Campos presentados incluyen:
 - **Parámetros a Optimizar:** Rangos (mínimo/máximo) de parámetros variables
 - **Restricciones del Sistema:** Condiciones obligatorias (ej. proporción máxima de masas)
 - **Configuración del Optimizador:**
 - Indicador de fitness preseleccionado (Exponente de Lyapunov LE)
 - Criterios de parada (nº máximo de generaciones, umbral de mejora)
 - **Parámetros del Algoritmo Bioinspirado:** Tamaño de población, tasas de mutación/crossover
 - **Configuración de la Simulación Base:**
 - Parámetros fijos: G , T_{\max} , dt
 - Selección de integrador de REBOUND (si aplica)
 - **Configuración de Visualización:**
 - Instantes de tiempo t_{vis} para captura de datos
 - Variables a visualizar (trayectoria 2D, energía vs tiempo)

2. Recibir Entradas del Usuario

- La UI captura activamente valores ingresados/seleccionados en cada campo

3. Iniciar Ciclo de Validación

- Validaciones incluyen:
 - 1(a) Tipo de dato correcto (valores numéricos donde corresponda)
 - 1(b) Rangos lógicos ($\min < \max$)
 - 1(c) Criterios de parada coherentes (n° generaciones > 0)
 - 1(d) $t_{\text{vis}} \in [0, T_{\max}]$
 - 1(e) Consistencia $T_{\max} \geq$ último t_{vis}
 - 1(f) Restricciones físicamente posibles (ej. proporción de masas > 0)

4. Decisión de Validez

- **Si ocurren errores:**
 - Generación de mensajes de error específicos
 - Resaltado de campos erróneos en UI
 - Retorno al paso de entrada de datos
- **Si válido:**
 - Creación de estructura ConfigurationData

5. Almacenar Configuración Validada

- Valores guardados en estructura ConfigurationData
- Accesible por otros componentes del sistema

6. Habilitar Inicio

- Activación de botón “Iniciar Simulación/Optimización”

Lógica Interna y Decisiones

- Ciclo de validación activado por acción de usuario
- Nodo de decisión principal: ¿Todas las entradas válidas?
 - Bucle de corrección hasta validación exitosa
 - Transición a ejecución tras validación satisfactoria

Manejo de Datos Específico

- **Datos crudos:** Valores directos de UI
- **Indicadores de validación:** Resultados booleanos por campo
- **ConfigurationData:** Estructura final con parámetros validados

Salidas Principales

- Estructura ConfigurationData completa
- Estado UI actualizado (mensajes de error o botón habilitado)

Interacciones Internas

- Comunicación con componentes UI (campos de texto, listas, botones)
- Preparación de ConfigurationData para algoritmo bioinspirado

Diagrama del Proceso

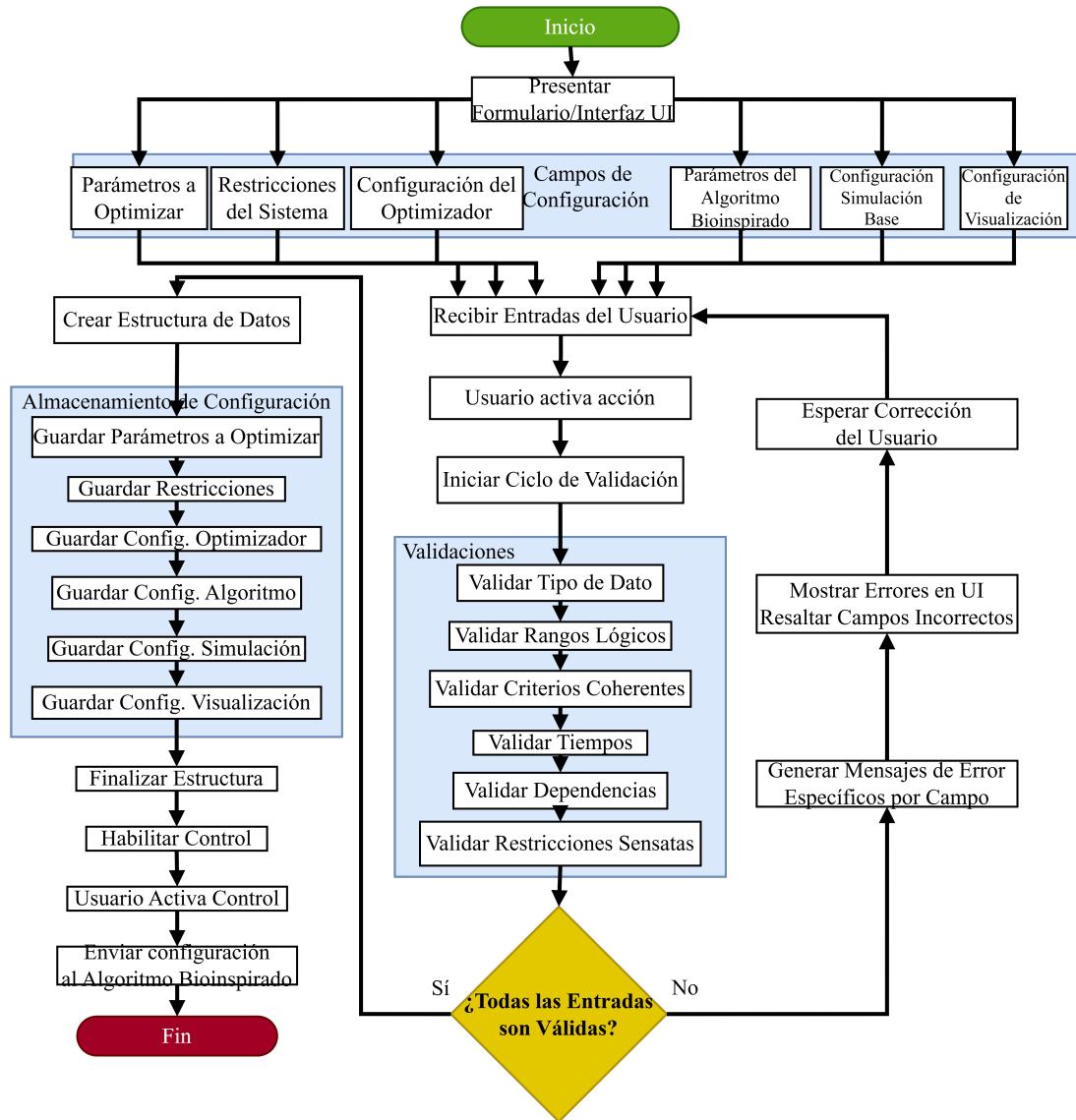


Figura 5.1: Diagrama de Proceso Interno 01: Captura de Parámetros

5.1.2. Proceso Interno 02: Inicializar Población

Objetivo del Proceso

El propósito principal de la actividad “**Inicializar Población**” es generar un conjunto inicial de N individuos que sirvan como punto de partida para el algoritmo bioinspirado del proyecto “*Modelo para representar comportamientos gravitacionales con dos cuerpos*”. Cada individuo representa un conjunto completo de parámetros a optimizar (ej masas de cuerpos), generados dentro de los rangos válidos especificados en la configuración.

Entradas Principales

- **ConfigurationData:** Estructura con:
 - Tamaño de población $N \in \mathbb{N}$
 - Rangos por parámetro: [mín, máx] (ej. [$\text{masa}_{1,2\text{mín}}$, $\text{masa}_{1,2\text{máx}}$])
- *Nota:* Restricciones funcionales se manejarán posteriormente

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 25.2

1. Leer Configuración Relevante

- Extraer de ConfigurationData:
 - N
 - Lista de parámetros a optimizar
 - Rangos asociados
- Verificar coherencia: $\forall \text{parámetro}, \text{mín} < \text{máx}$

2. Inicializar Contenedor de Población

- Crear estructura vacía (lista/array) para N individuos

3. Iniciar Bucle de Creación de Población

- (a) Inicializar contador $i = 0$
- (b) **Mientras** $i < N$:
 - Generar parámetros candidatos:
 - $\forall \text{parámetro}: \text{valor aleatorio} \in [\text{min}, \text{max}]$
 - Agregar candidato a la población
 - $i \leftarrow i + 1$

4. Verificar Población Generada

- Confirmar $|\text{población}| = N$
- (*Opcional*) Analizar distribución/diversidad

5. Finalizar y Devolver Población Inicial

- Retornar estructura poblada lista para optimización

Lógica Interna y Decisiones

- **Control del bucle:** Garantiza generación exacta de N individuos
- **Generación en rangos:**
 - Aleatorización restringida por [mín, máx]
 - Elimina necesidad de validación posterior
- **Verificación crítica:**
 - Precondición: $\text{mín} < \text{máx} \forall \text{ parámetro}$
- **Diversidad opcional:**
 - Métricas estadísticas no requeridas en flujo principal

Manejo de Datos Específico

- **Entrada:** ConfigurationData
- **Intermedios:**
 - Valores aleatorios \in rangos
 - Estructuras temporales por individuo
- **Salida:** Lista/array de N individuos

Salidas Principales

- **Población inicial:**
 - N individuos con parámetros \in rangos
 - Entrada para evaluación de LE

Interacciones Internas

- **Con ConfigurationData:** Lectura de parámetros
- **Con generador aleatorio:** Producción de valores válidos
- **Con estructura de población:** Almacenamiento y gestión

Diagrama del Proceso

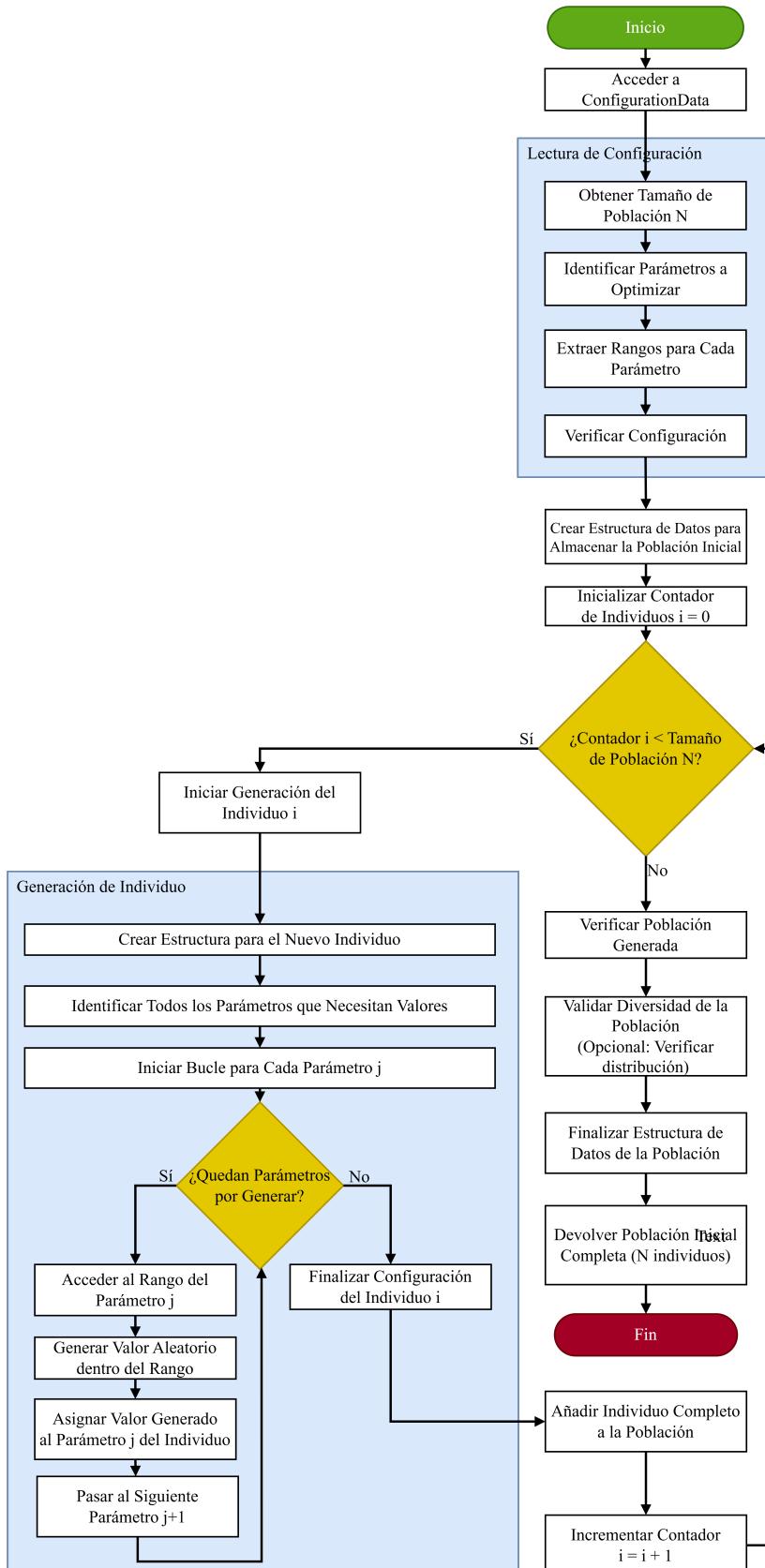


Figura 5.2: Diagrama de Proceso Interno 02: Inicializar Población

5.1.3. Proceso Interno 03: Generar Nueva Generación

Objetivo del Proceso

El propósito principal de la actividad “Generar Nueva Generación” es crear una nueva población de N individuos mediante operadores genéticos (Selección por Torneo, Cruzamiento SBX y Mutación Polinomial), garantizando que los parámetros permanezcan dentro de los rangos [mín, máx]. La evaluación de fitness se realiza posteriormente.

Entradas Principales

- Población actual:
 - N individuos con parámetros y F_p (fitness penalizado)
- ConfigurationData:
 - $N, P_c, P_m, \eta_c, \eta_m, k$
 - Rangos $[\min_i, \max_i]$ por parámetro

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 35.3

1. Inicializar Nueva Población

- Crear new_population vacío (capacidad N)

2. Iniciar Bucle de Generación

- (a) **While** $|\text{new_population}| < N$:

Paso 2a: Selección de padres

- Parent1 = Torneo3(k, F_p)
- Parent2 = Torneo3(k, F_p)

Paso 2b: Cruzamiento SBX

- Generar $r_1 \sim U[0, 1]$
- **If** $r_1 < P_c$:
 - $\{\text{Offspring1}, \text{Offspring2}\} \leftarrow \text{SBX}(\text{Parent1}, \text{Parent2}, \eta_c)$
- **Else**:
 - Clonar padres

Paso 2c: Mutación Polinomial

Para cada descendiente en $\{\text{Offspring1}, \text{Offspring2}\}$:

Para cada parámetro x_i del descendiente:

- Generar $r_2 \sim U[0, 1]$
- **If** $r_2 < P_m$:
 - $x_i \leftarrow \text{MutaciónPolinomial}(x_i, \eta_m, \min_i, \max_i)$

Paso 2d: Ajuste de límites

Para cada parámetro x_i en cada descendiente:

- $x_i \leftarrow \text{clip}(x_i, \min_i, \max_i)$

Paso 2e: Añadir descendientes

- Agregar Offspring1 a new_population
- Si $|\text{new_population}| < N$, agregar Offspring2

3. Devolver Nueva Población

- Retornar new_population con N individuos

Lógica Interna y Decisiones

- Control del bucle:
 - Condición de término: $|\text{new_population}| = N$
 - Manejo de N impar con adición condicional de Offspring2
- Operadores probabilísticos:
 - Cruzamiento: $P_c \in [0, 1]$
 - Mutación: $P_m \in [0, 1]$ por parámetro
- Preservación de restricciones:
 - Ajuste post-operadores: $x_i \in [\min_i, \max_i]$

Manejo de Datos Específico

- Entradas:
 - Población actual: Lista de estructuras {parámetros, F_p }
 - Parámetros operadores: $\{P_c, P_m, \eta_c, \eta_m, k\}$
- Intermedios:
 - Padres seleccionados
 - Descendientes pre-ajuste
- Salida:
 - new_population: Lista de N individuos no evaluados

Salidas Principales

- Nueva población:
 - N individuos con parámetros en rangos válidos
 - Listos para evaluación de fitness

Interacciones Internas

- **Con población actual:** Lectura de F_p para torneos
- **Con módulos genéticos:**
 - Torneo3(k), SBX3(η_c), MutaciónPolinomial3(η_m)
- **Con gestor de restricciones:** Aplicación de clip()

Diagrama del Proceso

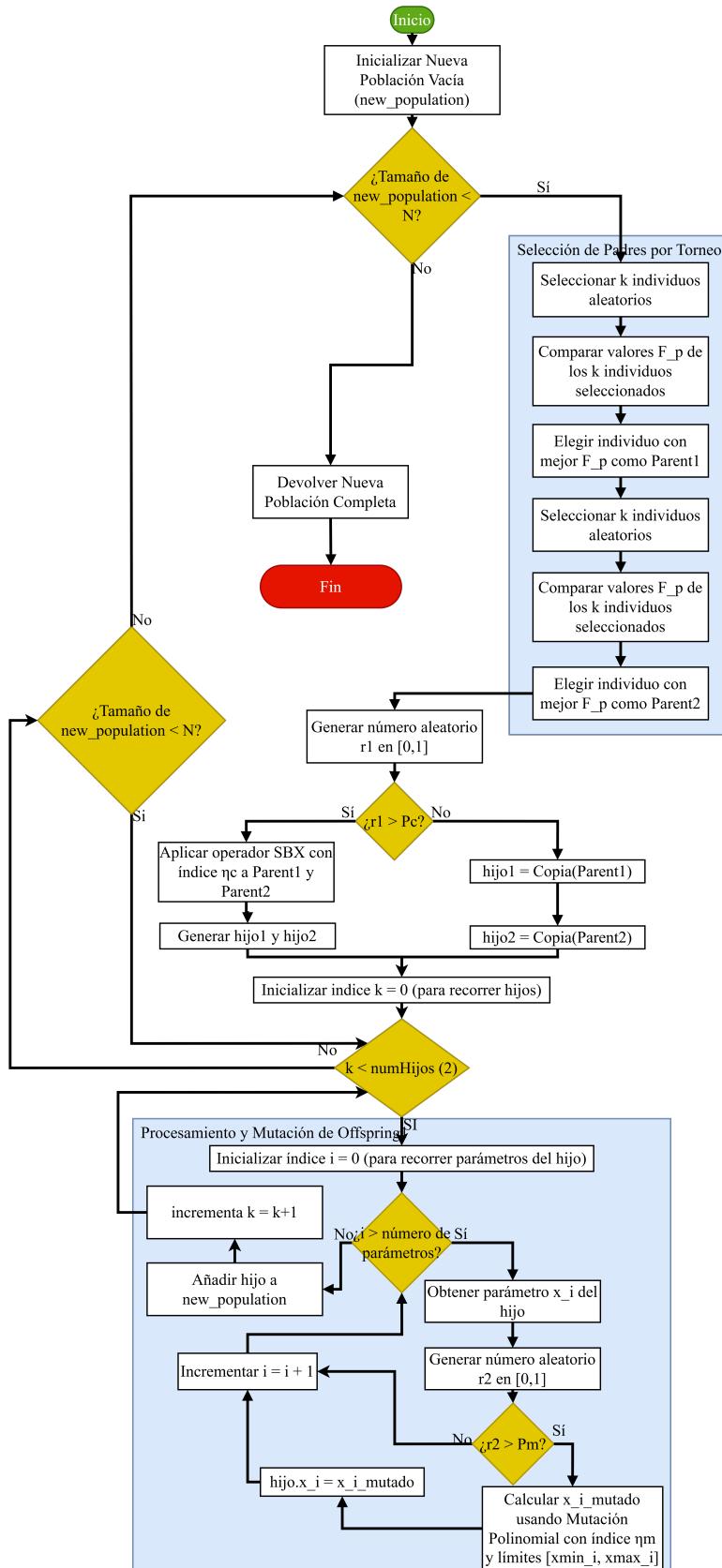


Figura 5.3: Diagrama de Proceso Interno 03: Generar Nueva Generación

5.1.4. Proceso Interno 04: Crear Nueva Simulación

Objetivo del Proceso

Instanciar un entorno de simulación vacío usando REBOUND, preparado para:

- Configuración posterior de parámetros
- Simulación de dinámica gravitacional de 2 cuerpos
- Cálculo del Exponente de Lyapunov (LE)

Entradas Principales

- **Trigger:** Señal de solicitud de creación
- *Nota:* No se requieren datos específicos de entrada

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 45.4

1. Invocar API de REBOUND

```
sim = rebound.Simulation()
```

2. Inicialización Interna

REBOUND ejecuta:

- Asignación de memoria para estructura `Simulation`
- Creación de contenedor vacío para partículas
- Configuración de valores por defecto:
 - $t_0 = 0$
 - $G = 6.674 \times 10^{-11} \text{ N} \cdot \text{m}^2/\text{kg}^2$
 - Integrador: WHFast

3. Generar Referencia

- Retorna handler/puntero: `Simulation*`
- Ejemplo: 0x7ffe3c3d8b50

4. Transferir Control

- Pasa referencia al proceso “Definir Condiciones”

Lógica Interna y Decisiones

- **Flujo lineal:** Sin bifurcaciones condicionales
- **Manejo de errores:**
 - Excepciones de API manejadas en nivel superior
- **Defaults:**
 - Valores modificables en configuración posterior

Manejo de Datos Específico

- **Entrada:** Solo trigger de activación
- **Intermedios:**
 - Estructura `Simulation` en memoria
 - Estado inicial: $t = 0$, partículas = \emptyset
- **Salida:** Referencia a `Simulation`

Salidas Principales

- **Simulation***: Puntero a instancia REBOUND vacía
 - Lista para configuración de parámetros
 - Integrable en flujo de optimización

Interacciones Internas

- **Con API REBOUND:**
 - Constructor `rebound.Simulation4()`
 - Gestión interna de memoria
- **Con subsistema de memoria:**
 - Asignación dinámica (4KB a MB según complejidad)
- **Con flujo de control:**
 - Paso de referencia a siguiente módulo

Diagrama del Proceso

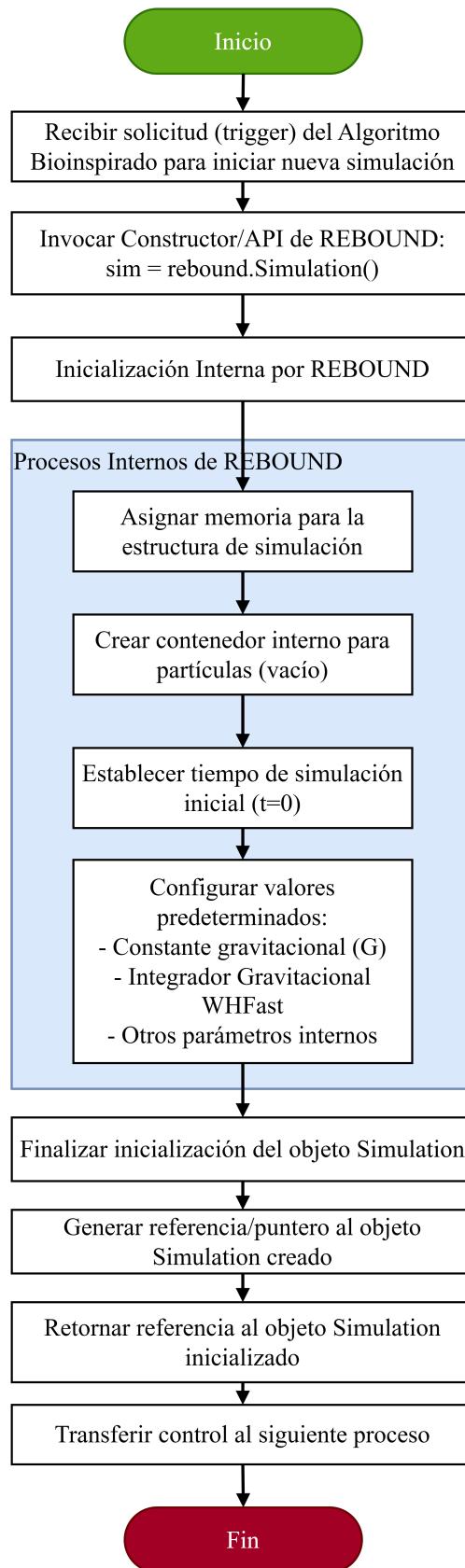


Figura 5.4: Diagrama de Proceso Interno 04: Crear Simulación

5.1.5. Proceso Interno 05: Agregar Cuerpos

Objetivo del Proceso

Incorporar un cuerpo celeste con propiedades físicas ($m, \mathbf{x}, \mathbf{v}$) a una simulación REBOUND existente, permitiendo:

- Configuración de sistemas de 2 cuerpos
- Cálculo posterior de estabilidad dinámica (LE)
- Integración en el ciclo de optimización

Entradas Principales

- **sim**: Referencia a objeto `Simulation` de REBOUND
- **body_params**: Estructura con:
 - $m \in \mathbb{R}^+$ (masa)
 - $\mathbf{x} = [x, y, z]$ (posición inicial)
 - $\mathbf{v} = [v_x, v_y, v_z]$ (velocidad inicial)

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 5.12

1. Recibir Parámetros

- Capturar `sim` y `body_params`

2. Desempaquetar Parámetros

- Extraer: $m, x, y, z, v_x, v_y, v_z$

3. Validación Opcional

- Chequear:
 - $\forall v \in \{m, x, y, z, v_x, v_y, v_z\} : v \neq \text{NaN}, \infty$
 - $m > 0$

4. Invocar API REBOUND

```
sim.add(m=m, x=x, y=y, z=z,
        vx=vx, vy=vy, vz=vz)
```

5. Procesamiento Interno REBOUND

- Asignar memoria para nueva partícula
- Actualizar contador: $n_{partículas} \leftarrow n_{partículas} + 1$

- Recalcular propiedades del sistema (opcional)

6. Verificación de Estado (Opcional)

- Confirmar $n_{partículas} = n_{prev} + 1$

Lógica Interna y Decisiones

- **Flujo lineal:** Sin bifurcaciones principales
- **Validaciones:**
 - Opcionales, dependientes de implementación
 - Manejo de errores delegado a REBOUND

Manejo de Datos Específico

- **Entradas:**
 - Referencia a simulación existente
 - Parámetros físicos estructurados
- **Intermedios:**
 - Estructura interna de partícula en REBOUND
- **Salida:**
 - Simulación modificada in-place

Salidas Principales

- `sim` actualizado con nuevo cuerpo
 - Listo para integración numérica
 - Configuración adicional posible

Interacciones Internas

- **Con API REBOUND:**
 - Método `add5()` para gestión de partículas
 - Administración interna de memoria
- **Con subsistema físico:**
 - Actualización de propiedades del sistema

Diagrama del Proceso

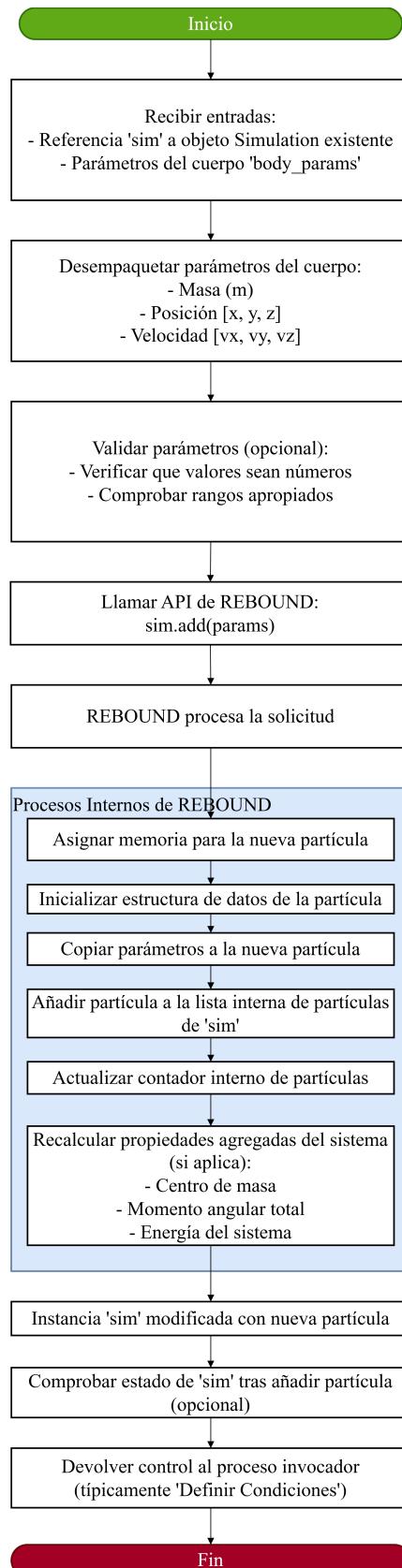


Figura 5.5: Diagrama de Proceso Interno 05: Agregar Cuerpos

5.1.6. Proceso Interno 06: Definir Condiciones

Objetivo del Proceso

Configurar parámetros operativos clave en una simulación REBOUND para:

- Establecer condiciones de integración numérica
- Definir constantes físicas fundamentales
- Preparar simulación para cálculo de LE

Entradas Principales

- **sim**: Referencia a objeto `Simulation` configurado
- **sim_params**: Estructura con:
 - $dt \in \mathbb{R}^+$ (paso temporal)
 - $G \in \mathbb{R}^+$ (constante gravitacional)
 - $T_{\max} \in \mathbb{R}^+$ (tiempo máximo)

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 65.6

1. Recibir Instancia y Parámetros

- Capturar `sim` y `sim_params`

2. Desempaquetar Parámetros

- Extraer: dt, G, T_{\max}

3. Validación Opcional

$$\begin{cases} dt > 0 \\ G > 0 \\ T_{\max} > \text{sim.t} \end{cases}$$

4. Establecer Parámetros en REBOUND

```
sim.dt = dt    # Paso temporal
sim.G = G      # Constante gravitacional
```

5. Procesar Tiempo Máximo

- Almacenar T_{\max} para control externo
- No modifica estado interno de `sim`

6. Verificación de Compatibilidad

- Chequear coherencia integrador/parámetros
- Ejemplo: `sim.integrator == IAS15`

Lógica Interna y Decisiones

- **Validaciones:**
 - Condicionales según política del sistema
 - Manejo de errores por valores inválidos
- **Compatibilidad:**
 - Verificación implícita de estabilidad numérica

Manejo de Datos Específico

- **Entradas:**
 - Parámetros de configuración temporal/física
 - Instancia REBOUND preconfigurada
- **Intermedios:**
 - Estado modificado de `sim`
 - T_{\max} almacenado para ejecución
- **Salida:**
 - Simulación lista para ejecución
 - T_{\max} disponible como metadato

Salidas Principales

- `sim` configurado con:
 - dt para integración numérica
 - G para interacciones gravitatorias
- T_{\max} como parámetro de control

Interacciones Internas

- **Con API REBOUND:**
 - Modificación directa de propiedades `sim`
 - Consulta de estado del integrador
- **Con flujo de control:**
 - Transmisión de T_{\max} a siguiente etapa

Diagrama del Proceso

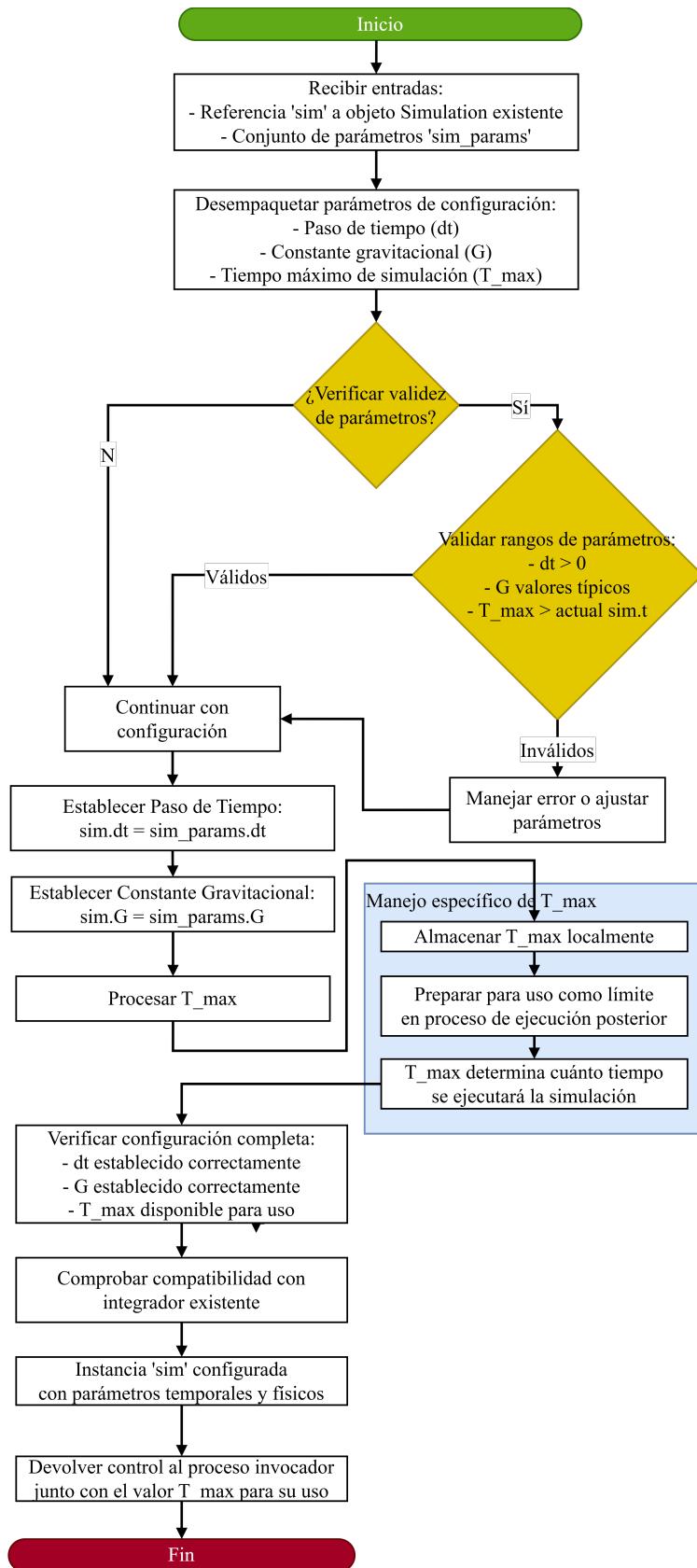


Figura 5.6: Diagrama de Proceso Interno 06: Definir Condiciones

5.1.7. Proceso Interno 07: Iniciar Simulación

Objetivo del Proceso

Ejecutar integración numérica en REBOUND para:

- Generar trayectoria completa del sistema
- Almacenar estados temporales
- Activar recolección de datos en $t_{\text{vis}} \in t_{\text{vis_list}}$
- Preparar datos para cálculo de LE

Entradas Principales

- **sim**: Objeto `Simulation` configurado
- $T_{\text{max}} \in \mathbb{R}^+$ (tiempo máximo)
- $t_{\text{vis_list}} \subset [0, T_{\text{max}}]$ (instantes de visualización)

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 75.7

1. Inicializar Almacenamiento

- Crear estructuras:
 - `time_steps = []`
 - `position_history = []`
 - `velocity_history = []`
- Capturar estado inicial en $t = 0$

2. Bucle Principal de Simulación

Paso 1: Condición: $\text{sim.t} < T_{\text{max}}$

Paso 2: Integración:

```
sim.integrate(sim.t + sim.dt)
```

Paso 3: Almacenamiento:

- Actualizar `time_steps`
- Registrar posiciones/velocidades

Paso 4: Verificación Visualización:

$$\exists t_{\text{vis}} \in t_{\text{vis_list}} \quad | \quad |\text{sim.t} - t_{\text{vis}}| < \epsilon$$

Paso 5: Recolección Condicional:

- Si coincide: Activar módulo de visualización
- Si no: Continuar integración

Lógica Interna y Decisiones

- **Control del bucle:** Condición $\text{sim.t} < T_{\max}$
- **Tolerancia visualización:** $\epsilon = dt/2$
- **Manejo de errores:** Excepciones REBOUND
- **Almacenamiento:** Registro completo vs. muestreo selectivo

Manejo de Datos Específico

- **Entradas:**
 - Estado inicial de `sim`
 - Parámetros temporales
- **Intermedios:**
 - Estructuras de almacenamiento dinámicas
 - Estados intermedios del sistema
- **Salida:**
 - `SimulationResult: {time_steps, position_history, velocity_history}`

Salidas Principales

- **SimulationResult:** Estructura con:
 - Serie temporal completa
 - Datos de posición/velocidad
 - Metadatos de ejecución

Interacciones Internas

- **Con API REBOUND:**
 - Llamadas a `sim.integrate7()`
 - Acceso a `sim.particles`
- **Con módulo de visualización:**
 - Activación en t_{vis}
 - Transferencia de datos parciales
- **Con sistema de almacenamiento:**
 - Gestión eficiente de grandes datasets

Diagrama del Proceso

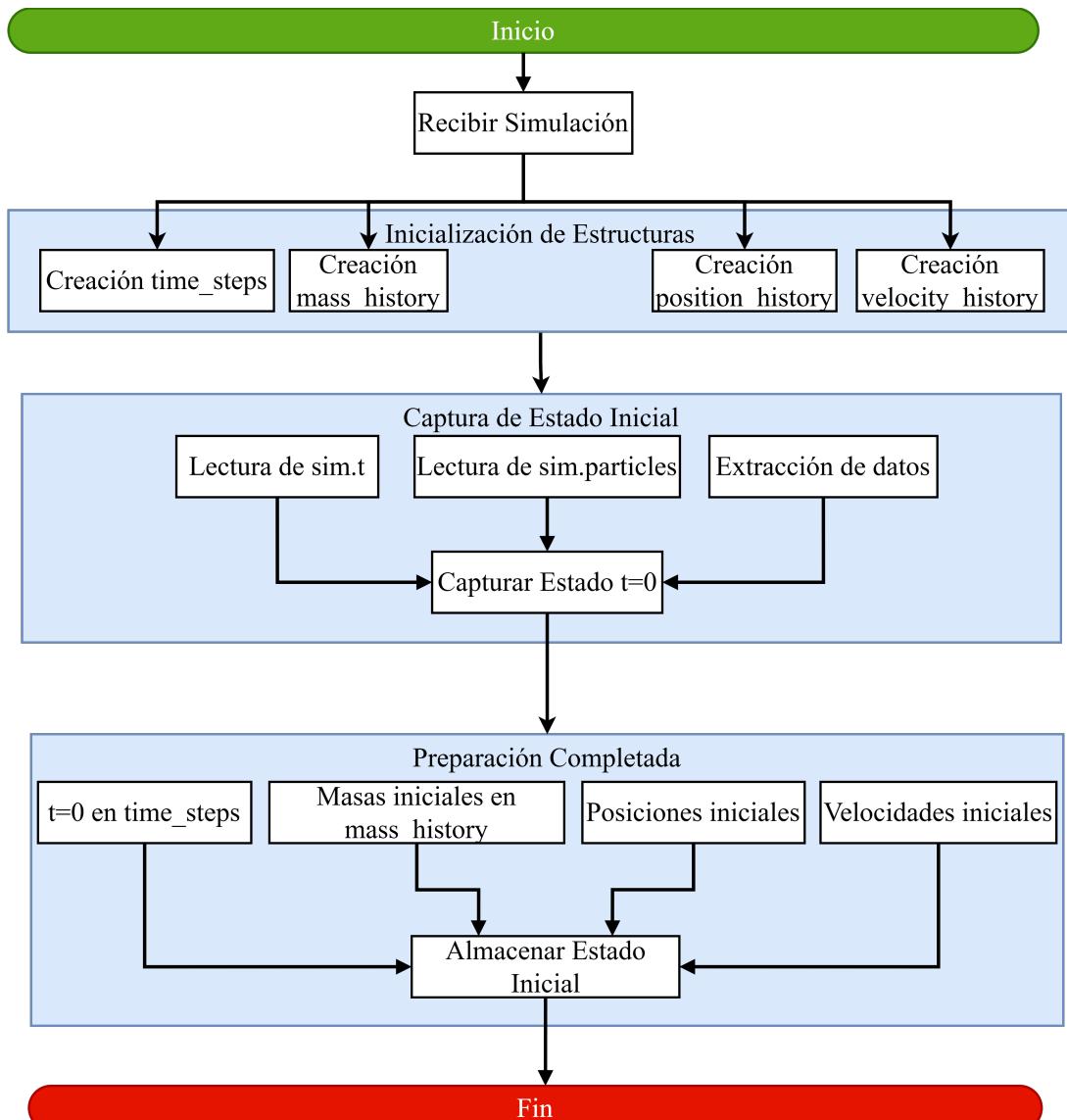


Figura 5.7: Diagrama de Proceso Interno 07: Iniciar Simulación

5.1.8. Proceso Interno 08: Avanzar un Paso de Simulación Usando al Integrador WHFast (Resolver Ecuaciones)

Objetivo del Proceso

Actualizar el estado del sistema gravitacional usando el esquema Drift-Kick-Drift (DKD) de WHFast:

$$H = H_{\text{Kepler}} + H_{\text{Interaction}}$$

Garantizando precisión en el cálculo de posiciones y velocidades para la evaluación del LE.

Entradas Principales

- **sim**: Objeto `Simulation` con:
 - $\mathbf{r}_i(t) = [x_i, y_i, z_i]$
 - $\mathbf{v}_i(t) = [v_{x_i}, v_{y_i}, v_{z_i}]$
 - m_i, dt , sistema coordenadas
- $t_{\text{target}} = t + dt$

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 85.8

1. Preparación de Coordenadas

- Transformación a coordenadas Jacobi si es necesario:

$$\mathbf{r}_{\text{Jacobi}} = T(\mathbf{r}_{\text{inercial}})$$

- Verificación modo seguro

2. Primer Drift ($dt/2$)

- Integración Kepleriana:

$$E - e \sin E = M \quad (\text{Ecuación de Kepler})$$

- Solucionador mejorado (Newton/Laguerre-Conway)
- Actualización posiciones: $\mathbf{r}_i(t + dt/2)$

3. Transformación para Interacción

- Conversión a coordenadas baricéntricas si es necesario

4. Kick (dt)

- Cálculo de aceleraciones:

$$\mathbf{a}_i = G \sum_{j \neq i} \frac{m_j (\mathbf{r}_j - \mathbf{r}_i)}{|\mathbf{r}_j - \mathbf{r}_i|^3}$$

- Actualización velocidades:

$$\mathbf{v}_i(t + dt) = \mathbf{v}_i(t) + \mathbf{a}_i \cdot dt$$

- Kernel modificado (si aplica)

5. Segundo Drift ($dt/2$)

- Integración Kepleriana final
- Posiciones finales: $\mathbf{r}_i(t + dt)$

6. Sincronización Final

- Transformación a coordenadas de reporte
- Actualización tiempo: $\text{sim.t} = t_{\text{target}}$
- Verificaciones modo seguro

Lógica Interna y Decisiones

- **Transformaciones coordenadas:**
 - Optimizadas para precisión numérica
 - Jacobi \leftrightarrow Baricéntricas
- **Modo seguro:** Verificaciones de estabilidad
- **Kernel:** Selección entre precisión/rendimiento

Manejo de Datos Específico

- **Entradas:** Estado del sistema en t
- **Intermedios:**
 - Estados parciales DKD
 - Coordenadas transformadas
- **Salida:** Estado del sistema en $t + dt$

Salidas Principales

- `sim` actualizado con:

- $\mathbf{r}_i(t + dt)$
- $\mathbf{v}_i(t + dt)$
- $t = t + dt$

Interacciones Internas

- **API REBOUND:**

- Módulo WHFast y solucionador Kepler
 - Transformaciones coordenadas
- **Gestor de memoria:** Almacenamiento estados intermedios
 - **Sistema f8sico:** Conservación propiedades dinámicas

Diagrama del Proceso

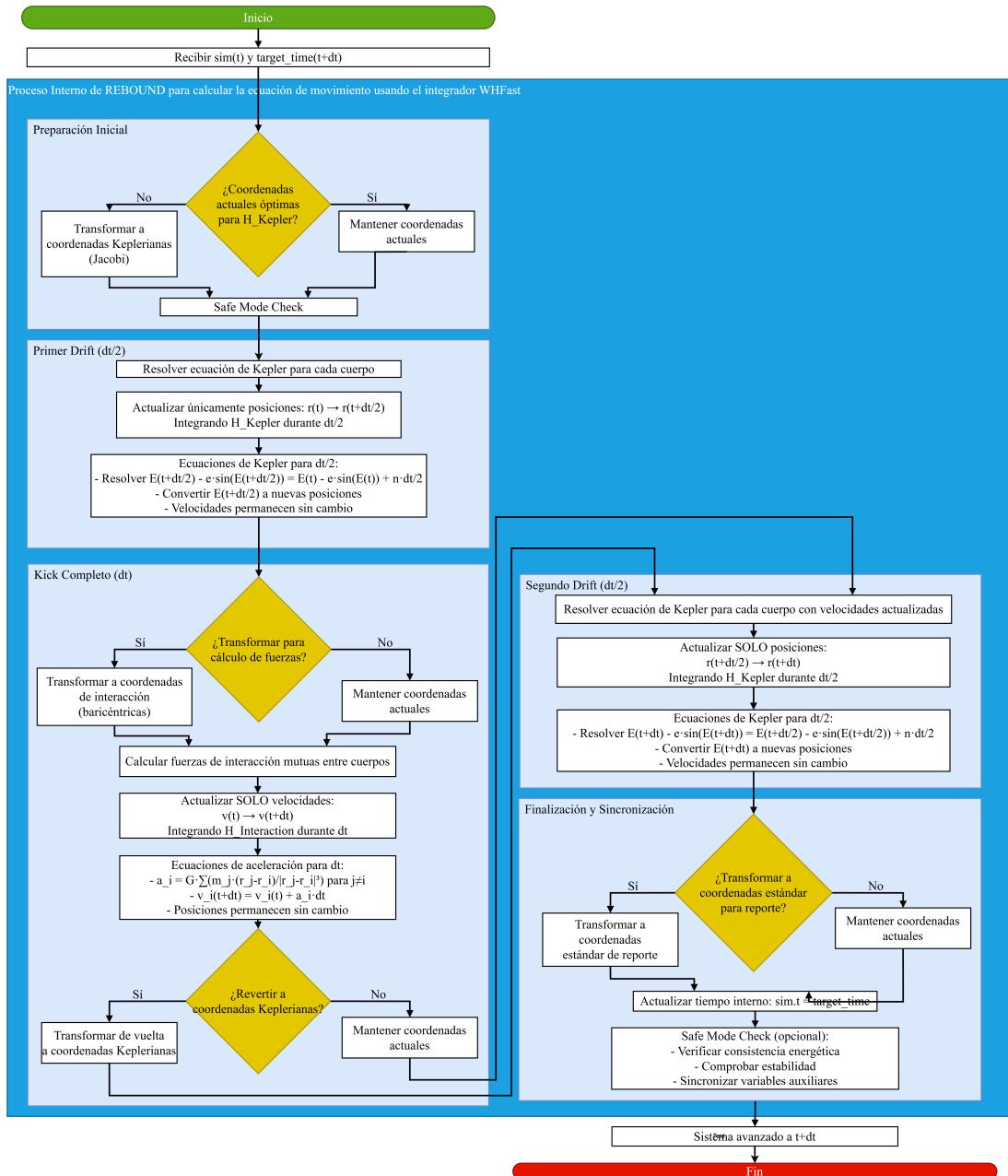


Figura 5.8: Diagrama de Proceso Interno 08: Resolver Ecuaciones

5.1.9. Proceso Interno 09: Actualizar Estado de las Partículas

Objetivo del Proceso

El propósito principal de la actividad “Actualizar Estado de las Partículas” es consolidar y reflejar oficialmente dentro del objeto `Simulation` de REBOUND (`sim`) el nuevo estado físico del sistema gravitacional, incluyendo las posiciones, velocidades, y tiempo de todas las partículas, tras la ejecución de un paso de integración numérica (dt) realizado por el integrador seleccionado (por ejemplo, WHFast). Esta actividad asegura que el objeto `sim` represente de manera coherente el estado del sistema en el nuevo instante de tiempo, preparándolo para las siguientes operaciones en el bucle de simulación, como almacenamiento o visualización.

Entradas Principales

- **Referencia al Objeto `Simulation` (`sim`):**
 - Un puntero o referencia a la instancia del objeto `Simulation` de REBOUND, recibida inmediatamente después de que la función de integración (por ejemplo, `sim.integrate9(t_nuevo)`) haya concluido.
 - Esta instancia contiene implícitamente:
 - Los nuevos valores de posición ($\mathbf{r}_{\text{nuevo}} = [x, y, z]$) y velocidad ($\mathbf{v}_{\text{nuevo}} = [v_x, v_y, v_z]$) de cada partícula, calculados por el integrador y almacenados en `sim.particles`.
 - El nuevo tiempo de simulación (`sim.t = t_{nuevo} = t_{anterior} + dt`), actualizado por el integrador.
 - Las masas de las partículas (m), que permanecen sin cambios en el contexto estándar del proyecto.

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 95.9

1. Finalización del Paso de Integración

- La actividad comienza justo después de que la función de integración de REBOUND (por ejemplo, `sim.integrate9(t_nuevo)`) retorna, habiendo completado el cálculo del nuevo estado del sistema para el tiempo t_{nuevo} .

2. Validar Tiempo de Simulación

- Se verifica que el tiempo interno de la simulación (`sim.t`) sea igual al tiempo objetivo ($t_{\text{nuevo}} = t_{\text{anterior}} + dt$), asegurando que el integrador ha avanzado correctamente el sistema al instante esperado.
- Esta validación confirma la sincronización temporal del estado del sistema.

3. Verificar Actualización del Estado Físico de las Partículas

- **Posiciones:**

- Se confirma que las posiciones de todas las partículas en `sim.particles` han sido actualizadas por el integrador a los nuevos valores ($\mathbf{r}_{\text{nuevo}} = [x, y, z]$), correspondientes al tiempo t_{nuevo} .

- **Velocidades:**

- Se verifica que las velocidades de todas las partículas en `sim.particles` han sido actualizadas a los nuevos valores ($\mathbf{v}_{\text{nuevo}} = [v_x, v_y, v_z]$), calculados por el integrador.

- **Masas:**

- En el contexto de este proyecto, donde no se implementa la modificación dinámica de masas durante la integración estándar, se confirma que las masas de las partículas (m) en `sim.particles` permanecen sin cambios respecto al paso anterior.
- Estas verificaciones aseguran que el estado físico del sistema, almacenado en `sim.particles`, refleja correctamente los resultados del paso de integración.

4. Validación de Coherencia (Opcional)

- **Conservación de Energía:**

- Opcionalmente, se calcula la energía total del sistema (cinética más potencial) en el nuevo estado y se compara con el estado anterior para verificar una conservación aproximada, dentro de los límites de precisión del integrador (por ejemplo, WHFast).

- **Conservación de Momento Angular:**

- Opcionalmente, se calcula el momento angular total y se verifica su conservación aproximada, asegurando que el integrador no ha introducido errores significativos.

- **Coherencia General:**

- Se realiza una comprobación general para asegurar que el estado físico es consistente (por ejemplo, posiciones y velocidades son valores numéricos válidos, no NaN ni infinitos).
- Estas validaciones son opcionales y pueden omitirse en implementaciones estándar si se confía en la robustez del integrador.

5. Actualización del Estado Computacional

- **Variables Auxiliares (Si Existen):**

- Se actualizan cualquier variable auxiliar interna de `sim` que pueda ser utilizada por REBOUND (por ejemplo, acumuladores para métricas de error o estados intermedios del integrador).

- **Sincronización de Coordenadas (Si Aplica):**
 - Si el integrador utiliza múltiples sistemas de coordenadas (por ejemplo, Jacobi para cálculos internos y baricéntricas para reporte), se sincronizan las representaciones para asegurar que el estado reportado en `sim.particles` esté en el sistema esperado por los procesos posteriores.
 - Estas operaciones garantizan que el estado computacional de `sim` esté alineado con el estado f9sico.

6. Estado Coherente del Sistema

- En este punto, el objeto `sim` representa de manera consistente el estado del sistema en el tiempo t_{nuevo} , con `sim.t` actualizado y `sim.particles` conteniendo las nuevas posiciones, velocidades, y masas (sin cambios).

7. Listo para la Siguiente Acción

- La instancia `sim`, con su estado interno completamente actualizado, está preparada para las operaciones siguientes en el bucle principal de simulación, como almacenar el estado, verificar tiempos de visualización, o ejecutar otro paso de integración.

Lógica Interna y Decisiones

- **Validación del Tiempo:**
 - La verificación de que `sim.t = t_nuevo` es una decisión implícita que asegura la sincronización temporal. Si no se cumple, podrá indicar un error en el integrador, pero esto es manejado por REBOUND internamente.
- **Validaciones Opcionales:**
 - La decisión de realizar verificaciones de conservación (energía, momento angular) o coherencia general depende de los requisitos de robustez del sistema. Si se implementan, introducen bifurcaciones donde un fallo (por ejemplo, energía no conservada) podrá desencadenar un manejo de errores, como lanzar una excepción.
- **Sincronización de Coordenadas:**
 - La necesidad de sincronizar sistemas de coordenadas depende de la configuración del integrador y los requisitos de los procesos posteriores. Esta decisión es condicional y se basa en el estado interno de `sim`.
- **Ausencia de Transformación Directa:**
 - No hay transformación de datos en esta actividad, ya que el integrador ya ha actualizado el estado en `sim`. La lógica se centra en verificar y consolidar ese estado, con decisiones orientadas a la validación y sincronización.

Manejo de Datos Específico

- **Datos de Entrada:**

- Referencia al objeto `Simulation (sim)` post-integración, que contiene implícitamente:
 - Nuevo tiempo (`sim.t = t_nuevo`).
 - Nuevas posiciones y velocidades en `sim.particles`.
 - Masas sin cambios en `sim.particles`.

- **Datos Intermedios:**

- Resultados de las verificaciones (por ejemplo, valores de energía o momento angular, si se calculan).
- Estado de variables auxiliares o sistemas de coordenadas sincronizados, si aplica.

- **Datos de Salida:**

- La instancia `sim` con su estado interno consolidado, representando el sistema en t_{nuevo} .

Salidas Principales

- **Instancia `sim` Actualizada:**

- La referencia al objeto `Simulation (sim)`, con su estado interno (tiempo `sim.t`, posiciones y velocidades en `sim.particles`) consolidado y coherente para el tiempo t_{nuevo} , lista para las siguientes acciones en el bucle de simulación.

Interacciones Internas

- **Con la API de REBOUND:**

- Depende directamente de la función de integración (por ejemplo, `sim.integrate()`), que modifica el estado interno de `sim` (`sim.t`, `sim.particles`) antes de que esta actividad comience.
- Accede a las propiedades de `sim` (como `sim.t` y `sim.particles`) para verificar el estado actualizado.

- **Con el Estado Interno de `sim`:**

- Inspecciona y, si es necesario, sincroniza las estructuras internas de `sim`, incluyendo el tiempo, las partículas, y posibles variables auxiliares.

- **Con el Flujo de Simulación:**

- Se integra en el bucle principal de simulación, actuando como el punto donde se consolida el estado tras cada paso de integración, preparando `sim` para procesos como almacenamiento o visualización.

Diagrama del Proceso

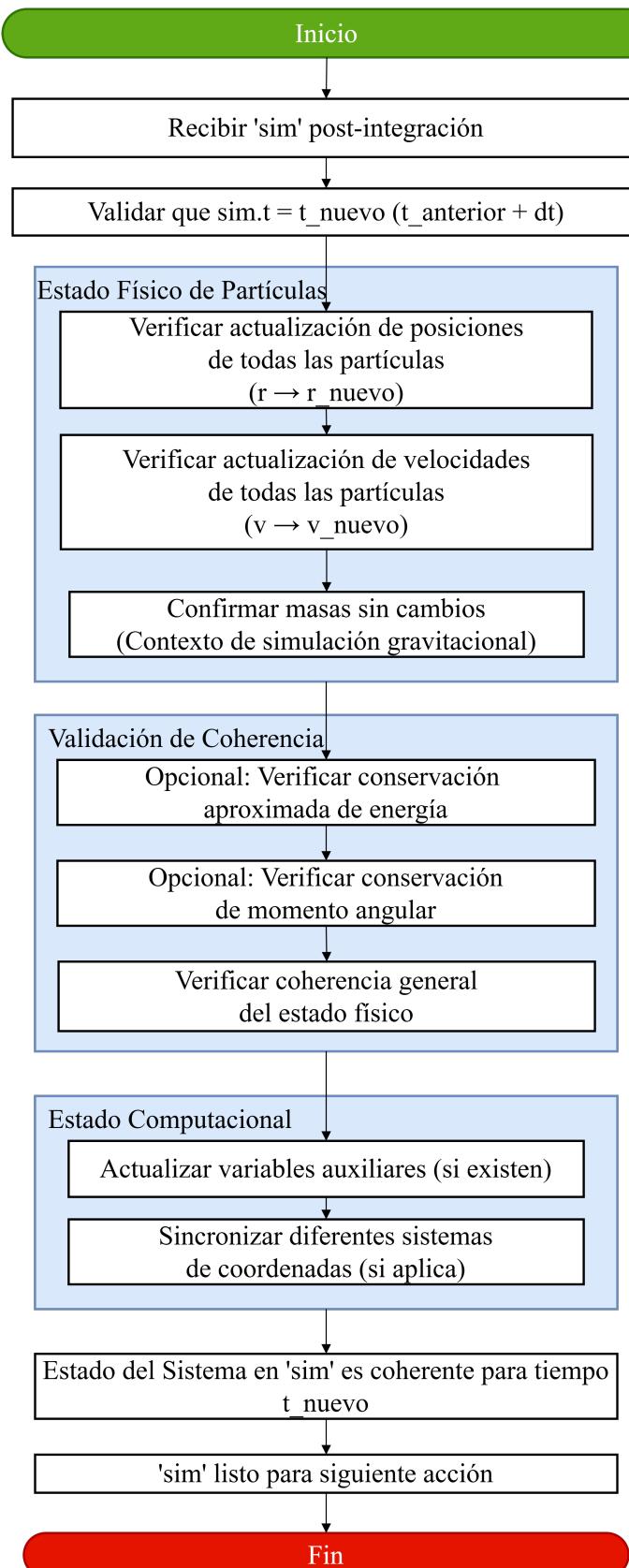


Figura 5.9: Diagrama de Proceso Interno 09: Actualizar Estado

5.1.10. Proceso Interno 10: Recolectar Datos para Visualización

Objetivo del Proceso

El propósito principal de la actividad “Recolectar Datos para Visualización” es extraer el estado fóísico actual del sistema gravitacional (tiempo, masas, posiciones y, opcionalmente, velocidades) desde la instancia de simulación de REBOUND (`sim`) cuando se recibe una señal indicando que el tiempo actual coincide con un instante designado para la visualización (t_{vis}). Los datos recolectados se organizan en una estructura coherente para ser procesados por el siguiente paso del módulo de visualización, facilitando la representación gráfica del sistema en el contexto de la simulación de dos cuerpos.

Entradas Principales

- **Señal de Activación:** Un evento o llamada originada desde el bucle “Iniciar Simulación”, que indica que el tiempo actual (`sim.t`) coincide con un instante de visualización (t_{vis}) de la lista `t_vis_list`.
- **Acceso al Estado Actual:** Una referencia a la instancia del objeto `Simulation` (`sim`) de REBOUND, que contiene el estado actual del sistema, incluyendo:
 - Tiempo actual (`sim.t`).
 - Masas (m), posiciones ($\mathbf{r} = [x, y, z]$), y velocidades ($\mathbf{v} = [v_x, v_y, v_z]$) de todas las partículas en `sim.particles`.
- Alternativamente, una copia o extracto del estado actual (tiempo, masas, posiciones, velocidades) proporcionada directamente por el proceso llamador.

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura05.10

1. Recepción de Activación

- La actividad es invocada por el bucle principal de simulación cuando se detecta que `sim.t` está suficientemente cerca de un tiempo de visualización (t_{vis}).
- Se recibe la referencia al objeto `sim` o los datos del estado actual del sistema, que serán utilizados para la recolección de datos.

2. Lectura del Estado Básico

- **Leer Tiempo Actual:** Se obtiene el valor del tiempo actual t desde `sim.t`, que representa el instante exacto de la simulación en el que se realiza la visualización.

- **Determinar Número de Partículas:** Se lee el número total de partículas en la simulación ($n = \text{sim.N}$), que indica cuántos cuerpos deben procesarse para extraer sus propiedades físicas.

3. Extracción de Datos por Partícula

- **Inicializar Arrays de Almacenamiento:** Se crean arrays o listas temporales para almacenar los datos recolectados:
 - `masas[]`: Para almacenar la masa de cada partícula.
 - `posiciones[][]`: Para almacenar los vectores de posición ($\mathbf{r} = [x, y, z]$) de cada partícula.
 - `velocidades[][]` (opcional): Para almacenar los vectores de velocidad ($\mathbf{v} = [v_x, v_y, v_z]$) si son requeridos por la visualización.
- **Bucle de Extracción:** Se itera sobre las n partículas (i de 0 a $n - 1$):
 - **Leer Masa:** Se extrae la masa de la partícula i (`masas[i] = sim.particles[i].m`).
 - **Leer Posición:** Se extrae el vector de posición de la partícula i (`posiciones[i] = [sim.particles[i].x, sim.particles[i].y, sim.particles[i].z]`).
 - **Verificar Requerimiento de Velocidades:** Se evalúa si la visualización requiere velocidades (por ejemplo, para dibujar vectores de velocidad en la interfaz gráfica).
 - Si se requieren: Se extrae el vector de velocidad de la partícula i (`velocidades[i] = [sim.particles[i].vx, sim.particles[i].vy, sim.particles[i].vz]`).
 - Si no se requieren: Se omite la lectura de velocidades, dejando `velocidades` vacía o sin inicializar.
 - El bucle continúa hasta procesar todas las partículas ($i < n$).

4. Empaqueado de Datos

- **Crear Estructura VisualizationState:** Se inicializa una estructura de datos temporal (por ejemplo, un diccionario o un objeto `VisualizationState`) para agrupar los datos recolectados.
- **Añadir Tiempo:** Se asigna el tiempo actual t a la estructura (`VisState.t = t`).
- **Añadir Masas:** Se asigna el array de masas a la estructura (`VisState.masas = masas[]`).
- **Añadir Posiciones:** Se asigna el array de posiciones a la estructura (`VisState.posiciones = posiciones[][]`).
- **Verificar Velocidades:** Se evalúa si se recolectaron velocidades.

- Si se recolectaron: Se asigna el array de velocidades a la estructura (`VisState.velocidades = velocidades[][]`).
- Si no se recolectaron: Se omite la inclusión de velocidades en la estructura.
- **Verificar Metadatos Adicionales:** Se evalúa si son necesarios metadatos adicionales para la visualización (por ejemplo, nombres de partículas, colores, tamaños u otros parámetros gráficos).
 - Si son necesarios: Se añaden los metadatos a la estructura (por ejemplo, `VisState.metadata = {nombres, colores, tamaños}`). Estos pueden provenir de configuraciones predefinidas o datos asociados a `sim`.
 - Si no son necesarios: Se omite la inclusión de metadatos.
- **Finalizar Empaquetado:** La estructura `VisualizationState` se completa, representando una instantánea coherente del estado del sistema en el tiempo t_{vis} .

5. Pasar Datos a Procesamiento

- La estructura `VisualizationState` se envía al siguiente paso del pipeline de visualización, el proceso “Procesar Datos”, que se encargará de transformar los datos para su renderización en la interfaz gráfica.

Lógica Interna y Decisiones

- **Requerimiento de Velocidades:** La decisión de recolectar velocidades introduce una bifurcación condicional. Si la visualización no requiere velocidades (por ejemplo, si solo se muestran posiciones de partículas), se omite su extracción, optimizando el uso de recursos.
- **Metadatos Adicionales:** La inclusión de metadatos depende de los requisitos específicos del módulo de visualización. Esta decisión permite flexibilidad para soportar diferentes estilos de visualización (por ejemplo, partículas con colores o tamaños personalizados).
- **Bucle de Extracción:** El bucle sobre las partículas es controlado por la condición $i < n$, asegurando que se procesen todas las partículas en la simulación. No hay bifurcaciones dentro del bucle más allá de la decisión sobre velocidades.
- **Manejo de Errores Implícito:** La lectura de datos desde `sim.particles` podría generar errores si el estado de `sim` es inválido (por ejemplo, valores `NAN`). Estos casos son manejados por REBOUND o por validaciones previas en el bucle de simulación, fuera del alcance de esta actividad.

Manejo de Datos Específico

- **Datos de Entrada:**
 - Señal de activación desde el bucle de simulación.

- Referencia a `sim` o extracto del estado actual (tiempo, masas, posiciones, velocidades).
- **Datos Intermedios:**
 - Arrays temporales para almacenar masas (`masas[]`), posiciones (`posiciones[][]`), y opcionalmente velocidades (`velocidades[][]`).
 - Valores individuales extraídos de `sim.particles` ($t, m, x, y, z, v_x, v_y, v_z$).
 - Metadatos adicionales, si se requieren.
- **Datos de Salida:**
 - Estructura `VisualizationState` (o equivalente), que contiene la instantánea del sistema: tiempo (t), masas, posiciones, velocidades (si aplica), y metadatos (si aplica).

Salidas Principales

- **Estructura `VisualizationState`:** Una estructura de datos que encapsula el estado relevante del sistema en el instante de visualización (t_{vis}), incluyendo el tiempo, las masas, las posiciones, y opcionalmente las velocidades y metadatos, lista para ser procesada por el módulo de visualización.

Interacciones Internas

- **Con el Bucle de Simulación:** La actividad es activada por el proceso “Iniciar Simulación” cuando `sim.t` coincide con un tiempo de visualización.
- **Con la API de REBOUND:** Accede al estado interno del objeto `Simulation` (`sim.t`, `sim.N`, `sim.particles`) para extraer los datos físicos del sistema.
- **Con Estructuras de Datos Temporales:** Crea y llena arrays temporales (`masas`, `posiciones`, `velocidades`) y una estructura `VisualizationState` para organizar los datos recolectados.
- **Con el Pipeline de Visualización:** Transfiere la estructura `VisualizationState` al proceso “Procesar Datos”, integrándose en el flujo de renderización gráfica.

Diagrama del Proceso

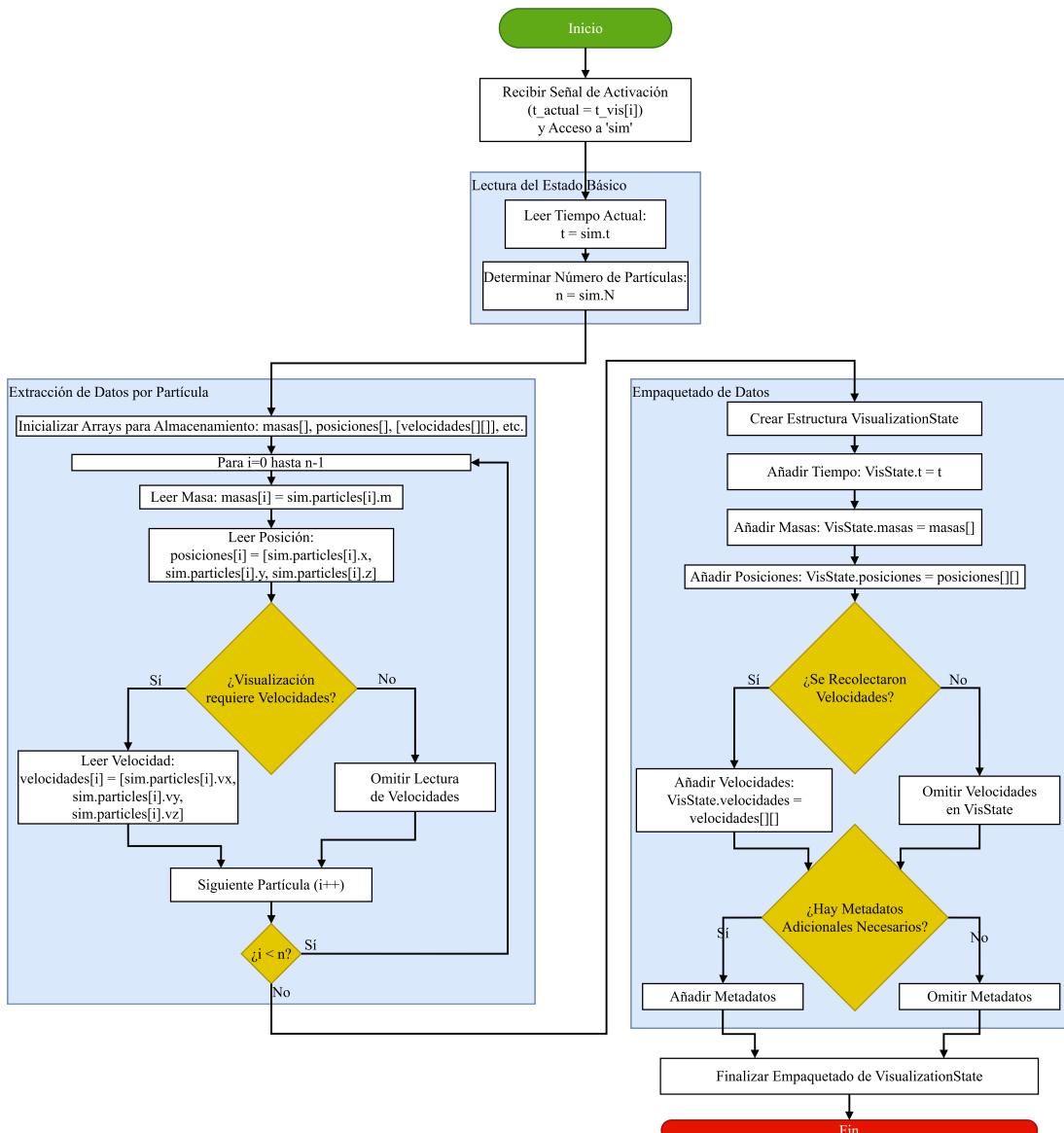


Figura 5.10: Diagrama de Proceso Interno 10: Recolectar Datos

5.1.11. Proceso Interno 11: Procesar Datos para Visualización

Objetivo del Proceso

El propósito principal de la actividad “Procesar Datos para Visualización” es transformar y formatear los datos brutos del estado del sistema gravitacional, recolectados en el proceso “Recolectar Datos”, para que sean compatibles y óptimos para la biblioteca o herramienta gráfica específica utilizada en la visualización. Esta actividad asegura que los datos estén en el formato adecuado y contengan solo la información relevante para generar gráficos como trayectorias 2D/3D o gráficos de energía, soportando la representación visual del comportamiento dinámico en el contexto de la simulación de dos cuerpos.

Entradas Principales

- **Estructura VisualizationState:** Una estructura de datos (por ejemplo, un diccionario o objeto) que contiene la instantánea del sistema en un instante de visualización, incluyendo:
 - Tiempo (t).
 - Masas (`masas[]`).
 - Posiciones (`posiciones[][]`: $\mathbf{r} = [x, y, z]$ por partícula).
 - Velocidades (`velocidades[][]`: $\mathbf{v} = [v_x, v_y, v_z]$ por partícula, si se recolectaron).
 - Metadatos opcionales (por ejemplo, nombres, colores, tamaños).
- **Configuración de Visualización:** Información que define el tipo de gráfico a generar (por ejemplo, trayectoria 2D XY, trayectoria 3D, gráfico de energía vs tiempo) y parámetros de transformación, como:
 - Límites de la ventana gráfica (por ejemplo, $x_{\min}, x_{\max}, y_{\min}, y_{\max}$).
 - Factores de escala (`scale_factor`) y traslación (`offset`).
 - Requerimientos de proyección (por ejemplo, de 3D a 2D).

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura1??

1. Recibir Datos Recolectados

- La actividad recibe la estructura `VisualizationState` con los datos brutos del estado del sistema en el instante de visualización.

2. Verificación de Configuración

- **Leer Configuración de Visualización:** Se accede a la configuración de visualización para determinar:

- El tipo de gráfico (por ejemplo, trayectoria 2D, trayectoria 3D, energía vs. tiempo).
- Parámetros de transformación (escala, traslación, proyección).
- Límites y escalas de la ventana gráfica.
- **Determinar Tipo de Gráfico:** Se identifica el tipo de gráfico especificado:
 - Trayectoria 2D/3D: Configura el procesamiento para extraer coordenadas espaciales.
 - Energía vs. Tiempo: Configura el procesamiento para incluir cálculos de energía.
 - Otro: Configura el procesamiento según los requerimientos específicos del gráfico.

3. Transformación de Coordenadas (Condicional)

- **Verificar Requerimiento de Transformación:** Se evalúa si las coordenadas físicas en `VisualizationState.posiciones` (potencialmente a gran escala) necesitan transformarse para adaptarse a la ventana gráfica o al formato de la biblioteca gráfica.
- **Si se requiere transformación:**
 - **Inicializar Arrays:** Se crean arrays para almacenar las coordenadas transformadas.
 - **Bucle por Partícula:** Para cada partícula:
 - **Escalado:** Se aplica un factor de escala para mapear las coordenadas físicas a la escala gráfica:

$$\mathbf{r}_{\text{scaled}} = (\mathbf{r}_{\text{original}} - \mathbf{r}_{\text{min}}) \times \text{scale_factor}$$

donde \mathbf{r}_{min} y scale_factor provienen de la configuración.

- **Traslación:** Se aplica un desplazamiento para centrar las coordenadas en la ventana gráfica:

$$\mathbf{r}_{\text{transformed}} = \mathbf{r}_{\text{scaled}} + \text{offset}$$

- **Proyección (Si Aplica):** Se evalúa si es necesaria una proyección de 3D a 2D (por ejemplo, para gráficos 2D).

- Si se requiere: Se aplica una matriz de proyección \mathbf{P} :

$$\mathbf{r}_{\text{2D}} = \mathbf{P} \times \mathbf{r}_{\text{transformed}}$$

donde \mathbf{P} es la matriz de proyección definida por la configuración.

- Si no se requiere: Se mantienen las coordenadas transformadas sin proyección.

- El bucle continúa hasta procesar todas las partículas.

- **Si no se requiere transformación:** Se mantienen las coordenadas originales ($\mathbf{r}_{\text{transformed}} = \mathbf{r}_{\text{original}}$).

4. Selección de Datos Relevantes

- **Iniciar Selección:** Se identifican los datos necesarios según el tipo de gráfico especificado.
- **Según Tipo de Gráfico:**
 - Trayectoria 2D: Se extraen las componentes x e y de las posiciones transformadas, almacenándolas en arrays como `x_coords[]` y `y_coords[]`.
 - Trayectoria 3D: Se extraen las componentes x , y , z , almacenándolas en `x_coords[]`, `y_coords[]`, y `z_coords[]`.
 - Energía vs. Tiempo: Se extraen el tiempo (t), masas (m), posiciones (\mathbf{r}), y velocidades (\mathbf{v} , si disponibles) para cálculos posteriores de energía.
 - Otro: Se extraen los componentes relevantes según la configuración específica del gráfico.
- Los datos seleccionados se almacenan para el siguiente paso.

5. Cálculos Derivados (Condicional)

- **Verificar Requerimiento de Cálculos:** Se evalúa si el tipo de gráfico requiere cálculos derivados (por ejemplo, energía o momento angular).
- **Si se requieren cálculos:**
 - **Energía:** Se calculan:
 - Energía cinética:
$$E_k = \sum_i (0.5 \times m_i \times |\mathbf{v}_i|^2)$$
 - Energía potencial:
$$E_p = -G \sum_{i < j} \frac{m_i m_j}{|\mathbf{r}_i - \mathbf{r}_j|}$$
- Energía total:

$$E_{\text{total}} = E_k + E_p$$

- **Momento Angular:** Se calcula el momento angular total:

$$\mathbf{L} = \sum_i m_i (\mathbf{r}_i \times \mathbf{v}_i)$$

- **Otros Cálculos:** Se realizan cálculos específicos según la configuración (por ejemplo, distancias relativas o velocidades relativas).
- **Si no se requieren cálculos:** Se omiten los cálculos derivados, manteniendo solo los datos seleccionados.

6. Formateo para API Gráfica

- **Iniciar Formateo:** Se prepara la conversión de los datos seleccionados y derivados al formato requerido por la biblioteca gráfica (por ejemplo, Matplotlib, Pygame, Plotly).
- **Identificar Formato Requerido:** Se determina el formato esperado por la API:
 - Listas Separadas: Crear arrays individuales por eje (por ejemplo, `x_list = [x1, x2, ...], y_list = [y1, y2, ...]`).
 - Lista de Tuplas: Crear una lista de puntos (por ejemplo, `points = [(x1, y1), (x2, y2), ...]`).
 - Arrays NumPy: Convertir los datos a arrays NumPy (por ejemplo, `X = np.array1([x1, x2, ...]), Y = np.array1([y1, y2, ...])`).
 - Diccionario/Objeto: Crear una estructura con metadatos (por ejemplo, `data = {'x': [x1, x2, ...], 'y': [y1, y2, ...], 'tipo': 'scatter'}`).
- Los datos se convierten al formato seleccionado, incluyendo cualquier valor derivado (como energías).

7. Finalizar Estructura de Datos

- Se completa la estructura de datos formateada, que contiene los datos transformados, seleccionados, y posiblemente derivados, en el formato exacto requerido por la biblioteca gráfica.

8. Pasar Datos Formateados

- La estructura de datos formateada se envía al proceso “Generar Gráficos” del pipeline de visualización, donde se utilizará para renderizar la representación gráfica.

Lógica Interna y Decisiones

- **Tipo de Gráfico:** La selección del tipo de gráfico (trayectoria 2D, 3D, energía, otro) determina las ramas de procesamiento, afectando la selección de datos y los cálculos derivados.
- **Transformación de Coordenadas:** La decisión de transformar coordenadas depende de la configuración de visualización. Si es necesaria, se aplican escalado, traslación, y proyección condicionalmente (por ejemplo, proyección 3D a 2D solo para gráficos 2D).
- **Cálculos Derivados:** La necesidad de cálculos como energía o momento angular introduce una bifurcación. Gráficos como trayectorias omiten estos cálculos, mientras que gráficos de energía los requieren.

- **Formato de la API:** La elección del formato (listas, tuplas, NumPy, diccionario) depende de la biblioteca gráfica, permitiendo flexibilidad para diferentes herramientas.
- **Manejo de Errores Implícito:** Errores como datos inválidos (por ejemplo, divisiones por cero en el cálculo de energía potencial) son manejados por validaciones previas o excepciones en la biblioteca gráfica, fuera del alcance directo de esta actividad.

Manejo de Datos Específico

- **Datos de Entrada:**
 - Estructura `VisualizationState` con tiempo, masas, posiciones, velocidades (si aplica), y metadatos.
 - Configuración de visualización (tipo de gráfico, parámetros de transformación).
- **Datos Intermedios:**
 - Coordenadas transformadas (escaladas, trasladadas, proyectadas: $\mathbf{r}_{\text{transformed}}$, \mathbf{r}_{2D}).
 - Subconjuntos de datos seleccionados (por ejemplo, `x_coords`, `y_coords`).
 - Valores derivados (por ejemplo, energía cinética E_k , potencial E_p , total E_{total} , momento angular \mathbf{L}).
- **Datos de Salida:**
 - Estructura de datos formateada (por ejemplo, `x_list`, `y_list`, arrays NumPy, o diccionario) específica para la API de la biblioteca gráfica.

Salidas Principales

- **Datos Formateados para Visualización:** Una estructura de datos que contiene los datos del estado del sistema, seleccionados, transformados, y formateados según los requisitos de la biblioteca gráfica, lista para ser utilizada directamente en el proceso “Generar Gráficos”.

Interacciones Internas

- **Con el Proceso Anterior:** Recibe la estructura `VisualizationState` del proceso “Recolectar Datos”.
- **Con la Configuración de Visualización:** Lee los parámetros de configuración para determinar el tipo de gráfico, transformaciones, y formato de salida.
- **Con Cálculos Matemáticos:** Realiza operaciones como escalado, traslación, proyección, y cálculos de energía o momento angular, según sea necesario.

- **Con el Pipeline de Visualización:** Transfiere los datos formateados al proceso “Generar Gráficos”, integrándose en el flujo de renderización gráfica.

Diagrama del Proceso

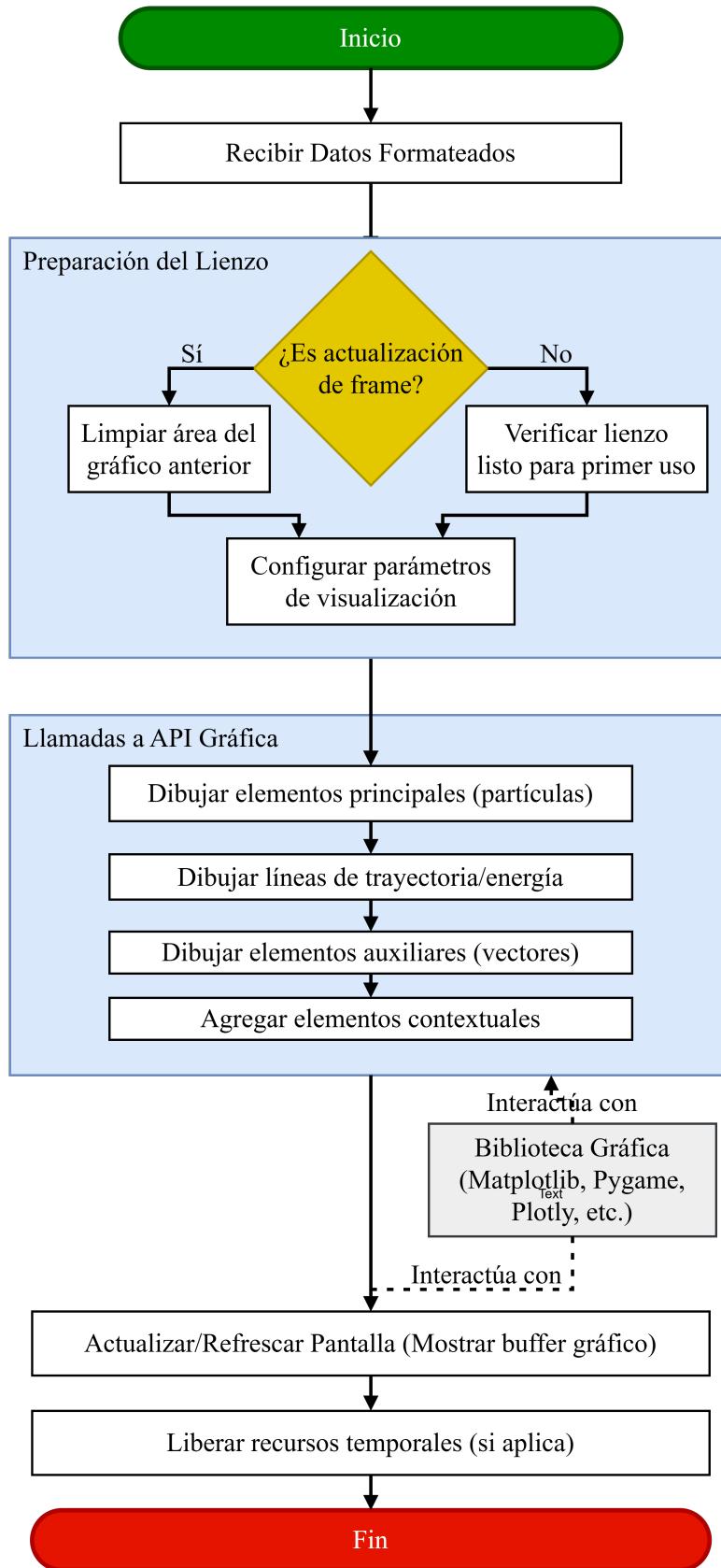


Figura 5.11: Diagrama de Proceso Interno 12: Generar Gráfico

5.1.12. Proceso Interno 12: Generar Gráficos

Objetivo del Proceso

El propósito principal de la actividad “Generar Gráficos” es utilizar los datos preprocesados y formateados del estado del sistema gravitacional para crear o actualizar una representación visual en la pantalla, empleando las funciones específicas de la biblioteca gráfica seleccionada (por ejemplo, `Matplotlib`, `Pygame`, `Plotly`). Esta actividad asegura que el estado del sistema, como las posiciones de las partículas o los valores de energía, se visualice de manera clara y precisa para el usuario, soportando el análisis y monitoreo de la simulación en el contexto del proyecto.

Entradas Principales

- **Datos Formateados:** Una estructura de datos generada por el proceso “Procesar Datos”, que contiene información lista para graficar, como:
 - Coordenadas para trayectorias (por ejemplo, `x_list`, `y_list` para 2D, o `x_list`, `y_list`, `z_list` para 3D).
 - Valores escalares para gráficos de energía (por ejemplo, `[t, E_total]`).
 - Configuración visual adicional (por ejemplo, colores, tamaños, tipos de gráfico).
- **Contexto Gráfico:** Acceso al lienzo, ventana, o superficie de dibujo proporcionada por la biblioteca gráfica, que sirve como el área donde se renderizará la visualización.

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 25.11

Recibir Datos Formateados

- La actividad recibe la estructura de datos formateada, que contiene las coordenadas, valores derivados (como energía), y parámetros visuales necesarios para generar el gráfico.

Preparar Lienzo

- **Verificar si es Actualización de Frame:**
 - Se evalúa si el gráfico es una actualización de un frame anterior (como en una animación) o el primer dibujo en el lienzo.
- **Si es una actualización:**
 - Se limpia el área del gráfico anterior para evitar superposiciones. Esto puede implicar:
 - Rellenar el fondo con un color sólido (por ejemplo, blanco o negro).

- Eliminar objetos gráficos previos (por ejemplo, puntos o líneas) del buffer.
- Si es la primera vez o no es animación:
 - Se verifica que el lienzo esté inicializado y listo para dibujar, sin necesidad de limpieza.
 - Esta preparación asegura que el lienzo esté en un estado adecuado para el nuevo dibujo.

Configurar Parámetros de Visualización

- Se establecen los parámetros gráficos definidos en la configuración, como el estilo de los puntos, colores, tamaños, o propiedades del gráfico (por ejemplo, escala de ejes, títulos).
- Estos parámetros se pasan a las funciones de dibujo de la biblioteca gráfica.

Llamar Funciones de Dibujo de la API Gráfica

- Dibujar Elementos Principales (Partículas):
 - Se utilizan los datos formateados para dibujar los elementos primarios, como las partículas. Por ejemplo:
 - Para una trayectoria 2D, se llama a funciones como `draw_circle2()` (Pygame) o `scatter2()` (Matplotlib) para cada partícula, usando coordenadas `x_list` y `y_list`.
 - El tamaño o color de los puntos puede reflejar la masa de las partículas, según la configuración.
- Dibujar Líneas de Trayectoria o Energía:
 - Se dibujan líneas o curvas que conectan puntos para representar trayectorias o series temporales. Por ejemplo:
 - Para trayectorias, se usa `plot2()` o `draw_line2()` para conectar posiciones consecutivas.
 - Para gráficos de energía, se usa `plot_line2()` para añadir un punto (`[t, E_total]`) a la serie temporal.
- Dibujar Elementos Auxiliares (Vectores):
 - Si los datos incluyen velocidades y la configuración lo requiere, se dibujan vectores de velocidad usando funciones como `draw_line2()` o `arrow2()` para cada partícula.
- Agregar Elementos Contextuales:
 - Se añaden elementos gráficos como ejes, etiquetas, títulos, y leyendas, utilizando funciones como `set_xlabel2()`, `set_title2()`, o `add_legend2()` en la API gráfica.

- Cada llamada a la API utiliza los datos formateados como argumentos, asegurando que la representación visual sea precisa.

Actualizar/Refrescar Pantalla

- Una vez que todos los elementos gráficos han sido dibujados en el buffer gráfico, se llama a la función de la biblioteca gráfica que hace visible el buffer en la pantalla. Por ejemplo:
 - `pygame.display.flip()` en Pygame.
 - `plt.draw()` o `fig.show()` en Matplotlib.
 - `plotly.render()` en Plotly.
- Esto asegura que el usuario vea la representación gráfica actualizada.

Liberar Recursos Temporales (Si Aplica)

- Se liberan recursos temporales utilizados durante el dibujo, como buffers intermedios o estructuras de datos auxiliares, para optimizar el uso de memoria.
- En la mayoría de los casos, las bibliotecas gráficas manejan esta limpieza automáticamente.

Lógica Interna y Decisiones

- **Actualización vs Primer Dibujo:** La decisión de limpiar el lienzo depende de si el gráfico es una actualización en una animación o un dibujo inicial. Esta bifurcación asegura que las animaciones no acumulen artefactos visuales de frames anteriores.
- **Tipo de Elementos Gráficos:** La selección de funciones de dibujo (por ejemplo, `scatter` para partículas, `plot` para líneas) depende implícitamente del tipo de gráfico especificado en los datos formateados (trayectoria, energía, etc.).
- **Elementos Auxiliares:** La inclusión de vectores de velocidad, etiquetas, o leyendas es condicional, basada en la configuración visual y la disponibilidad de datos (por ejemplo, velocidades).
- **Manejo de Errores Implícito:** Errores como formatos de datos incompatibles con la API gráfica o fallos en el lienzo son manejados por la biblioteca gráfica, que puede lanzar excepciones. Esta actividad asume que los datos formateados son válidos, gracias al procesamiento previo.

Manejo de Datos Específico

- **Datos de Entrada:**
 - Estructura de datos formateada (por ejemplo, `x_list`, `y_list`, `[t, E_total]`) con coordenadas, valores escalares, y parámetros visuales.

- Contexto gráfico (lienzo o ventana de la biblioteca gráfica).
- **Datos Intermedios:**
 - Ningún dato numérico nuevo se genera; los datos formateados se consumen directamente como parámetros para las funciones de dibujo.
- **Datos de Salida:**
 - No se generan datos numéricos como salida; el resultado es la modificación del buffer gráfico, que produce una representación visual en la pantalla.

Salidas Principales

- **Representación Gráfica Actualizada:** La visualización del estado del sistema (por ejemplo, posiciones de partículas, trayectorias, o gráficos de energía) renderizada y visible en la interfaz gráfica, proporcionando al usuario una representación clara del comportamiento dinámico.

Interacciones Internas

- **Con el Proceso Anterior:** Recibe los datos formateados del proceso “Procesar Datos”.
- **Con la Biblioteca Gráfica:** Interactúa extensamente con la API de la biblioteca gráfica (por ejemplo, `Matplotlib`, `Pygame`, `Plotly`) para dibujar elementos y actualizar la pantalla.
- **Con el Lienzo Gráfico:** Modifica el estado del lienzo o ventana gráfica, ya sea limpiándolo, dibujando nuevos elementos, o refrescando el buffer para mostrar la visualización.
- **Con el Pipeline de Visualización:** Completa el flujo de visualización, transformando los datos formateados en una salida gráfica que el usuario puede observar.

Diagrama del Proceso

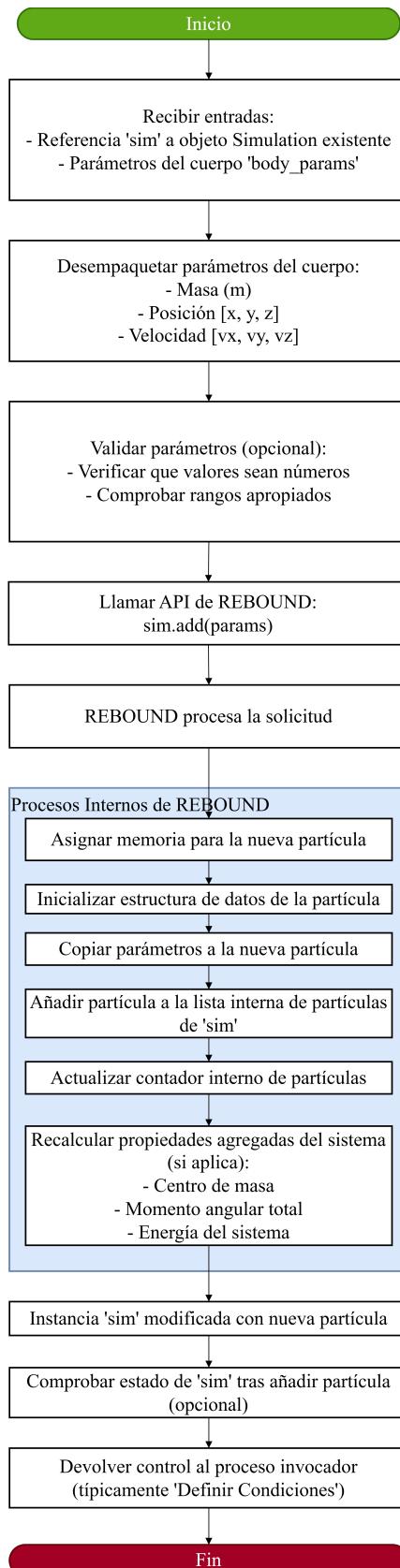


Figura 5.12: Diagrama de Proceso Interno 05: Agregar Cuerpos

5.1.13. Proceso Interno 13: Evaluar Fitness

Objetivo del Proceso

El propósito principal es calcular el *fitness penalizado* $F_p(x) = f(x) + \lambda_p P(x)$, donde:

- $f(x)$: Exponente de Lyapunov (*LE*)
- $P(x) = \sum \max(g_k(x), 0)^2 + \sum |h_k(x)|^2$
- λ_p : Parámetro de penalización

Entradas Principales

- **Parámetros x :** Vector en \mathbb{R}^n (ej. $[m_1, m_2]$)
- **Datos REBOUND:** $\{\text{pos}(t), \text{vel}(t), \text{mass}(t)\}_{t=0}^{T_{\max}}$
- **Restricciones:**
 - $g_k(x) \leq 0$ (desigualdad)
 - $h_k(x) = 0$ (igualdad)
- $\lambda_p \in \mathbb{R}^+$

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 35.13

1. Recibir Resultados de Simulación

- Validar integridad de datos y T_{\max}

2. Calcular $f(x)$

- Cálculo de *LE*:

$$f(x) = \begin{cases} LE_{\text{calc}} & \text{si converge} \\ \text{INF} & \text{si falla} \end{cases}$$

3. Calcular $P(x)$

$$\begin{aligned} P(x) &= \sum_{k=1}^K \max(g_k(x), 0)^2 + \sum_{j=1}^J |h_j(x)|^2 \\ &+ \begin{cases} \text{PENALTY_LARGE} & \text{si simulación inviable} \end{cases} \end{aligned}$$

4. Calcular $F_p(x)$

$$F_p(x) = f(x) + \lambda_p \cdot P(x)$$

- Verificar $F_p \in \mathbb{R}$

Lógica Interna y Decisiones

- **Convergencia LE:**
 - Fallo $\Rightarrow f(x) \leftarrow \text{INF}$
- **Violaciones:**
 - g_k : $\max(g_k(x), 0) \Rightarrow$ solo violaciones positivas
 - h_k : L^2 -norm del desvío
- **Penalización catastrófica:**
 - Activa para simulaciones físicamente inviables

Manejo de Datos Específico

- **Entradas:**
 - $x \in [\min_i, \max_i]^n$
 - $\lambda_p > 0$
- **Intermedios:**
 - $LE_{\text{calc}} \in \mathbb{R}$
 - violaciones $\in \mathbb{R}^+$
- **Salida:**
 - $F_p(x) \in \mathbb{R}$

Salidas Principales

- **Fitness penalizado:** $F_p(x)$ ordenable para selección

Interacciones Internas

- **REBOUND:** Provee $\{\text{pos}(t), \text{vel}(t)\}$
- **Módulo LE:** Calcula $f(x)$
- **Gestor de restricciones:** Evalúa $g_k(x), h_j(x)$

Diagrama del Proceso

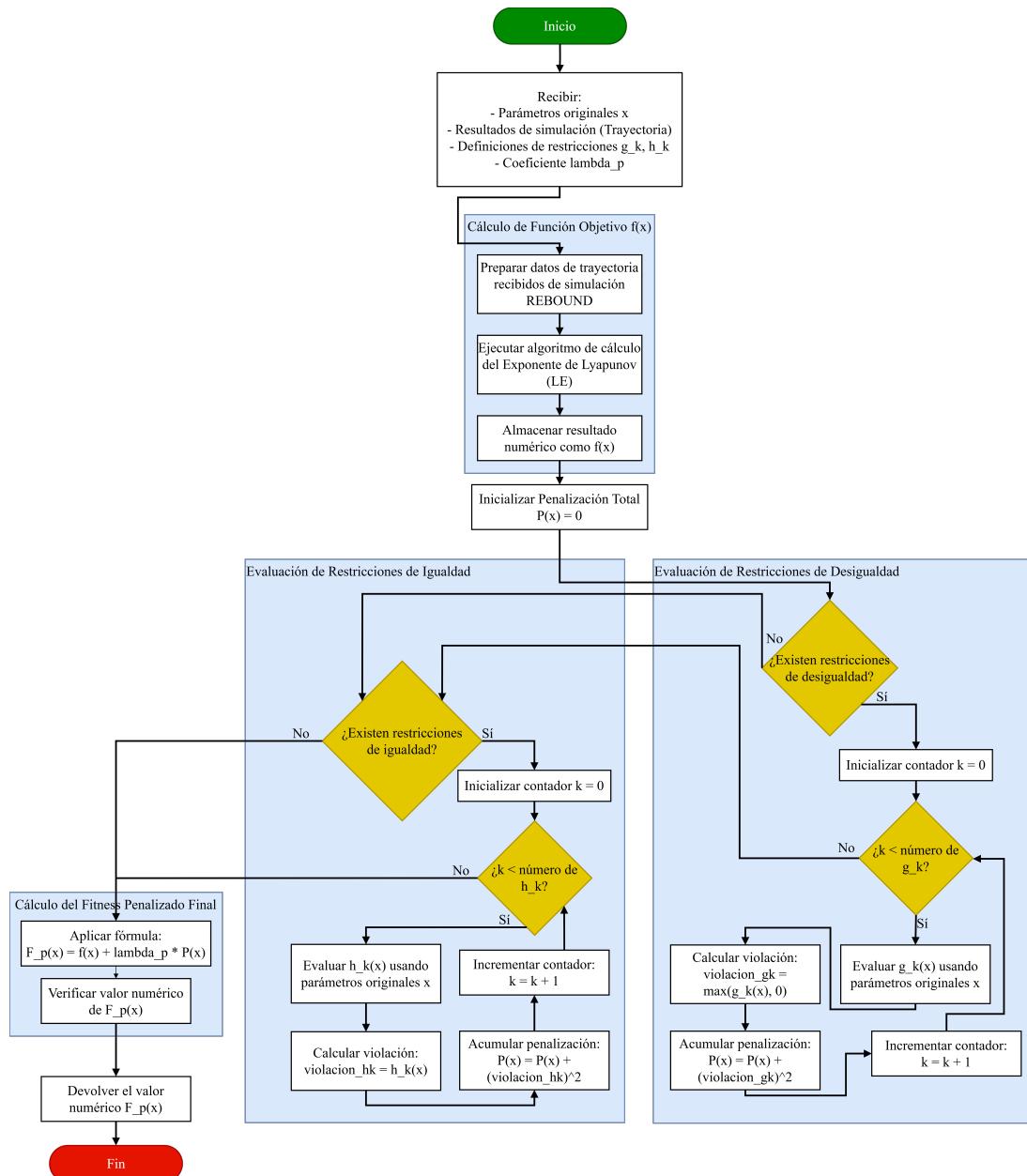


Figura 5.13: Diagrama de Proceso Interno 13: Calcular Fitness

5.1.14. Proceso Interno 14: Formar Siguiente Población

Objetivo del Proceso

El propósito principal es construir la población para la próxima iteración del algoritmo genético, combinando:

- Los k mejores individuos (élite) de `Poblacion_Actual`
- Los $(N - k)$ mejores descendientes

Manteniendo el tamaño poblacional N .

Entradas Principales

- **Población Actual:**
 - N individuos $\{x_i, F_p^i\}_{i=1}^N$
- **Descendencia:**
 - N individuos nuevos $\{x'_j, F_p^j\}_{j=1}^N$
- **Parámetros:**
 - $k \in \mathbb{N}$ (tamaño de élite)
 - $N \in \mathbb{N}$ (tamaño población)

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 45.14

1. Ordenar Poblaciones

- `Poblacion_Actual` \leftarrow sort_asc($\{F_p^i\}$)
- `Descendencia` \leftarrow sort_asc($\{F_p^j\}$)

2. Inicializar Población Siguiente

- Crear `Poblacion_Siguiente` = \emptyset

3. Aplicar Elitismo

$$\text{Poblacion_Siguiente} \leftarrow \begin{cases} \text{primeros } k \text{ de Poblacion_Actual} & \text{si } k > 0 \\ \emptyset & \text{si } k = 0 \end{cases}$$

4. Completar con Descendencia

$$\text{num_a_copiar} = N - |\text{Poblacion_Siguiente}|$$

$$\text{Poblacion_Siguiente} \leftarrow \text{Poblacion_Siguiente} \cup \{\text{primeros num_a_copiar de Descendencia}\}$$

5. Verificación Final

- Asegurar $|\text{Poblacion_Siguiente}| = N$
- Manejo de errores (relleno adicional si es necesario)

Lógica Interna y Decisiones

- **Elitismo condicional:**
 - $k > 0 \Rightarrow$ preservar élite
 - $k = 0 \Rightarrow$ reemplazo completo
- **Selección por fitness:**
 - Ordenamiento basado en F_p ascendente
 - $\text{num_a_copiar} = N - k$ garantiza tamaño poblacional

Manejo de Datos Específico

- **Entradas:**
 - Dos poblaciones ordenables por F_p
 - Parámetros k, N de configuración
- **Intermedios:**
 - Poblaciones ordenadas
 - Contador num_a_copiar
- **Salida:**
 - $\text{Poblacion_Siguiente}$ con N individuos

Salidas Principales

- **Población siguiente:**
 - $\{\text{élite}\} \cup \{\text{mejores descendientes}\}$
 - Lista para próxima iteración del algoritmo

Interacciones Internas

- **Con módulo de ordenación:** Clasificación por F_p
- **Con gestor de poblaciones:** Fusión controlada de individuos
- **Con ConfigurationData:** Lectura de k y N

Diagrama del Proceso

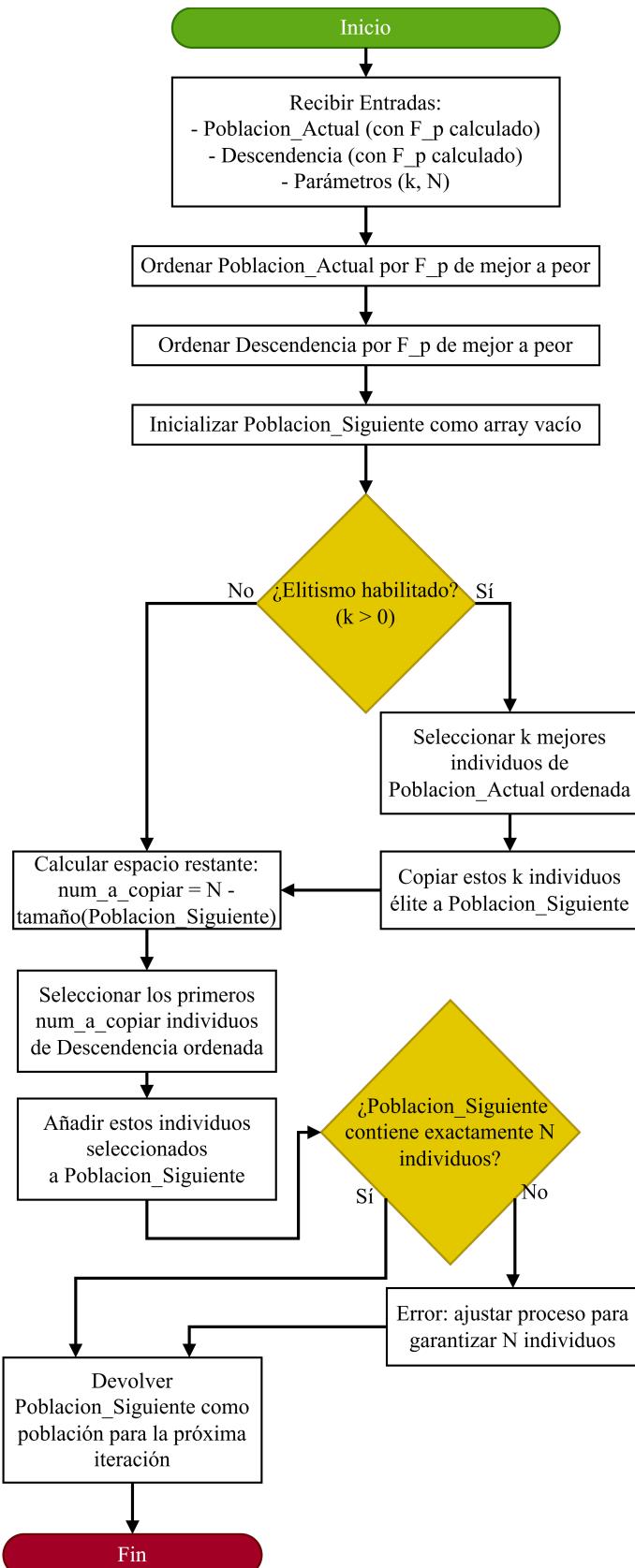


Figura 5.14: Diagrama de Proceso Interno 14: Selección de Individuos

5.1.15. Proceso Interno 15: Mostrar Resultados

Objetivo de la Actividad

El propósito principal de la actividad “**Mostrar Resultados**” es presentar de manera clara y comprensible al usuario final, a través de la interfaz gráfica (UI), la solución óptima encontrada por el Algoritmo Bioinspirado al concluir el proceso de optimización. Esto incluye el mejor conjunto de parámetros (ej. masas finales optimizadas) y su correspondiente valor de fitness, definido como el Exponente de Lyapunov (*LE*) minimizado. Opcionalmente, puede incluir una visualización final estática de la trayectoria asociada a la solución óptima.

Entradas Principales

- **Señal de finalización:** Evento que indica término del proceso de optimización
- **Parámetros óptimos:** Valores numéricos finales (ej. masas de cuerpos)
- **Valor de fitness:** *LE* asociado a la solución óptima
- **Datos de trayectoria (opcional):** Posiciones/velocidades de la simulación REBOUND
- **Resumen de optimización (opcional):** Detalles del proceso de optimización

Sub-pasos Secuenciales

Este apartado es proporcionado para profundizar y describir de forma textual cada paso contenido dentro del diagrama del proceso descrito en la figura 5.15.

1. Recibir Señal de Finalización

- UI detecta notificación de término del Algoritmo Bioinspirado

2. Recibir Datos Finales

- Datos recibidos incluyen:
 - Parámetros óptimos (valores numéricos)
 - *LE* final
 - Bandera de existencia de datos de trayectoria
 - Resumen de optimización (si está disponible)

3. Formatear Datos para Presentación

- Conversión a formatos legibles:
 - Parámetros: valor → cadena (ej. ‘‘0.1234’’)
 - *LE* → ‘‘LE: -0.567’’

- Generación de etiquetas descriptivas (ej. ‘‘Masa Cuerpo 1 Final:’’)
- Resumen → texto descriptivo (ej. ‘‘50 generaciones’’)

4. Manejar Visualización Final (Condicional)

Condición 1: Verificar existencia de datos de trayectoria

Condición 2: Confirmar configuración de visualización habilitada

Si ambas condiciones se cumplen:

- Preparación de datos:
 - Extracción de posiciones/velocidades
 - Determinación de escalas (ej. rango de coordenadas)
 - Generación de metadatos (etiquetas ejes, título)
- Llamado al Módulo de Visualización
- Integración del gráfico en UI

Si falla alguna condición:

- Mostrar solo resultados textuales

5. Presentar Resultados en Pantalla

- Actualización de elementos UI:
 - Parámetros óptimos formateados con etiquetas
 - *LE* con interpretación (ej. ‘‘indica estabilidad’’)
 - Resumen de optimización (si existe)
 - Gráfico estático (si se generó)
- Habilitación de controles:
 - Botones Guardar, Reiniciar, Cerrar

6. Esperar Interacción del Usuario

- UI permanece en estado receptivo para acciones posteriores

Lógica Interna y Decisiones

- **Decisión clave:** Generar visualización gráfica
 - Depende de:
 - 5(a) Datos de trayectoria disponibles
 - 5(b) Configuración de visualización activada
 - Manejo de opcionales:

- Omisión automática de secciones sin datos

Manejo de Datos Específico

- Entradas:

- Parámetros: `float/double`
- LE : `float/double`
- Trayectoria: `arrays/listas` (opcional)

- Intermedios:

- Cadenas formateadas
- Datos gráficos procesados

- Salidas:

- Texto UI formateado
- Gráfico estático (opcional)

Salidas Principales

- UI actualizada con:

- Resultados numéricos y textuales
- Visualización gráfica (condicional)
- Controles interactivos habilitados

Interacciones Internas

- Con Algoritmo Bioinspirado:

- Recepción asincrónica de datos finales

- Con UI:

- Actualización dinámica de componentes visuales

- Con Módulo de Visualización:

- Integración gráfica bajo demanda

Diagrama del Proceso

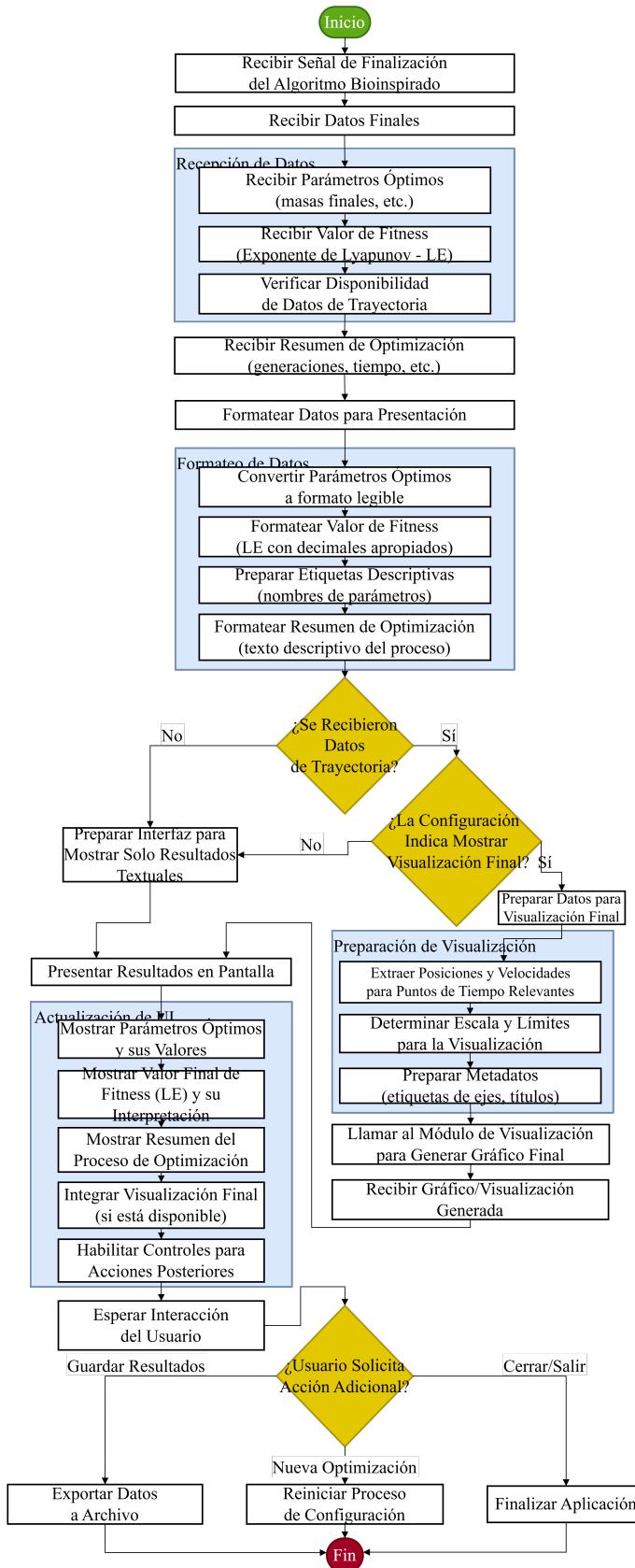


Figura 5.15: Diagrama de Proceso Interno 15: Mostrar Resultados

5.1.16. Matriz de Procesos

La Matriz de Resumen de Procesos presentada en esta sección ofrece un resumen consolidado y estructurado de los procesos clave del proyecto “Modelo para representar comportamientos gravitacionales con dos cuerpos”. Cada fila de la tabla describe un proceso específico, incluyendo su nombre, objetivo, los actores o componentes involucrados, el evento que lo inicia, las entradas y salidas principales, y un resumen de los pasos esenciales. Esta disposición permite a los usuarios obtener una visión clara y rápida del propósito y funcionamiento de cada proceso dentro del sistema, facilitando la comprensión y el análisis sin necesidad de consultar descripciones más extensas.

Los procesos detallados en la tabla son fundamentales para el desarrollo del proyecto, cubriendo desde la captura de parámetros ingresados por el usuario hasta la creación de gráficos que ilustran los comportamientos gravitacionales simulados. Estos procesos trabajan en conjunto para alcanzar el objetivo principal del proyecto: minimizar el Exponente de Lyapunov (LE) mediante algoritmos bioinspirados, apoyándose en herramientas como la biblioteca REBOUND para simulaciones y métodos de optimización como FMM/Barnes-Hut.

Esta matriz está diseñada para ser una herramienta práctica tanto para analistas de sistemas como para diseñadores de software y otros stakeholders involucrados. Su estructura, con columnas específicas como “*Trigger* (Evento)” y “Pasos Clave”, destaca las dependencias y el flujo de datos entre los procesos, ofreciendo una perspectiva integral del sistema. En esencia, la tabla no solo actúa como una referencia rápida, sino que también ilustra cómo los distintos elementos del proyecto —interfaz, simulación, optimización y visualización— se integran para cumplir con los objetivos establecidos.

Tabla 5.1: Matriz de Procesos

Nombre del Proceso	Objetivo	Actores Involucrados	Inicio (Evento)	Entradas Principales	Salidas Principales	Pasos Clave
Captura Parámetros	Recopilar, validar y almacenar parámetros de configuración del usuario.	UI, Módulo de Validación	Usuario interactúa con la UI para configurar.	Evento de inicio, interacciones del usuario (clicks, selecciones, texto).	Estructura ConfigurationData validada, estado de UI actualizado.	<ol style="list-style-type: none"> Presentar formulario en UI. Recibir y validar entradas. Almacenar configuración validada. Habilitar control de inicio.
Mostrar Resultados	Presentar solución óptima y visualización final al usuario.	UI, Módulo de Visualización, Algoritmo Bioinspirado	Señal de finalización del Algoritmo Bioinspirado.	Parámetros óptimos, valor de fitness, datos de trayectoria (opcional).	Actualización visual de la UI con resultados finales.	<ol style="list-style-type: none"> Recibir datos finales. Formatear datos para presentación. (Condicional) Generar visualización final. Presentar resultados en UI.
Inicializar Población	Generar conjunto inicial de N individuos dentro de rangos válidos.	Algoritmo Bioinspirado, Generador Aleatorio	Inicio del proceso de optimización.	Tamaño de población (N), rangos de parámetros.	Lista de N individuos (conjuntos de parámetros).	<ol style="list-style-type: none"> Leer configuración (N, rangos). Generar parámetros dentro de rangos. Crear lista de población inicial.

Continúa en la siguiente página

Tabla 5.1 – continuación de la página anterior

Nombre del Proceso	Objetivo	Actores Involucrados	Inicio (Evento)	Entradas Principales	Salidas Principales	Principales Pasos Clave
Generar Nueva Generación	Crear nueva población de N individuos usando operadores genéticos.	Algoritmo Bio-inspirado, Operadores Genéticos	Inicio de cada iteración del algoritmo genético.	Población actual, parámetros de configuración (P_c , P_m , η_c , η_m , k).	Lista de N nuevos individuos (conjuntos de parámetros).	<ol style="list-style-type: none"> 1. Seleccionar padres (Torneo). 2. Aplicar cruzamiento (SBX) condicional. 3. Aplicar mutación (Polinomial) condicional. 4. Corregir límites de parámetros.
Evaluar Fitness	Calcular fitness penalizado (F_p) combinando LE y violaciones.	R E B O U N D, Módulo de Cálculo de LE, Restricciones	Evaluación de cada individuo en el algoritmo.	Parámetros del individuo, resultados de simulación, restricciones, lambda_p.	Valor numérico de $F_p(x)$.	<ol style="list-style-type: none"> 1. Calcular LE ($f(x)$). 2. Calcular penalización $P(x)$ por violaciones. 3. Calcular $F_p(x) = f(x) + \lambda_p \cdot P(x)$.
Formar Siguiente Población	Construir la población para la siguiente iteración con elitismo.	Algoritmo Bio-inspirado	Después de generar descendencia y evaluar fitness.	Población actual, descendencia, tamaño de élite (k).	Lista de N individuos para la siguiente generación.	<ol style="list-style-type: none"> 1. Ordenar poblaciones por F_p. 2. Copiar k mejores individuos (<i>élite</i>). 3. Completar con mejores descendientes.

Continúa en la siguiente página

Tabla 5.1 – continuación de la página anterior

Nombre del Proceso	Objetivo	Actores Invu lucrados	Inicio (Even- to)	Entradas Principales	Salidas Princi- pales	Pasos Clave
Crear Nueva Simulación	Instanciar un nuevo entorno de simulación vacío en REBOUND.	REBOUND, Algoritmo Bioinspirado	Solicitud del Algoritmo Bioinspirado para nueva simulación.	Solicitud de creación (trigger).	Referencia a nuevo objeto Simulation.	<ol style="list-style-type: none"> 1. Invocar constructor de REBOUND. 2. Inicialización interna por REBOUND. 3. Retornar referencia a ‘Simulation’.
Agregar Cuerpos	Añadir una partícula con propiedades físicas a la simulación.	REBOUND, Algoritmo Bioinspirado	Configuración de la simulación antes de ejecutar.	Referencia a sim, parámetros del cuerpo (masa, posición, velocidad).	Instancia sim modificada con nueva partícula.	<ol style="list-style-type: none"> 1. Recibir parámetros del cuerpo. 2. Invocar API de REBOUND para añadir partícula. 3. Actualización interna por REBOUND.
Definir Condiciones	Configurar parámetros operativos de la simulación (dt , G , T_{\max}).	REBOUND, Algoritmo Bioinspirado	Antes de iniciar la ejecución de la simulación.	Referencia a sim, parámetros de configuración (dt , G , T_{\max}).	Instancia sim configurada, valor T_{\max} para ejecución.	<ol style="list-style-type: none"> 1. Recibir y desempaquetar parámetros. 2. Establecer dt y G en sim. 3. Procesar T_{\max} para uso posterior.

Continúa en la siguiente página

Tabla 5.1 – continuación de la página anterior

Nombre del Proceso	Objetivo	Actores Involucrados	Inicio (Evento)	Entradas Principales	Salidas Principales	Principales	Pasos Clave
Iniciar/Ejecutar Simulación	Ejecutar la integración numérica paso a paso hasta T_{\max} .	R E B O U N D, Módulo de Visualización	Inicio de la simulación para un individuo.	Referencia a <code>sim</code> , T_{\max} , lista de tiempos de visualización.	Estructura <code>SimulationResult</code> con trayectoria completa.		<ol style="list-style-type: none"> 1. Inicializar almacenamiento de resultados. 2. Bucle: avanzar un paso, almacenar estado, verificar visualización. 3. Empaquetar y devolver resultados.
Avanzar un Paso de Simulación	Avanzar el estado del sistema un paso dt usando WH-Fast (DKD).	REBOUND (Integrador WH-Fast)	Cada iteración del bucle de simulación.	Estado actual de <code>sim</code> , target_time ($t + dt$).	Instancia <code>sim</code> actualizada al tiempo target_time.		<ol style="list-style-type: none"> 1. Primer Drift ($dt/2$). 2. Kick (dt). 3. Segundo Drift ($dt/2$). 4. Finalización y sincronización.
Actualizar Estado	Consolidar el nuevo estado del sistema tras la integración.	REBOUND	Después de cada paso de integración.	Referencia a <code>sim</code> post-integración.	Instancia <code>sim</code> con estado consolidado.		<ol style="list-style-type: none"> 1. Validar tiempo de simulación. 2. Verificar actualización de posiciones y velocidades. 3. Confirmar masas sin cambios.

Continúa en la siguiente página

Tabla 5.1 – continuación de la página anterior

Nombre del Proceso	Objetivo	Actores Involucrados	Inicio (Evento)	Entradas Principales	Salidas Principales	Pasos Clave
Recolectar Datos	Extraer estado actual del sistema en instantes de visualización.	Módulo de Visualización, RE BOUND	Coincidencia de sim.t con t_vis.	Referencia a sim o estado actual.	Estructura VisualizationState con instantánea del sistema.	<ol style="list-style-type: none"> Leer tiempo, masas, posiciones (y velocidades si aplica). Empaquetar datos en VisualizationState.
Procesar Datos	Transformar y formatear datos para la biblioteca gráfica.	Módulo de Visualización	Recepción de Visualization State.	Visualización, configuración de visualización.	Datos formateados para la API gráfica.	<ol style="list-style-type: none"> Verificar configuración de visualización. (Condicional) Transformar coordenadas. Seleccionar datos relevantes. (Condicional) Calcular valores derivados. Formatear para API gráfica.
Generar Gráficos	Dibujar o actualizar la representación visual en la pantalla.	Módulo de Visualización, Biblioteca Gráfica	Recepción de datos formateados.	Datos formateados, contexto gráfico.	Representación gráfica actualizada en la UI.	<ol style="list-style-type: none"> Preparar lienzo (limpiar si es actualización). Llamar funciones de dibujo de la API gráfica. Actualizar/refrescar pantalla.

5.2. Diagrama de Actividades

La representación explícita del flujo de trabajo operativo es fundamental para el diseño, la implementación y la validación del sistema propuesto en este trabajo. Dicho sistema integra técnicas de optimización bioinspiradas con simulaciones gravitacionales de N-cuerpos, utilizando la biblioteca REBOUND y abordando inicialmente el caso de dos cuerpos. La comprensión detallada de la secuencia de actividades, las bifurcaciones lógicas y la interacción entre los componentes clave —*Usuario, Interfaz de Usuario, Motor de Optimización* (Algoritmo Bioinspirado), *Simulador* (REBOUND) y *Módulo de Visualización*— resulta esencial para gestionar la complejidad inherente a esta integración metodológica.

El diagrama de actividades presentado en la Sección 5.2.1 sirve como una representación formal de este proceso dinámico. Modela cómo el algoritmo bioinspirado gestiona la generación de conjuntos de parámetros, inicia ejecuciones de simulación en REBOUND y evalúa los resultados para refinar iterativamente la búsqueda de soluciones que satisfagan criterios predefinidos (p. ej., estabilidad orbital).

Mediante la descomposición del proceso global en pasos discretos y la asignación de responsabilidades a través de carriles (swimlanes), el diagrama clarifica la lógica operativa y establece una base para una implementación estructurada y verificable. Asegura que la integración del ciclo de optimización con el motor de simulación sea coherente con los objetivos del proyecto, facilitando el desarrollo de una solución robusta para la exploración de escenarios de N-cuerpos bajo criterios de optimización específicos.

5.2.1. Diagrama de Actividades

A continuación, se presenta el diagrama de actividades que modela el flujo operativo del sistema, abarcando la optimización de parámetros mediante algoritmos bioinspirados y la ejecución de simulaciones N-cuerpos con REBOUND.

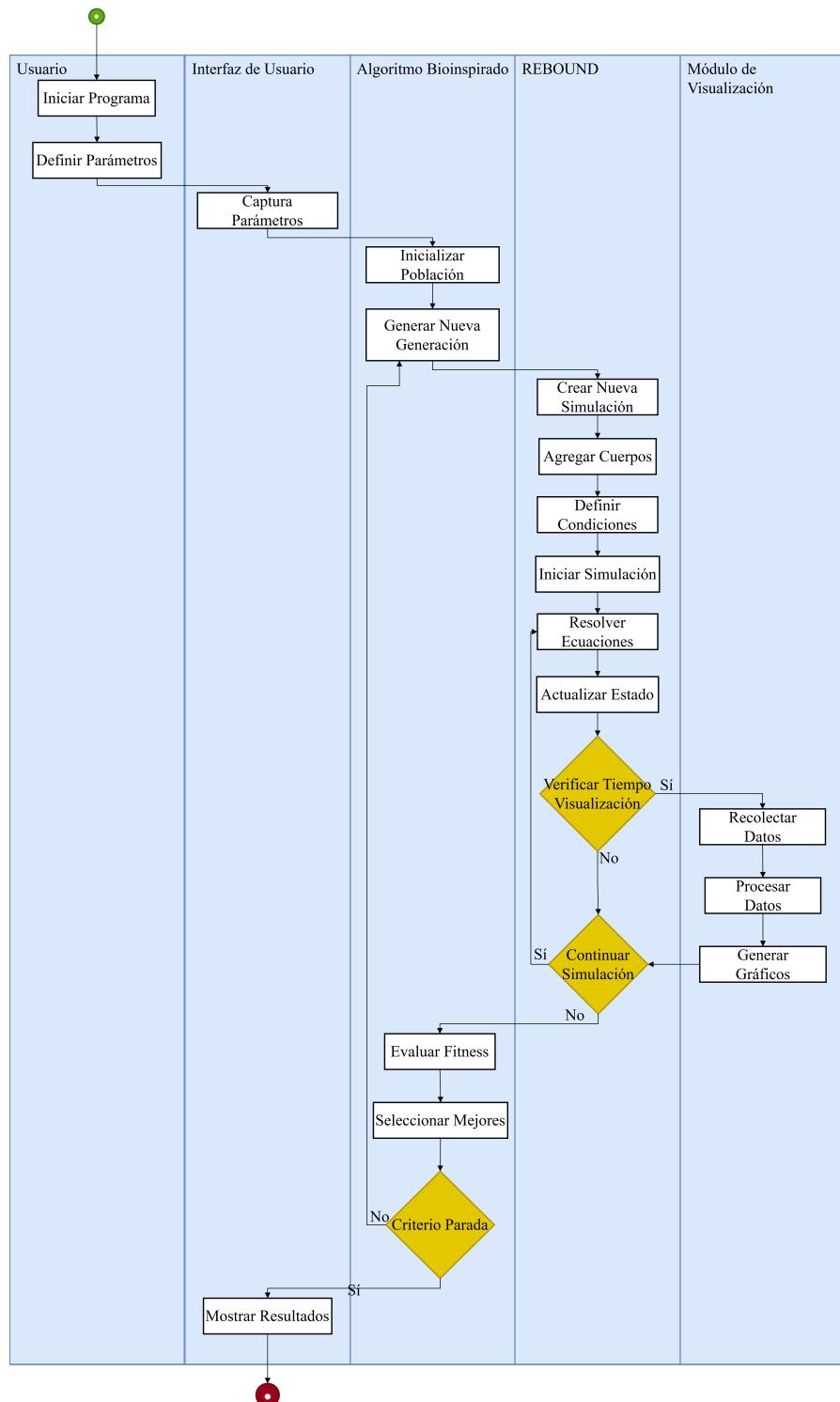


Figura 5.16: Diagrama de Actividades UML del proceso de simulación y optimización.

5.2.2. Descripción del Diagrama de Actividades

Estructura del Diagrama: Carriles (Swimlanes)

El diagrama se organiza en cinco carriles (swimlanes), cada uno representando un actor o componente lógico del sistema, delineando sus responsabilidades específicas:

1. **Usuario:** Representa al actor humano que interactúa con el sistema. Es responsable de iniciar la ejecución y proporcionar la configuración inicial requerida.
2. **Interfaz de Usuario (UI):** Componente software que media la interacción entre el *Usuario* y el núcleo del sistema. Gestiona la captura de parámetros de entrada y la presentación de resultados.
3. **Algoritmo Bioinspirado:** Componente central responsable de la optimización. Administra la población de soluciones candidatas (conjuntos de parámetros), aplica operadores evolutivos (generación, evaluación, selección) y determina la terminación del proceso según criterios definidos.
4. **REBOUND:** Representa la biblioteca de simulación gravitacional. Ejecuta simulaciones de N-cuerpos (inicialmente, 2 cuerpos) basadas en los parámetros proporcionados por el *Algoritmo Bioinspirado* para cada individuo evaluado. Cabe destacar que se instancia una nueva simulación para cada conjunto de parámetros a evaluar.
5. **Módulo de Visualización:** Componente encargado de generar representaciones gráficas del estado del sistema en instantes de tiempo específicos, definidos por el *Usuario*, para el monitoreo de la evolución dinámica.

Flujo Detallado de Actividades

El proceso modelado en el diagrama sigue la secuencia lógica que se describe a continuación:

1. Inicialización y Configuración:

- El *Usuario* inicia la ejecución de la aplicación (**Iniciar Programa**).
- El *Usuario* especifica los parámetros de configuración inicial, incluyendo rangos de búsqueda para la optimización, la función objetivo, criterios de parada, y los instantes discretos de tiempo para la visualización (**Definir Parámetros**).
- La *Interfaz de Usuario* capture y valida esta información (**Captura Parámetros**).

2. Ciclo Iterativo de Optimización y Simulación:

- La *UI* transmite los parámetros de configuración al *Algoritmo Bioinspirado*, el cual inicializa su población de soluciones candidatas (**Inicializar Población**).
- **Inicio del Bucle de Optimización:** El algoritmo genera una nueva generación de conjuntos de parámetros candidatos aplicando sus operadores definidos (**Generar Nueva Generación**).

- **Ejecución de Simulaciones por Individuo:** Para cada conjunto de parámetros (individuo) en la generación actual:
 - Se instruye a *REBOUND* para crear una nueva instancia de simulación (**Crear Nueva Simulación**).
 - Se añaden los cuerpos celestes al entorno de simulación con las propiedades especificadas por el conjunto de parámetros actual (**Agregar Cuerpos**).
 - Se configuran las condiciones restantes de la simulación, como el integrador numérico y el paso de tiempo (**Definir Condiciones**).
 - *REBOUND* inicia la simulación (**Iniciar Simulación**).
 - **Bucle Interno de Simulación (REBOUND):**
 - *REBOUND* resuelve las ecuaciones de movimiento para avanzar un paso de tiempo (**Resolver Ecuaciones**).
 - Actualiza el estado del sistema (posiciones, velocidades) (**Actualizar Estado**).
 - Verifica si el tiempo de simulación actual (t_{sim}) corresponde a uno de los instantes de visualización solicitados (t_{vis}) (**Verificar Tiempo Visualización**).
 - Si $t_{\text{sim}} = t_{\text{vis}}$ (Condición Verdadera): Se activa el *Módulo de Visualización*, que recolecta los datos relevantes del estado actual (**Recolectar Datos**), los procesa (**Procesar Datos**) y genera la salida gráfica correspondiente (**Generar Gráficos**). La simulación continúa.
 - Si $t_{\text{sim}} \neq t_{\text{vis}}$ (Condición Falsa): La simulación continúa directamente al siguiente paso.
 - *REBOUND* verifica si se ha alcanzado la condición de término de la simulación (p. ej., tiempo final T_{max}) (**Continuar Simulación**). Si no se ha alcanzado (Condición Falsa), retorna a **Resolver Ecuaciones**. Si se ha alcanzado (Condición Verdadera: **Simulación Terminada**), la ejecución para este individuo finaliza.
 - **Evaluación:** Tras la finalización de la simulación para un individuo, el *Algoritmo Bioinspirado* utiliza los resultados (p. ej., una métrica derivada del estado final o la trayectoria) para calcular su valor de aptitud (fitness) (**Evaluar Fitness**). Este paso se repite para todos los individuos de la generación.
 - **Selección y Criterio de Parada:** El *Algoritmo Bioinspirado* aplica sus mecanismos de selección basados en la aptitud calculada (**Seleccionar Mejores**) y verifica si se cumple el criterio global de parada (p. ej., número máximo de generaciones, convergencia de la aptitud) (**Criterio Parada**).

- Si el criterio no se cumple (Condición Falsa): Se procede a generar la siguiente generación (**Generar Nueva Generación**), continuando el ciclo de optimización.
- Si el criterio se cumple (Condición Verdadera): El ciclo de optimización concluye.

3. Finalización:

- El *Algoritmo Bioinspirado* transfiere la mejor solución encontrada (conjunto de parámetros óptimos y su aptitud asociada) a la *Interfaz de Usuario*.
- La *Interfaz de Usuario* presenta los resultados finales al *Usuario* (**Mostrar Resultados**).
- El proceso finaliza (nodo final de actividad).

5.2.3. Aspectos Clave del Diseño Reflejados en el Diagrama

El diagrama de actividades pone de manifiesto varios aspectos fundamentales del diseño del sistema:

- **Modularidad y Separación de Responsabilidades:** La estructura en carriles delimita claramente las funciones del componente de optimización (*Algoritmo Bioinspirado*) y del componente de simulación física (*REBOUND*), promoviendo un diseño modular.
- **Naturaleza Iterativa del Proceso:** El bucle principal que engloba **Generar Nueva Generación**, la ejecución de simulaciones en *REBOUND*, **Evaluuar Fitness** y **Criterio Parada** representa visualmente el núcleo iterativo característico de los algoritmos evolutivos y otros métodos de optimización bioinspirados.
- **Simulación como Evaluación de Aptitud:** Desde la perspectiva del optimizador, el módulo *REBOUND* actúa como una función objetivo (o parte de ella), evaluando la calidad (aptitud) de un conjunto de parámetros candidato mediante la ejecución de una simulación física.
- **Instanciación Independiente de Simulaciones:** El flujo muestra que cada evaluación de un individuo dentro del ciclo de optimización requiere la creación y ejecución de una nueva instancia de simulación en *REBOUND*, configurada con los parámetros específicos de dicho individuo.
- **Control Discreto de la Visualización:** La generación de visualizaciones está integrada en el bucle de simulación de *REBOUND* pero se activa únicamente en puntos temporales discretos especificados por el *Usuario*, evitando una sobrecarga continua.

5.3. diccionario de datos

En el desarrollo de este proyecto, la representación adecuada de los datos es funda-

mental para garantizar la consistencia, comprensión y el correcto funcionamiento del sistema, buscando facilitar tanto la interpretación del modelo, así como su mantenimiento y expansión futura.

Por lo cual, en esta sección se mostrará el diccionario de datos, donde se describirán los atributos de los datos, sus significados, sus restricciones y ejemplos que ilustran su uso.

Tabla 5.2: Cuerpo celeste.

Nombre del atributo	Descripción	Tipo de dato	Rango	Ejemplo
masa	Masa del cuerpo celeste que influye en su atracción gravitacional.	float	> 0 (positivos) para la primera masa, para la segunda masa mayor a 0 y menor a 10 veces la primera masa	1.0
a	Semieje mayor de la órbita, define el tamaño de la órbita elíptica, es la mitad del eje largo de la elipse y en caso de ser circular es el radio.	float	> 0 (positivos)	1.0
e	Excentricidad orbital, indica cuán alargada es la órbita siendo un círculo perfecto si la excentricidad es 0 y mientras más estirada, más cercana a 1	float	[0, 1)	0.1
inc_deg	Inclinación orbital respecto al plano de referencia, en grados.	float	[0°, 180°]	30.0
perturba	Indica si se aplica una perturbación inicial pequeña a los parámetros.	bool	True o False	True

Tabla 5.3: Simulación.

Nombre del atributo	Descripción	Tipo de dato	Rango	Ejemplo
t_max	Tiempo total de simulación en años.	float	> 0 (positivos)	100.0
N_steps	Número de pasos a almacenar, define la resolución temporal de la simulación. A mayor número de pasos, mayor resolución.	entero	> 0 (positivos)	1000
sim.units	Unidades de la simulación, siendo en orden unidad de distancia, unidad de tiempo y unidad de masa.	texto	AU, yr, Msun	AU, yr, Msun
sim.integrator	Indica el integrador numérico que se va a utilizar para avanzar la simulación en el tiempo.	texto	ias15, whfast, BS, mercurius	ias15
x, y, z	Guarda las posiciones de nuestros cuerpos	array(float)> 0 (positivos)	[5.0, 230.0, 20.0]	

Tabla 5.4: Métricas.

Nombre del atributo	Descripción	Tipo de dato	Rango	Ejemplo
times	Array que guarda los tiempos en la simulación.	array(float)> 0 (positivos)	[0.0, 100.0, 200.0]	
energy	Energía total del sistema, incluye la energía cinética y potencial.	float	Valor real	-0.5
a_arr, a_pert	Array que guarda el semieje mayor de la órbita en cada paso de la simulación y el array de semiejes mayores perturbados.	array(float)> 0 (positivos)	[1.0, 1.5, 2.0]	
e_arr, e_pert	Array que guarda la excentricidad orbital en cada paso de la simulación y el array de excentricidades perturbadas.	array(float){0, 1}	[0.1, 0.2, 0.3]	
Exponente de Lyapunov (λ)	Indica la tasa de crecimiento exponencial de las perturbaciones, relacionada con la estabilidad del sistema.	float	Valor real	0.01

5.3.1. Relacion de datos

En esta sección explicaremos como se interconectan los distintos atributos descritos previamente para tener un mayor entendimiento del sistema, así como el flujo de datos que se tiene.

1. times

Para conocer los tiempos en la simulación, lo que hacemos es generar un array de N_steps elementos, empezando por el 0 hasta t_max, cada elemento va a estar a una distancia de t_max entre N_steps

2. x, y, z, energy, a_arr, a_pert, e_arr, e_pert

Nos apoyamos de las funciones que nos brinda REBOUND el cual con su función de simulation nos brinda todos esos datos pero para ello necesitamos conocer los tiempos en la simulación (times), así como las masas de los cuerpos involucrados, sus semiejes mayores, su excentricidad y el integrador que se va a utilizar

3. Exponente de Lyapunov

Para poder calcular el exponente de Lyapunov necesitaremos calcular primero lo siguiente: a_arr, e_arr, a_pert, e_pert, para poder así obtener nuestro delta de orbital

CAPÍTULO 6

Diseño

6.1. Diseño Arquitectónico

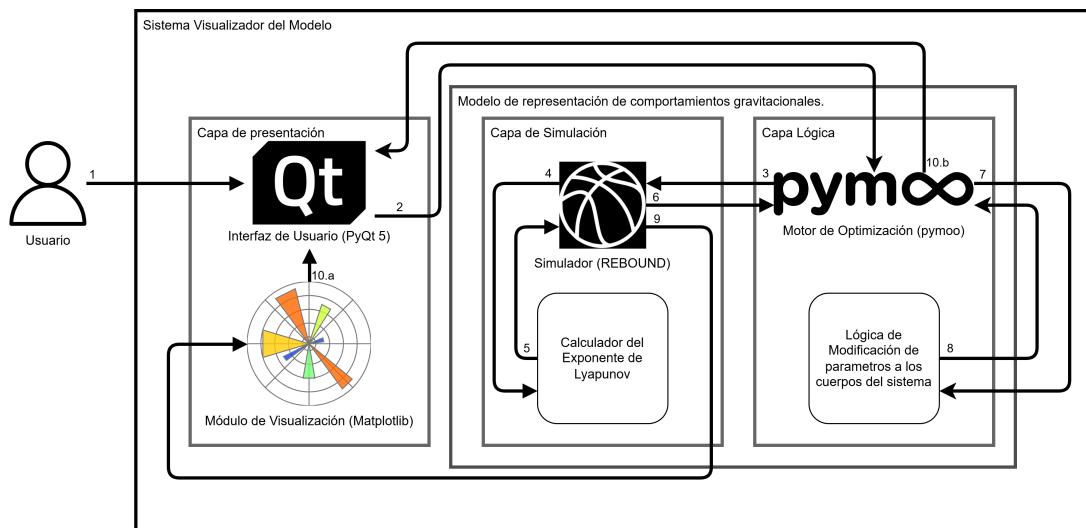


Figura 6.1: Diagrama de Arquitectura.

6.1.1. Introducción y Objetivos de Diseño

El presente diseño arquitectónico describe la estructura y organización del “Sistema Visualizador del Modelo”, concebido para la simulación, optimización y análisis de comportamientos gravitacionales, con un enfoque inicial en sistemas de dos cuerpos. La arquitectura se fundamenta en principios de **modularidad, encapsulamiento y separación de incumbencias**, con el fin de garantizar la extensibilidad, mantenibilidad y robustez del sistema. Se ha adoptado un paradigma arquitectónico híbrido que combina un enfoque basado en componentes, una estructuración en capas lógicas y una comunicación parcialmente dirigida por eventos.

Los objetivos primordiales que guían este diseño son:

1. **Modularidad y Cohesión:** Facilitar el desarrollo, prueba y mantenimiento independientes de los distintos módulos funcionales.

2. **Bajo Acoplamiento:** Minimizar las dependencias entre componentes para permitir la sustitución o mejora de tecnologías individuales (e.g., motor de simulación, biblioteca de optimización) con un impacto reducido en el resto del sistema.
3. **Capacidad de Modificación Dinámica:** Soportar la alteración de parámetros clave del sistema (ej. masas) durante la ejecución del ciclo de optimización.
4. **Eficiencia Computacional:** Integrar herramientas especializadas como REBOUND para la simulación N-cuerpos y pymoo para la optimización bioinspirada, buscando un balance entre precisión y tiempo de ejecución.
5. **Visualización Interactiva:** Proveer retroalimentación visual clara y comprensible del proceso de simulación y optimización al usuario a través de PyQt5 y Matplotlib.

La arquitectura propuesta se articula en torno a tres capas principales: Presentación, Lógica y Simulación/Datos, cada una con componentes especializados que interactúan mediante interfaces bien definidas.

6.1.2. Arquitectura en Capas y Componentes Principales

El sistema se organiza en las siguientes capas lógicas, cada una agrupando componentes con responsabilidades cohesivas:

Capa de Presentación

Propósito General: Gestionar toda la interacción con el usuario final, desde la captura de datos de entrada hasta la visualización de resultados y el estado del sistema. Esta capa abstrae los detalles de la interfaz gráfica y la representación de datos del resto de la lógica del sistema.

1. 2.1.1. Interfaz de Usuario (UI)

- **Tecnología:** PyQt5.
- **Responsabilidades Fundamentales:**
 - **Recolección y Validación de Parámetros:** Presenta formularios y controles para que el usuario ingrese los parámetros iniciales de la simulación (e.g., masas, posiciones, velocidades iniciales de los cuerpos) y los parámetros de configuración del algoritmo de optimización (e.g., tamaño de la población, número de generaciones, criterios de parada, rangos de los parámetros a optimizar). Implementa rutinas de validación para asegurar la coherencia y admisibilidad de los datos antes de iniciar cualquier proceso.
 - **Iniciación y Control de Procesos:** Actúa como el punto de entrada para disparar el ciclo de optimización invocando al Motor de Optimización. Permite al usuario iniciar, pausar o detener los procesos.
 - **Presentación de Resultados y Retroalimentación:** Despliega

los resultados finales de la optimización (mejor conjunto de parámetros, valor de fitness correspondiente) y provee retroalimentación en tiempo real sobre el progreso del algoritmo de optimización (e.g., número de generación actual, mejor fitness de la generación).

- **Datos Manejados:**

- *Entrada:* Parámetros de configuración de simulación y optimización, eventos de control del usuario.
- *Salida:* Representación textual y gráfica de resultados, estado del sistema.

2. 2.1.2. Módulo de Visualización

- **Tecnología:** Matplotlib, integrado en la UI de PyQt5.
- **Responsabilidades Fundamentales:**
 - **Generación de Representaciones Gráficas:** Transforma los datos numéricos provenientes de la Capa de Simulación (trayectorias, estados energéticos) y de la Capa Lógica (progreso de la optimización) en visualizaciones comprensibles.
 - **Renderizado y Actualización Dinámica:** Es responsable de renderizar los gráficos (e.g., trayectorias orbitales 2D/3D, evolución de la energía del sistema, convergencia del fitness) dentro de los widgets designados en la UI./ Soporta la actualización dinámica de estos gráficos para reflejar el estado cambiante durante la simulación y la optimización.
 - **Procesamiento de Datos para Visualización:** Realiza las transformaciones necesarias sobre los datos crudos (e.g., escalado, selección de subconjuntos de datos, cálculo de métricas derivadas para visualización) para adecuarlos a los requisitos de Matplotlib y a las preferencias del usuario.
- **Datos Manejados:**
 - *Entrada:* Series temporales de posiciones y velocidades, valores de energía, métricas de fitness, configuración de visualización.
 - *Salida:* Objetos gráficos de Matplotlib (figuras, ejes, primitivas de dibujo).

Capa Lógica

Propósito General: Orquestar el proceso de optimización de parámetros. Contiene la inteligencia para guiar la búsqueda de soluciones óptimas y adaptar los parámetros del sistema basándose en el rendimiento observado.

1. 2.2.1. Motor de Optimización

- **Tecnología:** pymoo.

- **Responsabilidades Fundamentales:**

- **Gestión del Algoritmo Bioinspirado:** Implementa el ciclo de vida de un algoritmo genético (u otro paradigma bioinspirado soportado por `pymoo`). Esto incluye la inicialización de una población de soluciones candidatas (conjuntos de parámetros), la aplicación iterativa de operadores evolutivos (selección, cruzamiento, mutación) para generar nuevas poblaciones, y la gestión de criterios de parada.
- **Evaluación de la Función Objetivo (Fitness):** Para cada individuo (conjunto de parámetros) de la población, coordina su evaluación. Esto implica invocar a la Capa de Simulación para ejecutar una simulación con dichos parámetros y obtener las trayectorias, y luego utilizar el Calculador del Exponente de Lyapunov para obtener un valor escalar de fitness que cuantifique la “calidad” de esa solución.
- **Orquestación del Flujo de Optimización:** Controla el flujo iterativo, decidiendo cuándo generar una nueva población, cuándo evaluar, y cuándo terminar el proceso.

- **Datos Manejados:**

- *Entrada:* Configuración del algoritmo de optimización (desde la UI), valores de fitness (desde el Calculador LE).
- *Salida:* Conjuntos de parámetros para evaluación (hacia la Capa de Simulación), mejor solución encontrada y métricas de progreso (hacia la UI).

2. 2.2.2. Lógica de Modificación de Parámetros

- **Responsabilidades Fundamentales:**

- **Aplicación de Variaciones Paramétricas:** Actúa como intermediario para aplicar los conjuntos de parámetros generados por el Motor de Optimización (`pymoo`) a las instancias de simulación en REBOUND. Asegura que los parámetros propuestos por los operadores genéticos (e.g., masas, componentes de velocidad/posición inicial) se traduzcan correctamente en la configuración de cada simulación individual.
- **Retroalimentación y Adaptación del Proceso de Optimización (Potencial):** Aunque el flujo principal es que `pymoo` genere los parámetros, este módulo podría incorporar lógicas para adaptar dinámicamente el propio proceso de optimización. Por ejemplo, basándose en la convergencia observada o la diversidad de la población, podría ajustar hiperparámetros de `pymoo` (como tasas de mutación/cruzamiento) o refinar los rangos de búsqueda de los parámetros, interactuando para ello con el Motor de Optimización.

- **Datos Manejados:**

- *Entrada:* Nuevos conjuntos de parámetros (individuos) generados por pymoo.
- *Salida:* Parámetros configurados para instancias específicas del Simulador REBOUND.

Capa de Simulación y Datos

Propósito General: Ejecutar las simulaciones físicas y realizar los cálculos necesarios para evaluar la calidad de las configuraciones paramétricas. Esta capa encapsula la física del problema y las herramientas numéricas para su resolución.

1. 2.3.1. Simulador

- **Tecnología:** REBOUND.
- **Responsabilidades Fundamentales:**
 - **Modelado del Sistema Físico:** Configura y ejecuta simulaciones del problema de N-cuerpos (inicialmente dos cuerpos) de acuerdo con los parámetros proporcionados (masas, posiciones iniciales, velocidades iniciales, constante gravitacional G , paso de tiempo dt , tiempo máximo de simulación T_{max}).
 - **Integración Numérica:** Utiliza los integradores numéricos provistos por REBOUND (e.g., WHFast, IAS15) para avanzar el estado del sistema en el tiempo, calculando las trayectorias de los cuerpos bajo la influencia de sus interacciones gravitacionales mutuas.
 - **Provisión de Datos de Trayectoria:** Exporta los resultados de la simulación, típicamente como series temporales de las posiciones y velocidades de cada cuerpo, que son consumidos por el Calculador del Exponente de Lyapunov y, opcionalmente, por el Módulo de Visualización.
- **Datos Manejados:**
 - *Entrada:* Parámetros físicos y de simulación (masas, estado inicial, G , dt , T_{max} , integrador a usar).
 - *Salida:* Series temporales de vectores de posición y velocidad para cada cuerpo.

2. 2.3.2. Calculador del Exponente de Lyapunov (LE)

- **Responsabilidades Fundamentales:**
 - **Cuantificación de la Estabilidad Dinámica:** Implementa el algoritmo para calcular el Exponente de Lyapunov Máximo (MLE) a partir de las trayectorias generadas por el Simulador. El LE es una métrica fundamental en la teoría de sistemas dinámicos que mide la tasa promedio de separación exponencial de trayectorias inicialmente cercanas. Un LE positivo indica comportamiento caótico, mientras que un LE cercano a cero o negativo sugiere estabilidad u orbitalidad regular.

- **Función de Fitness:** Provee el valor del LE calculado como la función objetivo (o un componente principal de ella) al Motor de Optimización. El objetivo de la optimización es, típicamente, encontrar conjuntos de parámetros que minimicen el LE, buscando configuraciones de mayor estabilidad.
- **Procesamiento de Trayectorias:** Toma como entrada los datos de trayectoria (series temporales de estado) de dos o más simulaciones ligeramente perturbadas (o utiliza métodos basados en la evolución de vectores tangentes) para estimar la tasa de divergencia.
- **Datos Manejados:**
 - *Entrada:* Datos de trayectoria (posiciones, velocidades a lo largo del tiempo) de REBOUND.
 - *Salida:* Un valor escalar representando el Exponente de Lyapunov.

6.1.3. Interacciones Principales y Flujos de Datos y Control

El funcionamiento coordinado del sistema se basa en una serie de interacciones clave entre sus componentes. A continuación, se detallan estos flujos de control y datos, numerados según la descripción original y expandidos para mayor claridad:

1. Usuario → Interfaz de Usuario (Inicio y Configuración Inicial):

- **Disparador:** Acción del operador humano.
- **Acción:** El usuario interactúa con la Interfaz de Usuario (UI) para introducir los parámetros fundamentales de la simulación (e.g., valores nominales o rangos para las masas de los dos cuerpos, sus posiciones y velocidades iniciales), la constante gravitacional G , el paso de tiempo dt para la integración numérica, y el tiempo máximo de simulación T_{max} para cada evaluación individual. Adicionalmente, se configuran los parámetros del Motor de Optimización (`pymoo`), tales como el tamaño de la población de soluciones, el número máximo de generaciones, los criterios de parada, y la especificación de qué parámetros físicos (e.g., masas) serán optimizados junto con sus rangos de búsqueda permitidos. La activación de un control en la UI (e.g., botón “Iniciar Simulación/Optimización”) formaliza la entrada de estos datos.
- **Datos Intercambiados:** Estructuras de datos o variables conteniendo los valores numéricos y selecciones de configuración ingresados por el usuario.
- **Resultado Esperado:** La UI valida internamente la coherencia y admisibilidad de los parámetros (e.g., rangos lógicos, tipos de datos correctos) y los prepara para su posterior procesamiento y distribución a las capas correspondientes.

2. Interfaz de Usuario → Capa de Simulación (Configuración de Simulación Base/Visualización Directa - *Contexto Específico*):

- **Disparador:** Podría ser una acción del usuario para una simulación directa sin optimización, o un paso de configuración previo a la optimización.
- **Acción:** En un escenario donde el usuario desee ejecutar una simulación única con parámetros fijos (sin pasar por el ciclo de optimización bioinspirada) o para establecer una simulación base, la UI enviaría los parámetros validados (masas, posiciones, dt , G , T_{max}) directamente al subsistema de simulación (específicamente al Simulador REBOUND a través de una lógica de control simplificada).
- **Datos Intercambiados:** Conjunto completo de parámetros físicos y de simulación necesarios para una ejecución de REBOUND.
- **Resultado Esperado:** El Simulador REBOUND se configura y está listo para ejecutar una integración numérica. *Nota: En el flujo principal de optimización, esta interacción directa es menos prominente, ya que el Motor de Optimización gestiona las invocaciones a REBOUND.*

3. Interfaz de Usuario → Motor de Optimización (pymoo) (Inicio del Ciclo Evolutivo):

- **Disparador:** Evento de “inicio” generado por la UI tras la validación de los parámetros de configuración (como se describe en la Interacción 1), específicamente cuando el objetivo es la optimización.
- **Acción:** La UI transfiere la configuración completa del problema de optimización al Motor de Optimización (pymoo). Esto incluye la definición de las variables de diseño (qué parámetros físicos se optimizan), sus límites (rangos de búsqueda), la función objetivo (implícitamente, minimizar el LE), y los parámetros del algoritmo genético (tamaño de población, número de generaciones, etc.).
- **Datos Intercambiados:** Objeto o estructura de datos que encapsula la definición del problema de optimización y los hiperparámetros del algoritmo genético.
- **Resultado Esperado:** pymoo se inicializa, creando la primera población de individuos (conjuntos de parámetros candidatos) de acuerdo con la estrategia de inicialización configurada (e.g., muestreo aleatorio dentro de los rangos).

4. Motor de Optimización (pymoo) → Simulador (REBOUND) (Solicitud de Evaluación de Individuo):

- **Disparador:** Necesidad del Motor de Optimización de evaluar el fitness de cada individuo (conjunto de parámetros) en la población actual.
- **Acción:** Para cada individuo, pymoo instruye (a través de la Lógica de Modificación de Parámetros si actúa como intermediaria, o directamente) al Simulador REBOUND para que ejecute una simulación. Los

parámetros específicos del individuo (e.g., masas m_1, m_2 propuestas por `pymoo`) se utilizan para configurar esta instancia particular de la simulación, junto con los parámetros de simulación globales (G , dt , T_{max}).

- **Datos Intercambiados:** Vector de parámetros del individuo actual y parámetros de simulación fijos.
- **Resultado Esperado:** El Simulador `REBOUND` ejecuta una integración numérica completa para el conjunto de parámetros dado, generando una trayectoria.

5. Simulador (`REBOUND`) → Calculador del Exponente de Lyapunov (Provisión de Trayectorias):

- **Disparador:** Conclusión exitosa de una simulación en `REBOUND` para un individuo específico.
- **Acción:** `REBOUND` hace disponibles los datos de la trayectoria resultante. Esto típicamente comprende una serie temporal de los vectores de estado (posiciones y velocidades) de los dos cuerpos a lo largo del tiempo de simulación T_{max} .
- **Datos Intercambiados:** Estructura de datos (e.g., array multidimensional, lista de tuplas) que representa la evolución temporal del sistema.
- **Resultado Esperado:** El Calculador LE recibe estos datos y está preparado para realizar el cómputo del Exponente de Lyapunov.

6. Motor de Optimización (`pymoo`) → Simulador (`REBOUND`) (Invocaciones Múltiples para Evaluación):

- **Clarificación:** Esta interacción es una generalización de la Interacción 4. No es una interacción separada en el flujo secuencial, sino que enfatiza que el Motor de Optimización (`pymoo`) realizará múltiples solicitudes de simulación a `REBOUND`, una por cada individuo en la población actual que necesita ser evaluado, y esto se repetirá para cada nueva generación de individuos cuyos parámetros han sido “mutados” (modificados por operadores genéticos como cruzamiento y mutación).
- **Acción y Resultado:** Idénticos a la Interacción 4, pero repetidos para cada individuo de la población que `pymoo` necesita evaluar en una generación dada.

7. Motor de Optimización (`pymoo`) → Interfaz de Usuario (Retroalimentación de Progreso y Resultados):

- **Disparador:** Típicamente al final de cada generación del algoritmo evolutivo, o al alcanzar un criterio de parada.
- **Acción:** `pymoo` comunica al UI el estado actual del proceso de optimización. Esto incluye métricas como el número de la generación actual, el mejor valor de fitness (LE mínimo) encontrado en esa generación

o hasta el momento, y opcionalmente, los parámetros del individuo que logró dicho mejor fitness. Si el proceso ha concluido, se envían los resultados finales.

- **Datos Intercambiados:** Valores numéricos de métricas de optimización, y potencialmente el vector de parámetros del mejor individuo.
- **Resultado Esperado:** La UI actualiza sus elementos visuales (e.g., campos de texto, barras de progreso, gráficos de convergencia) para informar al usuario sobre el avance y los resultados de la optimización.

8. Lógica de Modificación de Parámetros → Motor de Optimización (`pymoo`) (Adaptación Dinámica del Proceso de Optimización):

- **Disparador:** Podría ser activado por hitos en el proceso de optimización (e.g., después de un cierto número de generaciones, o si se detecta estancamiento en la convergencia del fitness).
- **Acción:** La Lógica de Modificación de Parámetros, basándose en el análisis del progreso de la optimización (e.g., diversidad de la población, tasa de mejora del fitness), podría proponer ajustes a los hiperparámetros del Motor de Optimización (`pymoo`). Esto podría incluir la modificación dinámica de los rangos de búsqueda para ciertos parámetros físicos (si se observa que las mejores soluciones se concentran en subregiones) o el ajuste de las tasas de los operadores genéticos (e.g., aumentar la mutación si la diversidad es baja).
- **Datos Intercambiados:** Nuevos valores para hiperparámetros de `pymoo` o nuevos límites para las variables de diseño.
- **Resultado Esperado:** `pymoo` incorpora estos ajustes en las subsiguientes generaciones, potencialmente mejorando la eficiencia o la capacidad de exploración del algoritmo de optimización.

9. Simulador (REBOUND) → Motor de Optimización (`pymoo`) (Provisión de Datos Adicionales para Análisis):

- **Disparador:** Finalización de una simulación en REBOUND, concurrente con la Interacción 5.
- **Acción:** Además de las trayectorias completas necesarias para el cálculo del LE, REBOUND podría reenviar al Motor de Optimización (posiblemente a través de la Lógica de Modificación de Parámetros) datos intermedios o estados finales específicos (e.g., energía total final del sistema, momento angular, distancia mínima entre cuerpos durante la simulación).
- **Datos Intercambiados:** Valores escalares o vectores representando métricas adicionales de la simulación.
- **Resultado Esperado:** `pymoo` o la Lógica de Modificación de Parámetros pueden utilizar estos datos para análisis de convergencia más sofisticados, para la implementación de restricciones en el problema de

optimización (e.g., penalizar soluciones que no conservan la energía), o para refinar la función de fitness más allá del simple LE.

10. Visualización y Actualizaciones de UI:

- (a) Módulo de Visualización → Interfaz de Usuario (Renderizado de Gráficos):
 - **Disparador:** Solicitud de la UI para visualizar una trayectoria específica (e.g., la del mejor individuo de la generación actual) o datos de simulación.
 - **Acción:** El Módulo de Visualización, utilizando `Matplotlib`, procesa los datos de trayectoria o estado recibidos (que originalmente provinieron de `REBOUND` y fueron gestionados por la UI o el Motor de Optimización) y genera las representaciones gráficas correspondientes (e.g., órbitas, gráficos de energía vs. tiempo). Estos gráficos se renderizan o actualizan dentro de los widgets designados en la UI.
 - **Datos Intercambiados:** Objetos gráficos de `Matplotlib`.
 - **Resultado Esperado:** El usuario observa una representación visual actualizada del comportamiento del sistema o del progreso de la optimización.
- (b) Motor de Optimización → UI / Indicadores (Actualización de Estado en Tiempo Real):
 - **Disparador:** Emisión de eventos de estado por parte de `pymoo` (como se mencionó en la Interacción 7).
 - **Acción:** La UI recibe estos eventos o datos de estado (e.g., número de generación, mejor fitness actual) y actualiza los componentes visuales correspondientes, como barras de progreso, etiquetas de texto o gráficos de convergencia de fitness.
 - **Datos Intercambiados:** Métricas de estado de la optimización.
 - **Resultado Esperado:** El usuario obtiene una retroalimentación continua y en tiempo (casi) real sobre el estado del proceso de optimización.

6.1.4. Estilo y Paradigmas Arquitectónicos

La arquitectura del sistema se adhiere a los siguientes estilos y paradigmas para alcanzar los objetivos de diseño:

- Arquitectura Basada en Componentes (Component-Based Architecture):
 - **Justificación:** El sistema se descompone en módulos lógicos cohesivos (UI, Módulo de Visualización, Motor de Optimización, Lógica de Modificación de Parámetros, Simulador, Calculador LE). Cada componente encapsula una funcionalidad específica y expone interfaces bien

definidas (e.g., APIs programáticas, llamadas a funciones) para la interacción.

- **Beneficios:** Este enfoque promueve la reutilización de componentes, facilita el desarrollo paralelo y las pruebas unitarias aisladas. Por ejemplo, el Simulador REBOUND es un componente externo con su propia API, y el Motor de Optimización pymoo también opera como una biblioteca con interfaces claras. La sustitución de pymoo por otro motor de optimización sería factible modificando principalmente la Lógica de Modificación de Parámetros y las interacciones con la UI, siempre que el nuevo motor cumpla con un protocolo de comunicación similar para la evaluación de individuos.
- **Arquitectura en Capas (Layered Architecture):**
 - **Justificación:** Las capas (Presentación, Lógica, Simulación/Datos) establecen una jerarquía de abstracción y control. Las interacciones ocurren típicamente entre capas adyacentes. La Capa de Presentación interactúa con la Capa Lógica, la cual a su vez interactúa con la Capa de Simulación/Datos.
 - **Beneficios:** Este estilo reduce el acoplamiento global del sistema. Por ejemplo, la UI (Capa de Presentación) no necesita conocer los detalles internos de la integración numérica en REBOUND (Capa de Simulación/Datos); solo se comunica a través de la Capa Lógica. Esto mejora la mantenibilidad, ya que los cambios dentro de una capa (e.g., actualizar la biblioteca de graficación en la Capa de Presentación) tienen un impacto mínimo en otras capas, siempre que las interfaces entre capas se mantengan estables.
- **Arquitectura Dirigida por Eventos (Event-Driven Architecture) - Parcial:**
 - **Justificación:** Principalmente en la interacción entre la UI y el Motor de Optimización, y para la actualización de la visualización. La UI genera eventos (e.g., “botón Iniciar presionado”) que disparan acciones en la Capa Lógica. A su vez, el Motor de Optimización puede emitir eventos (e.g., “nueva generación completada”, “mejor fitness actualizado”) que son capturados por la UI para actualizar la visualización de progreso.
 - **Beneficios:** Permite un comportamiento reactivo y desacoplado, especialmente para la interfaz de usuario. Las actualizaciones de la UI pueden ocurrir asincrónicamente a medida que el proceso de optimización avanza, proporcionando una experiencia de usuario más fluida y retroalimentación en tiempo real sin bloquear el hilo principal de la UI durante cálculos intensivos.

6.1.5. Consideraciones Adicionales de Diseño

- **Mantenibilidad:** La clara separación de responsabilidades y el uso de interfaces definidas entre componentes facilitan la comprensión del código,

la localización de errores y la implementación de modificaciones o nuevas funcionalidades con menor riesgo de introducir efectos secundarios no deseados.

- **Escalabilidad Funcional:** La arquitectura está diseñada para ser extensible. Añadir un nuevo tipo de algoritmo de optimización implicaría desarrollar un nuevo componente o adaptar el existente en la Capa Lógica que interactúe con la API de `pymoo` o una biblioteca alternativa. De manera similar, soportar un nuevo integrador numérico en `REBOUND` sería una configuración dentro de la Capa de Simulación. La escalabilidad en términos de rendimiento para problemas de $N > 2$ cuerpos es inherentemente soportada por `REBOUND` y `pymoo` (que maneja vectores de parámetros), aunque requeriría ajustes en la Lógica de Modificación de Parámetros y en la función de fitness.
- **Capacidad de Pruebas:** La modularidad intrínseca permite la implementación de pruebas unitarias para cada componente de forma aislada (e.g., probar el Calculador LE con trayectorias predefinidas, verificar la lógica de los operadores genéticos en `pymoo` con funciones objetivo sintéticas). Las pruebas de integración se centrarían en verificar las interacciones correctas entre componentes y capas (e.g., asegurar que los parámetros de la UI se transmitan correctamente al Simulador a través del Motor de Optimización).
- **Protocolos de Comunicación:** Las interacciones entre componentes se realizan principalmente mediante llamadas a funciones directas dentro del mismo proceso. El intercambio de datos se realiza mediante estructuras de datos definidas (e.g., listas, diccionarios de Python, arrays de NumPy para trayectorias).

Bibliografía

- [1] S. Orlov. «C++ Playground for Numerical Integration Method Developers». En: *Supercomputing*. Ed. por V. Voevodin y S. Sobolev. Cham: Springer International Publishing, 2017, págs. 418-429. DOI: 10.1007/978-3-319-71255-0_34.
- [2] R. Juan D. Ayala P. Brunet e I. Navazo. «Object representation by means of nonminimal division quadtrees and octrees». En: *ACM Trans. Graph.* 4.1 (1985), págs. 41-59. DOI: 10.1145/3973.3975.
- [3] J. Maguire y L. Woodcock. «Artificial Intelligent Molecular Dynamics and Hamiltonian Surgery». Tesis doct. Research Gate, 2005. URL: https://www.researchgate.net/profile/Leslie-Woodcock/publication/355486308_Artificial_Intelligent_Molecular_Dynamics_and_Hamiltonian_Surgery/links/617535ba0be8ec17a921a0c1/Artificial-Intelligent-Molecular-Dynamics-and-Hamiltonian-Surgery.pdf.
- [4] Michael P. Allen y Dominic J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, jun. de 2017. ISBN: 9780198803195. DOI: 10.1093/oso/9780198803195.001.0001. URL: <https://doi.org/10.1093/oso/9780198803195.001.0001>.
- [5] Peter Atkins y Julio de Paula. *Physical Chemistry*. 9th. Oxford: Oxford University Press, 2008. ISBN: 9780195685220. URL: <http://www.worldcat.org/isbn/9780195685220>.
- [6] D. Potter I. Alonso Asensio C. Dalla Vecchia y J. Stadel. «Mesh-free hydrodynamics in pkdgrav3 for galaxy formation simulations». En: *Monthly Notices of the Royal Astronomical Society* 519.1 (2022), págs. 300-317. DOI: 10.1093/mnras/stac3447.
- [7] Dong-Hong Wu, Sheng Jin y Jason H. Steffen. «Enhanced Stability in Planetary Systems with Similar Masses». En: *The Astronomical Journal* 169.1 (dic. de 2024), pág. 28. DOI: 10.3847/1538-3881/ad9388. URL: <https://dx.doi.org/10.3847/1538-3881/ad9388>.
- [8] H. Rein y S.-F. Liu. «REBOUND: an open-source multi-purpose N-body code for collisional dynamics». En: *Astronomy & Astrophysics* 537 (2012). Published online 20 January 2012, pág. 10. DOI: 10.1051/0004-6361/201118085. URL: <https://doi.org/10.1051/0004-6361/201118085>.

- [9] Hanno Rein y col. «Hybrid symplectic integrators for planetary dynamics». En: *Monthly Notices of the Royal Astronomical Society* 485.4 (jun. de 2019), págs. 5490-5497. DOI: 10.1093/mnras/stz769. arXiv: 1903.04972 [astro-ph.EP].
- [10] Sir Isaac Newton. *Philosophiae Naturalis Principia Mathematica*. Trad. por Andrew Motte. Revised and edited by Florian Cajori. University of California Press, 1934. The Royal Society, 1687.
- [11] A. M. LYAPUNOV and. «The general problem of the stability of motion». En: *International Journal of Control* 55.3 (1992), págs. 531-534. DOI: 10 . 1080 / 00207179208934253. eprint: <https://doi.org/10.1080/00207179208934253>. URL: <https://doi.org/10.1080/00207179208934253>.
- [12] N.V. Kuznetsov y G.A. Leonov. «On stability by the first approximation for discrete systems». En: *Proceedings. 2005 International Conference Physics and Control, 2005*. Ago. de 2005, págs. 596-599. DOI: 10.1109/PHYCON.2005.1514053.
- [13] N. V. Kuznetsov, T. A. Alexeeva y G. A. Leonov. «Invariance of Lyapunov exponents and Lyapunov dimension for regular and irregular linearizations». En: *Nonlinear Dynamics* 85.1 (feb. de 2016), págs. 195-201. ISSN: 1573-269X. DOI: 10.1007/s11071-016-2678-4. URL: <http://dx.doi.org/10.1007/s11071-016-2678-4>.
- [14] V. I. Oseledec. «A multiplicative ergodic theorem. Lyapunov characteristic exponents of dynamical systems». En: *Transactions of the Moscow Mathematical Society* 19 (1968), págs. 197-231.
- [15] P. Cvitanović y col. *Chaos: Classical and Quantum*. Disponible en <http://ChaosBook.org>. Niels Bohr Institute, Copenhagen, 2012. URL: <https://www.mathnet.ru/eng/mmo214>.
- [16] Ya B Pesin. «CHARACTERISTIC LYAPUNOV EXPONENTS AND SMOOTH ERGODIC THEORY». En: *Russian Mathematical Surveys* 32.4 (ago. de 1977), pág. 55. DOI: 10.1070/RM1977v032n04ABEH001639. URL: <https://dx.doi.org/10.1070/RM1977v032n04ABEH001639>.
- [17] Giancarlo Benettin y col. «Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems - A method for computing all of them. I - Theory. II - Numerical application». En: *Meccanica* 15 (mar. de 1980), págs. 21-30. DOI: 10.1007/BF02128236.
- [18] A. Wolf y col. «Determining Lyapunov exponents from a time series». En: *Physica D: Nonlinear Phenomena* 16.3 (jul. de 1985), págs. 285-317. DOI: 10.1016/0167-2789(85)90011-9. URL: <http://chaos.utexas.edu/manuscripts/1085774778.pdf>.
- [19] B. Quarles y col. «The instability transition for the restricted 3-body problem: III. The Lyapunov exponent criterion». En: *Astronomy & Astrophysics* 533 (ago. de 2011), A2. ISSN: 1432-0746. DOI: 10.1051/0004-6361/201016199. URL: <http://dx.doi.org/10.1051/0004-6361/201016199>.

- [20] Gennady Leonov y Nikolay Kuznetsov. «TIME-VARYING LINEARIZATION AND THE PERRON EFFECTS». En: *International Journal of Bifurcation and Chaos* 17 (abr. de 2007), págs. 1079-1107. DOI: 10.1142/S0218127407017732.
- [21] James L. Kaplan y James A. Yorke. «Chaotic behavior of multidimensional difference equations». En: *Functional Differential Equations and Approximation of Fixed Points*. Ed. por Heinz-Otto Peitgen y Hans-Otto Walther. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979, págs. 204-227. ISBN: 978-3-540-35129-0.
- [22] N.V. Kuznetsov. «The Lyapunov dimension and its estimation via the Leonov method». En: *Physics Letters A* 380.25–26 (jun. de 2016), págs. 2142-2149. ISSN: 0375-9601. DOI: 10.1016/j.physleta.2016.04.036. URL: <http://dx.doi.org/10.1016/j.physleta.2016.04.036>.
- [23] Louis M. Pecora y col. «Fundamentals of synchronization in chaotic systems, concepts, and applications.» En: *Chaos* 7 4 (1997), págs. 520-543. URL: <https://api.semanticscholar.org/CorpusID:17209888>.
- [24] I. Shimada y T. Nagashima. «A Numerical Approach to Ergodic Problem of Dissipative Dynamical Systems». En: *Progress of Theoretical Physics* 61.6 (jun. de 1979), págs. 1605-1616. DOI: 10.1143/PTP.61.1605.
- [25] Jack Wisdom y Matthew Holman. «Symplectic maps for the N-body problem.» En: *The Astronomical Journal* 102 (oct. de 1991), págs. 1528-1538. DOI: 10.1086/115978.
- [26] T.J. Stuchi. «Symplectic integrators revisited». En: *Brazilian Journal of Physics* (2002). ISSN: 0103-9733. DOI: <https://doi.org/10.1590/S0103-97332002000500022>. URL: <https://www.scielo.br/j/bjp/a/NgqxKCKkBmrC93BgP4FJ4fk/?lang=en#>.
- [27] Siu A. Chin y C. R. Chen. «Forward Symplectic Integrators for Solving Gravitational Few-Body Problems». En: *Celestial Mechanics and Dynamical Astronomy* 91.3 (2005), págs. 301-322. ISSN: 1572-9478. DOI: 10.1007/s10569-004-4622-z. URL: <https://doi.org/10.1007/s10569-004-4622-z>.
- [28] David M Hernandez y Matthew J Holman. «`jscpj_encke_hhj/scpj`: an integrator for gravitational dynamics with a dominant mass that achieves optimal error behaviour». En: *Monthly Notices of the Royal Astronomical Society* 502.1 (dic. de 2020), págs. 556-563. ISSN: 1365-2966. DOI: 10.1093/mnras/staa3945. URL: <http://dx.doi.org/10.1093/mnras/staa3945>.
- [29] Will M. Farr y Edmund Bertschinger. «Variational Integrators for the GravitationalN-Body Problem». En: *The Astrophysical Journal* 663.2 (jul. de 2007), págs. 1420-1433. ISSN: 1538-4357. DOI: 10.1086/518641. URL: <http://dx.doi.org/10.1086/518641>.
- [30] Haruo Yoshida. «Recent progress in the theory and application of symplectic integrators». En: *Celestial Mechanics and Dynamical Astronomy* 56.1 (1993), págs. 27-43. ISSN: 1572-9478. DOI: 10.1007/BF00699717. URL: <https://doi.org/10.1007/BF00699717>.

- [31] D. M. Hernandez y E. Bertschinger. «Symplectic integration for the collisional gravitational N-body problem». En: *Monthly Notices of the Royal Astronomical Society* 452.2 (2015), págs. 1934-1944. DOI: 10.1093/mnras/stv1439.
- [32] Haruo Yoshida. «Construction of higher order symplectic integrators». En: *Physics Letters A* 150.5 (1990), págs. 262-268. ISSN: 0375-9601. DOI: [https://doi.org/10.1016/0375-9601\(90\)90092-3](https://doi.org/10.1016/0375-9601(90)90092-3). URL: <https://www.sciencedirect.com/science/article/pii/0375960190900923>.
- [33] Hanno Rein, David S. Spiegel y David Tamayo. *REBOUND Documentation and Integrators*. Conceptual reference to REBOUND documentation and related papers. See H. Rein and D. S. Spiegel, "IAS15: A fast, adaptive, high-order integrator for gravitational dynamics, accurate to machine precision," MNRAS, vol. 446, pp. 1424–1437, Jan. 2015; H. Rein and D. Tamayo, "WHFAST: A fast and unbiased implementation of the Wisdom-Holman algorithm for long-term gravitational simulations," MNRAS, vol. 452, pp. 376–388, Sep. 2015. 2025. URL: <https://rebound.readthedocs.io/en/latest/integrators/>.
- [34] David M Hernandez. «Should N-body integrators be symplectic everywhere in phase space?» En: *Monthly Notices of the Royal Astronomical Society* 486.4 (abr. de 2019), págs. 5231-5238. ISSN: 1365-2966. DOI: 10.1093/mnras/stz884. URL: <http://dx.doi.org/10.1093/mnras/stz884>.
- [35] Alexander S. Glasser y Hong Qin. «A gauge-compatible Hamiltonian splitting algorithm for particle-in-cell simulations using finite element exterior calculus». En: *Journal of Plasma Physics* 88.2 (2022), pág. 835880202. DOI: 10.1017/S0022377822000290.
- [36] Xiongbiao Tu, Qiao Wang y Yifa Tang. *Symplectic Integrators in Corotating Coordinates*. 2022. arXiv: 2202.03779 [astro-ph.IM]. URL: <https://arxiv.org/abs/2202.03779>.
- [37] Hanno Rein y Daniel Tamayo. «whfast: a fast and unbiased implementation of a symplectic Wisdom–Holman integrator for long-term gravitational simulations». En: *Monthly Notices of the Royal Astronomical Society* 452.1 (jul. de 2015), págs. 376-388. ISSN: 0035-8711. DOI: 10.1093/mnras/stv1257. eprint: <https://academic.oup.com/mnras/article-pdf/452/1/376/4912268/stv1257.pdf>. URL: <https://doi.org/10.1093/mnras/stv1257>.
- [38] Ernst Hairer, Christian Lubich y Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. 2.^a ed. Springer Series in Computational Mathematics. Springer Berlin, Heidelberg, 2006. ISBN: 978-3-540-30666-5. DOI: 10.1007/3-540-30666-8. URL: <https://link.springer.com/book/10.1007/3-540-30666-8>.
- [39] Hiroshi Kinoshita, Haruo Yoshida e Hiroshi Nakai. «Symplectic integrators and their application to dynamical astronomy». En: *Celestial Mechanics and Dynamical Astronomy* 50.1 (1990), págs. 59-71. ISSN: 1572-9478. DOI: 10.1007/BF00048986. URL: <https://doi.org/10.1007/BF00048986>.

- [40] J. Wisdom. «The resonance overlap criterion and the onset of stochastic behavior in the restricted three-body problem». En: *The Astronomical Journal* 85.8 (1980), págs. 1122-1133. ISSN: 0004-6256. DOI: 10.1086/112778. URL: <https://ui.adsabs.harvard.edu/abs/1980AJ.....85.1122W>.
- [41] REBOUND documentation. *Integrators - WHFast*. <https://rebound.readthedocs.io/en/latest/integrators/>. Accessed: Abril 04, 2025. 2025.
- [42] J. Wisdom, M. Holman y J. Touma. «Symplectic Correctors». En: *Fields Institute Communications* 10 (ene. de 1996), pág. 217.
- [43] Pejvak Javaheri, Hanno Rein y Dan Tamayo. «WHFast512: A symplectic N-body integrator for planetary systems optimized with AVX512 instructions». En: *The Open Journal of Astrophysics* 6 (jul. de 2023). ISSN: 2565-6120. DOI: 10.21105/astro.2307.05683. URL: <http://dx.doi.org/10.21105/astro.2307.05683>.
- [44] REBOUND documentation. *Advanced settings for WHFast: Extra speed, accuracy, and additional forces*. https://rebound.readthedocs.io/en/latest/ipython_examples/AdvWHFast/. Accessed: Abril 04, 2025. 2025.
- [45] REBOUND documentation. *WHFast tutorial*. https://rebound.readthedocs.io/en/latest/ipython_examples/WHFast/. Accessed: Abril 04, 2025. 2025.
- [46] L Greengard y V Rokhlin. «A fast algorithm for particle simulations». En: *Journal of Computational Physics* 73.2 (1987), págs. 325-348. ISSN: 0021-9991. DOI: [https://doi.org/10.1016/0021-9991\(87\)90140-9](https://doi.org/10.1016/0021-9991(87)90140-9). URL: <https://www.sciencedirect.com/science/article/pii/0021999187901409>.
- [47] Per-Gunnar Martinsson. *Encyclopedia entry on “Fast Multipole Methods”*. Ago. de 2012. URL: https://amath.colorado.edu/faculty/martinss/2014_CBMS/Refs/2012_fmm_encyclopedia.pdf.
- [48] Barry A. Cipra. «The Best of the 20th Century: Editors Name Top 10 Algorithms». En: *SIAM News* 33.4 (feb. de 2000). URL: https://web.tecnico.ulisboa.pt/~mcasquilho/CD_Casquilho/PRINT/NotebookSC_1LP_22pp.pdf.
- [49] Rick Beatson y Leslie Greengard. *A short course on fast multipole methods*. Short course. 1997. URL: https://math.nyu.edu/~greengar/shortcourse_fmm.pdf.
- [50] Long Chen. *Introduction to Fast Multipole Methods*. Unpublished manuscript or notes. URL: <https://www.math.uci.edu/~chenlong/MathPKU/FMMsimple.pdf>.
- [51] Lexing Ying. «A pedestrian introduction to fast multipole methods». En: *Science China Mathematics* 55.5 (2012), págs. 1043-1051. ISSN: 1869-1862. DOI: 10.1007/s11425-012-4392-0. URL: <https://doi.org/10.1007/s11425-012-4392-0>.

- [52] Andrew W. Appel. «An Efficient Program for Many-Body Simulation». En: *SIAM Journal on Scientific and Statistical Computing* 6.1 (1985), págs. 85-103. DOI: 10.1137/0906008. eprint: <https://doi.org/10.1137/0906008>. URL: <https://doi.org/10.1137/0906008>.
- [53] Josh Barnes y Piet Hut. «A hierarchical O($N \log N$) force-calculation algorithm». En: *Nature* 324.6096 (1986), págs. 446-449. ISSN: 1476-4687. DOI: 10.1038/324446a0. URL: <https://doi.org/10.1038/324446a0>.
- [54] H. Cheng, L. Greengard y V. Rokhlin. «A Fast Adaptive Multipole Algorithm in Three Dimensions». En: *Journal of Computational Physics* 155.2 (1999), págs. 468-498. ISSN: 0021-9991. DOI: <https://doi.org/10.1006/jcph.1999.6355>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999199963556>.
- [55] V Rokhlin. «Rapid solution of integral equations of scattering theory in two dimensions». En: *Journal of Computational Physics* 86.2 (1990), págs. 414-439. ISSN: 0021-9991. DOI: [https://doi.org/10.1016/0021-9991\(90\)90107-C](https://doi.org/10.1016/0021-9991(90)90107-C). URL: <https://www.sciencedirect.com/science/article/pii/002199919090107C>.
- [56] Björn Engquist y Lexing Ying. «A Fast Directional Algorithm for High Frequency Acoustic Scattering in Two Dimensions». En: *Communications in Mathematical Sciences* 7.2 (2009), págs. 327-345.
- [57] J. Carrier, L. Greengard y V. Rokhlin. «A Fast Adaptive Multipole Algorithm for Particle Simulations». En: *SIAM Journal on Scientific and Statistical Computing* 9.4 (1988), págs. 669-686. DOI: 10.1137/0909044. eprint: <https://doi.org/10.1137/0909044>. URL: <https://doi.org/10.1137/0909044>.
- [58] Lexing Ying, George Biros y Denis Zorin. «A kernel-independent adaptive fast multipole algorithm in two and three dimensions». En: *Journal of Computational Physics* 196.2 (2004), págs. 591-626. ISSN: 0021-9991. DOI: <https://doi.org/10.1016/j.jcp.2003.11.021>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999103006090>.
- [59] Walter Dehnen. «A Hierarchical O(N) Force Calculation Algorithm». En: *Journal of Computational Physics* 179.1 (2002), págs. 27-42. ISSN: 0021-9991. DOI: <https://doi.org/10.1006/jcph.2002.7026>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999102970269>.
- [60] Rio Yokota y Lorena A Barba. «A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems». En: *The International Journal of High Performance Computing Applications* 26.4 (2012), págs. 337-346. DOI: 10.1177/1094342011429952. eprint: <https://doi.org/10.1177/1094342011429952>. URL: <https://doi.org/10.1177/1094342011429952>.
- [61] Josh Barnes y Piet Hut. «A hierarchical $O(N \log N)$ force-calculation algorithm». En: *Nature* 324.6096 (1986), págs. 446-449. ISSN: 1476-4687. DOI: 10.1038/324446a0. URL: <https://doi.org/10.1038/324446a0>.

- [62] John Dubinski. «A parallel tree code». En: *New Astronomy* 1.2 (1996), págs. 133-147. ISSN: 1384-1076. DOI: [https://doi.org/10.1016/S1384-1076\(96\)00009-7](https://doi.org/10.1016/S1384-1076(96)00009-7). URL: <https://www.sciencedirect.com/science/article/pii/S1384107696000097>.
- [63] John K. Salmon. «Parallel Hierarchical N-body Methods». Advisors: Geoffrey C. Fox, Thomas A. Prince. Division: Physics, Mathematics and Astronomy. Publicly available. Dissertation (Ph.D.) Pasadena, CA: California Institute of Technology, 1991. DOI: 10.7907/3J5V-VR08. URL: <https://resolver.caltech.edu/CaltechTHESIS:04112011-113813016>.
- [64] Joshua E. Barnes. «A modified tree code: Don't laugh; It runs». En: *Journal of Computational Physics* 87.1 (1990), págs. 161-170. ISSN: 0021-9991. DOI: [https://doi.org/10.1016/0021-9991\(90\)90232-P](https://doi.org/10.1016/0021-9991(90)90232-P). URL: <https://www.sciencedirect.com/science/article/pii/002199919090232P>.
- [65] Susanne Pfalzner y Paul Gibbon. *Many-Body Tree Methods in Physics*. Cambridge University Press, 1996. ISBN: 9780511529368. DOI: <https://doi.org/10.1017/CBO9780511529368>. URL: <https://www.cambridge.org/core/books/manybody-tree-methods-in-physics/74A783F229C08A4BE8D2982FA0F58960>.
- [66] Maia Aguirre Pascual de Zulueta. «Estudio e implementación del algoritmo Barnes-Hut para el cálculo de la interacción gravitatoria entre N-cuerpos». Bachelor's thesis. Universidad del País Vasco, dic. de 2020. URL: <https://addi.ehu.es/handle/10810/49069>.
- [67] Badri Munier y col. «On the parallelization and performance analysis of Barnes–Hut algorithm using Java parallel platforms». En: *SN Applied Sciences* 2.4 (mar. de 2020), pág. 601. ISSN: 2523-3971. DOI: 10.1007/s42452-020-2386-z. URL: <https://doi.org/10.1007/s42452-020-2386-z>.
- [68] U Becciani y col. «A parallel tree code for large N-body simulation: dynamic load balance and data distribution on a CRAY T3D system». En: *Computer Physics Communications* 106.1 (1997), págs. 105-113. ISSN: 0010-4655. DOI: [https://doi.org/10.1016/S0010-4655\(97\)00102-1](https://doi.org/10.1016/S0010-4655(97)00102-1). URL: <https://www.sciencedirect.com/science/article/pii/S0010465597001021>.
- [69] U. Becciani, V. Antonuccio-Delogu y M. Gambera. «A Modified Parallel Tree Code for N-Body Simulation of the Large-Scale Structure of the Universe». En: *Journal of Computational Physics* 163.1 (2000), págs. 118-132. ISSN: 0021-9991. DOI: <https://doi.org/10.1006/jcph.2000.6557>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999100965574>.
- [70] Martin Burtscher y Keshav Pingali. «Chapter 6 - An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm». En: *GPU Computing Gems Emerald Edition*. Ed. por Wen-mei W. Hwu. Applications of GPU Computing Series. Boston: Morgan Kaufmann, 2011, págs. 75-92. ISBN: 978-0-12-384988-5. DOI: <https://doi.org/10.1016/B978-0-12-384988-5.00006-1>. URL: <https://www.sciencedirect.com/science/article/pii/B9780123849885000061>.

- [71] Tsuyoshi Hamada y col. «A novel multiple-walk parallel algorithm for the Barnes–Hut treecode on GPUs – towards cost effective, high performance N-body simulation». En: *Computer Science - Research and Development* 24.1 (sep. de 2009), págs. 21-31. ISSN: 1865-2042. DOI: 10.1007/s00450-009-0089-1. URL: <https://doi.org/10.1007/s00450-009-0089-1>.
- [72] John K. Salmon y Michael S. Warren. «Skeletons from the Treecode Closet». En: *Journal of Computational Physics* 111.1 (1994), págs. 136-155. ISSN: 0021-9991. DOI: <https://doi.org/10.1006/jcph.1994.1050>. URL: <https://www.sciencedirect.com/science/article/pii/S0021999184710503>.
- [73] J. S. Bagla. *Cosmological N-Body simulation: Techniques, Scope and Status*. 2004. arXiv: astro-ph/0411043 [astro-ph]. URL: <https://arxiv.org/abs/astro-ph/0411043>.
- [74] Laurens van der Maaten y Geoffrey Hinton. «Visualizing Data using t-SNE». En: *Journal of Machine Learning Research* 9.86 (2008), págs. 2579-2605. URL: <http://jmlr.org/papers/v9/vandermaaten08a.html>.
- [75] Laurens van der Maaten. *Barnes-Hut-SNE*. 2013. arXiv: 1301.3342 [cs.LG]. URL: <https://arxiv.org/abs/1301.3342>.
- [76] ZeCheng Gan y ZhenLi Xu. «Efficient implementation of the Barnes-Hut octree algorithm for Monte Carlo simulations of charged systems». En: *Science China Mathematics* 57.7 (jul. de 2014), págs. 1331-1340. ISSN: 1869-1862. DOI: 10.1007/s11425-014-4783-5. URL: <https://doi.org/10.1007/s11425-014-4783-5>.
- [77] Mathias Winkel y col. «A massively parallel, multi-disciplinary Barnes–Hut tree code for extreme-scale N-body simulations». En: *Computer Physics Communications* 183.4 (2012), págs. 880-889. ISSN: 0010-4655. DOI: <https://doi.org/10.1016/j.cpc.2011.12.013>. URL: <https://www.sciencedirect.com/science/article/pii/S0010465511004012>.
- [78] Hubert Nguyen, ed. *GPU Gems 3*. Addison-Wesley Professional, 2007. ISBN: 978-0321515261. URL: <https://developer.nvidia.com/gpugems/gpugems3/contributors>.
- [79] S. M. Sinha. *Mathematical Programming: Theory and Methods*. New Delhi, India: Elsevier (A Division of Reed Elsevier India Pvt. Ltd.), 2006, págs. xv, 572. ISBN: 81-312-0376-X. URL: <https://www.sciencedirect.com/book/9788131203767/mathematical-programming>.
- [80] David G. Luenberger y Yinyu Ye. *Linear and Nonlinear Programming*. 2nd. Reimpresión 2003 por Springer US. Reading, MA, USA: Addison-Wesley Publishing Company, 1984, pág. 491. ISBN: 0201157942.
- [81] Stephen P. Bradley, Arnoldo C. Hax y Thomas L. Magnanti. *Applied Mathematical Programming*. Reading, MA, USA: Addison-Wesley Publishing Company, 1977, pág. 716. ISBN: 978-0-201-00464-9. URL: https://books.google.com/books/about/Applied_Mathematical_Programming.html?id=MSWdWv3Gn5cC.

- [82] Dimitri P. Bertsekas. *Nonlinear Programming*. 2nd. Belmont, MA, USA: Athena Scientific, 1999, pág. 780. ISBN: 1886529000. URL: <https://vlsicad.eecs.umich.edu/BK/Slots/cache/www.athenasc.com/nonlinbook.html>.
- [83] Jorge Nocedal y Stephen J. Wright. *Numerical Optimization*. New York, NY, USA: Springer Science+Business Media, 2006, págs. xxii, 664. ISBN: 0-387-30303-0. DOI: 10.1007/978-0-387-40065-5. URL: <https://link.springer.com/book/10.1007/978-0-387-40065-5>.
- [84] Stephen P. Boyd y Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004, págs. xxii, 716. ISBN: 0-521-83378-7. URL: https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.
- [85] Stephen P. Bradley, Arnoldo C. Hax y Thomas L. Magnanti. «Capítulo 9». En: *Applied Mathematical Programming*. Reading, MA, USA: Addison-Wesley Publishing Company, 1977. ISBN: 020100464X.
- [86] Stephen P. Bradley, Arnoldo C. Hax y Thomas L. Magnanti. «Capítulo 11». En: *Applied Mathematical Programming*. Reading, MA, USA: Addison-Wesley Publishing Company, 1977. ISBN: 020100464X.
- [87] Stephen Boyd y Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004. ISBN: 978-0521833783.
- [88] Stephen Boyd y Lieven Vandenberghe. *Convex Optimization – Slides*. Presentation slides. Presentation slides accompanying the book "Convex Optimization". Revised by Stephen Boyd, Lieven Vandenberghe, and Parth Nobel. Version date on title page of PDF: 2023. 2023.
- [89] Aaron Sidford. *MS&E 213 / CS 269O : Bonus Chapter 1 Feasibility Problem and Cutting Plane Methods*. Course Notes, Stanford University. Document dated November 17, 2020. Nov. de 2020.
- [90] Arne E. Eiben y James E. Smith. *Introduction to Evolutionary Computing*. English. 2.^a ed. Natural Computing Series. Second edition. Includes 55 b/w and 12 colour illustrations. Part of the Springer Computer Science package. © Springer-Verlag GmbH Germany, part of Springer Nature 2015. Berlin, Heidelberg: Springer, 2015, págs. XII + 287. ISBN: 978-3-662-44873-1. DOI: 10.1007/978-3-662-44874-8. URL: <https://doi.org/10.1007/978-3-662-44874-8>.
- [91] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.
- [92] Julian Blank y Kalyanmoy Deb. «pymoo: Multi-Objective Optimization in Python». En: *IEEE Access* 8 (2020). Introduces pymoo, a Python framework for multi-objective optimization., págs. 89497-89509. DOI: 10.1109/ACCESS.2020.2990567.
- [93] Kalyanmoy Deb. «Multi-Objective Optimization». En: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. por Edmund K. Burke y Graham Kendall. Boston, MA: Springer US, 2005, págs. 273-316. ISBN: 978-0-387-28356-2. DOI: 10.1007/0-387-28356-0_10. URL: https://doi.org/10.1007/0-387-28356-0_10.

- [94] Kalyanmoy Deb y Ram Bhushan Agrawal. «Simulated Binary Crossover for Continuous Search Space». En: *Complex Systems* 9.2 (1995). Introduces the SBX crossover operator for real-coded genetic algorithms., págs. 115-148.
- [95] Kalyanmoy Deb y Mohit Goyal. «A combined genetic adaptive search (GeneAS) for engineering design». En: *Computer Science and Informatics* 26.4 (1996). Proposes the GeneAS algorithm combining binary and real-coded genetic algorithms., págs. 30-45. URL: <https://api.semanticscholar.org/CorpusID:18387364>.
- [96] Qt Group. *Qt Group Frequently Asked Questions*. Qt Group page, last edited May 1, 2025. 2025. URL: <https://www.qt.io/faq>.
- [97] The Qt Company. *Qt Framework: Development Framework for Cross-platform Applications*. Official Qt product page. 2025. URL: <https://www.qt.io/product/framework>.
- [98] BusinessCloud staff. *Introduction to Qt development: A beginner's guide*. BusinessCloud news article, posted April 24, 2025. 2025. URL: <https://businesscloud.co.uk/news/introduction-to-qt-development-a-beginners-guide/>.
- [99] Ivan Kubara. *What is Qt framework, Why to Use It, and How?* Lemberg Solutions blog post (Embedded Engineering). 2023. URL: <https://lembertsolutions.com/blog/why-use-qt-framework>.
- [100] Qt Wiki contributors. *Qt History*. Qt Wiki page (last edited August 14, 2024). 2024. URL: https://wiki.qt.io/Qt_History.
- [101] Trolltech (archived). *Signals and Slots*. Archived Qt 3.1.2 documentation (Trolltech, 2003). 2003. URL: https://web.mit.edu/~firebird/arch/sun4x_58/doc/html/signalsandslots.html.
- [102] TutorialsPoint. *PyQt Tutorial*. Online tutorial (TutorialsPoint). 2025. URL: <https://www.tutorialspoint.com/pyqt/index.htm>.
- [103] The Qt Company. *Qt for Python: Design GUI with Python (Python Bindings for Qt)*. Official Qt page on Python bindings (PySide6). 2025. URL: <https://www.qt.io/qt-for-python>.
- [104] The Qt Company. *Qt Use Cases*. Official Qt page on industry use cases. 2025. URL: <https://www.qt.io/use-cases>.
- [105] Ontosight AI. *Introduction to PyQt Basics*. OntoSight AI glossary entry. 2024. URL: <https://ontosight.ai/glossary/term/introduction-to-pyqt-basics>.
- [106] The Qt Company. *Qt Quick Module*. Qt 6.9 Documentation (Qt Quick overview). 2025. URL: <https://doc.qt.io/qt-6/qtquick-index.html>.
- [107] Ellie Yantsan. «What is Qt – Key Features of Qt Development». En: *IT Supply Chain (Industry Talk)* (2024). Industry blog post. URL: <https://itsupplychain.com/what-is-qt-key-features-of-qt-development/>.
- [108] GeeksforGeeks. *Python GUI – PyQt vs. Tkinter*. GeeksforGeeks tutorial (last updated Dec 11, 2020). 2020. URL: <https://www.geeksforgeeks.org/python-gui-pyqt-vs-tkinter/>.

- [109] Jake VanderPlas. *Python Data Science Handbook: Essential Tools for Working with Data*. Accessed via the O'Reilly learning platform. Released November 2016. Sebastopol, CA: O'Reilly Media, Inc., 2016. ISBN: 9781491912058. URL: <https://www.oreilly.com/library/view/python-data-science/9781491912126/ch04.html> (visitado 12-05-2025).
- [110] John D. Hunter. «Matplotlib: A 2D Graphics Environment». En: *Computing in Science 'I&E' Engineering* 9.3 (2007), págs. 90-95. DOI: 10.1109/MCSE.2007.55.
- [111] Matplotlib Development Team. *Matplotlib – Visualization with Python*. 2025. URL: <https://matplotlib.org/> (visitado 12-05-2025).
- [112] ActiveState. *What Is Matplotlib In Python? How to use it for plotting?* 2025. URL: <https://www.activestate.com/resources/quick-reads/what-is-matplotlib-in-python-how-to-use-it-for-plotting/> (visitado 12-05-2025).
- [113] GeeksforGeeks. *Scientific Computing with Python*. 2025. URL: <https://www.geeksforgeeks.org/scientific-computing-with-python/> (visitado 12-05-2025).
- [114] Matplotlib Development Team. *History – Matplotlib 3.10.3 documentation*. 2025. URL: <https://matplotlib.org/stable/project/history.html> (visitado 12-05-2025).
- [115] Eric Jones, Travis Oliphant, Pearu Peterson y col. «matplotlib». En: *The Architecture of Open Source Applications, Volume II*. Ed. por Amy Brown y Greg Wilson. Accessed via AOSA website. Published March 30, 2012. Raleigh, NC: Lulu.com, 2012, pág. 390. ISBN: 9781105571817. URL: <https://aosabook.org/en/v2/matplotlib.html> (visitado 12-05-2025).
- [116] desosa 2021. *matplotlib - From Vision to Architecture*. 2021. URL: <https://2021.desosa.nl/projects/matplotlib/posts/essay2-vision-to-architecture/> (visitado 12-05-2025).
- [117] Matplotlib Development Team. *Matplotlib Architecture - YouTube*. 2025. URL: <https://www.youtube.com/watch?v=1pyj7x7SI8Y> (visitado 12-05-2025).
- [118] delftswa. *Matplotlib - The Python 2D Plotting Library*. 2017. URL: <https://delftswa.gitbooks.io/desosa-2017/content/matplotlib/chapter.html> (visitado 12-05-2025).
- [119] Matplotlib Development Team. *Pyplot tutorial – Matplotlib 3.10.3 documentation*. 2025. URL: <https://matplotlib.org/stable/tutorials/pyplot.html> (visitado 12-05-2025).
- [120] Reddit user discussion. *Matplotlib sucks : r/bioinformatics*. Reflects user perception, not necessarily an academic source, but indicates common beginner issues. 2022. URL: https://www.reddit.com/r/bioinformatics/comments/t13510/matplotlib_sucks/ (visitado 12-05-2025).
- [121] GeeksforGeeks. *Introduction to Matplotlib*. 2025. URL: <https://www.geeksforgeeks.org/python-introduction-matplotlib/> (visitado 12-05-2025).

- [122] Trung Vu y col. «Mastering data visualization with Python: practical tips for researchers». En: *PeerJ Comput Sci* 9 (ene. de 2023), e1203. DOI: 10.7717/peerj-cs.1203. URL: <https://pmc.ncbi.nlm.nih.gov/articles/PMC10728683/> (visitado 12-05-2025).
- [123] Yury Zhauniarovich. *Matplotlib Graphs in Research Papers*. 2022. URL: <https://zhauniarovich.com/post/2022/2022-09-matplotlib-graphs-in-research-papers/> (visitado 12-05-2025).
- [124] llogo.dev. *Interactive Plots with Matplotlib Animations in Python*. 2023. URL: <https://llogo.dev/posts/interactive-plots-matplotlib-animations-python/> (visitado 12-05-2025).
- [125] SciPy Lecture Notes Team. *1.1. Python scientific computing ecosystem*. 2025. URL: <https://scipy-lectures.org/intro/intro.html> (visitado 12-05-2025).
- [126] Michael Waskom. «Seaborn: statistical data visualization». En: *Journal of Open Source Software* 6.60 (abr. de 2021), pág. 3021. DOI: 10.21105/joss.03021.
- [127] The Event Horizon Telescope Collaboration y col. «First M87 Event Horizon Telescope Results. I. The Shadow of the Supermassive Black Hole». En: *The Astrophysical Journal Letters* 875.1 (abr. de 2019). Matplotlib was used in the analysis pipeline, though not explicitly cited in this specific paper usually, it's a known tool used by the collaboration, pág. L1. DOI: 10.3847/2041-8213/ab0ec7.
- [128] Rubin H. Landau, Manuel J. Páez y Cristian C. Bordeianu. *Computational Physics: Problem Solving with Python*. 3rd. Representative textbook using Python/Matplotlib for physics examples. Wiley-VCH, 2015.
- [129] Real Python. *Exploring Astrophysics in Python With pandas and Matplotlib*. 2025. URL: <https://realpython.com/courses/astrophysics-pandas-matplotlib/> (visitado 12-05-2025).
- [130] arpmay. *N-body-Problem*. Example of application. 2025. URL: <https://github.com/arpmay/N-body-Problem> (visitado 12-05-2025).
- [131] John Garrett y col. *garrettj403/SciencePlots: Matplotlib styles for scientific plotting*. 2025. URL: <https://github.com/garrettj403/SciencePlots> (visitado 12-05-2025).
- [132] Matplotlib Development Team. *Quick start guide – Matplotlib 3.10.3 documentation*. 2025. URL: https://matplotlib.org/stable/users/explain/quick_start.html (visitado 12-05-2025).
- [133] Reddit user discussion. *Unpopular opinion: Matplotlib is a bad library : r/Python*. Reflects user perception. 2022. URL: https://www.reddit.com/r/Python/comments/u8j6fn/unpopular_opinion_matplotlib_is_a_bad_library/ (visitado 12-05-2025).
- [134] Stack Overflow discussion. *gnuplot vs Matplotlib*. Community discussion on performance. 2009. URL: <https://stackoverflow.com/questions/911655/gnuplot-vs-matplotlib> (visitado 12-05-2025).

- [135] GeeksforGeeks. *Data Visualization using Matplotlib in Python*. 2025. URL: <https://www.geeksforgeeks.org/data-visualization-using-matplotlib/> (visitado 12-05-2025).
- [136] Matplotlib Development Team. *Customizing Matplotlib with style sheets and rcParams*. 2025. URL: <https://matplotlib.org/stable/users/explain/customizing.html> (visitado 12-05-2025).
- [137] Stephen Boyd y Lieven Vandenberghe. *Convex Optimization*. Accessed: 2025-05-12. Cambridge: Cambridge University Press, 2004. ISBN: 9780521833783. URL: <https://www.cambridge.org/gb/universitypress/subjects/statistics-probability/optimization-or-and-risk/convex-optimization?format=HB>.