

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“ANÁLISIS DEL MÉTODO DE DERIVACIÓN DE PASO
COMPLEJO COMO ALTERNATIVA DE MITIGACIÓN DE
ERRORES DE CANCELACIÓN EN LA RESOLUCIÓN DE
SISTEMAS DE ECUACIONES NO LINEALES.”

NICOLÁS ALONZO CASTRO ESPINOSA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Claudio Torres López, Ph.D.
Profesor Correferente: Roberto León, Ph.D.

Enero - 2025

DEDICATORIA

Dedico este logro a mis padres, a mi familia, a mis amigos y a todos aquellos que me han acompañado durante este duro año, agradeciendo profundamente el apoyo y la confianza que me brindaron.

AGRADECIMIENTOS

Quiero expresar mi más profundo agradecimiento a todas las personas que, de una u otra manera, han sido fundamentales en este proceso.

En primer lugar, agradezco a los profesores de la Universidad Federico Santa María, cuya formación ha sido esencial para llegar a este logro. En particular, agradezco al profesor Claudio Torres por su apoyo incondicional en este proceso, y también a incontables otros profesores que han dejado una huella importante en mi desarrollo académico y personal. Cada uno de ellos me ha brindado herramientas valiosas para enfrentar los desafíos que se presentaron a lo largo de este camino.

Agradezco a mi familia, que ha sido mi pilar a lo largo de todo este camino. Su comprensión, amor y apoyo constante me han permitido mantenerme firme, incluso en los momentos más difíciles. A mis amigos, quienes, a pesar de la distancia, siempre han estado a mi lado. La distancia no nos separa, y su compañía y motivación han sido fundamentales para seguir adelante.

Este logro no es solo mío, sino que es compartido por cada persona que, con su apoyo, amor y enseñanzas, me ha ayudado a transformarme en quien soy el día de hoy.

Powered@NLHPC: Esta investigación/tesis fue parcialmente apoyada por la infraestructura de supercómputo del NLHPC (CCSS210001)

RESUMEN

Resumen— Al desarrollar algoritmos, es fundamental garantizar tanto una alta precisión en los resultados como un tiempo de cómputo eficiente, ya que estos criterios determinan su viabilidad práctica. Por ello, resulta esencial diversificar las herramientas disponibles para su diseño. En este contexto, se propone analizar el método de aproximación de derivadas mediante paso complejo, aplicado a la resolución de sistemas de ecuaciones mediante el método de Newton-Krylov. El método de paso complejo se selecciona debido a su mayor robustez frente a la pérdida de precisión numérica, en comparación con métodos tradicionales como las diferencias finitas.

Palabras Clave— *Aproximación de paso complejo; Diferencias Finitas; Aproximación de paso multicomplejo; GMRES; LGMRES; Método de Newton; Método de Newton-Krylov;*

ABSTRACT

Abstract— When developing algorithms, it is crucial to ensure both high precision in the results and efficient computation time, as these criteria determine their practical viability. Therefore, it is essential to diversify the available tools for their design. In this context, we propose analyzing the method of derivative approximation using the complex step method, applied to the solution of systems of equations using the Newton-Krylov method. The complex step method is chosen due to its greater robustness against numerical precision loss, compared to traditional methods such as finite differences.

Keywords— *Complex step approximation; Finite Differences; Multicomplex step approximation; GMRES; LGMRES; Newton method; Newton-Krylov method;*

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	X
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Contextualización y Relevancia de la problemática	2
1.1.1 El Rol de las Derivadas en Cálculos Numéricos	2
1.1.2 Impacto de los Errores Numéricos en Aplicaciones Prácticas	2
1.2 Limitaciones de los Métodos Convencionales	3
1.2.1 Diferencias Finitas y Sus Restricciones	3
1.2.2 Errores de Cancelación: Origen y Consecuencias	4
1.3 Objetivos del Análisis	4
1.3.1 Problemas Clave a Abordar	4
1.3.2 Planteamiento del Método Propuesto	5
1.4 Justificación del Enfoque Propuesto	5
1.4.1 Ventajas del Método de Paso Complejo	5
1.4.2 Comparación Inicial con Otros Métodos Alternativos	5
1.5 Problemas de Malas Aproximaciones de Derivadas en Métodos Iterativos	6
1.5.1 Impacto de las Malas Aproximaciones en el Método GMRES	6
1.5.2 Espacios de Krylov y Errores en la Aproximación de Derivadas	7
1.5.3 Posibles Soluciones y Mejoras	7
CAPÍTULO 2: MARCO CONCEPTUAL	8
2.1 Análisis Numérico y Álgebra Computacional	8
2.1.1 Distinción entre Análisis Numérico y Álgebra Computacional	8
2.1.2 Rol de los Métodos Numéricos en la Solución de Problemas Matemáticos	8
2.2 Representación de Números en Computadoras	9
2.2.1 Estructura y Limitaciones de los Números Flotantes	9
2.2.2 Errores Asociados con la Representación Numérica en Computadoras	11
2.2.3 Conclusión	11
2.3 Métodos de Cálculo de Derivadas	12
2.3.1 Método de Diferencias Finitas	12
2.3.2 Método del Paso Complejo	17
2.3.3 Diferenciación Automática	23
2.3.4 Derivadas de Orden Superior	24
2.3.5 Cálculo de Derivadas en Espacios Multidimensionales	27

2.3.6	Diferencias finitas en Espacios Multidimensionales	28
2.3.7	Método de Paso Complejo en Espacios Multidimensionales	29
2.3.8	Método Paso Multi-Complejo	30
2.4	Método de Newton-Krylov	36
2.4.1	Método de Newton	37
2.4.2	Espacios de Krylov y Método GMRES	41
2.4.3	Unión entre el Método de Newton y GMRES	46
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN		47
3.1	Motivación y Objetivos del Trabajo	47
3.2	Entorno Computacional	47
3.2.1	Lenguajes de Programacion	47
3.2.2	Infraestructura	48
3.3	Implementación en Python	49
3.3.1	Compatibilidad con SciPy	49
3.3.2	Modificación del algoritmo de Newton-Krylov	49
3.3.3	Repositorio	50
3.4	Implementación en Julia	50
3.4.1	Julia como Lenguaje Emergente	50
3.4.2	Implementación del Algoritmo de Newton-Krylov	50
3.4.3	Repositorio	51
3.5	Precisión en la Representación Numérica	51
3.6	Ajuste dinámico del paso	52
3.7	Experimentos Anexos	53
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		54
4.1	Comparación de Diferentes Niveles de Precisión Flotante	54
4.1.1	Motivacion	54
4.1.2	Rendimiento del Dato Numérico	54
4.2	Problemas Benchmark	56
4.3	Análisis Comparativo: Diferencias Finitas vs. Paso Complejo	58
4.3.1	Motivación	58
4.3.2	Obtención del Valor Numérico Real	58
4.3.3	Comparación de Resultados	59
4.3.4	Conclusiones	61
4.4	Análisis Comparativo: Residuo Mínimo Generalizado (GMRES)	62
4.4.1	Motivación	62
4.4.2	Calidad del Espacio	63
4.4.3	Comparación de Resultados	66
4.5	Análisis Comparativo en Método Newton-Krylov	72
4.5.1	Motivación	72
4.5.2	Análisis de Métodos de Ajuste Dinámico del Paso Diferencial	72
4.5.3	Comparación de Adaptadores	72
4.5.4	Comparación de Resultados	75

4.5.5	Conclusiones	81
4.6	Comparación entre Python y Julia	81
4.6.1	Motivación	81
4.6.2	Comparación	82
4.6.3	Conclusiones	85
4.7	Validación de Experimentos Anexos	85
4.7.1	Motivación	85
4.7.2	Extensión de Puntos de Aproximación	86
4.7.3	Derivadas de Orden Superior y Método de Paso Multicomplejo	89
CAPÍTULO 5: CONCLUSIONES		92
ANEXOS		94
5.1	README del Repositorio de Experimentos	94
5.2	Implementación de diferencias finitas en Scipy	94
5.3	Implementación de paso complejo en Scipy	95
5.4	Implementación de diferencias finitas en Julia	95
5.5	Implementación de paso complejo en Julia	96
5.6	Problema Sparse I	97
5.7	Problema Sparse II	98
5.8	Problema Dense I	99
5.9	Problema Dense II	99
5.10	Gráficos adicionales, producto jacobiano-vector	100
5.11	Gráficos adicionales, ortonormalidad espacio de Krylov	102
5.12	Gráficos adicionales, un paso de Newton-Krylov	105
5.13	Gráficos adicionales, adaptadores	109
5.14	Gráficos adicionales, Newton-Krylov completo	111
5.15	Gráficos adicionales, comparacion Lenguajes	119
REFERENCIAS BIBLIOGRÁFICAS		123

ÍNDICE DE FIGURAS

1	Error absoluto en la derivada, mediante diferencias finitas.	3	21	Ortonormalidad del espacio, problema Normal $n = 50$	63
2	Error absoluto en la derivada mediante diferencias finitas y paso complejo.	6	22	Ortonormalidad del espacio, problema Sparse I $n = 25000$	64
3	Ejemplo de distribución de números flotantes a lo largo de la recta numérica.	10	23	Ortonormalidad del espacio, problema Dense I $n = 25000$	64
4	Error absoluto en la derivada mediante diferencias finitas hacia adelante y centrales.	14	24	Ortonormalidad del espacio, problema Dense II $n = 25000$	65
5	Visualización del error absoluto en la derivada, mediante diferencias finitas y paso complejo.	18	25	Error de un paso de Newton, prob. Normal $n = 50$	67
6	Error absoluto en la derivada, mediante paso complejo y paso complejo central.	20	26	Tiempo de un paso de Newton, prob. Normal $n = 50$	67
7	Error absoluto y tiempo de cómputo de la derivada, calculada mediante diferencias finitas, paso complejo y diferenciación automática.	24	27	Equilibrio de rendimiento de un paso de Newton, prob. Normal $n = 50$	67
8	Error absoluto en la 2ª derivada, mediante diferencias finitas y paso complejo.	27	28	Error de un paso de Newton, prob. Normal $n = 25000$	67
9	Representación de árbol de número multicomplejo de grado n	32	29	Tiempo de un paso de Newton, prob. Normal $n = 25000$	67
10	Visualización del error absoluto de la 3ª derivada, mediante diferencias finitas, paso complejo y paso multicomplejo.	34	30	Equilibrio de rendimiento de un paso de Newton, prob. Normal $n = 25000$	68
11	Tiempos de cómputo de sumar dos vectores en distintos lenguajes.	55	31	Error de un paso de Newton, prob. Sparse I $n = 50$	68
12	Tiempos de cómputo de multiplicar dos vectores en distintos lenguajes.	55	32	Tiempo de un paso de Newton, prob. Sparse I $n = 50$	68
13	Tiempos de cómputo de multiplicar un vector por la constante 2 en distintos lenguajes.	55	33	Equilibrio de rendimiento de un paso de Newton, prob. Sparse I $n = 50$	68
14	Tiempos de cómputo de aplicar la función $\sin()$ a vectores en distintos lenguajes.	55	34	Error de un paso de Newton, prob. Sparse I $n = 25000$	69
15	Jerarquía de los distintos métodos entre sí.	58	35	Tiempo de un paso de Newton, prob. Sparse I $n = 25000$	69
16	Error de la derivada entre los métodos.	59	36	Equilibrio de rendimiento de un paso de Newton, prob. Sparse I $n = 25000$	69
17	Tiempo de cómputo de la derivada entre los métodos.	59	37	Error de un paso de Newton, prob. Dense II $n = 50$	69
18	Error absoluto y tiempo de computo para el problema Normal.	60	38	Tiempo de un paso de Newton, prob. Dense II $n = 50$	69
19	Error absoluto y tiempo de computo para el problema Sparse I.	61	39	Equilibrio de rendimiento de un paso de Newton, prob. Dense II $n = 50$	70
20	Error absoluto y tiempo de computo para el problema Dense II.	61	40	Error de un paso de Newton, prob. Dense II $n = 25000$	70
			41	Tiempo de un paso de Newton, prob. Dense II $n = 25000$	70
			42	Equilibrio de rendimiento de un paso de Newton, prob. Dense II $n = 25000$	70

43	Ratio de error, Sparse II con adaptadores. .	73	70	Error absoluto y tiempo de computo para el problema Sparse II.	101
44	Ratio de tiempo, Sparse II con adaptadores.	73	71	Error absoluto y tiempo de computo para el problema Dense I.	102
45	Ratio error, Dense I con adaptadores. . .	74	72	Error absoluto y tiempo de computo para el problema Dense II.	102
46	Ratio de tiempo, Dense I con adaptadores.	74	73	Ortonormalidad del espacio, problema Uniform $n = 50$	103
47	Ratio error, problema Sparse I, $n = 50$. .	76	74	Ortonormalidad del espacio, problema Sparse I $n = 50$	103
48	Ratio tiempo, problema Sparse I, $n = 50$. .	77	75	Ortonormalidad del espacio, problema Sparse II $n = 50$	104
49	Equilibrio de rendimiento, problema Sparse I, $n = 50$	77	76	Ortonormalidad del espacio, problema Sparse II $n = 25000$	104
50	Ratio error, problema Sparse I, $n = 25000$.	78	77	Ortonormalidad del espacio, problema Dense I $n = 50$	105
51	Ratio tiempo, problema Sparse I, $n = 25000$.	78	78	Ortonormalidad del espacio, problema Dense II $n = 50$	105
52	Equilibrio de rendimiento, problema Sparse I, $n = 25000$	79	79	Error de un paso de Newton, prob. Uniform $n = 50$	106
53	Ratio error, problema Dense II, $n = 25000$.	79	80	Tiempo de un paso de Newton, prob. Uniform $n = 50$	106
54	Ratio tiempo, problema Dense II, $n = 25000$.	80	81	Equilibrio de rendimiento de un paso de Newton, prob. Uniform $n = 50$	106
55	Equilibrio de rendimiento, problema Dense II, $n = 25000$	80	82	Error de un paso de Newton, prob. Uniform $n = 25000$	106
56	Ratio de error Julia/Python, problema Sparse I, $n = 50$	82	83	Tiempo de un paso de Newton, prob. Uniform $n = 25000$	106
57	Ratio de error Julia/Python, problema Sparse I, $n = 25000$	82	84	Equilibrio de rendimiento de un paso de Newton, prob. Uniform $n = 25000$	107
58	Ratio de error Julia/Python, problema Dense I, $n = 50$	83	85	Error de un paso de Newton, prob. Sparse II $n = 50$	107
59	Ratio de error Julia/Python, problema Dense I, $n = 25000$	83	86	Tiempo de un paso de Newton, prob. Sparse II $n = 50$	107
60	Ratio de tiempo Julia/Python, problema Sparse I, $n = 50$	84	87	Equilibrio de rendimiento de un paso de Newton, prob. Sparse II $n = 50$	107
61	Ratio de tiempo Julia/Python, problema Sparse I, $n = 25000$	84	88	Error de un paso de Newton, prob. Sparse II $n = 25000$	108
62	Ratio de tiempo Julia/Python, problema Dense I, $n = 50$	84	89	Tiempo de un paso de Newton, prob. Sparse II $n = 25000$	108
63	Ratio de tiempo Julia/Python, problema Dense I, $n = 25000$	84	90	Equilibrio de rendimiento de un paso de Newton, prob. Sparse II $n = 25000$	108
64	Evolución del error a medida que se añaden puntos a las aproximaciones.	87	91	Error de un paso de Newton, prob. Dense I $n = 50$	108
65	Evolución del error a medida que se añaden puntos decimales a las aproximaciones. .	89			
66	Norma de la derivada a medida que se incrementa el orden.	90			
67	Tiempo de computo de la derivada a medida que se incrementa el orden.	91			
68	Error absoluto y tiempo de computo para el problema Uniform.	100			
69	Error absoluto y tiempo de computo para el problema Sparse I.	100			

92	Tiempo de un paso de Newton, prob. Dense I $n = 50$	108	113	Ratio error, problema Dense II, $n = 50$. . .	118
93	Equilibrio de rendimiento de un paso de Newton, prob. Dense I $n = 50$	109	114	Ratio tiempo, problema Dense II, $n = 50$. . .	118
94	Error de un paso de Newton, prob. Dense I $n = 25000$	109	115	Equilibrio de rendimiento, problema Dense II, $n = 50$	119
95	Tiempo de un paso de Newton, prob. Dense I $n = 25000$	109	116	Ratio de error Julia/Python, problema Uniform, $n = 50$	119
96	Equilibrio de rendimiento de un paso de Newton, prob. Dense I $n = 25000$	109	117	Ratio de error Julia/Python, problema Normal, $n = 50$	120
97	Ratio de error, Sparse I con adaptadores. . .	110	118	Ratio de error Julia/Python, problema Sparse II, $n = 50$	120
98	Ratio de tiempo, Sparse I con adaptadores. . .	110	119	Ratio de error Julia/Python, problema Sparse II, $n = 25000$	120
99	Ratio de error, Dense II con adaptadores. . .	111	120	Ratio de error Julia/Python, problema Dense II, $n = 50$	120
100	Ratio de tiempo, Dense II con adaptadores. . .	111	121	Ratio de error Julia/Python, problema Dense II, $n = 25000$	121
101	Ratio error, problema Sparse II, $n = 50$. . .	112	122	Ratio de tiempo Julia/Python, problema Uniform, $n = 50$	121
102	Ratio tiempo, problema Sparse II, $n = 50$. . .	112	123	Ratio de tiempo Julia/Python, problema Normal, $n = 50$	121
103	Equilibrio de rendimiento, problema Sparse II, $n = 50$	113	124	Ratio de tiempo Julia/Python, problema Sparse II, $n = 50$	121
104	Ratio error, problema Sparse II, $n = 25000$. . .	113	125	Ratio de tiempo Julia/Python, problema Sparse II, $n = 25000$	122
105	Ratio tiempo, problema Sparse II, $n = 25000$. . .	114	126	Ratio de tiempo Julia/Python, problema Dense II, $n = 50$	122
106	Equilibrio de rendimiento, problema Sparse II, $n = 25000$	114	127	Ratio de tiempo Julia/Python, problema Dense II, $n = 25000$	122
107	Ratio error, problema Dense I, $n = 50$. . .	115			
108	Ratio tiempo, problema Dense I, $n = 50$. . .	115			
109	Equilibrio de rendimiento, problema Dense I, $n = 50$	116			
110	Ratio error, problema Dense I, $n = 25000$. . .	116			
111	Ratio tiempo, problema Dense I, $n = 25000$. . .	117			
112	Equilibrio de rendimiento, problema Dense I, $n = 25000$	117			

ÍNDICE DE TABLAS

1	Características de un nodo en la partición <i>main</i>	48
---	--	----

INTRODUCCIÓN

El cálculo numérico de derivadas es una herramienta esencial en disciplinas como la física, ingeniería y ciencias computacionales, permitiendo resolver problemas complejos como la optimización, simulación de fenómenos físicos y la resolución de sistemas de ecuaciones no lineales. Sin embargo, los métodos tradicionales como las diferencias finitas para la aproximación de la matriz Jacobiana presentan limitaciones inherentes, particularmente los errores de cancelación y redondeo, que comprometen tanto la precisión como la estabilidad numérica de los cálculos.

Este trabajo propone analizar el método de derivación de paso complejo como una alternativa robusta para superar estas limitaciones. Este método, al apoyarse en propiedades de los números complejos, logra evitar errores de cancelación y ofrece una mayor precisión en comparación con enfoques convencionales. Su implementación en algoritmos iterativos como Newton-Krylov puede mostrar beneficios significativos en términos de estabilidad y eficiencia computacional.

Un aspecto fundamental de este estudio es el desarrollo e implementación del método en los lenguajes de programación Python y Julia, seleccionados por su potencia y flexibilidad en cálculos científicos. Se han creado y probado librerías específicas para cada lenguaje, diseñadas para integrar el método de paso complejo en entornos computacionales modernos. En Python, se aprovechó la compatibilidad con bibliotecas ampliamente utilizadas como SciPy.

Además, este trabajo se apoya en la infraestructura de supercomputación proporcionada por el National Laboratory for High Performance Computing (NLHPC). Esta plataforma ha permitido ejecutar experimentos de gran tamaño. El acceso al clúster ha sido esencial para evaluar el rendimiento de las implementaciones, especialmente en problemas que involucren grandes sistemas de ecuaciones.

En conclusión, este estudio busca no solo analizar la precisión y eficiencia del método de paso complejo, sino también demostrar su aplicabilidad práctica en contextos computacionales avanzados. Al integrar herramientas modernas como Python, Julia y la infraestructura de supercomputación, se pretende establecer una base sólida para el uso de este método en aplicaciones científicas e ingenieriles donde la precisión y la robustez son críticas.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contextualización y Relevancia de la problemática

1.1.1. El Rol de las Derivadas en Cálculos Numéricos

El cálculo de derivadas es un paso crucial en diversas disciplinas científicas, como las de ingeniería, especialmente en problemas donde los cambios continuos y dinámicos son el tema central de estos. Las derivadas permiten comprender la tasa de cambio entre dos variables, conocimiento fundamental en áreas como la física, economía, biología y las ciencias de la computación. En términos numéricos, la derivada se utiliza para resolver problemas de optimización, modelar fenómenos de transferencia, describir movimientos e incluso en el diseño de algoritmos para aprendizaje automático [Ahsan, 2017]. En estos contextos, calcular derivadas exactas es crucial para lograr modelos precisos y realizar simulaciones que puedan predecir comportamientos futuros de manera confiable.

Un ejemplo más concreto y común de la aplicación de derivadas es la optimización de funciones, especialmente en métodos iterativos como el gradiente descendente [Phillips, 2023]. Este método, ampliamente utilizado en algoritmos de aprendizaje automático, depende fuertemente de la derivada de la función que se está optimizando para ajustar los parámetros del modelo. Si las derivadas son mal calculadas, los algoritmos pueden converger a soluciones incorrectas o ser mucho más lentos de lo esperado [Goodfellow *et al.*, 2016]. Por lo tanto, la precisión y eficiencia de los cálculos de derivadas son determinantes en la viabilidad de muchas aplicaciones prácticas.

Las derivadas también juegan un papel clave en la simulación de procesos físicos, como en la dinámica de fluidos o la electrodinámica, donde pequeños errores en las derivadas pueden tener efectos acumulativos y afectar drásticamente los resultados. Por ello, métodos precisos y eficientes para calcular derivadas son esenciales para las áreas de ingeniería y las ciencias.

1.1.2. Impacto de los Errores Numéricos en Aplicaciones Prácticas

Cuando se trata de estimar derivadas numéricamente, el enfoque tradicional es el uso de diferencias finitas, que aproxima la derivada de una función evaluando la diferencia entre valores cercanos de la función en el punto de interés [LeVeque, 2007]. Este método es popular por su simplicidad y facilidad de implementación, pero es susceptible a errores numéricos, principalmente los relacionados con la precisión de los cálculos en computadoras de precisión finita.

La fórmula básica para la obtención de la derivada por diferencias finitas es la siguiente:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$

esta fórmula se basa en la aproximación de la derivada mediante la diferencia entre dos puntos cercanos en la función, y su exactitud depende de la elección de h , el cual es idealmente un valor pequeño. Sin embargo, cuando se elige un valor muy pequeño, los errores de redondeo y de cancelación pueden amplificarse debido a la limitación de la precisión numérica de las computadoras [Muller *et al.*, 2018]. El uso de valores pequeños para h puede resultar en errores que afectan la precisión del resultado final de forma significativa. En particular, los errores de cancelación ocurren cuando se restan dos valores muy cercanos, distintos en el dominio de los números reales, pero representados de forma idéntica dentro de la computadora, cosa que afecta la precisión de los cálculos.

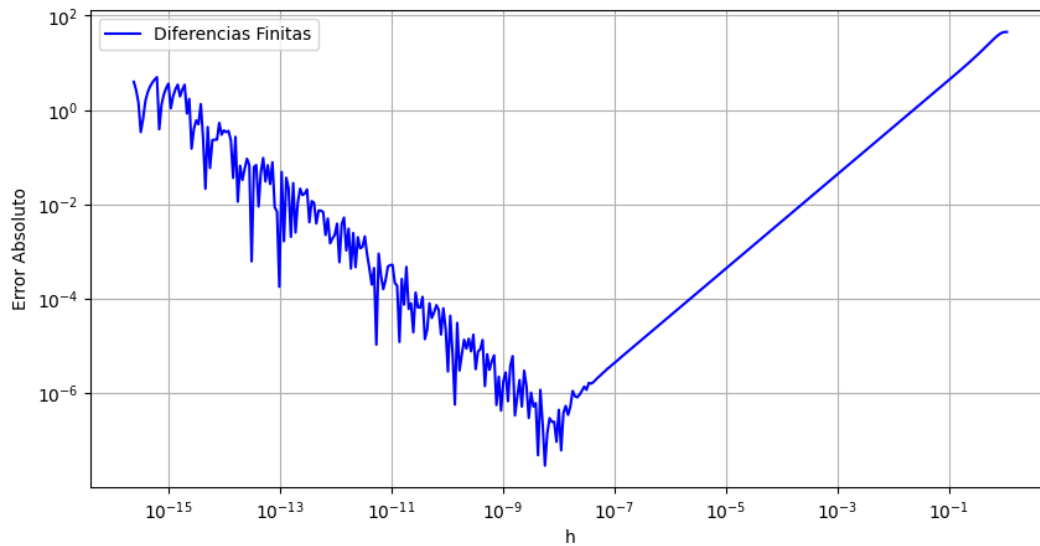


Figura 1: Error absoluto de la aproximación por diferencias finitas de $\left. \frac{d}{dx} \left(\frac{e^x}{\sin^3(x) + \cos^3(x)} \right) \right|_{x=3}$.
Fuente: Elaboración propia.

El problema de los errores numéricos es aún más crítico en sistemas complejos y en aplicaciones que requieren resultados extremadamente precisos, como en simulaciones físicas o en el procesamiento de señales, donde incluso pequeñas imprecisiones pueden afectar drásticamente los resultados [Higham, 2002].

1.2. Limitaciones de los Métodos Convencionales

1.2.1. Diferencias Finitas y Sus Restricciones

El método de diferencias finitas: $\frac{f(x+h) - f(x)}{h} = f'(x) + O(h)$, aunque ampliamente utilizado debido a su simplicidad, enfrenta varias limitaciones inherentes. La precisión de las aproximaciones de derivadas depende críticamente del tamaño de h . Cuando h es demasiado grande, la aproximación pierde precisión, y cuando es demasiado pequeño, los errores de

cancelación se vuelven dominantes. Esto crea un compromiso, en el que se debe equilibrar la precisión y la estabilidad numérica.

Otro problema a considerar de este método es que las diferencias finitas requieren de al menos dos evaluaciones de la función, lo que puede resultar costoso en términos de tiempo de cálculo, especialmente cuando se trata de funciones complicadas o en sistemas que requieren cálculos de derivadas en muchas dimensiones. Esto amplifica los problemas mencionados, debido a la necesidad de realizar muchas evaluaciones para cada componente de la derivada.

1.2.2. Errores de Cancelación: Origen y Consecuencias

Los errores de cancelación ocurren cuando se realizan operaciones entre números de magnitudes similares pero con signos opuestos, lo que provoca que los bits menos significativos de los números, de los que depende su distinción, se cancelen, resultando en una pérdida de precisión. Para ser más preciso, los errores ocurren cuando los bits que hacen diferir los números entre sí quedan fuera de la representación binaria del dígito por culpa de la regla de redondeo de la IEEE 754 [ove, 2019]. Este fenómeno es particularmente común en el cálculo de derivadas mediante diferencias finitas, especialmente cuando se elige un valor de h demasiado pequeño. En este contexto, los errores de redondeo pueden propagarse y aumentar a medida que se realizan más cálculos, lo que degrada la precisión global del algoritmo.

En aplicaciones como el análisis numérico de sistemas de ecuaciones diferenciales, donde el cálculo de derivadas es fundamental, estos errores pueden generar inestabilidades numéricas que afectan la calidad de la simulación. En particular, cuando se resuelven problemas de optimización o simulación dinámica, los errores de cancelación pueden alterar las trayectorias de los sistemas y desviar los resultados del comportamiento esperado. La corrección de estos errores es esencial para obtener resultados fiables y realistas en aplicaciones prácticas de simulación científica y de ingeniería [Higham, 2002] [Stoer y Bulirsch, 2002].

1.3. Objetivos del Análisis

1.3.1. Problemas Clave a Abordar

Este estudio se enfoca en abordar los problemas de precisión y estabilidad numérica en el cálculo de derivadas. Específicamente, se busca identificar los errores inherentes a los métodos convencionales, como las diferencias finitas, y encontrar soluciones que permitan mejorar tanto la precisión como la eficiencia de los cálculos. Entre los problemas clave a abordar se encuentran:

- Minimizar los errores de cancelación al calcular derivadas de funciones en entornos discretos.
- Mejorar la eficiencia computacional, reduciendo el tiempo de cálculo sin sacrificar la precisión.

- Evaluar la aplicabilidad de métodos alternativos, como el método de paso complejo, en comparación con métodos tradicionales [Baker y Johnson, 2018].

1.3.2. Planteamiento del Método Propuesto

Se propone el uso del método de paso complejo como una alternativa robusta para el cálculo de derivadas, especialmente en situaciones donde los métodos tradicionales como las diferencias finitas tienen limitaciones debido a los errores de cancelación. El método de paso complejo tiene la ventaja de evitar estos errores, ya que utiliza el dominio de los números complejos para calcular derivadas, lo que nos permitirá garantizar una mayor precisión sin necesidad de un límite h . Este método se basa en la fórmula:

$$f'(x) \approx \frac{\text{Im}(f(x + ih))}{h}$$

donde i es la unidad imaginaria y $\text{Im}()$ es la función que extrae la parte imaginaria de un número complejo. Este enfoque mejora la precisión al evitar las operaciones de cancelación, permitiéndole en muchos casos obtener una aproximación exacta hasta donde lo permite la máquina, incluso con valores muy pequeños de h [Van der Veen, 2012].

1.4. Justificación del Enfoque Propuesto

1.4.1. Ventajas del Método de Paso Complejo

El método de paso complejo tiene varias ventajas con respecto a otros métodos de cálculo de derivadas. A diferencia de las diferencias finitas, este método no está sujeto a los errores de cancelación, lo que lo convierte en una opción más precisa, especialmente en aplicaciones donde los cálculos numéricos deben ser altamente precisos. Además, al utilizar el paso complejo, el método de paso complejo es capaz de proporcionar resultados exactos hasta la precisión de máquina en muchos casos, lo que lo convierte en una alternativa muy atractiva para aplicaciones de alta precisión.

En términos computacionales, este método también es eficiente, ya que no requiere evaluaciones repetidas de la función, sino que solo requiere una evaluación en un punto cercano. Esto reduce el costo computacional en comparación con métodos que requieren múltiples evaluaciones [Van der Veen, 2012].

1.4.2. Comparación Inicial con Otros Métodos Alternativos

Cuando se compara el método de paso complejo con otros métodos alternativos, como las diferencias finitas de orden superior [2024] o la derivación simbólica [Grossman, 1989], se observa que el método de paso complejo ofrece una precisión y estabilidad numérica más que competentes. Las diferencias finitas de orden superior mejoran la precisión al utilizar

aproximaciones de mayor orden, pero aún son susceptibles a los errores de cancelación [Stoer y Bulirsch, 2002]. Por otro lado, la derivación simbólica es exacta, pero puede ser mucho más costosa computacionalmente, especialmente cuando se aplica a funciones complejas o a problemas de optimización multivariable.

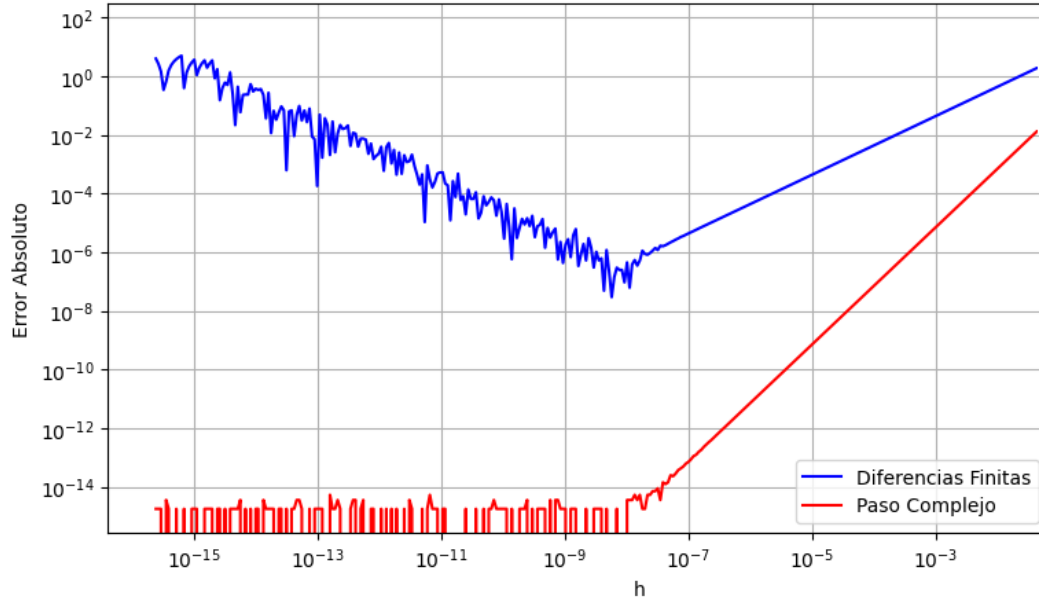


Figura 2: Error absoluto de la aproximación por diferencias finitas y paso complejo de $\frac{d}{dx} \left(\frac{e^x}{\sin^3(x) + \cos^3(x)} \right) \Big|_{x=3}$.

Fuente: Elaboración propia.

1.5. Problemas de Malas Aproximaciones de Derivadas en Métodos Iterativos

1.5.1. Impacto de las Malas Aproximaciones en el Método GMRES

El método del residuo mínimo generalizado (GMRES) [Saad, 2003] es un algoritmo iterativo ampliamente utilizado para resolver sistemas de ecuaciones lineales de gran tamaño. Este método se basa en construir una secuencia de aproximaciones sucesivas dentro de un espacio de Krylov generado por el sistema original. Los métodos iterativos, como GMRES, son muy eficaces cuando se utilizan con una correcta evaluación de los residuos, pero las malas aproximaciones pueden tener un impacto negativo en su desempeño y convergencia.

En el caso de GMRES, las malas aproximaciones del sistema de entrada, pueden influir negativamente en el cálculo de las direcciones de búsqueda en el espacio de Krylov. Si los residuos son calculados de manera inexacta, el vector residual, que es clave para construir el subespacio de Krylov, puede no estar alineado adecuadamente con la solución, lo que resulta en una convergencia más lenta o incluso en la falta de convergencia del método.

El rendimiento de GMRES depende críticamente de la precisión de las iteraciones previas, por lo que una mala aproximación de los residuos durante el cálculo puede generar errores acumulativos que afectan todas las aproximaciones sucesivas, desajustando la aproximación final. Un caso típico es cuando los residuos utilizados en el cálculo de un Jacobiano (que puede ser esencial en el contexto de métodos no lineales o en problemas de optimización) no son lo suficientemente precisos, lo que a su vez introduce distorsiones en la matriz del sistema, afectando la convergencia del método iterativo.

Este método es un punto clave en el presente análisis y será tratado de forma más profunda más adelante.

1.5.2. Espacios de Krylov y Errores en la Aproximación de Derivadas

Los espacios de Krylov [Saad, 2003] están formados por los productos sucesivos de la matriz del sistema con los vectores residuales, y son fundamentales para obtener una buena aproximación de la solución. En nuestro análisis, estos espacios se obtendrán a partir de aproximaciones de derivadas, por lo que los errores en la aproximación de estas distorsionarán la estructura de estos espacios, modificando descontroladamente sus valores numéricos y afectando la representación del problema original.

Cuando las aproximaciones de derivadas son inexactas, los subespacios generados pueden no reflejar correctamente los gradientes necesarios, lo que ralentiza la convergencia del algoritmo GMRES. Este desajuste en los vectores de búsqueda puede hacer que el espacio de Krylov no capture adecuadamente la información sobre la solución, lo que a su vez puede conducir a una convergencia prematura a una solución incorrecta.

El impacto de estos errores no se limita a un solo paso, sino que se propaga en cada iteración, amplificando las inexactitudes y dificultando la mejora de la solución. En estos casos, los errores acumulativos derivados de malas aproximaciones de derivadas pueden hacer que GMRES sea ineficaz, requiriendo más iteraciones o incluso fallando en encontrar una solución aceptable.

1.5.3. Posibles Soluciones y Mejoras

Para mitigar los efectos negativos de las malas aproximaciones de derivadas en GMRES y los espacios de Krylov en este contexto, se pueden emplear diversas estrategias. Una de ellas, y en la que se enfocará este análisis, es mejorar la precisión de las aproximaciones de las derivadas mediante el uso de métodos más robustos, como la derivación automática [Corliss *et al.*, 2013] o el método de paso complejo [Baker y Hughes, 1997]. Estas técnicas ayudan a minimizar los errores de cancelación que afectan el cálculo de los gradientes, mejorando así la estabilidad y la convergencia del algoritmo, ayudando a que la solución entregada corresponda a la del problema original.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Análisis Numérico y Álgebra Computacional

El análisis numérico y el álgebra computacional son dos ramas fundamentales de la matemática aplicada que se complementan en la resolución de problemas numéricos complejos. Aunque ambos enfoques se ocupan de encontrar soluciones a problemas matemáticos, tienen objetivos, métodos y herramientas distintas. A continuación, se detalla la distinción entre el análisis numérico y el álgebra computacional, así como el papel de los métodos numéricos en la solución de problemas matemáticos.

2.1.1. Distinción entre Análisis Numérico y Álgebra Computacional

El **Análisis Numérico** se enfoca en desarrollar y estudiar algoritmos que aproximen soluciones numéricas a problemas matemáticos, a estos se les llaman **métodos numéricos**. En este contexto, los problemas son generalmente expresados en términos de ecuaciones o sistemas de ecuaciones que no tienen una solución exacta o son computacionalmente inalcanzables debido a la complejidad de los cálculos. El objetivo del análisis numérico es encontrar soluciones aproximadas con un control preciso sobre el error y la precisión de los resultados [Sutton, 2019].

Por otro lado, el **Álgebra Computacional** incluye el **cálculo simbólico**, que se ocupa de la manipulación simbólica de expresiones algebraicas utilizando algoritmos para obtener representaciones exactas de soluciones matemáticas. En lugar de aproximarse a soluciones numéricas, el cálculo simbólico se enfoca en resolver ecuaciones algebraicas, simplificar expresiones y realizar operaciones simbólicas exactas. Estos métodos son cruciales cuando la exactitud algebraica es esencial y no se requiere solo una solución numérica [Cohen, 2003].

La principal diferencia entre ambos enfoques radica en el tipo de problemas que abordan: el análisis numérico se orienta hacia la aproximación de soluciones cuando no es posible obtener una exactitud perfecta, mientras que el álgebra computacional busca manipular expresiones de manera exacta para obtener soluciones simbólicas.

2.1.2. Rol de los Métodos Numéricos en la Solución de Problemas Matemáticos

Los **métodos numéricos** juegan un papel crucial en la solución de una amplia variedad de problemas matemáticos, especialmente aquellos que no pueden ser resueltos mediante métodos analíticos o algebraicos exactos. En aplicaciones prácticas, como la ingeniería, la física

y las ciencias computacionales, los problemas suelen involucrar ecuaciones diferenciales, optimización, o grandes sistemas de ecuaciones que no tienen soluciones exactas o son demasiado complejos para ser resueltos simbólicamente.

Estos métodos permiten la aproximación de soluciones mediante el uso de técnicas computacionales, permitiendo que los resultados sean obtenidos en un tiempo razonable y con un nivel de precisión adecuado para la aplicación en cuestión. Estos métodos se caracterizan por el uso de discretización, la iteración, y la minimización de errores, entre otros.

Un ejemplo común de la aplicación de métodos numéricos es la solución de ecuaciones no lineales. Métodos como el de Newton-Raphson o el método de bisección [Small y Wang, 2007] son utilizados para encontrar las raíces de ecuaciones cuando no es posible encontrar una solución exacta. Otro ejemplo es la resolución de sistemas de ecuaciones lineales mediante métodos iterativos como el de Jacobi o Gauss-Seidel [Watkins, 2004], que se aplican en problemas donde las matrices involucradas son demasiado grandes para ser invertidas directamente [Sauer, 2013].

Los métodos numéricos son fundamentales en la simulación de fenómenos físicos y la optimización de sistemas, permitiendo aproximar soluciones a problemas que involucran grandes cantidades de datos y complejidad computacional. Estos métodos proporcionan herramientas eficientes y precisas para manejar la incertidumbre y los errores inherentes a los cálculos, lo cual es esencial en muchas aplicaciones prácticas.

En resumen, mientras que el álgebra computacional se enfoca en manipular expresiones simbólicas de manera exacta, el análisis numérico aborda la resolución de problemas mediante aproximaciones computacionales, siendo ambos enfoques complementarios en la práctica matemática y científica.

2.2. Representación de Números en Computadoras

2.2.1. Estructura y Limitaciones de los Números Flotantes

En el contexto de la computación, los números reales no pueden ser representados de manera exacta debido a las limitaciones de memoria de las computadoras. Dado que los números reales son infinitos, se deben utilizar aproximaciones para representar una cantidad finita de estos, lo que da lugar a errores de redondeo y truncamiento en los cálculos [Muller *et al.*, 2018].

El estándar de precisión flotante IEEE 754, es el formato más comúnmente utilizado para representar números reales en las computadoras. Este estándar se emplea en la mayoría de las arquitecturas modernas y define los aspectos técnicos de la representación binaria de los números de punto flotante. Los números de punto flotante se dividen en tres partes fundamentales:

1. **Signo:** Un solo bit que indica si el número es positivo o negativo.
2. **Exponente:** Un conjunto de bits que define la magnitud del número mediante una potencia de base 2. Este exponente es clave para ajustar el valor de los números flotantes a una amplia gama de magnitudes.
3. **Fracción (o Mantisa):** Una secuencia binaria que representa la parte decimal del número, normalizada para que el valor principal sea siempre mayor o igual a 1 y menor que 2.

Matemáticamente, la representación de un número en punto flotante puede escribirse de la siguiente forma:

$$(-1)^{\text{Signo}} \cdot 2^{\text{Exponente}} \cdot 1.\text{Fracción}.$$

Es importante destacar que los números de punto flotante están diseñados para maximizar la cantidad posible de números a representar, optimizando el uso de los bits en la memoria por medio de una distribución no lineal de estos, manteniendo una mayor densidad en números cuya magnitud se aproxima a cero. Sin embargo, este diseño presenta ciertas limitaciones inherentes que afectan la precisión de las operaciones aritméticas. Dado que la fracción es un número binario finito, los números decimales exactos no siempre pueden ser representados con precisión. Esto da lugar a *errores de redondeo o truncamiento*, especialmente cuando se realiza una serie de operaciones sucesivas sobre números que no pueden ser representados de manera exacta [Stoer y Bulirsch, 2002].

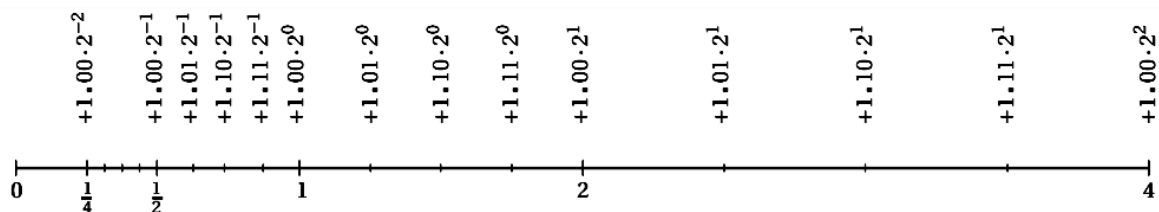


Figura 3: Ejemplo de distribución de números flotantes a lo largo de la recta numérica.
Fuente: Apuntes de Computación Científica.

Un fenómeno relevante en la representación de números flotantes es la *densidad* de los números representables. A medida que el exponente aumenta, los números representables se separan más entre sí. Esto implica que, a medida que aumenta el orden de un número, se pierde precisión en los dígitos menos significativos pequeños. Es decir, la densidad de números flotantes a través de la recta numérica va decreciendo mientras más nos alejamos del cero [Muller et al., 2018].

Cuando el resultado de una operación no puede ser representado exactamente debido a la limitación de precisión del sistema de punto flotante, se redondea al número representable más cercano. Este proceso de redondeo introduce errores de redondeo que pueden tener un impacto significativo en los cálculos, especialmente en problemas numéricos delicados.

2.2.2. Errores Asociados con la Representación Numérica en Computadoras

La representación numérica de los números reales en computadoras introduce varios tipos de errores que afectan los cálculos realizados con algoritmos numéricos. Estos errores pueden ser clasificados principalmente en dos categorías: errores de *redondeo* y errores de *truncamiento*, los cuales, si bien son similares, tienen sus matices.

1. **Errores de Redondeo:** Ocurren cuando un número no puede representarse exactamente debido a limitaciones en la precisión de los sistemas de almacenamiento, como en el caso de fracciones decimales no representables en binario.
2. **Errores de Truncamiento:** Surgen al eliminar partes necesarias para la representación exacta de un número, comunes en métodos iterativos como las series de Taylor o aproximaciones numéricas.

Ambos tipos de errores tienen un impacto directo en los resultados de los cálculos. En general, la propagación de estos errores depende de la estabilidad numérica de los algoritmos utilizados. Los algoritmos bien diseñados están optimizados para minimizar la propagación de estos errores; de lo contrario, pueden amplificar estos errores, llevando a resultados imprecisos o incluso incorrectos.

Los errores de cancelación son particularmente relevantes en métodos de diferencia finita, que se utilizan para aproximar derivadas o resolver ecuaciones diferenciales. En este contexto, la precisión de los cálculos puede ser afectada por la elección del valor de h , el parámetro que define el tamaño de los incrementos en las aproximaciones numéricas. Si h es demasiado pequeño, los valores de x_0 y $x_0 + h$ pueden ser tan cercanos que su diferencia es representada como cero debido a las limitaciones de precisión, lo que introduce un error en el cálculo de la derivada [Stoer y Bulirsch, 2002].

A medida que los cálculos numéricos se vuelven más complejos, es crucial el tratar estos errores. Los desarrolladores de algoritmos numéricos deben ser conscientes de estos errores y buscar métodos para minimizarlos, garantizando que los resultados sean lo más precisos posible, incluso cuando se enfrentan a las limitaciones inherentes a la representación de los números en computadoras.

2.2.3. Conclusión

La representación numérica de los números reales en computadoras es un área crítica en el análisis numérico. Aunque los números de punto flotante permiten representar una amplia gama de números con una cantidad limitada de memoria de forma muy eficiente, también introducen limitaciones que afectan la precisión de los cálculos. Los errores de redondeo y truncamiento son inevitables y pueden propagarse en los cálculos, afectando los resultados finales. La comprensión de estos errores es esencial para diseñar algoritmos numéricos estables y precisos que puedan manejar las limitaciones de la representación numérica en computadoras.

2.3. Métodos de Cálculo de Derivadas

2.3.1. Método de Diferencias Finitas

El método más comúnmente utilizado para el cálculo numérico de derivadas es el método de diferencias finitas. Este enfoque se basa en la expansión de Taylor y tiene una estructura similar a la definición clásica de la derivada a través de límites [Burden y Faires, 2010].

La expresión de la derivada en los números reales es la siguiente, ofreciendo un resultado exacto cuando el valor de h es infinitesimalmente pequeño:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}.$$

La expresión usual de diferencias finitas es:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h},$$

a pesar de que esta forma es idéntica a su definición de límite, la diferencia clave radica en su aplicación en un entorno computacional, que no opera con verdaderos números reales, sino con una precisión finita que introduce errores de redondeo y truncamiento.

El método consiste en aproximar numéricamente el valor de la derivada calculando la diferencia entre el valor de la función en el punto de interés y el valor de la función en un punto cercano, ligeramente perturbado. Esta expresión se deriva al despejar la primera derivada en la expansión de Taylor de la función en el punto de interés, considerando una pequeña perturbación [Atkinson, 1989].

La ecuación (2.3.1) se conoce como *diferencias finitas hacia adelante*, pero existen otras variaciones del método. Las expresiones para diferencias finitas hacia atrás y centrales [Ames, 2014] son:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h},$$
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h},$$

estas expresiones operan de manera similar a la diferencia finita hacia adelante, ofreciendo alternativas útiles dependiendo de la disponibilidad de valores de la función en diferentes puntos.

Contrario a lo que podría pensarse inicialmente, los fundamentos de este método no se originan simplemente de la búsqueda de imitar la definición del límite, sino de la solución de un sistema lineal obtenido a partir de una combinación de distintas expresiones de Taylor de la misma función [Atkinson, 1989]. Por ejemplo, al derivar el método de diferencias finitas hacia adelante, se considera la expansión de Taylor de la función en el punto x .

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$

sustrayendo $f(x)$, se obtiene:

$$f(x+h) - f(x) = hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$

y dividiendo ambos lados por h resulta en lo siguiente:

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{h}{2!}f''(x) + \frac{h^2}{3!}f'''(x) + \dots$$

Esto muestra que la aproximación de la derivada en el método de diferencias finitas también introduce términos de error que dependen de h , lo cual es fundamental para entender la precisión del método en la práctica.

El proceso para derivar el método de diferencias finitas centrales sigue el mismo enfoque que el método hacia adelante, pero se basa en puntos simétricamente distribuidos alrededor del punto de interés, lo cual permite reducir el orden del error en la aproximación resultante. Al utilizar la expansión de Taylor en $f(x+h)$ y $f(x-h)$, se obtienen expresiones que se combinan para cancelar algunos términos de error adicionales.

Considerando la expansión de Taylor para $f(x+h)$:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$

y para $f(x-h)$:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \dots$$

restando $f(x-h)$ de $f(x+h)$, los términos de orden par se cancelan, lo que reduce el error en la aproximación:

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2h^3}{3!}f'''(x) + \dots$$

dividiendo ambos lados por $2h$, se obtiene una aproximación para la derivada central con un término de error de orden h^2 :

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2).$$

Esta expresión nos permite calcular $f'(x)$ con un error cuadrático respecto a h , lo que indica que la precisión aumenta al reducir h , siempre y cuando este valor no sea tan pequeño como para introducir errores de cancelación debido a las limitaciones en la representación en punto flotante del sistema. Este método es utilizado en contextos donde se requiere una mayor precisión y estabilidad numérica.

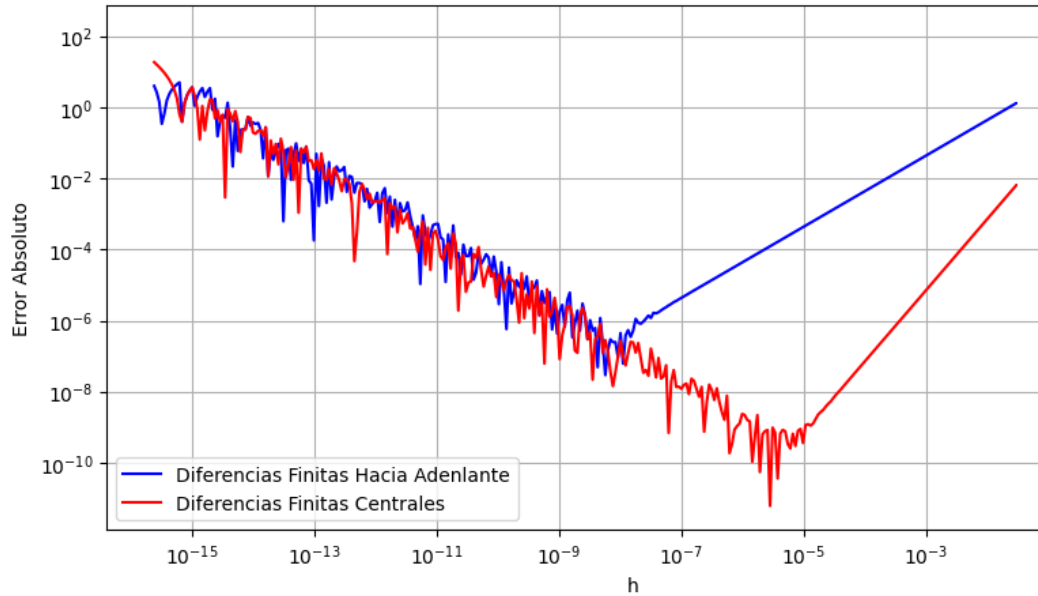


Figura 4: Error absoluto de la aproximación por diferencias finitas hacia adelante y centrales de $\frac{d}{dx} \left(\frac{e^x}{\sin^3(x) + \cos^3(x)} \right) \Big|_{x=3}$.

Fuente: Elaboración propia.

2.3.1.1. Convergencia y Error en Diferencias Finitas

El análisis de los métodos de diferencias finitas revela la existencia de una cantidad infinita de variantes. Cada método se define por un conjunto de puntos utilizados para calcular la diferencia. En el caso de la diferencia hacia adelante, se emplean los puntos $\{x, x + h\}$; para la diferencia hacia atrás, se usan $\{x, x - h\}$; mientras que la diferencia central se calcula con los puntos $\{x + h, x - h\}$ [Ames, 2014]. Sin embargo, la elección de los puntos puede ser arbitraria, lo cual permite construir métodos personalizados ajustando los puntos de evaluación en torno a x .

2.3.1.2. Método del Sistema Lineal para Coeficientes de Diferencias Finitas

Cada conjunto de puntos seleccionados genera una aproximación específica de la función, ya que permite reducir el error al eliminar términos de la expansión de Taylor. Como se discutió en secciones anteriores, este proceso utiliza las evaluaciones de la función en distintos puntos para construir combinaciones lineales que anulen los términos de la expansión, dejando únicamente el término correspondiente a la derivada de interés. Este procedimiento puede representarse como un sistema lineal de la forma $A\mathbf{x} = \mathbf{b}$, donde las columnas de la matriz están definidas por:

$$\mathbf{a}_i = \left(\frac{k_i^0}{0!} \quad \frac{k_i^1}{1!} \quad \frac{k_i^2}{2!} \cdots \frac{k_i^{n-1}}{(n-1)!} \right),$$

donde $\mathbf{a}_i$ contiene los factores, sin incluir el valor h , que acompañan a los n primeros coeficientes de la expansión de Taylor, siendo n la cantidad de puntos que se utilizaron en total y k_i la proporción del factor h .

El sistema generado se resuelve mediante álgebra matricial para determinar los coeficientes de la aproximación de $f'(x)$ [Ames et al., 2014]. La precisión del método está directamente relacionada con el orden del truncamiento, el cual depende de los términos que no se cancelan en la expansión de Taylor.

Como ejemplo, se presenta un sistema de ecuaciones generado a partir de los puntos $\{x, x+h, x+2h, x+3h, x+4h\}$:

$$A \cdot \mathbf{b} = \begin{pmatrix} 1 & \frac{1^0}{0!} & \frac{2^0}{0!} & \frac{3^0}{0!} & \frac{4^0}{0!} \\ 0 & \frac{1^1}{1!} & \frac{2^1}{1!} & \frac{3^1}{1!} & \frac{4^1}{1!} \\ 0 & \frac{1^2}{2!} & \frac{2^2}{2!} & \frac{3^2}{2!} & \frac{4^2}{2!} \\ 0 & \frac{1^3}{3!} & \frac{2^3}{3!} & \frac{3^3}{3!} & \frac{4^3}{3!} \\ 0 & \frac{1^4}{4!} & \frac{2^4}{4!} & \frac{3^4}{4!} & \frac{4^4}{4!} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

En este sistema, cada columna de la matriz A representa los coeficientes de la serie de Taylor, mientras que el vector b contiene los coeficientes multiplicadores de cada instancia de $f(x_i)$. Por ejemplo, con cinco puntos como en el ejemplo, el método de diferencias finitas para $f'(x)$ se aproxima como:

$$f'(x) \approx \frac{b_0 \cdot f(x) + b_1 \cdot f(x+h) + b_2 \cdot f(x+2h) + b_3 \cdot f(x+3h) + b_4 \cdot f(x+4h)}{h},$$

donde los valores b_i son obtenidos resolviendo el sistema mostrado en (2.3.1.2).

Como un ejemplo más pequeño podemos ver este caso, donde se emplean los puntos $\{x, x+h, x+2h\}$; el sistema simplificado resulta en:

$$A \cdot \mathbf{b} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0,5 & 2 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Tras resolver el sistema hallamos que los coeficientes b_i son $b_0 = \frac{-3}{2}$, $b_1 = 2$, $b_2 = \frac{-1}{2}$, dando como resultado la siguiente expresión para diferencias finitas con esos 3 coeficientes.

$$f'(x) \approx \frac{-\frac{3}{2}f(x) + 2f(x+h) - \frac{1}{2}f(x+2h)}{h}.$$

2.3.1.3. Método de Lagrange para Coeficientes de Diferencias Finitas

Otro enfoque para calcular los coeficientes de diferencias finitas consiste en utilizar la interpolación de Lagrange. Este método construye un polinomio $p(t)$ que interpola los puntos de evaluación $\{x_0, x_1, \dots, x_k\}$ [Ames *et al.*, 2014].

El polinomio de interpolación de Lagrange está dado por:

$$p(t) = \sum_{j=0}^k f(x_j) \prod_{\substack{m=0 \\ m \neq j}}^k \frac{t - x_m}{x_j - x_m},$$

donde cada término de la suma corresponde a un polinomio base $L_j(t)$ asociado al punto x_j :

$$L_j(t) = \prod_{\substack{m=0 \\ m \neq j}}^k \frac{t - x_m}{x_j - x_m}.$$

Para obtener los coeficientes, debemos derivar la expresión anterior respecto a t y posteriormente evaluarla en x . Esto produce:

$$c_j = \frac{d}{dt} \left(\prod_{\substack{m=0 \\ m \neq j}}^k \frac{t - x_m}{x_j - x_m} \right) \Big|_{t=x}.$$

Si consideramos tres puntos $\{x_0 = x, x_1 = x + h, x_2 = x + 2h\}$, los coeficientes c_j para la primera derivada se calculan como:-

$$c_0 = \frac{d}{dt} \left(\frac{t - x_1}{x_0 - x_1} \frac{t - x_2}{x_0 - x_2} \right) \Big|_{t=x} = \frac{d}{dt} \left(\frac{t - x - h}{x - x - h} \frac{t - x - 2h}{x - x - 2h} \right) \Big|_{t=x} =$$

$$\frac{d}{dt} \left(\frac{2h^2 - 3h(t - x) + (t - x)^2}{2h^2} \right) \Big|_{t=x} = \frac{-3h + 2x - 2x}{2h^2} = \frac{-3h}{2h^2} = \frac{-3}{2h},$$

$$c_1 = \frac{d}{dt} \left(\frac{t - x_0}{x_1 - x_0} \frac{t - x_2}{x_1 - x_2} \right) \Big|_{t=x} = \frac{d}{dt} \left(\frac{t - x}{x + h - x} \frac{t - x - 2h}{x + h - x - 2h} \right) \Big|_{t=x} =$$

$$\frac{d}{dt} \left(\frac{t(t - 2x) + x^2 + h(-2t + 2x)}{-h^2} \right) \Big|_{t=x} = \frac{-2h + 2x - 2x}{-h^2} = \frac{2h}{h^2} = \frac{2}{h},$$

$$c_2 = \frac{d}{dt} \left(\frac{t - x_0}{x_2 - x_0} \frac{t - x_1}{x_2 - x_1} \right) \Big|_{t=x} = \frac{d}{dt} \left(\frac{t - x}{x + 2h - x} \frac{t - x - h}{x + 2h - x - h} \right) \Big|_{t=x} =$$

$$\frac{d}{dt} \left(\frac{t^2 + x(-2t + x) + h(-t + x)}{2h^2} \right) \Big|_{t=x} = \frac{-h + 2(x - x)}{2h^2} = \frac{-h}{2h^2} = \frac{-1}{2h},$$

sustituyendo estos coeficientes, la primera derivada de $f(x)$ en x_0 se aproxima como:

$$f'(x) \approx \frac{-\frac{3}{2}f(x) + 2f(x + h) - \frac{1}{2}f(x + 2h)}{h},$$

donde se puede apreciar que la expresión resultante es idéntica a la obtenida en (2.3.1.2).

2.3.2. Método del Paso Complejo

El método del paso complejo se basa en principios similares a los del método de diferencias finitas, pero introduce una modificación clave: el incremento entre los puntos es imaginario, es decir, se utiliza un pequeño valor complejo en lugar de un valor real para el desplazamiento. Esta innovadora aproximación permite aprovechar las propiedades de los números complejos, logrando eliminar términos de error en la aproximación de derivadas, lo cual hace que este método sea más preciso y menos susceptible a errores de cancelación [Berthold y Edwards, 2015, Baker y Hughes, 1997]. En comparación con las diferencias finitas, el método del paso complejo mantiene una alta precisión incluso cuando se emplean valores pequeños para el parámetro de paso h , debido a la ausencia de errores de cancelación, lo que significa que la precisión del método está puramente limitada por la precisión máxima de la máquina.

Destacar que el método de paso complejo, a pesar de usar números complejos, los puntos de interés y las derivadas obtenidas seguirán siendo parte del dominio de los números reales.

Comenzamos con la expansión de Taylor para $f(x + ih)$:

$$f(x + ih) = f(x) + ihf'(x) - \frac{h^2}{2!}f''(x) - \frac{ih^3}{3!}f'''(x) + \dots$$

gracias a los números complejos, podemos obtener la parte imaginaria de esta expresión:

$$\text{Im}(f(x + ih)) = hf'(x) - \frac{h^3}{3!}f'''(x) + \dots$$

dividiendo por h :

$$\frac{\text{Im}(f(x + ih))}{h} = f'(x) - \frac{h^2}{3!}f'''(x) + \dots$$

de esta manera, podemos expresar la derivada como:

$$f'(x) = \frac{\text{Im}(f(x + ih))}{h} + \mathcal{O}(h^2),$$

$$f'(x) \approx \frac{\text{Im}(f(x + ih))}{h}.$$

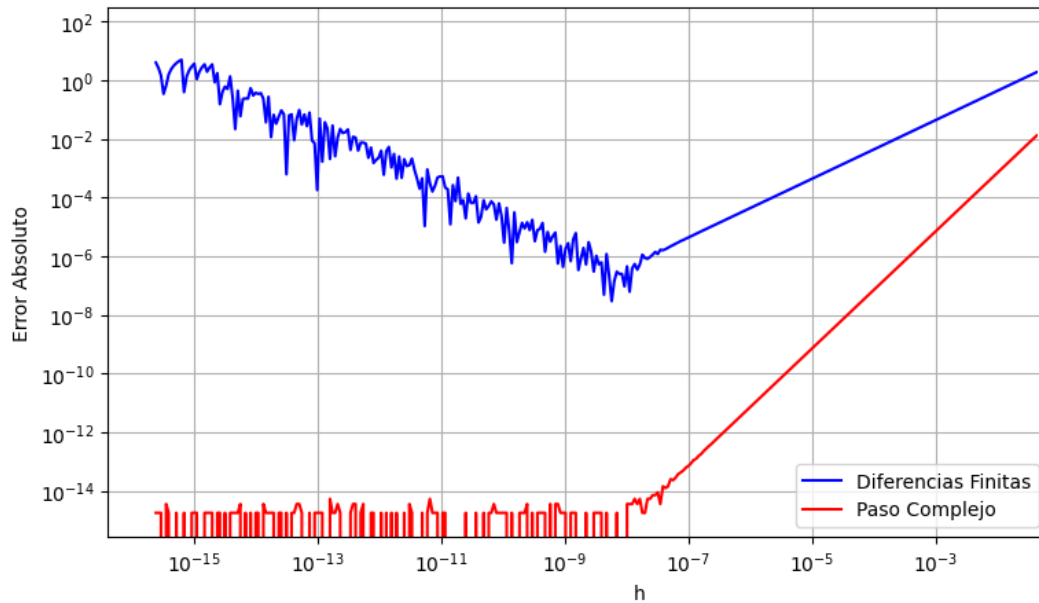


Figura 5: Error absoluto de la aproximación por diferencias finitas y paso complejo de $\left. \frac{d}{dx} \left(\frac{e^x}{\sin^3(x) + \cos^3(x)} \right) \right|_{x=3}$.

Fuente: Elaboración propia.

Al igual que en el caso de las diferencias finitas, el método del paso complejo también ofrece variaciones. La derivada utilizando el paso complejo hacia atrás se puede calcular de manera similar, siguiendo una expansión en serie de Taylor alrededor de un punto con un desplazamiento imaginario negativo [Baker y Hughes, 1997].

$$f(x - ih) = f(x) - ihf'(x) - \frac{h^2}{2!}f''(x) + \frac{ih^3}{3!}f'''(x) + \dots$$

extrayendo la parte imaginaria:

$$\text{Im}(f(x - ih)) = -hf'(x) + \frac{h^3}{3!}f'''(x) + \dots$$

procediendo de manera similar:

$$\frac{\text{Im}(f(x - ih))}{h} = -f'(x) + \frac{h^2}{3!}f'''(x) + \dots$$

finalmente, se obtiene:

$$f'(x) = -\frac{\text{Im}(f(x - ih))}{h} + \mathcal{O}(h^2),$$

$$f'(x) \approx -\frac{\text{Im}(f(x - ih))}{h}.$$

A su vez, para obtener la derivada a través del paso complejo central:

$$\text{Im}(f(x + ih)) = hf'(x) - \frac{h^3}{3!}f'''(x) + \dots$$

$$\text{Im}(f(x - ih)) = -hf'(x) + \frac{h^3}{3!}f'''(x) + \dots$$

al combinar estas dos expresiones:

$$\text{Im}(f(x + ih) - f(x - ih)) = 2hf'(x) - \frac{2h^3}{3!}f'''(x) + \dots$$

dividiendo por $2h$:

$$f'(x) = \frac{\text{Im}(f(x + ih) - f(x - ih))}{2h} + \mathcal{O}(h^2),$$

$$f'(x) \approx \frac{\text{Im}(f(x + ih) - f(x - ih))}{2h}.$$

Es interesante notar que el error del método de paso complejo central tiene el mismo orden de magnitud respecto a h que el método que utiliza solo un punto, cosa que no ocurre en el caso de las diferencias finitas. Esto nos indica que añadir más pasos alrededor del punto de interés puede llegar a ser contraproducente, pues estamos añadiendo computación al método sin reducir el error.

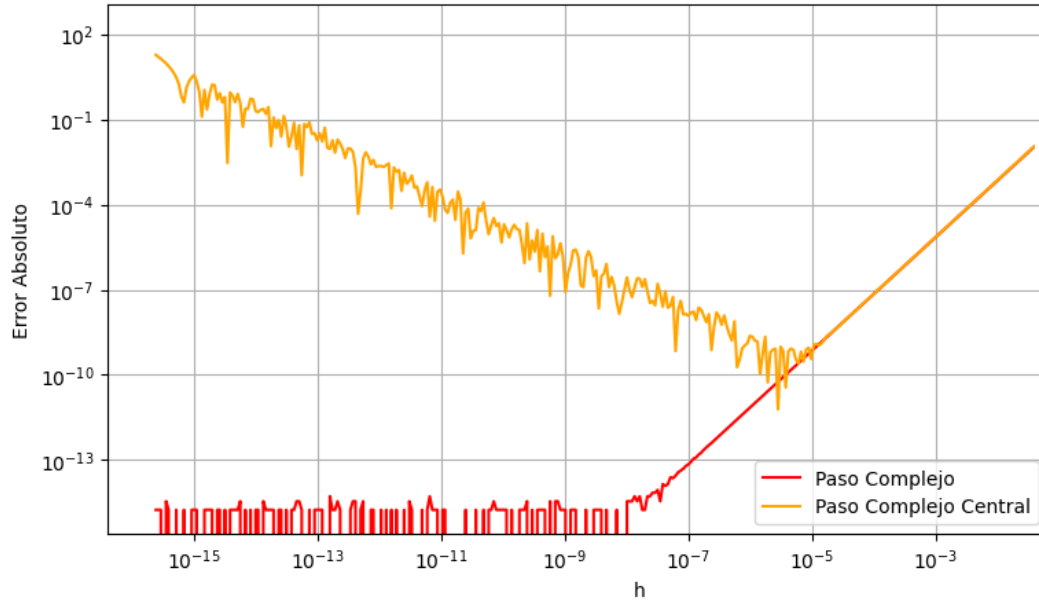


Figura 6: Error absoluto de la aproximación por paso complejo hacia adelante y central de $\left. \frac{d}{dx} \left(\frac{e^x}{\sin^3(x) + \cos^3(x)} \right) \right|_{x=3}$.

Fuente: Elaboración propia.

Nótese que la figura muestra que el error del método de paso complejo central empieza a aumentar tras un punto crítico; esto será discutido más adelante.

2.3.2.1. Convergencia y Error en el Método de Paso Complejo

El método de paso complejo es una extensión interesante de los métodos de diferencias finitas, que utiliza una pequeña perturbación imaginaria en lugar de una real para calcular derivadas numéricas con alta precisión y estabilidad. Al igual que en el caso de las diferencias finitas, existen infinitos posibles esquemas de paso complejo. Sin embargo, mientras que en las diferencias finitas es posible construir soluciones robustas al sistema lineal generando más puntos, en el método de paso complejo esta estrategia no siempre garantiza una reducción del error, ya que no todos los conjuntos de puntos adicionales aseguran que exista una solución con estos.

2.3.2.2. Método del Sistema Lineal para Coeficientes de Paso Complejo

De forma similar a las diferencias finitas, se emplean los factores de los términos de la expansión de Taylor para construir la matriz del sistema, esta vez considerando que solo usaremos

aquellos que tengan parte imaginaria, pues al aplicar la función Im el resto será eliminado, quedando las columnas de la matriz definidas como:

$$\mathbf{a}_i = \left(\frac{k_i^1}{1!} \quad \frac{k_i^3}{3!} \quad \frac{k_i^5}{5!} \cdots \frac{k_i^{2n-1}}{(2n-1)!} \right),$$

siendo n la cantidad de puntos que se utilizaron en total y k_i la proporción del factor h .

En teoría, cada punto adicional contribuye a eliminar términos de error, sin embargo, la elección de puntos complejos arbitrarios no garantiza que el sistema sea resoluble; esto significa que puede que no tenga solución si los puntos no están adecuadamente elegidos. Esto se debe a que la matriz del sistema puede perder independencia lineal en sus filas o columnas debido a las propiedades cíclicas de los números complejos.

Un ejemplo de construcción del sistema lineal con puntos $\{x + ih, x + 2ih, x + 3ih, x + 4ih\}$ muestra cómo se pueden emplear las expansiones de Taylor en los puntos complejos seleccionados. La siguiente ecuación ilustra este sistema:

$$A \cdot \mathbf{b} = \begin{pmatrix} \frac{1}{1!} & \frac{2}{1!} & \frac{3}{1!} & \frac{4}{1!} \\ \frac{-1^3}{3!} & \frac{-2^3}{3!} & \frac{-3^3}{3!} & \frac{-4^3}{3!} \\ \frac{1^5}{5!} & \frac{2^5}{5!} & \frac{3^5}{5!} & \frac{4^5}{5!} \\ \frac{-1^7}{7!} & \frac{-2^7}{7!} & \frac{-3^7}{7!} & \frac{-4^7}{7!} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

En este sistema el vector $\mathbf{b}$ representa los pesos que multiplicarán cada evaluación de la función. La expresión de la derivada aproximada utilizando estos cuatro puntos es:

$$f'(x) \approx \frac{\text{Im}(b_0 \cdot f(x + ih) + b_1 \cdot f(x + 2ih) + b_2 \cdot f(x + 3ih) + b_3 \cdot f(x + 4ih))}{h}.$$

Por otro lado, al intentar aplicar este método en un esquema de paso complejo central, usando los puntos $\{x - ih, x + ih\}$, el sistema pierde solubilidad, ya que la matriz se transforma en singular[Quarteroni y Saleri, 2007]. Considerando el sistema generado por estos dos puntos:

$$A \cdot \mathbf{b} = \begin{pmatrix} \frac{-1}{1!} & \frac{1}{1!} \\ \frac{-1^3}{3!} & \frac{1^3}{3!} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Las matrices singulares, caracterizadas por un determinante nulo, no pueden invertirse y reflejan una falta de independencia lineal entre las ecuaciones. En nuestro ejemplo puede verse claro, pues una columna es simplemente la otra multiplicada por -1 ; esto significa que no existe solución, pues al ser el vector de la derecha un vector canónico, al realizar la multiplicación $A \cdot \mathbf{b}$ obtendremos una restricción de $1 = 0$, lo cual es una contradicción.

El problema de singularidad implica que no es posible determinar los coeficientes b_0 y b_1 de manera que se anulen todos los términos de error. Sin embargo, esto no significa que no pueda diseñarse un método de paso complejo utilizando estos puntos. En particular, este caso corresponde al ejemplo presentado en la ecuación (2.3.2). Lo que la no resolubilidad del problema realmente indica es que el conjunto de puntos considerado no reduce el orden del error en comparación con un conjunto más pequeño de puntos.

En consecuencia, no resulta conveniente utilizar este conjunto de puntos, ya que el punto adicional no contribuye a minimizar el error y solo introduce un costo computacional innecesario.

2.3.2.3. Dominio del método

Este método es fascinante, ya que permite obtener una expresión para la derivada con un error que disminuye cuadráticamente, además de ser inmune a errores de cancelación en el método, puesto que no hace ningún tipo de resta. Esto es especialmente útil en contextos numéricos, donde los errores de cancelación pueden afectar la precisión, como en métodos más tradicionales como las diferencias finitas. Sin embargo, para aplicar el método del paso complejo de forma adecuada, es esencial que la función evaluada esté bien definida y sea diferenciable en el ámbito de los números complejos, lo que implica que debe ser holomorfa, es decir, analítica en un vecindario del punto considerado [Needham, 1997].

Una función compleja $f(z)$ es holomorfa en su dominio si es diferenciable en todos los puntos de este, y además su derivada es continua en todos los puntos. Esto implica que $f(z)$ satisface las ecuaciones de Cauchy-Riemann [James, 2011] en ese dominio. Matemáticamente, si $f(z) = u(x, y) + iv(x, y)$ es una función compleja de una variable $z = x + iy$, donde $u(x, y)$ y $v(x, y)$ son las funciones reales que representan las partes real e imaginaria de $f(z)$, la condición de que f sea holomorfa es que las siguientes ecuaciones se cumplan en su dominio:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \quad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}.$$

Estas ecuaciones garantizan que la función es localmente representable por una serie de potencias y que tiene propiedades de suavidad y regularidad en el dominio.

Aunque esta condición puede parecer una limitación, la mayoría de las funciones elementales cumplen con esta propiedad. Por ejemplo, los polinomios, las funciones exponenciales y las funciones trigonométricas son todas holomorfas. Funciones como el logaritmo y las raíces, por otro lado, pueden presentar puntos críticos, como en 0, pero esto es un comportamiento esperado que también ocurre en su dominio real. Este hecho sugiere que la mayoría de las funciones comunes a las que se aplicará este método no requerirán transformaciones adicionales.

2.3.3. Diferenciación Automática

La diferenciación automática es un método usado para calcular derivadas de manera más precisa mediante la regla de la cadena. Este método se distingue de las aproximaciones numéricas, como las diferencias finitas, y de la diferenciación simbólica en que no es una aproximación y es más eficiente en términos computacionales en comparación con la diferenciación simbólica completa [Corliss *et al.*, 2013].

A diferencia de las diferencias finitas, la diferenciación automática se basa en descomponer una función en operaciones elementales (como suma, producto y composición) que, en conjunto, construyen la función completa. Para cada operación elemental, se calcula la derivada exacta, aprovechando la linealidad y la regla de la cadena para propagar estas derivadas a través de la estructura de la función. Matemáticamente, la derivada de una función compuesta se representa como una secuencia de derivadas de funciones elementales:

$$\frac{d}{dx}f(g(x)) = f'(g(x)) \cdot g'(x).$$

La diferenciación automática se clasifica en dos enfoques principales: el **modo hacia adelante** y el **modo hacia atrás**. Sin embargo, su funcionamiento esencial es el mismo. Dado que este análisis no busca profundizar en este método, ni en las diferencias entre estos enfoques, se omitirán los detalles relacionados, pues no se consideran pertinentes.

2.3.3.1. Ventajas de la Diferenciación Automática

Una de las principales ventajas de la diferenciación automática es su precisión: a diferencia de las aproximaciones de diferencias finitas, no introduce errores de cancelación, ya que calcula la derivada exacta de cada operación elemental. Además, en comparación con el cálculo simbólico, la diferenciación automática es computacionalmente menos intensiva y evita el problema de “explosión de expresiones”, que ocurre cuando las expresiones simbólicas se vuelven incontrolablemente complejas [Naumann, 2012].

La diferenciación automática ha encontrado aplicaciones fundamentales en campos que requieren cálculos de gradientes precisos y eficientes. En optimización y aprendizaje automático, especialmente en el entrenamiento de redes neuronales profundas, la diferenciación automática es esencial, ya que permite calcular gradientes en funciones complejas que involucran millones de parámetros. Este método también es útil en simulaciones numéricas donde se requieren gradientes precisos para estimar la sensibilidad de los resultados a las condiciones iniciales.

2.3.3.2. Limitaciones y Comparación con Otros Métodos

A pesar de sus ventajas, este método no es adecuado para todos los problemas. Por ejemplo, en modelos donde las funciones y variables se actualizan de forma dinámica, como en simulaciones que involucran datos en tiempo real, la diferenciación automática puede ser menos flexible debido a todo el proceso de mapeado que debe hacer. Además, su implementación puede ser compleja en funciones que incluyen bucles o condiciones discontinuas, sumándole que el método es significativamente más lento que los otros métodos numéricos discutidos.

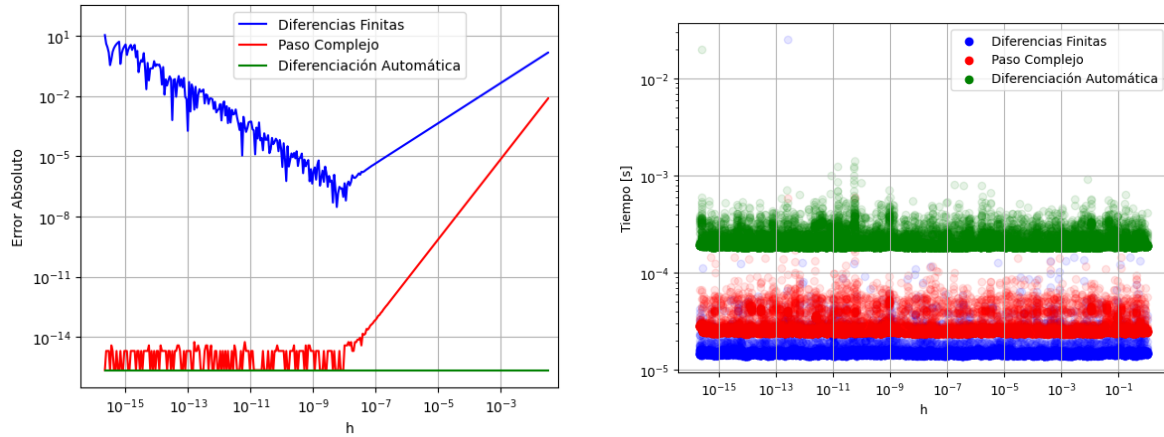


Figura 7: Error absoluto y tiempo de cómputo de la derivada, calculada mediante diferencias finitas, paso complejo y diferenciación automática de $\frac{d}{dx} \left(\frac{e^x}{\sin^3(x) + \cos^3(x)} \right) \Big|_{x=3}$

Fuente: Elaboración propia.

Debido a que el enfoque de diferenciación automática difiere en su aplicación y naturaleza de métodos como las diferencias finitas y el método del paso complejo, este trabajo no se centrará en un análisis profundo de la diferenciación automática. Sin embargo, este método se utilizará como punto de comparación en experimentos específicos, ya que proporciona un cálculo de alta precisión, lo cual permitirá contrastar resultados en contextos donde se espera precisión numérica alta.

2.3.4. Derivadas de Orden Superior

Ambos métodos, diferencias finitas y paso complejo, no se limitan al cálculo de derivadas de primer orden; pueden ser reestructurados para calcular derivadas de orden arbitrario. Como ejemplo, se presenta la siguiente expresión de diferencias finitas para obtener la segunda derivada utilizando los puntos $\{x, x + h, x + 2h\}$. Este método tiene un error de orden h :

$$f''(x) = \frac{f(x) - 2f(x + h) + f(x + 2h)}{h^2} + \mathcal{O}(h),$$

En esta expresión, los coeficientes $\{1, -2, 1\}$ se obtienen al cancelar términos correspondientes en las series de Taylor. Alternativamente, estos coeficientes pueden determinarse empleando el método del sistema lineal descrito previamente. El único ajuste necesario consiste en reubicar el valor 1 en el vector del lado derecho del sistema especificado en (2.3.1.2).

Para calcular la derivada de orden n , el valor 1 debe situarse en la posición $n + 1$ del vector, y la expresión resultante debe dividirse por h^n .

El sistema lineal para calcular los coeficientes de la derivada n , considerando el conjunto de puntos $K = \{k_0h, \dots, k_{n-1}h, \dots, k_{m-1}h\}$, donde $m \geq n + 1$, usando el vector canónico $\mathbf{e}_i \in \mathbb{R}^m$ puede representarse como:

$$A \cdot \mathbf{b} = \begin{pmatrix} \mathbf{a}_0^T & \cdots & \mathbf{a}_{n-1}^T & \cdots & \mathbf{a}_{m-1}^T \end{pmatrix} \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \\ \vdots \\ b_{m-1} \end{pmatrix} = \mathbf{e}_{n+1}.$$

$$\mathbf{a}_i = \left(\frac{k_i^0}{0!} \quad \cdots \quad \frac{k_i^{n-1}}{(n-1)!} \quad \cdots \quad \frac{k_i^{m-1}}{(m-1)!} \right)^T,$$

y la expresión final resultante corresponde a:

$$f^{(n)}(x) \approx \frac{\sum_{i=0}^{m-1} b_i f(x + k_i h)}{h^n}.$$

Este método es una generalización del descrito en (2.3.1.2).

Para obtener los coeficientes del paso complejo, se aplica el siguiente metodo, similar pero con una sutil diferencia.

Para calcular la derivada de orden n , el valor 1 debe situarse en la posición $l = \left\lceil \frac{n+1}{2} \right\rceil$ del vector, y la expresión resultante debe dividirse por h^n .

El sistema lineal para calcular los coeficientes de la derivada n , considerando el conjunto de puntos $K = \{k_0ih, \dots, k_{n-2}ih, \dots, k_{m-1}ih\}$, donde $m \geq n$, usando el vector canónico $\mathbf{e}_i \in \mathbb{R}^m$ puede representarse como:

$$A \cdot \mathbf{b} = \begin{pmatrix} \mathbf{a}_0^T & \cdots & \mathbf{a}_{n-2}^T & \cdots & \mathbf{a}_{m-1}^T \end{pmatrix} \begin{pmatrix} b_0 \\ \vdots \\ b_{n-2} \\ \vdots \\ b_{m-1} \end{pmatrix} = \mathbf{e}_l.$$

¹En caso de que $k_i = 0$, se define $0^0 = 1$

$$\mathbf{a}_j = G_n \left(\left(\frac{(ik_j)^0}{0!} \quad \dots \quad \frac{(ik_j)^{n-2}}{(n-2)!} \quad \dots \quad \frac{(ik_j)^{(2m+1)-1}}{((2m+1)-1)!} \right) \right),$$

donde $G_n(\mathbf{x}) : \mathbb{R}^{2m+1} \rightarrow \mathbb{R}^m$ devuelve un nuevo vector con solo sus componentes en posiciones impares si n es par, y las componentes en posiciones pares en caso contrario, pero sin la unidad compleja que acompaña a estos.

Es decir, si n es una derivada par, devolverá:

$$G_n = \left(\frac{(ik_j)^0}{0!} \quad \frac{(ik_j)^2}{2!} \quad \frac{(ik_j)^4}{4!} \quad \dots \right),$$

y si es impar, devolverá:

$$G_n = \text{Im} \left(\left(\frac{(ik_j)^1}{1!} \quad \frac{(ik_j)^3}{3!} \quad \frac{(ik_j)^5}{5!} \quad \dots \right) \right).$$

y la expresión final resultante corresponde a:

$$f^{(n)}(x) \approx \begin{cases} \frac{\text{Im} \left(\sum_{j=0}^{m-1} b_j f(x + k_j i h) \right)}{h^n} & \text{si } n \text{ es impar,} \\ \frac{\text{Re} \left(\sum_{j=0}^{m-1} b_j f(x + k_j i h) \right)}{h^n} & \text{si } n \text{ es par.} \end{cases}$$

Este método es una generalización del descrito en (2.3.2.2).

Otro factor significativo referente al paso complejo, es que aquí la expresión contiene una resta, lo que la hace susceptible a errores de cancelación, un fenómeno que se buscaba evitar. A pesar de esto, el error aún converge de forma veloz en comparación con su contraparte en diferencias finitas, lo que significa que en ciertas situaciones, el método sigue siendo eficiente para obtener una aproximación precisa de la derivada.

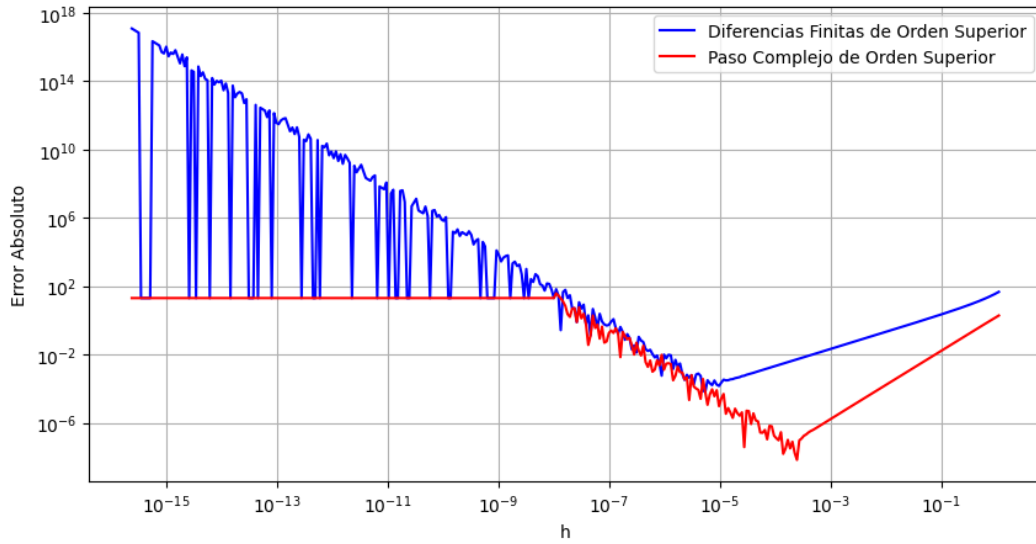


Figura 8: Error absoluto de la aproximación por diferencias finitas y paso complejo utilizando los puntos $\{0, h, 2h\}$ y $\{0, ih\}$ respectivamente, de $\frac{d^2}{dx} (e^x + \sin(x)) \Big|_{x=3}$.

Fuente: Elaboración propia.

En resumen, la obtención de derivadas de orden superior mediante diferencias finitas y el método de paso complejo depende crucialmente de la selección adecuada de los puntos de evaluación y de una comprensión precisa de los errores asociados. Ambos métodos presentan ventajas y limitaciones que deben analizarse en función del contexto específico de la aplicación.

Es importante considerar también que una elección inadecuada de los puntos puede conducir a un sistema de ecuaciones irresoluble debido a problemas de singularidad en la construcción de la aproximación del método de paso complejo.

Al igual que en el caso de las derivadas de primer orden, existen otros métodos para calcular los coeficientes necesarios. Sin embargo, profundizar en estos métodos excede el alcance del presente análisis, cuyo objetivo principal es comparar los resultados obtenidos mediante cada técnica. Se ha decidido incluir una breve exposición de estos procesos de obtención de coeficientes con el propósito de resaltar los paralelismos entre el método de paso complejo y las diferencias finitas, demostrando que ambos enfoques pueden cumplir funciones similares.

2.3.5. Cálculo de Derivadas en Espacios Multidimensionales

El análisis en este contexto se desarrolla dentro del marco del álgebra lineal, con el objetivo de aproximar de manera eficiente el resultado del producto de la matriz Jacobiana por

un vector. Esta aproximación es crucial, ya que, en la práctica, la matriz Jacobiana rara vez constituye el resultado final deseado, siendo comúnmente un componente intermedio de una operación matemática más compleja. Por ello, es fundamental garantizar que todos los métodos estudiados hasta ahora puedan traducirse y funcionar correctamente en contextos multivariados.

Consideremos una función vectorial $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$, definida por componentes $f_1, f_2, \dots, f_n$ de la forma:

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{bmatrix}.$$

El Jacobiano de $\mathbf{F}$ en $\mathbf{x}$, denotado $J_{\mathbf{F}}(\mathbf{x})$, es la matriz de derivadas parciales de cada componente de $\mathbf{F}$ con respecto a cada variable $x_1, x_2, \dots, x_n$

$$J_{\mathbf{F}}(\mathbf{x}) = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \\ \vdots \\ \nabla f_n(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}.$$

El producto entre el Jacobiano y un vector $\mathbf{v} \in \mathbb{R}^n$ es el siguiente:

$$J_{\mathbf{F}}(\mathbf{x}) \cdot \mathbf{v} = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \\ \vdots \\ \nabla f_n(\mathbf{x}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}.$$

2.3.6. Diferencias finitas en Espacios Multidimensionales

El objetivo es aproximar este producto mediante un esquema de diferencias finitas. Para ello, podemos calcular cada componente del gradiente de cada función f_i y luego se realiza la combinación lineal con el vector $\mathbf{v}$.

La aproximación de las derivadas parciales mediante diferencias finitas se expresa como:

$$J_{\mathbf{F}}(\mathbf{x}) \cdot \mathbf{v} \approx \begin{bmatrix} \frac{f_1((x_1+h, x_2, \dots, x_n)) - f_1(\mathbf{x})}{h} & \frac{f_1((x_1, x_2+h, \dots, x_n)) - f_1(\mathbf{x})}{h} & \dots & \frac{f_1((x_1, x_2, \dots, x_n+h)) - f_1(\mathbf{x})}{h} \\ \frac{f_2((x_1+h, x_2, \dots, x_n)) - f_2(\mathbf{x})}{h} & \frac{f_2((x_1, x_2+h, \dots, x_n)) - f_2(\mathbf{x})}{h} & \dots & \frac{f_2((x_1, x_2, \dots, x_n+h)) - f_2(\mathbf{x})}{h} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{f_n((x_1+h, x_2, \dots, x_n)) - f_n(\mathbf{x})}{h} & \frac{f_n((x_1, x_2+h, \dots, x_n)) - f_n(\mathbf{x})}{h} & \dots & \frac{f_n((x_1, x_2, \dots, x_n+h)) - f_n(\mathbf{x})}{h} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

A este método se le conoce como explícito, donde se calcula la totalidad de la matriz jacobiana para después ser multiplicado por el vector $\mathbf{v}$; esta aproximación resulta ser computacionalmente costosa debido a que requiere una gran cantidad de cálculos. Una alternativa más eficiente consiste en calcular directamente el producto $J_{\mathbf{F}}(\mathbf{x}) \cdot \mathbf{v}$ mediante una perturbación en la dirección de $\mathbf{v}$.

Si aplicamos el método de diferencias finitas en la dirección de los vectores, obtenemos la siguiente expresión para la derivada de $\mathbf{F}$ respecto a x_i :

$$\frac{\partial \mathbf{F}}{\partial x_i} \approx \frac{\mathbf{F}(\mathbf{x} + h\mathbf{e}_i) - \mathbf{F}(\mathbf{x})}{h},$$

aquí, $\mathbf{e}_i$ es el vector canónico en la dirección de la variable x_i . Este tipo de derivada se denomina derivada direccional, y proporciona una forma de calcular la tasa de cambio de la función $\mathbf{F}$ en una dirección específica.

Gracias a las propiedades de la matriz Jacobiana, al multiplicarla por un vector arbitrario, obtenemos la derivada direccional [Math StackExchange User, 2019]. Utilizando este enfoque, podemos calcular directamente el producto Jacobiano-vector en la dirección de $\mathbf{v}$ de la siguiente manera:

$$J_{\mathbf{F}}(\mathbf{x}) \cdot \mathbf{v} \approx \frac{\mathbf{F}(\mathbf{x} + h\mathbf{v}) - \mathbf{F}(\mathbf{x})}{h},$$

$$J_{\mathbf{F}}(\mathbf{x}) \cdot \mathbf{v} \approx \begin{bmatrix} \frac{f_1((x_1+v_1h, x_2+v_2h, \dots, x_n+v_nh)) - f_1(\mathbf{x})}{h} \\ \frac{f_2((x_1+v_1h, x_2+v_2h, \dots, x_n+v_nh)) - f_2(\mathbf{x})}{h} \\ \vdots \\ \frac{f_n((x_1+v_1h, x_2+v_2h, \dots, x_n+v_nh)) - f_n(\mathbf{x})}{h} \end{bmatrix},$$

esta aproximación resulta ser más eficiente que el cálculo directo del Jacobiano y es ampliamente utilizada en métodos numéricos como el de Newton-Krylov para la resolución de sistemas no lineales.

2.3.7. Método de Paso Complejo en Espacios Multidimensionales

Una vez derivada la expresión para el producto jacobiano-vector utilizando diferencias finitas, su adaptación al método de paso complejo resulta directa. Aplicando el mismo razonamiento algebraico al empleado para desarrollar el método de diferencias finitas en múltiples dimensiones, se obtiene la siguiente expresión análoga para el caso del paso complejo:

$$\frac{\partial \mathbf{F}}{\partial x_i} \approx \frac{\text{Im}(\mathbf{F}(\mathbf{x} + ih\mathbf{e}_i))}{h},$$

por ende, el producto jacobiano vector resulta:

$$J_{\mathbf{F}}(\mathbf{x}) \cdot \mathbf{v} \approx \frac{\text{Im}(\mathbf{F}(\mathbf{x} + ih\mathbf{v}))}{h}$$

$$J_F(\mathbf{x}) \cdot \mathbf{v} \approx \begin{bmatrix} \frac{\text{Im}(f_1(x_1+iv_1h, x_2+iv_2h, \dots, x_n+iv_nh))}{h} \\ \frac{\text{Im}(f_2(x_1+iv_1h, x_2+iv_2h, \dots, x_n+iv_nh))}{h} \\ \vdots \\ \frac{\text{Im}(f_n(x_1+iv_1h, x_2+iv_2h, \dots, x_n+iv_nh))}{h} \end{bmatrix}$$

Con este simple cambio, hemos obtenido una expresión para calcular el producto jacobiano-vector utilizando el método de paso complejo, lo cual será de gran utilidad en futuros experimentos.

2.3.8. Método Paso Multi-Complejo

Para nuestra fortuna, existe una forma de calcular derivadas de orden superior y evitar los errores de cancelación.

Los números multicomplejos son una generalización de los números complejos en la cual se pueden tener una cantidad arbitraria de variables complejas independientes entre sí; de esta cantidad depende su grado. Mientras que los números complejos estándar están compuestos por una parte real y una parte imaginaria, los números multicomplejos también se dividen en dos partes: el equivalente a la parte real sería un número multicomplejo de grado inferior y el equivalente a la parte imaginaria sería otro número multicomplejo del mismo grado inferior multiplicado por una nueva componente imaginaria. [Luo et al., 2016].

Un pequeño ejemplo para conceptualizar puede ser el siguiente, donde *Monocomplejo* corresponde a los números complejos estándar, que corresponden a un Multicomplejo de grado 1, y *Bicomplejo* corresponde a un Multicomplejo de grado 2.

Monocomplejo: $a + bi$

Bicomplejo: $(a + bi) + (c + di)j = a + bi + cj + dij$

Podemos observar que el número bicomplejo se compone de dos números complejos, uno de ellos multiplicado por una constante a la que llamamos j ; este número es una nueva constante imaginaria, totalmente independiente de i .

La aritmética de los números multicomplejos queda a definir según las necesidades del problema a resolver, siendo unos de los más famosos los *quaterniones*, usados principalmente para representar rotaciones en tres dimensiones [Kuipers, 2020]. Sin embargo, esta no es la definición que necesitamos para el problema a resolver.

Rigurosamente, los números multicomplejos pueden ser generalizados de la siguiente forma, donde el conjunto de los multicomplejos de grado n se describe mediante la relación recursiva:

$$\mathbb{C}^n = \{z_1 + z_2 i_n \mid z_1, z_2 \in \mathbb{C}^{n-1}, i_n^2 = -1\},$$

$$\mathbb{C}^0 = \mathbb{R}.$$

Donde, al analizar una correspondiente descomposición, se puede observar que para representar un número perteneciente a $\mathbb{C}^n$ se requieren un total de 2^n coeficientes reales.

2.3.8.1. Propiedades Algebraicas

Para cualquier unidad imaginaria i_n se debe cumplir que $i_n^2 = -1$. Además, no existe ninguna identidad que vincule productos entre unidades imaginarias de diferente grado; esto quiere decir que el producto entre unidades imaginarias $i_n \cdot i_m$ con $n \neq m$ no corresponde a ningún otro valor ni puede ser simplificado de ninguna forma. [Fritzsche, 2015].

Con estas definiciones, los números complejos y por consecuencia, multicomplejos, pueden representarse en forma matricial utilizando extensiones de la unidad real e imaginaria [Verheyleweghen, 2014], por ejemplo :

$$1 \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad i \equiv \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

La estructura resultante para números en $\mathbb{C}^n$ puede representarse como:

$$\mathbb{C}^n = \left\{ \begin{bmatrix} z_1 & -z_2 \\ z_2 & z_1 \end{bmatrix} \mid z_1, z_2 \in \mathbb{C}^{n-1} \right\}.$$

Al considerar números multicomplejos en $\mathbb{C}^2$, obtenemos una matriz de tamaño 4×4 :

$$\mathbb{C}^2 = \left\{ \begin{bmatrix} x_{11} & -x_{12} & -x_{21} & x_{22} \\ x_{12} & x_{11} & -x_{22} & -x_{21} \\ x_{21} & -x_{22} & x_{11} & -x_{12} \\ x_{22} & x_{21} & x_{12} & x_{11} \end{bmatrix} \mid x_{ij} \in \mathbb{R} \right\}.$$

Es evidente que el tamaño de la matriz crece exponencialmente conforme aumenta el grado del número multicomplejo.

Esta representación de los números multicomplejos evidencia que constituyen una extensión natural de los números complejos. Su álgebra elemental opera de manera esencialmente idéntica a la de los números complejos clásicos, aplicándose de forma recursiva entre sus componentes [Verheyleweghen, 2014].

A continuación, se muestran los paralelismos entre la evaluación compleja y multicompleja de la función $\cos(x)$:

$$z = x_1 + x_2 i, \quad z \in \mathbb{C}, \quad x_1, x_2 \in \mathbb{R},$$

$$\cos(z) = \cos(x_1) \cosh(x_2) - i \sin(x_1) \sinh(x_2).$$

$$z_m = x_1 + x_2 i_m, \quad z_m \in \mathbb{C}^m, \quad x_1, x_2 \in \mathbb{C}^{m-1},$$

$$\cos(z_m) = \cos(x_1) \cosh(x_2) - i_m \sin(x_1) \sinh(x_2).$$

2.3.8.2. Extracción de nuevas componentes imaginarias

Tal como existe el método de paso complejo, también existe el método del paso multicomplejo, el cual nos permite obtener derivadas de orden superior sin la necesidad de tener que enfrentarnos a errores de cancelación [Fritzsche, 2015].

$$f'(x) \approx \frac{\text{Im}(f(x + hi))}{h}.$$

Para pasar el método del paso complejo a un contexto multicomplejo, la función Im se redefine para manejar múltiples unidades imaginarias:

$$\text{Im}_{\sigma_k} : \mathbb{C}^n \rightarrow \mathbb{R}, \quad \text{Im}_{\sigma_k}(z) = x_{\sigma_k}$$

Donde σ_k denota el conjunto del producto de unidades imaginarias al que deseamos obtener su componente, siendo k un enumerador para estas, por ejemplo, σ_1 denota el conjunto $\{1\}$, que corresponde al valor i_1 , σ_3 denota el conjunto $\{1, 2\}$, que corresponde a $i_1 \cdot i_2$ y σ_{11} denota el conjunto $\{1, 3, 4\}$, que corresponde a $i_1 \cdot i_3 \cdot i_4$.

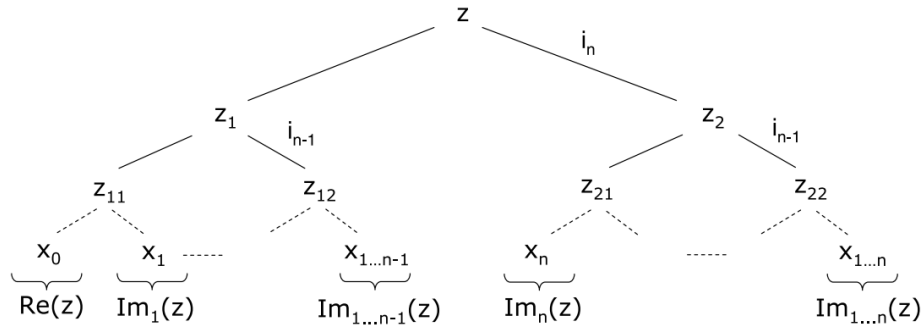


Figura 9: Representación de árbol de número multicomplejo de grado n .
Fuente: [2012].

2.3.8.3. Derivación Método de Paso Multicomplejo

Con la definición de números multicomplejos, es posible extender fácilmente el método de paso complejo al dominio multicomplejo para calcular derivadas de orden superior. Esta idea fue explorada por primera vez por Lantoine et al. en 2012 [2012].

Supongamos que f es una función holomorfa, es decir, infinitamente diferenciable, las mismas condiciones necesarias para el paso complejo. La serie de Taylor de f alrededor del punto $x_0 + i_1 h + \dots + i_n h$ se puede expresar como [Verheyleweghen, 2014]:

$$f(x_0 + i_1 h + \dots + i_n h) = f(x_0) + (i_1 h + \dots + i_n h) f'(x_0) + \frac{(i_1 h + \dots + i_n h)^2}{2!} f''(x_0) + \dots$$

$$f(x_0 + i_1 h + \dots + i_n h) = \sum_{j=0}^{\infty} \left(\left(\sum_{l=1}^n i_l \cdot h \right)^j \frac{f^{(j)}(x_0)}{j!} \right).$$

El término $\left(\sum_{l=1}^n i_l \cdot h \right)^j$ se puede expandir utilizando el teorema multinomial, el cual establece que:

$$\left(\sum_{i=1}^m x_i \right)^n = \sum_{k_1+k_2+\dots+k_m=n} \binom{n}{k_1, k_2, \dots, k_m} \prod_{i=1}^m x_i^{k_i},$$

donde el coeficiente multinomial se define como:

$$\binom{n}{k_1, k_2, \dots, k_m} = \frac{n!}{k_1! k_2! \dots k_m!},$$

descomponiendo el termino de la expansion de Taylor obtenemos lo siguiente:

$$\left(\sum_{l=1}^n i_l \cdot h \right)^j = \sum_{k_1+k_2+\dots+k_m=j} \frac{n!}{k_1! \dots k_m!} \prod_{l=1}^n (i_l \cdot h)^{k_l}.$$

La derivada de orden n es la única que contiene el término único $\prod_{l=1}^n i_l$, como se puede observar en el teorema multinomial. En este caso, nosotros estamos buscando la expresión que contenga el producto de todas las unidades imaginarias. En la productoria podemos ver que tales unidades están elevadas a k_l . Por la propiedad $i_n^2 = -1$, podemos llegar a ver que $k_1 = k_2 = \dots = k_n = 1$, pues por las propiedades cíclicas de estos números, estos desaparecerían del producto al ser elevados a otros enteros.

A la vez, por las mismas propiedades cíclicas, significa que esta expresión que hemos buscado no es única y estará presente otras infinitas veces², de forma homóloga al método de paso complejo tradicional; estos valores serán nuestro error en la aproximación.

²Debido a que cada i_n es independiente, sus productos serán cíclicos, dependiendo del orden del polinomio generado por Taylor.

Usando la función Im_{σ_k} definida en (2.3.8.2), se deduce que la derivada de orden n se puede calcular como:

$$f^{(n)}(x) \approx \frac{\text{Im}_{\sigma_{2^n-1}} \left(f \left(x + h \sum_{k=1}^n i_k \right) \right)}{h^n},$$

la cual es muy similar a la aproximación de primer orden del método de paso complejo. Al hacer h suficientemente pequeño, se puede obtener cualquier precisión deseada, limitada únicamente por la precisión de la máquina.

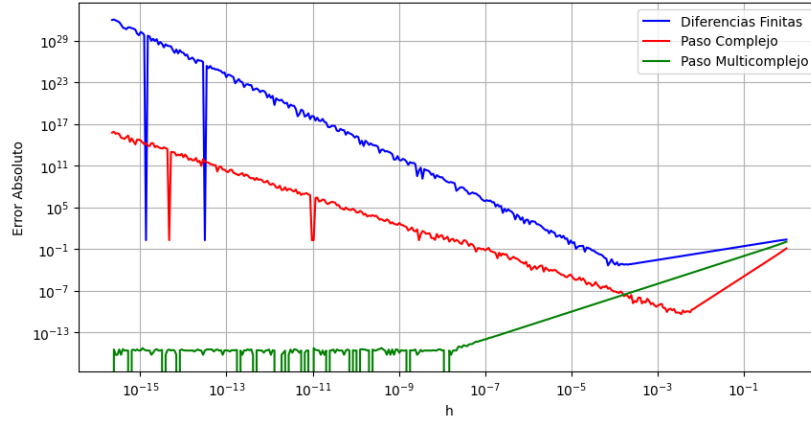


Figura 10: Error absoluto en la 3ª derivada de la función $\sin(\mathbf{x}) + \cos(\mathbf{x})$ en el punto $\mathbf{x} = \{1, 2, 3\}$, en la dirección $\mathbf{v} = \mathbf{1}_3$, calculado mediante el método de diferencias finitas, paso complejo utilizando los puntos $\{0, h, 2h, 3h\}$ y $\{ih, 2ih, 3ih\}$ respectivamente, y paso multicomplejo.

Fuente: Elaboración propia.

2.3.8.4. Derivadas Multidimensionales y Parciales

De manera análoga a lo descrito en la sección anterior, el método de paso multicomplejo puede extenderse a $\mathbb{R}^n$, lo que permite calcular derivadas parciales y direccionales. Sin embargo, la incorporación de múltiples unidades imaginarias ofrece un mayor control, ya que posibilita distribuir los pequeños incrementos de estas unidades de manera eficiente. Esto facilita la obtención de derivadas parciales de forma más intuitiva y natural [2012] a comparación de sus métodos hermanos, la diferencia finita y el paso complejo tradicional.

Las derivadas parciales de orden n se pueden aproximar por metodos tradicionales utilizando la fórmula:

$$\frac{\partial^n f(x_1, \dots, x_p)}{\partial x_1^{k_1} \dots \partial x_p^{k_p}} = \frac{\partial}{\partial x_1^{k_1}} \left(\frac{\partial}{\partial x_2^{k_2}} \dots \left(\frac{\partial f(x_1, \dots, x_p)}{\partial x_p^{k_p}} \right) \right),$$

$$\frac{\partial}{\partial x_i^{k_i}} = \frac{\partial}{\partial x_i} \left(\frac{\partial}{\partial x_i} \cdots k_i \text{ veces} \cdots \left(\frac{\partial f(x_1, \dots, x_p)}{\partial x_i} \right) \right),$$

donde cada término $\frac{\partial}{\partial x_i}$ puede aproximarse mediante métodos tradicionales, como se expone en (2.3.6).

Por otro lado, las derivadas parciales de orden n pueden aproximarse utilizando números multicomplejos mediante la siguiente fórmula:

$$\frac{\partial^n f(x_1, \dots, x_p)}{\partial x_1^{k_1} \cdots \partial x_p^{k_p}} \approx \frac{\text{Im}_{\sigma_{2^n-1}}(f(x_1 + h \sum_{j \in \mathcal{I}_1} i_j, \dots, x_p + h \sum_{j \in \mathcal{I}_p} i_j))}{h^n},$$

donde $k_1 + \dots + k_p = n$ y $\mathcal{I}_j$ son los índices correspondientes a las direcciones imaginarias asociadas a cada variable, estas son unicas para cada derivada que se desee calcular, y $\text{Im}_{\sigma_{2^n-1}}$ extrae el coeficiente real asociado a la combinación de componentes imaginarias seleccionadas.

Para una función de dos variables $f(x, y)$, las derivadas de segundo orden pueden calcularse como:

$$\begin{aligned} \frac{\partial^2 f(x, y)}{\partial x^2} &\approx \frac{\text{Im}_{12}(f(x + h(i_1 + i_2), y))}{h^2}, \\ \frac{\partial^2 f(x, y)}{\partial y^2} &\approx \frac{\text{Im}_{12}(f(x, y + h(i_1 + i_2)))}{h^2}, \\ \frac{\partial^2 f(x, y)}{\partial x \partial y} &\approx \frac{\text{Im}_{12}(f(x + hi_1, y + hi_2))}{h^2} = \frac{\text{Im}_{12}(f(x + hi_2, y + hi_1))}{h^2}. \end{aligned}$$

2.3.8.5. Conclusión

El método de paso multicomplejo representa una extensión poderosa del método de paso complejo, permitiendo el cálculo de derivadas de orden superior y multidimensionales con alta precisión y sin errores de cancelación. Este enfoque se fundamenta en la utilización de números multicomplejos, los cuales proporcionan múltiples unidades imaginarias independientes, permitiendo distribuir los incrementos necesarios de forma eficiente y natural.

Gracias a sus propiedades algebraicas y su capacidad de generalización, el método de paso multicomplejo es capaz de aproximar derivadas parciales y direccionales en $\mathbb{R}^n$ de manera intuitiva. Además, la estructura recursiva de los números multicomplejos asegura una implementación sistemática y escalable para problemas complejos. Esto lo convierte en una herramienta invaluable en la resolución de sistemas no lineales y el análisis de modelos matemáticos que requieren derivadas de alta precisión.

A pesar de sus ventajas, el método de paso multicomplejo presenta ciertas limitaciones que deben ser consideradas:

1. **Complejidad computacional:** La representación de números multicomplejos y sus operaciones implican un crecimiento exponencial en el número de componentes reales a medida que aumenta el grado n , lo que puede resultar en un aumento significativo en el tiempo y los recursos computacionales necesarios.

2. **Implementación técnica:** La aritmética multicompleja y su integración en sistemas computacionales presentan desafíos y su implementación en esta memoria no fue trivial, ya que este sistema numérico sigue siendo un área de nicho y existen pocas, si es que no ninguna, librerías ampliamente utilizadas que lo implementen de forma nativa.

Estas limitaciones deben ser evaluadas cuidadosamente al aplicar el método, especialmente en escenarios con restricciones computacionales o requisitos específicos que puedan impactar su rendimiento o viabilidad.

2.4. Método de Newton-Krylov

El método de Newton-Krylov es una técnica numérica ampliamente utilizada para resolver sistemas de ecuaciones no lineales en problemas de alta dimensionalidad. Este método combina la iteración del método de Newton, que busca soluciones aproximadas a sistemas no lineales, con métodos iterativos de espacios de Krylov, utilizados para resolver eficientemente sistemas lineales asociados sin necesidad de construir explícitamente el Jacobiano [Nocedal y Wright, 2006].

El objetivo del método es encontrar una solución $\mathbf{x}_r \in \mathbb{R}^n$ tal que $\mathbf{F}(\mathbf{x}_r) = 0$, donde $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ es una función no lineal. En cada iteración del método de Newton, la solución se aproxima mediante la fórmula:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_{\mathbf{F}}(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k),$$

donde $J_{\mathbf{F}}(\mathbf{x}_k)$ es la matriz Jacobiana de $\mathbf{F}(\mathbf{x})$ evaluada en $\mathbf{x}_k$. Sin embargo, calcular $J_{\mathbf{F}}(\mathbf{x}_k)^{-1}$ directamente es computacionalmente costoso, especialmente en sistemas de gran escala.

El método de Newton-Krylov aborda este desafío al reemplazar la resolución exacta del sistema lineal $J_{\mathbf{F}}(\mathbf{x}_k) \Delta \mathbf{x} = -\mathbf{F}(\mathbf{x}_k)$ con métodos iterativos de Krylov, como GMRES (Generalized Minimal Residual). Estos métodos buscan soluciones aproximadas en un subespacio de Krylov construido a partir del residuo inicial $\mathbf{r}_0 = -\mathbf{F}(\mathbf{x}_k)$. Esto permite calcular productos Jacobiano-vector $J_{\mathbf{F}}(\mathbf{x}_k) \mathbf{v}$ sin construir explícitamente $J_{\mathbf{F}}(\mathbf{x}_k)$.

Este metodo se entera mejor si lo analisamos por partes.

2.4.1. Método de Newton

El método de Newton, también conocido como Newton-Raphson [Hutton, 2004], como fue explicado en la sección anterior, es una técnica iterativa para encontrar soluciones de una función no lineal $\mathbf{F}(\mathbf{x})$, es decir, valores $\mathbf{x}_r$ tales que $\mathbf{F}(\mathbf{x}_k) = 0$.

2.4.1.1. Método de Newton en una Dimension

En una dimensión, se busca resolver $f(x) = 0$, donde $f : \mathbb{R} \rightarrow \mathbb{R}$ es una función diferenciable. El método de Newton se basa en aproximar $f(x)$ localmente mediante una línea tangente en un punto inicial x_0 . La iteración se realiza siguiendo la fórmula:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},$$

donde x_k es la aproximación actual, $f(x_k)$ es el valor de la función, y $f'(x_k)$ es su derivada en x_k .

Esta fórmula se deriva considerando la tangente a la curva de $f(x)$ en el punto x_k y buscando dónde esta tangente cruza el eje x .

2.4.1.2. Convergencia del Método de Newton

El método converge cuadráticamente cerca de la solución exacta x_r , bajo ciertas condiciones:

- $f(x)$ es diferenciable dos veces en un intervalo que contiene la raíz x_r .
- $f(x_r) = 0$, es decir, x_r es una raíz de la función.
- $f'(x_r) \neq 0$, lo que garantiza que la tangente en x_r no es horizontal.
- La aproximación inicial x_0 está lo suficientemente cerca de x_r .

Definimos el error en la k -ésima iteración como:

$$e_k = x_k - x_r.$$

Sustituyendo $x_k = x_r + e_k$ en la fórmula de Newton y usando un desarrollo de Taylor para $f(x_k)$ alrededor de x_r :

$$f(x_k) = f(x_r + e_k) = f(x_r) + f'(x_r)e_k + \frac{f''(x_r)}{2}e_k^2 + \mathcal{O}(e_k^3),$$

sabemos que $f(x_r) = 0$, por lo que:

$$f(x_k) \approx f'(x_r)e_k + \frac{f''(x_r)}{2}e_k^2,$$

entonces, reemplazando $f(x_k)$ en (2.4.1.1):

$$x_{k+1} = x_k - \frac{f'(x_r)e_k + \frac{f''(x_r)}{2}e_k^2}{f'(x_r)},$$

simplificando:

$$x_{k+1} = x_k + e_k + \frac{f''(x_r)}{2f'(x_r)}e_k^2,$$

esto lleva a:

$$x_{k+1} - x_k = e_k + \frac{f''(x_r)}{2f'(x_r)}e_k^2,$$

$$\frac{f(x_k)}{f'(x_k)} = e_k + \frac{f''(x_r)}{2f'(x_r)}e_k^2,$$

$$\frac{f(x_k)}{f'(x_k)} = x_k - x_r + \frac{f''(x_r)}{2f'(x_r)}e_k^2,$$

$$0 = x_k - \frac{f(x_k)}{f'(x_k)} - x_r + \frac{f''(x_r)}{2f'(x_r)}e_k^2,$$

$$0 = x_{k+1} - x_r + \frac{f''(x_r)}{2f'(x_r)}e_k^2,$$

$$0 = e_{k+1} + \frac{f''(x_r)}{2f'(x_r)}e_k^2,$$

$$|e_{k+1}| = \left| \frac{f''(x_r)}{2f'(x_r)}e_k^2 \right|,$$

por lo tanto:

$$|e_{k+1}| \propto |e_k^2|.$$

El término cuadrático $|e_k^2|$ indica que el error en la siguiente iteración es proporcional al cuadrado del error actual. Esto significa que, una vez que estamos lo suficientemente cerca de la raíz, el error disminuye cuadráticamente.

2.4.1.3. Método de Newton en Varias Dimensiones

En múltiples dimensiones, el problema se generaliza para resolver un sistema de ecuaciones no lineales $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, donde $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ es un vector de funciones diferenciables, y $\mathbf{x} \in \mathbb{R}^n$ es el vector de variables.

El método de Newton en este caso utiliza el Jacobiano de $\mathbf{F}$, denotado como $J_{\mathbf{F}}(\mathbf{x})$, que es la matriz de derivadas parciales:

$$J_{\mathbf{F}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}.$$

La iteración se define mediante la fórmula:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_{\mathbf{F}}(\mathbf{x}_k)^{-1}\mathbf{F}(\mathbf{x}_k),$$

donde $\mathbf{x}_k$ es la aproximación actual del vector solución, $\mathbf{F}(\mathbf{x}_k)$ es el vector de funciones evaluadas en $\mathbf{x}_k$ y $J_{\mathbf{F}}(\mathbf{x}_k)^{-1}$ es la inversa de la matriz Jacobiana evaluada en $\mathbf{x}_k$.

De forma homóloga al caso unidimensional usando la derivada, aquí en cada iteración, $J_{\mathbf{F}}(\mathbf{x}_k)$ aproxima localmente a $\mathbf{F}(\mathbf{x})$ mediante un modelo lineal. La fórmula actualiza $\mathbf{x}_k$ desplazándose hacia el punto donde este modelo lineal se anula.

En la práctica, en lugar de calcular explícitamente $J_F(\mathbf{x}_k)^{-1}$, se resuelve un sistema lineal obtenido de la siguiente forma:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_F(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k),$$

$$\mathbf{x}_{k+1} - \mathbf{x}_k = -J_F(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k),$$

$$\Delta \mathbf{x} = -J_F(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k),$$

$$J_F(\mathbf{x}_k) \Delta \mathbf{x} = -\mathbf{F}(\mathbf{x}_k).$$

Este enfoque evita el costo computacional de calcular la inversa del Jacobiano, procedimiento que es altamente costoso.

2.4.1.4. Convergencia del Método de Newton en Múltiples Dimensiones

El método de Newton en múltiples dimensiones también converge cuadráticamente cerca de la solución exacta $\mathbf{x}_r$, bajo ciertas condiciones similares al caso unidimensional. Estas condiciones son:

- $\mathbf{F}(\mathbf{x})$ es diferenciable dos veces en un entorno que contiene $\mathbf{x}_r$.
- $\mathbf{F}(\mathbf{x}_r) = \mathbf{0}$, es decir, $\mathbf{x}_r$ es la solución exacta.
- El Jacobiano $J_F(\mathbf{x}_r)$ es invertible, lo que garantiza la existencia de una solución única en el modelo lineal local.
- La aproximación inicial $\mathbf{x}_0$ está lo suficientemente cerca de $\mathbf{x}_r$.

Definimos el error en la k -ésima iteración como:

$$\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}_r.$$

Sustituyendo $\mathbf{x}_k = \mathbf{x}_r + \mathbf{e}_k$ en la iteración de Newton, tenemos:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_F(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k).$$

Al realizar un desarrollo de Taylor para $\mathbf{F}(\mathbf{x}_k)$ alrededor de $\mathbf{x}_r$, obtenemos:

$$\mathbf{F}(\mathbf{x}_k) = J_F(\mathbf{x}_r) \mathbf{e}_k + \frac{1}{2} H(\mathbf{e}_k, \mathbf{e}_k) + \mathcal{O}(\|\mathbf{e}_k\|^3),$$

donde $H(\mathbf{e}_k, \mathbf{e}_k)$ representa los términos cuadráticos asociados a la matriz Hessiana de $\mathbf{F}$ [Callahan, 2010].

Sustituyendo en la fórmula de Newton y considerando que $J_F(\mathbf{x}_k) \approx J_F(\mathbf{x}_r)$ para $\mathbf{x}_k$ cercano a $\mathbf{x}_r$, se obtiene:

$$\mathbf{x}_{k+1} = \mathbf{x}_r + \mathbf{e}_k - J_F(\mathbf{x}_r)^{-1} \left(J_F(\mathbf{x}_r) \mathbf{e}_k + \frac{1}{2} H(\mathbf{e}_k, \mathbf{e}_k) \right),$$

simplificando:

$$\begin{aligned} \mathbf{x}_{k+1} - \mathbf{x}_r &= -\frac{1}{2} J_F(\mathbf{x}_r)^{-1} H(\mathbf{e}_k, \mathbf{e}_k), \\ \mathbf{e}_{k+1} &= -\frac{1}{2} J_F(\mathbf{x}_r)^{-1} H(\mathbf{e}_k, \mathbf{e}_k), \end{aligned}$$

por lo tanto, el error en la $(k + 1)$ -ésima iteración es proporcional al cuadrado del error en la k -ésima iteración:

$$\|\mathbf{e}_{k+1}\| \propto \|\mathbf{e}_k\|^2,$$

El término $\|\mathbf{e}_k\|^2$ indica que el error disminuye cuadráticamente cerca de la solución exacta $\mathbf{x}_r$, duplicando aproximadamente la cantidad de dígitos correctos en cada iteración. Sin embargo, como en el caso unidimensional, la convergencia depende de una buena aproximación inicial y de la invertibilidad del Jacobiano en $\mathbf{x}_r$.

2.4.2. Espacios de Krylov y Método GMRES

Los métodos de Krylov constituyen una familia de técnicas iterativas diseñadas para resolver sistemas de ecuaciones lineales de la forma $A\mathbf{x} = \mathbf{b}$, donde $A \in \mathbb{R}^{n \times n}$ es una matriz cuadrada, $\mathbf{x} \in \mathbb{R}^n$ es el vector solución, y $\mathbf{b} \in \mathbb{R}^n$ representa el vector de términos independientes. Dentro de esta familia, el método GMRES (Generalized Minimal Residual, o Residuo Mínimo Generalizado) se destaca por aprovechar los denominados espacios de Krylov [Sogabe, 2023].

2.4.2.1. Espacios de Krylov

El espacio de Krylov de dimensión k , generado por una matriz $A \in \mathbb{R}^{n \times n}$ y un vector $\mathbf{r}_0 \in \mathbb{R}^n$, se define como:

$$\mathcal{K}_k(A, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \dots, A^{k-1}\mathbf{r}_0\},$$

donde $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$ es el residuo inicial correspondiente a una aproximación inicial $\mathbf{x}_0 \in \mathbb{R}^n$.

El propósito de los métodos basados en espacios de Krylov es construir una aproximación $\mathbf{x}_k \in \mathbb{R}^n$ del vector solución $\mathbf{x}$, cumpliendo:

$$\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}_k(A, \mathbf{r}_0).$$

Esto implica que $\mathbf{x}_k$ está formado por la suma de la aproximación inicial $\mathbf{x}_0$ y una combinación lineal de los vectores en el espacio de Krylov $\mathcal{K}_k(A, \mathbf{r}_0)$. En cada iteración k , se incrementa la dimensión del espacio de Krylov, lo que mejora la calidad de la aproximación obtenida.

2.4.2.2. Método GMRES

El método GMRES busca minimizar la norma del residuo $\|\mathbf{r}_k\| = \|\mathbf{b} - A\mathbf{x}_k\|$ en cada iteración, dentro del subespacio $\mathcal{K}_k(A, \mathbf{r}_0)$. Esto se formula como el problema de minimización, conocido como mínimos cuadrados [Sogabe, 2023]:

$$\mathbf{x}_k = \arg \min_{\mathbf{x} \in \mathbf{x}_0 + \mathcal{K}_k(A, \mathbf{r}_0)} \|\mathbf{b} - A\mathbf{x}\|.$$

El método se compone de la siguiente manera:

1. Construcción de la base ortonormal:

Se utiliza el algoritmo de la iteración de Arnoldi para generar una base ortonormal $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k\}$ del espacio de Krylov $\mathcal{K}_k(A, \mathbf{r}_0)$, satisfaciendo:

$$AV_k = V_{k+1}H_k,$$

donde $V_k \in \mathbb{R}^{n \times k}$ contiene las bases ortonormales, y $H_k \in \mathbb{R}^{(k+1) \times k}$ es una matriz Hessenberg superior.

La iteración de Arnoldi utiliza el proceso de Gram-Schmidt modificado para generar una secuencia de vectores ortonormales $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \dots$, conocidos como *vectores de Arnoldi*, de modo que para cada k , los vectores $\mathbf{v}_1, \dots, \mathbf{v}_k$ forman una base del subespacio de Krylov $\mathcal{K}_k(A, \mathbf{r}_0)$.

El algoritmo se describe de la siguiente forma:

1. Comenzar con un vector inicial $\mathbf{v}_1$ tal que $\|\mathbf{v}_1\| = 1$.
2. Para $k > 1$:
 - a) Definir $\mathbf{v}_k := A\mathbf{v}_{k-1}$.
 - b) Ortogonalización: Para $j < k$:
 - 1) Calcular $h_{j,k-1} = \mathbf{v}_j^\top \mathbf{v}_k$.
 - 2) Actualizar $\mathbf{v}_k := \mathbf{v}_k - h_{j,k-1} \mathbf{v}_j$.
 - c) Calcular $h_{k,k-1} = \|\mathbf{v}_k\|$.
 - d) Normalización: Normalizar el vector $\mathbf{v}_k$ dividiéndolo por su norma: $\mathbf{v}_k := \frac{\mathbf{v}_k}{h_{k,k-1}}$.

La iteración asegura la ortogonalidad de los vectores generados proyectando $\mathbf{v}_k$ sobre los vectores previos $\mathbf{v}_1, \dots, \mathbf{v}_{k-1}$.

Al finalizar la iteración, la relación entre los vectores generados y la matriz A se puede expresar como:

$$AV_k = V_{k+1}H_k$$

donde:

- $V_k \in \mathbb{R}^{n \times k}$ es la matriz que contiene los vectores ortonormales $\mathbf{v}_1, \dots, \mathbf{v}_k$.
- $H_k \in \mathbb{R}^{(k+1) \times k}$ es la matriz Hessenberg superior generada por la iteración de Arnoldi; nótese que los valores de h no definidos en la iteración toman el valor por defecto de 0.

2. Resolución del sistema reducido:

La aproximación $\mathbf{x}_k$ puede escribirse como:

$$\mathbf{x}_k = \mathbf{x}_0 + V_k \mathbf{y}_k$$

donde $V_k \in \mathbb{R}^{n \times k}$ es la matriz que contiene los vectores ortonormales $\mathbf{v}_1, \dots, \mathbf{v}_k$ como columnas y $\mathbf{y}_k \in \mathbb{R}^k$ es el vector que minimiza la norma del residuo $\mathbf{r}_k$.

La resolución de $\mathbf{y}_k$ se reduce a un problema de mínimos cuadrados que minimiza la norma:

$$\mathbf{y}_k = \arg \min_{\mathbf{y}_k \in \mathbb{R}^k} \|\beta \mathbf{e}_1 - H_k \mathbf{y}_k\|$$

donde $\beta = \|\mathbf{r}_0\|$ es la norma del residuo inicial, $\mathbf{e}_1 = (1, 0, \dots, 0)^\top$ es el primer vector de la base canónica de $\mathbb{R}^{k+1}$, y $H_k \in \mathbb{R}^{(k+1) \times k}$ es la matriz Hessenberg superior obtenida del proceso de Arnoldi.

3. Actualización del residuo:

Una vez obtenida la solución aproximada $\mathbf{x}_k$, el residuo se actualiza como:

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k,$$

donde $\mathbf{b}$ es el vector del lado derecho y $\mathbf{x}_k$ es la solución aproximada en la iteración k . La norma de este residuo, $\|\mathbf{r}_k\|$, proporciona una medida del error de la solución y se utiliza como criterio de convergencia del algoritmo.

Este valor debe disminuir en cada iteración hasta que se cumpla el criterio de convergencia $\|\mathbf{r}_k\| < \epsilon$, donde ϵ es un umbral predefinido. Si el residuo no es suficientemente pequeño, el proceso se repite con una nueva iteración.

2.4.2.3. GMRES Reiniciado

El método GMRES es especialmente ventajoso para matrices no simétricas y problemas donde los métodos directos son prohibitivamente costosos. Sin embargo, su principal limitación es el crecimiento del costo computacional y del almacenamiento debido a la necesidad de construir y almacenar la base de Krylov completa.

Para mitigar este problema, se utilizan variantes como GMRES reiniciado (Restarted GMRES) [Sogabe, 2023], el cual busca equilibrar la eficiencia computacional y la memoria. La idea fundamental detrás de GMRES reiniciado es limitar el número de iteraciones de la expansión de la base de Krylov a un valor predeterminado m , en lugar de continuar hasta que se alcance una solución exacta. Cada vez que se completa un conjunto de m iteraciones, el proceso se reinicia utilizando el último residuo como el nuevo punto de partida.

Matemáticamente, dado un sistema de ecuaciones lineales de la forma:

$$A\mathbf{x} = \mathbf{b}$$

y un residuo inicial $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$. El método GMRES genera una sucesión de vectores $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$ que forman una base de Krylov de dimensión m :

$$K_m(A, \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \dots, A^{m-1}\mathbf{r}_0\}$$

La solución aproximada en cada iteración es obtenida resolviendo un sistema lineal con la matriz H_m , que es la proyección de A sobre el subespacio generado por los vectores de Krylov. El proceso se reinicia después de cada conjunto de m iteraciones con un nuevo residuo.

El algoritmo GMRES reiniciado puede resumirse de la siguiente forma:

1. Se inicializa con un residuo $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$.
2. Se genera la base de Krylov de tamaño m .
3. Se resuelve el sistema lineal reducido para obtener la aproximación de la solución $\mathbf{x}_m$.
4. Si el número de iteraciones no ha alcanzado el límite máximo o la convergencia esperada, el proceso se reinicia utilizando el residuo actual.

Este procedimiento permite controlar el crecimiento del almacenamiento y el costo computacional, aunque puede perder precisión al reiniciar la base de Krylov después de m iteraciones.

2.4.2.4. Método LGMRES: Aceleración del GMRES Reiniciado

El algoritmo LGMRES (*Loose GMRES*) [Baker *et al.*, 2005] es una técnica diseñada para acelerar la convergencia del método GMRES reiniciado, abordando problemas comunes como la pérdida de ortogonalidad y el estancamiento en la convergencia. Este método combina ideas de preconditionamiento polinómico y subespacios aumentados, lo que permite mantener información valiosa entre ciclos de reinicio sin requerir cambios significativos en la implementación estándar del GMRES reiniciado.

En el GMRES reiniciado, los subespacios de aproximación generados en ciclos previos son descartados al inicio de cada reinicio, lo que puede ralentizar la convergencia. Para mitigar este problema, LGMRES introduce un mecanismo que agrega aproximaciones de errores pasados al subespacio de Krylov generado en el ciclo actual. En cada ciclo i , la solución aproximada se encuentra en el espacio combinado:

$$\mathbf{x}_{i+1} \in \mathbf{x}_i + \mathcal{K}_m(A, \mathbf{r}_i) + \text{span}\{\mathbf{z}_{i-k+1}, \dots, \mathbf{z}_i\}$$

donde $\mathbf{z}_j = \mathbf{x}_j - \mathbf{x}_{j-1}$ representa una aproximación al error después del ciclo j .

2.4.2.5. Implementación del Algoritmo

El procedimiento de LGMRES se implementa siguiendo los pasos:

1. Generar el subespacio de Krylov $\mathcal{K}_m(A, \mathbf{r}_i)$ mediante la iteración de Arnoldi para construir una base ortonormal V_m .
2. Aumentar el espacio de aproximación con los k vectores de error más recientes $\mathbf{z}_{i-k+1}, \dots, \mathbf{z}_i$.
3. Resolver el sistema reducido:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + V_{m+k} \mathbf{y}_{m+k},$$

donde V_{m+k} es la base del espacio combinado, y $\mathbf{y}_{m+k}$ minimiza la norma del residuo:

$$\mathbf{y}_{m+k} = \arg \min_{\mathbf{y}_{m+k}} \|\beta \mathbf{e}_1 - H_{m+k} \mathbf{y}_{m+k}\|$$

con $\beta = \|\mathbf{r}_i\|$ y H_{m+k} la matriz Hessenberg extendida.

4. Actualizar la solución aproximada y calcular el nuevo residuo:

$$\mathbf{r}_{i+1} = \mathbf{b} - A\mathbf{x}_{i+1}$$

Si $\|\mathbf{r}_{i+1}\| < \epsilon$, el algoritmo concluye; de lo contrario, se inicia un nuevo ciclo.

2.4.3. Unión entre el Método de Newton y GMRES

Habiendo explorado ambos métodos de manera independiente, podemos ahora analizar cómo combinarlos para dar lugar al mencionado método de Newton-Krylov.

La expresión para la búsqueda de ceros mediante el método de Newton, presentada en (2.4.1.3), es la siguiente:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_{\mathbf{F}}(\mathbf{x}_k)^{-1} \mathbf{F}(\mathbf{x}_k),$$

donde el cálculo de $J_{\mathbf{F}}(\mathbf{x}_k)^{-1}$ resulta ser un inconveniente debido al costo computacional. Sin embargo, como se mostró en (2.4.1.3), esta expresión es equivalente a la siguiente formulación:

$$J_{\mathbf{F}}(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = -\mathbf{F}(\mathbf{x}_k),$$

la cual puede reescribirse como el sistema lineal:

$$A\mathbf{x} = \mathbf{b}.$$

Este sistema puede ser resuelto, o al menos aproximado, utilizando GMRES o alguna de sus variantes. Esto significa que, en lugar de calcular explícitamente la matriz jacobiana inversa, podemos permitir que el método GMRES trabaje directamente sobre un sistema equivalente.

GMRES nos proporciona el valor de $\mathbf{x}$, que corresponde a $\mathbf{x}_{k+1} - \mathbf{x}_k$. Dado que $\mathbf{x}_k$ ya es conocido (es el valor de la iteración anterior), podemos calcular $\mathbf{x}_{k+1}$, la siguiente aproximación en el método de Newton, simplemente sumando $\mathbf{x}_k$.

Este procedimiento puede optimizarse aún más. Si bien GMRES requiere tres entradas principales (la matriz A , el vector solución $\mathbf{x}$, y el vector independiente $\mathbf{b}$), la matriz A no necesita ser almacenada explícitamente, ya que GMRES únicamente utiliza el producto matriz-vector $A\mathbf{r}_i$ para construir el espacio de Krylov. En este caso, sabemos que A corresponde al jacobiano, y este producto puede calcularse mediante métodos como diferencias finitas o el método del paso complejo, explorados anteriormente. Esto elimina la necesidad de construir explícitamente la matriz jacobiana, pues el producto jacobiano-vector puede obtenerse directamente.

La combinación del método de Newton con GMRES permite resolver sistemas no lineales de manera eficiente, evitando el cálculo directo del jacobiano o su inversa, lo que reduce significativamente los costos computacionales. El método Newton-Krylov aprovecha la robustez del método de Newton para aproximar raíces y la flexibilidad de GMRES para resolver sistemas lineales de gran escala mediante iteraciones. Además, la posibilidad de calcular el producto jacobiano-vector sin formar explícitamente la matriz jacobiana lo convierte en una técnica versátil y de amplio alcance en la resolución de problemas numéricos complejos.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

3.1. Motivación y Objetivos del Trabajo

El objetivo principal de este trabajo es analizar el método de paso complejo como una herramienta viable para la resolución de sistemas de ecuaciones y fomentar su uso en el ámbito numérico. Para ello, se evaluará su desempeño en comparación con métodos estándar, con el propósito de evidenciar su eficacia y versatilidad en diversas aplicaciones. Actualmente, el método de paso complejo está subestimado y tiene una adopción limitada; este análisis busca demostrar que posee el potencial para competir con las técnicas más avanzadas del estado del arte.

Con esta motivación, se propone desarrollar un conjunto de librerías que implementen estos métodos y sean compatibles con los entornos computacionales contemporáneos, proporcionando soluciones eficientes y accesibles para los usuarios.

Los métodos desarrollados serán evaluados en distintos niveles de complejidad, que incluyen:

- Cálculo de derivadas o del producto Jacobiano-Vector.
- Desempeño en métodos de Krylov, evaluando la calidad del espacio generado.
- Iteración completa del método de Newton-Krylov aplicado a problemas reales.

Las evaluaciones se llevarán a cabo considerando métricas clave como el tiempo de ejecución y la precisión de los resultados obtenidos. Otros atributos, como el uso de memoria, quedan fuera del análisis, ya que se espera que no difieran significativamente en comparación con las técnicas actuales del estado del arte.

3.2. Entorno Computacional

3.2.1. Lenguajes de Programación

El desarrollo propuesto se centrará en maximizar la usabilidad de los algoritmos diseñados. Para ello, se han seleccionado los siguientes lenguajes de programación como base para el análisis:

Python, un lenguaje de propósito general ampliamente utilizado en aplicaciones de análisis numérico, gracias a su extenso ecosistema de librerías especializadas. Entre estas, destaca *SciPy*, que servirá como referencia comparativa para evaluar el desempeño de los métodos desarrollados. Además, se busca que los algoritmos propuestos sean compatibles y trabajen en sinergia con las herramientas de esta librería, permitiendo su integración y facilitando su adopción por parte de los usuarios.

Por otro lado, se ha seleccionado el lenguaje emergente *Julia*, reconocido por su excepcional rendimiento en Computación Científica. *Julia* se perfila como un posible sustituto de *Python* en este ámbito, por lo que resulta pertinente evaluar cómo se desempeñan los métodos desarrollados en este entorno, destacando sus ventajas y limitaciones frente a su principal competidor.

3.2.2. Infraestructura

Para este trabajo, se contó con acceso a la red de computadores del NLHPC (Laboratorio Nacional de Computación de Alto Rendimiento), lo que permitió abordar problemas de gran envergadura y emplear algoritmos complejos. Esto aportó una valiosa sustentabilidad a los resultados obtenidos.

La infraestructura del NLHPC cuenta con varias particiones de computadoras. En este caso, se utilizó exclusivamente la partición denominada *main*, cuyos nodos presentan las siguientes características:

Tabla 1: Características de un nodo en la partición *main*

Característica	Especificación
Modelo del servidor	Lenovo ThinkSystem SR645 V3
Procesador	AMD EPYC 9754
Cantidad de CPUs	256
Memoria RAM	727 GB

En todos los experimentos, se mantuvo constante la cantidad de procesos y CPUs disponibles por proceso, configurándose en uno cada uno. Esto permitió obtener una medición más consistente de los resultados entre los distintos experimentos. La cantidad de memoria RAM varió según las necesidades de cada experimento, asegurando siempre un margen adicional respecto a los requerimientos mínimos.

3.3. Implementación en Python

3.3.1. Compatibilidad con SciPy

Garantizar la compatibilidad de los códigos y librerías desarrollados con SciPy es fundamental debido a su amplia aceptación y uso en diversos campos científicos y de ingeniería. SciPy ofrece un conjunto robusto y optimizado de algoritmos para tareas numéricas avanzadas, como optimización, resolución de ecuaciones diferenciales, manejo de matrices dispersas y cálculo de valores propios.

Aunque este análisis no hace un uso exhaustivo de todas las herramientas de SciPy, asegurar la interoperabilidad con esta biblioteca facilita la integración de nuestros desarrollos en futuros proyectos y contribuye a la colaboración en entornos basados en Python. Además, el acceso a las soluciones estándar de alto rendimiento de SciPy mejora la escalabilidad, mantenibilidad y reusabilidad del código.

Las versiones de software utilizadas en el desarrollo fueron las siguientes:

1. Python: 3.9.6
2. SciPy: 1.8.0
3. Numpy: 1.22.0

3.3.2. Modificación del algoritmo de Newton-Krylov

En SciPy, existen dos herramientas especialmente útiles para trabajar con sistemas. La primera es la interfaz *Jacobian*, que permite desarrollar clases para representar el Jacobiano, implementando métodos de interés como *matvec(self, v)*, que corresponde al producto Jacobiano-vector, donde se implementará el método de paso complejo. Además, incluye la función *_nonlin_wrapper*, que envuelve un objeto de clase Jacobiano y retorna una función para la búsqueda de ceros.

Con estas herramientas, nuestra estrategia consistirá en tomar como base la clase Jacobiano por defecto que ofrece SciPy, copiarla y modificarla. En lugar de utilizar diferencias finitas, se reemplazará por el método de paso complejo, adaptando también los métodos secundarios. Posteriormente, envolveremos esta nueva clase en el *wrapper*, obteniendo una función del método de Newton-Krylov, que será funcionalmente idéntica a la proporcionada en la documentación de SciPy https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton_krylov.html.

3.3.3. Repositorio

El repositorio que contiene el código generado puede ser encontrado en el siguiente link Github `python-complex_step_approximation-ndimensional-nderivative` (https://github.com/Ultrinik/python-complex_step_approximation-ndimensional-nderivative).

La implementación específica del producto jacobiano-vector presente en la librería también puede encontrarse en el anexo en las secciones 5.2 y 5.3.

3.4. Implementación en Julia

3.4.1. Julia como Lenguaje Emergente

A diferencia de otras herramientas consolidadas en el campo del análisis numérico, *Julia* se encuentra en una etapa emergente de desarrollo. Como resultado, aún no existe una biblioteca que domine de manera hegemónica este ámbito en el lenguaje *Julia*, tal como ocurre con *SciPy* en *Python*.

En este contexto, se ha optado por utilizar *Bifurcation Analysis*, ya que, dentro del repertorio de librerías disponibles, esta se destaca como una de las más recomendadas según métricas de Github para realizar estudios en análisis numérico y sistemas dinámicos. Esta elección se justifica por su amplia gama de métodos y herramientas que se alinean perfectamente con el enfoque de nuestro análisis.

Bifurcation Analysis no solo ofrece algoritmos avanzados para el análisis de bifurcaciones, como podría sugerir su nombre, sino que también es capaz de abordar problemas de diversas envergaduras. Esta flexibilidad y capacidad de adaptación son esenciales para nuestro trabajo, ya que buscamos que los algoritmos desarrollados puedan integrarse a problemas numéricos complejos que exigen enfoques robustos y eficientes.

Las versiones de software utilizadas en el desarrollo fueron las siguientes:

1. Julia: 1.9.0
2. BifurcationKit: 0.4.5

3.4.2. Implementación del Algoritmo de Newton-Krylov

Desafortunadamente, *BifurcationKit* no implementa el método de Newton-Krylov de forma nativa. Sin embargo, proporciona las herramientas necesarias para ensamblarlo. Para ello,

se requieren tres componentes principales. En primer lugar, necesitamos una función que calcule el producto Jacobiano-Vector. Para ello, utilizaremos tanto el método de diferencias finitas como el de paso complejo. Esta función se integrará en una clase denominada *BifurcationProblem*, que, entre otras cosas, incluye la función de interés.

En tercer lugar, se requiere la clase *NewtonPar*, que se encarga de definir el método de resolución lineal que se utilizará, en este caso, GMRES.

Con estos tres componentes principales, es posible invocar la función *newton*, la cual se comportará como el método de Newton-Krylov.

3.4.3. Repositorio

El repositorio que contiene el código generado puede ser encontrado en el siguiente link Github `julia-complex_step_approximation-ndimensional-nderivative` (https://github.com/Ultrinik/julia-complex_step_approximation-ndimensional-nderivative).

La implementación específica del producto jacobiano-vector presente en la librería también puede encontrarse en el anexo en las secciones 5.4 y 5.5.

3.5. Precisión en la Representación Numérica

La precisión numérica es un aspecto fundamental en nuestro análisis, por lo que es crucial tratar de manera coherente la representación numérica con la que trabajamos. En el caso de los números de punto flotante, tanto en *Python* como en *Julia*, el valor por defecto para la representación de estos números es de 64 bits.

Los números de punto flotante de 64 bits, también conocidos como *double precision*, tienen las siguientes propiedades:

- **Precisión de 15 a 17 dígitos decimales:** La representación de 64 bits permite una precisión de aproximadamente 15 a 17 dígitos decimales significativos.
- **Rango de valores:** El rango de valores representables va desde aproximadamente 10^{-308} hasta 10^{308} , lo que permite trabajar con un amplio rango de magnitudes.
- **Epsilon máquina:** Posee un epsilon máquina de $2^{-52} \approx 2,22 \times 10^{-16}$.

Sin embargo, este tipo de dato está diseñado únicamente para representar números reales, lo cual es insuficiente para nuestro caso, ya que el método de paso complejo requiere el uso de números complejos. Afortunadamente, tanto *Python* como *Julia* implementan de manera

nativa su propio tipo de número complejo, que consta de dos números flotantes de 64 bits. Esto resulta en una representación de 128 bits, donde cada componente posee las mismas propiedades descritas anteriormente.

El análisis se centrará en comparar el método de diferencias finitas con representación numérica de 64 bits frente al método de paso complejo con representación de 128 bits. Este contraste resulta especialmente relevante, ya que el método de paso complejo utiliza el doble de bits, lo que podría influir significativamente en el desempeño, convirtiéndose en un aspecto clave del estudio.

Existen otras opciones, como los flotantes de 32 bits, conocidos como *single precision*, y por ende, los números complejos de 64 bits (32 bits por componente). Estos tienen las siguientes propiedades:

- **Precisión de 6 a 9 dígitos decimales:** La representación de 32 bits permite una precisión de aproximadamente 6 a 9 dígitos decimales significativos.
- **Rango de valores:** El rango de valores representables va desde aproximadamente 10^{-38} hasta 10^{38} , lo que permite trabajar con un rango de magnitudes más reducido que el de 64 bits.
- **Epsilon máquina:** El epsilon máquina para esta representación es de aproximadamente $2^{-23} \approx 1,19 \times 10^{-7}$.

Finalmente, es pertinente reconocer la existencia de otras representaciones numéricas, como los flotantes de 16 o 128 bits. Sin embargo, estos tipos de datos son empleados principalmente en aplicaciones muy específicas y rara vez sustituyen a sus contrapartes de 32 o 64 bits en usos generales. Debido a su limitada aplicabilidad, muchas librerías no admiten estos tipos de datos. Por lo tanto, estos casos serán omitidos en el análisis realizado.

3.6. Ajuste dinámico del paso

El método de diferencias finitas es propenso a errores de cancelación, por lo que muchas librerías implementan un regularizador que ajusta dinámicamente el tamaño del paso h durante la ejecución del algoritmo. Esta técnica, conocida como *adaptive step size selection*, ayuda a mitigar los problemas numéricos.

La implementación en SciPy es [SciPy Developers, 2024]:

```
1 def _update_diff_step(self):
2     mx = abs(self.x0).max()
3     mf = abs(self.f0).max()
4     self.omega = self.rdiff * max(1, mx) / max(1, mf)
```

en su forma matemática:

$$h_{k+1} = h_0 \frac{\max(1, \max(\text{abs}(\mathbf{x}_k)))}{\max(1, \max(\text{abs}(\mathbf{F}(\mathbf{x}_k))))},$$

Esta estrategia permite mantener un equilibrio entre precisión y estabilidad al ajustar el tamaño del paso según la magnitud del problema, reduciendo la probabilidad de errores de cancelación.

En contraste, el método de paso complejo no requiere un adaptador en teoría, ya que no presenta los mismos problemas numéricos. Sin embargo, para evaluar su robustez en escenarios reales, se realizará un análisis comparativo entre ambos métodos para evaluar el impacto del adaptador en el desempeño del algoritmo.

3.7. Experimentos Anexos

El objetivo principal de este trabajo es analizar cómo el método de paso complejo se compara con las técnicas actuales del estado del arte en términos de eficiencia y precisión. Un objetivo secundario, pero no menos importante, es fomentar un mayor uso de este método en el campo del análisis numérico. En este contexto, consideramos relevante extender el análisis a escenarios adicionales que, aunque no están directamente relacionados con la implementación del método de Newton-Krylov, pueden proporcionar valiosos conocimientos sobre las capacidades y limitaciones del método de paso complejo.

Uno de los aspectos que exploraremos es el comportamiento del método al calcular derivadas de orden superior. Si bien estas derivadas no son directamente utilizadas en el método de Newton-Krylov, representan un caso interesante para evaluar la robustez del método de paso complejo en situaciones donde la precisión numérica se vuelve crítica. Estos experimentos nos permitirán medir el impacto de los errores de redondeo y la estabilidad del método frente a derivadas de mayor orden, aspectos clave en aplicaciones avanzadas del análisis numérico.

Además, examinaremos cómo el uso de un mayor número de puntos en la construcción de aproximaciones afecta la precisión y la estabilidad del método. Esto incluirá una evaluación detallada de cómo el método de paso multi-complejo, una extensión natural del método de paso complejo, se desempeña en comparación con otros métodos estándar. Particularmente, analizaremos su resistencia frente a errores de cancelación y cómo esta propiedad influye en su eficacia en problemas donde las aproximaciones tradicionales presentan limitaciones.

Los resultados de estos experimentos anexos no solo enriquecerán el análisis principal del trabajo, sino que también servirán como base para demostrar la versatilidad del método de paso complejo en una variedad de contextos. Este enfoque busca enfatizar que, lejos de ser una técnica limitada a casos específicos, el método tiene un amplio potencial de aplicación que merece ser explorado y promovido en futuras investigaciones.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

4.1. Comparación de Diferentes Niveles de Precisión Flotante

4.1.1. Motivación

Se analiza el impacto de utilizar diferentes niveles de precisión en los cálculos numéricos, evaluando tanto los tiempos de ejecución como los errores relacionados a los resultados, con el objetivo de identificar la máxima precisión alcanzable en cada método. En particular, se consideran dos enfoques: el método de diferencias finitas, que utiliza precisión de 64 bits, y el método de paso complejo, con precisiones de 128 y 64 bits. Todos los gráficos de resultados en este capítulo son de elaboración propia.

Un aspecto clave a considerar es que el método de diferencias finitas requiere un mínimo de dos evaluaciones de la función para obtener la derivada, mientras que el método de paso complejo solo necesita una evaluación, lo que podría equilibrar los tiempos de ejecución. Este último, sin embargo, utiliza números complejos, que consisten en dos números flotantes como componentes, lo cual podría afectar la eficiencia en términos de rendimiento. Este análisis busca determinar cómo la precisión de los tipos de datos afecta tanto la precisión numérica como el tiempo de cómputo.

4.1.2. Rendimiento del Dato Numérico

Primero, se busca evaluar el desempeño de distintos tipos de datos numéricos disponibles en los lenguajes Python y Julia, analizando experimentalmente las ideas discutidas previamente. Esta evaluación inicial no se centrará en el cálculo de derivadas, sino en el comportamiento de las operaciones elementales con estos tipos de datos, examinando la relevancia de las diferencias de rendimiento entre ellos.

Las operaciones se llevarán a cabo utilizando vectores aleatorios de longitud 1000, realizándose de manera elemento a elemento.

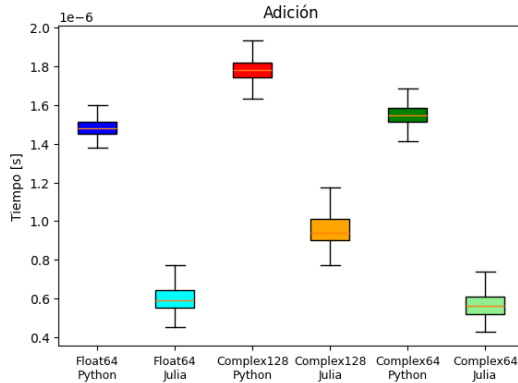


Figura 11: Tiempos de cómputo de sumas de vectores en distintos lenguajes.

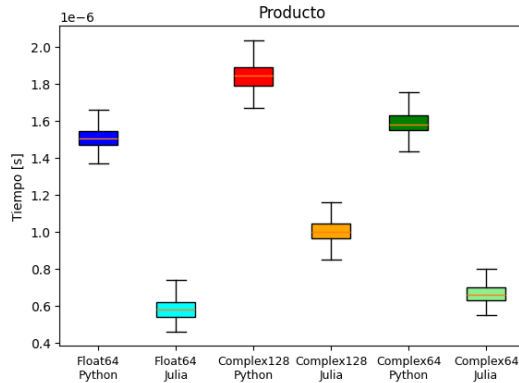


Figura 12: Tiempos de cómputo de multiplicar dos vectores en distintos lenguajes.

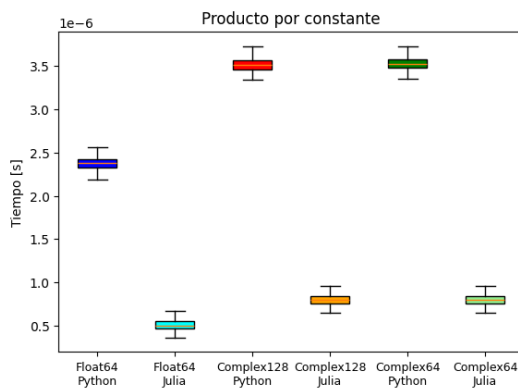


Figura 13: Tiempos de cómputo de multiplicar un vector por la constante 2 en distintos lenguajes.

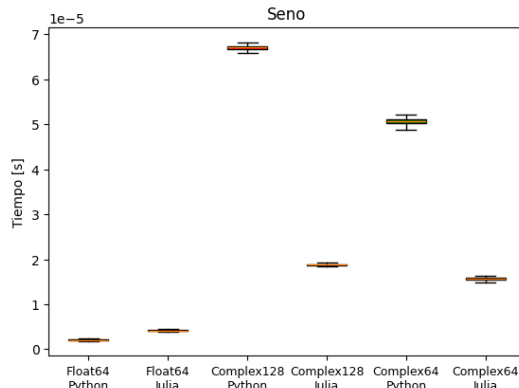


Figura 14: Tiempos de cómputo de aplicar la función $\sin()$ a vectores en distintos lenguajes.

4.1.2.1. Conclusiones

Tras los experimentos realizados, hay varios aspectos importantes que destacar. En primer lugar, *Julia* muestra, en general, un mejor desempeño que *Python*, siendo consistentemente más rápido por un margen variable. Sin embargo, existe una excepción al calcular el seno de dos números flotantes de 64 bits, donde *Julia* resulta marginalmente más lento.

Otro aspecto importante a destacar es que el tiempo de ejecución no varía de manera lineal con la cantidad de bits. Esto se evidencia en los números complejos de 64 bits, que no son igual de rápidos como los flotantes de la misma precisión, aunque en la mayoría de los casos presentan tiempos de ejecución similares. Por otro lado, los números complejos de 128 bits no duplican el tiempo de ejecución de los de 64 bits; en cambio, consistentemente muestran un tiempo inferior al esperado en esa proporción.

Además, tanto en *Python* como en *Julia*, al multiplicar un número complejo por un flotante de 64 bits, el tiempo de la operación parece ser independiente de la cantidad de bits, ya que ambos tipos de datos toman el mismo tiempo de ejecución. Esto podría deberse a un *overhead* innecesario entre tipos de datos, lo que haría que el sacrificio de bits sea irrelevante. Cabe considerar que este efecto podría reproducirse en otras operaciones más allá de la multiplicación, perjudicando severamente la opción de usar números complejos de 64 bits para ganar velocidad.

4.2. Problemas Benchmark

El conjunto de funciones utilizadas en nuestros experimentos es diverso en naturaleza y estructura, y fue elegido para abarcar patrones, densidades o distribuciones distintas. A continuación, se describen las características de cada problema evaluado:

- (i) **Problema Normal:** Una función lineal $F(x) = \mathbf{b} - Ax$, donde los coeficientes de A y $\mathbf{b}$ están distribuidos normalmente como $N(0, 1)$. La matriz A tiene tamaño $n \times n$ y los vectores x y $\mathbf{b}$ son de tamaño n . Este problema evalúa el desempeño en un contexto de distribuciones estadísticas gaussianas.
- (ii) **Problema Uniform:** Una función lineal $F(x) = \mathbf{b} - Ax$, donde los coeficientes de A y $\mathbf{b}$ están uniformemente distribuidos en $[-1, 1]$. Similar al problema normal, La matriz A tiene tamaño $n \times n$ y los vectores x y $\mathbf{b}$ son de tamaño n . Este caso evalúa el impacto de distribuciones estadísticas uniformes.
- (iii) **Problema Sparse I:** Una función cuadrática definida con un tamaño n . Este problema evalúa matrices dispersas con estructura regular, esta es una versión modificada de la función utilizada por Toint Toint [Toint, 1987]. El algoritmo de la función se encuentra descrito en: 5.6.
- (iv) **Problema Sparse II:** Una función diagonal multiplicada por una matriz cuasi-ortogonal, con tamaño n . Este problema pone a prueba el manejo de matrices dispersas combinadas con transformaciones ortogonales, esta es una versión modificada de la función utilizada en Cruz y Raydan [La Cruz y Raydan, 2003]. El algoritmo de la función se encuentra descrito en: 5.7.
- (v) **Problema Dense I:** Una función trigonométrica modificada definida como:

$$f_k(x) = n + k(1 - \cos(x_{i+1})) - \sin(x_{i+1}) - \sum_{j=1}^n \cos^{p+1}(x_j),$$

donde $k = np + i$ y $p = 1$. Este problema evalúa funciones densas altamente no lineales, esta es una versión modificada de la función utilizada por Moré et al [Moré, 1978]. El algoritmo de la función se encuentra descrito en: 5.8.

(vi) **Problema Densé II:** Una función logarítmica definida por:

$$f_k(x) = x_{i+1}^{p+1} \log \left(\sum_{j=1}^n x_j^2 + 1 \right) + x_{i+1},$$

donde $k = np + i$ y $p = 1$, esta es una versión modificada de la función utilizada por Sanhueza en [Sanhueza Roman,]. El algoritmo de la función se encuentra descrito en: 5.9.

Las dimensiones de los problemas se definieron según dos categorías:

- **Tamaño pequeño:** $n = 50$.
- **Tamaño grande:** $n = 25000$.

Como el objetivo principal de estos problemas es, entre otras cosas, ser utilizados en la búsqueda de ceros, resulta fundamental garantizar que cada función $\mathbf{F}$ tenga al menos una raíz. Para ello, cada función $\mathbf{F}$ se redefine como un componente de la siguiente forma:

$$\mathbf{G}(\mathbf{x}) = \mathbf{F}(\mathbf{x}) - \mathbf{F}(\mathbf{x}_r),$$

donde $\mathbf{x}_r$ es un vector arbitrario asignado como raíz. Para los problemas Normal, Sparse I y Sparse II, $\mathbf{x}_r$ es un vector distribuido normalmente según $N(0, 1)$; en los problemas Uniform, Densé I y Densé II, $\mathbf{x}_r$ es un vector uniformemente distribuido en el intervalo $[-1, 1]$.

Además, cada problema requiere un vector inicial $\mathbf{x}_0$, para proporcionar un mayor control sobre las condiciones iniciales. Este se define como:

$$\mathbf{x}_0 = \mathbf{x}_r + \alpha \mathbf{x}_z,$$

donde $\mathbf{x}_z$ es un vector aleatorio con propiedades idénticas a las de su correspondiente $\mathbf{x}_r$, pero con distintos valores generados aleatoriamente. El parámetro α es un valor numérico arbitrario que permite controlar la distancia entre la solución inicial $\mathbf{x}_0$ y la solución real $\mathbf{x}_r$, lo que resulta útil para los experimentos.

Para garantizar la consistencia entre diferentes instancias de los experimentos y entre dos lenguajes de programación distintos, se ha decidido generar previamente todas las matrices y vectores aleatorios necesarios. Estos se almacenan y luego son leídos en tiempo de ejecución por los algoritmos, evitando así inconsistencias en la generación de números aleatorios entre los distintos entornos.

4.3. Análisis Comparativo: Diferencias Finitas vs. Paso Complejo

4.3.1. Motivación

El cálculo del producto jacobiano-vector representa el primer paso fundamental en nuestro análisis, siendo en esta etapa donde se originan los errores de cancelación, los cuales son cruciales para este estudio. Estos errores, generados en los primeros cálculos, se propagan a lo largo del proceso y afectan los resultados finales, lo que hace necesario un análisis exhaustivo de su impacto.

Por esta razón, el primer nivel de análisis se centrará en la obtención de este vector, el cual será utilizado posteriormente por los algoritmos GMRES y Newton-Krylov.

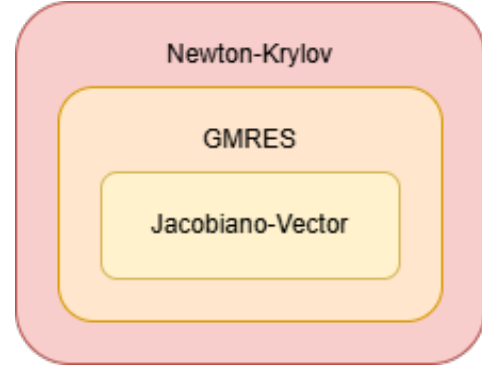


Figura 15: Jerarquía de los distintos métodos entre sí.

Fuente: Elaboración propia.

4.3.2. Obtención del Valor Numérico Real

Con el objetivo de obtener valores numéricos precisos para las derivadas, se empleó la biblioteca de Python `autograd`, que utiliza técnicas de diferenciación automática para calcular el jacobiano de manera exacta. Su función principal es servir como referencia, permitiendo obtener el valor exacto del producto entre la matriz jacobiana y el vector correspondiente, en lugar de recurrir a aproximaciones.

Este método **no** se incluye en el análisis comparativo, ya que solo se utilizará para obtener un resultado con el cual comparar los otros métodos. Se debe tener en cuenta que este método implica un mucho mayor esfuerzo computacional, pero proporcionará un resultado muy cercano al valor máximo teóricamente alcanzable por la máquina y al que apuntan obtener ambos métodos de aproximación.

A continuación, se presentan un par de gráficos que muestran el rendimiento de la diferenciación automática de `autograd`, comparado con los otros dos métodos numéricos en Python. La función fue evaluada en el punto $\mathbf{x} = [1, 2, 3]$, en la dirección $\mathbf{v} = \mathbf{1}_3$ y corresponde a la siguiente:

$$y_1 = e^{x_1+x_2} - \cos^3(x_1x_2), \quad y_2 = \sin^2(x_2) + x_3^2 - e^{x_2+x_3}, \quad y_3 = \cos^3(x_3x_1) - \sin^2(x_3) - x_1^2,$$

$$\mathbf{F}(\mathbf{x}) = \left(\frac{y_1}{y_2 + y_3}, \frac{y_2}{y_3 + y_1}, \frac{y_3}{y_1 + y_2} \right).$$

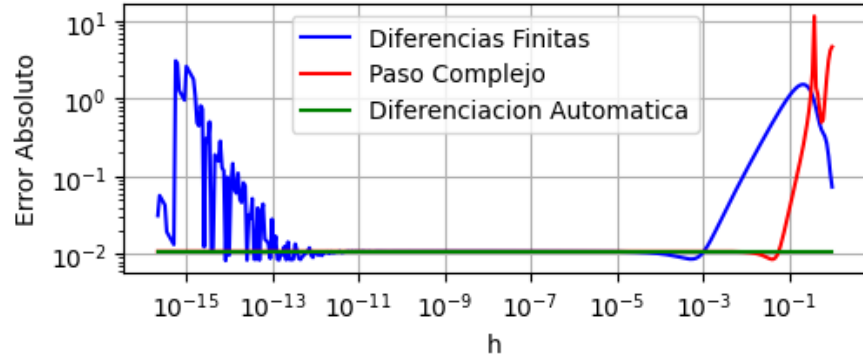


Figura 16: Error de la derivada entre los métodos.

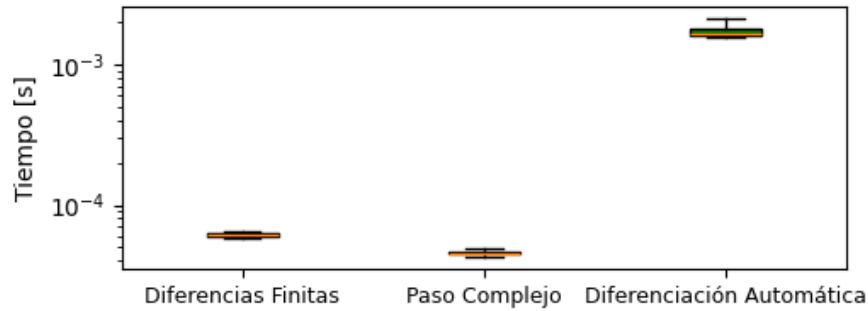


Figura 17: Tiempo de cómputo de la derivada entre los métodos.

4.3.3. Comparación de Resultados

Para evaluar la calidad de la solución proporcionada por ambos métodos de aproximación, se calcula la norma 2 de los valores entregados por los métodos y, posteriormente, el error absoluto entre esta norma y la norma de la derivada verdadera obtenida mediante autograd [aut,]. Es decir, graficaremos el error entre las normas, en lugar de la norma del error. Esto se hace por razones de memoria computacional, ya que almacenar miles de vectores resultado llega a consumir decenas de gigabytes por experimento, lo cual no es práctico. Sin embargo, el error entre las normas debería ser un indicador suficientemente confiable de la calidad de la solución.

En todos los experimentos siguientes, el punto objetivo para el cálculo de la derivada es un vector uniforme aleatorio de largo n , cuyos elementos están distribuidos entre $[-1, 1]$. Por su parte, el vector direccional v de la derivada corresponde a $\mathbf{1}_n$.

El experimento tiene una variable principal: el tamaño del paso diferencial, h , que varía desde la *epsilon machine* de float 64 hasta 1. Esto porque los errores de cancelación dependen directamente de este valor. Además, notece que en estos experimentos el valor h solo influye

como una constante multiplicativa, lo que implica que el tiempo de cómputo es independiente de este parámetro

Problema Normal

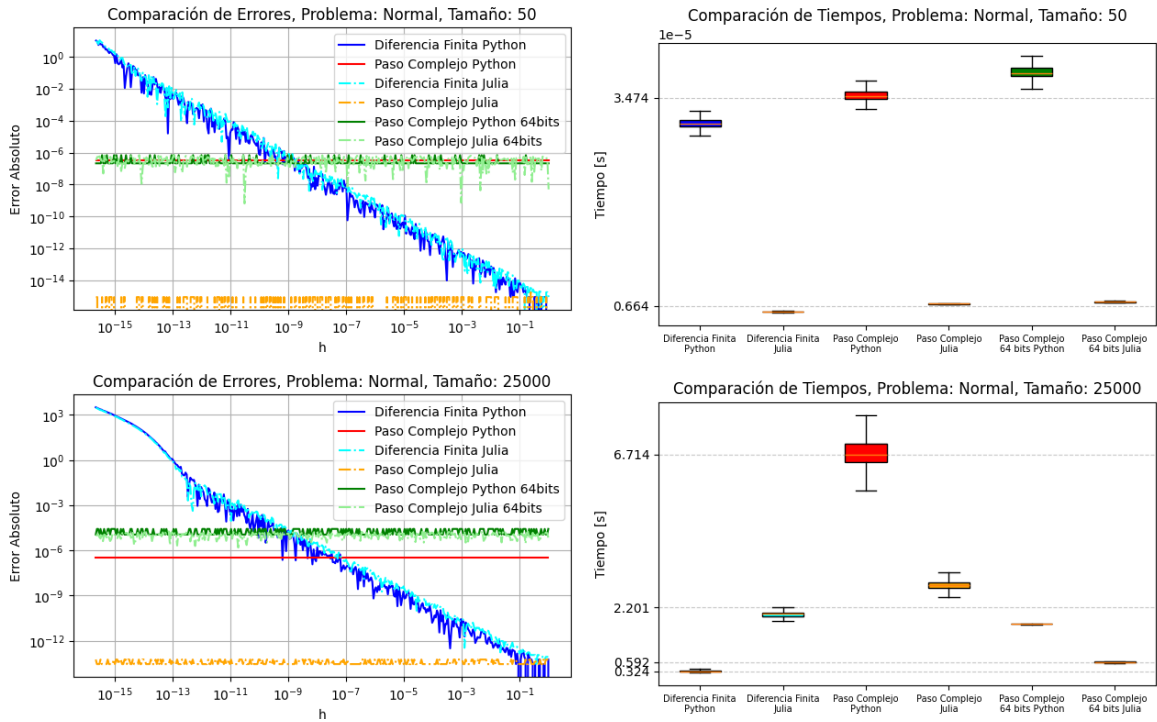


Figura 18: Error absoluto y tiempo de computo para el problema Normal.

Problema Sparse I

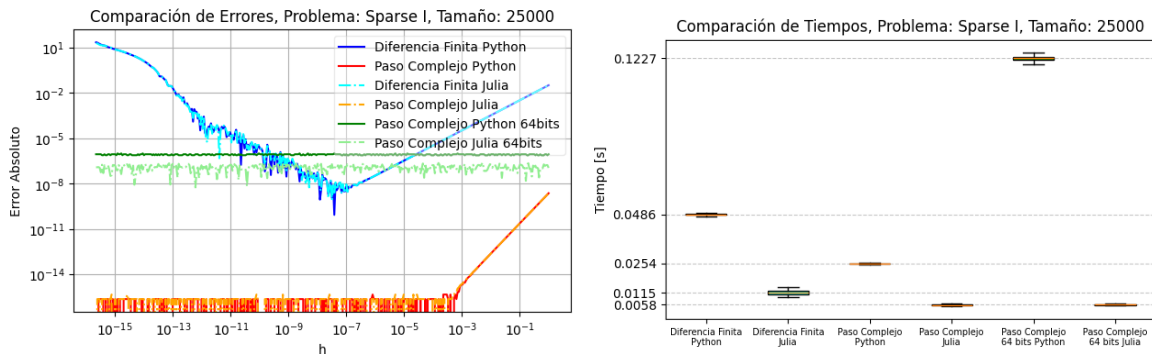


Figura 19: Error absoluto y tiempo de computo para el problema Sparse I.

Problema Dense II

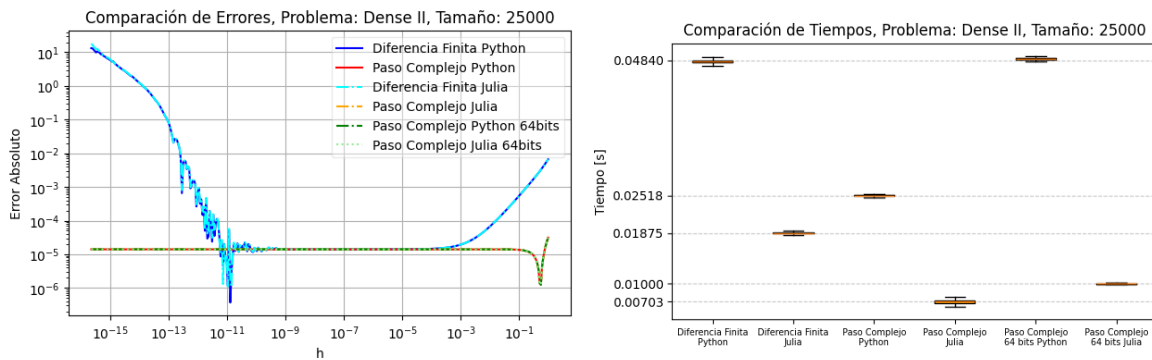


Figura 20: Error absoluto y tiempo de computo para el problema Dense II.

Experimentos adicionales no aportan mayor información; pueden ser encontrados en el anexo 5.10.

4.3.4. Conclusiones

Tras esta primera ronda de experimentos, se pueden observar diversos comportamientos. En primer lugar, los errores proporcionados por Python y Julia son esencialmente idénticos, permaneciendo superpuestos en los gráficos en casi todo momento, salvo cuando los errores numéricos comienzan a tomar relevancia. Esto es esperado, dado que los algoritmos son los mismos, y las diferencias radican principalmente en los tiempos de ejecución. Además, este comportamiento indica que las funciones entre ambos lenguajes son homólogas.

En segundo lugar, en los problemas lineales Normal y Uniform ocurren varios fenómenos. Se observa cómo el método de diferencias finitas comienza con un error casi nulo y cómo este incrementa a medida que se reduce el valor de h . Este comportamiento se debe a que, en funciones lineales, la derivada se puede calcular mediante una simple diferencia arbitraria entre dos puntos dividida por el tamaño del paso, como se puede ver a continuación, donde v es la dirección y c el tamaño arbitrario del paso:

$$\frac{\mathbf{F}(\mathbf{x} + \mathbf{v}c) - \mathbf{F}(\mathbf{x})}{c} = \frac{A(\mathbf{x} + \mathbf{v}c) + b - (A\mathbf{x} + b)}{c} \rightarrow \frac{A\mathbf{x} + A\mathbf{v}c - A\mathbf{x}}{c} \rightarrow \frac{A\mathbf{v}c}{c} \rightarrow A\mathbf{v} = J\mathbf{v}$$

sin embargo, lo expuesto es álgebra, pero el método numérico calcula todo y luego lo resta, por lo que al disminuir la distancia entre estos puntos los errores de cancelación aumentan, lo cual no ocurre con el método de paso complejo.

Además, en algunos problemas, como Normal, Uniform o Dense, se observa que el método de paso complejo parece tener una precisión limitada, a pesar de no existir una restricción explícita. Este comportamiento parece ser inconsistente tanto entre Python y Julia como entre los tamaños de los problemas. Esto sugiere que el fenómeno es producto de cómo los lenguajes manejan la aritmética de forma interna y no un artefacto del algoritmo de derivada, por lo que no debe ser considerado en futuros experimentos, ya que es una característica inherente a la función del problema y no al método utilizado.

Por último, se observa que el método de paso complejo de 64 bits da consistentemente peores resultados. Esto es esperado, dado que tiene una precisión numérica inferior que limita significativamente sus resultados. A excepción de los problemas lineales, este método también requiere un mayor tiempo de ejecución. Estas desventajas son suficientes para descartar este método de los experimentos subsecuentes.

4.4. Análisis Comparativo: Residuo Mínimo Generalizado (GMRES)

4.4.1. Motivación

La construcción del espacio de Krylov para resolver el sistema Jacobiano-Vector constituye un paso fundamental que depende de manera significativa de la aproximación de este producto, la cual se calcula mediante los métodos previamente descritos. Por ello, resulta crucial evaluar la calidad de los espacios generados a partir de los distintos métodos utilizados.

En este análisis, se evaluarán los métodos Restarted GMRES y LGMRES, ya que son los dos métodos GMRES disponibles en la biblioteca de SciPy para el algoritmo de Newton-Krylov, siendo LGMRES el método por defecto. En este contexto, el análisis se centrará en ambos. Cabe destacar que, en Julia, la implementación de BifurcationKit no incluye LGMRES, por lo que este método quedará fuera del análisis para este lenguaje.

En ambos métodos de Krylov, la cantidad máxima de iteraciones internas, m , se ha fijado en 20, ya que este es el valor por defecto establecido por SciPy en su implementación de Newton-Krylov.

4.4.2. Calidad del Espacio

Para medir la calidad del espacio, se analiza la ortogonalidad de los vectores de Arnoldi presentes en la matriz V , los cuales buscan ser ortonormales entre sí. Estos vectores dependen del sistema a resolver, que, como se discutió previamente, está influenciado por la aproximación del producto jacobiano-vector. Por lo tanto, es razonable suponer que los errores de cancelación pueden llegar a afectar la calidad de este espacio.

Para evaluar la ortogonalidad de los espacios generados, se puede aprovechar la propiedad de las matrices ortonormales. Una matriz M , cuyos vectores columna son mutuamente ortonormales, satisface la relación $M \cdot M^T = I$. Como métrica, se puede utilizar la suma de los valores absolutos de los elementos de $M \cdot M^T$, normalizada por la dimensión de la matriz y restando 1. Un valor cercano a cero indica un mayor grado de ortogonalidad en el espacio generado.

En los gráficos que se presentan a continuación, el eje horizontal muestra el tamaño del paso h y el vertical la distancia de la solución inicial α . El color indica el nivel de ortonormalidad del espacio discutido previamente; este tiene fijo el máximo como 1000, considerando cualquier valor mayor a este como irrelevante. Además, zonas sin colorear indican que el método cayó en algún fallo numérico, llevándolo a valores como `inf` o `nan`. Además, es importante tener en cuenta que la matriz V varía en cada iteración de GMRES y con ello su espacio, por lo que se ha optado por tomar la media como valor a graficar.

Problema Normal

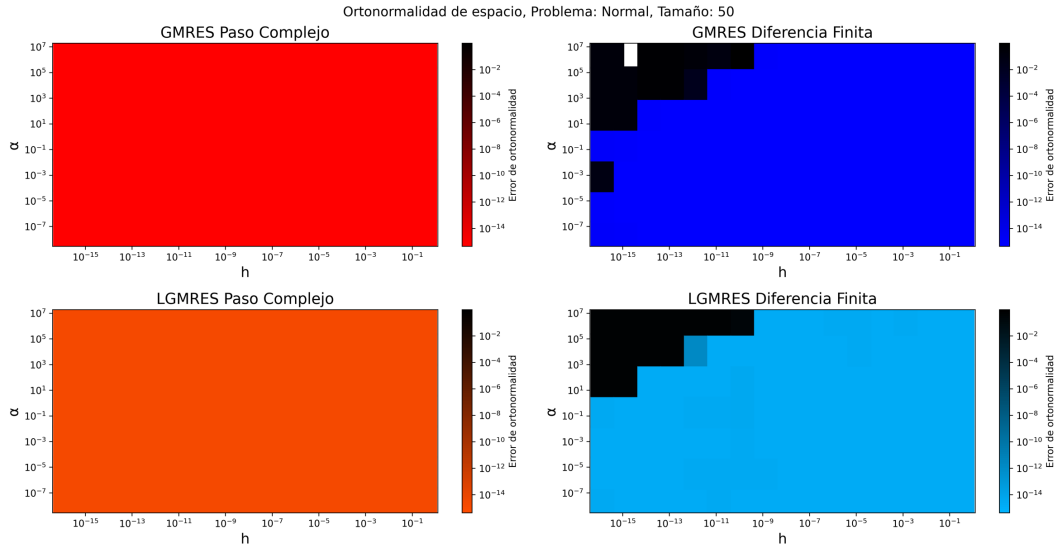


Figura 21: Ortonormalidad del espacio, problema Normal $n = 50$.

Problema Sparse I

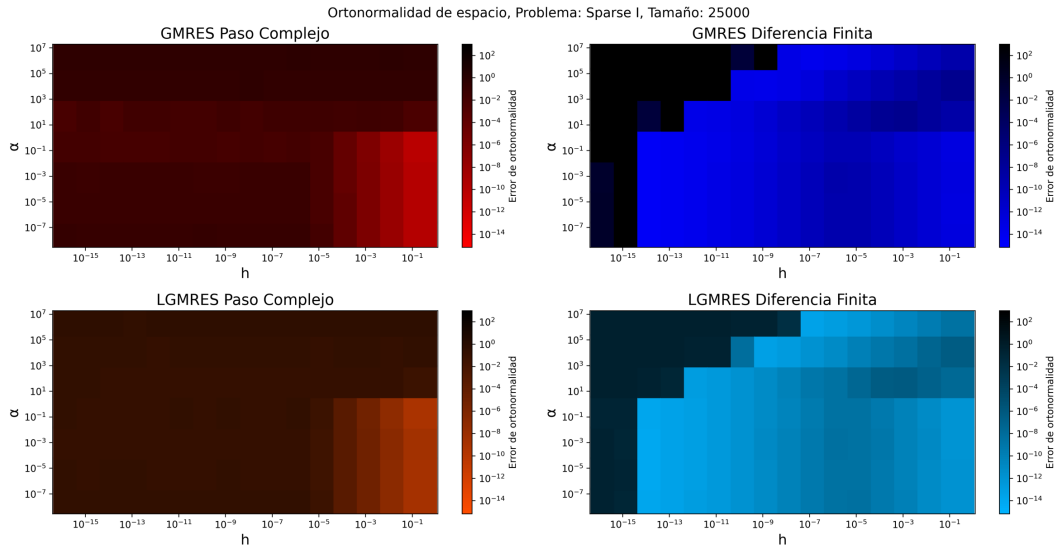


Figura 22: Ortonormalidad del espacio, problema Sparse I $n = 25000$.

Problema Dense I

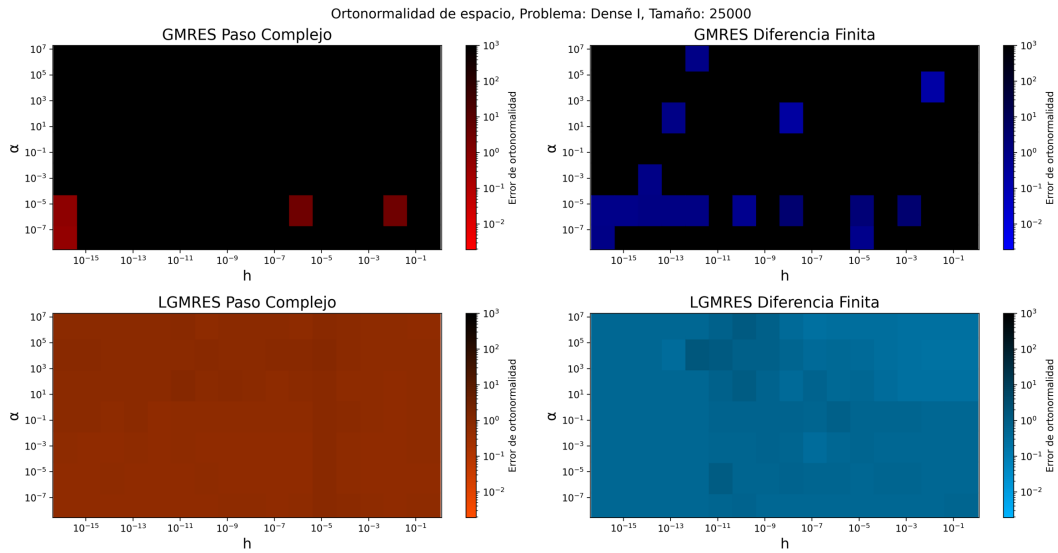


Figura 23: Ortonormalidad del espacio, problema Dense I $n = 25000$.

Problema Densé II

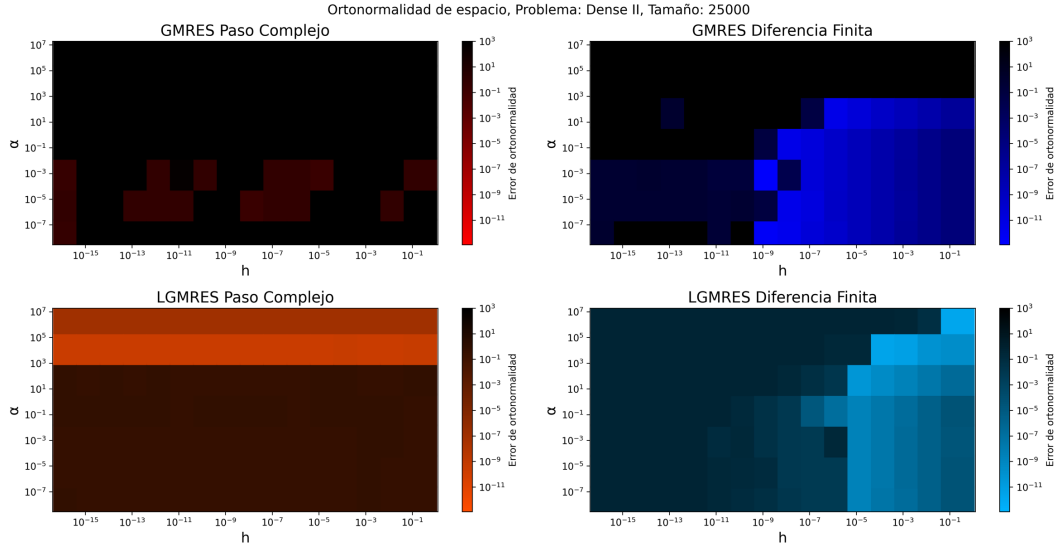


Figura 24: Ortonormalidad del espacio, problema Densé II $n = 25000$.

Experimentos adicionales no aportan mayor información; pueden ser encontrados en el anexo 5.11.

4.4.2.1. Conclusiones

Tras los experimentos realizados, se pueden identificar varios comportamientos. En primer lugar, se observa que la distancia α puede deteriorar significativamente la ortonormalidad del espacio. Este efecto parece impactar de forma más pronunciada el método de diferencias finitas que al método de paso complejo. Esto podría deberse a que un vector inicial de mayor magnitud genera mayores errores numéricos, lo cual también explicaría por qué el paso complejo resulta menos afectado. Además, un paso h más pequeño muestra, de manera consistente y abrupta, un empeoramiento en la solución obtenida por diferencias finitas, mientras que el paso complejo no necesariamente sufre el mismo efecto. Esto también es esperable, ya que los errores numéricos pueden conducir a valores de gran magnitud, dificultando los cálculos.

En segundo lugar, el método de paso complejo demuestra ser mucho más estable, manteniéndose generalmente constante a una misma altura. En contraste, las diferencias finitas presentan una mayor variación, que en algunas ocasiones resulta beneficiosa, ya que en contadas situaciones este método obtiene un valor inferior al del paso complejo.

Asimismo, se observa que los métodos GMRES y LGMRES pueden generar espacios considerablemente diferentes. Este comportamiento es esperado, ya que la principal diferencia entre ambos radica en el espacio generado. LGMRES, al incluir pasos adicionales de reutilización de errores, tiende a producir espacios con una altura más constante en comparación con su homólogo GMRES. Este comportamiento evidencia una diferencia entre GMRES y LGMRES, más bien no entre los métodos de paso complejo y diferencias finitas.

En conclusión, a pesar de las diferencias observadas, los métodos generan espacios bastante consistentes en términos de ortonormalidad, y ninguno muestra una superioridad clara sobre los demás. Esto no debe sorprender, ya que la calidad del espacio generado depende, en nuestro caso, de las magnitudes de los números utilizados. Como se observa en los gráficos, estas magnitudes rara vez alcanzan valores intratables debido a errores numéricos, siendo el método capaz de crear espacios de un nivel tolerable sin importar realmente las magnitudes con las que trabaje. Por lo tanto, los algoritmos logran crear un buen espacio ortonormal en la mayoría de los casos.

4.4.3. Comparación de Resultados

Hemos analizado los espacios de Krylov utilizados en la búsqueda de la solución. Sin embargo, este análisis no es suficiente, ya que lo que realmente importa es evaluar si el espacio generado representa adecuadamente el problema en estudio, independientemente del grado de ortonormalidad que posea.

Para evaluar la efectividad de los espacios generados en la búsqueda de la raíz, se realiza un experimento en el que se toma como solución final aquella obtenida tras una única iteración del método Newton-Krylov, es decir, el resultado de una sola llamada a GMRES.

En los gráficos que se presentan a continuación, el eje horizontal representa el tamaño del paso h y el vertical, la distancia inicial α . El color indica el residuo de la solución o el tiempo tomado; este tiene fijo el máximo como 1000, considerando cualquier valor mayor a este como irrelevante. Además, zonas sin colorear indican que el método cayó en algún fallo numérico, llevándolo a valores como `inf` o `nan`.

Además, se incluye un gráfico final, denominado gráfico de rendimiento, que compara el método de paso complejo con los métodos de diferencias finitas por medio de ratios, considerando tanto precisión como tiempo de ejecución. El código de colores utilizado es el siguiente: verde si el paso complejo es más preciso y más rápido, rojo si es menos preciso y más lento, azul si es menos preciso pero más rápido, y gris si es más preciso pero más lento. Destacar que en casos donde uno de los elementos del ratio no esté disponible, como cuando el método de diferencias finitas no converge, el punto no será graficado.

Problema Normal

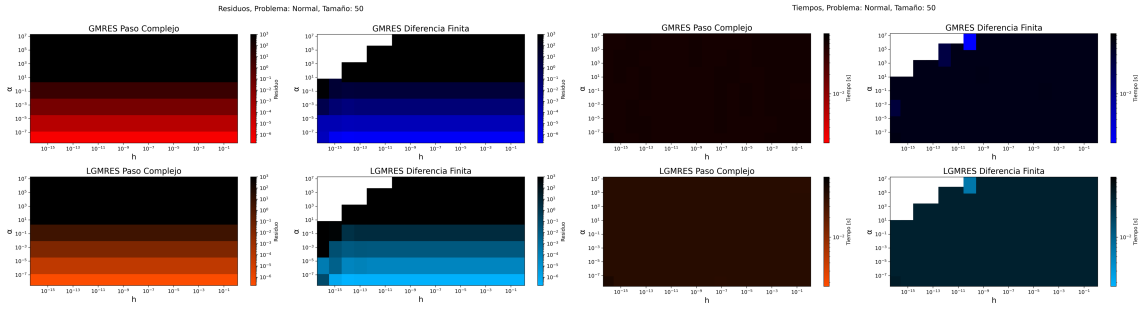


Figura 25: Error de un paso de Newton, Figura 26: Tiempo de un paso de Newton, prob. Normal $n = 50$.

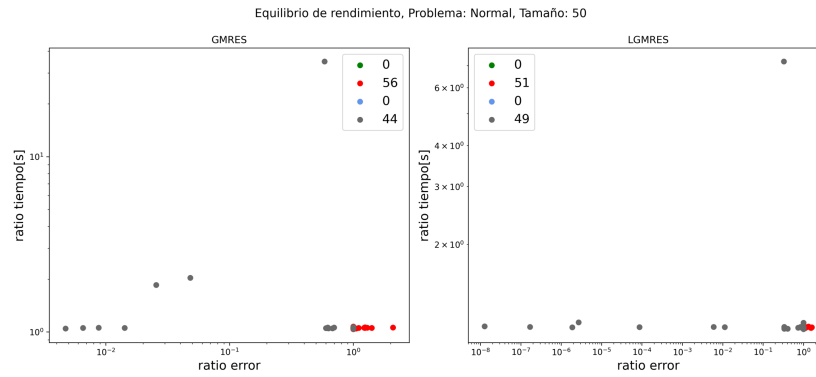


Figura 27: Equilibrio de rendimiento de un paso de Newton, prob. Normal $n = 50$.

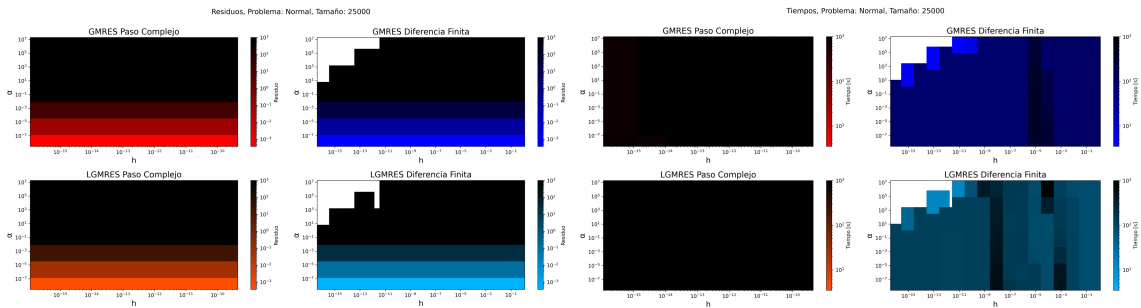


Figura 28: Error de un paso de Newton, Figura 29: Tiempo de un paso de Newton, prob. Normal $n = 25000$.

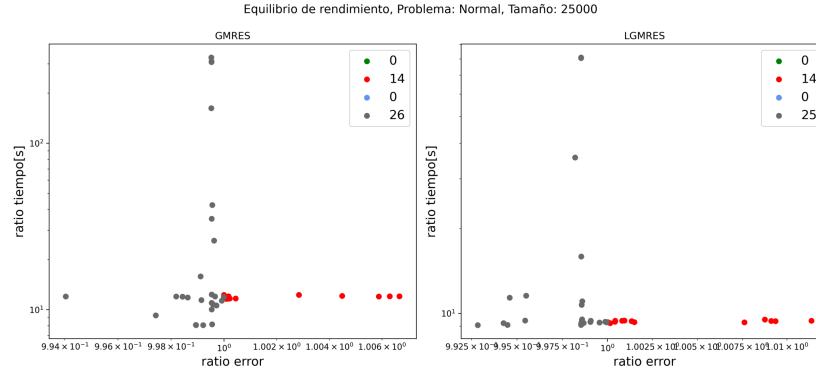


Figura 30: Equilibrio de rendimiento de un paso de Newton, prob. Normal $n = 25000$.

Problema Sparse I

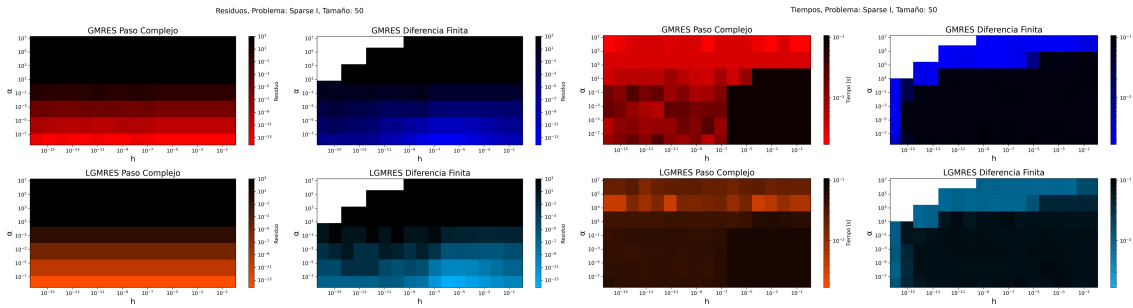


Figura 31: Error de un paso de Newton, Figura 32: Tiempo de un paso de Newton, prob. Sparse I $n = 50$.

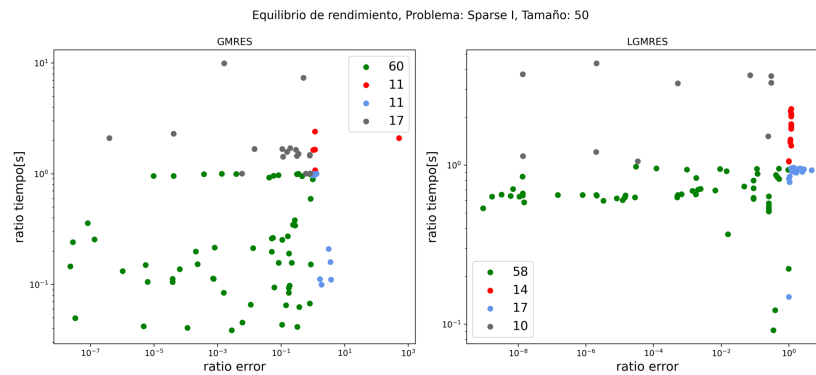


Figura 33: Equilibrio de rendimiento de un paso de Newton, prob. Sparse I $n = 50$.

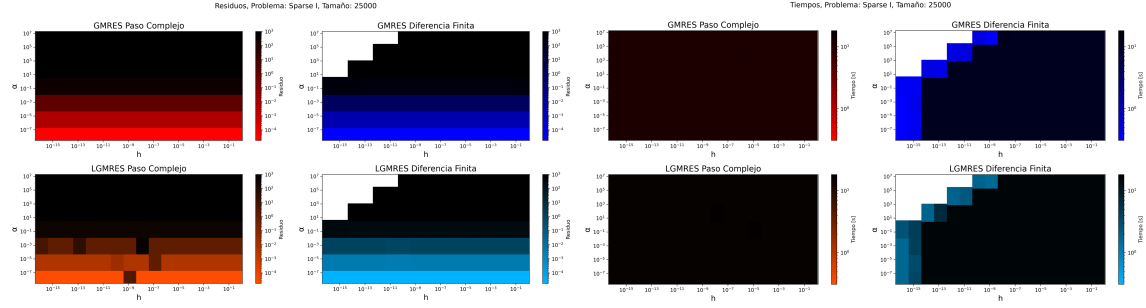


Figura 34: Error de un paso de Newton, Figura 35: Tiempo de un paso de Newton, prob. Sparse I $n = 25000$.

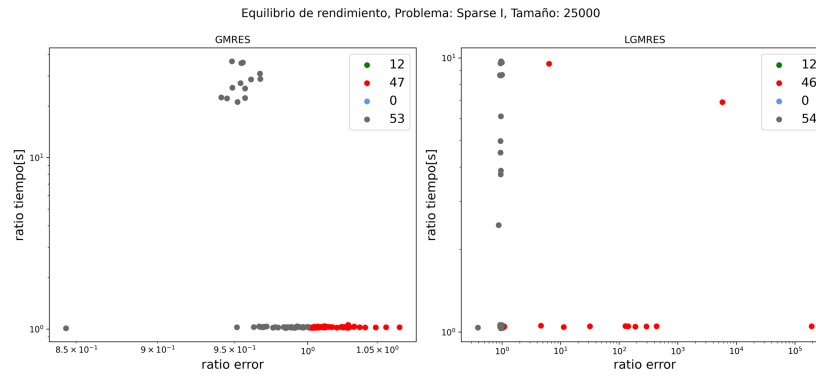


Figura 36: Equilibrio de rendimiento de un paso de Newton, prob. Sparse I $n = 25000$.

Problema Densa II

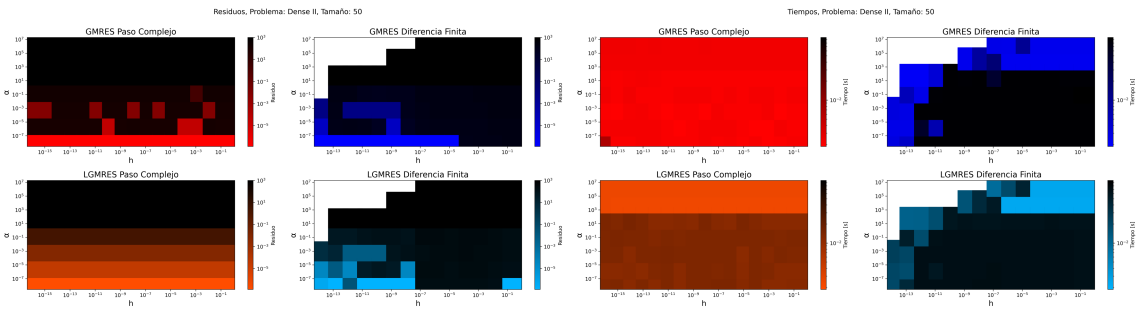


Figura 37: Error de un paso de Newton, Figura 38: Tiempo de un paso de Newton, prob. Densa II $n = 50$.

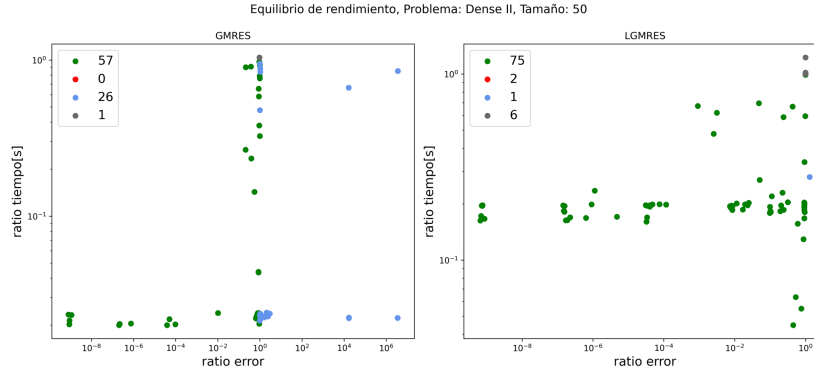


Figura 39: Equilibrio de rendimiento de un paso de Newton, prob. Dense II $n = 50$.

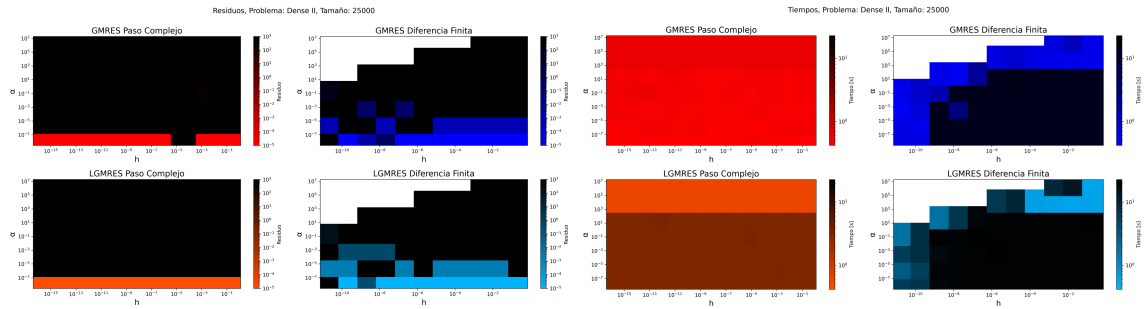


Figura 40: Error de un paso de Newton, Figura 41: Tiempo de un paso de Newton, prob. Dense II $n = 25000$.

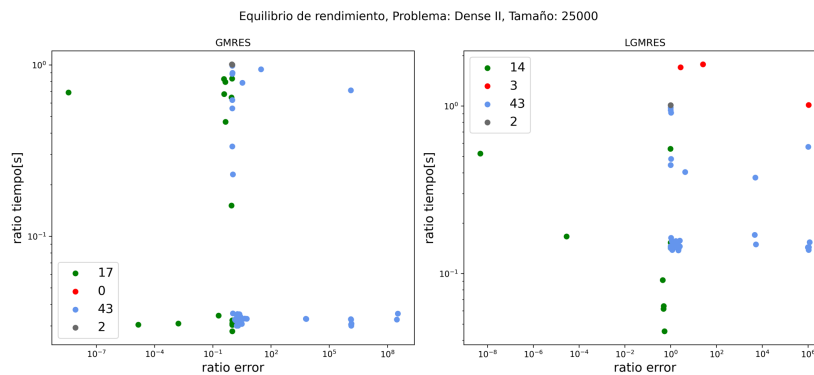


Figura 42: Equilibrio de rendimiento de un paso de Newton, prob. Dense II $n = 25000$.

Experimentos adicionales no aportan mayor información; pueden ser encontrados en el anexo 5.12.

4.4.3.1. Conclusiones

A partir de los experimentos, se pueden extraer varias conclusiones. En primer lugar, la distancia de la solución inicial parece afectar directamente la solución obtenida. Esto es lógico, ya que una solución inicial más alejada requerirá de más pasos para acercarse a la solución real.

También se observa que no existe una diferencia significativa entre los métodos GMRES y LGMRES. Aunque sus gráficos muestran diferencias claras en muchos casos, los rangos numéricos en los que ambas convergen son esencialmente los mismos, tanto en términos de tiempo como de error.

Se aprecia que el tiempo de ejecución se comporta de manera estable en la mayoría de casos, cosa que tiene sentido al estar realizando siempre una sola iteración., independientemente de la distancia inicial, lo que lleva a la decisión de eliminar estos problemas de futuros experimentos. También es relevante notar que, en el caso lineal, el método de paso complejo se muestra significativamente más lento, además de dar esencialmente el mismo resultado, lo cual se puede atribuir a lo discutido en 4.1.2, donde pareciera que la colosal cantidad de operaciones flotante-complejo ralentiza considerablemente el algoritmo. Por este motivo, además de que los problemas lineales no son un desafío en la mayoría de casos en la práctica, descontinuaremos los experimentos con estos problemas.

Un aspecto importante a destacar es la falta de convergencia por error numérico que presenta el método de diferencias finitas cuando el paso h es pequeño o la distancia α muy grande, este comportamiento se observa en todos los experimentos. El motivo es evidente: los errores de cancelación se vuelven tan significativos que el algoritmo no puede converger correctamente con tales valores. Además, se puede observar que el método de paso complejo nunca sufre este problema en ninguno de los experimentos, lo cual constituye una clara ventaja de este método.

En general, al observar los gráficos de rendimiento, se puede ver que el paso complejo parece ser consistentemente superior en la mayoría de los casos, en especial en los $n = 50$, o al menos supera a los otros métodos en uno de los dos criterios. Rara vez es peor, y en muchos casos, las diferencias son marginales. Este comportamiento es muy positivo, ya que confirma la hipótesis de que este método tiende a generar espacios que representan mejor el problema en comparación con las diferencias finitas, esto, al menos en una iteración. Una explicación para el empeoramiento del rendimiento en $n = 25000$ puede ser una amplificación del fenómeno discutido en 4.3.4.

4.5. Análisis Comparativo en Método Newton-Krylov

4.5.1. Motivación

El método de Newton-Krylov constituye el núcleo de este análisis, siendo el encargado de resolver el sistema de ecuaciones a través de una búsqueda de ceros. Representa el nivel más alto en el análisis, ya que implica una iteración completa de dicho método.

Este método, considerado como el estado del arte, servirá como referencia para la comparación con el método de paso complejo. La evaluación se realizará contra las implementaciones disponibles en las bibliotecas SciPy y BifurcationKit, por lo que el desempeño del método de Newton-Krylov se convierte en un aspecto fundamental de este análisis.

4.5.2. Análisis de Métodos de Ajuste Dinámico del Paso Diferencial

El método de referencia, Newton-Krylov en SciPy, utiliza diferencias finitas, lo que lo hace susceptible a errores de cancelación. No obstante, este inconveniente puede mitigarse mediante métodos que ajustan dinámicamente el tamaño del paso diferencial durante la ejecución, como se explicó en 3.6. Este mismo enfoque se ha implementado en la librería para Julia, garantizando la coherencia entre los experimentos realizados.

Para evaluar el impacto de estos métodos en el resultado final, se llevan a cabo experimentos comparativos en los que se elimina la adaptación del paso en el caso de las diferencias finitas y se incorpora en el método de paso complejo.

De los experimentos realizados, se esperan dos resultados principales: primero, que las diferencias finitas sufran una degradación significativa tanto en la calidad de la solución como en el tiempo de ejecución al eliminarse la adaptación del paso. En segundo lugar, se espera que el método de paso complejo no se vea afectado de manera significativa, ya que, una vez alcanzado el mínimo, el tamaño del paso diferencial se vuelve irrelevante.

Los gráficos de los experimentos se presentarán en formato de ratios, es decir, como la proporción de una magnitud respecto a otra. Dado que el objetivo de estos experimentos es comparar rendimientos, se utiliza esta forma para facilitar la visualización de los resultados.

4.5.3. Comparación de Adaptadores

En los gráficos que se presentan a continuación, el eje horizontal representa el tamaño del paso h y el vertical, la distancia inicial α . El color indica el ratio que se está comparando, siendo este *Sin Adaptador/Con Adaptador*; este tiene fijos los límites a $[0, 1, 10]$, con el objetivo

de que sea más fácil ver si aumenta o disminuye. Además, zonas sin colorear indican que el método cayó en algún fallo numérico, llevándolo a valores como `inf` o `nan`.

Problema Sparse II

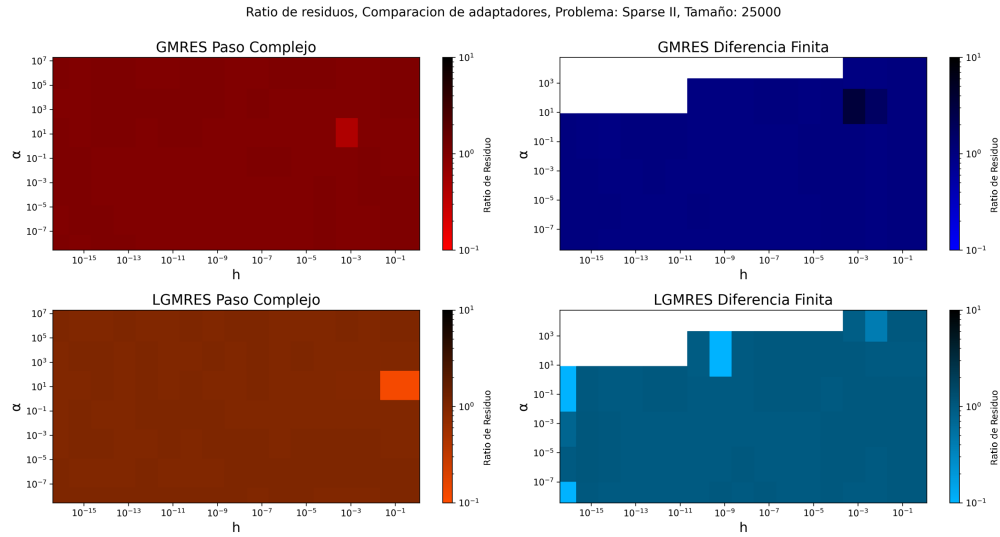


Figura 43: Ratio de error, Sparse II con adaptadores.

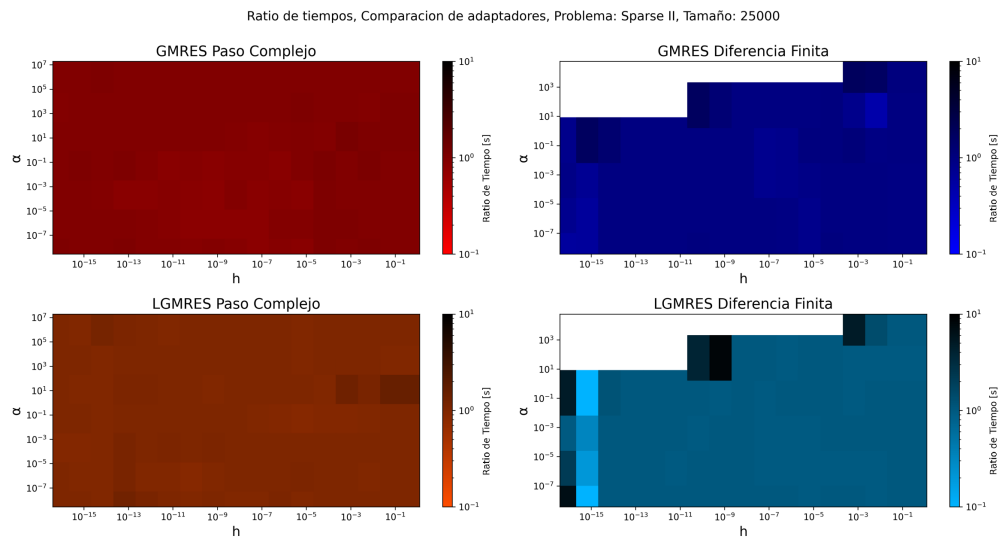


Figura 44: Ratio de tiempo, Sparse II con adaptadores.

Problema Dense I

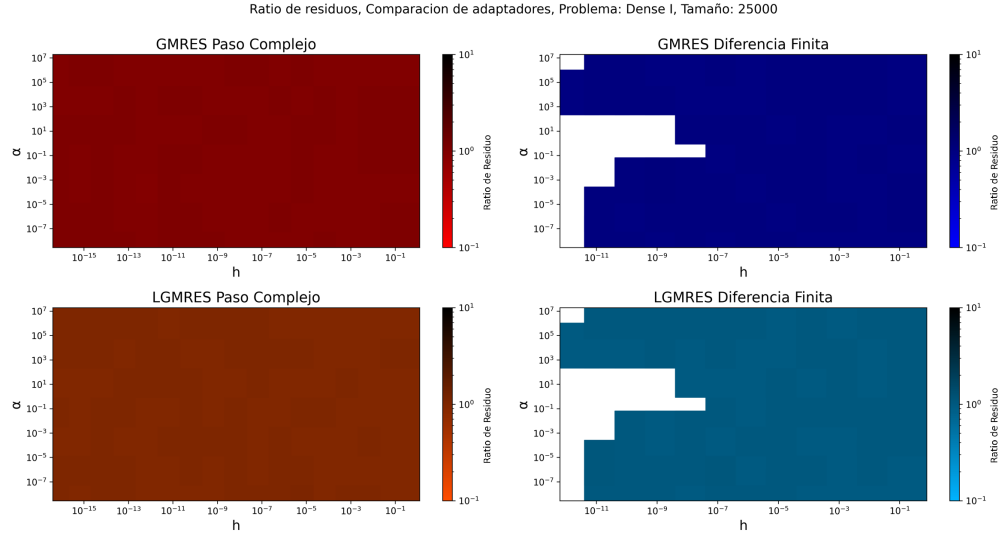


Figura 45: Ratio error, Dense I con adaptadores.

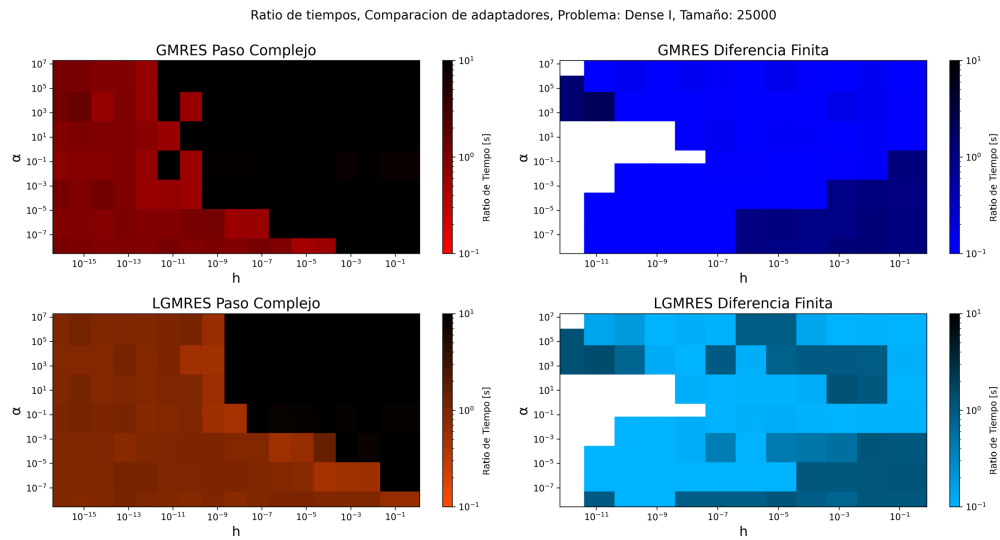


Figura 46: Ratio de tiempo, Dense I con adaptadores.

Experimentos adicionales no aportan mayor información; pueden ser encontrados en el anexo 5.13.

4.5.3.1. Conclusiones

Tras estos experimentos, se pueden observar comportamientos esperados. En primer lugar, el método de paso complejo no presenta una gran variabilidad dependiendo de si se utiliza un adaptador o no. Solo se observa una variación significativa cuando el paso h es demasiado grande, como en Dense I, momento en el cual el adaptador mejora la convergencia al reducir el tamaño de h . Sin embargo, no hay cambios relevantes cuando el paso ya es pequeño. Este comportamiento es esperado, ya que el adaptador está diseñado para mitigar los errores de cancelación, de los cuales el método de paso complejo no es susceptible. Por otro lado, el método de diferencias finitas sí se ve afectado por la ausencia de un adaptador, ya que sus variaciones, tanto en error como en tiempo, son mucho más significativas y consistentes. Esto indica que el adaptador es crucial para este último método, y es útil para ayudar a la estabilidad numérica. Con base en estas conclusiones, se optará por descartar el método de paso complejo con adaptador de paso en futuros experimentos, ya que demuestra ser obsoleto en la mayoría de los casos.

Además, se puede observar que no existen diferencias significativas entre los métodos GMRES y LGMRES, y la variación debida a la distancia entre la solución inicial y la solución real no parece verse afectada por los adaptadores.

4.5.4. Comparación de Resultados

En los gráficos que se presentan a continuación, el eje horizontal representa el tamaño del paso h y el vertical, la distancia inicial α . El color indica el ratio que se está comparando, siendo este *Método de paso complejo/Método de diferencias finitas*; este tiene fijos los límites a $[0, 1, 10]$, con el objetivo de que sea más fácil ver si aumenta o disminuye. Además, zonas sin colorear indican que el método cayó en algún fallo numérico, llevándolo a valores como `inf` o `nan`.

Asimismo, se incluye un último grupo de gráficos, denominados gráficos de rendimiento, en los que se muestra un resumen de cómo se compara el método de paso complejo con los diferentes métodos de diferencias finitas, considerando simultáneamente precisión y tiempo. El código de colores utilizado es el siguiente: verde si el paso complejo es más preciso y más rápido, rojo si es menos preciso y más lento, azul si es menos preciso pero más rápido, y gris si es más preciso pero más lento. Destacar que en casos donde uno de los elementos del ratio no esté disponible, como cuando el método de diferencias finitas no converge, el punto no será graficado.

Problema Sparse I

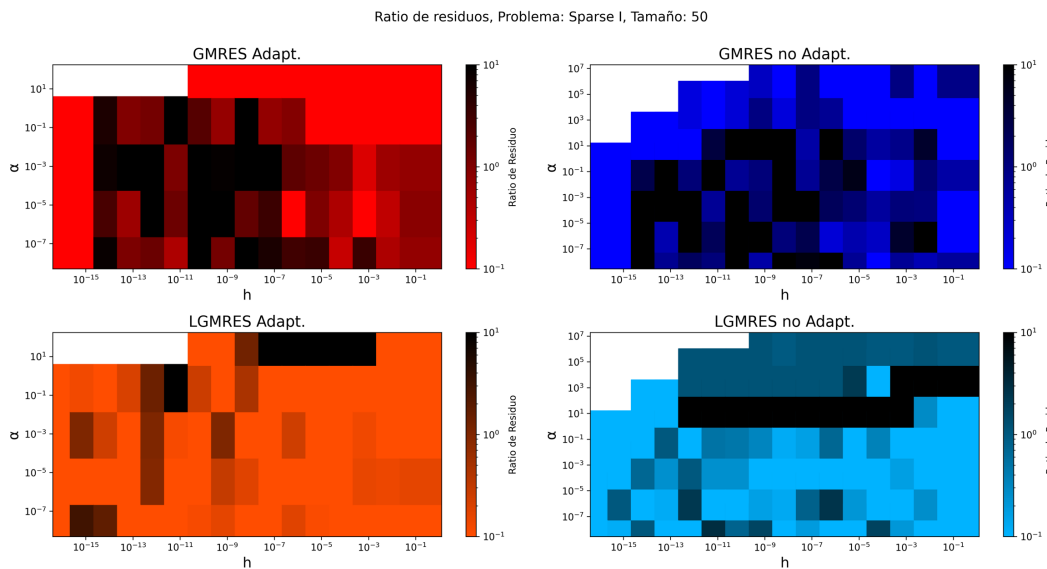


Figura 47: Ratio error, problema Sparse I, $n = 50$.

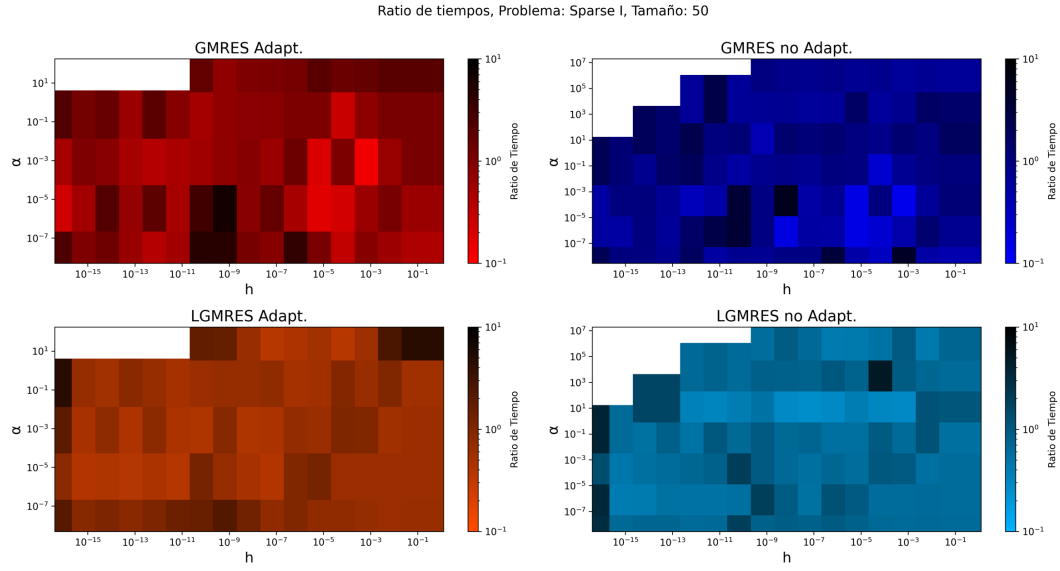


Figura 48: Ratio tiempo, problema Sparse I, $n = 50$.

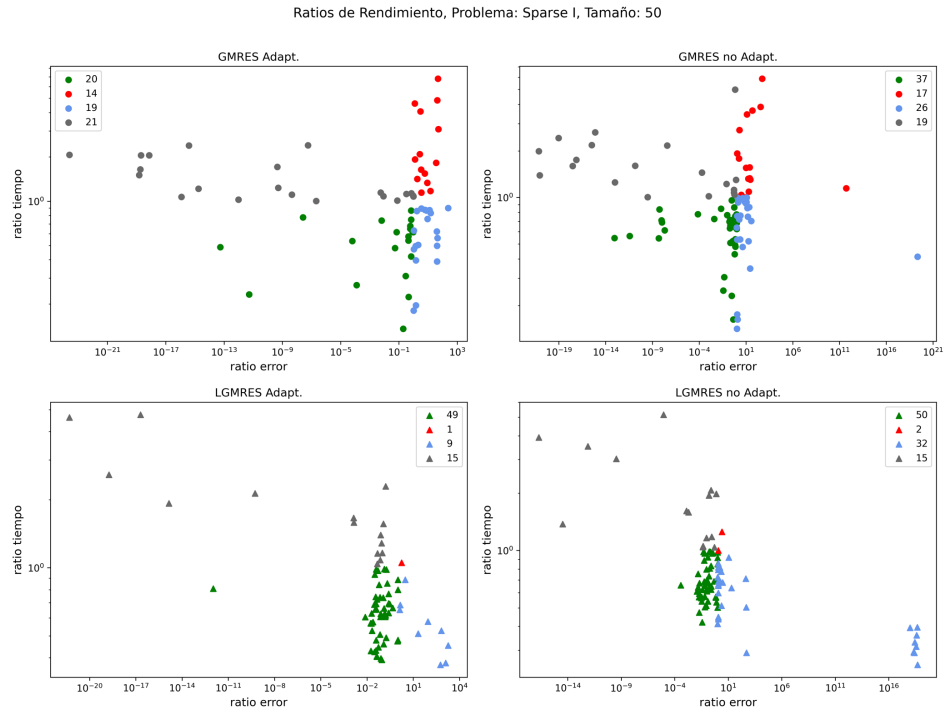


Figura 49: Equilibrio de rendimiento, problema Sparse I, $n = 50$.

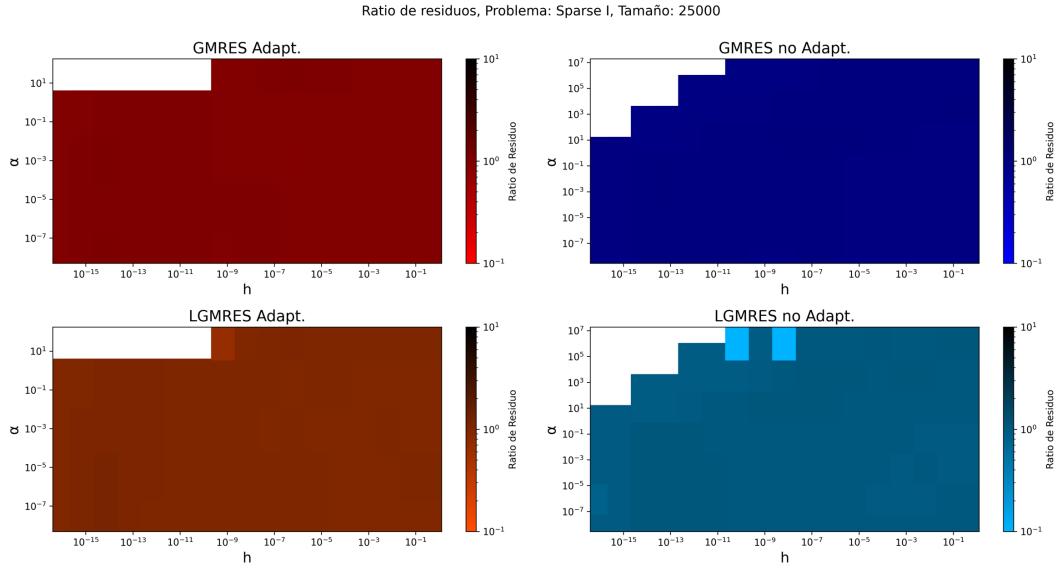


Figura 50: Ratio error, problema Sparse I, $n = 25000$.

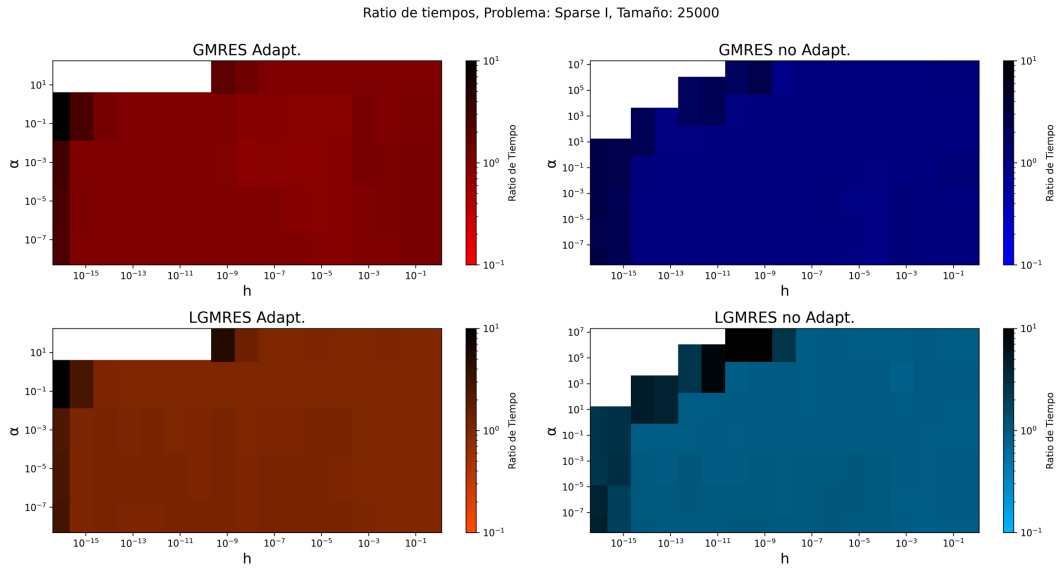


Figura 51: Ratio tiempo, problema Sparse I, $n = 25000$.

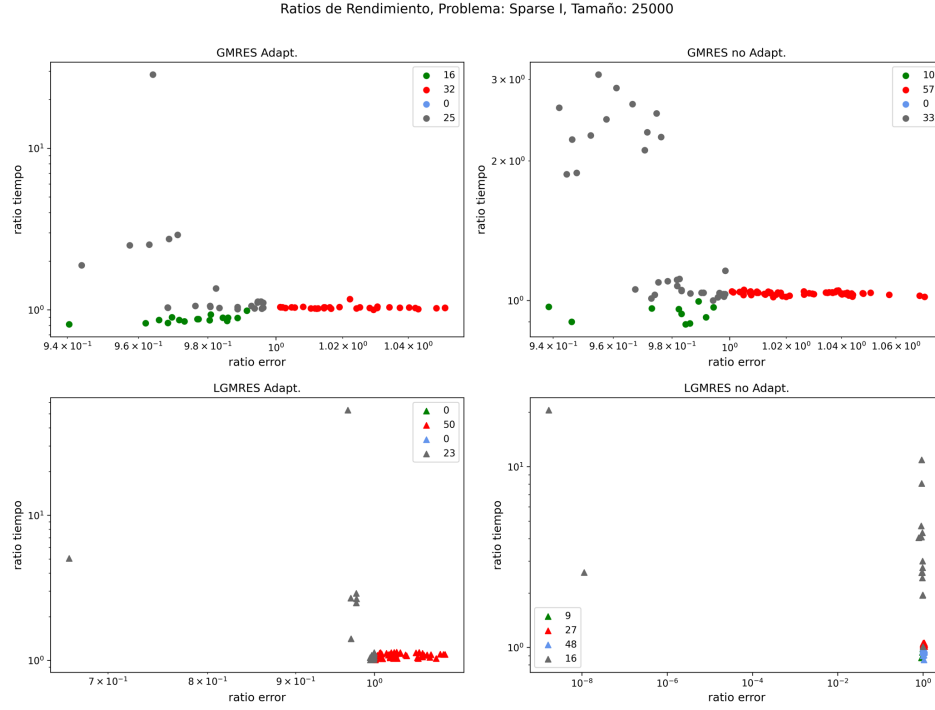


Figura 52: Equilibrio de rendimiento, problema Sparse I, $n = 25000$.

Problema Dense II

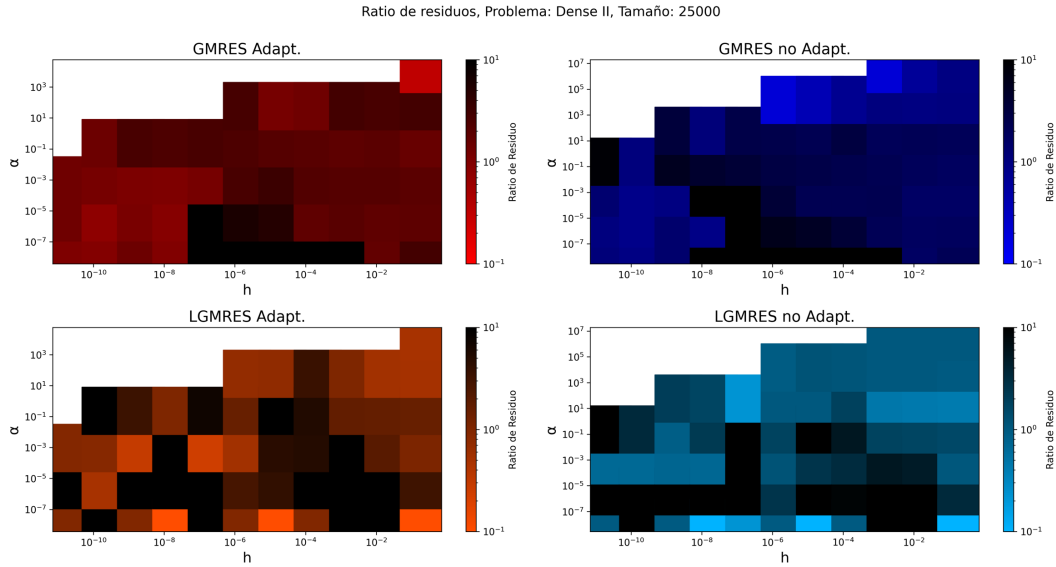


Figura 53: Ratio error, problema Dense II, $n = 25000$.

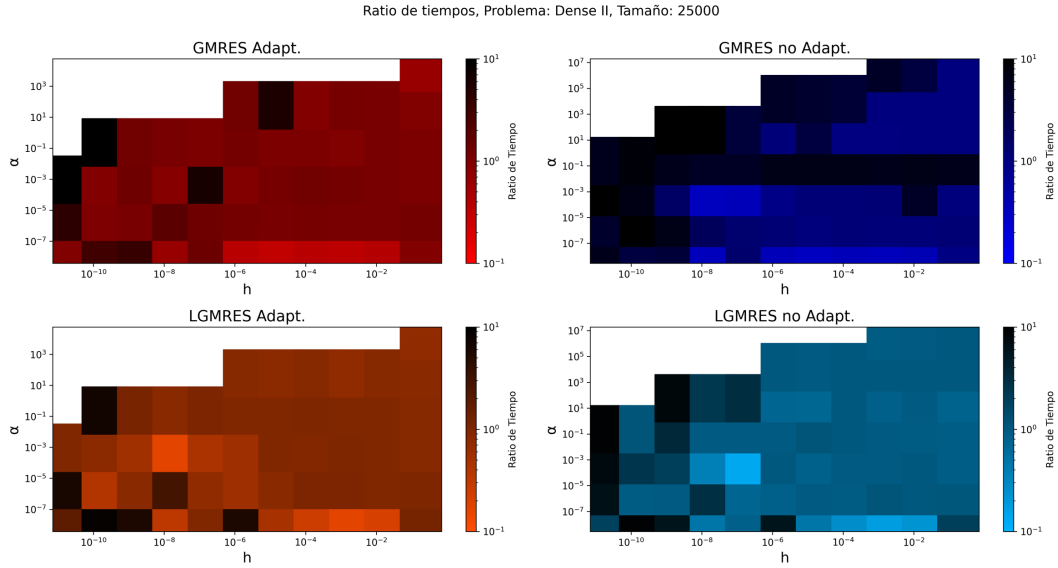


Figura 54: Ratio tiempo, problema Dense II, $n = 25000$.

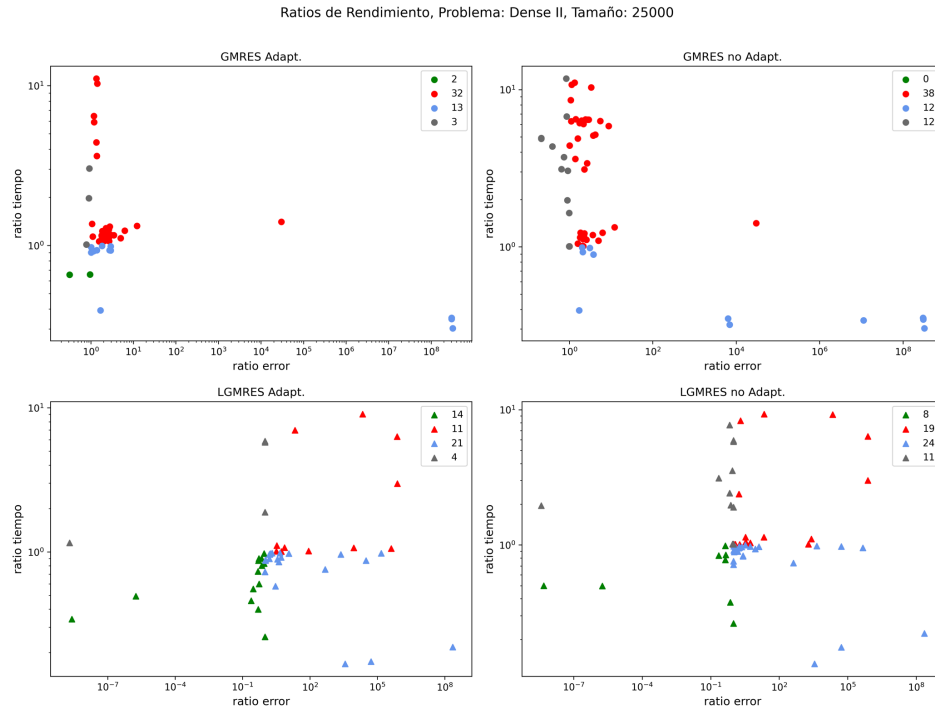


Figura 55: Equilibrio de rendimiento, problema Dense II, $n = 25000$.

Experimentos adicionales no aportan mayor información; pueden ser encontrados en el anexo 5.14.

4.5.5. Conclusiones

Tras esta ronda final de experimentos, se pueden observar comportamientos interesantes. En primer lugar, se nota que al eliminar el adaptador de paso en el método de diferencias finitas, el comportamiento de este varía. Sin embargo, después de las iteraciones del método de Newton, se logra encontrar una solución final lo suficientemente precisa, aunque este comportamiento puede variar entre los diferentes problemas.

Por otro lado, al comparar ambos métodos, se observa que son prácticamente idénticos en términos de calidad de la solución y tiempo de ejecución, teniendo una distribución de variación muy similar entre sí. El método de paso complejo es marginalmente inferior en la mayoría de los casos, aunque aún es capaz de competir de manera eficiente con el estado del arte.

En estos experimentos también se ha modificado el valor de α , lo cual permite realizar una comparación más exhaustiva del funcionamiento de los métodos. Al hacerlo, podemos analizar con mayor detalle las distintas convergencias, observando cómo, en algunos casos, alejar la solución inicial provoca que la variación en el rendimiento sea más estable. Esto no implica que el método converja más rápido o mejor, sino que las condiciones iniciales parecen tener menos influencia. Sin embargo, este comportamiento es relevante para la función puesta a prueba y no para el método en sí.

La aparente inferioridad del método de paso complejo no debe interpretarse como un fracaso. Lo que se demuestra es que un método tan sencillo, sin la necesidad de adaptadores, produce resultados muy buenos en general, siendo muy similares a los del método homólogo. Aunque la implementación realizada en este trabajo no logró superar de manera consistente al estado del arte, ha mostrado que el método de paso complejo tiene un gran potencial. Este podría llegar a funcionar mejor si se implementara específicamente para problemas particulares. Las variaciones realizadas en esta implementación fueron generalmente mínimas, y gran parte del backend aún asume que se está trabajando con diferencias finitas.

4.6. Comparación entre Python y Julia

4.6.1. Motivación

La comparación entre Python y Julia ha sido un tema clave a lo largo de este trabajo, ya que uno de los objetivos principales es fomentar el uso del método propuesto. En este contexto, se consideró relevante trasladar este análisis y los métodos a un lenguaje emergente como Julia, que ha demostrado un gran potencial en el ámbito del análisis numérico. La motivación detrás de esta decisión radica en que Julia, al ser un lenguaje optimizado para el rendimiento en cálculos científicos y numéricos, ofrece ventajas significativas en términos de velocidad y eficiencia. Además, al promover su adopción, se busca contribuir al crecimiento y fortale-

cimiento de una comunidad que respalde el desarrollo de herramientas especializadas. En este sentido, el objetivo secundario de este análisis es fomentar aún más el uso de Julia, ampliando su aplicación en áreas que tradicionalmente han estado dominadas por lenguajes más establecidos como Python.

4.6.2. Comparación

4.6.2.1. Error de la solución

Como era de esperarse, debido a la fijación de todos los vectores y matrices aleatorias, así como a la correcta homologación entre las funciones de los problemas, como se observó claramente en 4.3.3, las soluciones obtenidas en ambos casos son esencialmente idénticas. Esto era previsible, ya que ambos métodos deberían seguir el mismo procedimiento.

Sin embargo, al realizar el mismo análisis pero aplicando el método de Newton, se observa un comportamiento ligeramente relevante.

A continuación se muestra un gráfico de ratio entre el error de las soluciones finales de Julia y Python en Newton-Krylov.

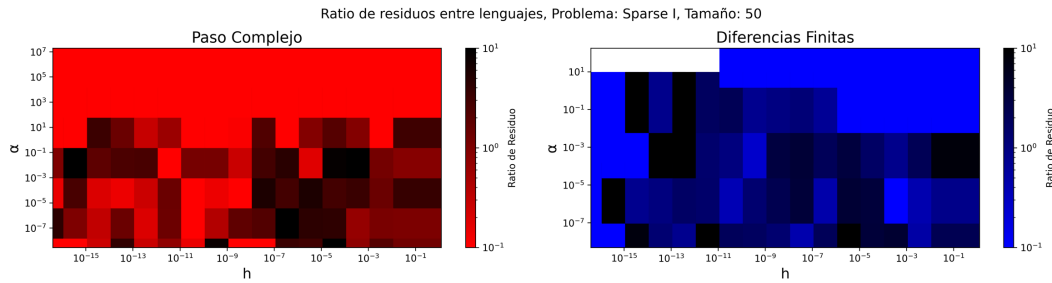


Figura 56: Ratio de error Julia/Python, problema Sparse I, $n = 50$.

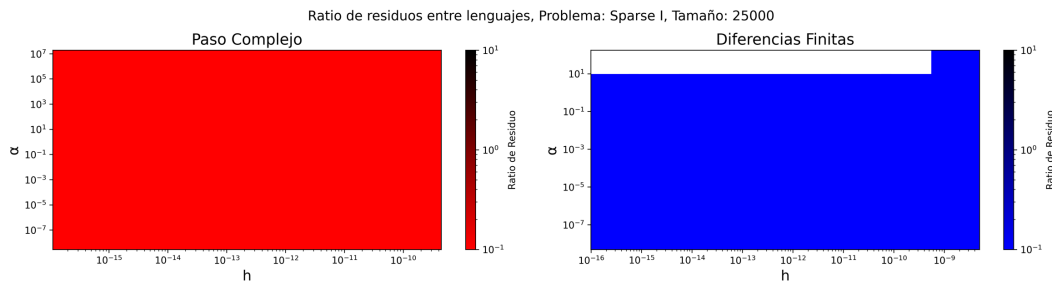


Figura 57: Ratio de error Julia/Python, problema Sparse I, $n = 25000$.

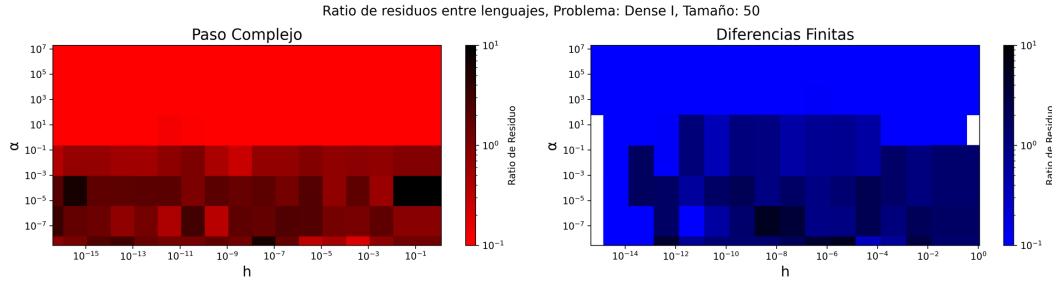


Figura 58: Ratio de error Julia/Python, problema Dense I, $n = 50$.

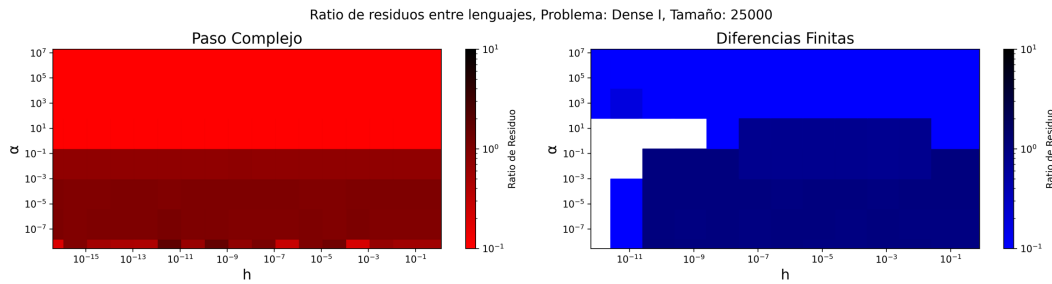


Figura 59: Ratio de error Julia/Python, problema Dense I, $n = 25000$.

Julia parece tener, en algunos casos, un rendimiento ligeramente inferior en la solución final, lo cual podría parecer una contradicción respecto a los resultados anteriores. Sin embargo, esto puede explicarse por la forma en que maneja la aritmética de manera interna o por otros pasos realizados por las funciones y métodos desde el backend de la librería, factores que escapan a nuestro control como usuarios. Sin embargo, por lo general, el resultado final es bastante similar, lo cual es de esperar.

4.6.2.2. Tiempo de Cómputo

Como era de esperarse, debido a la naturaleza del lenguaje, y como se observó en 4.3.3, Julia demuestra ser notablemente más rápida que Python en lo que respecta a cálculos numéricos simples en la mayoría de los casos.

Sin embargo, al igual que en el caso anterior, al realizar el mismo análisis pero aplicando el método de Newton, se observa un comportamiento relevante respecto a la escalabilidad del problema.

A continuación se muestran gráficos que muestran el ratio entre el tiempo de cómputo de las soluciones finales obtenidas por Julia y Python en el contexto de Newton-Krylov.

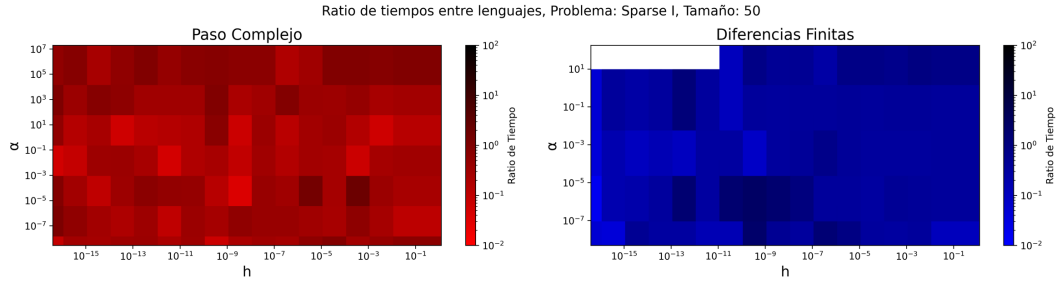


Figura 60: Ratio de tiempo Julia/Python, problema Sparse I, $n = 50$.

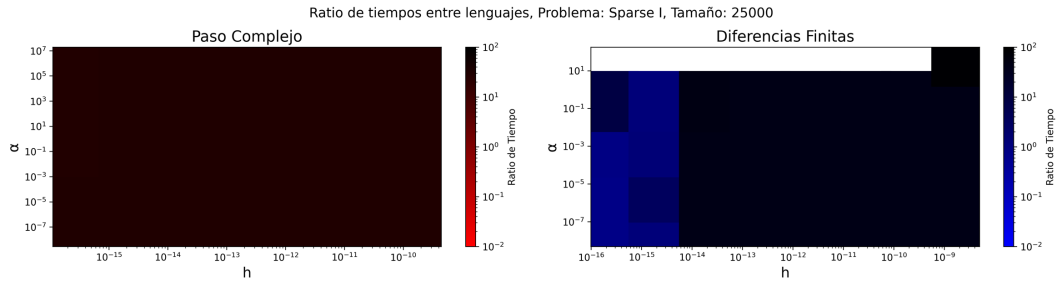


Figura 61: Ratio de tiempo Julia/Python, problema Sparse I, $n = 25000$.

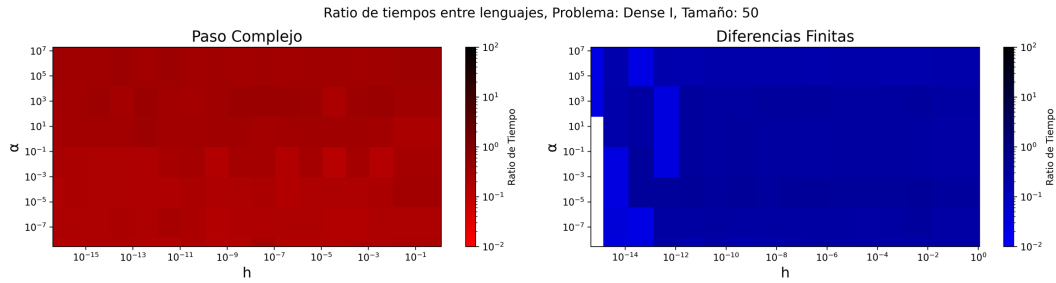


Figura 62: Ratio de tiempo Julia/Python, problema Dense I, $n = 50$.

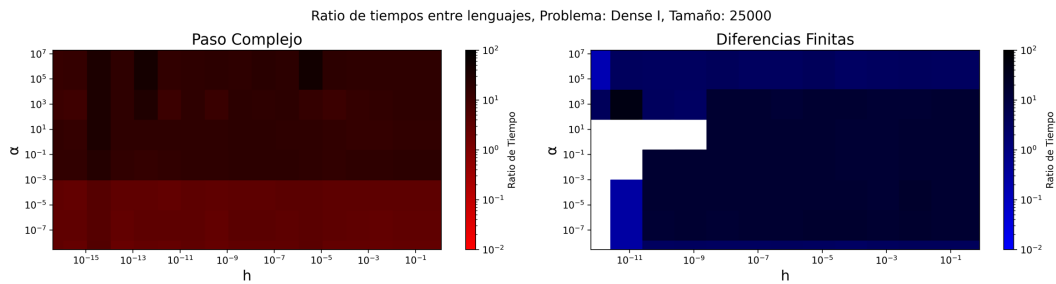


Figura 63: Ratio de tiempo Julia/Python, problema Dense I, $n = 25000$.

Aquí podemos observar que la diferencia entre los tiempos es consistentemente desfavorable para Julia en los problemas grandes, lo cual puede explicarse de manera similar al caso anterior. Sin embargo, en este caso es necesario prestar más atención, ya que el objetivo principal de Julia es ser más rápido, lo cual no parece cumplirse en el algoritmo de Newton-Krylov, pero sí cuando se limita a aritmética más simple. Esto sugiere que podría ser necesaria una reestructuración completa del backend del método en la librería utilizada.

4.6.3. Conclusiones

Julia ha demostrado ser más veloz que Python en cálculos aritméticos simples, además de entregar la misma solución. Sin embargo, como se observó claramente, al implementar un algoritmo más complejo utilizando la librería `BifurcationKit`, el método resultante ha mostrado un rendimiento consistentemente inferior al hacerse más grande. Es más, al ser de un tamaño reducido, tampoco parece ser más rápido, sino que es bastante equivalente. Esto podría deberse a dos motivos principales: la construcción del método podría no haber hecho un uso óptimo de las herramientas disponibles, o bien, el backend de la librería podría ser inadecuado para este tipo de problemas. De todas formas, una implementación adecuada de estos métodos debería ser completamente posible en este lenguaje, siempre que se tenga un buen entendimiento de los procesos y las librerías disponibles.

Ambos lenguajes y bibliotecas son de código abierto, por lo que no es descabellado pensar que en el futuro Julia pueda consolidarse de forma más robusta, siendo capaz de competir de manera consistente contra grandes bibliotecas como `SciPy`.

4.7. Validación de Experimentos Anexos

4.7.1. Motivación

Como se ha mencionado anteriormente, el objetivo no es únicamente evaluar el desempeño del método de paso complejo en la resolución de sistemas de ecuaciones, sino también explorar su aplicación en contextos más amplios. Por ello, se consideró relevante realizar una evaluación de su desempeño en otras áreas, aunque sin profundizar tanto en los detalles como se hizo en las secciones anteriores.

Este enfoque tiene como objetivo proporcionar una visión más amplia de la aplicabilidad del método de paso complejo, más allá de su uso específico en la resolución de ecuaciones lineales.

4.7.2. Extensión de Puntos de Aproximación

Tanto el método de diferencias finitas como el de paso complejo se fundamentan en la expansión de Taylor. Como se ha mostrado en la sección 2.3.4, es posible generar una cantidad infinita de variaciones de ambos métodos a partir de esta base, lo que hace relevante analizar el comportamiento de dichas variaciones.

4.7.2.1. Métodos de aproximación utilizando un mayor número de puntos

La incorporación de más puntos en la expansión introduce nuevos factores en la aproximación. Por un lado, contribuye a reducir el orden del error teórico, pero, por otro, incrementa la complejidad de la expresión. Esta mayor complejidad puede dar lugar a un aumento en los errores numéricos, lo que compromete la estabilidad de la solución y limita su efectividad práctica.

La función a la que se calculará la derivada es: $F(x) = \frac{e^x}{\sin^3(x) + \cos^3(x)}$, y se evaluará en el punto $x = 1$. Este experimento permitirá observar el impacto de los errores numéricos y la estabilidad de los resultados en casos de mayor complejidad.

Para el método de diferencias finitas, se construirán a partir del conjunto de puntos $P_k = \{0, h, 2h, \dots, kh\}$. Para el método de paso complejo, se construirán a partir del conjunto de puntos $P_k = \{ih, 2ih, \dots, kih\}$.

Se observa que, a medida que se añaden más puntos, la convergencia del método de diferencias finitas es más rápida y, solo en ocasiones, de mejor calidad. Sin embargo, los errores de cancelación tienden a manifestarse con valores de h mayores. Por otro lado, el método de paso complejo también se ve afectado tanto por el incremento en el orden de convergencia como por los errores de cancelación, aunque de manera más lenta.

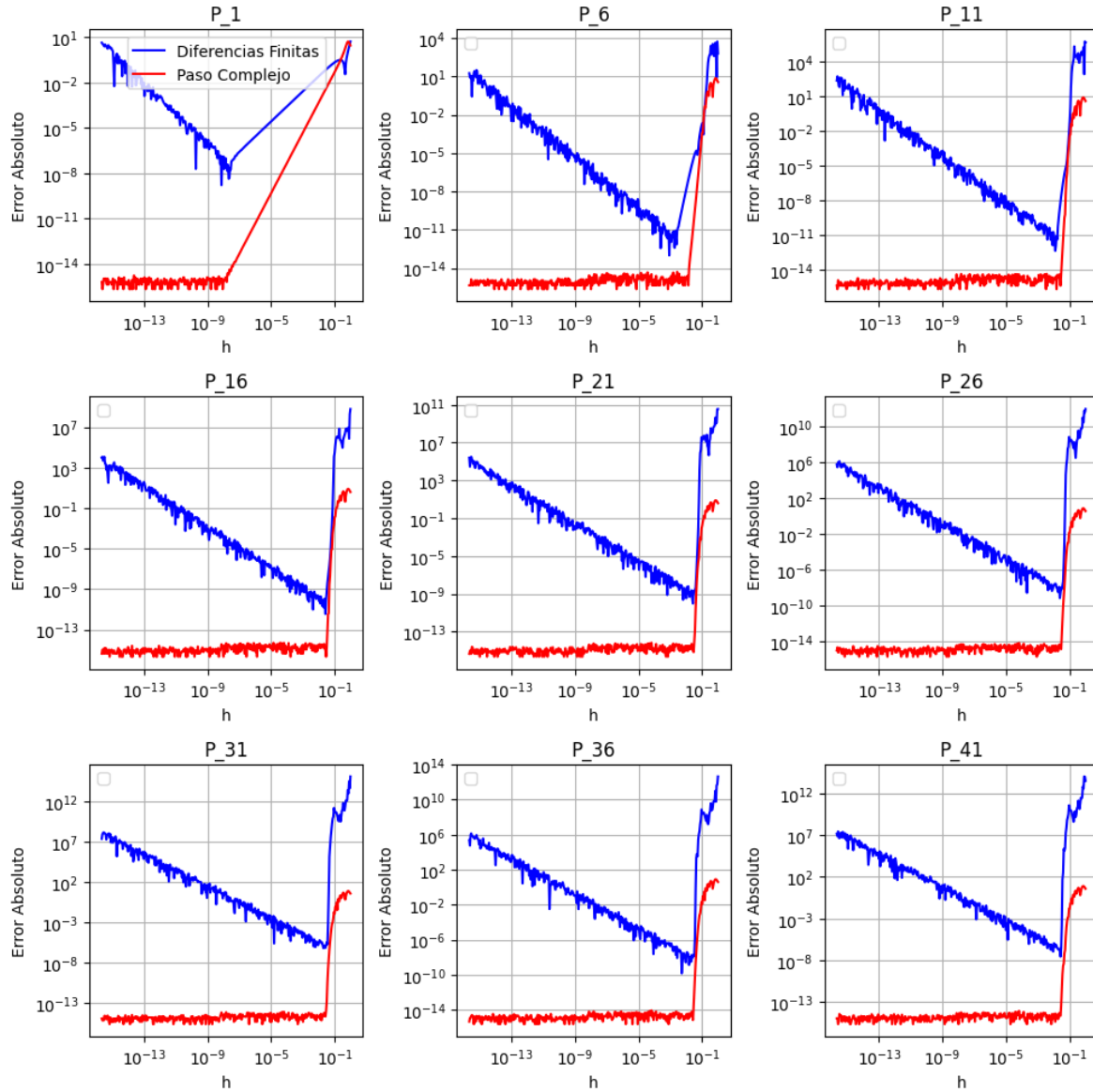


Figura 64: Evolución del error a medida que se añaden puntos a las aproximaciones.

4.7.2.2. Selección de Puntos no Múltiplos Enteros de h

A lo largo de este análisis, hemos trabajado con conjuntos de puntos que son múltiplos enteros del paso h , como $P_k = \{0h, 1h, 2h, \dots, kh\}$, donde h siempre está multiplicado por un número entero, y los puntos están distribuidos en secuencia. Sin embargo, no existe ninguna restricción teórica que limite la elección de puntos a múltiplos enteros, siempre y cuando sean distintos entre sí para garantizar la independencia lineal.

Lo que significa que podemos, por ejemplo, construir estos métodos utilizando conjuntos como $P = \{1h, 1\frac{1}{10}h, 1\frac{2}{10}h, 1\frac{3}{10}h, \dots\}$ o $P = \{-3h, 0, \frac{3}{2}h, 10h, \dots\}$ sin ningun problema. No obstante, esta elección no aporta beneficios significativos, ya que el error del método no variará de manera sustancial.

El error de los métodos en variables bajo nuestro control puede expresarse como $\mathcal{O}(\alpha h^\beta)$, donde h es el tamaño del paso, β es un número entero que depende de la cantidad de puntos seleccionados para construir el método, y α es un número real que depende del valor específico de los puntos elegidos. Esto se puede observar claramente en las derivaciones de los métodos presentadas en 2.3.1 y 2.3.2.

De esta expresión es evidente que los factores h y β son los más significativos en la reducción del error. El paso h tiende a ser lo más pequeño posible, y al elevarlo a un exponente entero como β , el error disminuye considerablemente. En comparación, α actúa como un multiplicador y, aunque puede influir, no suele ser más pequeño que h , por lo que su impacto en la reducción del error es marginal.

Si bien elegir conjuntos de puntos cercanos, como $P = \{1h, 1\frac{1}{100}h, 1\frac{2}{100}h \dots\}$ reducirá el valor de α , este lo hará solo de forma lineal, y además, al estar los puntos tan cercanos, podría dar paso a los ya tan discutidos errores numéricos, que pueden llegar a afectar gravemente el resultado.

A continuación veremos los resultados del mismo cálculo anterior, pero usando los conjuntos de puntos $P_k = \{0, \frac{h}{100}, \frac{2h}{100}, \dots, \frac{kh}{100}\}$ para diferencias finitas y para los métodos de paso complejo, que se construirán a partir del conjunto de puntos $P_k = \{\frac{ih}{100}, \frac{2ih}{100}, \dots, \frac{kih}{100}\}$.

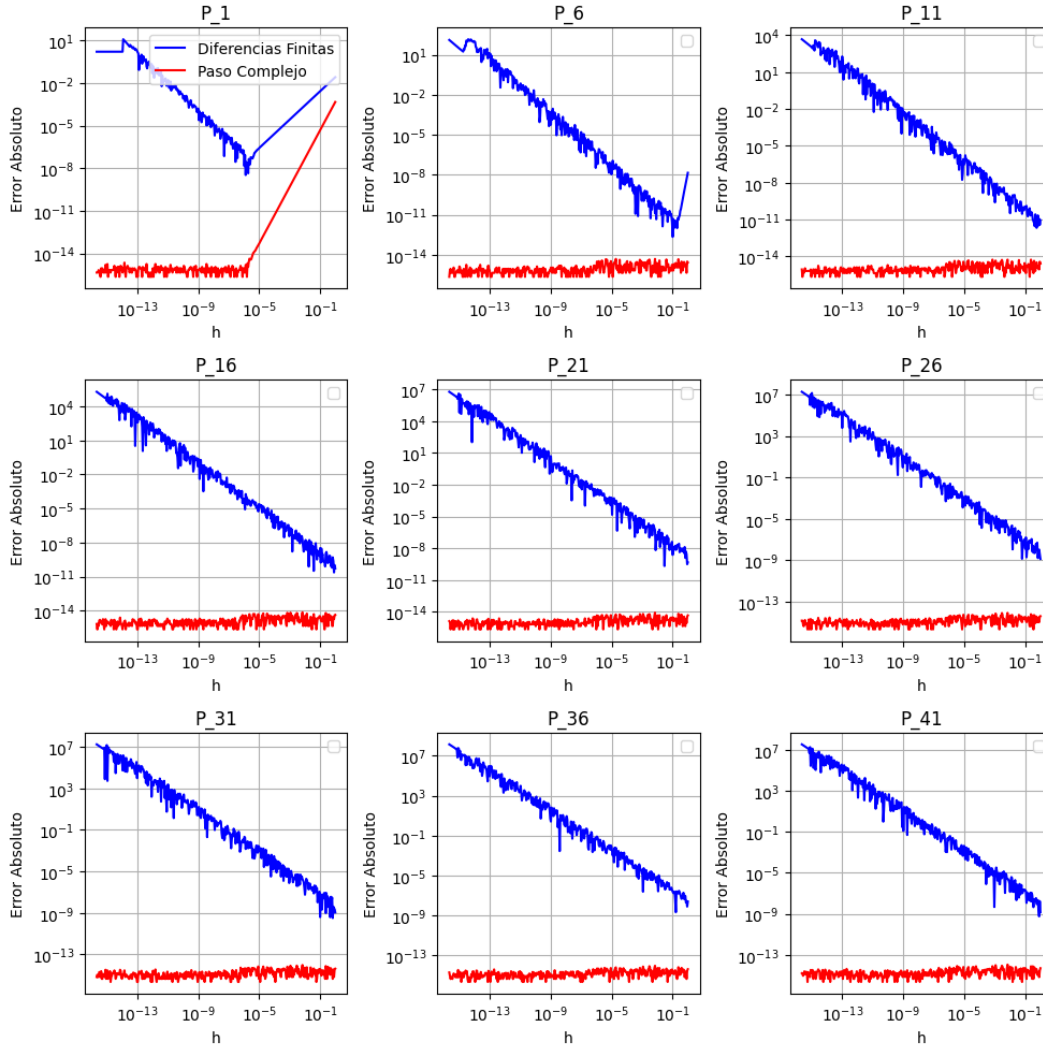


Figura 65: Evolución del error a medida que se añaden puntos decimales a las aproximaciones.

4.7.3. Derivadas de Orden Superior y Método de Paso Multicomplejo

Tanto el método de diferencias finitas como el de paso complejo pueden ser adaptados para calcular derivadas de cualquier orden, siempre que se disponga de una cantidad suficiente de puntos. No obstante, como se ha discutido previamente, esto puede dar lugar a la aparición de errores de cancelación.

Para mitigar los errores de cancelación mencionados, se puede emplear el método de paso multicomplejo descrito en 2.3.8. Este enfoque utiliza múltiples componentes imaginarias para calcular derivadas de cualquier orden, eliminando los errores de cancelación al realizar una única llamada a la función objetivo.

A continuación, analizaremos el comportamiento de los métodos al calcular derivadas de orden superior de la función:

$$\mathbf{F}(\mathbf{x}) = \left(\frac{e^{x_1}}{\cos^3(x_2) + \sin^3(x_3)}, x_1^3 - x_2^2 + x_3 - 1, x_3 \sin(x_1) \cos(x_1) \right)$$

evaluándola en el punto $\mathbf{x} = [1, 1, 1]$ y dirección $\mathbf{v} = [1, 1, 1]$. Este experimento nos permitirá observar el impacto de los errores numéricos y la estabilidad de los resultados en casos de mayor complejidad.

Para generar los métodos de diferencias finitas de orden superior, se utilizarán los conjuntos de puntos $P_k = \{0, h, 2h, \dots, kh\}$, mientras que para los métodos de paso complejo se emplearán $P_k = \{h, 2h, \dots, kh\}$. El método de paso multicomplejo, por su parte, no requiere puntos adicionales.

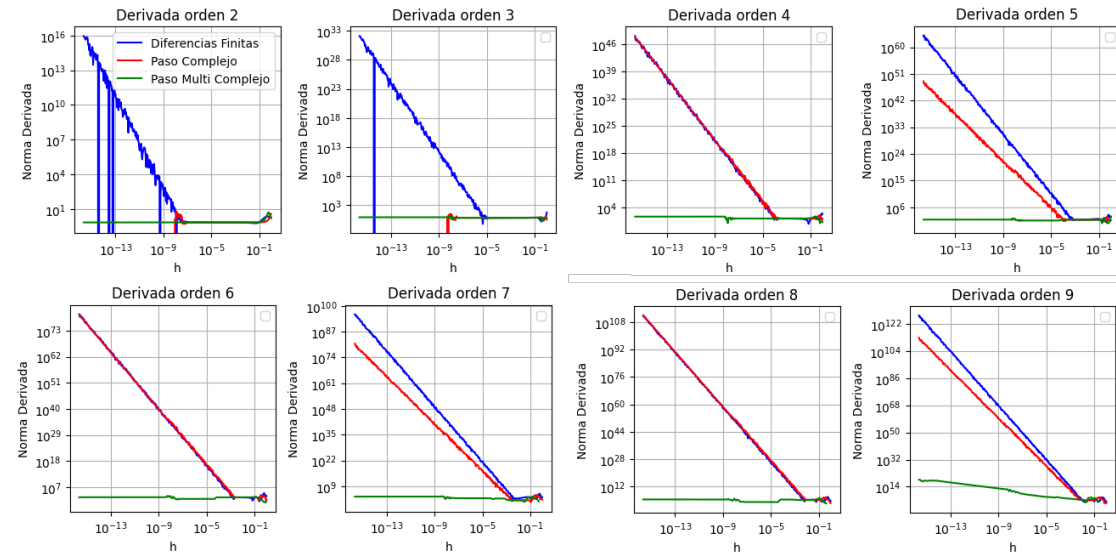


Figura 66: Norma de la derivada a medida que se incrementa el orden.

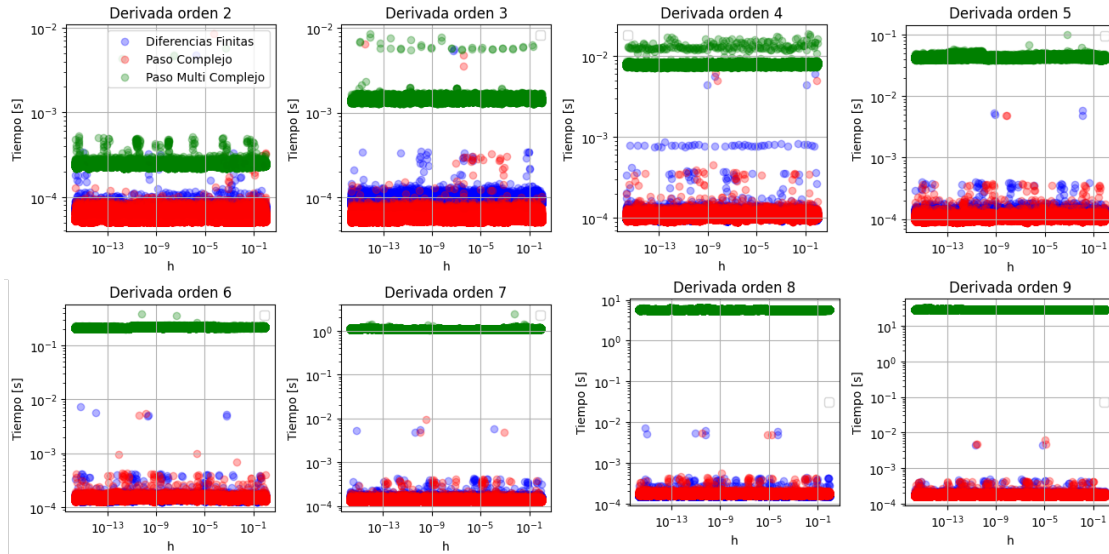


Figura 67: Tiempo de computo de la derivada a medida que se incrementa el orden.

Este método demuestra ser notablemente superior en términos de precisión, ya que es capaz de calcular derivadas de cualquier orden con alta exactitud y sin errores de cancelación. Esto lo distingue de los métodos de diferencias finitas y paso complejo, cuyos errores aumentan rápidamente en derivadas de orden elevado y a medida que el tamaño del paso disminuye.

Sin embargo, en cuanto a los tiempos de ejecución, el método multicomplejo presenta un desempeño considerablemente inferior. Puede llegar a tardar horas en calcular una sola derivada de órdenes superiores a los mostrados, especialmente si no se considera su crecimiento exponencial. Este comportamiento contrasta con los métodos clásicos, que producen resultados en fracciones de segundo. El problema radica en la explosión combinatoria de los componentes imaginarios únicos requeridos para los cálculos. Cada componente adicional implica más cálculos, lo que genera un cuello de botella difícil de mitigar.

Aunque una posible solución sería optimizar la implementación, como emplear un backend en C en lugar de Python, esto no resolvería completamente el problema fundamental del crecimiento exponencial de las combinaciones imaginarias. No obstante, el método sigue mostrando un gran desempeño en términos de precisión.

En conclusión, el método de paso multicomplejo es extremadamente útil cuando se requieren derivadas de alto orden, ya que ofrece alta precisión sin necesidad de trabajar con expresiones analíticas, como ocurre con la diferenciación automática. Esto es especialmente ventajoso cuando obtener las expresiones analíticas resulta complicado. Sin embargo, el tiempo de ejecución representa una restricción significativa debido a su crecimiento exponencial, lo que debe ser cuidadosamente considerado antes de emplear este método.

CAPÍTULO 5

CONCLUSIONES

El presente trabajo ha permitido analizar en profundidad las ventajas y limitaciones del método de derivación de paso complejo, posicionándolo como una alternativa robusta frente a métodos tradicionales como las diferencias finitas para el cálculo de derivadas numéricas. Este análisis no solo ha destacado su capacidad para mitigar errores de cancelación, sino que también ha demostrado que puede ser aplicado con éxito en la resolución de sistemas de ecuaciones no lineales.

Una de las principales conclusiones obtenidas es que el método de paso complejo supera significativamente a las diferencias finitas en términos de precisión y estabilidad numérica, especialmente en contextos donde los cálculos se ven afectados por las limitaciones inherentes a la representación en punto flotante de los sistemas computacionales. La eliminación efectiva de los errores de cancelación mediante el uso de incrementos imaginarios representa un avance significativo en la implementación de algoritmos numéricos más confiables.

En términos prácticos, la integración del método de paso complejo con esquemas iterativos como el método de Newton-Krylov ha mostrado ser una combinación prometedora, particularmente en la resolución de problemas multidimensionales y altamente no lineales. Los resultados numéricos obtenidos sugieren que esta integración no solo mejora la convergencia de los algoritmos iterativos, sino que también puede reducir el tiempo de ejecución en comparación con las estrategias convencionales.

Sin embargo, también se ha demostrado que, al extender estas estrategias clásicas mediante la incorporación de un adaptador dinámico para el tamaño del paso h , se mejora significativamente su calidad de respuesta, llegando a superar marginalmente al método integrado con el paso complejo. No obstante, esto no invalida lo desarrollado en este trabajo; más bien, pone de manifiesto que el método de paso complejo es un recurso infrautilizado, con un potencial considerable aún por explorar. Es posible que, mediante la implementación de estrategias aún no descubiertas, este método pueda superar de manera consistente el estado del arte actual.

Por otro lado, se han identificado ciertas limitaciones que podrían ser objeto de estudio futuro. Por ejemplo, la necesidad de que las funciones sean holomorfas en el ámbito de los números complejos podría restringir el conjunto de aplicaciones directas del método. Sin embargo, durante el desarrollo de este trabajo no se presentaron inconvenientes relacionados con esta condición.

Otra área que merece ser explorada es la de las representaciones numéricas. En este trabajo se utilizaron las implementaciones nativas de Python y Julia; sin embargo, un enfoque dedicado exclusivamente a este método podría permitir el desarrollo de algoritmos de bajo nivel y hasta de hardware especializado en este tipo de álgebra, lo que potencialmente incrementaría de manera significativa el rendimiento del método.

Además, se ha explorado de manera superficial el método de paso multicomplejo, una técnica relativamente reciente con apenas 12 años desde su descubrimiento en 2012. Esto lo posiciona como un método todavía en desarrollo y menos conocido en comparación con sus contrapartes más establecidas. No obstante, su potencial es significativo, y es posible que, con investigaciones adicionales, este método llegue a consolidarse como una herramienta estándar en el análisis numérico.

En conclusión, este trabajo no solo valida la relevancia del método de derivación de paso complejo como una herramienta eficaz para abordar problemas de cálculo numérico, sino que también establece un marco sólido para futuras investigaciones en este campo. Los avances logrados en la mitigación de errores numéricos y en la mejora de la estabilidad computacional proporcionan una base sólida para aplicaciones más precisas y eficientes en áreas como la simulación científica, la optimización y el aprendizaje automático. En última instancia, se espera que los resultados aquí presentados impulsen el desarrollo de técnicas numéricas más robustas y adaptables para enfrentar los desafíos futuros.

Todos los algoritmos desarrollados durante este trabajo han sido publicados de forma pública y permanente en GitHub, con el objetivo de contribuir a este propósito y fomentar la colaboración en la comunidad científica y técnica.

ANEXOS

5.1. README del Repositorio de Experimentos

El repositorio que contiene los códigos fuente, resultados y otros elementos relacionados con los experimentos puede encontrarse en el siguiente enlace: Repositorio de Experimentos.

A continuación, se incluye el contenido del archivo README del repositorio:

En este repositorio se encuentran la mayoría de los experimentos realizados en el capítulo 4 de la memoria. Cabe destacar que no todos los experimentos incluidos aquí han sido expuestos en la misma.

Las carpetas están organizadas de la siguiente manera:

- Códigos de experimentos: Contiene los códigos fuente de los scripts que ejecutan los experimentos y almacenan los datos necesarios para su posterior graficación. Estos scripts son invocados por los archivos .job.
- JOBS de experimentos: Incluye los archivos .job, utilizados para asignar tareas al NHLPC. Estos archivos contienen, entre otras cosas, las especificaciones del entorno computacional que se utilizará y el comando para ejecutar los scripts junto con sus parámetros.
- Outputs de experimentos: Contiene los archivos de salida de los experimentos, así como sus respectivos archivos de registro (.err.out). Existen tres tipos principales de archivos: 1. Residuo o norma: Incluye la norma de la solución final del experimento. 2. Tiempo: Indica la cantidad de segundos que tardó en completarse el experimento. 3. Convergencia: Contiene el número de iteraciones de Newton requeridas para converger. (Este último tipo no fue utilizado).
- Seeded Data: Almacena las matrices y vectores aleatorios generados para los experimentos. Estos datos están guardados en disco para evitar inconsistencias debido a variaciones en la generación de números aleatorios.

5.2. Implementación de diferencias finitas en Scipy

```
1 def matvec(self, v):
2     nv = norm(v)
3     if nv == 0:
4         return 0 * v
5     sc = self.omega / nv
6     r = (self.func(self.x0 + sc * v) - self.f0) / sc
7     if not np.all(np.isfinite(r)) and np.all(np.isfinite(v)):
8         raise ValueError('Function returned non-finite results')
9     return r
```

5.3. Implementación de paso complejo en Scipy

```
1 def matvec(self, v):
2     nv = norm(v)
3     if nv == 0:
4         return 0*v
5     sc = self.rdiff / nv
6     r = np.imag(self.func(self.x0 + sc * v * 1j)) / sc
7     if not np.all(np.isfinite(r)) and np.all(np.isfinite(v)):
8         raise ValueError('Function returned non-finite results')
9     return r
```

5.4. Implementación de diferencias finitas en Julia

```
1 function J_finite_diff(x, rdiff_vec)
2     Jv = function (v)
3         nv = norm(v)
4         if nv == 0
5             return 0 * v
6         end
7         h = rdiff_vec[2]
8         sc = h / nv
9         dr = (F(x .+ sc * v, nothing) .- F(x, nothing)) ./sc
10        if !all(isfinite, dr) && all(isfinite, v)
11            throw(ArgumentError("Function returned non-finite
12                                results"))
13        end
14        return dr
15    end
16 end
```

5.5. Implementación de paso complejo en Julia

```
1 function J_csa_diff(x, rdiff_vec)
2     Jv = function (v)
3         nv = norm(v)
4         if nv == 0
5             return 0 * v
6         end
7         h = rdiff_vec[2]
8         sc = h / nv
9         args = convert(Vector{data_type}, x .+ 1im * sc * v)
10        dr = imag(F(args, nothing)) ./sc
11        if !all(isfinite, dr) && all(isfinite, v)
12            throw(ArgumentError("Function returned non-finite
13                                results"))
14        end
15        return dr
16    end
17 end
return Jv
```

5.6. Problema Sparse I

Algorithm 1 Función del problema Sparse I

```
1: function quadratic_function( $x$ )
2:    $n \leftarrow \text{length}(x)$ 
3:    $f \leftarrow \text{array\_zeros}(n)$ 
4:    $f_0(x_i, x_{i+1}, x_{i+2}, x_{i+3}) \leftarrow x_i + 3 \cdot x_{i+1} \cdot (x_{i+2} - 1) + x_{i+3}^2 - 1$ 
5:    $f_1(x_i, x_{i+1}, x_{i+2}, x_{i+3}) \leftarrow (x_i + x_{i+1})^2 + (x_{i+2} - 1)^2 - x_{i+3} - 3$ 
6:    $f_2(x_i, x_{i+1}, x_{i+2}, x_{i+3}) \leftarrow x_i \cdot x_{i+1} - x_{i+2} \cdot x_{i+3}$ 
7:    $f_3(x_i, x_{i+1}, x_{i+2}, x_{i+3}) \leftarrow 2 \cdot x_i \cdot x_{i+2} + x_{i+1} \cdot x_{i+3} - 3$ 
8:    $f_4(x_i, x_{i+1}, x_{i+2}, x_{i+3}) \leftarrow (x_i + x_{i+1} + x_{i+2} + x_{i+3})^2 + (x_i - 1)^2$ 
9:    $f_5(x_i, x_{i+1}, x_{i+2}, x_{i+3}) \leftarrow x_i \cdot x_{i+1} \cdot x_{i+2} \cdot x_{i+3} + (x_{i+3} - 1)^2 - 1$ 
10:   $fs \leftarrow \{f_0, f_1, f_2, f_3, f_4, f_5\}$ 
11:   $counter \leftarrow 0$ 
12:  for  $i \leftarrow 1$  to  $n - 1$  do
13:     $idx \leftarrow 6 \cdot (i - 1)$ 
14:     $x_i, x_{i+1}, x_{i+2}, x_{i+3} \leftarrow x[i - 1], x[i], x[i + 1], x[i + 2]$ 
15:    for  $j \leftarrow 0$  to 5 do
16:       $f[idx + j] \leftarrow fs[j](x_i, x_{i+1}, x_{i+2}, x_{i+3})$ 
17:       $counter \leftarrow counter + 1$ 
18:      if  $counter \geq n$  then
19:        return  $f$ 
20:      end if
21:    end for
22:  end for
23: end function
```

5.7. Problema Sparse II

Algorithm 2 Función del problema Sparse II

```
1: function diagonal_function( $x$ )
2:    $n \leftarrow \text{length}(x)$ 
3:    $f \leftarrow \text{array\_zeros}(n)$ 
4:    $f_0(x_i, x_{i+1}, x_{i+2}) \leftarrow 0,6 \cdot x_{i+2} + 1,6 \cdot x_{i+2}^3 - 7,2 \cdot x_{i+1}^2 + 9,6 \cdot x_{i+1} - 4,8$ 
5:    $f_1(x_i, x_{i+1}, x_{i+2}) \leftarrow 0,48 \cdot x_{i+2} - 0,72 \cdot x_{i+1}^3 + 3,24 \cdot x_{i+1}^2 - 4,32 \cdot x_{i+1} - x_i + 0,2 \cdot x_i^3 + 2,16$ 
6:    $f_2(x_i, x_{i+1}, x_{i+2}) \leftarrow 1,25 \cdot x_i + 0,25 \cdot x_i^3$ 
7:    $fs \leftarrow \{f_0, f_1, f_2\}$ 
8:    $counter \leftarrow 0$ 
9:   for  $i \leftarrow 1$  to  $\lfloor n/3 \rfloor$  do
10:     $idx \leftarrow 3 \cdot (i - 1)$ 
11:     $x_i, x_{i+1}, x_{i+2} \leftarrow x[i + 2], x[i + 1], x[i]$ 
12:    for  $j \leftarrow 0$  to  $2$  do
13:       $f[idx + j] \leftarrow fs[j](x_i, x_{i+1}, x_{i+2})$ 
14:       $counter \leftarrow counter + 1$ 
15:      if  $counter \geq n$  then
16:        return  $f$ 
17:      end if
18:    end for
19:  end for
20:  return  $f$ 
21: end function
```

5.8. Problema Dense I

Algorithm 3 Función del problema Dense I

```
1: function dense_problem_I( $x, p$ )
2:    $n \leftarrow \text{lenght}(x)$ 
3:    $f \leftarrow \text{array\_zeros}(n)$ 
4:    $m \leftarrow n \cdot (p + 1)$ 
5:    $start \leftarrow \frac{m-n}{2}$ 
6:    $end \leftarrow start + n$ 
7:    $sumatory \leftarrow \sum \cos(x)^{p+1}$ 
8:   for  $i \leftarrow 0$  to  $n - 1$  do
9:      $k \leftarrow n \cdot p + i$ 
10:    if  $start \leq k < end$  then
11:       $idx \leftarrow k - start$ 
12:       $f[idx] \leftarrow n + k \cdot (1 - \cos(x[i])) - \sin(x[i]) - sumatory$ 
13:    end if
14:  end for
15:  return  $f$ 
16: end function
```

5.9. Problema Dense II

Algorithm 4 Función del problema Dense II

```
1: function dense_problem_II( $x, p$ )
2:    $n \leftarrow \text{length}(x)$ 
3:    $f \leftarrow \text{array\_zeros}(n)$ 
4:    $m \leftarrow n \cdot (p + 1)$ 
5:    $start \leftarrow \frac{m-n}{2}$ 
6:    $end \leftarrow start + n$ 
7:    $sum\_x\_squared \leftarrow \sum x^2 + 1$ 
8:   for  $i \leftarrow 0$  to  $n - 1$  do
9:      $k \leftarrow n \cdot p + i$ 
10:    if  $start \leq k < end$  then
11:       $idx \leftarrow k - start$ 
12:       $f[idx] \leftarrow x[i]^{p+1} \cdot \log(sum\_x\_squared) + x[i]$ 
13:    end if
14:  end for
15:  return  $f$ 
16: end function
```

5.10. Gráficos adicionales, producto jacobiano-vector

Problema Uniform

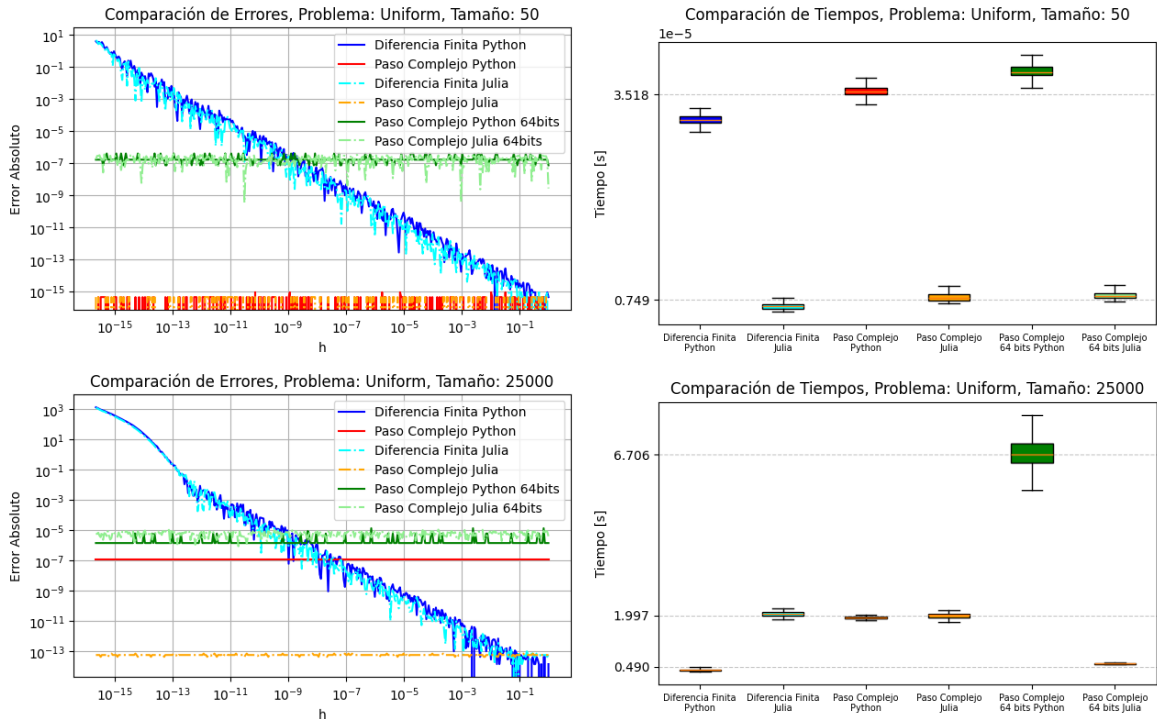


Figura 68: Error absoluto y tiempo de computo para el problema Uniform.

Problema Sparse I

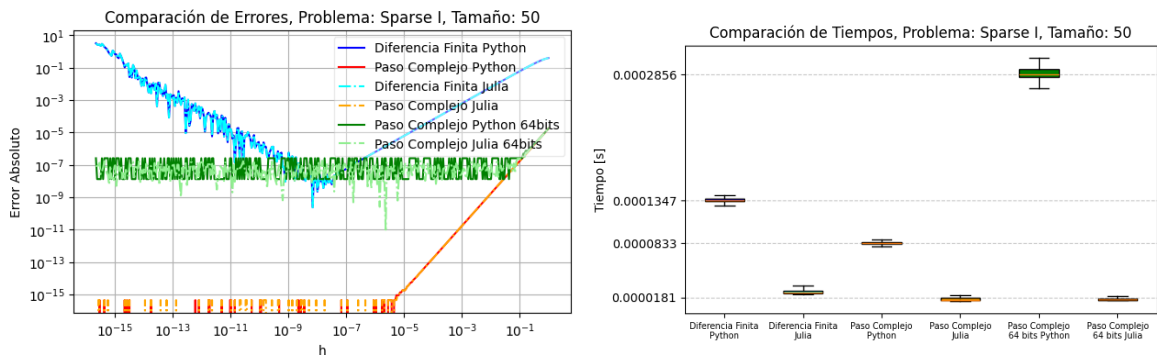


Figura 69: Error absoluto y tiempo de computo para el problema Sparse I.

Problema Sparse II

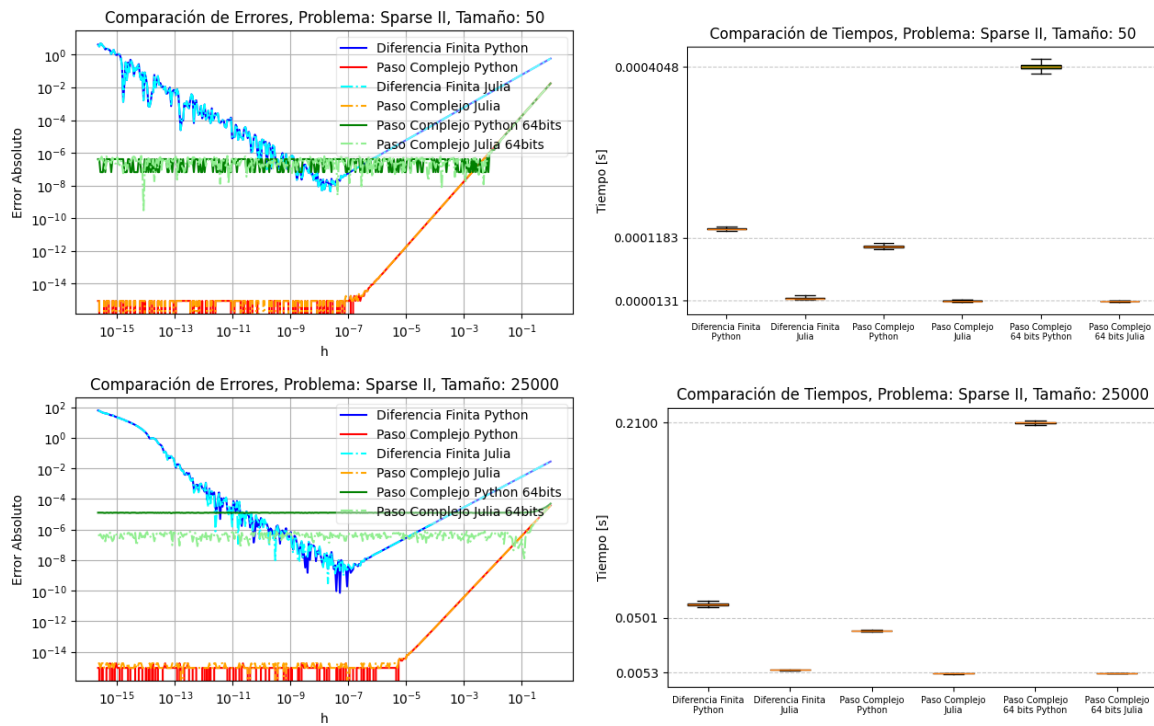


Figura 70: Error absoluto y tiempo de computo para el problema Sparse II.

Problema Dense I

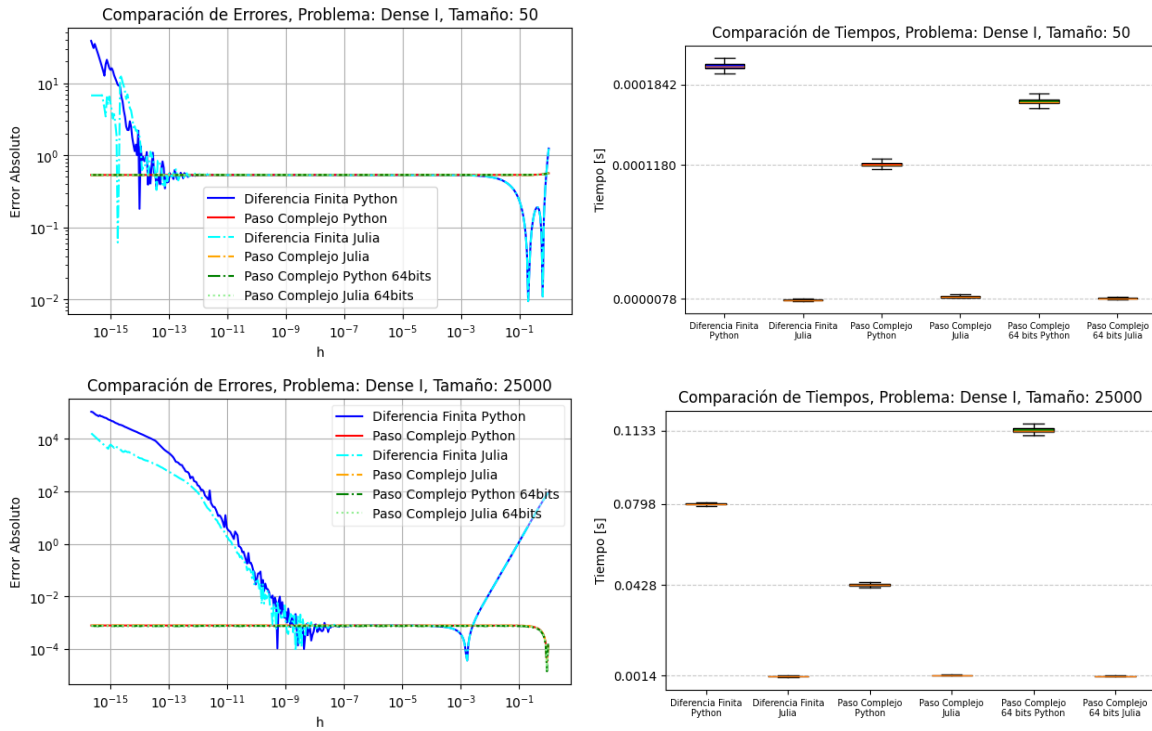


Figura 71: Error absoluto y tiempo de computo para el problema Dense I.

Problema Dense II

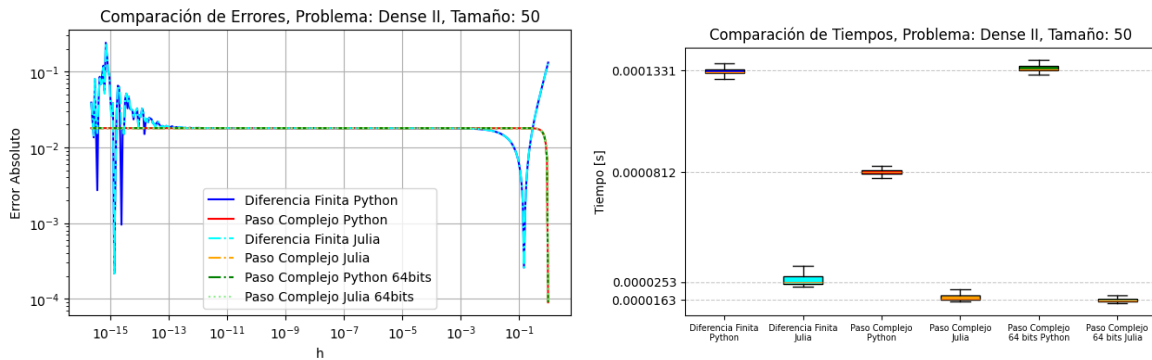


Figura 72: Error absoluto y tiempo de computo para el problema Dense II.

5.11. Gráficos adicionales, ortonormalidad espacio de Krylov

Problema Uniform

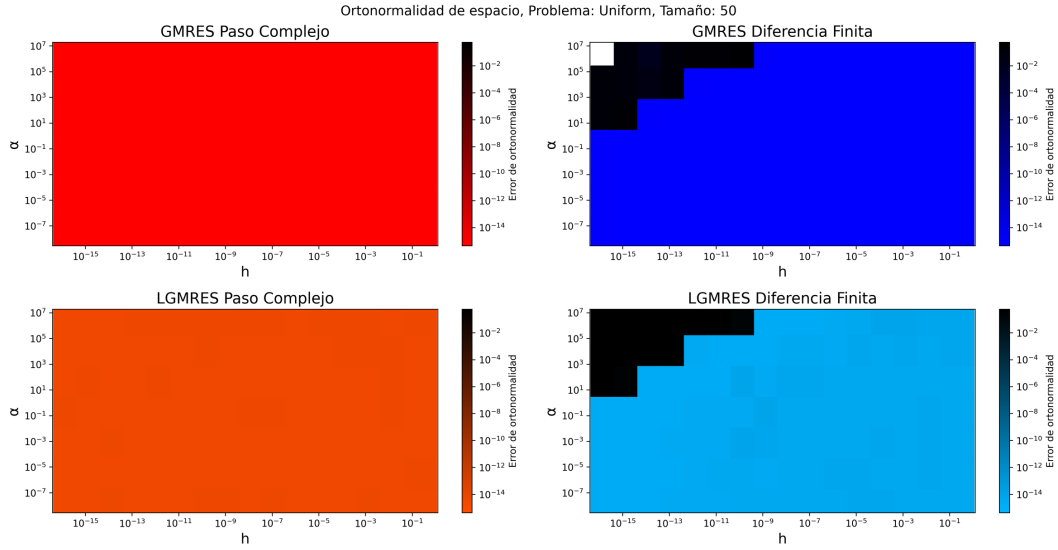


Figura 73: Ortonormalidad del espacio, problema Uniform $n = 50$.

Problema Sparse I

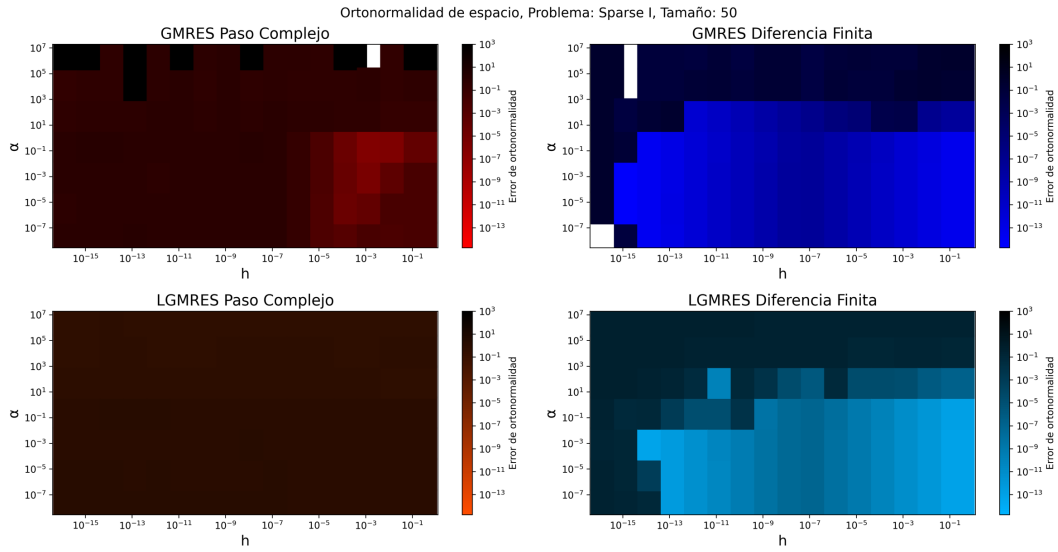


Figura 74: Ortonormalidad del espacio, problema Sparse I $n = 50$.

Problema Sparse II

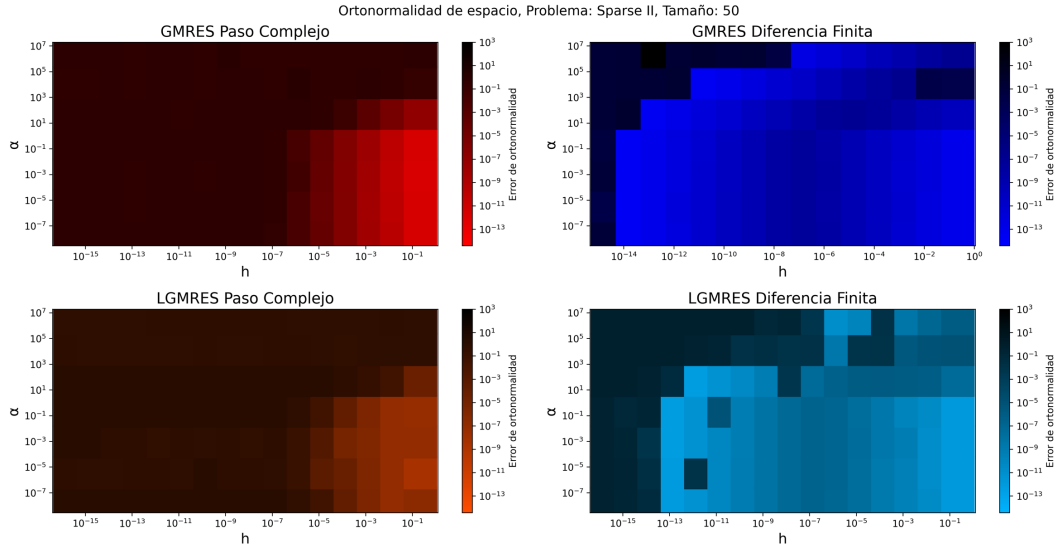


Figura 75: Ortonormalidad del espacio, problema Sparse II $n = 50$.

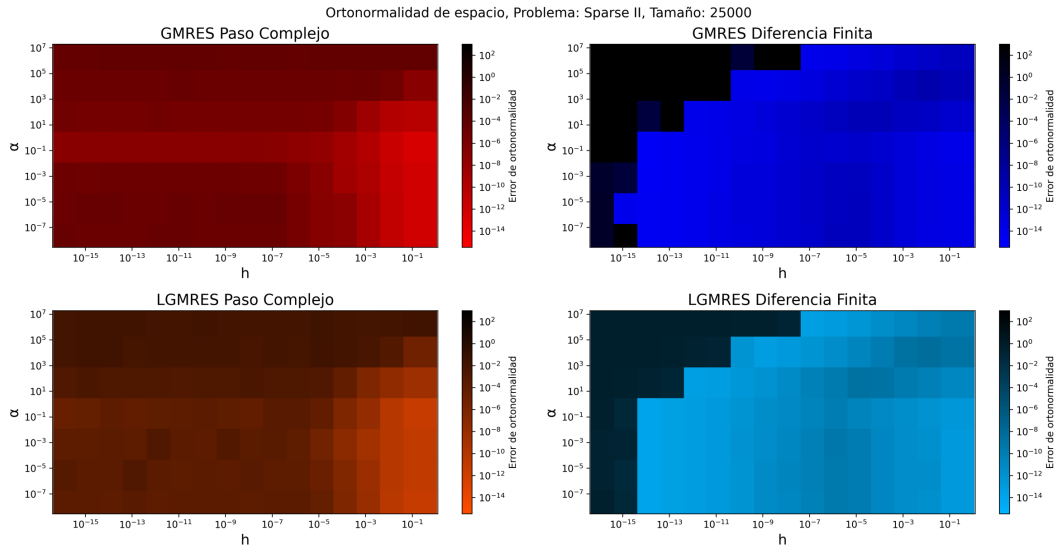


Figura 76: Ortonormalidad del espacio, problema Sparse II $n = 25000$.

Problema Dense I

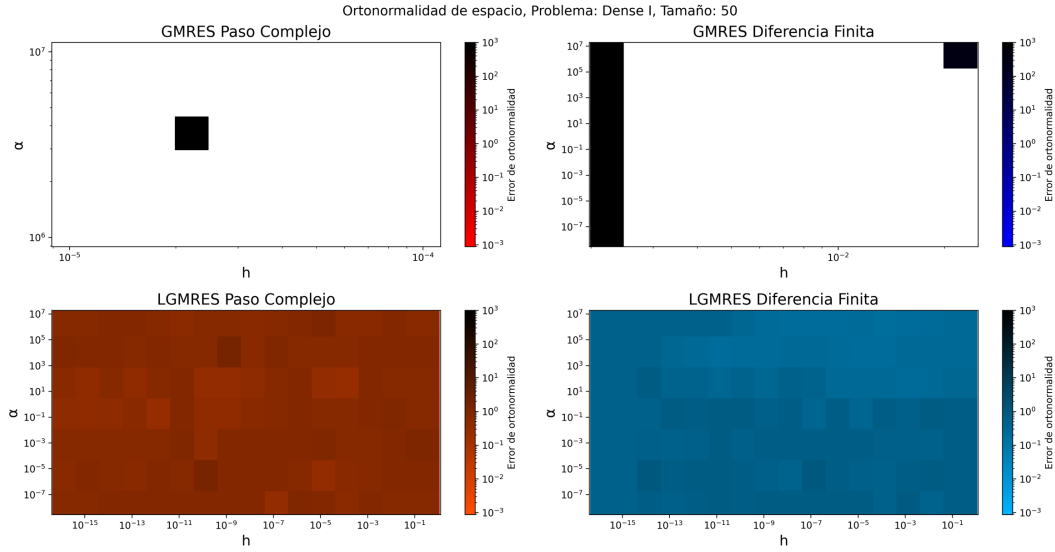


Figura 77: Ortonormalidad del espacio, problema Dense I $n = 50$.

Problema Dense II

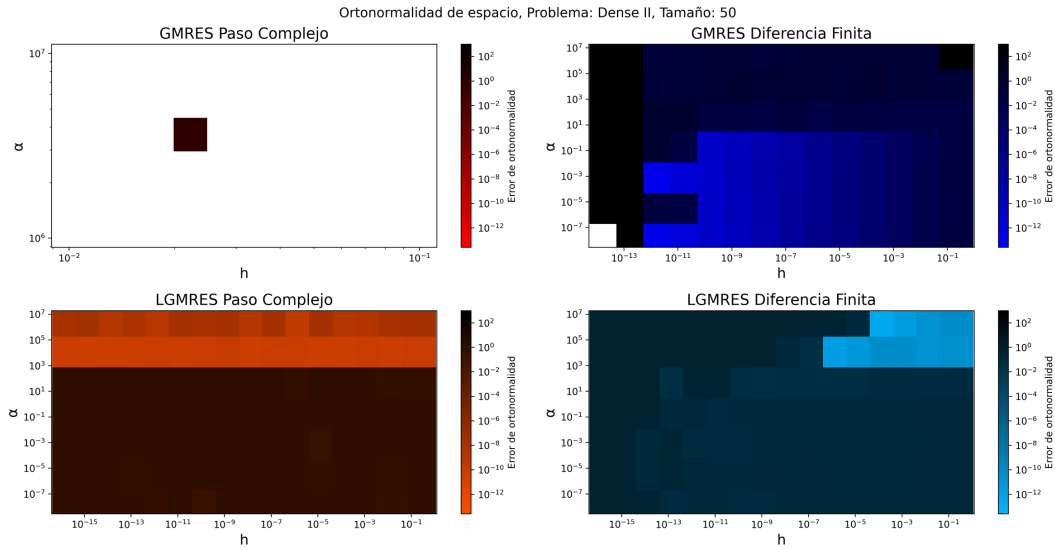


Figura 78: Ortonormalidad del espacio, problema Dense II $n = 50$.

5.12. Gráficos adicionales, un paso de Newton-Krylov

Problema Uniform

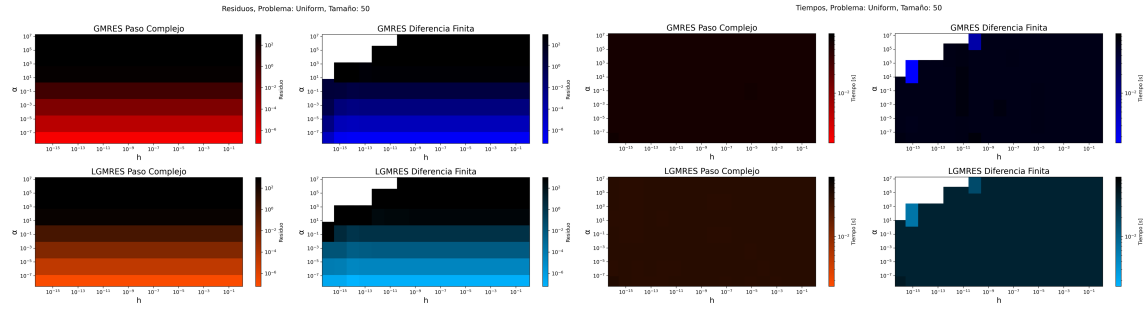


Figura 79: Error de un paso de Newton, Figura 80: Tiempo de un paso de Newton, prob. Uniform $n = 50$.

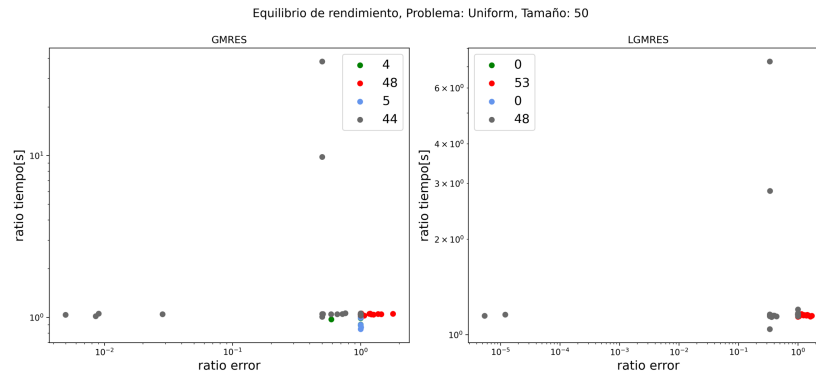


Figura 81: Equilibrio de rendimiento de un paso de Newton, prob. Uniform $n = 50$.

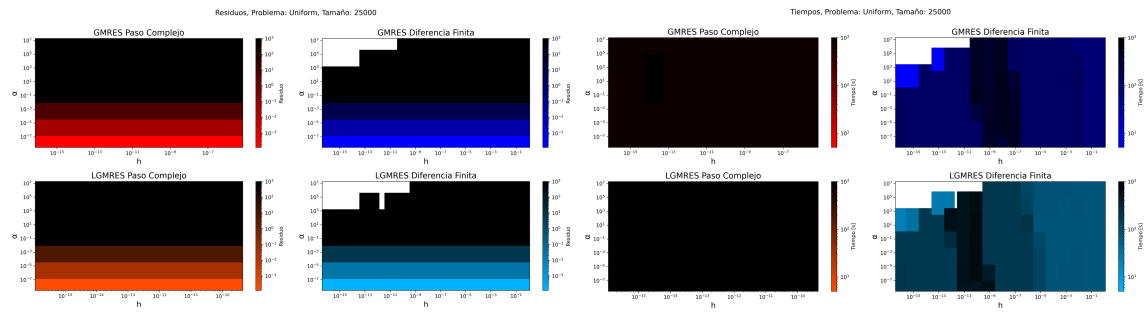


Figura 82: Error de un paso de Newton, Figura 83: Tiempo de un paso de Newton, prob. Uniform $n = 25000$.

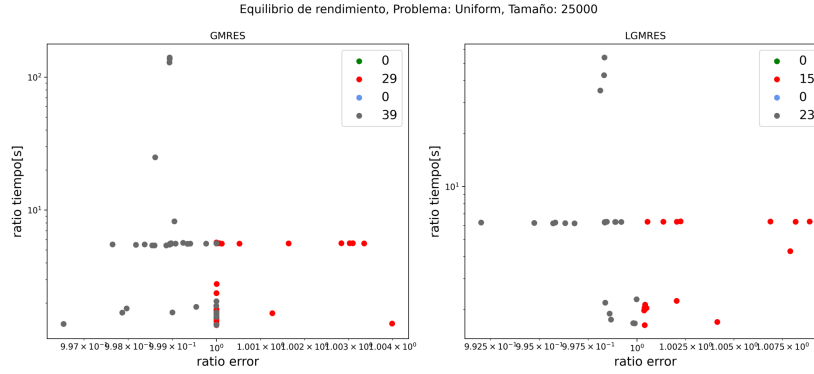


Figura 84: Equilibrio de rendimiento de un paso de Newton, prob. Uniform $n = 25000$.

Problema Sparse II

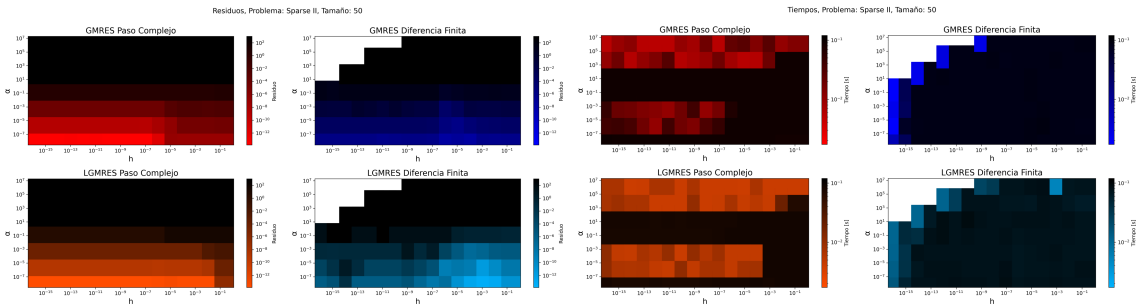


Figura 85: Error de un paso de Newton, Figura 86: Tiempo de un paso de Newton, prob. Sparse II $n = 50$.

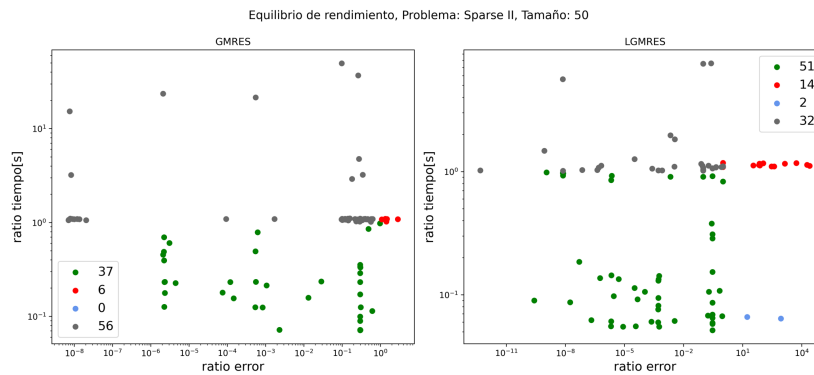


Figura 87: Equilibrio de rendimiento de un paso de Newton, prob. Sparse II $n = 50$.

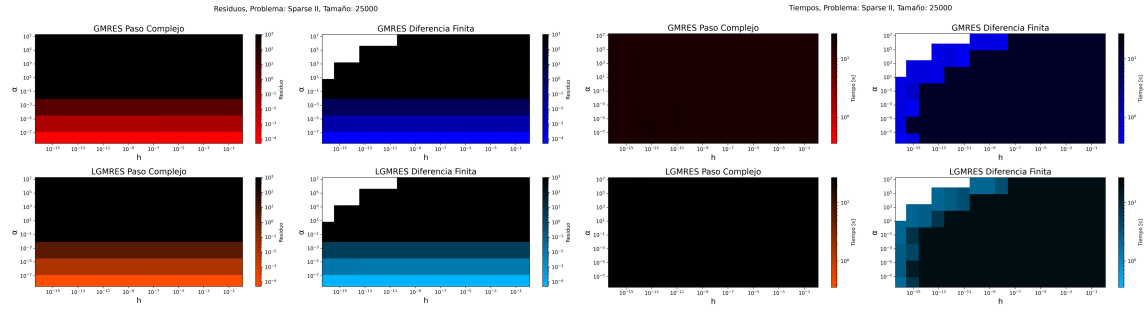


Figura 88: Error de un paso de Newton, Figura 89: Tiempo de un paso de Newton, prob. Sparse II $n = 25000$.

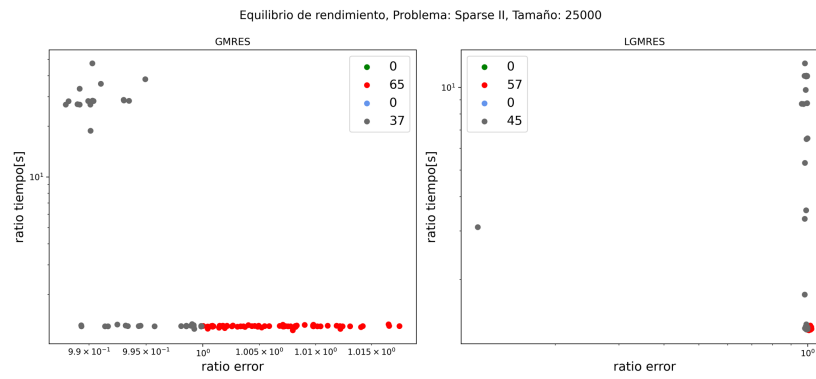


Figura 90: Equilibrio de rendimiento de un paso de Newton, prob. Sparse II $n = 25000$.

Problema Densa I

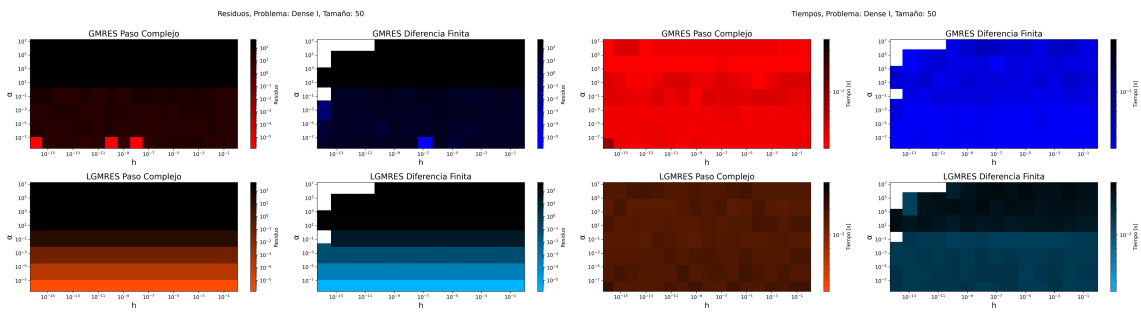


Figura 91: Error de un paso de Newton, Figura 92: Tiempo de un paso de Newton, prob. Dense I $n = 50$.

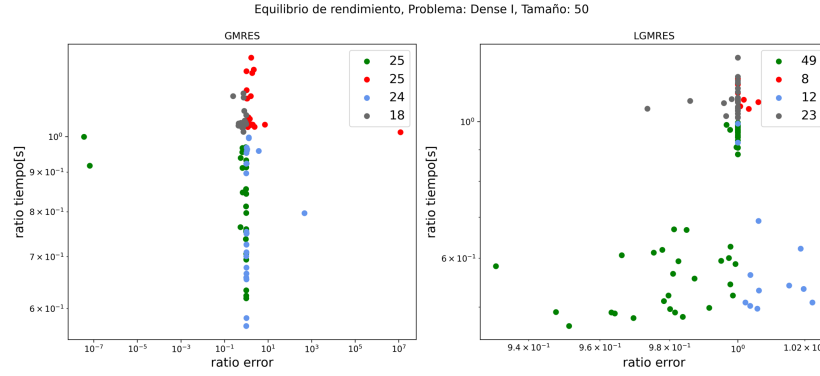


Figura 93: Equilibrio de rendimiento de un paso de Newton, prob. Dense I $n = 50$.

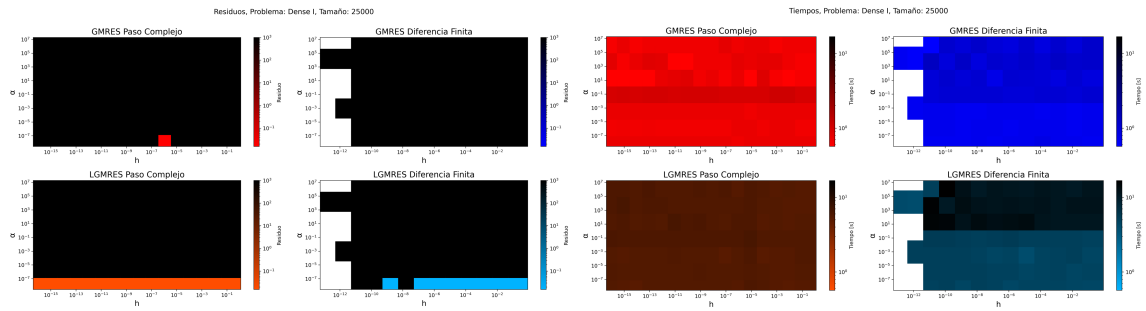


Figura 94: Error de un paso de Newton, Figura 95: Tiempo de un paso de Newton, prob. Dense I $n = 25000$.

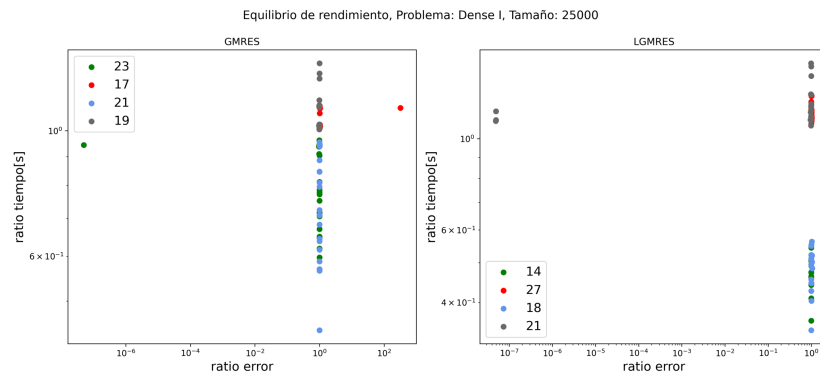


Figura 96: Equilibrio de rendimiento de un paso de Newton, prob. Dense I $n = 25000$.

5.13. Gráficos adicionales, adaptadores

Problema Sparse I

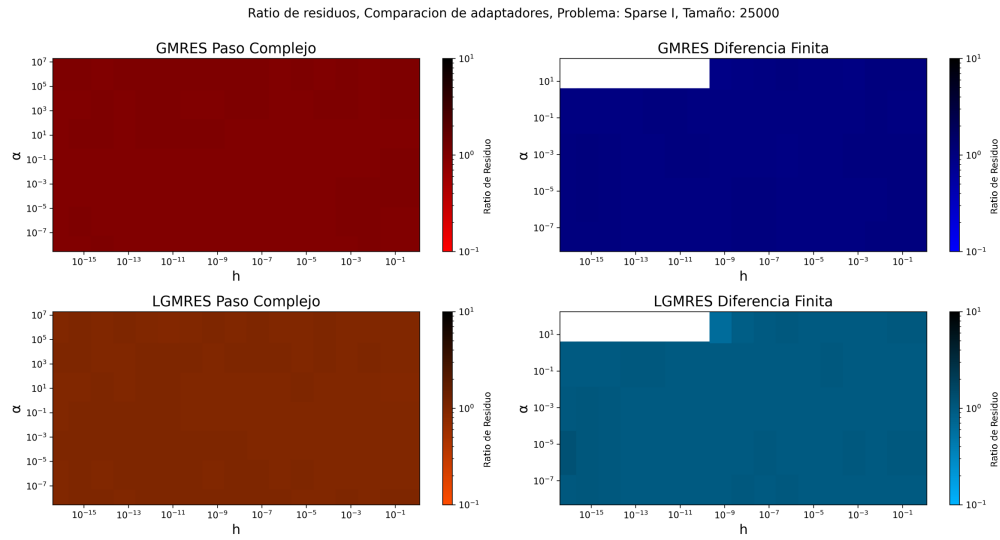


Figura 97: Ratio de error, Sparse I con adaptadores.

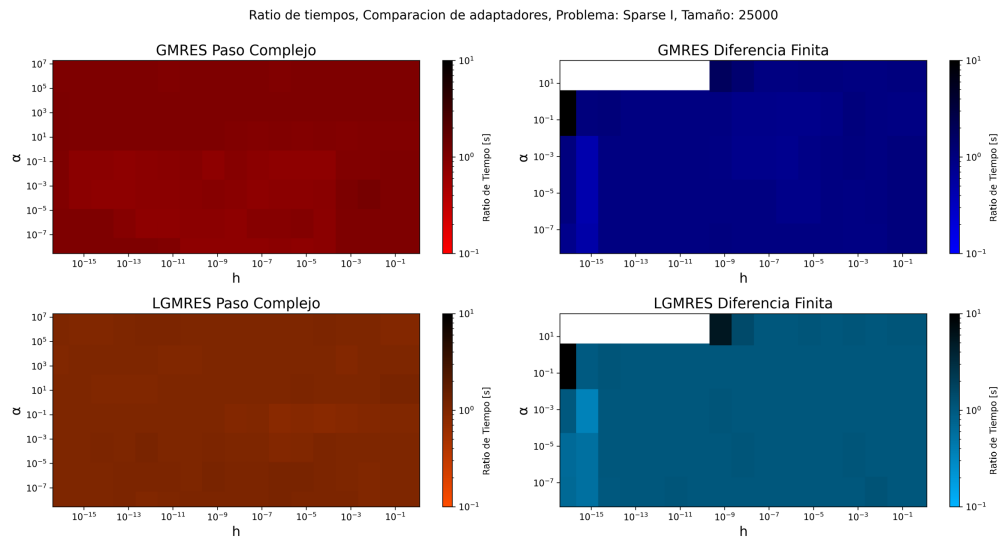


Figura 98: Ratio de tiempo, Sparse I con adaptadores.

Problema DENSE II

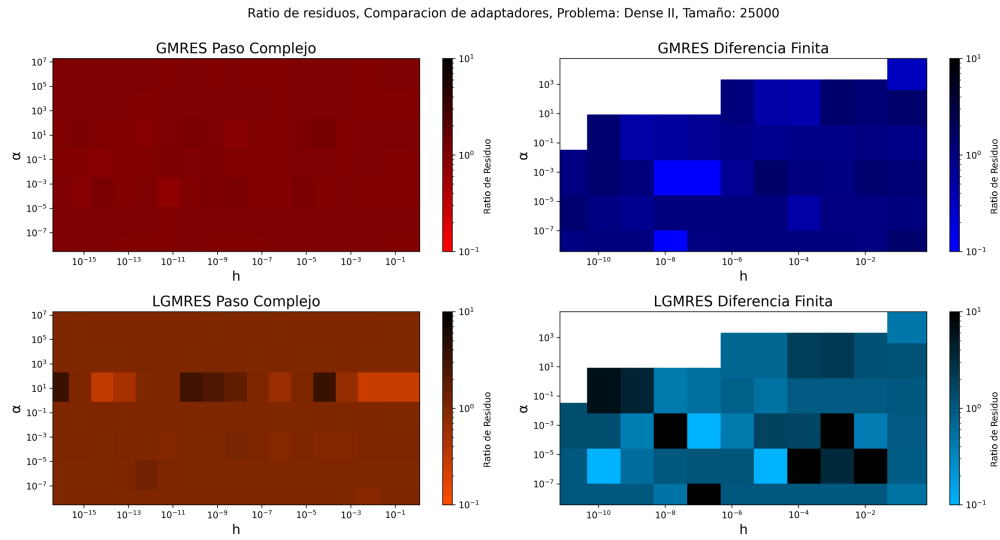


Figura 99: Ratio de error, Dense II con adaptadores.

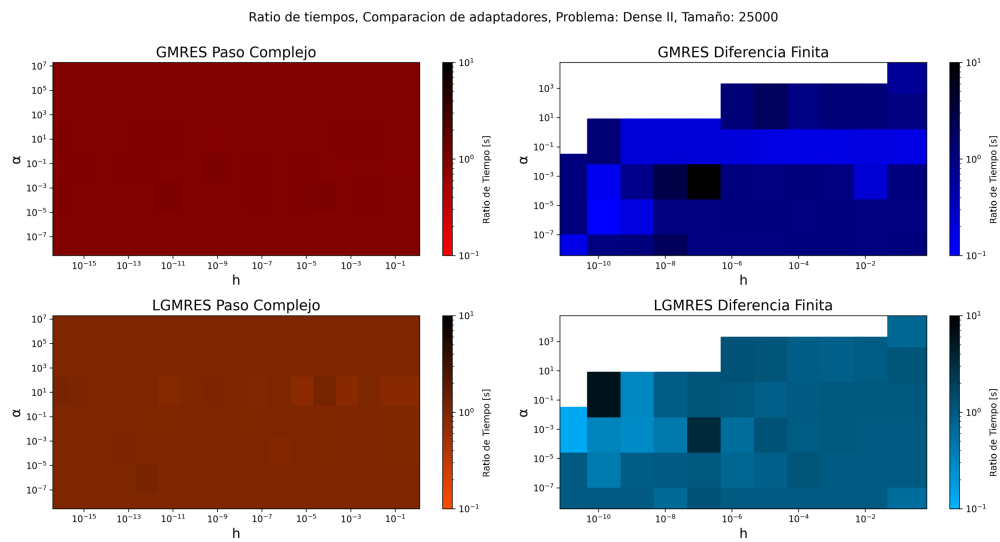


Figura 100: Ratio de tiempo, Dense II con adaptadores.

5.14. Gráficos adicionales, Newton-Krylov completo

Problema Sparse II

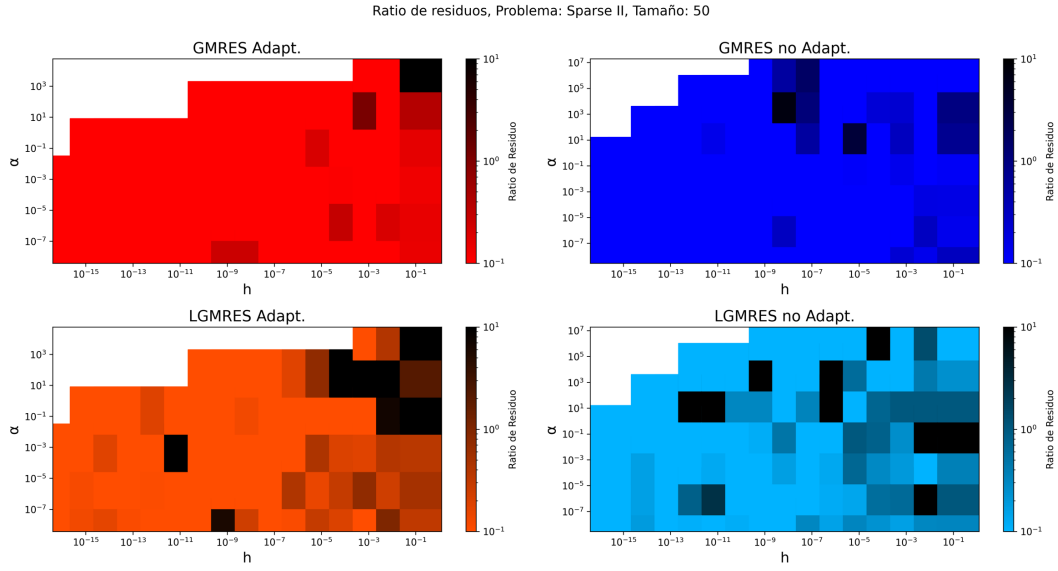


Figura 101: Ratio error, problema Sparse II, $n = 50$.

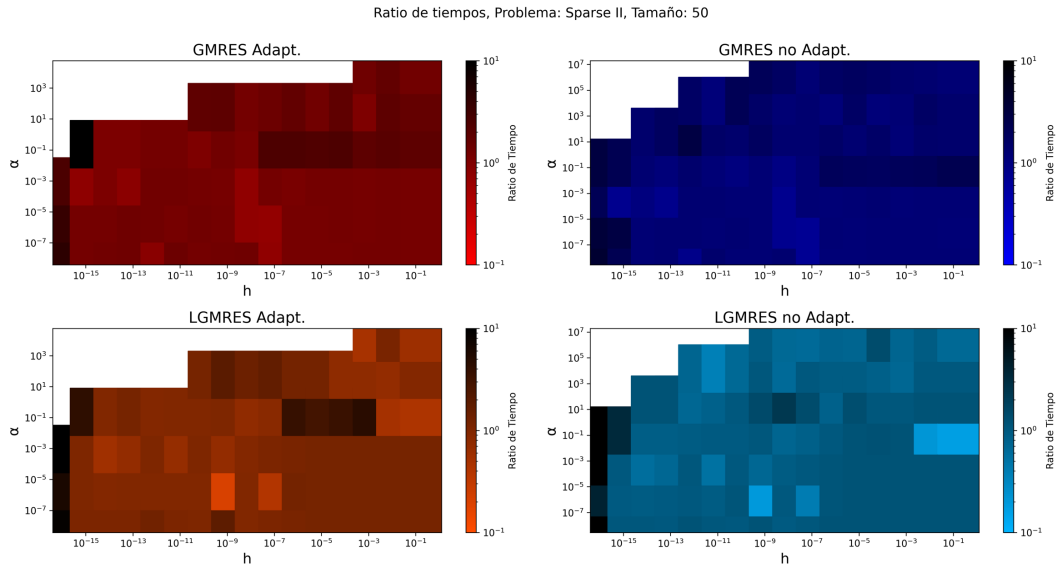


Figura 102: Ratio tiempo, problema Sparse II, $n = 50$.

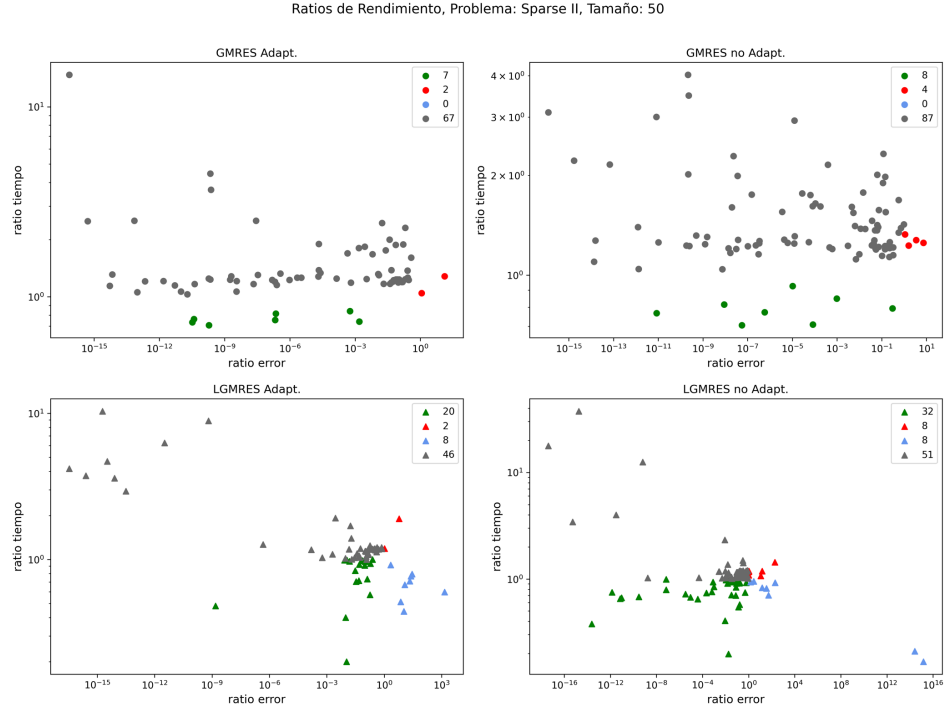


Figura 103: Equilibrio de rendimiento, problema Sparse II, $n = 50$.

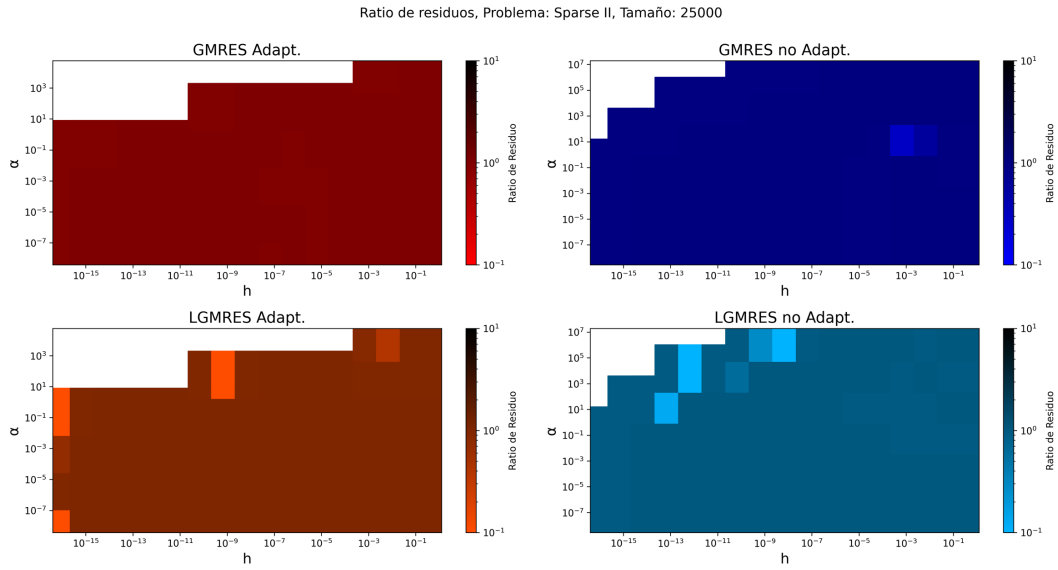


Figura 104: Ratio error, problema Sparse II, $n = 25000$.

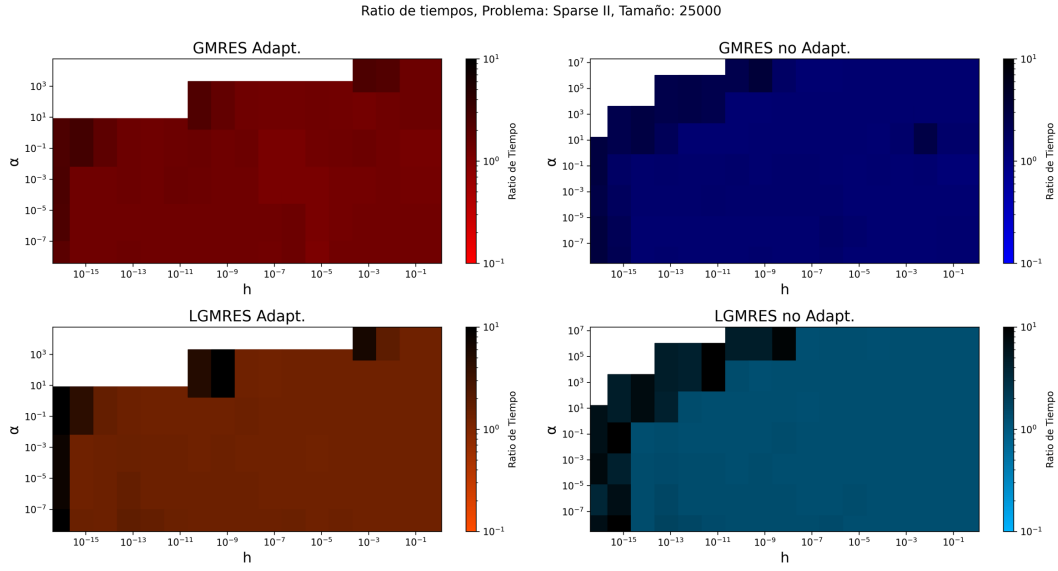


Figura 105: Ratio tiempo, problema Sparse II, $n = 25000$.

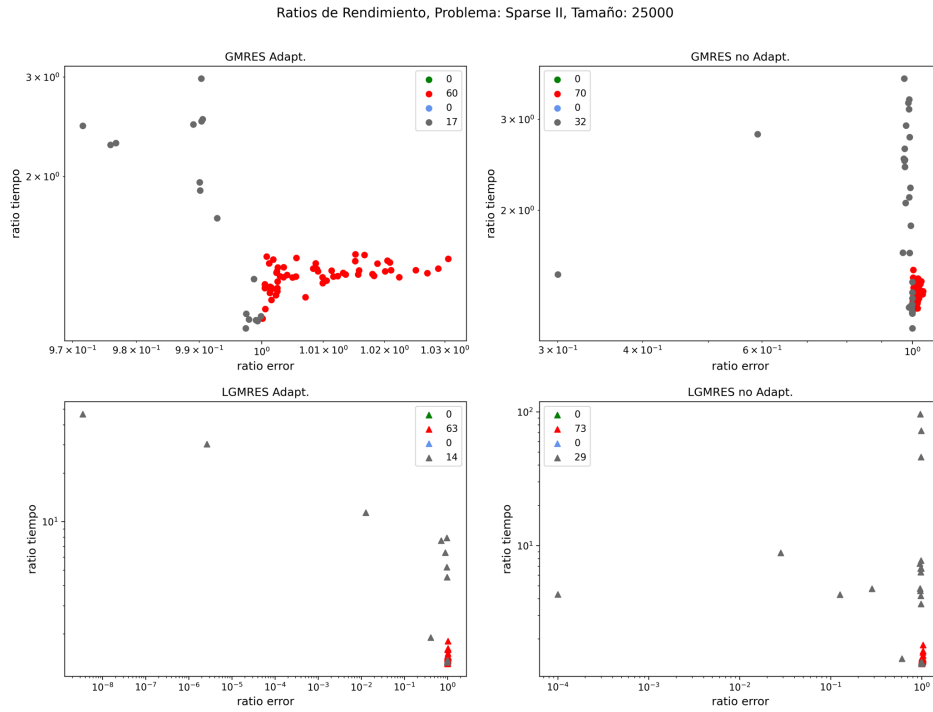


Figura 106: Equilibrio de rendimiento, problema Sparse II, $n = 25000$.

Problema Dense I

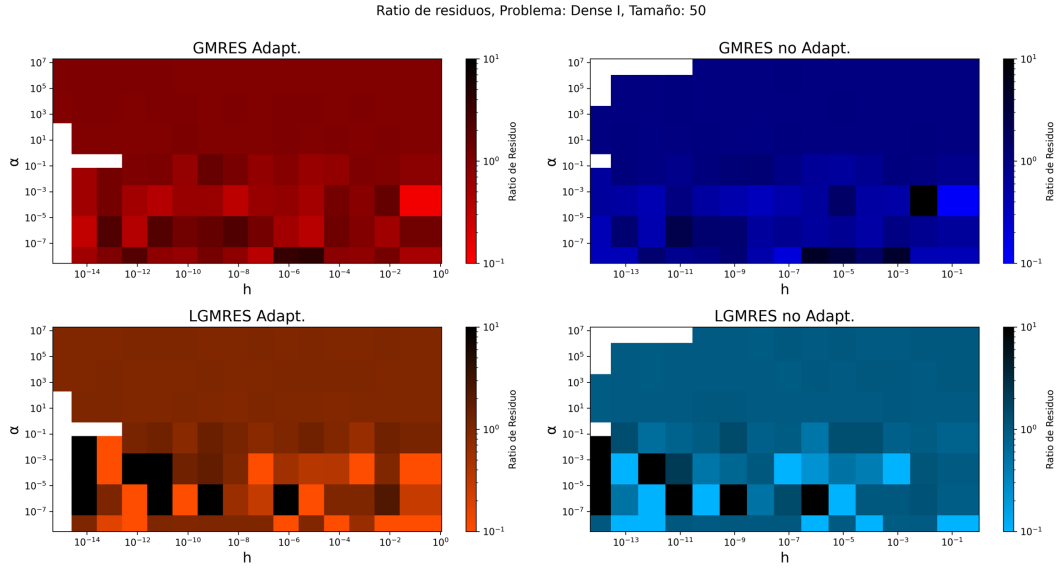


Figura 107: Ratio error, problema Dense I, $n = 50$.

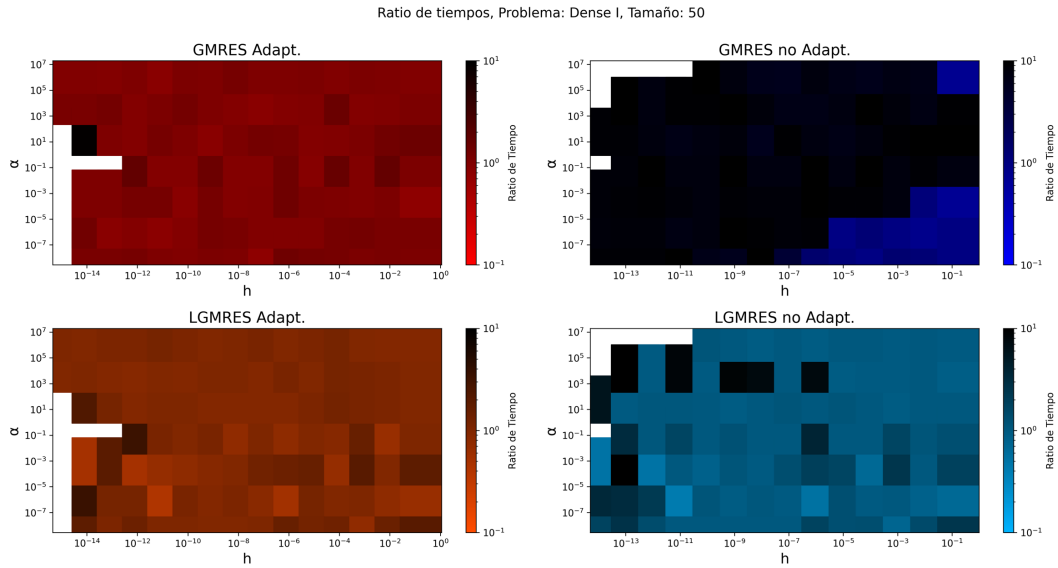


Figura 108: Ratio tiempo, problema Dense I, $n = 50$.

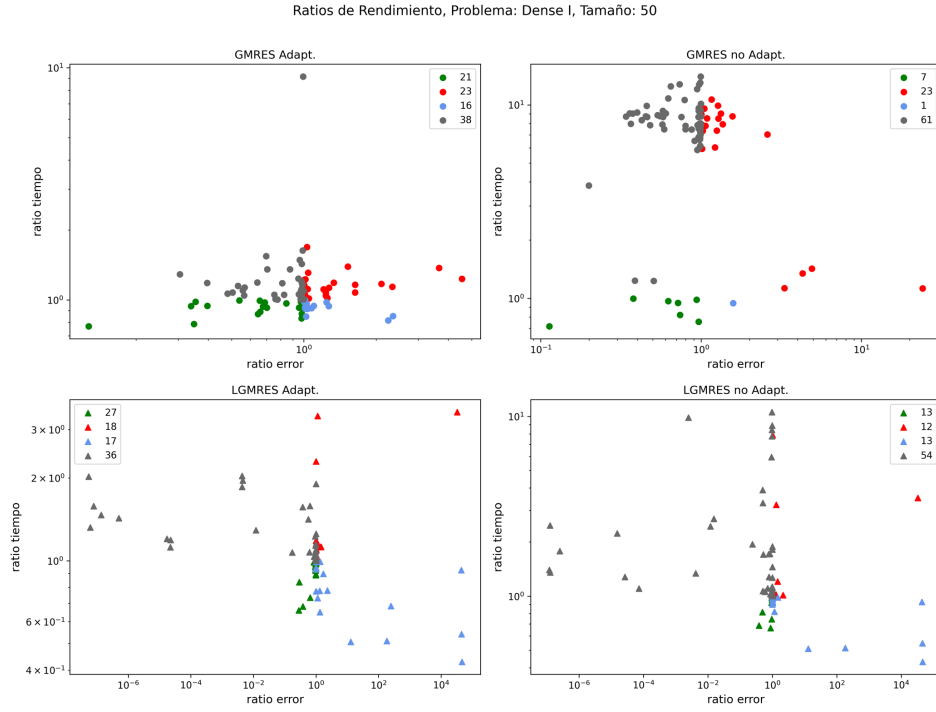


Figura 109: Equilibrio de rendimiento, problema Dense I, $n = 50$.

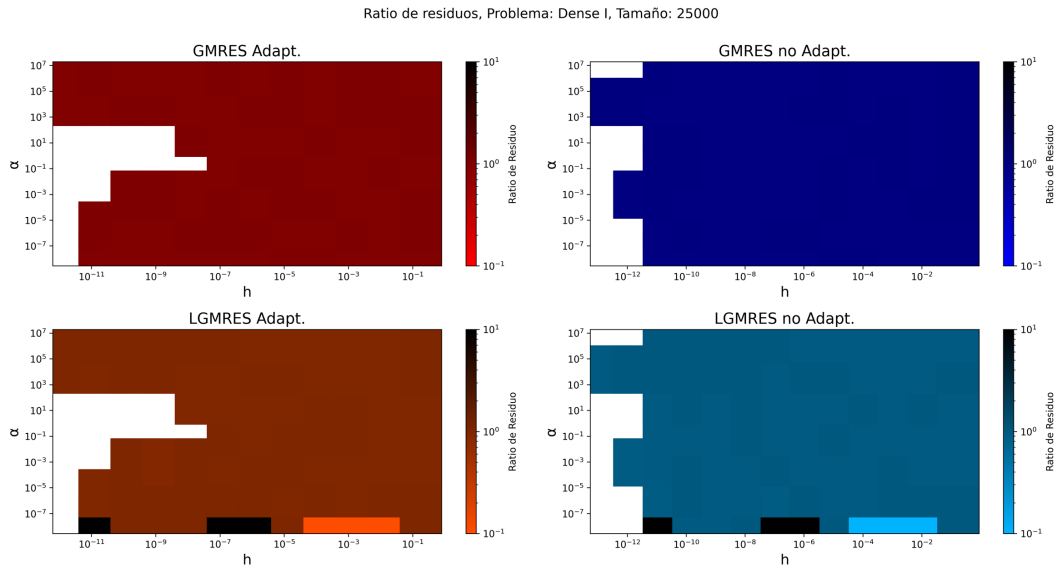


Figura 110: Ratio error, problema Dense I, $n = 25000$.

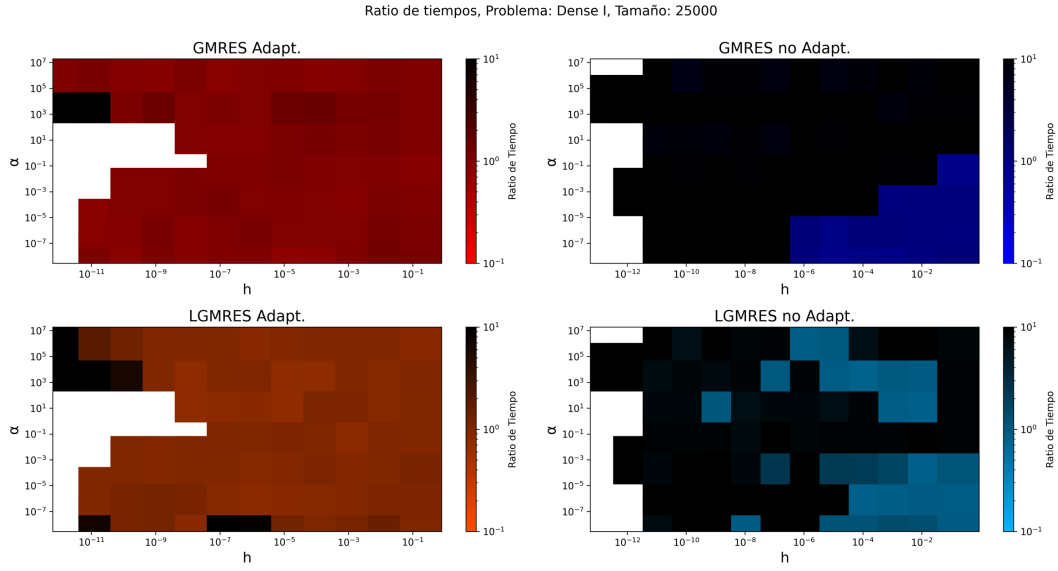


Figura 111: Ratio tiempo, problema Dense I, $n = 25000$.

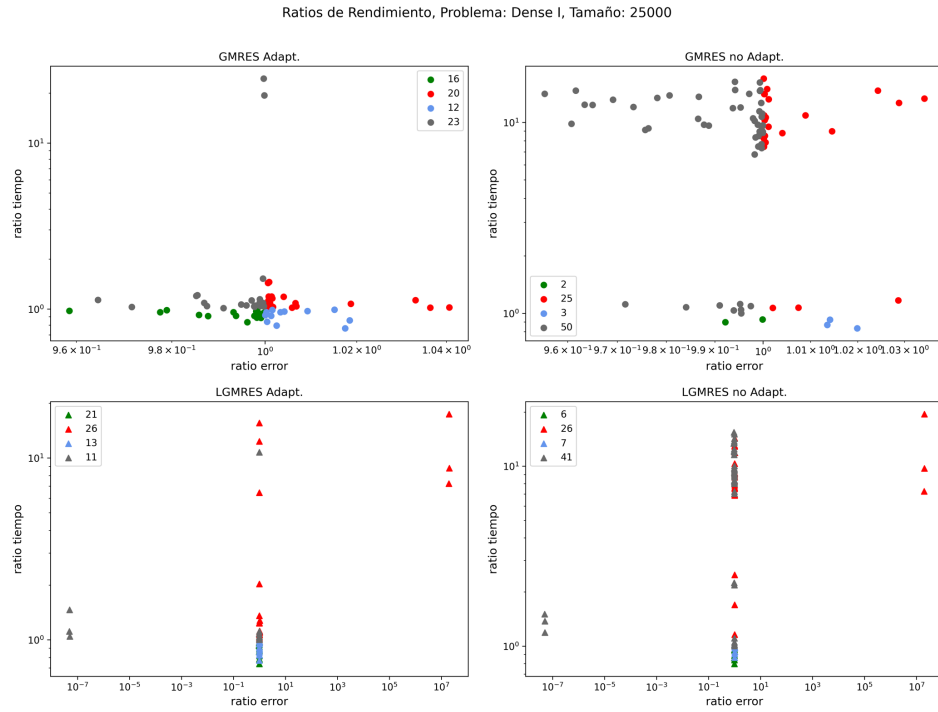


Figura 112: Equilibrio de rendimiento, problema Dense I, $n = 25000$.

Problema Dense II

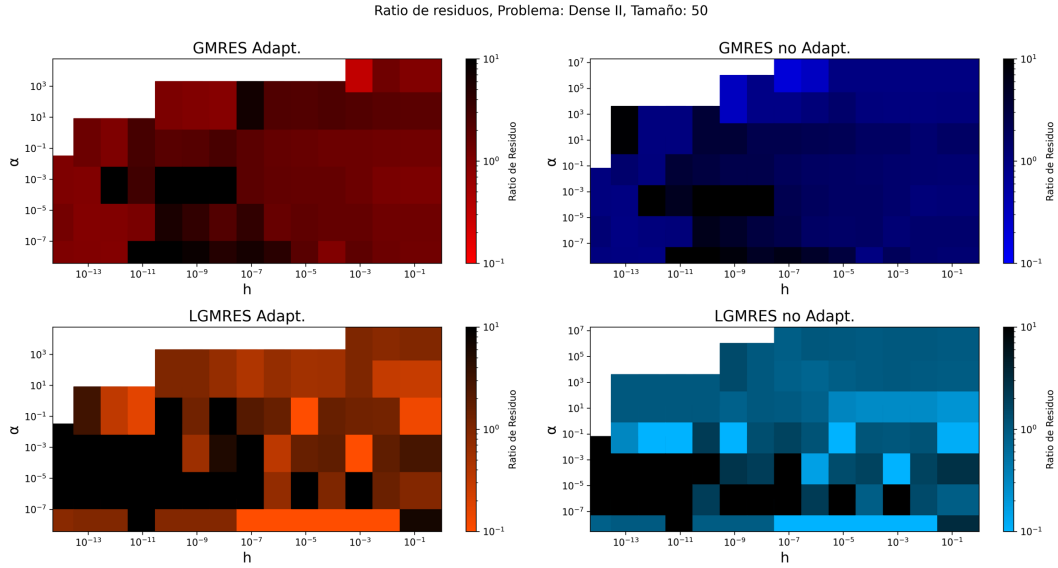


Figura 113: Ratio error, problema Dense II, $n = 50$.

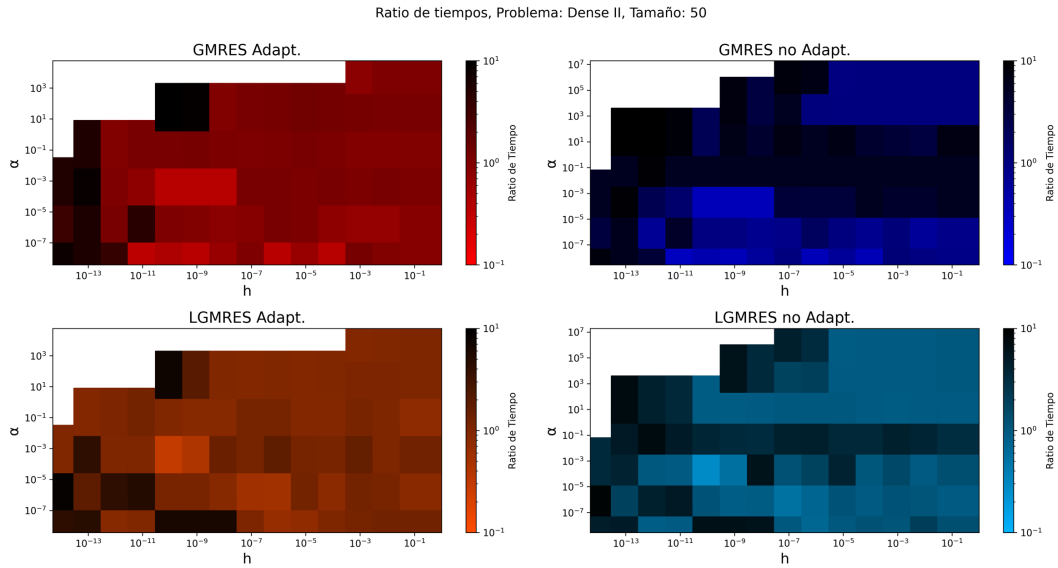


Figura 114: Ratio tiempo, problema Dense II, $n = 50$.

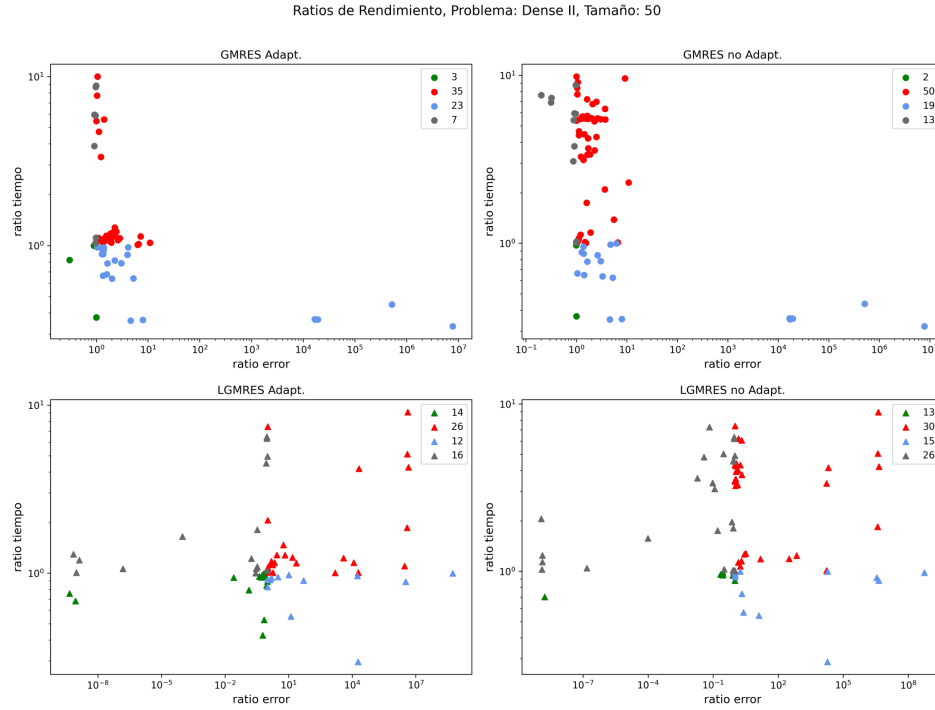


Figura 115: Equilibrio de rendimiento, problema Dense II, $n = 50$.

5.15. Gráficos adicionales, comparacion Lenguajes

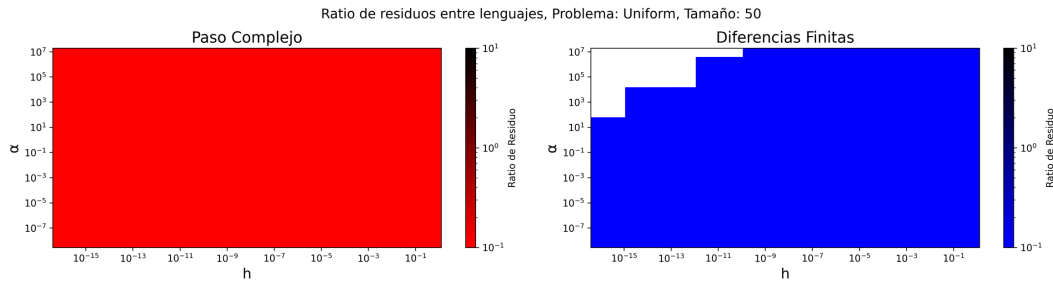


Figura 116: Ratio de error Julia/Python, problema Uniform, $n = 50$.

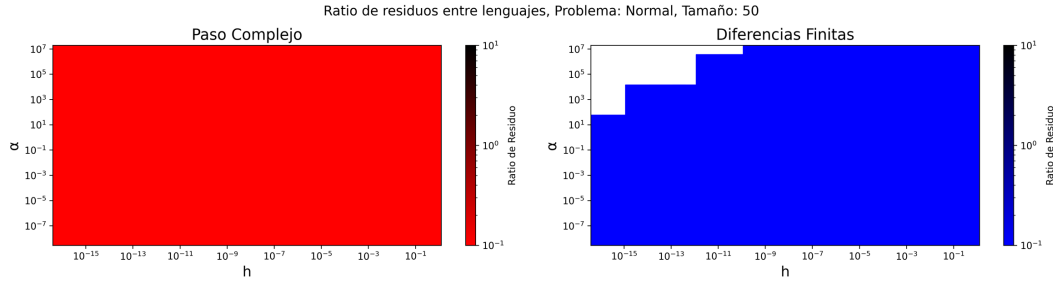


Figura 117: Ratio de error Julia/Python, problema Normal, $n = 50$.

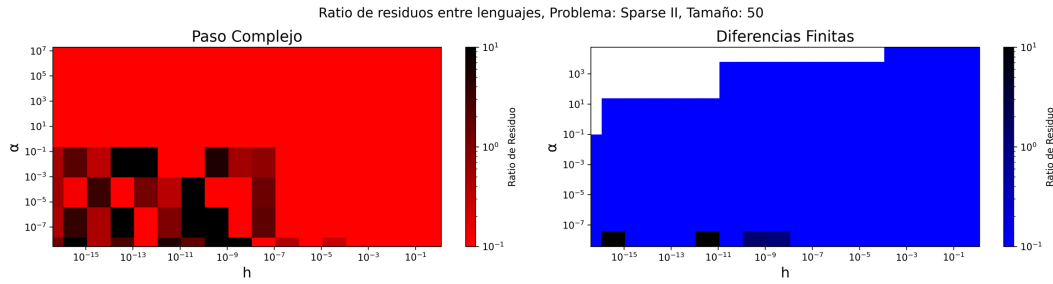


Figura 118: Ratio de error Julia/Python, problema Sparse II, $n = 50$.

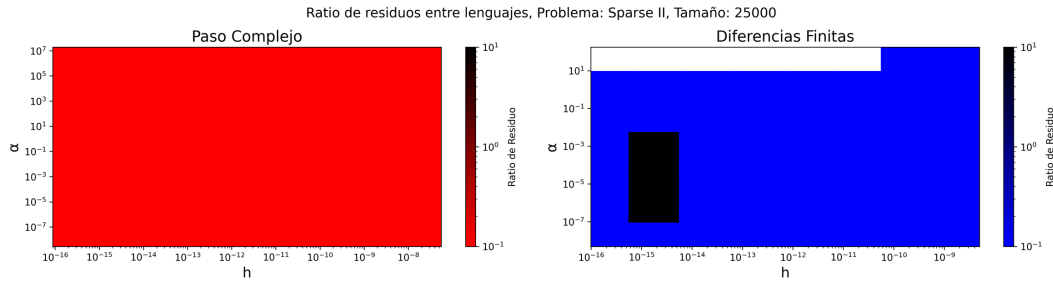


Figura 119: Ratio de error Julia/Python, problema Sparse II, $n = 25000$.

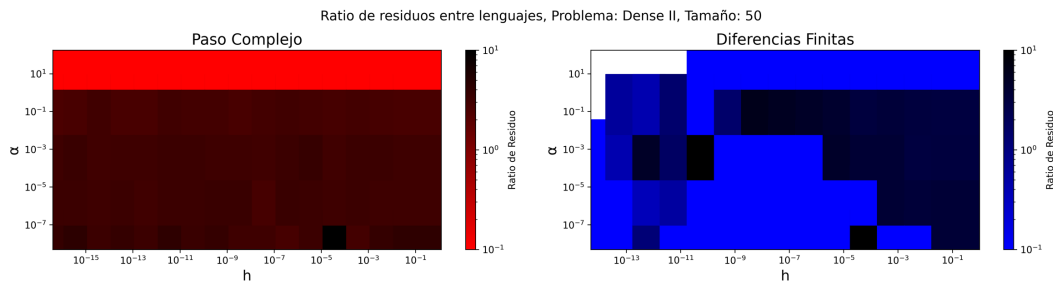


Figura 120: Ratio de error Julia/Python, problema Dense II, $n = 50$.

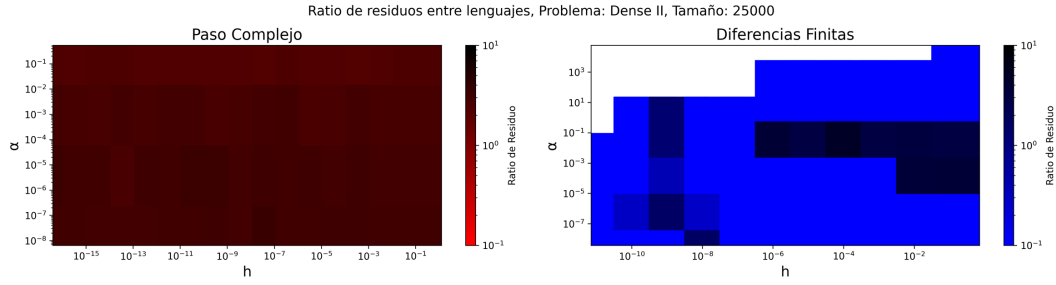


Figura 121: Ratio de error Julia/Python, problema Dense II, $n = 25000$.

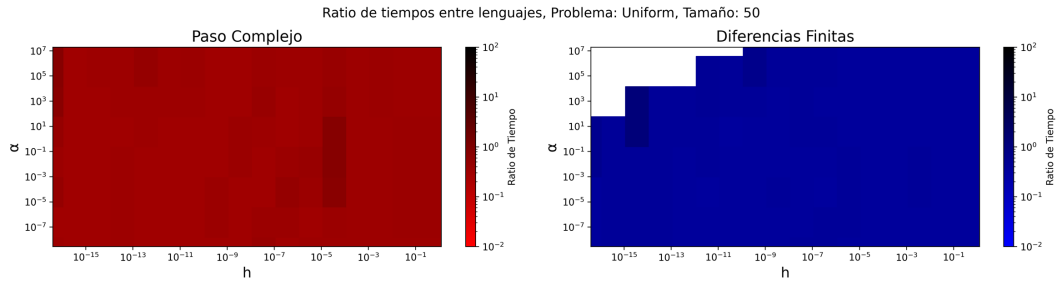


Figura 122: Ratio de tiempo Julia/Python, problema Uniform, $n = 50$.

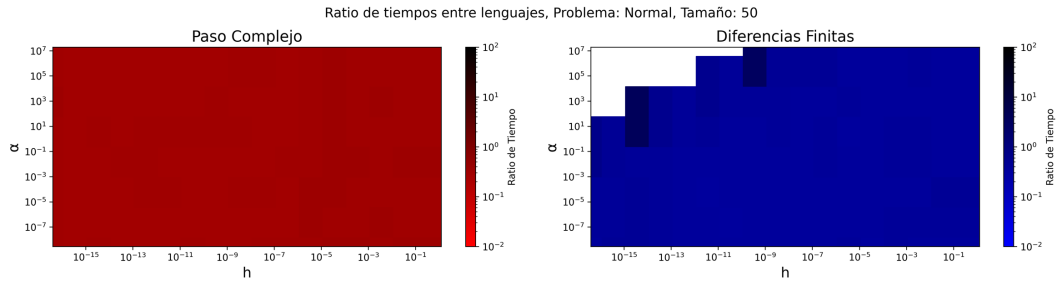


Figura 123: Ratio de tiempo Julia/Python, problema Normal, $n = 50$.

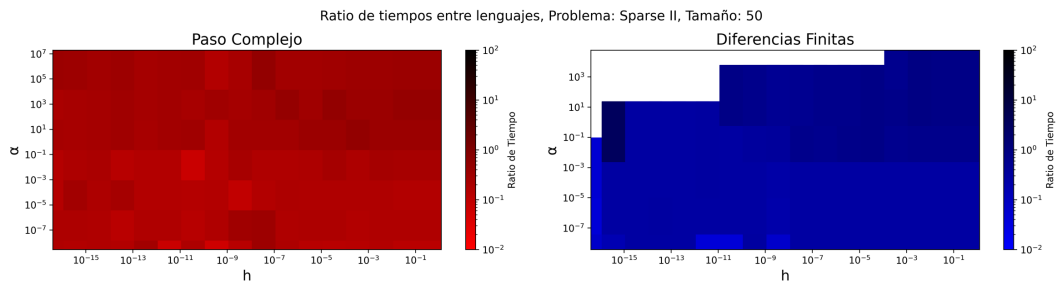


Figura 124: Ratio de tiempo Julia/Python, problema Sparse II, $n = 50$.

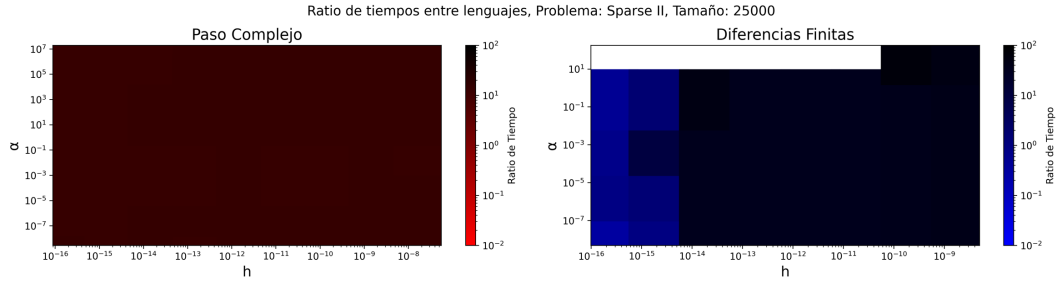


Figura 125: Ratio de tiempo Julia/Python, problema Sparse II, $n = 25000$.

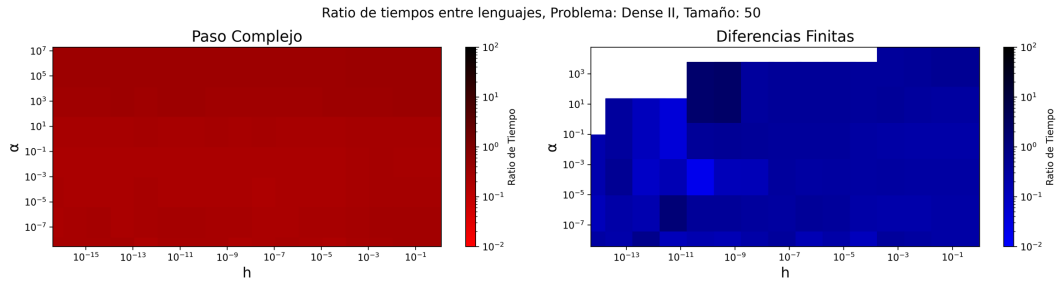


Figura 126: Ratio de tiempo Julia/Python, problema Dense II, $n = 50$.

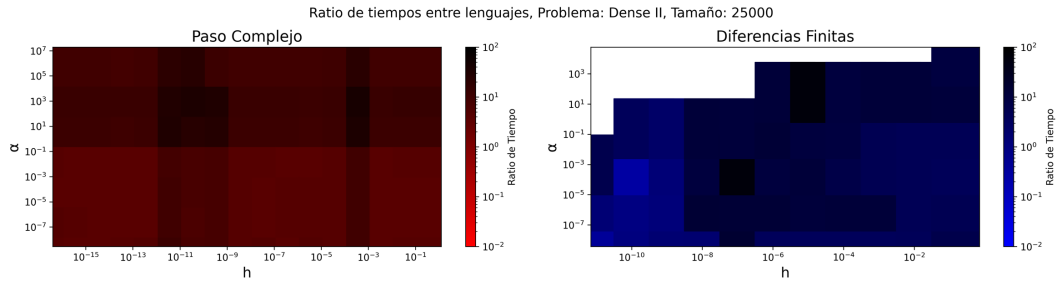


Figura 127: Ratio de tiempo Julia/Python, problema Dense II, $n = 25000$.

REFERENCIAS BIBLIOGRÁFICAS

- [aut,] Autograd documentation — autograd 1.0.0 documentation.
- [ove, 2019] (2019). Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84.
- [Ahsan, 2017] Ahsan, Z. (2017). *Differential Equations and Their Applications*. Delhi Phi Learning Private Limited.
- [Ames et al., 2014] Ames, W., Rheinboldt, W., y Jeffrey, A. (2014). *Numerical Methods for Partial Differential Equations*. Applications of Mathematics Series. Academic Press.
- [Ames, 2014] Ames, W. F. (2014). *Numerical Methods for Partial Differential Equations*. Academic Press, United Kingdom.
- [Atkinson, 1989] Atkinson, K. E. (1989). *An Introduction to Numerical Analysis*. Wiley.
- [Baker et al., 2005] Baker, A. H., Jessup, E. R., y Manteuffel, T. (2005). A technique for accelerating the convergence of restarted gmres. *SIAM Journal on Matrix Analysis and Applications*, 26(4):962–984.
- [Baker y Hughes, 1997] Baker, G. R. y Hughes, L. J. (1997). Complex-step differentiation. *Numerical Algorithms*, 16(4):401–415.
- [Baker y Johnson, 2018] Baker, R. y Johnson, M. (2018). The complex-step method for computing derivatives in numerical calculations. *Mathematics of Computation*, 87(309).
- [Berthold y Edwards, 2015] Berthold, J. W. y Edwards, K. L. (2015). Complex-step derivative approximation: an elegant and simple alternative to finite differences. *Journal of Computational Physics*, 284:1–9.
- [Burden y Faires, 2010] Burden, R. L. y Faires, J. D. (2010). *Numerical Analysis*. Brooks/Cole.
- [Callahan, 2010] Callahan, J. (2010). *Advanced Calculus: A Geometric View*. Undergraduate Texts in Mathematics. Springer New York.
- [Cohen, 2003] Cohen, J. S. (2003). *Computer Algebra and Symbolic Computation*. CRC Press, Boca Raton, FL.
- [Corliss et al., 2013] Corliss, G., Faure, C., Griewank, A., Hascoet, L., y Naumann, U. (2013). *Automatic Differentiation of Algorithms: From Simulation to Optimization*. Springer New York.
- [Fritzsche, 2015] Fritzsche, R. (2015). *Multicomplex Analysis: A New Approach to Complex Analysis*. Springer.

- [Goodfellow *et al.*, 2016] Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep Learning*. The MIT Press, Cambridge, Massachusetts.
- [Grossman, 1989] Grossman, R. (1989). *Symbolic Computation*. SIAM, Philadelphia.
- [Higham, 2002] Higham, N. J. (2002). *Accuracy and Stability of Numerical Algorithms*. SIAM.
- [Hutton, 2004] Hutton, D. V. (2004). *Numerical Methods for Engineers*. McGraw-Hill, New York, NY, USA.
- [James, 2011] James, G. (2011). *Advanced Modern Engineering Mathematics*. Pearson, Harlow, England; New York.
- [Kuipers, 2020] Kuipers, J. (2020). *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*. Princeton University Press.
- [La Cruz y Raydan, 2003] La Cruz, W. y Raydan, M. (2003). Nonmonotone Spectral Methods for Large-Scale Nonlinear Systems. *Optimization methods software*, 18(5):583–599.
- [2012] Gregory Lantoiné and others (2012). Using multicomplex variables for automatic computation of high-order derivatives. *ACM Transactions on Mathematical Software*, 38(3).
- [LeVeque, 2007] LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics.
- [Luo *et al.*, 2016] Luo, X., Zhang, S., y Hu, J. (2016). Multi-complex variable calculus and its applications. *Journal of Computational Mathematics*, 34(1):1–18.
- [Math StackExchange User, 2019] Math StackExchange User (2019). Directional derivative and jacobian matrix — math.stackexchange.com.
- [Moré, 1978] Moré, J. J. (1978). *The Levenberg-Marquardt algorithm: Implementation and theory*.
- [Muller *et al.*, 2018] Muller, J.-M., Brunie, N., de Dinechin, F., Jeannerod, C.-P., Joldes, M., Lefèvre, V., Melquiond, G., Revol, N., y Torres, S. (2018). *Handbook of Floating-Point Arithmetic (2nd ed.)*. Birkhäuser, Gewerbestrasse 11, 6330 Cham, Switzerland.
- [Naumann, 2012] Naumann, U. (2012). *The Art of Differentiating Computer Programs: An Introduction to Algorithmic Differentiation*. SIAM.
- [Needham, 1997] Needham, T. (1997). *Visual Complex Analysis*. Clarendon Press.
- [Nocedal y Wright, 2006] Nocedal, J. y Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, 2nd edición.
- [Phillips, 2023] Phillips, H. (2023). A simple introduction to gradient descent.

- [Quarteroni y Saleri, 2007] Quarteroni, A. y Saleri, F. (2007). *Numerical Mathematics*. Springer.
- [Saad, 2003] Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. SIAM.
- [Sanhueza Roman,] Sanhueza Roman, A. O. A limited-memory levenberg-marquardt algorithm for solving large-scale nonlinear least-square problems.
- [Sauer, 2013] Sauer, T. (2013). *Numerical Analysis: Pearson New International Edition*. Pearson Education.
- [SciPy Developers, 2024] SciPy Developers (2024). Dynamic step in scipy's optimize non-linear module. GitHub Repository.
- [Small y Wang, 2007] Small, C. G. y Wang, J. (2007). *Numerical Methods for Nonlinear Estimating Equations*. Clarendon Press, Oxford.
- [Sogabe, 2023] Sogabe, T. (2023). *Krylov Subspace Methods for Linear Systems: Principles of Algorithms*. Springer Series in Computational Mathematics. Springer Nature Singapore.
- [Stoer y Bulirsch, 2002] Stoer, J. y Bulirsch, R. (2002). *Introduction to Numerical Analysis*. Springer.
- [Sutton, 2019] Sutton, B. (2019). *Numerical Analysis: Theory and Experiments*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia.
- [Toint, 1987] Toint, P. L. (1987). On large scale nonlinear least squares calculations. *SIAM Journal on Scientific and Statistical Computing*, 8(3):416–435.
- [Van der Veen, 2012] Van der Veen, R. (2012). *Complex-Step Methods in Numerical Computations*. Springer.
- [2024] Ulziibayar Vandandoo and others (2024). *High-Order Finite Difference and Finite Element Methods for Solving Some Partial Differential Equations*. Synthesis Lectures on Engineering, Science, and Technology. Morgan & Claypool Publishers.
- [Verheyleweghen, 2014] Verheyleweghen (2014). *Computation of higher-order derivatives using the multi-complex step method*.
- [Watkins, 2004] Watkins, D. S. (2004). *Fundamentals of Matrix Computations*. John Wiley & Sons, Hoboken, NJ.