

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Estudio de las redes neuronales ortogonales y su aplicación en la ecuación de Bagley-Torvik

Juan Sebastian Manrique Moreno

Programa Académico de Física
Facultad de Ciencias Matemáticas y Naturales
Universidad Distrital Francisco José de Caldas

Bogotá D.C., Colombia
27 de octubre de 2025

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Facultad de Ciencias Matemáticas y Naturales

Estudio de las redes neuronales ortogonales y su aplicación en la ecuación de Bagley-Torvik

Monografía de trabajo de grado para optar por el título de Físico

Juan Sebastian Manrique Moreno

Dirigido por: Deccy Yaneth Trejos Angel, Ph.D.
Programa Académico de Matemáticas
Universidad Distrital Francisco José de Caldas

Evaluado por: Carlos Andrés Gómez Vasco, Ms.C.
Programa Académico de Física
Universidad Distrital Francisco José de Caldas

Programa Académico de Física
Facultad de Ciencias Matemáticas y Naturales
Universidad Distrital Francisco José de Caldas

MSC: 76A10, 34A08, 68T07.

Bogotá D.C., Colombia
27 de octubre de 2025

"Ninguno puede ser grande en una profesión sin amarla.
Amad la vuestra y hacedla amar de vuestros conciudadanos
por una conducta noble, dulce y virtuosa."[†]

∼ Francisco José de Caldas, 1814 ∼

[†]Referencia a la frase en: [1, p. 88]

Agradecimientos

En primer lugar quiero agradecer a Dios y a la vida por darme la oportunidad de estudiar aquello que siempre soñé: la Física. Agradezco profundamente a la Universidad Distrital Francisco José de Caldas, a la Facultad de Ciencias Matemáticas y Naturales y al Programa Académico de Física, por ser el espacio donde pude crecer, aprender y convertirme en la persona y el académico que soy hoy. Este camino no solo me ha formado en el ámbito profesional, sino también en lo humano, y por eso llevaré siempre conmigo lo que esta universidad me enseñó.

A mi madre, con todo mi corazón, gracias por ser mi pilar más firme y constante, por acompañarme en cada paso, por sostenerme cuando el cansancio y la duda me pesaban, y por darme siempre la fuerza para seguir. Gracias por creer en mí incluso cuando yo mismo dudaba, por apoyar cada uno de mis procesos académicos de la manera que fuera posible y, sobre todo, por haber hecho de mí el ser humano que soy. Mami, este logro es tanto tuyo como mío, y no hay palabras suficientes para expresar mi gratitud y amor hacia ti, solo espero recompensarte con creces todo el sacrificio dedicado a mí.

A mis hermanas Daniela y Laura, y mi tía Cristina, gracias por preocuparse siempre por mí, por alegrarse con mis logros y animarme en los momentos difíciles, por compartir con entusiasmo cada paso de este recorrido. Su cariño y apoyo han sido una motivación constante y una fuente de alegría en este camino.

Quiero agradecer también a mis docentes Carlos Vasco y John Díaz, por haber formado la escuela que llevo en mi pensamiento y en mi manera de entender la Física. Gracias por ser guías y referentes, por cada enseñanza, cada corrección y cada conversación que me ayudó a ser mejor. Ustedes forjaron no solo mis conocimientos, sino también mi vocación, y me llena de orgullo poder decir que fueron mis profesores. Espero, con el tiempo, honrar su enseñanza y dejar en alto el nombre de quienes me formaron y siempre llevaré conmigo como maestros y como ejemplo.

A mi amigo Santiago Beltrán, gracias por tu compañía incondicional, por estar presente en cada etapa, por haber compartido conmigo tanto en el colegio como en la universidad. Me alegra ver cómo seguimos creciendo juntos, cumpliendo sueños y alcanzando metas. Nuestra amistad ha perdurado a través de los años y las etapas, y espero que siga siendo así por mucho más tiempo.

Finalmente, pero no por eso menos importante, quiero agradecer a todos mis amigos que me acompañaron desde el inicio de esta aventura. Gracias por ser parte de este proceso, por compartir desvelos, risas, proyectos, por estar en los momentos de triunfo y en los de caída, por esas tardes en los galpones o en los techos de la Macarena simplemente charlando, por esos almuerzos juntos y claramente, por esas tardes de ocio productivo. Gracias por las noches de estudio interminables, por las conversaciones que nos hicieron crecer, por las bromas que aliviaron la carga, por cada recuerdo que hoy guardo con enorme cariño. A ustedes, Santiago Talero, Laura Oliveros, Julián Aros, Daniel Ramírez, Santiago Criollo, César Mendoza, Laura Triana, María Moreno, Sofía Cortés, Pablo Patiño, Sebastián Sánchez, Marcos Torres y Zaudí Gómez, gracias de corazón por acompañarme y ser parte de este capítulo de mi vida. Empezamos juntos un 20 de septiembre de 2020, en circunstancias diferentes y desafiantes, pero con el mismo sueño que hoy nos une. Cinco años después seguimos aquí, juntos, demostrando que los comienzos difíciles forjan vínculos fuertes.

Siempre seremos la primera cohorte de Física, los primeros matriculados, los que confiaron en esto a penas inició y no se fueron, los originales. Gracias por haber sido y seguir siendo parte de este viaje.

A mi equipo de trabajo...

A mi profesora Deccy, quiero expresar mi más profundo agradecimiento por su constante apoyo, orientación y acompañamiento a lo largo de todo este proceso. Su compromiso y paciencia fueron un pilar fundamental para que este trabajo pudiera materializarse. En cada etapa del camino, usted estuvo presente, ofreciendo su guía con una combinación única de rigor académico y empatía humana.

Su disposición para escuchar, su claridad al enseñar y la confianza que depositó en mí marcaron una diferencia invaluable. Gracias a su dedicación, pude aprender no solo sobre la ciencia detrás de este proyecto, sino también sobre la disciplina, la perseverancia y el amor por el conocimiento.

Agradezco sinceramente el tiempo, la energía y el cariño que invirtió en mi formación. Este logro también es suyo, profesora.

De corazón, ¡gracias por todo!



Deccy Yaneth Trejos Angel, Ph.D.

A mi profesor Carlos, quiero extender un agradecimiento muy especial por todo el apoyo brindado a este trabajo. Usted fue una pieza fundamental en su desarrollo y, aunque su colaboración fue siempre por pura voluntad, lo sentí como si hubiera sido mi codirector. Cada uno de sus aportes y observaciones ha contribuido de manera decisiva a que este proyecto sea lo que es hoy.

Sin su guía y dedicación, no creo que hubiera sido capaz de presentar un trabajo tan correcto, elegante y bien elaborado como considero que es este. No tengo más que palabras de gratitud hacia usted y el más sincero deseo de que este sea solo el comienzo de una gran trayectoria académica compartida.

Agradezco también a la profesora Deccy por haberme presentado con usted, y a la vida por darme la oportunidad de trabajar a su lado. Espero sinceramente que podamos seguir colaborando por mucho más tiempo.

Gracias, de corazón.



Carlos Arturo Peña Rincón, Ph.D.

Contenido

Resumen **1**

Abstract **2**

Introducción **3**

I El problema de Bagley-Torvik **4**

1 Preliminares: Cálculo fraccionario **6**
1.1 Función Gamma $\Gamma(z)$ . . . 6
1.2 Funciones de Mittag-Leffler . . . 7
1.2.1 Definición de las funciones de Mittag-Leffler . . . 7
1.2.2 Derivada de las funciones de Mittag-Leffler . . . 8
1.3 Definiciones de la derivada fraccionaria . . . 9
1.3.1 Integral fraccionaria . . . 9
1.3.2 Derivada de Riemann-Liouville . . . 9
1.3.3 Derivada de Caputo . . . 10
1.4 Relaciones de la derivada de Riemann-Liouville y la derivada de Caputo . . . 12
1.5 Algunas propiedades de las derivadas de Riemann-Liouville y Caputo . . . 12
1.6 Ecuaciones diferenciales fraccionarias ordinarias (FODE) . . . 13
1.6.1 Ecuación diferencial lineal fraccionaria ordinaria de un término . . . 13
1.6.2 Ecuación diferencial lineal fraccionaria ordinaria de dos términos . . . 14
1.6.3 Ecuación diferencial lineal fraccionaria ordinaria de tres términos . . . 14
1.7 Solución analítica de ecuaciones diferenciales fraccionarias ordinarias (FODE) . . . 14
1.7.1 Convolución de funciones . . . 14
1.7.2 Transformada de Laplace . . . 15
1.7.2.1 Definición . . . 15
1.7.2.2 Algunas propiedades útiles . . . 15
1.7.2.3 Aplicación en derivadas de orden entero . . . 16
1.7.2.4 Aplicación en derivadas de orden fraccionario . . . 17
1.7.3 Funciones de Green fraccionarias . . . 17

2 La ecuación de Bagley-Torvik **20**
2.1 Viscocidad y tensión cortante . . . 20
2.1.1 El problema de Rayleigh (primer problema de Stokes) . . . 21
2.1.2 El carácter fraccionario en viscoelástica . . . 22
2.2 Planteamiento físico del problema . . . 23
2.3 Deducción matemática . . . 23
2.4 Solución analítica por medio de la función de Green . . . 25

II Las redes neuronales ortogonales **27**

3 Redes Neuronales Artificiales (ANN) **29**
3.1 Definición y fundamentos . . . 29

3.2 Estructura general . . . 29
3.2.1 Neuronas artificiales . . . 31
3.2.2 Capas . . . 31
3.2.3 Funciones de activación . . . 32
3.2.4 Funciones de costo . . . 33
3.3 Entrenamiento . . . 34
3.3.1 Métodos de primer orden . . . 35
3.3.1.1 SGD (Stochastic Gradient Descent) con Momentum . . . 35
3.3.1.2 Adam (Adaptive Moment Estimation) . . . 35
3.3.1.3 AdamW (Adam with Weight Decay) . . . 35
3.3.1.4 RMSprop (Root Mean Square Propagation) . . . 35
3.3.1.5 Adagrad (Adaptive Gradient) . . . 36
3.3.2 Métodos de segundo orden . . . 36
3.3.2.1 BFGS (Broyden-Fletcher-Goldfarb-Shanno) . . . 36
3.3.2.2 L-BFGS-B (Limited-memory BFGS with Bounds) . . . 36
3.3.2.3 Newton-CG (Newton Conjugate Gradient) . . . 37
3.4 Arquitecturas principales . . . 37
3.4.1 El perceptrón de Rosenblatt . . . 37
3.4.1.1 Arquitectura del perceptrón . . . 37
3.4.1.2 Regla de aprendizaje . . . 37
3.4.1.3 Teorema de convergencia . . . 38
3.4.1.4 Su importancia . . . 38
3.4.2 Perceptrón Multicapa (MLP) . . . 38
3.4.3 Otras arquitecturas de redes neuronales populares . . . 39
3.4.3.1 Redes Neuronales Convolucionales (CNN) . . . 39
3.4.3.2 Redes Neuronales Recurrentes (RNN) . . . 39
3.4.3.3 Autoencoders . . . 40
3.4.3.4 Redes Generativas Antagónicas (GANs) . . . 40

4 Polinomios Ortogonales **41**
4.1 Fundamentos . . . 41
4.1.1 Propiedad de ortogonalidad . . . 42
4.1.2 Fórmula de recurrencia . . . 42
4.1.3 Propiedad de los ceros . . . 42
4.1.4 Polinomios ortogonales clásicos . . . 43
4.2 Polinomios de Hermite . . . 44
4.3 Polinomios de Laguerre . . . 45
4.4 Polinomios de Legendre . . . 46

5 Redes Neuronales Ortogonales **48**
5.1 Definición y estructura . . . 48
5.2 Tipos de ONN utilizadas . . . 49
5.2.1 HerNN (Hermite Neural Network) . . . 49
5.2.2 LaNN (Laguerre Neural Network) . . . 49
5.2.3 LeNN (Legendre Neural Network) . . . 50
5.3 Función de costo . . . 50
5.4 Entrenamiento . . . 50

6 Extreme Learning Machine (ELM) **51**
6.1 Definición . . . 51
6.2 Pseudo-inversa generalizada de Moore-Penrose . . . 51
6.3 Funcionamiento y aplicación . . . 54

6.3.1 Funcionamiento . . . 54
6.3.2 Aplicación a una *FODE* de tres términos . . . 55

III Resultados **56**

7 Problema a resolver **57**

8 Resultados **58**
8.1 Bitácora de resultados . . . 58
8.1.1 Métodos de entrenamiento de primer orden . . . 58
8.1.1.1 Resultados para: HerNN . . . 58
8.1.1.2 Resultados para: LaNN . . . 59
8.1.1.3 Resultados para: LeNN . . . 60
8.1.1.4 Mejores resultados (HerNN vs. LaNN vs. LeNN) . . . 61
8.1.2 Métodos de segundo orden . . . 62
8.1.2.1 Resultados para: HerNN . . . 62
8.1.2.2 Resultados para: LaNN . . . 63
8.1.2.3 Resultados para: LeNN . . . 64
8.1.2.4 Mejores resultados (HerNN vs. LaNN vs. LeNN) . . . 65
8.1.3 Extreme Learning Machine (ELM) . . . 66
8.1.3.1 Resultados para: HerNN . . . 67
8.1.3.2 Resultados para: LaNN . . . 68
8.1.3.3 Resultados para: LeNN . . . 69
8.1.3.4 Mejores resultados (HerNN vs. LaNN vs. LeNN) . . . 70

9 Análisis de resultados y conclusiones **71**

Referencias **74**

Anexos **A**

A Códigos **A**
A.1 Método de la matriz de Podlubny . . . A
A.2 Métodos de primer orden . . . B
A.3 Métodos de segundo orden . . . C
A.4 ELM . . . E
A.4.1 HerNN . . . E
A.4.2 LaNN . . . E
A.4.3 LeNN . . . F

Resumen

Esta monografía se inscribe en el ámbito de la física matemática y aborda la interacción entre el cálculo fraccionario y las redes neuronales ortogonales en la modelación de sistemas viscoelásticos con memoria, tomando como caso representativo la ecuación diferencial fraccionaria de Bagley-Torvik. En la primera parte se examina el marco teórico del cálculo fraccionario y su papel en la descripción de fenómenos de disipación y relajación temporal, derivando rigurosamente la ecuación de Bagley-Torvik a partir del problema de Stokes para un fluido newtoniano. En la segunda, se revisan las bases matemáticas y propiedades de las redes neuronales ortogonales sustentadas en polinomios clásicos de Hermite, Laguerre y Legendre, destacando su coherencia con los distintos dominios físicos de aplicación. Finalmente, se compara el desempeño de tres arquitecturas (HerNN, LaNN y LeNN) bajo diversos esquemas de entrenamiento para aproximar la solución de dicha ecuación.

Los resultados experimentales revelan un patrón consistente: el esquema *Extreme Learning Machine* (ELM) proporciona la mejor combinación de precisión y eficiencia computacional, alcanzando para la arquitectura de Legendre (LeNN) una pérdida de 0,04035, un RMSE de 0,20088 y un coeficiente de correlación $r = 0{,}9929$, con tiempos de ejecución inferiores a 0,1 s. Los métodos iterativos de segundo orden (BFGS y Newton-CG) también mostraron un rendimiento notable, aunque con mayor variabilidad según la arquitectura y las condiciones numéricas. El análisis teórico-numérico evidenció además la influencia del intervalo de convergencia de los polinomios y la sensibilidad del operador fraccionario frente a la discretización temporal, factores determinantes en la estabilidad de la aproximación. En conjunto, los resultados confirman que la red de Legendre entrenada mediante ELM ofrece la representación más precisa y estable de la dinámica viscoelástica descrita por la ecuación de Bagley-Torvik, consolidando el enfoque fraccionario-ortogonal como una herramienta eficaz para el estudio de sistemas físicos con memoria y comportamiento no local.

Palabras clave: Cálculo fraccionario; Ecuación de Bagley-Torvik; Física matemática; Viscoelasticidad; Redes neuronales ortogonales; Polinomios ortogonales; Extreme Learning Machine.

Abstract

This monograph lies within the field of mathematical physics and explores the interplay between fractional calculus and orthogonal neural networks in modeling viscoelastic systems with memory, taking the fractional Bagley-Torvik equation as a representative case. The first part examines the theoretical framework of fractional calculus and its role in describing dissipation and temporal relaxation, deriving the Bagley-Torvik equation rigorously from Stokes' problem for a Newtonian fluid. The second part reviews the mathematical foundations of orthogonal neural networks based on classical Hermite, Laguerre, and Legendre polynomials, emphasizing their consistency with distinct physical domains. Finally, a comparative analysis of three architectures (HerNN, LaNN, and LeNN) under various training schemes is conducted to approximate the equation's numerical solution.

Experimental results show a clear pattern: the *Extreme Learning Machine* (ELM) scheme achieves the best balance between accuracy and computational efficiency, with the Legendre-based network (LeNN) reaching a loss of 0,04035, RMSE of 0,20088, and correlation coefficient $r = 0{,}9929$ in less than 0,1 s of computation. Second-order iterative methods (BFGS and Newton-CG) also yielded competitive results, though with greater sensitivity to the network architecture and numerical conditions. The theoretical-numerical analysis further highlighted the impact of the convergence interval of orthogonal polynomials and the discretization sensitivity of the fractional operator on solution stability. Overall, the findings confirm that the Legendre network trained with ELM provides the most accurate and stable representation of the viscoelastic dynamics described by the Bagley-Torvik equation, establishing the fractional-orthogonal framework as a powerful tool for modeling physical systems with memory and nonlocal behavior.

Keywords: Fractional calculus; Bagley-Torvik equation; Mathematical physics; Viscoelasticity; Orthogonal neural networks; Orthogonal polynomials; Extreme Learning Machine.

Introducción

El estudio de ecuaciones diferenciales fraccionarias ha adquirido en las últimas décadas una relevancia notable en el campo de la física matemática y de las ciencias aplicadas, en gran medida debido a su capacidad para modelar fenómenos de carácter complejo en los que intervienen memoria y efectos no locales [2]-[8]. Entre estas ecuaciones destaca la propuesta por Bagley y Torvik en 1984 [2], destinada a describir el movimiento de una placa unida a un resorte e inmersa en un fluido newtoniano. La ecuación de Bagley-Torvik constituye una primera aproximación de las aplicaciones del cálculo fraccionario en la física, con un problema relativamente sencillo.

En paralelo al desarrollo del cálculo fraccionario, el avance del *Machine Learning* ha impulsado la creación de técnicas de aproximación basadas en arquitecturas neuronales cada vez más sofisticadas [9]-[12]. Dentro de este ámbito, las redes neuronales ortogonales representan una línea de investigación particular, pues incorporan en su estructura funciones de activación definidas a partir de polinomios ortogonales, logrando con ello un vínculo entre la teoría clásica de aproximación y los métodos de aprendizaje automático [13]. Estas arquitecturas no solo heredan las propiedades de estabilidad y convergencia propias de los polinomios ortogonales [14]-[16], sino que además se benefician de la versatilidad de los modelos neuronales, constituyendo una alternativa prometedora para la resolución de problemas diferenciales de elevada complejidad [17]-[21].

En este contexto, el objetivo general de la presente monografía consiste en aproximar numéricamente la ecuación de Bagley-Torvik mediante redes neuronales ortogonales basadas en distintas familias de polinomios, en particular las arquitecturas *Hermite Neural Network* (HerNN) [19], [20], *Laguerre Neural Network* (LaNN) [18] y *Legendre Neural Network* (LeNN) [17], [21], [22]. Este propósito busca evaluar y comparar el desempeño de dichas estructuras en términos de precisión, estabilidad y eficiencia, a partir del contraste de las métricas obtenidas en la función costo asociada a la solución numérica del problema.

Para ello, la primera parte de este trabajo examina los fundamentos teóricos del cálculo fraccionario, haciendo especial énfasis en la formulación y características de la ecuación de Bagley-Torvik. En esta sección se abordan los principios matemáticos del cálculo fraccionario así como la deducción matemática de la ecuación de Bagley-Torvik a partir del problema físico, dando cumplimiento al segundo objetivo específico planteado.

La segunda parte revisa el marco teórico fundamental de las redes neuronales ortogonales y su aplicación en distintos contextos de aproximación, presentando los conceptos esenciales sobre su estructura, propiedades matemáticas y métodos de entrenamiento. Con ello, se da respuesta al primer objetivo específico, orientado a establecer las bases conceptuales necesarias para comprender el funcionamiento y las ventajas de este tipo de arquitecturas neuronales.

Finalmente, se presentan los resultados tras realizar la aplicación numérica de las redes neuronales ortogonales en la aproximación de la ecuación de Bagley-Torvik. Se presentan los procedimientos de implementación, las métricas de evaluación y el análisis comparativo del desempeño de las arquitecturas HerNN, LaNN y LeNN, dando respuesta al último objetivo específico, que busca identificar la configuración más eficiente y precisa para la resolución del problema bajo estudio.

Parte I

El problema de Bagley-Torvik

Entre las aplicaciones más destacadas del cálculo fraccionario en física se encuentra la ecuación de Bagley-Torvik, propuesta en 1984 [2], la cual describe el movimiento de una placa unida a un resorte e inmersa en un fluido newtoniano, como el agua, el aceite o el aire. Desde su planteamiento, numerosos autores han profundizado en su formulación, deducción y en la solución asociada [3], [7], [21], [23], [24].

En este apartado se presenta el marco teórico que fundamenta la ecuación diferencial de Bagley-Torvik, se desarrolla su deducción y se expone su solución analítica, con el propósito de ofrecer una visión integral de sus principios matemáticos y físicos.

1 Preliminares: Cálculo fraccionario **6**
- 1.1 Función Gamma $\Gamma(z)$ . . . 6
- 1.2 Funciones de Mittag-Leffler . . . 7
  - 1.2.1 Definición de las funciones de Mittag-Leffler . . . 7
  - 1.2.2 Derivada de las funciones de Mittag-Leffler . . . 8
- 1.3 Definiciones de la derivada fraccionaria . . . 9
  - 1.3.1 Integral fraccionaria . . . 9
  - 1.3.2 Derivada de Riemann-Liouville . . . 9
  - 1.3.3 Derivada de Caputo . . . 10
- 1.4 Relaciones de la derivada de Riemann-Liouville y la derivada de Caputo . . . 12
- 1.5 Algunas propiedades de las derivadas de Riemann-Liouville y Caputo . . . 12
- 1.6 Ecuaciones diferenciales fraccionarias ordinarias (FODE) . . . 13
  - 1.6.1 Ecuación diferencial lineal fraccionaria ordinaria de un término . . . 13
  - 1.6.2 Ecuación diferencial lineal fraccionaria ordinaria de dos términos . . . 14
  - 1.6.3 Ecuación diferencial lineal fraccionaria ordinaria de tres términos . . . 14
- 1.7 Solución analítica de ecuaciones diferenciales fraccionarias ordinarias (FODE) . . . 14
  - 1.7.1 Convolución de funciones . . . 14
  - 1.7.2 Transformada de Laplace . . . 15
    - 1.7.2.1 Definición . . . 15
    - 1.7.2.2 Algunas propiedades útiles . . . 15
    - 1.7.2.3 Aplicación en derivadas de orden entero . . . 16
    - 1.7.2.4 Aplicación en derivadas de orden fraccionario . . . 17
  - 1.7.3 Funciones de Green fraccionarias . . . 17

2 La ecuación de Bagley-Torvik **20**
- 2.1 Viscocidad y tensión cortante . . . 20
  - 2.1.1 El problema de Rayleigh (primer problema de Stokes) . . . 21
  - 2.1.2 El carácter fraccionario en viscoelástica . . . 22
- 2.2 Planteamiento físico del problema . . . 23

2.3 Deducción matemática 23
2.4 Solución analítica por medio de la función de Green 25

Capítulo 1

Preliminares: Cálculo fraccionario

El cálculo fraccionario es una rama del análisis matemático con más de 300 años de historia. Constituye una generalización del cálculo clásico, al extender los operadores habituales de diferenciación e integración a órdenes no enteros. Su origen, o al menos la primera referencia documentada, se encuentra en una carta de Leibniz a L'Hôpital fechada el 30 de septiembre de 1695 [25]. Según la adaptación presentada por Lazarević et al. [26], la conversación fue la siguiente:

> Leibniz planteó a L'Hôpital la siguiente cuestión: «*¿Se puede generalizar el significado de las derivadas de orden entero a derivadas de orden no entero?*»
>
> Intrigado, L'Hôpital replicó con una pregunta sencilla: «*¿Y si el orden fuera* $1/2$*?*»
>
> A lo que Leibniz respondió: «*Esto conducirá a una paradoja, de la cual algún día se extraerán consecuencias útiles.*»

A partir de esta cuestión planteada por Leibniz y L'Hôpital, el cálculo fraccionario comenzó a desarrollarse como un área de interés para las mentes matemáticas más destacadas de Europa. En 1730, el matemático suizo Leonhard Euler realizó una importante contribución al respecto:

> *Si n es un número entero positivo y p una función de x, entonces la derivada de $p^n(x)$ con respecto a x puede expresarse mediante una fórmula algebraica. La cuestión que surge es qué ocurre con esta razón de cambio cuando n ya no es un número entero, sino un exponente fraccionario.*

Esta cita corresponde a una adaptación de la traducción al inglés realizada por Stacy G. Langton del artículo original de Euler [27].

Posteriormente, numerosos matemáticos realizaron aportes significativos en esta área, entre ellos Lagrange en 1772, Laplace en 1812, Lacroix en 1819, Fourier en 1822, Riemann en 1847, Green en 1859, Grunwald en 1867 y Letnikov en 1868, entre otros hasta la actualidad [26].

A continuación se expone el marco fundamental del cálculo fraccionario, iniciando con definiciones que permiten extender de manera rigurosa el concepto clásico de derivada hacia órdenes no enteros. Este proceso de generalización constituye la base teórica sobre la cual se construyen las formulaciones posteriores, siendo la función gamma ($\Gamma(z)$) el primer objeto matemático indispensable para formalizar dicha extensión.

1.1 | Función Gamma $\Gamma(z)$

Una de las funciones más importantes en el análisis matemático por diversas razones es la función Gamma de Euler, así esta se define como

Definición 1.1.1 Función Gamma

La función Gamma es definida mediante la siguiente integral definida:

$$\Gamma(z) = \int_0^\infty \mathrm{e}^{-t} t^{z-1}\,\mathrm{d}t, \tag{1.1.1}$$

cuya convergencia se encuentra en $\Re(z) > 0$.

Tomado de: [6], [7]

Donde su principal importancia en el contexto del cálculo fraccionario radica en la siguiente propiedad

Teorema 1.1.1 Propiedad de la función Gamma

Una propiedad importante de la función Gamma es:

$$\Gamma(z+1) = z\Gamma(z). \tag{1.1.2}$$

Véase la demostración en: [6, cap. 2, p. 24]

Este teorema entrega de manera formal una forma de generalizar el concepto del factorial a los números complejos z con $\Re(z) > 0$, dominio en el cual converge $\Gamma(z)$. Esto se puede ver así

Corolario 1.1.1 La generalización del factorial

Como un caso específico de (1.1.2), para $\Re(z) > 0$, se obtiene

$$\Gamma(z+1) = z\Gamma(z) = z\,(z-1)\,\Gamma(z-1) = \cdots = z!. \tag{1.1.3}$$

Véase la demostración en: [6, cap. 2, p. 25]

1.2 | Funciones de Mittag-Leffler

Las funciones de Mittag-Leffler constituyen una extensión directa de la función exponencial, al incorporar la generalización del factorial en su definición. Su importancia en el cálculo fraccionario es análoga a la importancia que tiene la función exponencial en el cálculo clásico.

1.2.1 | Definición de las funciones de Mittag-Leffler

Partiendo de la serie de Taylor compleja de la función exponencial

$$\mathrm{e}^z = \sum_{k=0}^{\infty} \frac{z^k}{k!} = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(k+1)},$$

donde la igualdad se da gracias al Corolario 1.1.1, siendo esta la base fundamental de las funciones de Mittag-Leffler. De esta manera se tiene:

Definición 1.2.1 Función de Mittag-Leffler de un parámetro

La expresión matemática para la función de Mittag-Leffler es

$$\mathrm{E}_{\alpha}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + 1)}, \tag{1.2.1}$$

donde $\alpha \in \mathbb{C}$. La condición de convergencia en el infinito es $\Re(\alpha) > 0$.

Tomado de: [6], [7]

A partir de la Definición 1.2.1 se pueden notar casos particulares como $\mathrm{E}_1(z) = \mathrm{e}^z$, $\mathrm{E}_{1/2}(z) = \mathrm{e}^{z^2}\mathrm{erfc}(-z)$ y $\mathrm{E}_2(z) = \cosh(\sqrt{z})$, entre otros. Continuando con la generalización, se puede definir:

Definición 1.2.2 Función de Mittag-Leffler de dos parámetros

La expresión matemática para la función de Mittag-Leffler es

$$\mathrm{E}_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{z^k}{\Gamma(\alpha k + \beta)}, \tag{1.2.2}$$

donde $\alpha, \beta \in \mathbb{C}$. La condición de convergencia en el infinito es $\Re(\alpha) > 0$ y $\Re(\beta) > 0$.

Tomado de: [6], [7]

A partir de la Definición 1.2.2 y teniendo en cuenta la Definición 1.2.1, se puede notar la relación $\mathrm{E}_{\alpha,1}(z) = \mathrm{E}_{\alpha}(z)$ como un caso particular de utilidad, no obstante, al igual que en el anterior hay casos particulares para ciertas combinaciones de α y β.

La generalización de las funciones de Mittag-Leffler se puede realizar hasta un número n de parámetros, en este caso particular es pertinente estudiar hasta la función de dos parámetros dada su aplicación en la solución de la ecuación de Bagley-Torvik [6], [7], [24].

1.2.2 | Derivada de las funciones de Mittag-Leffler

En la anterior sección se definió la función de Mittag-Leffer de un parámetro para poder definir la función de Mittag-Leffler de dos parámetros. Teniendo en cuenta que esta segunda es de interés particular, su n-ésima derivada donde $n \in \mathbb{N}$ y para $n = 0$ la función queda inalterada tal como se define la derivada de orden entero, es:

Teorema 1.2.1 Derivada de la función de Mittag-Leffler de dos parámetros

La derivada de orden entero de la función de Mittag-Leffler de dos parámetros, $\mathrm{E}_{\alpha,\beta}(z)$, puede ser expresada como

$$\frac{\mathrm{d}^n}{\mathrm{d}z^n}\mathrm{E}_{\alpha,\beta}(z) = \sum_{k=0}^{\infty} \frac{(k+n)!}{k!\Gamma\left(\alpha\left(k+n\right)+\beta\right)} z^k. \tag{1.2.3}$$

Véase la demostración en: [6, cap. 2, p. 49]

1.3 | Definiciones de la derivada fraccionaria

A diferencia de la derivada de orden entero, no existe una única definición para la derivada fraccionaria, como se mencionó previamente varios autores han entregado distintos aportes a la construcción de la definición de la derivada fraccionaria (por ejemplo la derivada de Grünwald-Letnikov para cálculo numéricos, la derivada de Riesz para aplicaciones en física, entre otros [5]-[7]). En este apartado se revisarán las definiciones más relevantes para el desarrollo de la deducción de la ecuación de Bagley-Torvik.

1.3.1 | Integral fraccionaria

En primera medida es pertinente definir la integral fraccionaria, ya que a partir de ella se construyen las nociones de derivada fraccionaria. Ambas definiciones, la de Riemann-Liouville y la de Caputo, coinciden en su integral asociada, diferenciándose únicamente en el orden de aplicación de los operadores: en Riemann-Liouville se integra primero y luego se deriva, mientras que en Caputo ocurre lo contrario.

La integral fraccionaria parte de la fórmula de integración iterada de Cauchy [5]-[7]

$$\frac{\mathrm{d}^{-n}}{\mathrm{d}t^{-n}}f(t) = \underbrace{\int_0^t \int_0^\tau \cdots \int_0^\tau}_{n \text{ veces}} f(\tau)\, \underbrace{\mathrm{d}\tau_1\, \mathrm{d}\tau_2 \cdots \mathrm{d}\tau_n}_{n \text{ veces}} = \frac{1}{(n-1)!}\int_0^t \frac{f(\tau)}{(t-\tau)^{1-n}}\,\mathrm{d}\tau, \tag{1.3.1}$$

que al encontrar un factorial y usando el Corolario 1.1.1, se generaliza aplicando el Corolario 1.1.1 lo cual dirige a la definición de integral fraccionaria de Riemann-Liouville:

Definición 1.3.1 Integral fraccionaria de Riemann-Liouville

La integral fraccionaria de orden α de una función $f(t)$, está dada por

$$\overset{\mathrm{RL}}{{}_{t_0}}\mathscr{D}_t^{-\alpha} f(t) = \frac{1}{\Gamma(\alpha)}\int_{t_0}^t \frac{f(\tau)}{(t-\tau)^{1-\alpha}}\,\mathrm{d}\tau, \tag{1.3.2}$$

donde $\alpha > 0$ y $t > t_0$.

Tomado de: [5]-[7]

Esta definición coincide con la definición de integral fraccionaria de Caputo en su totalidad:

Definición 1.3.2 Integral fraccionaria de Caputo

La integral fraccionaria de orden γ de una función $f(t)$, está dada por

$$\overset{\mathrm{C}}{{}_{t_0}}\mathscr{D}_t^{-\gamma} f(t) = \frac{1}{\Gamma(\gamma)}\int_{t_0}^t \frac{f(\tau)}{(t-\tau)^{1-\gamma}}\,\mathrm{d}\tau, \tag{1.3.3}$$

donde $\gamma > 0$ y $t > t_0$.

Tomado de: [5]-[7]

1.3.2 | Derivada de Riemann-Liouville

La derivada de Riemann-Liouville es una de las definiciones más antiguas y conocidas del cálculo fraccionario, esta se define como

Definición 1.3.3 Derivada de Riemann-Liouville

La derivada de Riemann-Liouville de orden α de una función $f(t)$, está dada por

$$ {}^{\mathrm{RL}}_{t_0}\mathscr{D}^{\alpha}_{t} f(t) = \frac{1}{\Gamma(n-\alpha)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \int_{t_0}^{t} \frac{f(\tau)}{(t-\tau)^{1+\alpha-n}} \,\mathrm{d}\tau, \tag{1.3.4} $$

donde $n-1 < \alpha < n$ y $n = \lceil \alpha \rceil$, es decir el menor entero mayor o igual que α con $\alpha \in \mathbb{R}$.

Tomado de: [6], [7]

Veamos un ejemplo práctico de la aplicación de esta definición sobre una función $f(t)$. Sea $f(t) = t^{\mu}$ (la función potencia), con $\mu > -1$. Entonces, la derivada de Riemann-Liouville entre 0 y t de orden α se puede calcular como

$$ \begin{aligned} {}^{\mathrm{RL}}_{0}\mathscr{D}^{\alpha}_{t} t^{\mu} &= \frac{1}{\Gamma(n-\alpha)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \int_{0}^{t} \frac{\tau^{\mu}}{(t-\tau)^{1+\alpha-n}} \,\mathrm{d}\tau, \\ &= \frac{1}{\Gamma(n-\alpha)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \underbrace{\left(t^{\mu+n-\alpha} \int_{0}^{1} u^{\mu} (1-u)^{n-\alpha-1} \,\mathrm{d}u \right)}_{\text{Tomando } \tau = tu \text{ y } \mathrm{d}\tau = t\mathrm{d}u}, \\ &= \frac{1}{\Gamma(n-\alpha)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \left(t^{\mu+n-\alpha} \frac{\Gamma(\mu+1)\Gamma(n-\alpha)}{\Gamma(\mu+n-\alpha+1)} \right), \\ &= \frac{1}{\cancel{\Gamma(n-\alpha)}} \frac{\Gamma(\mu+1)\cancel{\Gamma(n-\alpha)}}{\Gamma(\mu+n-\alpha+1)} \frac{\mathrm{d}^n}{\mathrm{d}t^n} \left(t^{\mu+n-\alpha} \right), \\ &= \frac{\Gamma(\mu+1)}{\cancel{\Gamma(\mu+n-\alpha+1)}} \left(\frac{\cancel{\Gamma(\mu+n-\alpha+1)}}{\Gamma(\mu-\alpha+1)} t^{\mu-\alpha} \right), \\ &= \frac{\Gamma(\mu+1)}{\Gamma(\mu-\alpha+1)} t^{\mu-\alpha}. \end{aligned} \tag{1.3.5} $$

Teniendo este resultado, podemos ver qué sucede con la derivada de Riemann-Liouville de la función $f(t) = \mathrm{e}^{\lambda t}$ entre 0 y t de orden α, para ello, partimos de la serie de Taylor de la función exponencial, así

$$ \begin{aligned} {}^{\mathrm{RL}}_{0}\mathscr{D}^{\alpha}_{t} \mathrm{e}^{\lambda t} &= {}^{\mathrm{RL}}_{0}\mathscr{D}^{\alpha}_{t} \left(\sum_{k=0}^{\infty} \frac{(\lambda t)^k}{\Gamma(k+1)} \right), \\ &= \sum_{k=0}^{\infty} \frac{\lambda^k}{\Gamma(k+1)} {}^{\mathrm{RL}}_{0}\mathscr{D}^{\alpha}_{t} t^{k}, \\ &= \sum_{k=0}^{\infty} \frac{\lambda^k}{\cancel{\Gamma(k+1)}} \underbrace{\left(\frac{\cancel{\Gamma(k+1)}}{\Gamma(k-\alpha+1)} t^{k-\alpha} \right)}_{\text{Usando el resultado anterior}}, \\ &= t^{-\alpha} \sum_{k=0}^{\infty} \frac{(\lambda t)^k}{\Gamma(k-\alpha+1)}, \\ &= t^{-\alpha} \mathrm{E}_{1,1-\alpha}(\lambda t). \end{aligned} \tag{1.3.6} $$

Resultado al cual llegamos usando las propiedades de linealidad de la derivada de Riemann-Liouville (véase Teorema 1.5.1) y la definición de la función de Mittag-Leffler de dos parámetros (véase Definición 1.2.2).

1.3.3 | Derivada de Caputo

La derivada de Caputo es otra de las definiciones más relevantes del cálculo fraccionario, esta se define como

Definición 1.3.4 Derivada de Caputo

La derivada de Caputo de orden α de una función $f(t)$, está dada por

$$ {}^{\mathrm{C}}_{t_0}\mathscr{D}^{\alpha}_{t} f(t) = \frac{1}{\Gamma(n-\alpha)} \int_{t_0}^{t} \frac{f^{(n)}(\tau)}{(t-\tau)^{1+\alpha-n}}\, \mathrm{d}\tau, \tag{1.3.7} $$

donde $n = \lceil \alpha \rceil$ es un número entero con $\alpha \in \mathbb{R}$. A su vez, la derivada n-ésima de la función $f(t)$ es continua.

Tomado de: [6], [7]

Tanto la derivada fraccionaria de Riemann-Liouville como la de Caputo se fundamentan en la noción de integral fraccionaria vista previamente, diferenciándose únicamente en el orden de aplicación de los operadores. En el caso de Riemann-Liouville, se procede a derivar una integral fraccionaria, mientras que en el caso de Caputo se integra una derivada entera, lo cual explica las diferencias estructurales entre ambas definiciones aunque compartan la misma base integral [6]-[8].

Nuevamente, veamos un ejemplo práctico de la aplicación de esta definición sobre una función $f(t)$. Sea $f(t) = t^{\mu}$ (la función potencia), con $\mu > -1$. Entonces, la derivada de Caputo entre 0 y t de orden α se puede calcular como

$$ \begin{aligned} {}^{\mathrm{C}}_{0}\mathscr{D}^{\alpha}_{t} t^{\mu} &= \frac{1}{\Gamma(n-\alpha)} \int_{0}^{t} \frac{(\tau^{\mu})^{(n)}}{(t-\tau)^{1+\alpha-n}}\, \mathrm{d}\tau, \\ &= \frac{1}{\Gamma(n-\alpha)} \int_{0}^{t} \frac{\Gamma(\mu+1)}{\Gamma(\mu-n+1)} \tau^{\mu-n} (t-\tau)^{n-\alpha-1}\, \mathrm{d}\tau, \\ &= \frac{\Gamma(\mu+1)}{\Gamma(\mu-n+1)\Gamma(n-\alpha)} \underbrace{\left(t^{\mu-\alpha} \int_{0}^{1} u^{\mu-n} (1-u)^{n-\alpha-1}\, \mathrm{d}u \right)}_{\text{Tomando } \tau = tu \text{ y } \mathrm{d}\tau = t\mathrm{d}u}, \\ &= \frac{\Gamma(\mu+1)}{\cancel{\Gamma(\mu-n+1)\Gamma(n-\alpha)}} \left(\frac{\cancel{\Gamma(\mu-n+1)\Gamma(n-\alpha)}}{\Gamma(\mu-\alpha+1)} t^{\mu-\alpha} \right), \\ &= \frac{\Gamma(\mu+1)}{\Gamma(\mu-\alpha+1)} t^{\mu-\alpha}. \end{aligned} \tag{1.3.8} $$

Notando que el resultado es idéntico al obtenido con la derivada de Riemann-Liouville en (1.3.5). Sin embargo, si se considera la función $f(t) = \mathrm{e}^{\lambda t}$ entre 0 y t de orden α, no necesariamente se obtiene un resultado idéntico al de Riemann-Liouville. Partiendo de la definición de la derivada de Caputo y usando la serie de Taylor de la función exponencial, se tiene

$$ \begin{aligned} {}^{\mathrm{C}}_{0}\mathscr{D}^{\alpha}_{t} \mathrm{e}^{\lambda t} &= \frac{1}{\Gamma(n-\alpha)} \int_{0}^{t} \frac{\left(\mathrm{e}^{\lambda\tau}\right)^{(n)}}{(t-\tau)^{1+\alpha-n}}\, \mathrm{d}\tau, \\ &= \frac{\lambda^{n}}{\Gamma(n-\alpha)} \int_{0}^{t} \frac{\mathrm{e}^{\lambda\tau}}{(t-\tau)^{1+\alpha-n}}\, \mathrm{d}\tau, \\ &= \frac{\lambda^{n}}{\Gamma(n-\alpha)} \int_{0}^{t} \sum_{k=0}^{\infty} \frac{\lambda^{k}}{\Gamma(k+1)} \frac{\tau^{k}}{(t-\tau)^{1+\alpha-n}}\, \mathrm{d}\tau, \\ &= \frac{\lambda^{n}}{\Gamma(n-\alpha)} \sum_{k=0}^{\infty} \frac{\lambda^{k}}{\Gamma(k+1)} \underbrace{\left(t^{k+n-\alpha} \int_{0}^{1} u^{k} (1-u)^{n-\alpha-1}\, \mathrm{d}u \right)}_{\text{Tomando } \tau = tu \text{ y } \mathrm{d}\tau = t\mathrm{d}u}, \\ &= \frac{\lambda^{n} t^{n-\alpha}}{\cancel{\Gamma(n-\alpha)}} \sum_{k=0}^{\infty} \frac{(\lambda t)^{k}}{\cancel{\Gamma(k+1)}} \left(\frac{\cancel{\Gamma(k+1)\Gamma(n-\alpha)}}{\Gamma(k+1+n-\alpha)} \right), \end{aligned} $$

$$= \lambda^n t^{n-\alpha} \sum_{k=0}^{\infty} \frac{(\lambda t)^k}{\Gamma(k+1+n-\alpha)},$$
$$= \lambda^n t^{n-\alpha} \mathrm{E}_{1,1+n-\alpha}(\lambda t). \tag{1.3.9}$$

Resultado al cual se llega usando las propiedades de linealidad de la integral y la definición de la función de Mittag-Leffler de dos parámetros (véase Definición 1.2.2), donde a su vez, se puede evidenciar que es diferente al obtenido con la derivada de Riemann-Liouville en la (1.3.6).

1.4 | Relaciones de la derivada de Riemann-Liouville y la derivada de Caputo

Como se pudo evidenciar, ambas definiciones son similares en su deducción pero distintas en su representación ya que para Riemann-Liouville primero se realiza la integral y luego se deriva, a diferencia de Caputo que realiza el proceso contrario. Existen algunas definiciones las cuales, bajo ciertas condiciones, presentan una relación entre ellas siendo este el caso de Riemann-Liouville y Caputo.

Teorema 1.4.1 Relación entre las definiciones de Caputo y Riemann-Liouville

La relación entre la derivada de Riemann-Liouville y la derivada de Caputo está dada por

$$ {}^{\mathrm{C}}_{t_0}\mathscr{D}_t^{\alpha} f(t) = {}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{\alpha} f(t) - \sum_{k=0}^{n-1} \frac{f^{(k)}(t_0)}{\Gamma(k-\alpha+1)} (t-t_0)^{k-\alpha}, \tag{1.4.1}$$

donde $n-1 < \alpha < n$ y $n = \lceil \alpha \rceil$ con $\alpha \in \mathbb{R}$.

Véase la demostración en: [6, cap. 3, p. 88]

Observaciones:

- Para un problema cuyas condiciones iniciales sean todas nulas, lo que sería $f^{(k)}(t_0) = 0$ para $k = 0, 1, 2, \ldots, n-1$, la derivada de Riemann-Liouville y la derivada de Caputo son equivalentes.
- En el caso que $\alpha < 0$ se tiene que los operadores ya no representan la derivada de una función, sino que representan la integral de orden $|\alpha|$. En este caso ambas definiciones también son equivalentes.

Viendo un caso particular, desarrollemos cuando $t_0 = 0$ y $\alpha = 3/2$, por tanto, $n = 2$, así

$$ {}^{\mathrm{C}}_{0}\mathscr{D}_t^{3/2} f(t) = {}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2} f(t) + \sum_{k=0}^{1} \frac{f^{(k)}(0)}{\Gamma\left(k-\frac{1}{2}\right)} t^{k-\frac{3}{2}} = {}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2} f(t) + \frac{f(0)}{\Gamma\left(-\frac{1}{2}\right)} t^{-\frac{3}{2}} + \frac{f^{(1)}(0)}{\Gamma\left(\frac{1}{2}\right)} t^{-\frac{1}{2}}, \tag{1.4.2}$$

el cual es muy importante ya que se puede evidenciar que en una ecuación diferencial de segundo orden entero y orden fraccional $1 < \alpha < 2$, si la derivada usada es la derivada de Riemann-Liouville, esta no será equivalente a la derivada de Caputo directamente a menos que se considere un residuo de las condiciones iniciales.

1.5 | Algunas propiedades de las derivadas de Riemann-Liouville y Caputo

Al igual que con el operador clásico de derivada, la derivada fraccionaria también cumple ciertas propiedades particulares las cuales serán expresadas a continuación:

Teorema 1.5.1 Propiedad lineal - principio de superposición

La derivada fraccionaria cumple la propiedad lineal, es decir

$$ {}_{t_0}\mathscr{D}_t^{\alpha}\left(f(t)+g(t)\right) = {}_{t_0}\mathscr{D}_t^{\alpha} f(t) + {}_{t_0}\mathscr{D}_t^{\alpha} g(t). \tag{1.5.1} $$

Véase la demostración en: [6, cap. 3, p. 91]

Teorema 1.5.2 Ley de conmutación no se satisface para la derivada de Riemann-Liouville

Si $\alpha, \beta > 0$, y $f(t)$ tiene r derivadas continuas donde $r = \text{máx}\left(\lceil\alpha\rceil, \lceil\beta\rceil\right)$, la derivada secuencial de Riemann-Liouville viene dada de la forma

$$ {}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{\alpha}\left[{}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{\beta} f(t)\right] = {}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{\alpha+\beta} f(t) - \sum_{k=0}^{n-1} \frac{(t-t_0)^{k-\alpha-n}}{\Gamma\left(1+k-n-\alpha\right)} {}^{\mathrm{RL}}_{t_0}\mathscr{D}^{k+\beta-n} f(t)\Big|_{t_0}, \tag{1.5.2} $$

donde $n = \lceil\beta\rceil$, y la derivada secuencial de orden fraccionario no satisface la ley de conmutación.

Véase la demostración en: [6, cap. 3, p. 93]

Corolario 1.5.1 Derivadas secuenciales de orden entero

Si alguna de las derivadas secuenciales es de orden entero, satisface que

$$ \frac{\mathrm{d}^n}{\mathrm{d}t^n}\left[{}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{\alpha} f(t)\right] = {}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{n+\alpha} f(t) \tag{1.5.3} $$

$$ {}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{\alpha}\left[\frac{\mathrm{d}^n}{\mathrm{d}t^n} f(t)\right] = {}^{\mathrm{RL}}_{t_0}\mathscr{D}_t^{n+\alpha} f(t) - \sum_{k=0}^{n-1} \frac{(t-t_0)^{k-\alpha-n}}{\Gamma\left(1+k-n-\alpha\right)} f^{(k)}(t_0). \tag{1.5.4} $$

Véase la demostración en: [6, cap. 3, p. 93]

1.6 | Ecuaciones diferenciales fraccionarias ordinarias (FODE)

Al igual que con el cálculo clásico, se puede proponer ecuaciones diferenciales donde al menos uno de los operadores diferenciales sean de orden no entero, de esta manera, se definen las ecuaciones diferenciales fraccionarias ordinarias (FODE, por sus siglas en inglés) de uno, dos y tres términos.[a]

Particularmente se toman las ecuaciones diferenciales hasta tres términos debido a que la ecuación de Bagley-Torvik es una ecuación diferencial fraccionaria ordinaria de tres términos con coeficientes constantes y lineal [2], [7].

1.6.1 | Ecuación diferencial lineal fraccionaria ordinaria de un término

La ecuación diferencial lineal fraccionaria de un término con coeficientes constantes se define de la forma:

[a] Se usará la siguiente notación: $\mathcal{D}_t^{\alpha}$, para denotar la derivada fraccionaria general respecto a t de orden α.

Definición 1.6.1 ***FODE*** **de un término**

La forma general de una ecuación diferencial lineal fraccionaria de un término con coeficientes constantes es

$$a\mathcal{D}_t^{\alpha} y(t) = u(t). \tag{1.6.1}$$

Tomado de: [6], [7]

1.6.2 | Ecuación diferencial lineal fraccionaria ordinaria de dos términos

La ecuación diferencial lineal fraccionaria de dos términos con coeficientes constantes se define de la forma:

Definición 1.6.2 ***FODE*** **de dos términos**

La forma general de una ecuación diferencial lineal fraccionaria de dos términos con coeficientes constantes es

$$a\mathcal{D}_t^{\alpha} y(t) + by(t) = u(t). \tag{1.6.2}$$

Tomado de: [6], [7]

1.6.3 | Ecuación diferencial lineal fraccionaria ordinaria de tres términos

La ecuación diferencial lineal fraccionaria de tres términos con coeficientes constantes se define de la forma:

Definición 1.6.3 ***FODE*** **de tres términos**

La forma general de una ecuación diferencial lineal fraccionaria de tres términos con coeficientes constantes es

$$a\mathcal{D}_t^{\beta} y(t) + b\mathcal{D}_t^{\alpha} y(t) + cy(t) = u(t). \tag{1.6.3}$$

Tomado de: [6], [7]

1.7 | Solución analítica de ecuaciones diferenciales fraccionarias ordinarias (FODE)

1.7.1 | Convolución de funciones

La convolución es una operación matemática entre dos funciones $f(t)$ y $g(t)$ que produce una tercera $h(t) = f(t) * g(t)$, la cual describe cómo una de ellas modifica o "suaviza" a la otra. En términos simples, mide el grado de superposición entre ambas funciones cuando una se invierte y desplaza en el tiempo o en el espacio.

Definición 1.7.1 Convolución

Dadas dos funciones $f(t)$ y $g(t)$ definidas para todo $t \geq 0$. Se define su **convolución** como

$$h(t) = (f * g)(t) = \int_0^t f(t-\tau)g(\tau)\,\mathrm{d}\tau, \tag{1.7.1}$$

para todo $t \geq 0$.

Tomado de: [6], [7]

1.7.2 | Transformada de Laplace

La transformada de Laplace es una técnica muy popular usada para resolver ecuaciones diferenciales ordinarias y parciales al transformar el problema diferencial en un problema algebraico. Este apartado revisará su definición y ciertas propiedades que serán de utilidad.

1.7.2.1 | Definición

La transformada de Laplace unilateral (la más trabajada) se define como:

Definición 1.7.2 Transformada de Laplace

Suponga $f(t)$ una función definida para $t \geq 0$. La **transformada de Laplace** de $f(t)$, denotada por $\mathscr{L}\{f(t)\}$ o $F(s)$ viene dada de la forma

$$\mathscr{L}\{f(t)\} = F(s) = \int_0^\infty f(t)e^{-st}\,\mathrm{d}t. \tag{1.7.2}$$

Tomado de: [28]

La transformada inversa de Laplace es aquella que al aplicar sobre $F(s)$ devuelve una función $f(t)$, esta puede ser expresada explícitamente sin embargo requiere una complejidad tal que no se abordará en este trabajo [28].

1.7.2.2 | Algunas propiedades útiles

Algunas propiedades de la transformada de Laplace que serán de utilidad son:

Teorema 1.7.1 Existencia de la transformada de Laplace

Suponga una función continua por partes $f(t)$ en el intervalo $[0, \infty)$ y de orden exponencial con $|f(t)| \leq M\mathrm{e}^{at}$ para todo $t \geq 0$. Entonces $\mathscr{L}\{f(t)\}$ existe para todo $s > a$.

Véase la demostración en: [28, cap. 8, p. 480]

Teorema 1.7.2 Linealidad de la transformada de Laplace

Si $f(t)$ y $g(t)$ son funciones y α y β son constantes, entonces

$$\mathscr{L}\{\alpha f(t) + \beta g(t)\} = \alpha\,\mathscr{L}\{f(t)\} + \beta\,\mathscr{L}\{g(t)\}. \tag{1.7.3}$$

Véase la demostración en: [28, cap. 8, p. 482]

Teorema 1.7.3 Producto de las transformadas de Laplace

Si $f(t)$ y $g(t)$ son funciones continuas por partes en $[0,\infty)$ y de orden exponencial, entonces

$$\mathscr{L}\{f(t) * g(t)\} = \mathscr{L}\{f(t)\} \cdot \mathscr{L}\{g(t)\}, \tag{1.7.4}$$

donde $f(t) * g(t)$ es la convolución de $f(t)$ y $g(t)$ dada en la Definición 1.7.1.

Véase la demostración en: [28, cap. 8, pp. 494-495]

Las anteriores propiedades serán suficientes para el trabajo a realizar en la deducción de la ecuación de Bagley-Torvik.

1.7.2.3 | Aplicación en derivadas de orden entero

Como se ha mencionado con anterioridad, una de las ventajas que ofrece la transformada de Laplace es cambiar el problema diferencial por un problema algebraico, no obstante esto es posible siempre y cuando se tenga en general un problema de valor inicial. Así, las transformadas de Laplace para derivadas de orden entero son entonces:

Teorema 1.7.4 Transformada de Laplace de la n-ésima derivada ordinaria de orden entero

Si la transformada de Laplace de $f(t)$ y sus derivadas $f'(t), f''(t), \cdots, f^{(n-1)}, f^{(n)}(t)$ existen, es decir, cumplen el Teorema 1.7.1, entonces

$$\mathscr{L}\left\{f^{(n)}(t)\right\} = s^n \mathscr{L}\{f(t)\} - \sum_{k=0}^{n-1} s^k f^{(k)}(0), \tag{1.7.5}$$

es la transformada de Laplace para la n-ésima derivada de $f(t)$.

Véase la demostración en: [28, cap. 8, p. 483]

Teorema 1.7.5 Transformada de Laplace de la n-ésima derivada parcial de orden entero

Si la transformada de Laplace de $f(x,t)$ y sus derivadas parciales respecto a la variable de integración en la transformada (usualmente t) $\partial_t f(x,t), \partial_{tt} f(x,t), \cdots, \partial_{t^{(n-1)}} f(x,t), \partial_{t^{(n)}} f(x,t)$ existen, es decir, cumplen el Teorema 1.7.1, entonces estas siguen el Teorema 1.7.4; en cambio, si las derivadas son dependientes a una variable distinta a la variable de integración, aplicando el método de derivación bajo signo integral se demuestra que

$$\mathscr{L}\left\{\frac{\partial^{(n)}}{\partial x^{(n)}} f(x,t)\right\} = \frac{\mathrm{d}^{(n)}}{\mathrm{d}x^{(n)}} F(x,s), \tag{1.7.6}$$

es decir, la n-ésima derivada parcial pasa a ser la derivada ordinaria de orden n-ésimo de la transformada de Laplace de $f(x,t)$.[b]

Véase la demostración en: [28, cap. 8, p. 502]

[b] Tenga en cuenta que esto solo se aplica para derivación consecutiva respecto a la misma variable independientemente de la variable de integración (tal como se puede ver en la expansión de la derivada $\partial_{t^{(n)}} f(x,t)$); **no se aplica de igual manera para derivadas parciales mixtas**.

1.7.2.4 | Aplicación en derivadas de orden fraccionario

Debido a que las definiciones de derivada fraccionaria varían, de la misma manera hay varias definiciones para la transformada de Laplace de una derivada fraccionaria, uno para cada definición de derivada. Se tomarán las definiciones de la derivada fraccionaria de Riemann-Liouville (1.3.4) y de Caputo (1.3.7).

Definición 1.7.3 Transformada de Laplace para la derivada de Riemann-Liouville

La transformada de Laplace para la definición de derivada fraccionaria de Riemann-Liouville se define como:

$$\int_0^\infty {}^{\mathrm{RL}}_{0}\mathscr{D}_t^\alpha y(t)\mathrm{e}^{-st}\,\mathrm{d}t = s^\alpha \tilde{y}(s) - \sum_{k=0}^{n-1} s^k \left[{}^{\mathrm{RL}}_{0}\mathscr{D}_t^\alpha y(t)\right]_{t=0} \tag{1.7.7}$$

donde $\tilde{y}(s)$ es la transformada de Laplace de $y(t)$ y $n-1 < \alpha \leq n$.

Tomado de: [7]

Definición 1.7.4 Transformada de Laplace para la derivada de Caputo

La transformada de Laplace para la definición de derivada fraccionaria de Caputo se define como:

$$\int_0^\infty {}^{\mathrm{C}}_{t_0}\mathscr{D}_t^\alpha y(t)\mathrm{e}^{-st}\,\mathrm{d}t = s^\alpha \tilde{y}(s) - \sum_{k=0}^{n-1} s^{\alpha-k-1} y^{(k)}(0) \tag{1.7.8}$$

donde $\tilde{y}(s)$ es la transformada de Laplace de $y(t)$ y $n-1 < \alpha \leq n$.

Tomado de: [7]

Finalmente, a lo que respecta a la transformada inversa de Laplace para cada caso, se aplica como es conocida clásicamente, esto debido a que el problema es algebraico al igual que en el cálculo tradicional.

1.7.3 | Funciones de Green fraccionarias

La noción de función de Green fraccionaria fue introducida por primera vez por S. I. Meshkov y posteriormente desarrollada por K. S. Miller y B. Ross para ecuaciones diferenciales fraccionarias que contienen únicamente derivadas de orden $k\alpha$, donde k es entero [5], [7]. Estas funciones han demostrado ser fundamentales en la construcción de soluciones para problemas de valor inicial de ecuaciones diferenciales fraccionarias lineales con coeficientes constantes, reduciendo la complejidad del problema a la determinación de la función de Green correspondiente.

Una definición más general de la función de Green fraccionaria permite abordar ecuaciones con órdenes de derivación arbitrarios y proporciona representaciones explícitas para casos especiales de gran relevancia práctica.

Considere la ecuación diferencial fraccionaria

$${}_0\mathcal{L}_t y(t) = f(t), \tag{1.7.9}$$

donde ${}_0\mathcal{L}_t$ representa uno o varios operadores diferenciales arbitrarios aplicados sobre $y(t)$. Con condiciones iniciales homogéneas, es decir

$${}_0\mathcal{D}_t^{\sigma_k - 1} y(t)\Big|_{t=0} = 0$$

para $k = 1, \ldots, n$. Las funciones de Green fraccionarias se definen como

Definición 1.7.5 Función de Green fraccionaria

Si la función $G(t,\tau)$ satisface las siguiente condiciones:

1. **Ecuación diferencial homogénea**:

$$ {}_0\mathcal{L}_t G(t,\tau) = 0, $$

para todo $\tau \in (0,t)$.

2. **Condiciones de salto**:

$$ \lim_{\tau \to t^-} \left[{}_\tau\mathcal{D}_t^{\sigma_k - 1} G(t,\tau) \right] = \delta_{k,n}, $$

para $k = 0, 1, \ldots, n$, donde $\delta_{k,n}$ es la función delta de Kronecker.

3. **Condiciones iniciales**:

$$ \lim_{\substack{\tau, t \to 0^+ \\ \tau < t}} \left[{}_\tau\mathcal{D}_t^{\sigma_k} G(t,\tau) \right] = 0, $$

para $k = 0, 1, \ldots, n-1$,

es llamada **función de Green fraccionaria** de la ecuación (1.7.9).

Tomado de: [7]

Las funciones de Green fraccionarias cumplen las siguientes propiedades:

Teorema 1.7.6 Solución de una ecuación diferencial fraccionaria con problema de valor inicial

Tomando nuevamente el problema mostrado en (1.7.9), suponiendo que $G(t,\tau)$ es la función de Green fraccionaria asociada, entonces la solución del problema de valor inicial es

$$ y(t) = \int_0^t G(t,\tau) f(\tau)\, \mathrm{d}\tau. \tag{1.7.10} $$

Véase la demostración en: [7, cap. 5, pp. 150-152]

Teorema 1.7.7 Equivalencia de una función de Green fraccionaria

Sea una ecuación diferencial fraccionaria con coeficientes constantes, entonces

$$ G(t,\tau) \equiv G(t-\tau), \tag{1.7.11} $$

lo cual es consecuencia de que las funciones de Green fraccionarias para este caso, pueden ser obtenidas mediante el método de la transformada de Laplace visto previamente.

Véase la demostración en: [7, cap. 5, p. 152]

Teorema 1.7.8 Derivadas de la función de Green fraccionaria

Las derivadas apropiadas de una función de Green fraccionaria $G(t,\tau)$, forman un conjunto de soluciones linealmente independientes para una ecuación diferencial homogénea ($f(t) = 0$).

Véase la demostración en: [7, cap. 5, pp. 152-153]

Estas propiedades permiten trabajar con ecuaciones diferenciales fraccionarias generalizadas de uno, dos, tres y más términos. Las deducciones de las funciones de Green para ciertos casos se realiza mediante

el método de la transformada de Laplace, particularmente aplicando la transformada inversa a una expresión predeterminada. A continuación se presentan las funciones de Green fraccionarias $G(t,\tau)$ para los diferentes casos mencionados anteriormente:

Teorema 1.7.9 Función de Green fraccionaria para una *FODE* de un término

Tomando la ecuación diferencial fraccionaria de un término (1.6.1), su función de Green fraccionaria está dada por

$$G_1(t) = \frac{1}{a}\frac{t^{\alpha-1}}{\Gamma(\alpha)}. \tag{1.7.12}$$

Aplicando el Teorema 1.7.6, la solución para el problema de valor inicial es entonces

$$y(t) = \frac{1}{a\Gamma(\alpha)}\int_0^t \frac{u(\tau)}{(t-\tau)^{1-\alpha}}\,\mathrm{d}\tau = \frac{1}{a}\,{}_0\mathcal{D}_t^{-\alpha}u(t), \tag{1.7.13}$$

que por coincidencia con (1.3.2) y (1.3.3), resulta la integral fraccionaria generalizada $\mathcal{D}_t^{-\alpha}$.

Véase la demostración en: [7, cap. 5, p. 154]

Teorema 1.7.10 Función de Green fraccionaria para una *FODE* de dos términos

Tomando la ecuación diferencial fraccionaria de dos términos (1.6.2), su función de Green fraccionaria está dada por

$$G_2(t) = \frac{1}{a}t^{\alpha-1}\mathrm{E}_{\alpha,\alpha}\left(-\frac{b}{a}t^{\alpha}\right). \tag{1.7.14}$$

Véase la demostración en: [7, cap. 5, p. 154]

Teorema 1.7.11 Función de Green fraccionaria para una *FODE* de tres términos

Tomando la ecuación diferencial fraccionaria de tres términos (1.6.3), su función de Green fraccionaria está dada por

$$G_3(t) = \frac{1}{a}\sum_{k=0}^{\infty}\frac{(-1)^k}{k!}\left(\frac{c}{a}\right)^k t^{\beta(k+1)-1}\mathrm{E}^{(k)}_{\beta-\alpha,\beta+\alpha k}\left(-\frac{b}{a}t^{\beta-\alpha}\right), \tag{1.7.15}$$

donde $\mathrm{E}^{(k)}_{\beta-\alpha,\beta+\alpha k}(x)$ es la derivada k-ésima de la función de Mittag-Leffler de dos términos presentada en (1.2.3).

Véase la demostración en: [7, cap. 5, p. 155]

Capítulo 2

La ecuación de Bagley-Torvik

Como se mencionó al inicio de este apartado, la ecuación de Bagley-Torvik es uno de los modelos más utilizados la hora de referirse a las aplicaciones directas del cálculo fraccionario en física, en este capítulo se procederá con el planteamiento y deducción de dicha ecuación junto a la solución analítica que ofrece el autor Podlubny en su libro [7].

2.1 | Viscocidad y tensión cortante

Antes de abordar el planteamiento del problema resulta necesario comprender el fenómeno de la viscosidad. Esta es una propiedad característica de los denominados fluidos viscosos, los cuales se oponen al movimiento relativo entre sus capas adyacentes, manifestando así una resistencia interna al flujo.

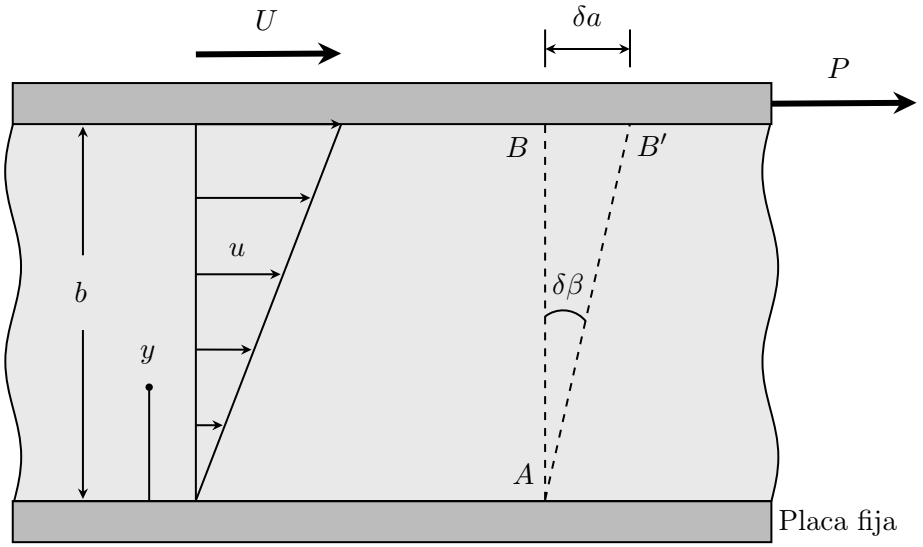


Figura 2.1: Esquema de un fluido contenido entre dos placas paralelas, una fija y otra móvil. La figura corresponde a una adaptación de [29].

En la Figura 2.1 se ilustra el comportamiento general de un fluido sometido a deformación entre dos placas paralelas. Si sobre la placa superior se aplica una fuerza P, esta se desplaza una distancia mínima δa con una velocidad constante U, de modo que la línea vertical $\overline{AB}$ gira un pequeño ángulo $\delta\beta$ hacia la nueva posición $\overline{AB'}$. Para que el fluido resista la fuerza aplicada debe generarse una *tensión cortante* σ en la interfaz entre el fluido y la placa. El equilibrio se logra únicamente cuando $P = \sigma A$, siendo A el área efectiva de la placa móvil. De esta manera, al aplicarse una tensión cortante sobre un fluido, este se deforma de manera continua, en concordancia con su propia definición.

Al continuar con el análisis de la Figura 2.1, se observa que el fluido en contacto con la placa móvil adquiere su misma velocidad U, mientras que aquel en contacto con la placa fija permanece en reposo. Entre ambas superficies, la velocidad varía linealmente con la posición y, de modo que $u = u(y) = Uy/b$. Al derivar esta expresión se obtiene el gradiente de velocidad $\mathrm{d}u/\mathrm{d}y = U/b$, que en este caso es constante, aunque en otros regímenes de flujo puede no serlo.

Durante un incremento de tiempo δt, la línea $\overline{AB}$ gira un ángulo pequeño $\delta\beta$. Al emplear la aproximación de ángulo pequeño, es decir, el primer término del desarrollo de Taylor de $\tan(\delta\beta)$, se obtiene

$$\tan(\delta\beta) \approx \delta\beta = \frac{\delta a}{b},$$

donde δa representa la separación entre los puntos B y B'. Como este desplazamiento ocurre a velocidad U, se cumple que $\delta a = U\delta t$, lo que conduce a

$$\delta\beta = \frac{U}{b}\delta t.$$

Dado que $\delta\beta$ depende no solo de la fuerza aplicada P sino también del tiempo, no resulta adecuado relacionarla de manera directa con la tensión cortante σ. En su lugar, se introduce la *tasa de deformación* $\dot{\gamma}$, definida como

$$\dot{\gamma} = \lim_{\delta t \to 0} \frac{\delta\beta}{\delta t} = \frac{U}{b} = \frac{\mathrm{d}u}{\mathrm{d}y},$$

la cual, en este caso particular, coincide con el gradiente de velocidad. En el experimento se observa que la tensión cortante es directamente proporcional a la tasa de deformación, de modo que $\sigma \propto \mathrm{d}u/\mathrm{d}y$. Al igual que la velocidad, la tensión cortante puede variar en el tiempo en proporción a la variación de aquella.

Definición 2.1.1 Fluido newtoniano

Se denomina ***fluido newtoniano*** a aquel para el cual la relación entre la tensión cortante y el gradiente de velocidad es lineal, es decir,

$$\sigma(z,t) = \mu\frac{\partial v(z,t)}{\partial z}, \tag{2.1.1}$$

donde μ es la **constante de viscosidad** (o simplemente viscosidad) del fluido.

Tomado de: [29]

La mayoría de los fluidos de uso cotidiano, como el agua, el aire, la gasolina y diversos aceites, se comportan de acuerdo con este modelo [29].

2.1.1 | El problema de Rayleigh (primer problema de Stokes)

El problema de Rayleigh, conocido clásicamente como el primer problema de Stokes, constituye un caso canónico en dinámica de fluidos que sirve para describir el establecimiento de un flujo unidireccional en un fluido viscoso provocado por el movimiento de una pared plana [30]. Partiendo de las ecuaciones de Navier-Stokes para un medio homogéneo e incompresible y bajo condiciones de contorno transitorias, la configuración física que se considera es la de un fluido que ocupa la región semiinfinita ($y \geq 0$) delimitada por una pared rígida ubicada en $y = 0$. Inicialmente el fluido está en reposo y, en el instante $t = 0^+$, la pared experimenta un arranque brusco y pasa a desplazarse en sentido paralelo al eje x con una velocidad prescrita $u_w(t) = f(t)$.

Adoptando la hipótesis de flujo puramente unidireccional, $\mathbf{v} = (u(y,t), 0, 0)$, y suponiendo la ausencia de gradientes de presión y de fuerzas de cuerpo en la dirección del movimiento, la dinámica de las

velocidades queda gobernada por la forma reducida de la ecuación de conservación del momento para un fluido newtoniano, es decir,

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial y^2} \quad \text{para } y > 0, t > 0. \tag{2.1.2}$$

Esta ecuación, análoga a la ecuación de difusión [31], expone cómo el momento generado en la pared se transmite hacia el interior del fluido por efecto de la difusión viscosa; en términos constitutivos refleja un modelo local e instantáneo en el que la tensión de cizallamiento depende linealmente del gradiente de velocidad presente, tal y como se indicó en la Definición 2.1.1.

La resolución del problema exige la imposición de condiciones iniciales y de contorno que, en este planteamiento, son las siguientes:

1. **Condición inicial:** reposo absoluto en $t \leq 0$: $u(y, 0) = 0$ para $y \geq 0$.
2. **Condición de contorno en la pared:** $u(0, t) = u_w(t) = f(t)$ para $t > 0$.
3. **Condición en el infinito:** $u(y, t) \to 0$ cuando $y \to \infty$.

El problema de Rayleigh puede visualizarse en la Figura 2.2a, donde se expone de forma esquemática la configuración física y las condiciones de contorno que determinan la evolución temporal y espacial de la velocidad.

2.1.2 | El carácter fraccionario en viscoelástica

La principal limitación del modelo clásico newtoniano, que resulta evidente al aplicar el esquema anterior al problema de Rayleigh, es su incapacidad para capturar la naturaleza hereditaria o de memoria propia de los materiales viscoelásticos [32]. En efecto, las derivadas enteras incorporadas en el modelo clásico representan respuestas de tensión esencialmente instantáneas y locales, mientras que en medios viscoelásticos la tensión presente en un instante depende de manera acumulativa de la historia temporal de la deformación, hecho que exige un tratamiento no local en el tiempo.

El cálculo fraccionario proporciona un marco natural para describir esa memoria decreciente (fading memory) [33], y en el régimen lineal ello se incorpora fácilmente mediante relaciones constitutivas que incluyen operadores diferenciales de orden no entero, por ejemplo $\sigma = \mu_v D_t^q \varepsilon$, donde D_t^q denota la derivada fraccionaria temporal de orden q con $0 < q < 1$ [33]. El parámetro q desempeña en este contexto el papel de un coeficiente reológico que permite modelar comportamientos intermedios entre la elasticidad pura (límite cuando $q \to 0$) y la viscosidad clásica (caso $q = 1$), de modo que el ajuste de q posibilita una interpolación continua entre dichos extremos [32], [34].

La fundamentación teórica de este enfoque quedó sólidamente establecida por Bagley y Torvik en 1983, quienes mostraron que ciertas teorías moleculares para polímeros viscoelásticos conducen a relaciones constitutivas equivalentes a la formulación mediante cálculo fraccionario [3]. Un resultado notable de esa línea de trabajo es la predicción de ordenes fraccionarios específicos, por ejemplo, $q = 1/2$ en algunos modelos moleculares, que inducen un amortiguamiento dependiente de la frecuencia. Cuando este tipo de amortiguamiento fraccionario se acopla a la dinámica de una placa en el contexto del problema de Stokes, surge de forma natural lo que se llamará *la ecuación de Bagley-Torvik*, que actúa como prototipo para el modelado de sistemas que presentan amortiguamiento fraccionario y ofrece una descripción más fiel de la disipación y la memoria viscoelásticas que la provista por los modelos enteros clásicos [2], [4], [34].

2.2 | Planteamiento físico del problema

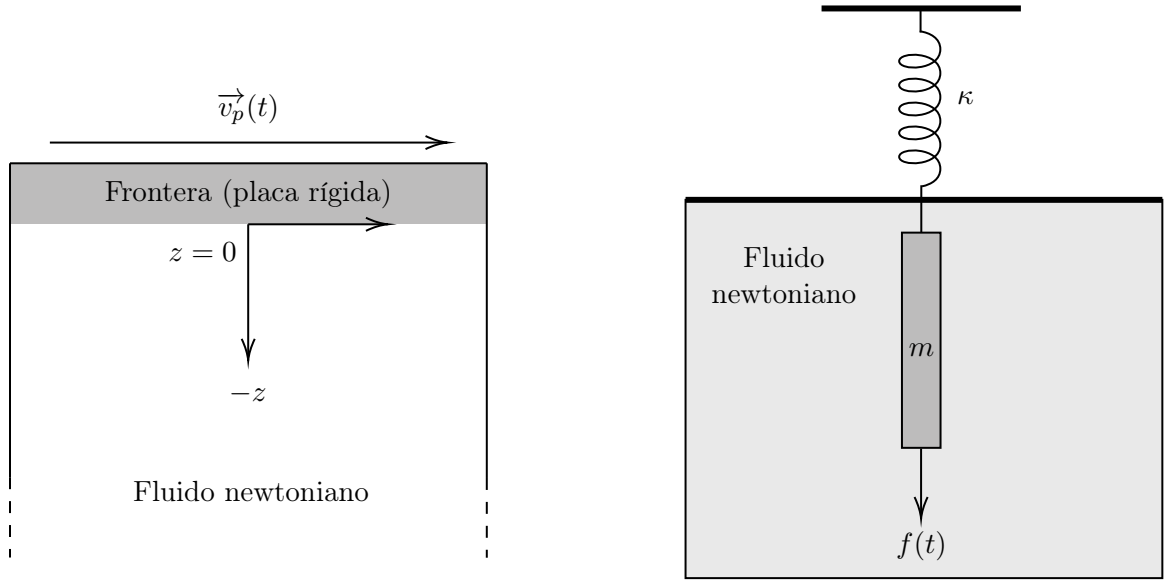
Partiendo del problema de Rayleigh ya visto previamente, se considera un pozo semiinfinito que contiene en su interior un fluido newtoniano. En la parte superior del pozo se encuentra una placa que roza con el fluido, generando una tensión cortante $\sigma(z,t)$, tal como se muestra en la Figura 2.2a. Este primer modelo permite analizar el comportamiento del fluido en la región de contacto con la placa móvil.

Posteriormente, se estudia el caso en que la misma placa, de masa m y área transversal A, está unida a un resorte de constante elástica κ y se encuentra completamente inmersa en el fluido, como se ilustra en la Figura 2.2b.

Las condiciones iniciales del sistema son homogéneas, es decir, tanto la velocidad del fluido como la de la placa son nulas en $t = 0$. En cuanto a las condiciones de frontera para el modelo del pozo, estas son de tipo Dirichlet, ya que se fija el valor de la velocidad del fluido $v(z,t)$:

$$\lim_{z\to 0} v(z,t) = v_p(t), \quad \text{y} \quad \lim_{z\to -\infty} v(z,t) = 0. \tag{2.2.1}$$

Estas condiciones son coherentes con las ya mencionadas en el problema de Rayleigh, con la sutileza que por construcción en este caso se toma la condición de frontera en el límite hacia $z \to -\infty$ a diferencia de lo planteado por Rayleigh. El objetivo es obtener la ecuación de movimiento del sistema masa-resorte-amortiguador, considerando que el amortiguamiento proviene de la acción viscosa del fluido newtoniano, cuyo comportamiento ha sido previamente caracterizado.



(a) Fluido newtoniano contenido en un pozo semiinfinito, con una frontera superior rígida que se desplaza a velocidad $v_p(t)$.

(b) Placa rígida de masa m, unida a un resorte de constante elástica κ e inmersa en un fluido newtoniano.

Figura 2.2: Planteamiento general del problema. Las figuras corresponden a adaptaciones de las presentadas en [2].

2.3 | Deducción matemática

Comenzando con el estudio del primer modelo, se parte de la ecuación (2.1.2) teniendo en cuenta que $\nu = \mu/\rho$ y es conocido como viscocidad cinemática

$$\rho\frac{\partial v}{\partial t} = \mu\frac{\partial^2 v}{\partial z^2},$$

donde ρ es la densidad del fluido, μ es su viscocidad y $v(z,t)$ su velocidad. Para solucionar la ecuación se usará el método de la transformada de Laplace (véase la Definición 1.7.2 y el Teorema 1.7.5) sobre el dominio temporal, esto es

$$\mathscr{L}\left\{\rho\frac{\partial v}{\partial t}\right\} = \mathscr{L}\left\{\mu\frac{\partial^2 v}{\partial z^2}\right\}$$
$$\rho\left(s\tilde{v}(z,s) - v(z,0)\right) = \mu\frac{\mathrm{d}^2\tilde{v}(z,s)}{\mathrm{d}z^2},$$

donde, por construcción de las condiciones iniciales: $v(z,0) = 0$. De esta manera se obtiene la ecuación diferencial ordinaria

$$\frac{\mathrm{d}^2\tilde{v}\,(z,s)}{\mathrm{d}z^2} - \frac{\rho}{\mu}s\tilde{v}\,(z,s) = 0, \tag{2.3.1}$$

suponiendo una solución de la forma $\tilde{v}(z,s) = A\mathrm{e}^{\lambda z}$, y reemplazando

$$\frac{\mathrm{d}^2}{\mathrm{d}z^2}\left(A\mathrm{e}^{\lambda z}\right) - \frac{\rho}{\mu}s\left(A\mathrm{e}^{\lambda z}\right) = 0$$
$$A\lambda^2\mathrm{e}^{\lambda z} - \frac{\rho}{\mu}sA\mathrm{e}^{\lambda z} = 0$$
$$\underbrace{A\mathrm{e}^{\lambda z}}_{A\neq 0}\left(\lambda^2 - \frac{\rho}{\mu}s\right) = 0$$
$$\lambda^2 - \frac{\rho}{\mu}s = 0 \therefore \lambda = \pm\left(\frac{\rho}{\mu}s\right)^{1/2}.$$

Tomando por construcción las condiciones en la frontera (2.2.1), junto a sus respectivas transformadas de Laplace

$$\lim_{z\to-\infty}\tilde{v}(z,s) = A_1\underbrace{\lim_{z\to-\infty}\mathrm{e}^{\left(\frac{\rho}{\mu}s\right)^{1/2}z}}_{0} + A_2\underbrace{\lim_{z\to-\infty}\mathrm{e}^{-\left(\frac{\rho}{\mu}s\right)^{1/2}z}}_{\text{Tiende a }\infty} = A_2\lim_{z\to-\infty}\mathrm{e}^{-\left(\frac{\rho}{\mu}s\right)^{1/2}z} = 0 \therefore A_2 = 0$$
$$\lim_{z\to 0}\tilde{v}(z,s) = A_1\underbrace{\lim_{z\to 0}\mathrm{e}^{\left(\frac{\rho}{\mu}s\right)^{1/2}z}}_{1} = \tilde{v}_p(t) \therefore A_1 = \tilde{v}_p(t)$$

obteniendo la solución de la ecuación diferencial (2.3.1) en el dominio s como

$$\tilde{v}(z,s) = \tilde{v}_p(t)\mathrm{e}^{\left(\frac{\rho}{\mu}s\right)^{1/2}z}.$$

Teniendo en cuenta que se está trabajando con un fluido newtoniano, teniendo en cuenta la Definición 2.1.1, se puede estudiar el rozamiento del fluido con la placa por medio de la tensión cortante $\sigma(z,t)$ entre ellos. Al aplicar la transformada de Laplace sobre (2.1.1) usando el Teorema 1.7.5 se obtiene

$$\sigma(z,t) = \mu\frac{\partial v(z,t)}{\partial z} \overset{\mathscr{L}}{\Longrightarrow} \mathscr{L}\left\{\sigma(z,s)\right\} = \tilde{\sigma}(z,s) = \mu\frac{\mathrm{d}\tilde{v}(z,s)}{\mathrm{d}z},$$

desarrollando la derivada y aplicando la transformada inversa de Laplace

$$\tilde{\sigma}(z,s) = (\mu\rho s)^{1/2}\,\tilde{v}(z,s) \overset{\mathscr{L}^{-1}}{\Longrightarrow} \mathscr{L}^{-1}\left\{\tilde{\sigma}(z,s)\right\} = \sigma(z,t) = (\mu\rho)^{1/2}\,\mathscr{L}^{-1}\left\{\left(\frac{1}{s^{1/2}}\right)(s\tilde{v}(z,s))\right\},$$

a partir del Teorema 1.7.3 se puede demostrar fácilmente que si se aplica la transformada inversa sobre el producto se obtiene la convolución de las funciones; usando entonces la Definición 1.7.1 resulta finalmente que la tensión cortante del fluido inmerso dentro del pozo semiinfinito es

$$\sigma(z,t) = \sqrt{\mu\rho}\left(\frac{1}{\Gamma\left(\frac{1}{2}\right)}\frac{\mathrm{d}}{\mathrm{d}t}\int_0^t \frac{v(\tau,t)}{(t-\tau)^{1/2}}\,\mathrm{d}\tau\right)$$

donde la expresión que está entre paréntesis coincide con (1.3.4) para $\alpha = 1/2$. Aplicando la condición en la frontera $\lim_{z\to 0} v(z,t) = v_p(t)$, se reduce la expresión para la tensión cortante en

$$\sigma(t) = \sqrt{\mu\rho}\,{}^{\mathrm{RL}}_{0}\mathscr{D}_t^{1/2}(v_p(t)). \tag{2.3.2}$$

Tras haber obtenido la tensión cortante se puede continuar con el segundo modelo que deriva en la ecuación de Bagley-Torvik. Realizando un estudio dinámico de lo planteado en la Figura 2.2b, se obtiene la ecuación de movimiento del sistema masa-resorte-amortiguador

$$m\ddot{x}(t) = -\kappa x(t) - 2A\sigma(t) + f(t)\begin{cases}\text{La fuerza elástica } F_e \text{ es } F_e = -\kappa x(t)\\ \text{La fricción viscosa } F_v \text{ es } F_v = -2A\sigma(t)\end{cases}$$

donde m es la masa de la placa, κ es la constante del resorte y A es el área transversal de la placa. Reemplazando la expresión (2.3.2) teniendo en cuenta que $v_p(t) \equiv \dot{x}(t)$ al ser el movimiento de la placa, se obtiene

$$m\ddot{x}(t) + 2A\sqrt{\mu\rho}\,{}^{\mathrm{RL}}_{0}\mathscr{D}_t^{1/2}(\dot{x}(t)) + \kappa x(t) = f(t). \tag{2.3.3}$$

Tomando las condiciones iniciales homogéneas, es decir, $x(0) = 0$ y $\dot{x}(0) = 0$ y gracias al Corolario 1.5.1, particularmente la expresión (1.5.4) resulta finalmente la ecuación de Bagley-Torvik

$$m\ddot{x}(t) + 2A\sqrt{\mu\rho}\,{}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2}(x(t)) + \kappa x(t) = f(t), \tag{2.3.4}$$

luego, gracias al Teorema 1.4.1 se puede expresar la derivada de Riemann-Liouville como una derivada de Caputo, es decir ${}^{\mathrm{RL}}_{0}\mathscr{D}_t^{1/2}(\dot{x}(t)) = {}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2}(x(t)) = {}^{\mathrm{C}}_{0}\mathscr{D}_t^{3/2}(x(t))$.

2.4 | Solución analítica por medio de la función de Green

Podlubny en su libro *Fractional Differential Equations* propone una solución analítica para la ecuación de Bagley-Torvik utilizando la función de Green fraccionaria como solución a una ecuación diferencial fraccionaria ordinaria de tres términos [7]. Según la Definición 1.6.3, tomando las constantes $a = m$, $b = 2A\sqrt{\mu\rho}$, $c = \kappa$; los órdenes de derivación $\beta = 2$ y $\alpha = 3/2$, junto a $u(t) = f(t)$ y usando el Teorema 1.7.11 se puede expresar la solución en términos de la función de Green asociada $G(t)$ de la forma

$$G(t) = \frac{1}{m}\sum_{k=0}^{\infty}\frac{(-1)^k}{k!}\left(\frac{\kappa}{m}\right)^k t^{2k+1}\mathrm{E}^{(k)}_{\frac{1}{2},2+\frac{3}{2}k}\left(-\frac{2A}{m}\sqrt{\mu\rho t}\right), \tag{2.4.1}$$

que finalmente, gracias al Teorema 1.7.6 y dado un problema de valor inicial $x(0) = x_0$ y $\dot{x}(0) = v_0$ se puede expresar la solución analítica para la ecuación de Bagley-Torvik como

$$x(t) = \int_0^t G(t-\tau)f(\tau)\,\mathrm{d}\tau, \tag{2.4.2}$$

donde $G(t-\tau)$ corresponde a la función de Green fraccionaria 2.4.1 trasladada un valor τ y $f(\tau)$ es el forzamiento aplicado al sistema.

Ejemplo: la gráfica a continuación muestra la solución analítica de la ecuación de Bagley-Torvik para el problema solucionado en este trabajo (véase Capítulo 7), usando los valores $m = 1$, $2A\sqrt{\mu\rho} = 0{,}5$, $\kappa = 0{,}5$ y el forzamiento externo

$$f(t) = \begin{cases} 8, & 0 < t < 1, \\ 0, & t \geq 1. \end{cases}$$

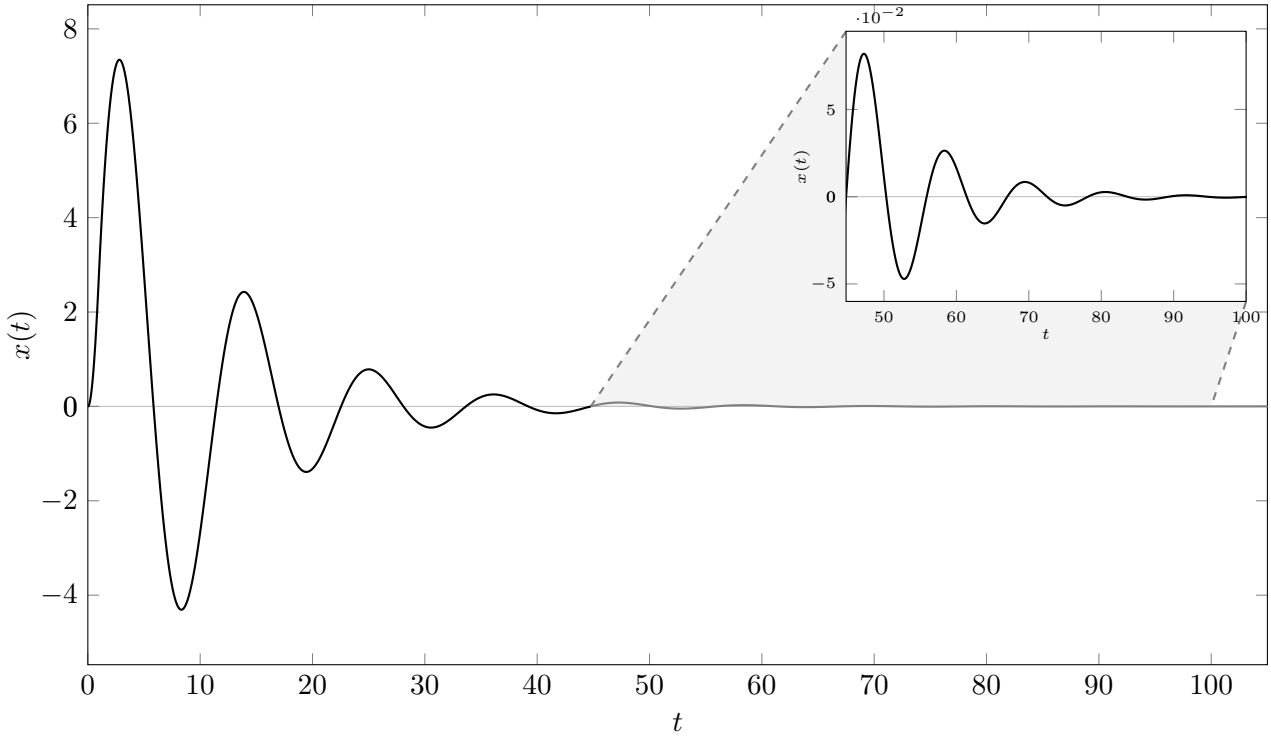


Figura 2.3: Solución analítica de la ecuación de Bagley-Torvik para el problema planteado en el Capítulo 7.

Parte II

Las redes neuronales ortogonales

Las redes neuronales artificiales constituyen uno de los pilares fundamentales de la inteligencia artificial moderna, pues permiten modelar relaciones complejas no lineales y aproximar funciones con alto grado de precisión [9]-[11]. Entre ellas, las *Redes Neuronales Ortogonales* destacan como una arquitectura particular en la que las funciones de activación se construyen a partir de polinomios ortogonales de manera que enfoque combina la potencia de las arquitecturas neuronales con la riqueza matemática de la teoría de los polinomios ortogonales, garantizando propiedades de estabilidad, unicidad y convergencia en el proceso de aprendizaje [13], [14], [16], [35], [36].

En este apartado se presenta el marco conceptual y matemático necesario para comprender el funcionamiento de las redes neuronales ortogonales, junto a métodos de entrenamiento útiles para la misma que fueron usados en la realización de este trabajo.

3 Redes Neuronales Artificiales (ANN) 29
3.1 Definición y fundamentos . . . 29
3.2 Estructura general . . . 29
3.2.1 Neuronas artificiales . . . 31
3.2.2 Capas . . . 31
3.2.3 Funciones de activación . . . 32
3.2.4 Funciones de costo . . . 33
3.3 Entrenamiento . . . 34
3.3.1 Métodos de primer orden . . . 35
3.3.1.1 SGD (Stochastic Gradient Descent) con Momentum . . . 35
3.3.1.2 Adam (Adaptive Moment Estimation) . . . 35
3.3.1.3 AdamW (Adam with Weight Decay) . . . 35
3.3.1.4 RMSprop (Root Mean Square Propagation) . . . 35
3.3.1.5 Adagrad (Adaptive Gradient) . . . 36
3.3.2 Métodos de segundo orden . . . 36
3.3.2.1 BFGS (Broyden-Fletcher-Goldfarb-Shanno) . . . 36
3.3.2.2 L-BFGS-B (Limited-memory BFGS with Bounds) . . . 36
3.3.2.3 Newton-CG (Newton Conjugate Gradient) . . . 37
3.4 Arquitecturas principales . . . 37
3.4.1 El perceptrón de Rosenblatt . . . 37
3.4.1.1 Arquitectura del perceptrón . . . 37
3.4.1.2 Regla de aprendizaje . . . 37
3.4.1.3 Teorema de convergencia . . . 38
3.4.1.4 Su importancia . . . 38
3.4.2 Perceptrón Multicapa (MLP) . . . 38

3.4.3 Otras arquitecturas de redes neuronales populares . . . 39
3.4.3.1 Redes Neuronales Convolucionales (CNN) . . . 39
3.4.3.2 Redes Neuronales Recurrentes (RNN) . . . 39
3.4.3.3 Autoencoders . . . 40
3.4.3.4 Redes Generativas Antagónicas (GANs) . . . 40

4 Polinomios Ortogonales **41**
4.1 Fundamentos . . . 41
4.1.1 Propiedad de ortogonalidad . . . 42
4.1.2 Fórmula de recurrencia . . . 42
4.1.3 Propiedad de los ceros . . . 42
4.1.4 Polinomios ortogonales clásicos . . . 43
4.2 Polinomios de Hermite . . . 44
4.3 Polinomios de Laguerre . . . 45
4.4 Polinomios de Legendre . . . 46

5 Redes Neuronales Ortogonales **48**
5.1 Definición y estructura . . . 48
5.2 Tipos de ONN utilizadas . . . 49
5.2.1 HerNN (Hermite Neural Network) . . . 49
5.2.2 LaNN (Laguerre Neural Network) . . . 49
5.2.3 LeNN (Legendre Neural Network) . . . 50
5.3 Función de costo . . . 50
5.4 Entrenamiento . . . 50

6 Extreme Learning Machine (ELM) **51**
6.1 Definición . . . 51
6.2 Pseudo-inversa generalizada de Moore-Penrose . . . 51
6.3 Funcionamiento y aplicación . . . 54
6.3.1 Funcionamiento . . . 54
6.3.2 Aplicación a una *FODE* de tres términos . . . 55

Capítulo 3

Redes Neuronales Artificiales (ANN)

3.1 | Definición y fundamentos

Las *Redes Neuronales Artificiales* (ANN, por sus siglas en inglés) son modelos computacionales inspirados en la estructura y el funcionamiento del cerebro humano (véase Figura 3.1) [9], [11]. La ambición principal detrás de las ANN es simular la inteligencia artificial a través de la construcción de máquinas cuya arquitectura replique las funciones del sistema nervioso, el cual opera como un computador altamente complejo, no lineal y paralelo [9], [11]. Una red está constituida por unidades fundamentales denominadas *neuronas artificiales* o perceptrones [11], [37], unidades que reciben señales de neuronas predecesoras y transmiten salidas a neuronas subsiguientes a través de interconexiones [37]. La capacidad de procesamiento se determina mediante los pesos ajustables y un sesgo (bias), que son parámetros esenciales de la red [9], [37].

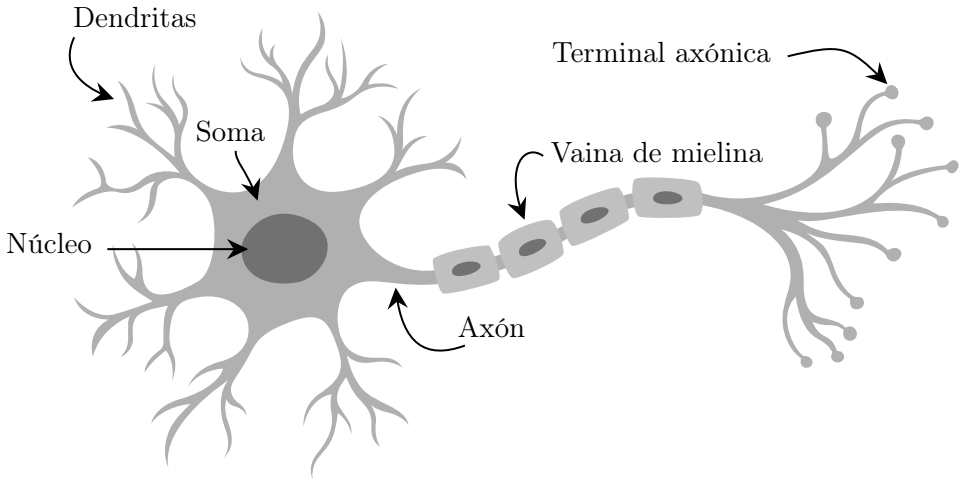
Desde una perspectiva matemática, la red neuronal se puede conceptualizar como un grafo acíclico dirigido en el que los enlaces están parametrizados por los pesos [9]. La funcionalidad central se logra mediante la composición recursiva de funciones simples para construir funciones de aproximación más complejas [9], lo cual establece a las ANN como un método general y práctico para aprender funciones con valores reales, discretos o vectoriales a partir de ejemplos de entrenamiento [12].

El proceso de aprendizaje se formaliza como un problema de optimización continua [9], donde el objetivo principal es minimizar la función de costo o error (L) [9], [38], que cuantifica la discrepancia entre la salida predicha ($\hat{y}$) y el valor deseado (y) [9]. En el contexto de la solución de ecuaciones diferenciales ordinarias (ODE) y parciales (PDE), el método propuesto se basa en la capacidad de aproximación funcional de las redes feedforward, resultando en la construcción de una solución en una forma analítica cerrada y diferenciable [38]; en este esquema, el modelo de función se expresa como la suma de dos términos, donde el segundo término involucra la red neuronal cuyos parámetros (pesos y sesgos) son ajustados para satisfacer la ecuación diferencial y minimizar la cantidad de error derivada directamente de ella [19], [38].

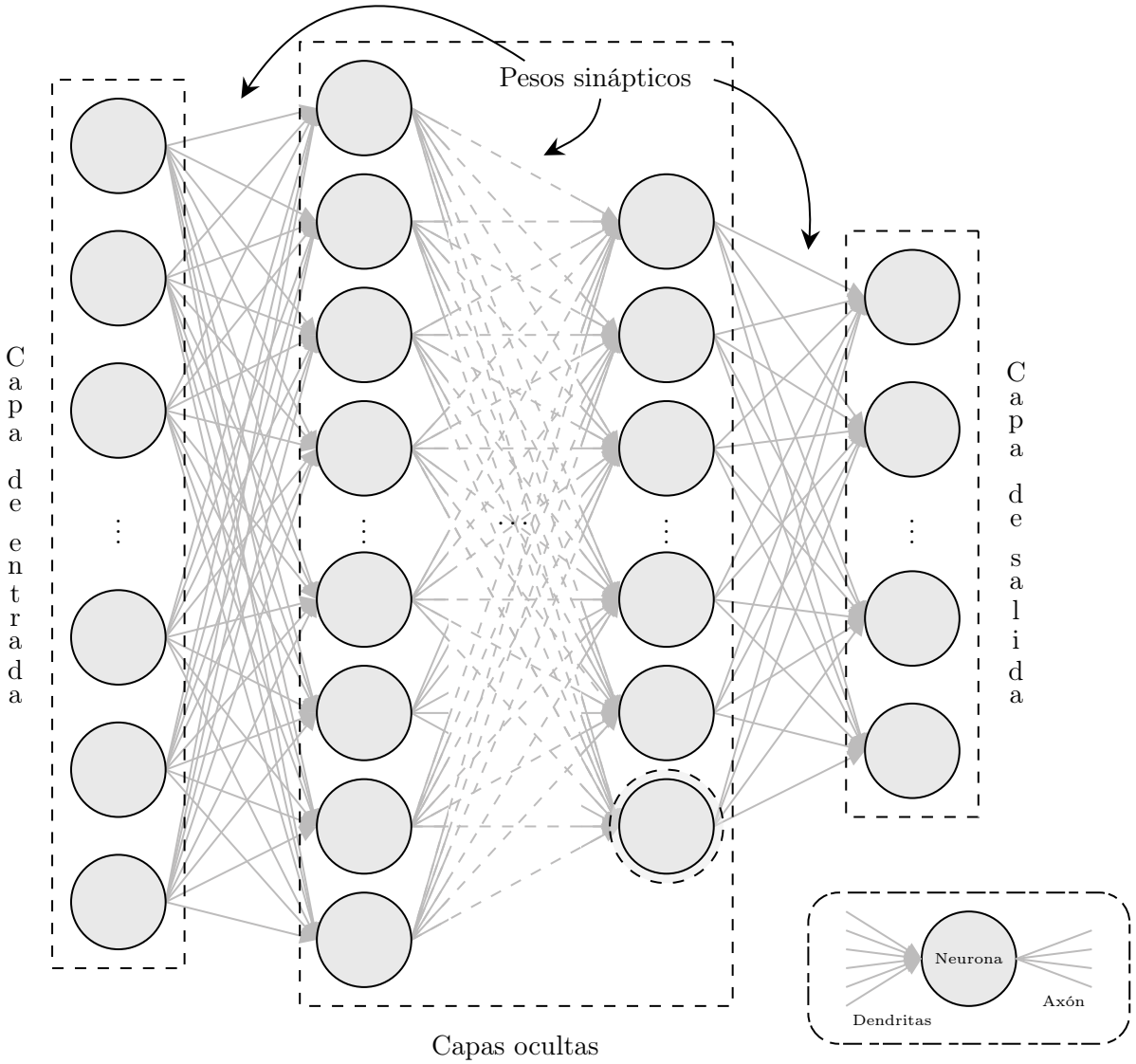
3.2 | Estructura general

La estructura general de las redes neuronales se compone de las siguientes partes:

1. Neuronas artificiales.
2. Capas: de entrada, ocultas y de salida.
3. Funciones de activación y costo.
4. Mecanismos de entrenamiento.



(a) Representación de una neurona biológica señalando sus partes principales.



(b) Esquema de una red neuronal profunda, enfatizando la relación entre sus partes y las partes de una neurona biológica.

Figura 3.1: Inspiración del modelo de *red neuronal* a partir de una neurona biológica. Las figuras corresponden a adaptaciones de las presentadas en [39].

A partir de ellos, se pueden construir diversas arquitecturas de redes neuronales, cada una adaptada a diferentes tipos de problemas y datos [9].

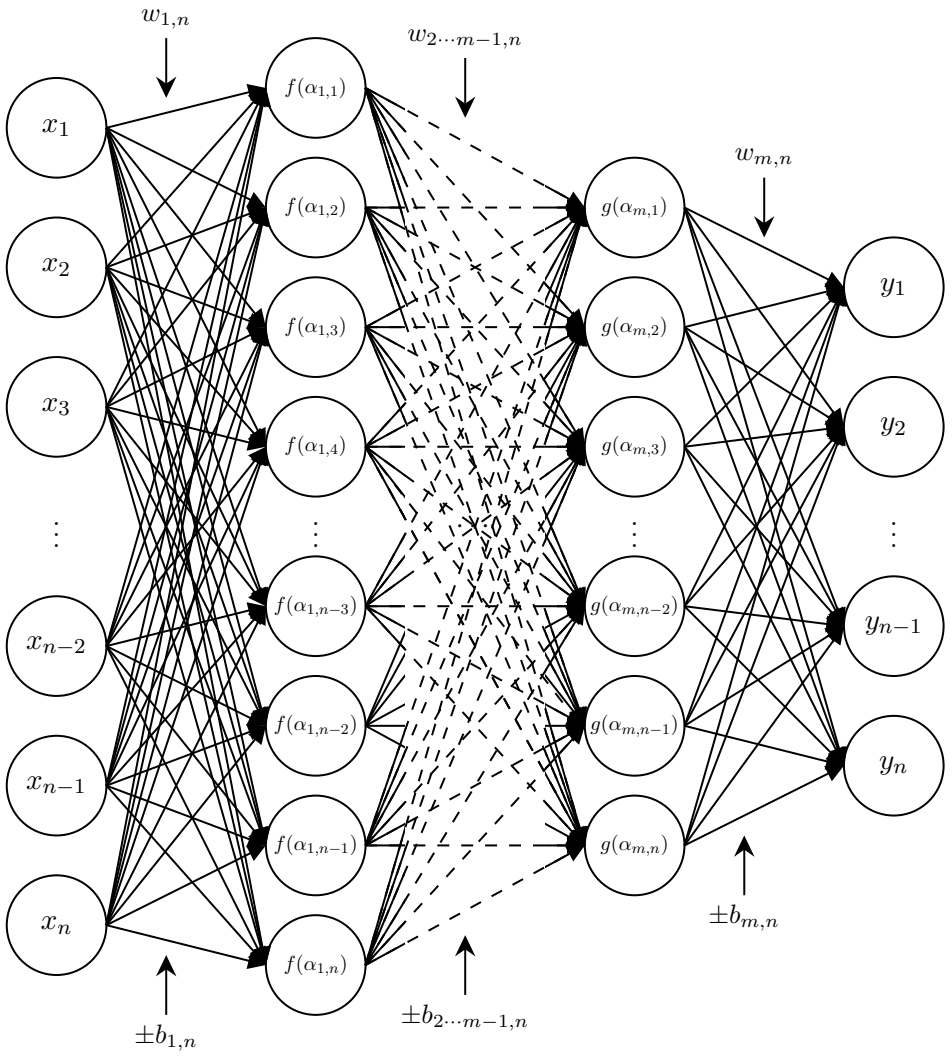


Figura 3.2: Esquema de una red neuronal profunda, señalando sus partes principales en términos de construcción como lo son sus *neuronas de entrada*, *pesos y sesgos*, *funciones de activación* y *neuronas de salida*.

3.2.1 | Neuronas artificiales

Las neuronas artificiales o simplemente neuronas, son las unidades básicas de procesamiento en una red neuronal. Cada neurona recibe múltiples entradas, las cuales son ponderadas por pesos sinápticos y ajustadas por un sesgo. La salida de la neurona se obtiene aplicando una función de activación a la suma ponderada de las entradas más el sesgo.

A su vez, las neuronas son el componente básico que contienen las *capas* de una red neuronal; esto se puede ver ejemplificado en la Figura 3.1b [9].

3.2.2 | Capas

Las capas son conjuntos de neuronas que trabajan juntas para procesar la información en una red neuronal. Existen tres tipos principales de capas:

- **Capa de entrada:** Recibe los datos iniciales y los transmite a las capas ocultas.

- **Capas ocultas:** Realizan el procesamiento intermedio, extrayendo características y patrones de los datos.
- **Capa de salida:** Produce el resultado final de la red, como una clasificación o una predicción.

Esta organización se puede ver ejemplificada en el esquema de la Figura 3.1b [9].

3.2.3 | Funciones de activación

Las funciones de activación permiten a cada neurona realizar un procesamiento de la información, como ya se vio previamente las capas conectan las neuronas en donde, la salida de unas neuronas se convierte en la entrada de otras, esto introduce no linealidades en el modelo, permitiendo a la red aprender relaciones complejas. Las funciones de activación son en general funciones bien definidas y apropiadas para la finalidad de la red neuronal, por ejemplo, en este trabajo las funciones de activación que se utilizaron fueron diversas familias de polinomios ortogonales, sin embargo, existen funciones de activación las cuales son objetivamente aceptadas por la comunidad científica para la mayoría de las casos [9], [11], [12], [38], [40], algunas de ellas son:

Definición 3.2.1 Función signo

La función signo es una función matemática que asigna un valor de 1 o -1 dependiendo del signo del valor de entrada. Se define como:

$$\operatorname{sign}(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x = 0 \\ -1 & \text{si } x < 0 \end{cases} \tag{3.2.1}$$

Tomado de: [9]

Definición 3.2.2 Función sigmoide

La función sigmoide es una función matemática que mapea cualquier valor real a un rango entre 0 y 1. Se define como:

$$\Psi(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{3.2.2}$$

donde $\mathrm{e} \equiv \exp$ es la función exponencial, base del logaritmo natural.

Tomado de: [9]

Definición 3.2.3 Función tangente hiperbólica

La función tangente hiperbólica es una función matemática que mapea cualquier valor real a un rango entre -1 y 1. Se define como:

$$\tanh(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}} = \frac{\mathrm{e}^{2x} - 1}{\mathrm{e}^{2x} + 1}. \tag{3.2.3}$$

Tomado de: [9]

Las anteriores funciones de activación son las más comunes a la hora de introducir no linealidades en las redes neuronales, sin embargo, existen funciones que al contrario, introducen linealidad en las redes neuronales, como por ejemplo:

Definición 3.2.4 Función lineal

La función lineal es una función matemática que mapea cualquier valor real a sí mismo. Se define como:

$$\Psi(x) = x \tag{3.2.4}$$

Tomado de: [9]

Definición 3.2.5 Función ReLU

La función ReLU (Rectified Linear Unit) es una función matemática que mapea cualquier valor real a un rango entre 0 y ∞. Se define como:

$$\text{ReLU}(x) = \text{máx}\{x, 0\} \tag{3.2.5}$$

Tomado de: [9]

Definición 3.2.6 *Hard* tanh

La función "hard tanh" es una variante de la función ReLU muy particular ya que su gráfico es similar al de la función tangente hiperbólico (de ahí su nombre) que mapea cualquier valor real entre -1 y 1 como una función lineal y todo lo que esté por fuera del intervalo es igual a 0. Se define como:

$$\Psi(x) = \text{máx}\{\text{mín}\{x, 1\}, -1\} \tag{3.2.6}$$

Tomado de: [9]

3.2.4 | Funciones de costo

Las funciones de costo son un componente esencial en el entrenamiento de redes neuronales, ya que cuantifican la discrepancia entre las salidas predichas por el modelo y los valores deseados, de esta manera, actúan como guías para el proceso de optimización, permitiendo que los algoritmos de entrenamiento ajusten los parámetros de la red de manera adecuada.

A continuación se presentan algunas de las funciones de costo más utilizadas:

Definición 3.2.7 Error Cuadrático Medio (MSE)

El error cuadrático medio es una función de costo utilizada principalmente en problemas de regresión. Calcula el promedio del cuadrado de las diferencias entre las salidas predichas $\hat{y}_i$ y los valores reales y_i:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{3.2.7}$$

Tomado de: [11]

Definición 3.2.8 Entropía Cruzada Binaria

La entropía cruzada binaria se emplea en tareas de clasificación binaria. Penaliza fuertemente las predicciones erróneas con alta confianza:

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right). \tag{3.2.8}$$

Tomado de: [12]

Definición 3.2.9 Entropía Cruzada Categórica

La entropía cruzada categórica se utiliza en problemas de clasificación multiclase, midiendo la discrepancia entre la distribución verdadera y la predicha por el modelo:

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log \hat{y}_{ik}. \tag{3.2.9}$$

Tomado de: [10]

Definición 3.2.10 Función Hinge

La función Hinge se aplica principalmente en clasificadores de margen, aunque también se utiliza en redes neuronales binarias. Su forma es:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i \hat{y}_i). \tag{3.2.10}$$

Tomado de: [9]

Definición 3.2.11 Divergencia de Kullback-Leibler (KL)

La divergencia KL mide la diferencia entre dos distribuciones de probabilidad P y Q. Es ampliamente usada en modelos probabilísticos y autoencoders variacionales:

$$D_{KL}(P \parallel Q) = \sum_{x} P(x) \log \frac{P(x)}{Q(x)}. \tag{3.2.11}$$

Tomado de: [10]

3.3 | Entrenamiento

El proceso de entrenamiento de una red neuronal se fundamenta en la optimización iterativa de sus parámetros, con el propósito de minimizar una función de costo que cuantifica el error de predicción. Para llevar a cabo esta tarea se emplean algoritmos de optimización numérica que, dependiendo de la información que aprovechan de la función de costo, se pueden organizar en dos grandes familias: los métodos de primer orden y los métodos de segundo orden.[c]

[c] Todos los métodos pertenecientes a esta sección fueron tomados de varios autores, se presenta un resumen de aquellos métodos usados en este trabajo para el entrenamiento de la red. Las referencias se encuentran al inicio de cada sección.

3.3.1 | Métodos de primer orden

Los métodos de primer orden utilizan únicamente la información del gradiente de la función de costo respecto a los parámetros de la red. En cada iteración, los pesos se actualizan siguiendo la dirección opuesta al gradiente, ya que este indica la dirección de máximo incremento del error [9], [11], [12], [41]-[43].

3.3.1.1 | SGD (Stochastic Gradient Descent) con Momentum

El descenso estocástico por gradiente con momentum acumula información de los gradientes anteriores en una variable de velocidad, lo que permite suavizar la trayectoria de actualización de los parámetros.

$$\begin{aligned} \mathbf{v}_t &= \beta \mathbf{v}_{t-1} + (1-\beta)\nabla f(\mathbf{w}_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \mathbf{v}_t \end{aligned}$$

donde $\mathbf{v}_t$ es la velocidad acumulada, β el coeficiente de momentum, $\nabla f(\mathbf{w}_t)$ el gradiente del costo respecto a los pesos en el paso t, η la tasa de aprendizaje y $\mathbf{w}_t$ el vector de parámetros.

3.3.1.2 | Adam (Adaptive Moment Estimation)

Adam combina la estimación de momentos de primer y segundo orden del gradiente, aplicando una corrección por sesgo en cada paso.

$$\begin{aligned} \mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\nabla f(\mathbf{w}_t) \\ \mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)[\nabla f(\mathbf{w}_t)]^2 \\ \hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1-\beta_1^t}, \quad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \end{aligned}$$

donde $\mathbf{m}_t$ es la estimación del primer momento (media de gradientes), $\mathbf{v}_t$ la del segundo momento (media de gradientes al cuadrado), β_1 y β_2 los coeficientes de decaimiento exponencial, $\hat{\mathbf{m}}_t$ y $\hat{\mathbf{v}}_t$ las versiones corregidas por sesgo, η la tasa de aprendizaje, ϵ un valor pequeño para estabilidad numérica y $\mathbf{w}_t$ los parámetros del modelo.

3.3.1.3 | AdamW (Adam with Weight Decay)

AdamW extiende Adam incorporando un término de decaimiento de pesos desacoplado de la estimación de momentos.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} + \lambda \mathbf{w}_t \right)$$

donde λ es el coeficiente de decaimiento de pesos, y el resto de los términos mantienen el mismo significado que en Adam.

3.3.1.4 | RMSprop (Root Mean Square Propagation)

RMSprop ajusta la tasa de aprendizaje dividiendo el gradiente por la raíz cuadrada de una media móvil de los cuadrados de gradientes pasados.

$$\mathbf{v}_t = \beta \mathbf{v}_{t-1} + (1-\beta)[\nabla f(\mathbf{w}_t)]^2$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{\mathbf{v}_t + \epsilon}} \nabla f(\mathbf{w}_t)$$

donde $\mathbf{v}_t$ representa la media móvil de los cuadrados de gradientes, β controla el grado de suavizado, η es la tasa de aprendizaje, ϵ evita divisiones por cero y $\mathbf{w}_t$ los parámetros actualizados.

3.3.1.5 | Adagrad (Adaptive Gradient)

Adagrad acumula los cuadrados de todos los gradientes históricos y ajusta cada paso en función de esta suma acumulada.

$$\mathbf{G}_t = \mathbf{G}_{t-1} + [\nabla f(\mathbf{w}_t)]^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{\mathbf{G}_t + \epsilon}} \nabla f(\mathbf{w}_t)$$

donde $\mathbf{G}_t$ es la suma acumulada de los cuadrados de gradientes, η la tasa de aprendizaje, ϵ un término de estabilidad numérica y $\mathbf{w}_t$ los parámetros del modelo.

3.3.2 | Métodos de segundo orden

Los métodos de segundo orden utilizan información del gradiente y de la curvatura de la función de costo, expresada mediante la matriz Hessiana o aproximaciones de la misma donde la matriz Hessiana es aquella conformada por las segundas derivadas de la función, a diferencia del gradiente que es la primera derivada, de ahí el nombre de *métodos de segundo orden* a comparación de los de primer orden. Esto permite calcular pasos de actualización que consideran tanto la dirección como la curvatura de la superficie de error [9], [11], [12], [42].

3.3.2.1 | BFGS (Broyden-Fletcher-Goldfarb-Shanno)

BFGS mantiene y actualiza iterativamente una aproximación de la matriz Hessiana inversa, mejorando la dirección de actualización en cada paso.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{B}_t^{-1} \nabla f(\mathbf{w}_t)$$
$$\mathbf{B}_{t+1} = \mathbf{B}_t + \frac{\mathbf{y}_t \mathbf{y}_t^T}{\mathbf{y}_t^T \mathbf{s}_t} - \frac{\mathbf{B}_t \mathbf{s}_t \mathbf{s}_t^T \mathbf{B}_t}{\mathbf{s}_t^T \mathbf{B}_t \mathbf{s}_t}$$

donde $\mathbf{w}_t$ es el vector de parámetros, η la tasa de aprendizaje, $\mathbf{B}_t$ una aproximación de la matriz Hessiana (segunda derivada del costo), $\nabla f(\mathbf{w}_t)$ el gradiente del costo, $\mathbf{s}_t = \mathbf{w}_{t+1} - \mathbf{w}_t$ representa el cambio en los parámetros entre iteraciones y $\mathbf{y}_t = \nabla f(\mathbf{w}_{t+1}) - \nabla f(\mathbf{w}_t)$ el cambio en los gradientes consecutivos.

3.3.2.2 | L-BFGS-B (Limited-memory BFGS with Bounds)

L-BFGS-B es una variante de BFGS que almacena solo un número limitado de pares de actualizaciones recientes, reduciendo el costo de memoria.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{H}_t \nabla f(\mathbf{w}_t)$$

donde $\mathbf{w}_t$ son los parámetros, η la tasa de aprendizaje, $\mathbf{H}_t$ una aproximación limitada de la matriz Hessiana inversa construida recursivamente usando únicamente los últimos m pares $(\mathbf{s}_i, \mathbf{y}_i)$, y $\nabla f(\mathbf{w}_t)$ el gradiente del costo en el paso t.

3.3.2.3 | Newton-CG (Newton Conjugate Gradient)

Newton-CG utiliza el método de Newton, resolviendo de manera aproximada el sistema lineal definido por el Hessiano mediante gradiente conjugado, en lugar de invertirlo de forma directa.

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{H}_t^{-1} \nabla f(\mathbf{w}_t)$$
$$\mathbf{H}_t \mathbf{d}_t = -\nabla f(\mathbf{w}_t)$$

donde $\mathbf{w}_t$ son los parámetros, η la tasa de aprendizaje, $\mathbf{H}_t$ el Hessiano del costo, $\nabla f(\mathbf{w}_t)$ el gradiente, y $\mathbf{d}_t$ la dirección de descenso obtenida mediante el método de gradiente conjugado que aproxima la solución de $\mathbf{H}_t^{-1} \nabla f(\mathbf{w}_t)$.

3.4 | Arquitecturas principales

3.4.1 | El perceptrón de Rosenblatt

El perceptrón de Rosenblatt, propuesto en 1958, se reconoce como la primera arquitectura formal de red neuronal artificial [44]. Su diseño se inspiró en la organización neuronal biológica, con el propósito de construir un modelo computacional capaz de aprender patrones a partir de ejemplos. El perceptrón constituye un clasificador lineal que transforma un vector de entrada $\mathbf{x} \in \mathbb{R}^n$ en una salida binaria mediante una combinación lineal de sus componentes seguida de una función de activación de umbral [9], [11], [12].

3.4.1.1 | Arquitectura del perceptrón

La estructura básica del perceptrón consiste en un conjunto de n entradas ponderadas por los coeficientes sinápticos w_i, los cuales se suman junto con un sesgo b. Matemáticamente, la salida del perceptrón puede escribirse como:

$$y = \phi(\mathbf{w}^T \mathbf{x} + b) \tag{3.4.1}$$

donde $\phi(\cdot)$ es una función de activación de tipo escalón:

$$\phi(v) = \begin{cases} 1, & v > 0 \\ 0, & v \leq 0 \end{cases} \tag{3.4.2}$$

Este mecanismo convierte al perceptrón en un separador lineal de patrones, dividiendo el espacio de entrada en dos regiones definidas por un hiperplano [11], [44].

3.4.1.2 | Regla de aprendizaje

El aprendizaje en el perceptrón se implementa ajustando los pesos de manera iterativa a partir de ejemplos de entrenamiento. La regla de actualización de pesos se formula como:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \left(d(k) - y(k)\right) \mathbf{x}(k) \tag{3.4.3}$$
$$b(k+1) = b(k) + \eta \left(d(k) - y(k)\right) \tag{3.4.4}$$

donde $d(k)$ es la salida deseada para el patrón $\mathbf{x}(k)$, $y(k)$ es la salida producida por el perceptrón, y $\eta > 0$ es la tasa de aprendizaje [11], [44].

3.4.1.3 | Teorema de convergencia

El *teorema de convergencia del perceptrón*, demostrado por Rosenblatt [44] y posteriormente formalizado por Novikoff, establece que si el conjunto de patrones de entrenamiento es linealmente separable, el algoritmo de aprendizaje del perceptrón converge en un número finito de pasos hacia una solución que clasifica correctamente todos los ejemplos [11]. Este resultado confiere al perceptrón una base matemática sólida como modelo de aprendizaje supervisado.

Teorema 3.4.1 Teorema de convergencia del perceptrón

Sea un conjunto finito de pares de entrenamiento $\{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$ con $\mathbf{x}_i \in \mathbb{R}^m$ y $d_i \in \{+1, -1\}$. Supongamos que las dos clases son *linealmente separables*, es decir, existe un vector $\mathbf{w}^* \in \mathbb{R}^m$ y un sesgo b^* tales que

$$d_i \left(\mathbf{w}^{*T}\mathbf{x}_i + b^*\right) \geq \gamma > 0 \qquad \text{para todo } i = 1, \ldots, N,$$

donde γ es la *margen* mínima separadora. Sea además

$$R \coloneqq \max_{1 \leq i \leq N} \|\mathbf{x}_i\|.$$

Consideremos el algoritmo del perceptrón con regla de corrección (con tasa de aprendizaje $\eta > 0$, y con la convención de incorporar el sesgo en el vector de pesos mediante $\tilde{\mathbf{x}} = (1, \mathbf{x})$, $\tilde{\mathbf{w}} = (b, \mathbf{w})$):

$$\tilde{\mathbf{w}}(t+1) = \tilde{\mathbf{w}}(t) + \eta\, d_i \tilde{\mathbf{x}}_i \quad \text{cuando } d_i\, \tilde{\mathbf{w}}(t)^T \tilde{\mathbf{x}}_i \leq 0.$$

Entonces, el algoritmo converge en un número finito de actualizaciones: existe un entero $T_{\text{máx}} < \infty$ tal que para algún $t \leq T_{\text{máx}}$ se alcanza un vector de pesos que clasifica correctamente todos los patrones de entrenamiento (es decir, $d_i\, \tilde{\mathbf{w}}(t)^T \tilde{\mathbf{x}}_i > 0$ para todo i). Además, si se elige $\eta = 1$, una cota superior para el número de actualizaciones $T_{\text{máx}}$ viene dada por la desigualdad de Novikoff:

$$T_{\text{máx}} \leq \left(\frac{R}{\gamma}\right)^2.$$

Véase la demostración en: [11, cap. 1, pp. 50-55]

3.4.1.4 | Su importancia

A pesar de su simplicidad, el perceptrón de Rosenblatt tuvo un papel histórico fundamental en el desarrollo de las redes neuronales. Representó la primera demostración práctica de que un sistema inspirado en el cerebro humano podía aprender a clasificar patrones sin intervención explícita. Su capacidad de resolver problemas de clasificación linealmente separables lo convirtió en una herramienta pionera en aplicaciones iniciales de reconocimiento de patrones y procesamiento de señales. Además, sentó las bases conceptuales y algorítmicas sobre las que se desarrollaron arquitecturas más avanzadas de redes neuronales en décadas posteriores.

3.4.2 | Perceptrón Multicapa (MLP)

El perceptrón multicapa (MLP, por sus siglas en inglés) constituye una extensión directa del perceptrón de Rosenblatt. Su diseño incorpora una o varias capas ocultas de neuronas entre la entrada y la salida (véase la Figura 3.2), lo que le permite aproximar funciones no lineales y superar la limitación de separación lineal inherente al perceptrón simple [11], [12].

Basada en la conformación de la arquitectura, cada neurona en las capas ocultas aplica una transformación no lineal a una combinación lineal de sus entradas, introduciendo de este modo capacidad de representación compleja lo que permite expresar matemáticamente la red de la forma

$$y_k = f\left(\sum_{j=1}^{m_1} w_{kj}^{(2)}\,\phi\left(\sum_{i=1}^{m_0} w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right), \tag{3.4.5}$$

donde $x_i \in \mathbb{R}^{m_0}$ son las entradas, $w_{ji}^{(\cdot)}$ y $b_j^{(\cdot)}$ son los pesos y sesgos tanto de la capa de entrada, como de las capas ocultas, como de la salida, y $\phi(\cdot)$ y $f(\cdot)$ son funciones de activación aplicadas tanto en cada capa oculta, como en la salida (véase la subsección 3.2.3) [11], [12], [38].

El entrenamiento del MLP se realiza mediante el algoritmo de *retropropagación del error*, que aplica la regla de la cadena para propagar gradientes hacia atrás a través de las capas [11], [42]. Dado que este proceso se basa en el cálculo de gradientes de primer orden, el MLP se entrena principalmente con métodos de primer orden (véase la subsección 3.3.1).

3.4.3 | Otras arquitecturas de redes neuronales populares

A continuación se presentan otras arquitecturas de redes neuronales que si bien es cierto son más específicas y especializadas para algunos campos y no fueron utilizadas en el trabajo, mencionarlas robustece las aplicaciones reales de las redes neuronales.

3.4.3.1 | Redes Neuronales Convolucionales (CNN)

Las redes convolucionales (CNN) fueron diseñadas para procesar datos con estructura espacial, como imágenes o señales bidimensionales en las cuales sustituyen las conexiones totalmente densas por filtros locales (*kernels*) que se desplazan sobre la entrada, extrayendo características relevantes de manera jerárquica [40]. Así, matemáticamente, una capa convolucional se expresa como:

$$X^{(l+1)} = \sigma\left(P\left(X^{(l)} \otimes K^{(l+1)} + b^{(l+1)}\right)\right), \tag{3.4.6}$$

donde $\otimes$ denota la convolución, $K^{(l+1)}$ son los filtros, $b^{(l+1)}$ los sesgos, P la operación de *pooling*, y σ una función de activación. El entrenamiento de las CNN también se realiza con métodos de primer orden, utilizando retropropagación adaptada a las operaciones de convolución.

3.4.3.2 | Redes Neuronales Recurrentes (RNN)

Las redes recurrentes están diseñadas para procesar datos secuenciales, como texto, series temporales o señales de voz, donde incorporan conexiones de retroalimentación que permiten mantener un estado oculto dependiente de entradas anteriores [40]. La dinámica básica de un RNN está dada por:

$$h_{t+1} = \sigma\left(W_x x_t + W_h h_t + b_h\right), \tag{3.4.7}$$

$$y_{t+1} = W_y h_t + b_y, \tag{3.4.8}$$

donde h_t es el estado oculto en el tiempo t, x_t la entrada, y y_t la salida y su entrenamiento se realiza mediante *Backpropagation Through Time* (BPTT), un algoritmo de gradiente de primer orden.

3.4.3.3 | Autoencoders

Un autoencoder es una red neuronal diseñada para aprender representaciones latentes comprimidas de los datos. Consta de un codificador, que proyecta la entrada a un espacio de menor dimensión, y un decodificador, que intenta reconstruir la entrada a partir de dicha representación [11] lo que se expresa matemáticamente como:

$$z = \phi(W_e x + b_e), \tag{3.4.9}$$

$$\hat{x} = \phi(W_d z + b_d), \tag{3.4.10}$$

donde (W_e, b_e) corresponden al codificador y (W_d, b_d) al decodificador. Al igual que con las anteriores, su entrenamiento minimiza el error de reconstrucción, y se realiza con métodos de primer orden mediante retropropagación.

3.4.3.4 | Redes Generativas Antagónicas (GANs)

Las GANs constan de dos redes entrenadas simultáneamente: un generador $G(z)$ que produce muestras sintéticas a partir de un vector de ruido, y un discriminador $D(x)$ que estima la probabilidad de que una muestra provenga de los datos reales [40, cap. 1]. Su formulación se plantea como un problema min-max:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)} \left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)} \left[\log(1 - D(G(z)))\right]. \tag{3.4.11}$$

Finalmente, su entrenamiento se basa en métodos de primer orden, utilizando descenso por gradiente estocástico alternado para ambos modelos (véase subsección 3.3.1).

Capítulo 4

Polinomios Ortogonales

Los polinomios ortogonales constituyen una clase especial de polinomios que se caracterizan por satisfacer una propiedad de ortogonalidad con respecto a una función peso en un intervalo específico, o, de manera más general, a un funcional de momentos. Esta propiedad fundamental permite estudiar de manera sistemática estructuras algebraicas y analíticas que emergen en diversas áreas del análisis matemático.

4.1 | Fundamentos

Para comprender la teoría de los polinomios ortogonales es necesario primero introducir los conceptos que permiten formalizar la ortogonalidad algebraica, comenzando con los funcionales de momentos, que constituyen la base para definir y estudiar estas sucesiones de polinomios.

Definición 4.1.1 Funcional de momentos

Sea $\{\mu_n\}_{n=0}^{\infty}$ una sucesión de números complejos y sea $\mathfrak{L}$ una función definida sobre el espacio vectorial de todos los polinomios, tal que cumple

$$\mathfrak{L}[x^n] = \mu_n, \quad n = 0, 1, 2, \ldots, \tag{4.1.1}$$

$$\mathfrak{L}[\alpha_1\pi_1(x) + \alpha_2\pi_2(x)] = \alpha_1\mathfrak{L}[\pi_1(x)] + \alpha_2\mathfrak{L}[\pi_2(x)] \tag{4.1.2}$$

para todos los números complejos α_1, α_2 y todos los polinomios $\pi_1(x), \pi_2(x)$. En este caso, $\mathfrak{L}$ se denomina *funcional de momentos* determinado por la sucesión formal de momentos $\{\mu_n\}$. Cada número μ_n recibe el nombre de *momento de orden n*.

Tomado de: [14]

Estos momentos de orden n corresponden a los valores que entrega la aplicación del funcional sobre una *sucesión de polinomios ortogonales*. Partiendo de un funcional de momentos, se puede construir una sucesión de polinomios que satisfacen una relación de ortogonalidad estricta, conocida como polinomios ortogonales. Esta relación se formaliza en la siguiente definición:

Definición 4.1.2 Sucesión de polinomios ortogonales

Una sucesión $\{P_n(x)\}_{n=0}^{\infty}$ se denomina *sucesión de polinomios ortogonales* con respecto a un funcional de momentos $\mathfrak{L}$ si, para todos los enteros no negativos m y n, se cumplen las siguientes condiciones:

(I) Cada $P_n(x)$ es un polinomio de grado n,
(II) $\mathfrak{L}[P_m(x)P_n(x)] = 0$ siempre que $m \neq n$,
(III) $\mathfrak{L}[P_n^2(x)] \neq 0$.

Tomado de: [14]

4.1.1 | Propiedad de ortogonalidad

La Definición 4.1.2 introduce una primera noción de ortogonalidad mediante la acción del funcional de momentos sobre una sucesión de polinomios, sin embargo, resulta conveniente establecer de manera más precisa la condición que garantiza dicha ortogonalidad, lo cual se formaliza en el siguiente teorema:

Teorema 4.1.1 Ortogonalidad de los polinomios

Sea $\mathfrak{L}$ un funcional de momentos y sea $\{P_n(x)\}$ una sucesión de polinomios. Entonces, las siguientes condiciones son equivalentes:

(a) La sucesión $\{P_n(x)\}$ es una sucesión de polinomios ortogonales respecto a $\mathfrak{L}$;
(b) Se cumple que $\mathfrak{L}[\pi(x)P_n(x)] = 0$ para todo polinomio $\pi(x)$ de grado $m < n$, mientras que $\mathfrak{L}[\pi(x)P_n(x)] \neq 0$ si $m = n$;
(c) La relación $\mathfrak{L}[x^m P_n(x)] = K_n \delta_{mn}$ se verifica, con $K_n \neq 0$ y $m = 0, 1, \ldots, n$, siendo δ_{mn} la delta de Kronecker.

Véase la demostración en: [14, cap. 1, pp. 8-9]

4.1.2 | Fórmula de recurrencia

Antes de continuar con el desarrollo de las propiedades principales características de los polinomios es preciso revisar el teorema de Favard.

Teorema 4.1.2 Teorema de Favard

Sean $\{c_n\}_{n=1}^{\infty}$ y $\{\ell_n\}_{n=1}^{\infty}$ sucesiones arbitrarias de números complejos y sea $\{P_n(x)\}_{n=0}^{\infty}$ definido por la fórmula de recurrencia

$$\begin{aligned} P_n(x) &= (x - c_n)\, P_{n-1}(x) - \ell_n P_{n-2}(x), \quad n = 1, 2, 3, \ldots, \\ P_{-1}(x) &= 0, \quad P_0(x) = 1 \end{aligned} \tag{4.1.3}$$

Entonces hay un único funcional de momentos $\mathfrak{L}$ tal que

$$\mathfrak{L}[1] = \ell_1, \quad \mathfrak{L}[P_m(x)P_n(x)] = 0 \quad \text{para } m \neq n, \quad m, n = 0, 1, 2, \ldots.$$

$\mathfrak{L}$ es cuasi-definido y $\{P_n(x)\}$ es la sucesión de polinomios ortogonales mónica correspondiente si y sólo si $\ell_n \neq 0$ mientras que $\mathfrak{L}$ es definido positivo si y sólo si c_n es real y $\ell_n > 0$ $(n \geq 1)$.

Véase la demostración en: [14, cap. 1, p. 22]

Este teorema dice básicamente que si un funcional de momentos $\mathfrak{L}$ es cuasi-definido, es decir, tiene asociado una sucesión de polinomios ortogonales, los polinomios pertenecientes a dicha sucesión pueden generarse a partir de la fórmula de recurrencia dada en (4.1.3). Esto es demasiado importante ya que como se verá más adelante, gracias a este teorema podremos expresar las familias de polinomios ortogonales usadas como fórmulas de recurrencia, permitiendo una mejor manipulación computacional.

4.1.3 | Propiedad de los ceros

Con la definición formal de ortogonalidad entregada por el Teorema 4.1.1 respecto a un funcional de momentos establecido, resulta pertinente examinar una propiedad fundamental de los ceros que satisfacen todos los polinomios ortogonales, junto con un corolario de particular relevancia que se deriva de ella.

Teorema 4.1.3 Separación de los ceros

Los ceros de $P_n(x)$ y $P_{n+1}(x)$ están mutuamente separados uno del otro. Esto es,

$$x_{n+1,i} < x_{n,i} < x_{n+1,i+1}, \quad i = 1, 2, \ldots, n. \tag{4.1.4}$$

Véase la demostración en: [14, cap. 1, p. 28]

Corolario 4.1.1

Para cada $k \geq 1$, $\{x_{nk}\}_{n=k}^{\infty}$ es una sucesión decreciente y $\{x_{n,n-k+1}\}_{n=k}^{\infty}$ es una sucesión creciente. En particular, los límites

$$\xi_i = \lim_{n\to\infty} x_{n,i}, \quad \eta_j = \lim_{n\to\infty} x_{n,n-j+1}, \quad i, j = 1, 2, 3, \ldots, \tag{4.1.5}$$

todos existen, al menos para el sistema de los números reales extendidos.

Véase la demostración en: [14]

El teorema anterior establece que los ceros del polinomio P_n se intercalan con los ceros del polinomio P_{n+1}, configurando una estructura en la cual entre dos ceros consecutivos de P_{n+1} se encuentra invariablemente un cero de P_n, lo que revela un patrón de alternancia que caracteriza la distribución de las raíces en sucesiones de polinomios ortogonales de grados consecutivos. Por su parte, el corolario garantiza la existencia de los límites cuando se consideran las sucesiones formadas por los ceros extremos de cada polinomio, tanto aquellos ubicados hacia el extremo izquierdo del espectro como los situados hacia el extremo derecho, permitiendo así definir las constantes ξ_i y η_j que delimitan el comportamiento asintótico de estas raíces. Esta existencia de límites resulta fundamental para caracterizar el soporte efectivo del funcional de momentos, lo cual conduce a la siguiente definición.

Definición 4.1.3 El verdadero intervalo de ortogonalidad

El intervalo cerrado, $[\xi_1, \eta_1]$, es llamado el *verdadero intervalo de ortogonalidad* de una sucesión de polinomios ortogonales para $\mathfrak{L}$.

Tomado de: [14]

El verdadero intervalo de ortogonalidad constituye el intervalo cerrado más pequeño que contiene todos los ceros de todos los polinomios $P_n(x)$ de la sucesión ortogonal, y puede demostrarse que el conjunto de todos estos ceros forma un conjunto soporte para $\mathfrak{L}$, lo que establece que dicho intervalo es precisamente el menor intervalo cerrado que funciona como conjunto soporte del funcional de momentos [14]. Cabe señalar que, si bien el autor opta por expresar el verdadero intervalo de ortogonalidad como un intervalo cerrado por conveniencia notacional en sus desarrollos posteriores, en el presente trabajo se adoptará la convención más frecuente en la literatura especializada de considerar dicho intervalo en su forma abierta, particularmente al caracterizar las familias de polinomios ortogonales clásicos.

Es importante reconocer que cada sucesión de polinomios ortogonales posee su propio verdadero intervalo de ortogonalidad, el cual caracteriza el dominio natural donde los polinomios exhiben sus propiedades fundamentales y donde se concentra el soporte de la medida asociada al funcional $\mathfrak{L}$, determinando así el comportamiento de los ceros y las propiedades particulares de la sucesión [14]-[16].

4.1.4 | Polinomios ortogonales clásicos

Existen unas familias de polinomios ortogonales las cuales han sido estudiadas ampliamente a lo largo de la historia por sus propiedades particulares y por su aparición recurrente en problemas fundamentales

de la física matemática, la teoría de aproximación y el análisis numérico. Estas familias, conocidas colectivamente como *polinomios ortogonales clásicos*:

Definición 4.1.4 Polinomios ortogonales clásicos

Sean σ y τ polinomios de grado a lo sumo 2 y exactamente 1, respectivamente, con ceros reales y distintos, y sea ρ una función tal que

$$[\sigma(x)\rho(x)]' = \tau(x)\rho(x), \quad \left| \int_a^b x^k \rho(x) dx \right| < +\infty, \tag{4.1.6}$$

donde (a, b) es un intervalo abierto de la recta real en el cual $\rho > 0$. Se dice que una sucesión de polinomios ortogonales $\{P_n\}_{n=0}^{\infty}$ es clásica si resulta ortogonal respecto a la función peso ρ que satisface la ecuación diferencial de Pearson dada en (4.1.6).

Tomado de: [16]

Dicho intervalo abierto (a, b) en el cual $\rho > 0$ es justamente el intervalo verdadero de ortogonalidad para cada sucesión de polinomios ortogonales clásicos según la Definición 4.1.3. A partir de la definición dada para los polinomios ortogonales clásicos surge un teorema el cual permite realizar una caracterización de los mismos.

Teorema 4.1.4 Teorema de caracterización

Los siguientes son equivalentes:

1. $\{P_n\}_{n=0}^{\infty}$ es una familia clásica según la definición Definición 4.1.4.
2. $\{P_n\}_{n=0}^{\infty}$ es ortogonal y la sucesión de sus derivadas $\{P_n'\}_{n=0}^{\infty}$ también es ortogonal.
3. $\{P_n\}_{n=0}^{\infty}$ es ortogonal y la sucesión de sus k-ésimas derivadas $\left\{P_n^{(k)}\right\}_{n=0}^{\infty}$ también es ortogonal.
4. $\{P_n\}_{n=0}^{\infty}$ es solución de la ecuación diferencial de tipo hipergeométrico

$$\sigma(x)P_n''(x) + \tau(x)P_n' + \lambda_n P_n(x) = 0. \tag{4.1.7}$$

5. $\{P_n\}_{n=0}^{\infty}$ se representa mediante la fórmula de Rodrigues

$$P_n(x) = \frac{B_n}{\rho(x)} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \left(\sigma^n(x)\rho(x)\right). \tag{4.1.8}$$

Véase la demostración en: [16, cap. 3, pp. 65-69]

A continuación se presentan las definiciones de las familias de polinomios ortogonales clásicos que se utilizaron.

4.2 | Polinomios de Hermite

Partiendo del Teorema 4.1.4, tomando $\sigma(x) = 1$, $\tau(x) = -2x$, $\lambda_n = 2n$ y $\rho(x) = \mathrm{e}^{-x^2}$, se define la familia de polinomios de Hermite como:

Definición 4.2.1 Polinomios de Hermite

La familia de polinomios clásicos de Hermite $H_n(x)$ se define a partir de su fórmula de Rodrigues

$$H_n(x) = (-1)^n \mathrm{e}^{x^2} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \mathrm{e}^{-x^2}. \tag{4.2.1}$$

Cuya ecuación diferencial asociada viene dada de la forma

$$H_n''(x) - 2xH_n'(x) + 2nH_n(x) = 0. \tag{4.2.2}$$

Tomado de: [14], [16]

Usando el Teorema 4.1.2 y a partir de unas manipulaciones de la fórmula de Rodrigues (4.2.1) se obtienen las siguientes expresiones que también permiten generar los polinomios de Hermite:

- **Fórmula explícita:**

$$H_n(x) = n! \sum_{k=0}^{\lfloor n/2 \rfloor} \frac{(-1)^k (2x)^{n-2k}}{(n-2k)!k!}. \tag{4.2.3}$$

- **Fórmula de recurrencia:**

$$H_{n+1}(x) = 2xH_n - 2nH_{n-1}. \tag{4.2.4}$$

Donde esta última será la utilizada para generar los polinomios computacionalmente. El verdadero intervalo de ortogonalidad para los polinomios de Hermite es $(-\infty, \infty)$.

4.3 | Polinomios de Laguerre

Partiendo del Teorema 4.1.4, tomando $\sigma(x) = x$, $\tau(x) = -x + \alpha + 1$, $\lambda_n = n$ y $\rho(x) = x^\alpha \mathrm{e}^{-x}$ para $\alpha > -1$, se define la familia de polinomios de Laguerre como:

Definición 4.3.1 Polinomios de Laguerre

La familia de polinomios clásicos de Laguerre $L_n^{(\alpha)}(x)$ se define a partir de su fórmula de Rodrigues

$$L_n^{(\alpha)}(x) = \frac{\mathrm{e}^x}{n! x^\alpha} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \left[x^{n+\alpha} \mathrm{e}^{-x} \right]. \tag{4.3.1}$$

Cuya ecuación diferencial asociada viene dada de la forma

$$x \left(L_n^{(\alpha)}(x) \right)'' - (x - \alpha - 1) \left(L_n^{(\alpha)}(x) \right)' + n \left(L_n^{(\alpha)}(x) \right) = 0. \tag{4.3.2}$$

Que para el caso particular en que $\alpha = 0$ se tiene $L_n(x) = L_n^0(x)$.

Tomado de: [14], [16]

Usando el Teorema 4.1.2 y a partir de unas manipulaciones de la fórmula de Rodrigues (4.3.1) para $\alpha = 0$ particularmente, se obtienen las siguientes expresiones que también permiten generar los polinomios de Laguerre:

- **Fórmula explícita:**

$$L_n(x) = \sum_{k=0}^{n} \binom{n}{n-k} \frac{(-x)^k}{k!}. \tag{4.3.3}$$

- **Fórmula de recurrencia:**

$$L_{n+1}(x) = \frac{(2n+1-x)L_n - nL_{n-1}}{n+1}. \tag{4.3.4}$$

Donde esta última será la utilizada para generar los polinomios computacionalmente. El verdadero intervalo de ortogonalidad para los polinomios de Laguerre es $[0, \infty)$

4.4 | Polinomios de Legendre

A diferencia de los anteriores casos, los polinomios de Legendre son un caso particular de los polinomios de Jacobi, por lo que primero definiendo los polinomios de Jacobi partiendo del Teorema 4.1.4, tomando $\sigma(x) = 1 - x^2$, $\tau(x) = -(\alpha + \beta + 2)x + \beta - \alpha$, $\lambda_n = n(n + \alpha + \beta + 1)$ y $\rho(x) = (1-x)^{\alpha}(1+x)^{\beta}$ para $\alpha, \beta > -1$, se define la familia de polinomios de Jacobi como:

Definición 4.4.1 Polinomios de Jacobi

La familia de polinomios clásicos de Jacobi $P_n^{(\alpha,\beta)}(x)$ se define a partir de su fórmula de Rodrigues

$$P_n^{(\alpha,\beta)}(x) = (-2)^{-n}(n!)^{-1}(1-x)^{-\alpha}(1+x)^{-\beta}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left[(1-x)^{n+\alpha}(1+x)^{n+\beta}\right]. \tag{4.4.1}$$

Cuya ecuación diferencial asociada viene dada de la forma

$$\left(1-x^2\right)\left(P_n^{(\alpha,\beta)}(x)\right)'' - \left[(\alpha+\beta+2)x + \beta - \alpha\right]\left(P_n^{(\alpha,\beta)}(x)\right)' + n(n+\alpha+\beta+1)\left(P_n^{(\alpha,\beta)}(x)\right) = 0. \tag{4.4.2}$$

Tomado de: [14], [16]

Luego, para el caso particular en que $\alpha = \beta = 0$ se recae a la definición de los polinomios de Legendre, es decir $P_n(x) = P_n^{0,0}(x)$, por lo que a partir de la Definición 4.4.1 se obtiene:

Definición 4.4.2 Polinomios de Legendre

La familia de polinomios clásicos de Legendre $P_n(x)$ se define a partir de su fórmula de Rodrigues

$$P_n(x) = \frac{1}{2^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left[\left(x^2-1\right)^n\right]. \tag{4.4.3}$$

Cuya ecuación diferencial asociada viene dada de la forma

$$\left(1-x^2\right)P_n(x)'' - 2xP_n(x)' + n(n+1)P_n(x) = 0. \tag{4.4.4}$$

Tomado de: [14], [16]

Usando el Teorema 4.1.2 y a partir de unas manipulaciones de la fórmula de Rodrigues (4.4.3) se obtienen las siguientes expresiones que también permiten generar los polinomios de Legendre:

- **Fórmula explícita:**

$$P_n(x) = 2^{-n}\sum_{k=0}^{n}\binom{n}{n-k}\binom{n}{k}(x-1)^k(x+1)^{k+1}. \tag{4.4.5}$$

- **Fórmula de recurrencia:**

$$P_{n+1} = \frac{(2n+1)\,xP_n - nP_{n-1}}{n+1}. \tag{4.4.6}$$

Donde esta última será la utilizada para generar los polinomios computacionalmente. El verdadero intervalo de ortogonalidad para los polinomios de Legendre es $[-1, 1]$ (al igual que con los polinomios de Jacobi debido a que provienen de ellos).

Capítulo 5

Redes Neuronales Ortogonales

5.1 | Definición y estructura

Las redes neuronales ortogonales (ONN, por sus siglas en inglés) son una clase especializada de redes neuronales que utilizan funciones de activación basadas en polinomios ortogonales, como los polinomios de Hermite, Laguerre y Legendre entre otros [13], [17], [36].

Este tipo de redes fue propuesto por Yang y Tseng [13] como una alternativa para realizar aproximaciones funcionales ya sea en un dominio discreto o continuo, aprovechando las propiedades matemáticas de los polinomios ortogonales para mejorar la capacidad de generalización y estabilidad del modelo. En su artículo original realizan la propuesta para una red neuronal conformada a partir de los polinomios de Legendre en la cual solamente se tienen pesos de salida, no se tienen pesos de entrada ni sesgos (véase), lo cual permite simplificar bastante el entrenamiento de la red lo que la vuelve muy eficiente en términos computacionales.

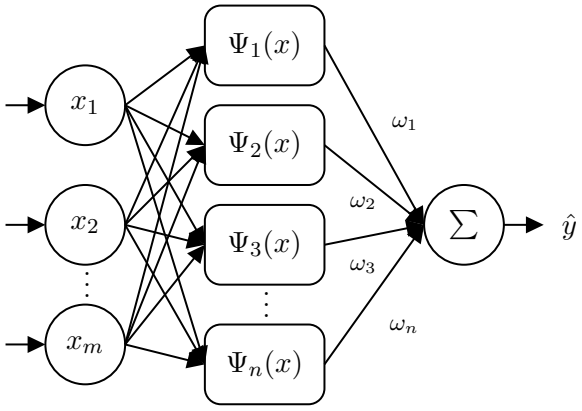


Figura 5.1: Representación gráfica de las redes neuronales ortogonales ONN. La representación matemática de la red junto a sus partes se puede ver en (5.1.1).

Esta red neuronal permite ser escrita matemáticamente como una combinación lineal de la sucesión de polinomios ortogonales seleccionados con los pesos sinápticos de la salida como coeficientes, es decir, si se tiene una red neuronal ortogonal con N neuronas y se denota la función de activación ortogonal como $\Psi_n(x)$, donde n es el grado del polinomio ortogonal, entonces la salida de la red neuronal para una entrada x se puede expresar como

$$\hat{y}(x) = \sum_{n=1}^{N} \omega_n \Psi_n(x) \tag{5.1.1}$$

donde ω_n son los pesos sinápticos de la salida que deben ser ajustados durante el proceso de entrenamiento para minimizar la función de costo.

El entrenamiento de este tipo de red neuronal puede ser tan complejo a como el de una red neuronal tradicional, sin embargo, al no tener pesos de entrada ni sesgos, el proceso de entrenamiento se simplifica considerablemente, ya que solamente se deben ajustar los pesos de salida, no obstante, esto será explorado más adelante [17], [21], [36], [37].

5.2 | Tipos de ONN utilizadas

Como se mencionó previamente, este tipo de red depende de la sucesión de polinomios ortogonales que se vaya a utilizar, en ese sentido, existen tantas redes neuronales ortogonales como sucesiones de polinomios ortogonales existan, sin embargo, en este trabajo se estudiaron las siguientes tres redes neuronales ortogonales:

5.2.1 | HerNN (Hermite Neural Network)

Las redes neuronales de Hermite (HerNN) se construyen a partir de la familia de polinomios ortogonales clásicos de Hermite (véase la Figura 5.2). Al igual que con las demás ONN se puede escribir matemáticamente como

$$\hat{y}(x) = \sum_{n=1}^{m} \omega_n H_n(x) \tag{5.2.1}$$

donde $\hat{y}(x)$ es la salida de la red, ω_n el conjunto de pesos de salida y $H_n(x)$ el polinomio n-ésimo de Hermite el cual es generado a partir de la fórmula de recurrencia (4.2.4) [19], [20].

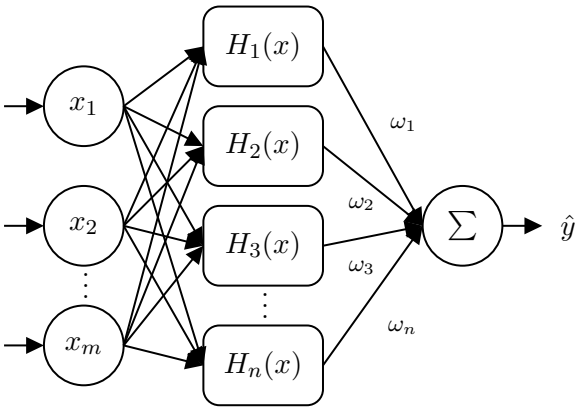


Figura 5.2: Esquema de una red neuronal de Hermite.

5.2.2 | LaNN (Laguerre Neural Network)

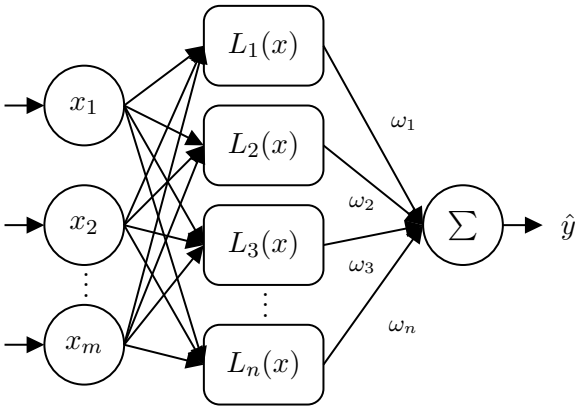


Figura 5.3: Esquema de una red neuronal de Laguerre.

Las redes neuronales de Laguerre (LaNN) se construyen a partir de la familia de polinomios ortogonales clásicos de Laguerre (véase la Figura 5.3). Al igual que con las demás ONN se puede escribir matemáticamente como

$$\hat{y}(x) = \sum_{n=1}^{m} \omega_n L_n(x) \tag{5.2.2}$$

donde $\hat{y}(x)$ es la salida de la red, ω_n el conjunto de pesos de salida y $L_n(x)$ el polinomio n-ésimo de Laguerre el cual es generado a partir de la fórmula de recurrencia (4.3.4) [18].

5.2.3 | LeNN (Legendre Neural Network)

Las redes neuronales de Legendre (LeNN) se construyen a partir de la familia de polinomios ortogonales clásicos de Legendre (véase la Figura 5.4). Al igual que con las demás ONN se puede escribir matemáticamente como

$$\hat{y}(x) = \sum_{n=1}^{m} \omega_n P_n(x) \tag{5.2.3}$$

donde $\hat{y}(x)$ es la salida de la red, ω_n el conjunto de pesos de salida y $P_n(x)$ el polinomio n-ésimo de Legendre el cual es generado a partir de la fórmula de recurrencia (4.4.6) [17], [21], [22].

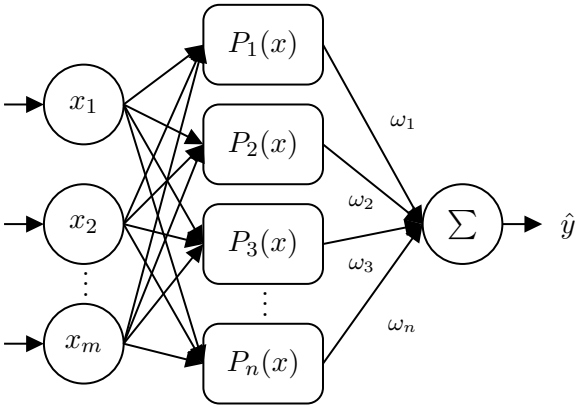


Figura 5.4: Esquema de una red neuronal de Legendre.

5.3 | Función de costo

La función de costo utilizada para guiar el proceso de entrenamiento es el error cuadrático medio (véase la Definición 3.2.7). Esta medida se define a partir de la discrepancia entre la salida deseada y y la salida producida por la red $\hat{y}$, estableciendo el error como $E = y - \hat{y}$, o mejor escrito con base en su definición

$$J = \frac{1}{2}E^2, \tag{5.3.1}$$

donde el factor $\frac{1}{2}$ se introduce para simplificar las derivadas en el cálculo del gradiente [13].

5.4 | Entrenamiento

El proceso de entrenamiento en las redes neuronales ortogonales (ONN) se fundamenta en la estructura lineal de sus coeficientes y en el uso de funciones base ortogonales, lo cual asegura la existencia de soluciones únicas y estables para los pesos sinápticos. A diferencia de las redes multicapa, donde la optimización puede verse afectada por la presencia de mínimos locales, las ONN se caracterizan por una formulación que conduce a una solución bien definida [13].

En su formulación original, el entrenamiento de las ONN se llevó a cabo mediante métodos de primer orden (véase la subsección 3.3.1), en particular a través de la *regla delta generalizada*, que constituye una extensión del algoritmo de gradiente descendente. Bajo este esquema, los pesos de salida se ajustan iterativamente en la dirección opuesta al gradiente de la función de costo ya mencionada. Dada la ortogonalidad de las funciones de base, este procedimiento garantiza la convergencia hacia una solución óptima en un número finito de pasos [13].

Con el paso del tiempo, se han reportado en la literatura otros enfoques de entrenamiento que buscan mejorar la eficiencia y escalabilidad de las ONN, entre ellos destaca el método *Extreme Learning Machine (ELM)* (véase el Capítulo 6), en el cual los pesos de entrada se asignan aleatoriamente y permanecen fijos, lo que reduce el entrenamiento a la solución de un sistema lineal, en general, como no se tienen pesos de entrada para este tipo de red ya se recae a una red apta para usar este método de entrenamiento [22], [36].

Capítulo 6

Extreme Learning Machine (ELM)

6.1 | Definición

El *Extreme Learning Machine* (ELM) es una técnica de entrenamiento rápido para redes neuronales de tipo perceptrón de una sola capa oculta (SLFN, por sus siglas en inglés). A diferencia de los métodos tradicionales basados en gradiente descendente, ELM fija los pesos de entrada de manera aleatoria y entrena únicamente los pesos de salida mediante una solución analítica en forma cerrada. Esto elimina la necesidad de un procedimiento iterativo de optimización y reduce significativamente el tiempo de cómputo [22], [36].

6.2 | Pseudo-inversa generalizada de Moore-Penrose

El concepto de inversa generalizada surge de la necesidad de extender la noción de matriz inversa a casos en los que la matriz original no es cuadrada o no es invertible. En 1955, Penrose [45] demostró que para toda matriz finita A, cuadrada o rectangular, con elementos reales o complejos, existe una única matriz X que satisface un conjunto de cuatro condiciones fundamentales. Estas condiciones, conocidas como *ecuaciones de Penrose*, caracterizan de manera única la *pseudo-inversa de Moore-Penrose* [35].

Definición 6.2.1 Ecuaciones de Penrose

Sea $A \in \mathbb{C}^{m\times n}$. Una matriz $X \in \mathbb{C}^{n\times m}$ se denomina *pseudo-inversa de Moore-Penrose* de A si satisface las cuatro ecuaciones:

$$AXA = A, \tag{6.2.1}$$
$$XAX = X, \tag{6.2.2}$$
$$(AX)^* = AX, \tag{6.2.3}$$
$$(XA)^* = XA. \tag{6.2.4}$$

La matriz que cumple (6.2.1)-(6.2.4) se denota por $A^\dagger$.

Tomado de: [35], [45]

Estas condiciones garantizan que $A^\dagger$ hereda propiedades de la inversa usual cuando esta existe, pero al mismo tiempo proporciona una solución coherente en el caso general. El resultado central de Penrose establece su existencia y unicidad.

Teorema 6.2.1 Existencia y unicidad (Penrose)

Para toda $A \in \mathbb{C}^{m\times n}$ existe una única matriz $A^\dagger \in \mathbb{C}^{n\times m}$ que satisface las cuatro ecuaciones de Penrose (6.2.1)-(6.2.4).

Véase la demostración en: [35], [45]

Un aspecto fundamental en el cálculo práctico de la pseudo-inversa es su representación mediante la descomposición en valores singulares (SVD), que además de garantizar existencia y unicidad, ofrece un procedimiento numérico estable.

Teorema 6.2.2 Representación mediante la SVD

Sea la descomposición en valores singulares (SVD) de A dada por

$$A = U \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} V^* = U_r \Sigma_r V_r^*,$$

donde $\Sigma_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$ contiene los $r = \operatorname{rank}(A)$ valores singulares no nulos y $U_r \in \mathbb{C}^{m\times r}$, $V_r \in \mathbb{C}^{n\times r}$. Entonces la pseudo-inversa se expresa como

$$A^\dagger = V_r \Sigma_r^{-1} U_r^*.$$

En particular, la SVD proporciona un procedimiento numérico directo y robusto para calcular $A^\dagger$.

Véase la demostración en: [35]

Más allá de la caracterización algebraica, la pseudo-inversa tiene una interpretación natural en términos de problemas de mínimos cuadrados. En este contexto, permite identificar de manera única la solución de norma mínima.

Teorema 6.2.3 Solución de norma mínima del problema de mínimos cuadrados

Para $A \in \mathbb{C}^{m\times n}$ y $b \in \mathbb{C}^m$, la solución del problema de mínimos cuadrados

$$\min_{x\in\mathbb{C}^n} \|Ax - b\|_2$$

que tenga norma mínima está dada por

$$x^\star = A^\dagger b.$$

Además, $x^\star$ es la única solución con menor norma entre todas las soluciones de mínimos cuadrados.

Véase la demostración en: [35]

La pseudo-inversa posee además un conjunto amplio de propiedades estructurales que la conectan con proyecciones ortogonales y simetrías matriciales.

Teorema 6.2.4 Propiedades básicas

Sean $A \in \mathbb{C}^{m\times n}$ y $A^\dagger$ su pseudo-inversa. Se verifican las propiedades siguientes:

1. $AA^\dagger$ y $A^\dagger A$ son proyectores hermíticos (idempotentes y autoadjuntos).
2. $R(AA^\dagger) = R(A)$ y $R(A^\dagger A) = R(A^*)$.
3. $\text{rank}(A^\dagger) = \text{rank}(A)$.
4. $(A^\dagger)^* = (A^*)^\dagger$ y $(A^\dagger)^\dagger = A$.
5. Si A es Hermítica (resp. semidefinida positiva), entonces $A^\dagger$ también lo es.

Véase la demostración en: [35]

En los casos particulares en que A posee rango completo, la pseudo-inversa coincide con fórmulas más simples en términos de matrices inversas clásicas.

Corolario 6.2.1 Casos de rango completo

1. Si $A \in \mathbb{C}^{m\times n}$ tiene rango completo por columnas ($\text{rank}(A) = n$, $m \geq n$), entonces
$$A^\dagger = (A^*A)^{-1}A^*.$$
2. Si A tiene rango completo por filas ($\text{rank}(A) = m$, $n \geq m$), entonces
$$A^\dagger = A^*(AA^*)^{-1}.$$

Estas expresiones se derivan de la representación por SVD y de las propiedades de los proyectores ortogonales.

Véase la demostración en: fullrank-cases

Una perspectiva alternativa de la pseudo-inversa surge al considerar la regularización de Tikhonov. Bajo este marco, $A^\dagger$ aparece como el límite de soluciones regularizadas, lo que justifica su uso en problemas mal condicionados.

Teorema 6.2.5 Carácter variacional y regularización

El pseudo-inverso aparece como límite regularizado de la solución de mínimos cuadrados de Tikhonov:
$$\lim_{\lambda\downarrow 0}(A^*A + \lambda I)^{-1}A^* \;=\; A^\dagger.$$

Así, la familia $(A^*A + \lambda I)^{-1}A^*$ proporciona una forma numérica estable de aproximar $A^\dagger$ cuando A es mal condicionada.

Véase la demostración en: variational-reg

Finalmente, es importante destacar que la pseudo-inversa admite múltiples representaciones y métodos de cálculo, tanto exactos como aproximados, lo cual refuerza su relevancia en la teoría y práctica del álgebra lineal aplicada.

Teorema 6.2.6 Representaciones alternativas y métodos de cómputo

Además de la SVD, existen varias representaciones y métodos para construir $A^\dagger$:

- Fórmulas basadas en factorizaciones de rango completo $A = FG$ que conducen a $A^\dagger = G^*(F^*AG^*)^{-1}F^*$ cuando las inversas intermedias existen.
- Métodos iterativos y por columnas, como el método de Greville, que construyen $A^\dagger$ de forma recursiva.
- Algoritmos numéricos estables basados en reducción bidiagonal y SVD (Golub-Kahan, Golub-Reinsch). Estos métodos son preferidos en cómputo científico por su estabilidad y eficiencia.

Véase la demostración en: computational-methods

6.3 | Funcionamiento y aplicación

El funcionamiento del *Extreme Learning Machine* (ELM) se fundamenta en la idea de asignar aleatoriamente los pesos de entrada y los sesgos de la capa oculta, de modo que el entrenamiento se reduzca al cálculo de los pesos de salida. Dicho cálculo se realiza resolviendo un sistema lineal mediante la pseudo-inversa de Moore-Penrose.

6.3.1 | Funcionamiento

El procedimiento se resume en los siguientes pasos:

1. **Inicialización aleatoria:** los pesos de entrada $\mathbf{w}_j$ y los sesgos b_j de las neuronas ocultas se asignan de manera aleatoria y se mantienen fijos durante todo el entrenamiento.
2. **Cálculo de la matriz H:** a partir de las entradas de entrenamiento $\{x_i\}_{i=1}^N$, se calcula la salida de la capa oculta mediante una función de activación $\phi(\cdot)$:

$$H_{ij} = \phi(\mathbf{w}_j \cdot \mathbf{x}_i + b_j), \quad i = 1, \ldots, N,\ j = 1, \ldots, m, \tag{6.3.1}$$

   donde m es el número de neuronas en la capa oculta.
3. **Obtención de los pesos de salida:** los pesos de salida β se obtienen resolviendo el sistema

$$H\beta = T, \tag{6.3.2}$$

   donde T es la matriz de salidas deseadas. Como en general H no es cuadrada, la solución se obtiene mediante la *pseudo-inversa de Moore-Penrose*, definida en las ecuaciones (6.2.1)-(6.2.4) (véase la Definición 6.2.1), cuya existencia y unicidad están garantizadas por el Teorema 6.2.1.

En consecuencia, la solución de norma mínima se expresa como

$$\beta = H^\dagger T, \tag{6.3.3}$$

según establece el Teorema 6.2.3. El cálculo práctico de $H^\dagger$ se realiza mediante la descomposición en valores singulares, de acuerdo con el Teorema 6.2.2.

Este marco convierte el entrenamiento en un problema estrictamente lineal, resuelto de forma analítica, lo que explica la rapidez y estabilidad numérica del ELM.

6.3.2 | Aplicación a una *FODE* de tres términos

Una de las aplicaciones destacadas del ELM es la resolución de ecuaciones diferenciales ordinarias de orden arbitrario. Consideremos la ecuación diferencial fraccionaria de tres términos (1.6.3):

$$a\mathcal{D}_t^{\beta} y(t) + b\mathcal{D}_t^{\alpha} y(t) + cy(t) = u(t), \quad t \in \Omega, \tag{6.3.4}$$

con condiciones iniciales (o de frontera, dependiendo el caso) apropiadas.

Para aproximar la solución con ELM, se discretiza el dominio Ω en N puntos $\{t_i\}_{i=1}^{N}$, luego, en cada punto se impone la ecuación diferencial, sustituyendo $y(t)$ por la salida de la red $\hat{y}(t;\omega)$, parametrizada únicamente por los pesos de salida ω:

$$y(t) \approx \sum_{j=1}^{m} \omega_j \, \phi(\mathbf{w}_j \cdot t + b_j), \tag{6.3.5}$$

que realizando las derivadas se obtienen expresiones para $\mathcal{D}_t^{\alpha,\beta} y(t)$, de modo que en cada punto x_i se cumple:

$$a\sum_{j=1}^{m} \omega_j \mathcal{D}_t^{\beta}\phi(\mathbf{w}_j \cdot t_i + b_j) + b\sum_{j=1}^{m} \omega_j \mathcal{D}_t^{\alpha}\phi(\mathbf{w}_j \cdot t_i + b_j) + c\sum_{j=1}^{m} \omega_j \phi(\mathbf{w}_j \cdot t_i + b_j) = u(t_i). \tag{6.3.6}$$

Esto se traduce en un sistema matricial de la forma

$$A\beta = U, \tag{6.3.7}$$

donde:

- $A \in \mathbb{R}^{N\times m}$ es la matriz cuyos elementos son

$$A_{ij} = a\mathcal{D}_t^{\beta}\phi(\mathbf{w}_j \cdot t_i + b_j) + b\mathcal{D}_t^{\alpha}\phi(\mathbf{w}_j \cdot t_i + b_j) + c\phi(\mathbf{w}_j \cdot t_i + b_j),$$

- $\beta \in \mathbb{R}^m$ es el vector de pesos de salida (cuyos elementos son los pesos ω_j) y,
- $F \in \mathbb{R}^N$ es el vector con componentes $u(t_i)$.

Finalmente, la solución en el sentido de mínimos cuadrados se obtiene como

$$\beta = A^{\dagger}F, \tag{6.3.8}$$

donde $A^{\dagger}$ es la pseudo-inversa generalizada de Moore-Penrose. De esta forma, la resolución de la ecuación diferencial queda reducida a un problema algebraico lineal, alineado con los resultados teóricos presentados en los teoremas anteriores [22], [36].

Parte III

Resultados

Finalmente, esta sección expone los resultados mediante la aplicación de las *Redes Neuronales Ortogonales* en la aproximación numérica de la *ecuación de Bagley-Torvik* para un problema particular, empleando distintos métodos de entrenamiento a partir de los cuales se evaluaron métricas de rendimiento y precisión de la aproximación con el propósito de identificar, en primer lugar, aquellos esquemas de entrenamiento que proporcionaron un desempeño superior y, en segundo lugar, determinar cuál de las arquitecturas consideradas resulta más adecuada para abordar eficazmente este problema.

7 Problema a resolver **57**

8 Resultados **58**

8.1 Bitácora de resultados . . . 58

8.1.1 Métodos de entrenamiento de primer orden . . . 58

8.1.1.1 Resultados para: HerNN . . . 58

8.1.1.2 Resultados para: LaNN . . . 59

8.1.1.3 Resultados para: LeNN . . . 60

8.1.1.4 Mejores resultados (HerNN vs. LaNN vs. LeNN) . . . 61

8.1.2 Métodos de segundo orden . . . 62

8.1.2.1 Resultados para: HerNN . . . 62

8.1.2.2 Resultados para: LaNN . . . 63

8.1.2.3 Resultados para: LeNN . . . 64

8.1.2.4 Mejores resultados (HerNN vs. LaNN vs. LeNN) . . . 65

8.1.3 Extreme Learning Machine (ELM) . . . 66

8.1.3.1 Resultados para: HerNN . . . 67

8.1.3.2 Resultados para: LaNN . . . 68

8.1.3.3 Resultados para: LeNN . . . 69

8.1.3.4 Mejores resultados (HerNN vs. LaNN vs. LeNN) . . . 70

9 Análisis de resultados y conclusiones **71**

Problema a resolver

Dentro de los problemas a resolver para la ecuación de Bagley-Torvik está uno que se caracteriza por ser bastante común. Este problema es conocido en la literatura como el *problema de Cauchy* la cual se puede encontrar en [4], [7], [21], [23], [24], [46], [47], cuya hipótesis es la siguiente:

> Suponga la ecuación de Bagley-Torvik (2.3.4) con las condiciones iniciales $x(0) = x'(0) = 0$, cuyas constantes son: $m = 1$, $2A\sqrt{\mu\rho} = 0{,}5$, $\kappa = 0{,}5$ y el forzamiento externos viene dado de la forma:
>
> $$f(t) = \begin{cases} 8, & 0 < t < 1, \\ 0, & t \geq 1. \end{cases}$$

La solución analítica de este problema se puede tomar directamente de la función de Green fraccionaria asociada (2.4.1) para los valores de las constantes dadas (véase Sección 2.4). Sin embargo, esta solución puede ser un poco costosa en términos computacionales de calcular, por tanto, Podlubny [48] propone un método numérico basado en el método matricial para resolver este problema.

Tomando el dominio solución $[0, 30]$ se obtiene la siguiente gráfica como solución analítica del problema planteado:

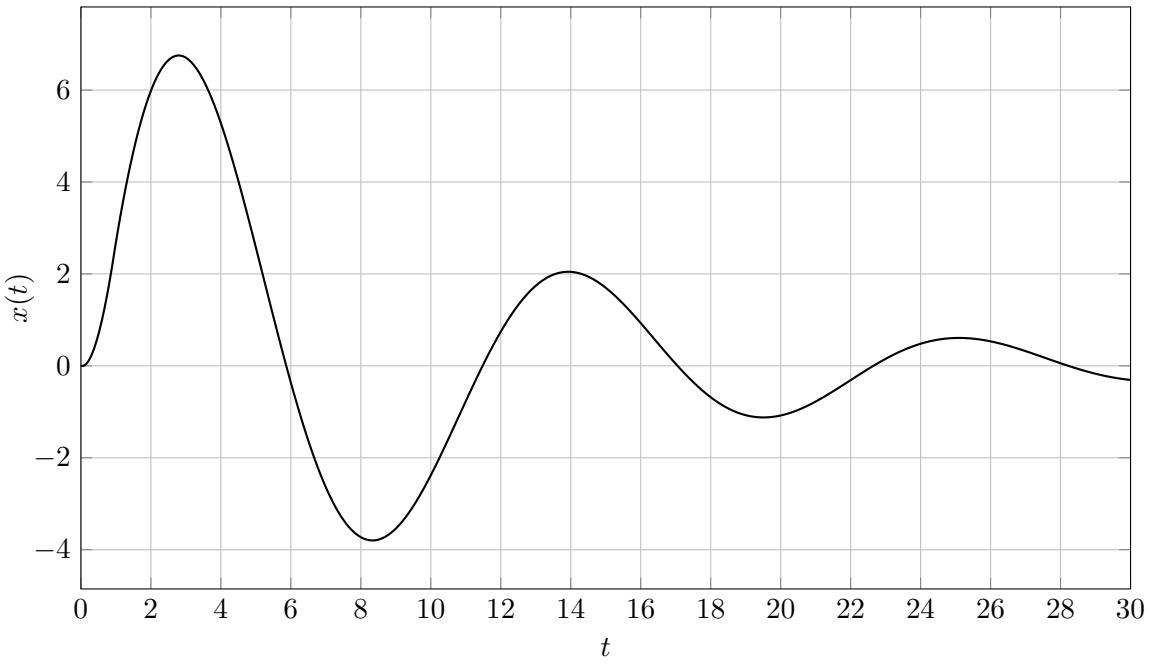


Figura 7.1: Solución analítica de la ecuación de Bagley-Torvik para el problema planteado en el dominio $[0, 30]$.

Capítulo 8

Resultados

8.1 | Bitácora de resultados

Como se ha visto previamente, las redes neuronales ortogonales que se utilizaron fueron la red neuronal de Hermite (HerNN, por sus siglas en inglés), la red neuronal de Laguerre (LaNN, por sus siglas en inglés) y la red neuronal de Legendre (LeNN, por sus siglas en inglés). Luego se realizó una *búsqueda de hiperparámetros* para cada una de las redes en donde se varió principalmente el número de neuronas por red para ver cuál es la de mayor rendimiento, junto a ello también se utilizaron diferentes métodos de entrenamiento para cada red. A continuación, se presentan los resultados obtenidos para cada uno de los métodos de entrenamiento utilizados.

8.1.1 | Métodos de entrenamiento de primer orden

El código general creado para construir las distintas redes con los distintos métodos de entrenamiento se puede encontrar en el Apéndice A.2. A continuación se presentan tanto una tabla como un gráfico que contiene los métodos mencionados en la subsección 3.3.1 con la mejor configuración encontrada para cada red posterior a la realización de la búsqueda de hiperparámetros.

8.1.1.1 | Resultados para: HerNN

Tabla 8.1: Mejores resultados para la red HerNN. El tiempo fue medido en segundos.

Método	Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r	Épocas
SGD	No hubo convergencia							
Adam	6	3.120562	2.198089	1.672379	5.830256	2.414592	-0.025107	600
AdamW	6	3.373748	2.197818	1.673675	5.833811	2.415328	-0.025732	600
RMSprop	6	0.480511	2.199888	1.670091	5.904923	2.430005	-0.038235	116
Adagrad	6	2.597023	2.198145	1.672078	5.829845	2.414507	-0.025035	800

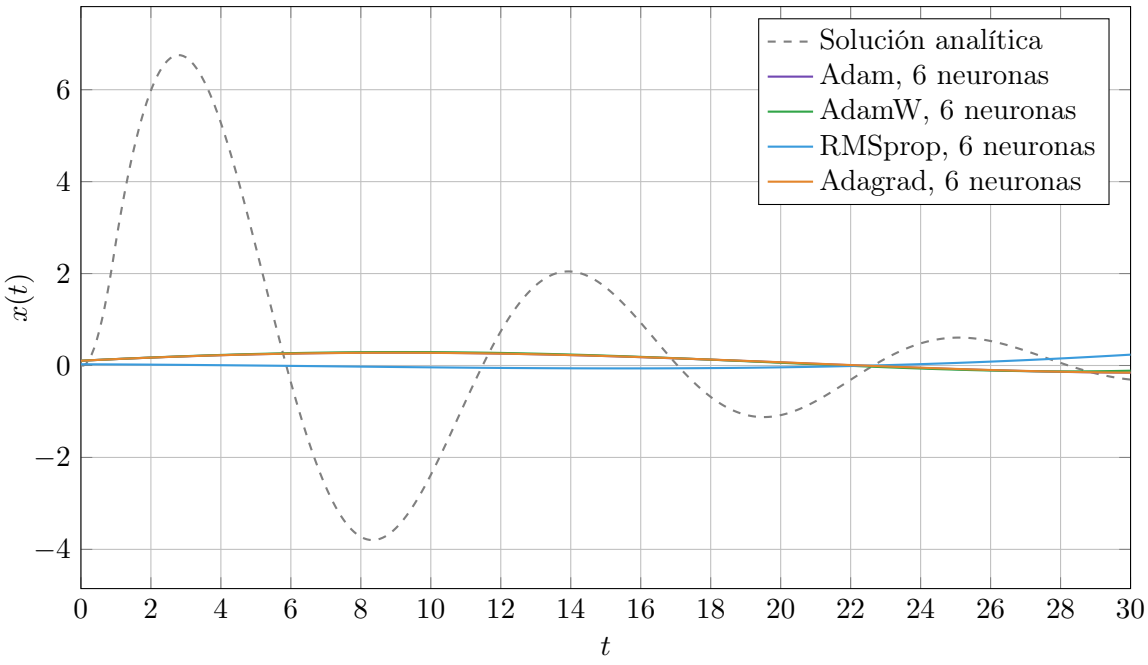


Figura 8.1: Comparativa de los mejores resultados obtenidos para HerNN con la solución analítica.

Discusión

En el caso de la red HerNN se observa que ninguno de los métodos de primer orden logró un ajuste satisfactorio frente a la solución de referencia, ya que incluso los mejores resultados presentaron valores negativos de r acompañados de pérdidas y errores relativamente altos. El método Adam, por ejemplo, arrojó una pérdida cercana a 2,2, un MAE superior a 1,6 y un RMSE de 2,41, mientras que AdamW ofreció métricas prácticamente idénticas sin aportar mejoras significativas. RMSprop redujo de manera considerable el tiempo de ejecución, resolviendo en menos de medio segundo frente a más de tres segundos en los otros casos, aunque mantuvo un error elevado y un r todavía más bajo, cercano a $-0{,}04$. Adagrad repitió este mismo patrón, confirmando que la red HerNN mostró un desempeño deficiente con todos los optimizadores de primer orden y, de hecho, con SGD ni siquiera alcanzó convergencia, por lo que bajo estas condiciones no se obtuvieron aproximaciones fiables.

8.1.1.2 | Resultados para: LaNN

Tabla 8.2: Mejores resultados para la red LaNN. El tiempo fue medido en segundos.

Método	Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r	Épocas
SGD	40	2.712042	0.458864	1.502315	2.918221	1.708280	0.486903	800
Adam	40	3.112563	0.763360	2.168863	7.930390	2.816095	-0.394364	600
AdamW	40	3.223336	0.757022	2.181655	7.873119	2.805908	-0.384294	600
RMSprop	40	2.703194	0.684778	2.124738	7.486531	2.736153	-0.316322	700
Adagrad	40	2.664206	0.769682	2.158799	7.846353	2.801134	-0.379588	800

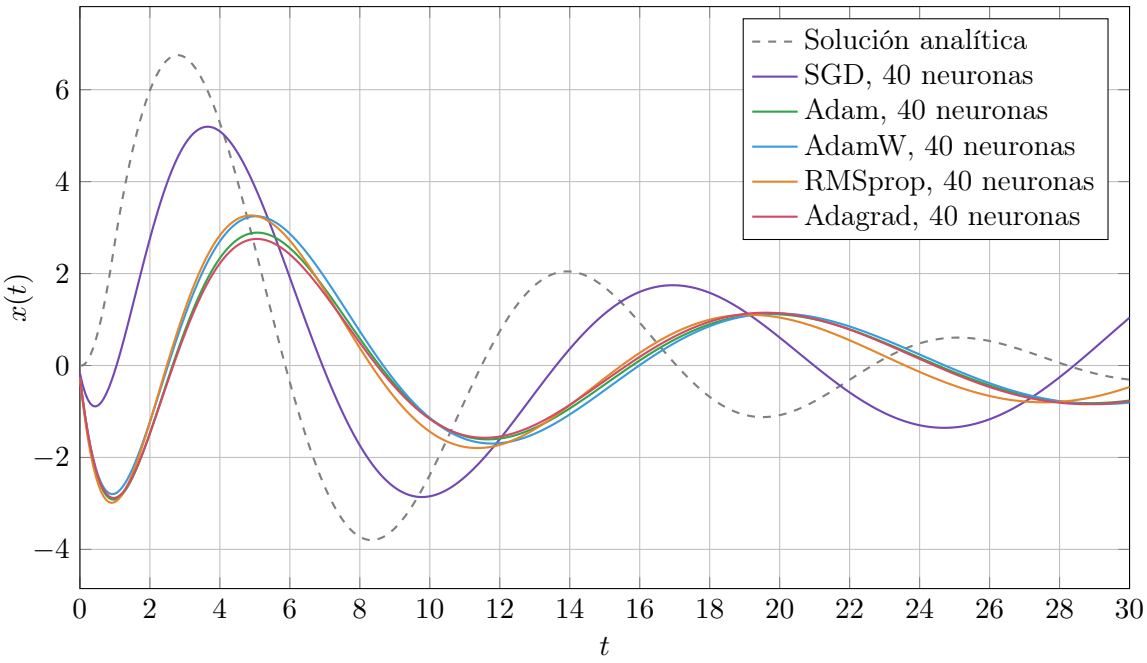


Figura 8.2: Comparativa de los mejores resultados obtenidos para LaNN con la solución analítica.

Discusión

En la red LaNN el panorama fue distinto, puesto que dentro de los métodos de primer orden se evidenció un comportamiento mucho más estable. El optimizador que ofreció el mejor desempeño fue SGD con 40 neuronas, alcanzando un r positivo de 0.48 y un RMSE de 1.70, con una pérdida de 0.46 que refleja un ajuste más cercano a la solución analítica en comparación con los otros métodos. Adam, AdamW, RMSprop y Adagrad, en contraste, no lograron replicar dicho rendimiento, ya que en todos los casos el r resultó negativo y los errores crecieron hasta valores de RMSE en torno a 2.7-2.8 acompañados de pérdidas superiores a 0.68. El tiempo de ejecución se mantuvo en un rango estrecho, alrededor de tres segundos, de manera que la principal diferencia radicó en la calidad del ajuste y no en el coste computacional. Así, dentro de LaNN y en este contexto específico, el método SGD fue claramente superior a los demás.

8.1.1.3 | Resultados para: LeNN

Tabla 8.3: Mejores resultados para la red LeNN. El tiempo fue medido en segundos.

Método	Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r	Épocas
SGD	18	2.803095	0.282474	1.262746	2.715502	1.647878	0.522546	800
Adam	20	3.228836	0.431192	1.710028	6.084042	2.466585	-0.069729	600
AdamW	12	3.299770	0.455652	1.694613	6.090718	2.467938	-0.070903	600
RMSprop	12	2.644610	0.391500	1.632580	5.464121	2.337546	0.039269	700
Adagrad	12	2.657931	0.446480	1.691293	6.060230	2.461753	-0.065542	800

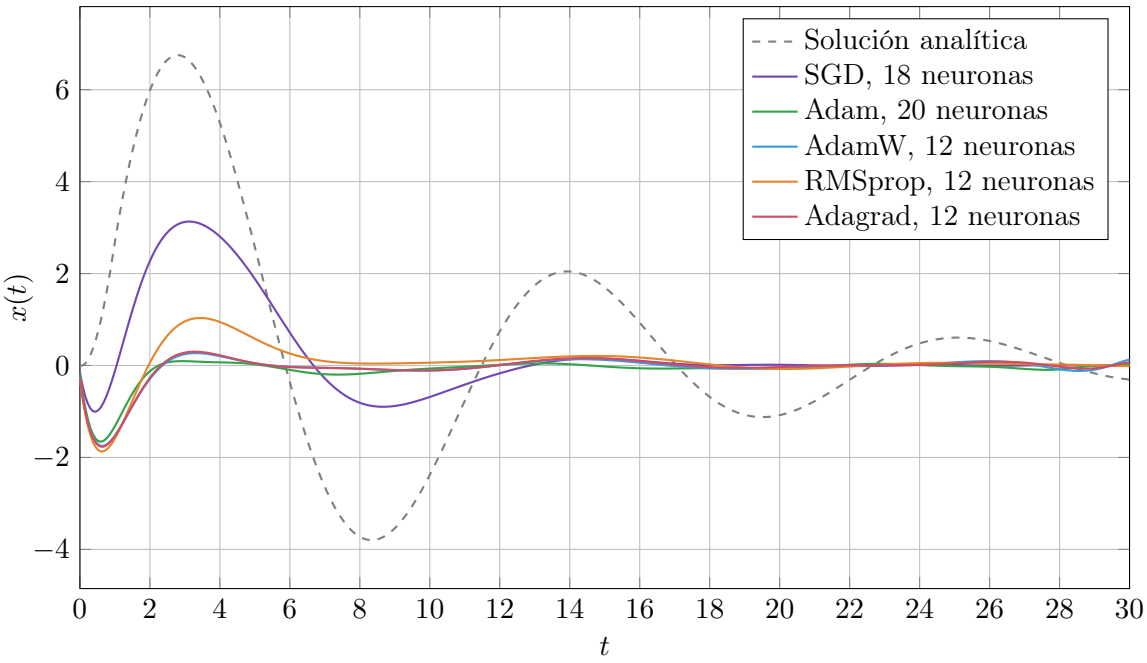


Figura 8.3: Comparativa de los mejores resultados obtenidos para LeNN con la solución analítica.

Discusión

En el análisis de la red LeNN se identificó un comportamiento similar al de LaNN, dado que nuevamente el optimizador SGD resultó ser el más adecuado. Con 18 neuronas, este método alcanzó un r de 0.52, el mayor entre todas las configuraciones de primer orden, acompañado de un RMSE de 1.64 y una pérdida de 0.28, que en conjunto representan el mejor ajuste observado. Los demás optimizadores no se acercaron a este rendimiento, pues Adam, AdamW, RMSprop y Adagrad ofrecieron métricas con r muy cercanos a cero o negativos y errores notablemente más altos, como en el caso de Adam con un RMSE de 2.46 y una pérdida de 0.43. Los tiempos de ejecución oscilaron entre 2.6 y 3.3 segundos, lo que confirma que la diferencia esencial radicó en la calidad del ajuste, situando nuevamente a SGD como el método preferido para esta red.

8.1.1.4 | Mejores resultados (HerNN vs. LaNN vs. LeNN)

Finalmente, se presenta un resumen de los mejores métodos para cada red, luego, se comparan las tres redes para ver cuál fue la de mejor desempeño.

Tabla 8.4: Mejores resultados para cada una de las redes. El tiempo fue medido en segundos.

Red	Método	Neuronas	Tiempo	Pérdida	RMSE	r	Épocas
HerNN	AdamW	8	3.331957	2.191537	2.416519	-0.026743	600
LaNN	SGD	40	2.712042	0.458864	1.708280	0.486903	800
LeNN	SGD	18	2.803095	0.282474	1.647878	0.522546	800

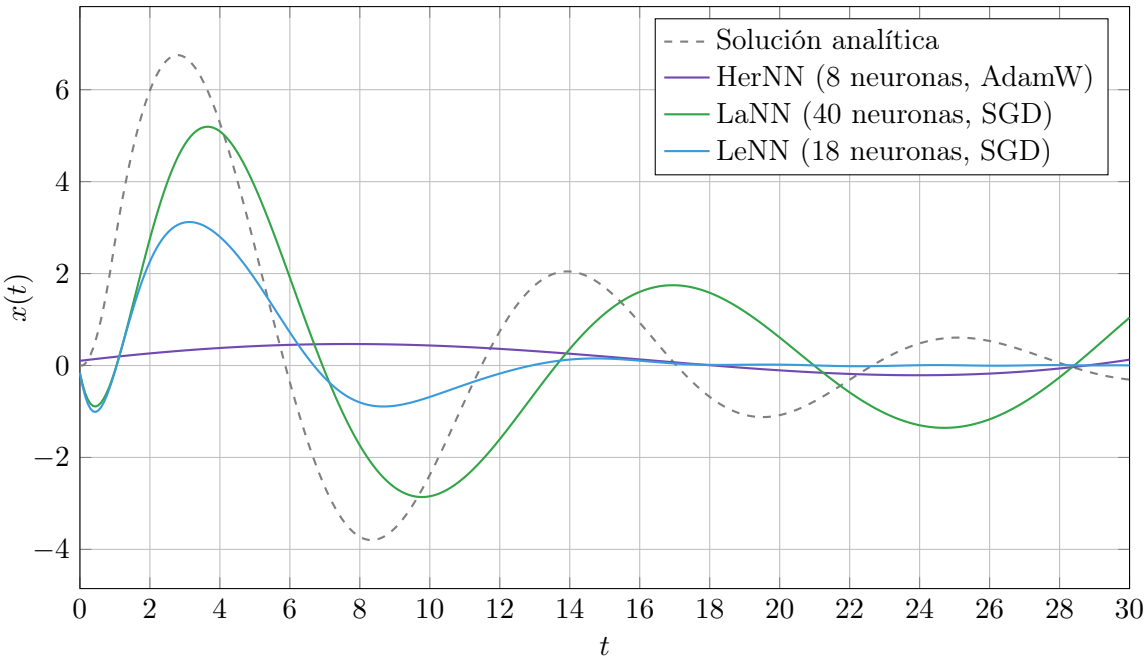


Figura 8.4: Comparación de los mejores resultados de cada red con la solución analítica.

Discusión

Al comparar directamente las tres redes bajo métodos de primer orden se observa una clara ventaja de LaNN y LeNN sobre HerNN. Mientras que HerNN, incluso con su mejor configuración mediante AdamW con 8 neuronas, apenas alcanzó un r negativo y un RMSE superior a 2.4, tanto LaNN como LeNN lograron métricas con valores positivos de r y errores más bajos. LaNN, entrenada con SGD, consiguió un r de 0.48 con un RMSE de 1.70, y LeNN, también con SGD, mejoró aún más hasta un r de 0.52 y un RMSE de 1.64. Los tiempos de ejecución no mostraron diferencias relevantes entre las tres configuraciones, permaneciendo en torno a los tres segundos, de manera que la superioridad de LaNN y LeNN se sustentó únicamente en la calidad del ajuste. Dentro de este marco, LeNN con SGD ofreció el resultado más sólido y confiable.

8.1.2 | Métodos de segundo orden

El código general creado para construir las distintas redes con los distintos métodos de entrenamiento se puede encontrar en el Apéndice A.3. A continuación se presentan tanto una tabla como un gráfico que contiene los métodos mencionados en la subsección 3.3.2 con la mejor configuración encontrada para cada red posterior a la realización de la búsqueda de hiperparámetros.

8.1.2.1 | Resultados para: HerNN

Tabla 8.5: Mejores resultados para la red HerNN. El tiempo fue medido en segundos.

Método	Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r
BFGS	No hubo convergencia						
L-BFGS-B	12	0.540200	850.759802	1.638143	5.334147	2.309577	0.062121
Newton-CG	8	0.001302	894.176517	1.701393	5.914208	2.431914	-0.039868

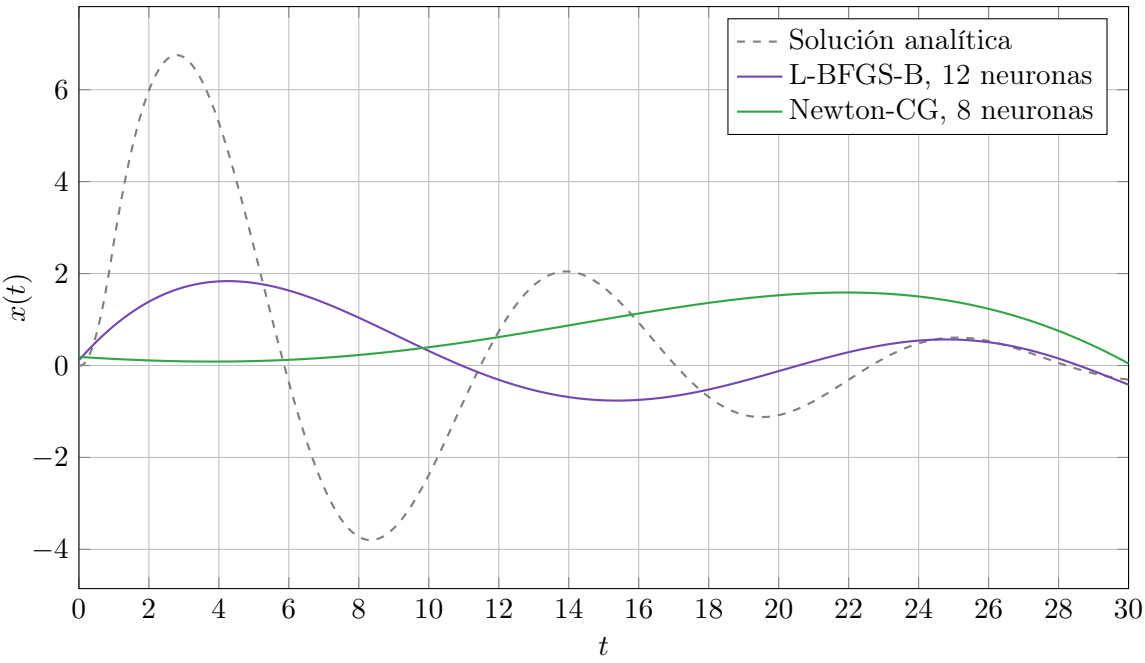


Figura 8.5: Comparativa de los mejores resultados obtenidos para HerNN con la solución analítica.

Discusión

En el caso de HerNN, los métodos de segundo orden tampoco ofrecieron resultados alentadores, puesto que los valores de las métricas permanecieron muy por debajo de lo esperado. Ni L-BFGS-B ni Newton-CG lograron un ajuste adecuado, ya que las pérdidas fueron extremadamente elevadas, superando 850 en ambos casos, y los errores RMSE se mantuvieron en torno a 2.3 con r apenas cercanos a cero o negativos. El tiempo de ejecución fue llamativamente bajo en Newton-CG, que resolvió en milisegundos, pero este rendimiento computacional no se tradujo en una mejor aproximación. En cambio, BFGS no alcanzó convergencia, lo que confirma que para esta red los métodos de segundo orden no constituyen una alternativa viable.

8.1.2.2 | Resultados para: LaNN

Tabla 8.6: Mejores resultados para la red LaNN. El tiempo fue medido en segundos.

Método	Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r
BFGS	38	0.008935	115.899486	0.265721	0.140835	0.375280	0.975238
L-BFGS-B	32	0.709823	119.277376	0.418471	0.250054	0.500054	0.956034
Newton-CG	35	0.012226	115.877292	0.263660	0.142084	0.376940	0.975018

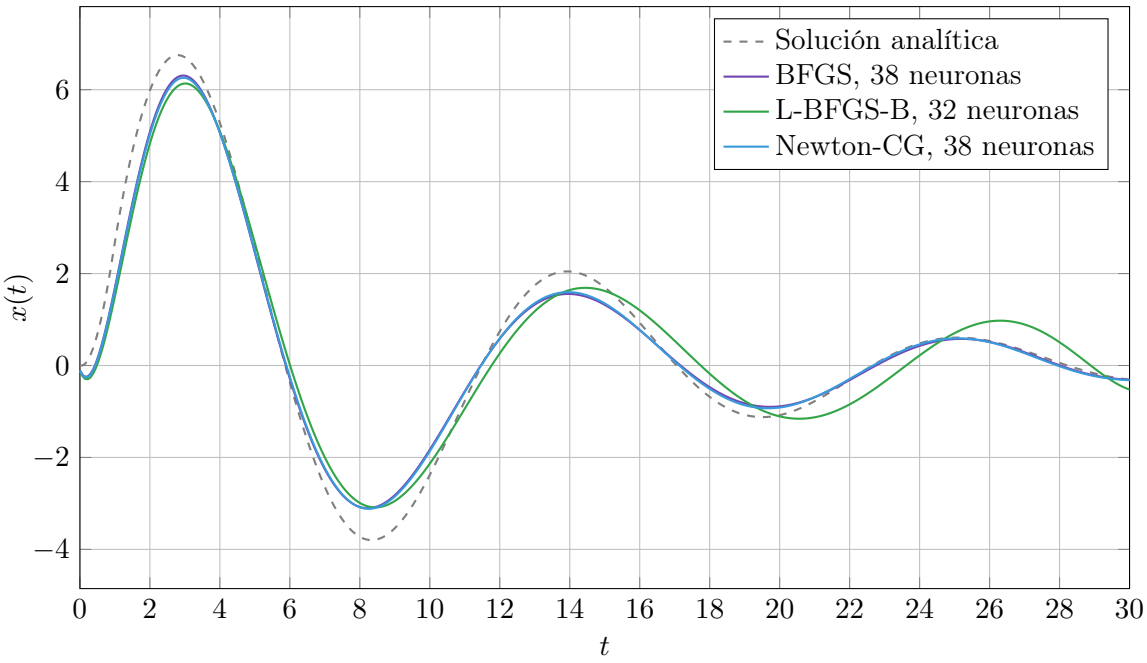


Figura 8.6: Comparativa de los mejores resultados obtenidos para LaNN con la solución analítica.

Discusión

Con LaNN los resultados fueron radicalmente distintos, ya que los métodos de segundo orden permitieron aproximaciones precisas. En particular, BFGS alcanzó métricas muy satisfactorias, con un r de 0.97, un RMSE de 0.37 y una pérdida de 115, lo que refleja un ajuste cercano a la solución de referencia. Newton-CG se comportó de forma muy similar, con resultados casi idénticos tanto en pérdida como en error, aunque con un tiempo de ejecución ligeramente mayor. L-BFGS-B, si bien menos preciso que los anteriores, mantuvo un buen desempeño con un r de 0.95 y un RMSE de 0.50, aunque requirió un mayor tiempo computacional. En este escenario, BFGS se consolidó como el método más confiable y equilibrado para entrenar LaNN mediante segundo orden.

8.1.2.3 | Resultados para: LeNN

Tabla 8.7: Mejores resultados para la red LeNN. El tiempo fue medido en segundos.

Método	Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r
BFGS	15	0.002378	97.315732	0.227707	0.082613	0.287425	0.985475
L-BFGS-B	15	0.265105	97.315760	0.227906	0.082819	0.287782	0.985438
Newton-CG	15	0.004550	97.315732	0.227707	0.082613	0.287425	0.985475

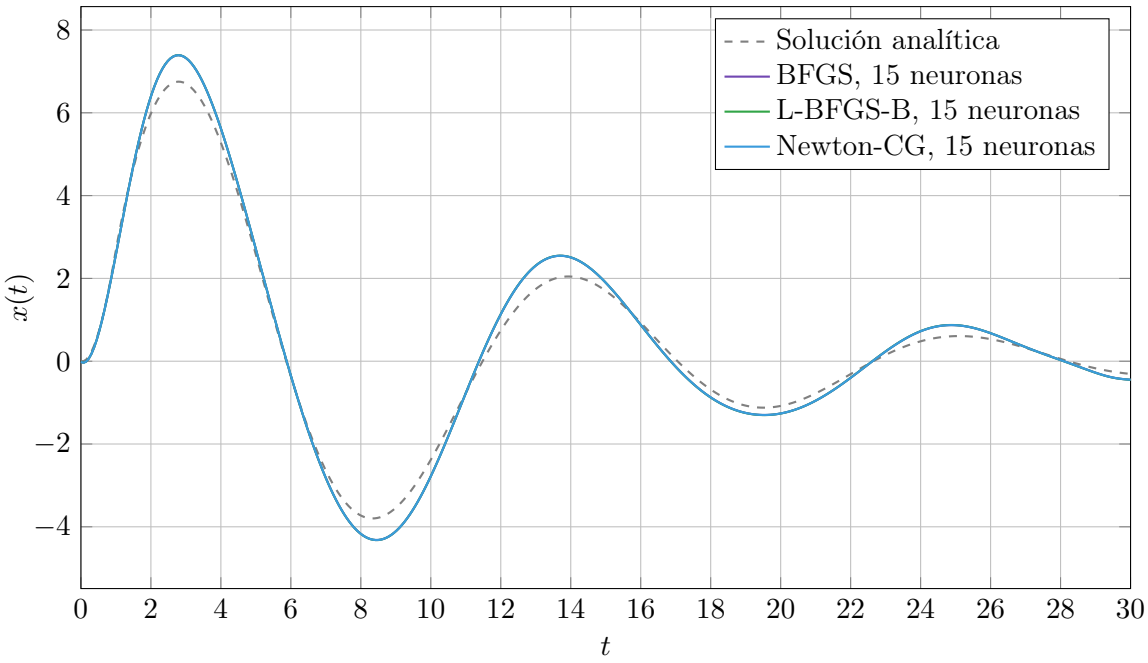


Figura 8.7: Comparativa de los mejores resultados obtenidos para LeNN con la solución analítica.

Discusión

En la aplicación de métodos de segundo orden a LeNN se obtuvieron resultados altamente consistentes y satisfactorios. Tanto BFGS como Newton-CG y L-BFGS-B produjeron métricas prácticamente equivalentes, con r en torno a 0.985 y errores muy bajos, con RMSE apenas por encima de 0.28. Las pérdidas se mantuvieron todas alrededor de 97, lo que confirma la estabilidad de los resultados. La diferencia más evidente se encontró en los tiempos de ejecución, pues BFGS y Newton-CG resolvieron en pocos milisegundos, mientras que L-BFGS-B requirió más de 0.26 segundos. De esta forma, aunque los tres métodos alcanzaron un ajuste sobresaliente, BFGS y Newton-CG constituyeron las alternativas más eficientes para esta red.

8.1.2.4 | Mejores resultados (HerNN vs. LaNN vs. LeNN)

Finalmente, se presenta un resumen de los mejores métodos para cada red, luego, se comparan las tres redes para ver cuál fue la de mejor desempeño.

Tabla 8.8: Mejores resultados para cada una de las redes. El tiempo fue medido en segundos.

Red	Método	Neuronas	Tiempo	Pérdida	RMSE	r
HerNN	L-BFGS-B	12	0.540200	850.759802	2.309577	0.062121
LaNN	BFGS	38	0.008935	115.899486	0.375280	0.975238
LeNN	Newton-CG	15	0.004550	97.315732	0.287425	0.985475

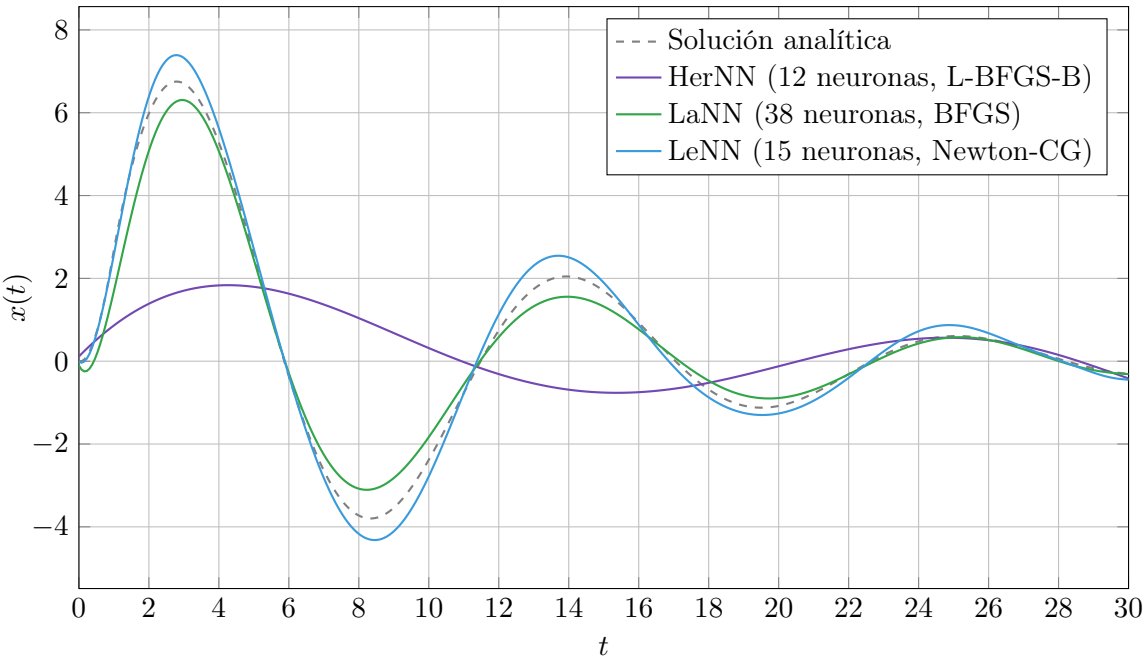


Figura 8.8: Comparación de los mejores resultados de cada red con la solución analítica.

Discusión

En la comparación de los resultados de segundo orden entre las tres redes se aprecia una marcada superioridad de LaNN y LeNN sobre HerNN. Esta última apenas alcanzó un r de 0.06 con un RMSE de 2.3, lo que contrasta con los ajustes logrados por las otras dos arquitecturas. LaNN, entrenada con BFGS, obtuvo un r de 0.97 y un RMSE de 0.37, mientras que LeNN con Newton-CG superó esos valores con un r de 0.98 y un RMSE de 0.28. Además, tanto LaNN como LeNN ofrecieron tiempos de cómputo reducidos, muy por debajo de un segundo, con lo cual no solo se logró un ajuste más preciso sino también una eficiencia destacable. Bajo estas condiciones, LeNN con Newton-CG se perfiló como la mejor opción en el ámbito de segundo orden.

8.1.3 | Extreme Learning Machine (ELM)

Para la construcción de las redes usando el método de Extreme Learning Machine se partió de la suposición de la salida de la red como ansatz, así, reemplazando directamente la salida de la red (5.1.1) en la ecuación de Bagley-Torvik (2.3.4), resulta

$$
\begin{aligned}
m\frac{\mathrm{d}^2}{\mathrm{d}t^2}\left(\sum_{i=1}^{N}\omega_i\Psi_i(t_j)\right) + 2A\sqrt{\mu\rho}\,{}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2}\left(\sum_{i=1}^{N}\omega_i\Psi_i(t_j)\right) + \kappa\left(\sum_{i=1}^{N}\omega_i\Psi_i(t_j)\right) &= f(t_j)\\
\sum_{i=1}^{N}\omega_i\left(m\frac{\mathrm{d}^2}{\mathrm{d}t^2}\Psi_i(t_j)\right) + \sum_{i=1}^{N}\omega_i\left(2A\sqrt{\mu\rho}\,{}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2}\Psi_i(t_j)\right) + \sum_{i=1}^{N}\omega_i\left(\kappa\Psi_i(t_j)\right) &= f(t_j)\\
\sum_{i=1}^{N}\omega_i\left[m\frac{\mathrm{d}^2}{\mathrm{d}t^2}\Psi_i(t_j) + 2A\sqrt{\mu\rho}\,{}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2}\Psi_i(t_j) + \kappa\Psi_i(t_j)\right] &= f(t_j),
\end{aligned}
$$

así, tomando la matriz $\mathbb{A}$ donde sus elementos vienen dados por la expresión

$$
A_{ij} = m\frac{\mathrm{d}^2}{\mathrm{d}t^2}\Psi_i(t_j) + 2A\sqrt{\mu\rho}\,{}^{\mathrm{RL}}_{0}\mathscr{D}_t^{3/2}\Psi_i(t_j) + \kappa\Psi_i(t_j)
$$

y llamando $\mathbf{w}_i = [\omega_1, \omega_2, \ldots, \omega_i]^T$ el vector de pesos de salida junto a $\mathbf{f}_j = [f_1, f_2, \ldots, f_j]^T$ el vector de forzamiento, se obtiene el sistema lineal

$$\sum_{i=0}^{N} \omega_i A_{ij} = f_j \Longrightarrow \mathbb{A}\mathbf{w} = \mathbf{f}, \quad j = 1, 2, 3, \ldots, n, \tag{8.1.1}$$

donde n es el número de puntos que tiene el dominio $t \in [0, 30]$ tras discretizarlo. Las condiciones iniciales se agregan como una fila extra en cada vector con su respectivo valor en el vector de forzamiento, también se le asocia un peso debido a que también se debe hacer un entrenamiento de la condición inicial para reproducir correctamente los resultados.

A continuación se muestran algunos resultados principales de la búsqueda de hiperparámetros realizada para cada una de las redes:

8.1.3.1 | Resultados para: HerNN

Tabla 8.9: Fragmento de la búsqueda de hiperparámetros realizada para HerNN. El tiempo fue medido en segundos.

Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r
6	0.057728	13.591011	3.136662	13.591011	3.686599	-1.389645
8	0.051073	0.326171	0.502157	0.326171	0.571114	0.942651
10	0.044678	0.053604	0.147941	0.053604	0.231525	0.990575
12	0.064476	0.047872	0.148960	0.047872	0.218796	0.991583
14	0.061282	1.291126	1.002651	1.291126	1.136277	0.772987
16	0.066581	1.375234	1.033734	1.375234	1.172704	0.758199
18	0.076318	1.479614	1.070953	1.479614	1.216394	0.739846

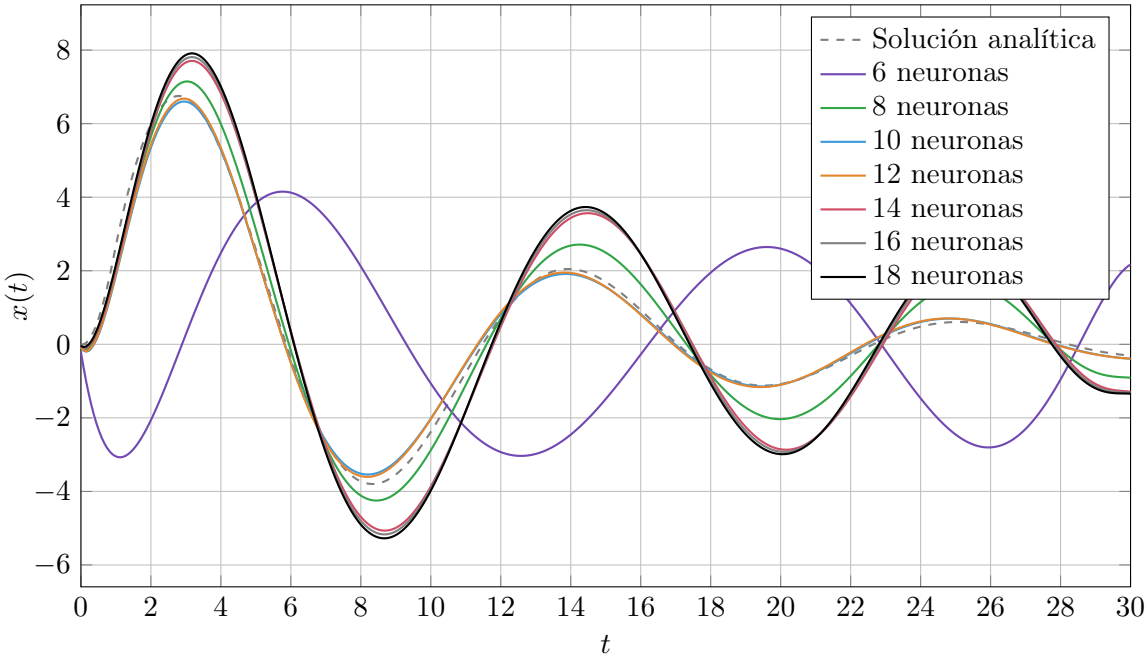


Figura 8.9: Comparación de varias HerNN con varias configuraciones neuronales con la solución analítica.

Discusión

En la búsqueda de hiperparámetros de HerNN bajo ELM se aprecia un comportamiento no monótono en las métricas conforme aumenta el número de neuronas. Con seis neuronas el desempeño fue deficiente, ya que la pérdida superó 13 y el r fue negativo, lo que indica un ajuste inadecuado. La calidad del resultado mejoró de manera drástica a partir de ocho neuronas, con una pérdida de 0.32, un RMSE de 0.57 y un r de 0.94. El rendimiento se optimizó con diez y doce neuronas, alcanzando pérdidas inferiores a 0.06, errores mínimos (RMSE de 0.23 y 0.21 respectivamente) y valores de r superiores a 0.99, que representan el ajuste más preciso. Sin embargo, al aumentar a 14 neuronas o más, la red comenzó a perder capacidad predictiva, ya que las métricas se deterioraron con pérdidas en torno a 1.2-1.4 y r por debajo de 0.77. Estos resultados confirman que la mejor configuración para HerNN con ELM se encuentra en un rango estrecho entre 10 y 12 neuronas.

8.1.3.2 | Resultados para: LaNN

Tabla 8.10: Fragmento de la búsqueda de hiperparámetros realizada para LaNN. El tiempo fue medido en segundos.

Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r
32	0.240864	0.059161	0.183740	0.059161	0.243230	0.989598
34	0.448183	0.055681	0.179848	0.055681	0.235968	0.990210
36	0.534314	0.052996	0.177875	0.052996	0.230208	0.990682
38	0.475863	0.050871	0.177131	0.050871	0.225546	0.991056
40	0.548537	0.063315	0.201183	0.063315	0.251625	0.988868
42	0.653444	0.068141	0.207774	0.068141	0.261039	0.988019
44	0.885340	0.072705	0.213810	0.072705	0.269638	0.987217

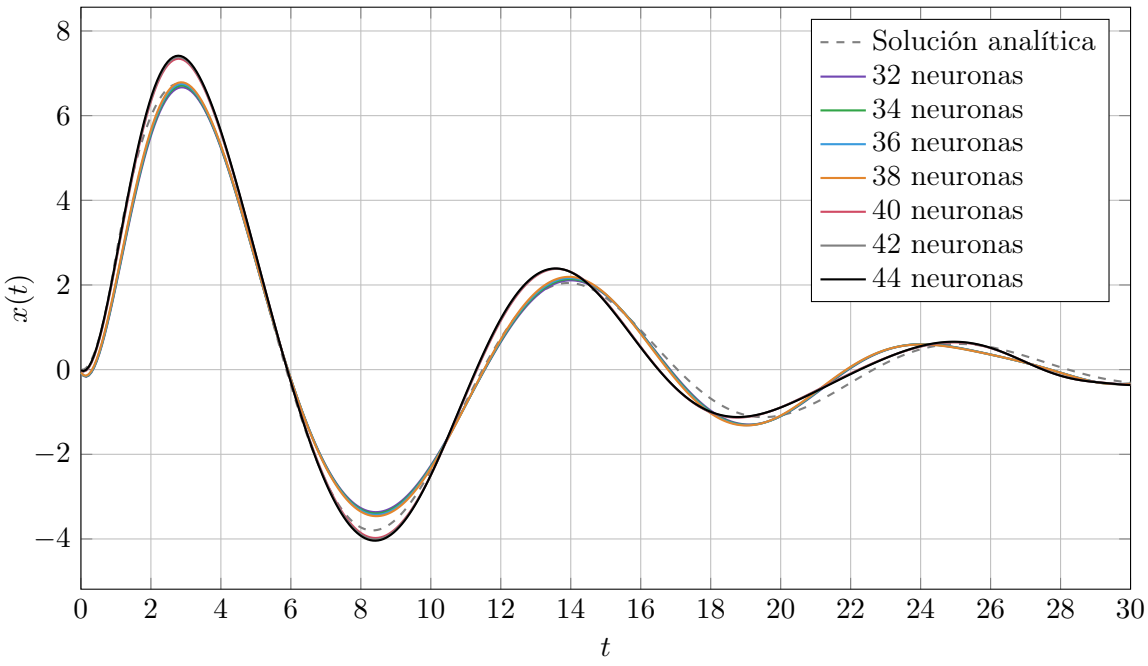


Figura 8.10: Comparación de varias LaNN con varias configuraciones neuronales con la solución analítica.

Discusión

En LaNN el ajuste mediante ELM mostró un desempeño altamente estable, con valores de pérdida reducidos y r superiores a 0.98 en todas las configuraciones exploradas. La precisión aumentó progresivamente entre 32 y 38 neuronas, alcanzando en este último caso la mejor combinación de métricas

con una pérdida de 0.05, un RMSE de 0.22 y un r de 0.99. A partir de 40 neuronas, la calidad de los resultados empezó a decaer ligeramente, con incrementos graduales en la pérdida y el error, además de un descenso en r hacia valores cercanos a 0.987. Los tiempos de ejecución crecieron conforme aumentó el número de neuronas, sin embargo, se mantuvieron por debajo de un segundo en todos los casos. En consecuencia, la configuración de 38 neuronas representó el mejor balance entre precisión y coste computacional para LaNN bajo este esquema de entrenamiento.

8.1.3.3 | Resultados para: LeNN

Tabla 8.11: Fragmento de la búsqueda de hiperparámetros realizada para LeNN. El tiempo fue medido en segundos.

Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r
8	0.157835	0.353218	0.525061	0.353218	0.594322	0.937895
10	0.094486	0.076608	0.182905	0.076608	0.276782	0.986530
12	0.094307	0.119044	0.212234	0.119044	0.345028	0.979069
14	0.101933	0.040353	0.155055	0.040353	0.200881	0.992905
16	0.121097	0.215661	0.369643	0.215661	0.464393	0.962081
18	0.128017	0.561420	0.580563	0.561420	0.749280	0.901288
20	0.151335	0.664469	0.629147	0.664469	0.815150	0.883169

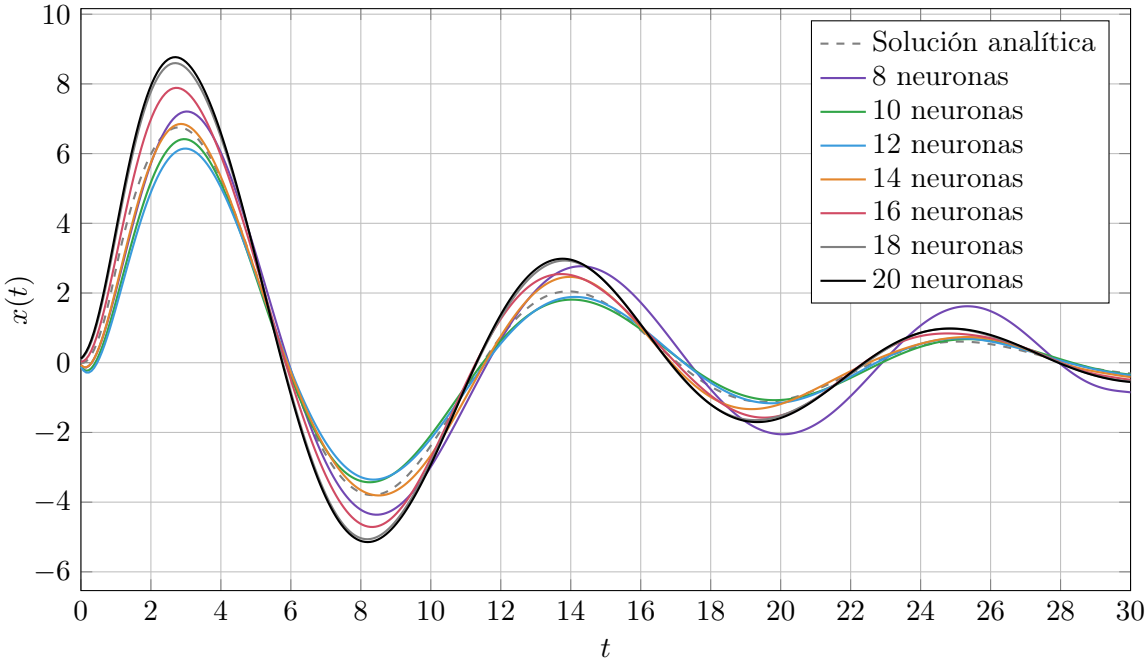


Figura 8.11: Comparación de varias LeNN con varias configuraciones neuronales con la solución analítica.

Discusión

En LeNN los resultados de ELM evidenciaron una marcada dependencia del desempeño respecto al número de neuronas. Con configuraciones bajas, como ocho neuronas, el r alcanzó 0.93, lo cual representó un ajuste aceptable aunque con errores todavía altos (RMSE de 0.59). El rendimiento mejoró significativamente con diez y catorce neuronas, alcanzando valores sobresalientes, especialmente en el caso de 14 neuronas donde se obtuvo la menor pérdida (0.04), un RMSE de 0.20 y un r de 0.993, constituyendo el mejor resultado entre todas las configuraciones evaluadas. En contraste, al aumentar el número de

neuronas por encima de 16, la calidad de la aproximación decreció de forma evidente, con pérdidas crecientes y r que descendió hasta 0.88 con 20 neuronas. Estos resultados confirman que, al igual que en HerNN, existe un rango óptimo acotado, en este caso en torno a 14 neuronas, donde la red logra su mejor rendimiento.

8.1.3.4 | Mejores resultados (HerNN vs. LaNN vs. LeNN)

Finalmente, se presenta un resumen de las mejores configuraciones neuronales para cada red comparando las tres redes para ver cuál fue la de mejor desempeño.

Tabla 8.12: Mejores resultados para cada una de las redes aplicando ELM. El tiempo fue medido en segundos.

Red	Neuronas	Tiempo	Pérdida	MAE	MSE	RMSE	r
HerNN	12	0.064476	0.047872	0.148960	0.047872	0.218796	0.991583
LaNN	38	0.475863	0.050871	0.177131	0.050871	0.225546	0.991056
LeNN	14	0.101933	0.040353	0.155055	0.040353	0.200881	0.992905

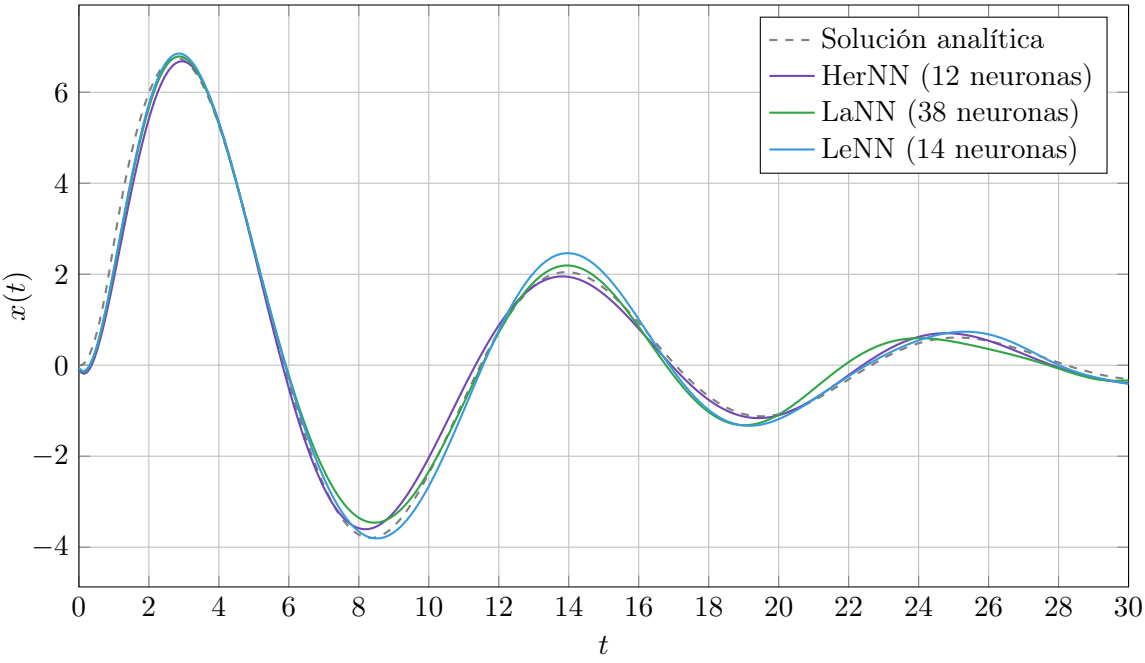


Figura 8.12: Comparación de las mejores redes entrenadas con ELM, con la solución analítica.

Discusión

La comparación global de ELM entre las tres arquitecturas confirma la notable superioridad de este método frente a los enfoques iterativos, tanto en precisión como en eficiencia computacional. En HerNN se encontró un rango óptimo entre 10 y 12 neuronas, donde las pérdidas descendieron a valores inferiores a 0.05 y el r se mantuvo por encima de 0.99, aunque con menos de ocho o más de catorce neuronas el rendimiento se degradó rápidamente. En LaNN el comportamiento fue más estable y la mejor configuración se alcanzó con 38 neuronas, logrando un RMSE de 0.22 y un r de 0.99, mientras que un mayor número de neuronas solo añadió coste computacional sin aportar mejoras. En LeNN se identificó un punto óptimo particularmente claro en 14 neuronas, donde se registró la pérdida más baja (0.04), el menor error (RMSE de 0.20) y el mayor r (0.993), superando a las otras dos redes. De este modo, aunque las tres arquitecturas demostraron una excelente capacidad de ajuste bajo ELM, LeNN con 14 neuronas se consolidó como la configuración más precisa y eficiente.

Capítulo 9

Análisis de resultados y conclusiones

El presente capítulo sintetiza y contrasta los resultados obtenidos a lo largo de las experimentaciones, comparando de manera integrada los distintos esquemas de entrenamiento y las tres arquitecturas de redes neuronales ortogonales consideradas (HerNN, LaNN y LeNN) con el objetivo de identificar tanto la mejor estrategia de entrenamiento como la arquitectura más adecuada para el problema planteado a la ecuación Bagley-Torvik (véase el Capítulo 7). La comparación se realiza sobre la base de las métricas reportadas: pérdida (con base en la función de costo asociada), MAE (*Mean Absolute Error*), MSE (*Mean Square Error*), RMSE (*Root Mean Square Error*), r (coeficiente de correlación); y de los tiempos de cómputo medidos en cada configuración, y además incorpora consideraciones teórico-numéricas relevantes, en particular la influencia del intervalo de convergencia de los polinomios ortogonales y la sensibilidad del cálculo numérico de la derivada fraccionaria.

Los resultados ponen de manifiesto un patrón claro: el esquema de entrenamiento mediante *Extreme Learning Machine* (ELM) produjo, en conjunto, las mejores aproximaciones en términos de precisión y tiempo de cálculo, mientras que los métodos iterativos se comportaron de manera heterogénea según la arquitectura y el orden del optimizador empleado. Específicamente, las ejecuciones con ELM mostraron métricas muy favorables para las tres redes evaluadas: HerNN alcanzó una pérdida de 0,047872, RMSE = 0,218796 y r = 0,991583 en aproximadamente 0,068 s; LaNN presentó una pérdida de 0,050871, RMSE = 0,225546 y r = 0,991056 con 0,381 s de cómputo; LeNN consiguió la mejor configuración global con pérdida 0,040353, RMSE = 0,200881 y $r = 0{,}992905$ en 0,0949 s, lo cual evidencia que ELM permite combinar alta precisión con tiempos de respuesta extremadamente bajos en este problema concreto (véase las tablas y figuras correspondientes del Capítulo 8). Estas cifras sitúan a ELM como la alternativa más eficiente desde la doble perspectiva de exactitud y rapidez, coherente con la naturaleza cerrada de su solución y con los resultados reportados en la bibliografía sobre ELM aplicada a problemas de ecuaciones diferenciales y aproximación funcional [22], [36].

Entre los métodos iterativos, la distinción por orden resulta determinante. Los optimizadores de primer orden (SGD, Adam, AdamW, RMSprop, Adagrad) ofrecieron rendimientos modestos y con variabilidad importante según la arquitectura: HerNN mostró comportamientos insatisfactorios con estos métodos, alcanzando valores de r negativos y errores elevados (por ejemplo, Adam produjo pérdida $\approx$ 2,20, RMSE $\approx$ 2,41 y $r \approx -0{,}025$), mientras que LaNN y LeNN consiguieron ajustes razonables con SGD (LaNN: $r \approx 0{,}487$, RMSE $\approx$ 1,708; LeNN: $r \approx 0{,}523$, RMSE $\approx$ 1,648), aunque con errores aún lejos de los obtenidos por ELM. Por su parte, los métodos de segundo orden demostraron un comportamiento marcadamente superior para LaNN y LeNN: BFGS y Newton-CG alcanzaron para LaNN $r \approx 0{,}975$ y RMSE $\approx$ 0,375 (con BFGS mostrando la pérdida reportada en $\sim$ 115,90 y tiempo de cómputo 0,0089 s), mientras que para LeNN los métodos de segundo orden alcanzaron $r \approx 0{,}985$ y RMSE $\approx$ 0,287 (pérdida $\sim$ 97,32, con Newton-CG ejecutándose en 0,0046 s), resultados que los sitúan muy próximos a la solución de referencia y notablemente por encima de los alcanzados por optimizadores de primer orden para las mismas arquitecturas. En contraste, HerNN no se benefició de manera apreciable de los métodos de segundo orden en las condiciones ensayadas: BFGS no convergió y L-BFGS-B y Newton-CG mostraron pérdidas extremadamente elevadas y errores RMSE del orden de 2,3, con r cercanos a cero, lo que sugiere una interacción desfavorable entre esa arquitectura y las características del problema bajo los

parámetros utilizados.

La interpretación de estos patrones exige atender a la relación entre la base ortogonal empleada, su intervalo verdadero de convergencia y la naturaleza del dominio físico de la ecuación de Bagley-Torvik. Los polinomios de Legendre están naturalmente definidos y convergen de modo uniforme sobre intervalos cerrados como $[-1, 1]$, lo que facilita una aproximación estable cuando el dominio temporal se reescalifica adecuadamente; los polinomios de Laguerre son especialmente adecuados para dominios semidefinidos $[0, \infty)$ y, por tanto, resultan conceptualmente coherentes con problemas definidos en tiempo positivo sin necesidad de mapeos complejos; los polinomios de Hermite, por su parte, son idóneos para espacios con soporte en la recta entera y con peso gaussiano, pero su comportamiento en intervalos largos y positivos (como $[0, 30]$) puede requerir un tratamiento cuidadoso para evitar desbordamientos o pérdida de ortogonalidad efectiva. En el experimento concreto, el dominio temporal elegido ($t \in [0, 30]$) favorece de manera natural a bases que se adapten bien al semieje positivo o que puedan ser mapeadas con estabilidad numérica; de ahí que LaNN (Laguerre) y LeNN (Legendre, tras escalado) brinden resultados sobresalientes, y que LeNN, una vez ajustada apropiadamente, sea capaz de ofrecer la combinación más favorable entre convergencia de la serie ortogonal y condición numérica del problema, tal y como se refleja en las mejores métricas obtenidas con ELM y con optimizadores de segundo orden. Estas observaciones están en consonancia con el análisis teórico sobre propiedades de convergencia y ortogonalidad de polinomios clásicos y con las recomendaciones prácticas de la literatura al elegir bases ortogonales según el soporte del problema, confirmando la pertinencia del estudio comparativo desarrollado en torno a las redes neuronales ortogonales.

No obstante, es imprescindible subrayar que las discrepancias observadas entre métodos de entrenamiento y arquitecturas no se explican únicamente por la elección del optimizador o por la propiedad de la base; una fuente dominante de variabilidad fue la aproximación numérica de la derivada fraccionaria en la ecuación de Bagley-Torvik, cuyo cálculo requiere discretizaciones sensibles y operadores con memoria que amplifican errores de truncamiento y redondeo. Los esquemas comunes para la derivada de orden fraccionario (por ejemplo, formulaciones de Riemann-Liouville o aproximaciones basadas en series de potencias y fórmulas de tipo Grünwald-Letnikov) implican convoluciones discretas con kernels tipo potencia que son intrínsecamente mal condicionados para pasos discretos grandes o para ventanas temporales extensas; esta propiedad numérica provoca acumulación de error, alta sensibilidad al tamaño de paso y, en ocasiones, matrices de diseño con mal condicionamiento que afectan tanto a procesos iterativos de optimización como al cálculo de la pseudo-inversa en ELM [6]. En la práctica, esto se tradujo en variaciones de métricas que, tras un análisis cuidadoso, pueden atribuirse en gran medida a la estabilidad numérica del operador fraccionario y no exclusivamente a la capacidad de representación de la red o al optimizador escogido, una conclusión que coincide con discusiones clásicas sobre estabilidad numérica en optimización y regularización de problemas inversos [35], [42]. Este análisis teórico y experimental confirma así la validez del marco conceptual abordado sobre el cálculo fraccionario y su aplicación directa a la ecuación de Bagley-Torvik.

En consecuencia, integrando la evidencia empírica y la discusión teórico-numérica, la conclusión que emana del análisis es doble y concreta: por un lado, entre los métodos de entrenamiento evaluados, **ELM constituye la opción más recomendable para abordar la ecuación de Bagley-Torvik en las condiciones experimentales consideradas**, al proporcionar la mejor combinación de precisión (RMSE y r superiores a las alternativas) y eficiencia computacional (tiempos de ejecución por debajo de la décima de segundo en las mejores configuraciones); por otro lado, **la arquitectura que mejor se adapta al problema es LeNN (red neuronal de Legendre) empleada con ELM**, ya que registró la pérdida más baja (0.04035), el RMSE más reducido (0.20088) y el mayor r (0.9929) dentro del conjunto de pruebas, lo que indica una aproximación más precisa y consistente frente a LaNN y HerNN bajo las mismas condiciones. Esta elección queda además justificada por la adecuación de la base de Legendre a aproximaciones sobre intervalos cerrados una vez que el dominio temporal ha sido reescalado apropiadamente, lo cual reduce efectos de oscilación y mejora la convergencia uniforme de la serie de aproximación, dando cumplimiento al propósito general de contrastar el desempeño de las

distintas redes ortogonales en la aproximación numérica de un modelo fraccionario de referencia.

No obstante, se recomienda prudencia metodológica: antes de generalizar la recomendación conviene explorar dos aspectos adicionales en trabajos futuros o implementaciones prácticas, a saber,

(i) optimizar la discretización del operador fraccionario y adoptar esquemas numéricos con control de error y regularización explícita para el cálculo de derivadas fraccionarias, dado que la inestabilidad numérica del operador fue una de las causas principales de discrepancias entre métodos; y

(ii) evaluar la robustez de la elección de la base respecto al reescalado del dominio y al incremento del grado polinomial, especialmente en casos donde el dominio real difiera sensiblemente de los intervalos canónicos de ortogonalidad.

De la misma forma, en aplicaciones donde el dominio sea semiinfinito o donde resulte natural el uso de pesos exponenciales, LaNN (Laguerre) puede representar una alternativa competitiva, y en problemas con soporte sobre la recta completa la HerNN (Hermite) puede ser preferible si se emplean transformaciones y técnicas de control numérico apropiadas [13], [22], [36], [40].

Finalmente, esta monografía se cierra con la exposición de un estudio teórico integral sobre las aplicaciones del cálculo fraccionario en el ámbito de la física matemática, complementado con la revisión y aplicación de arquitecturas ortogonales basadas en distintos polinomios clásicos y con la implementación comparativa de estrategias de entrenamiento en la resolución numérica de la ecuación de Bagley-Torvik. El trabajo realizado permite constatar no solo la amplitud de los marcos teóricos abordados, sino también su potencial de aplicación en problemas físicos diversos, lo que abre un horizonte de investigación prácticamente inagotable. Resulta natural preguntarse cómo se comportarían estas aproximaciones si, en lugar de ecuaciones ordinarias, se consideraran ecuaciones en derivadas parciales, o qué aportes podrían surgir al emplear familias de polinomios ortogonales distintas de las clásicas. Asimismo, surge el interrogante de si es posible concebir redes neuronales ortogonales en las que no solo las neuronas, sino también las capas, respondan a estructuras ortogonales, o incluso extender la construcción hacia arquitecturas más complejas como redes recurrentes o convolucionales. Estas y muchas otras cuestiones emergidas durante el desarrollo de la investigación constituyen un punto de partida fértil para futuros trabajos, consolidando así la relevancia académica de este campo y su proyección hacia nuevas fronteras del conocimiento.

Muchas gracias por leer.

Referencias

[1] R. O. E. Libardo y F. J. de Caldas, *Tres documentos del coronel de ingenieros Francisco José de Caldas : padre de la ingeniería colombiana y fundador de nuestra facultad*. Medellín: Editorial Universidad de Antioquia, abr. de 2010, ISBN: 9789587143706. dirección: https://bibliotecadigital.udea.edu.co/server/api/core/bitstreams/50faf3b9-6fc7-4223-bec5-c8d025feca1d/content.

[2] P. J. Torvik y R. L. Bagley, «On the Appearance of the Fractional Derivative in the Behavior of Real Materials,» *Journal of Applied Mechanics*, vol. 51, n.º 2, págs. 294-298, jun. de 1984, ISSN: 1528-9036. DOI: 10.1115/1.3167615.

[3] R. L. Bagley y P. J. Torvik, «A Theoretical Basis for the Application of Fractional Calculus to Viscoelasticity,» *Journal of Rheology*, vol. 27, n.º 3, págs. 201-210, jun. de 1983, ISSN: 1520-8516. DOI: 10.1122/1.549724.

[4] T. M. Atanackovic y D. Zorica, «On the Bagley-Torvik Equation,» *Journal of Applied Mechanics*, vol. 80, n.º 4, mayo de 2013, ISSN: 1528-9036. DOI: 10.1115/1.4007850.

[5] R. Herrmann, *Fractional calculus, An introduction for physicists*, 2. ed. Singapore: World Scientific Publishing Company, 2014, Description based upon print version of record, ISBN: 9789814551090.

[6] D. Xue, *Fractional Calculus, High-Precision Algorithms and Numerical Implementations*, 1st ed. 2024., L. Bai, ed. Singapore: Imprint: Springer, 2024, ISBN: 9789819920709.

[7] I. Podlubny, *Fractional differential equations, An introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications* (Mathematics in science and engineering v. 198). San Diego: Academic Press, 1999, Includes bibliographical references (p. 313-335) and index., ISBN: 9780080531984.

[8] I. Batiha, S. Alshorman, I. Jebril y M. Hammad, «A Brief Review about Fractional Calculus,» *International Journal of Open Problems in Computer Science and Mathematics*, vol. 15, págs. 39-56, dic. de 2022.

[9] C. C. Aggarwal, *Neural networks and deep learning, A textbook*, Second Edition. Cham: Springer International Publishing AG, 2023, ISBN: 9783031296413.

[10] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016. dirección: http://www.deeplearningbook.org.

[11] S. S. Haykin, *Neural networks and learning machines*, 3. ed. New York: Prentice-Hall, 2009, ISBN: 0131471392.

[12] T. M. Mitchell, *Machine learning* (McGraw-Hill international editions), [Nachdr.] New York [u.a.]: McGraw-Hill, 2013, ISBN: 0071154671.

[13] S.-S. Yang y C.-S. Tseng, «An orthogonal neural network for function approximation,» *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 26, n.º 5, págs. 779-785, 1996, ISSN: 1083-4419. DOI: 10.1109/3477.537319.

[14] T. S. Chihara, *Introduction to Orthogonal Polynomials*. Dover Publications, Incorporated, 2011, ISBN: 9780486141411.

[15] W. Gautschi, *Orthogonal polynomials, Computation and approximation* (Oxford scholarship online). Oxford: Oxford University Press, 2020, Previously issued in print: 2004. - Includes bibliographical references and index. - Description based on print version record, ISBN: 9780191916571.

[16] R. E. N. A. T. O. ALVAREZ-NODARSE, *POLINOMIOS HIPERGEOMETRICOS Y Q-POLINOMIOS*. PRENSAS UNIVERSITARIAS DE ZARAGOZA, 2003, ISBN: 9788477336372.

[17] S. Mall y S. Chakraverty, «Application of Legendre Neural Network for solving ordinary differential equations,» *Applied Soft Computing*, vol. 43, págs. 347-356, jun. de 2016, ISSN: 1568-4946. DOI: 10.1016/j.asoc.2015.10.069.

[18] Y. Chen, H. Yu, X. Meng, X. Xie, M. Hou y J. Chevallier, «Numerical solving of the generalized Black-Scholes differential equation using Laguerre neural network,» *Digital Signal Processing*, vol. 112, pág. 103 003, mayo de 2021, ISSN: 1051-2004. DOI: 10.1016/j.dsp.2021.103003.

[19] S. Mall y S. Chakraverty, «Hermite Functional Link Neural Network for Solving the Van der Pol–Duffing Oscillator Equation,» *Neural Computation*, vol. 28, n.º 8, págs. 1574-1598, ago. de 2016, ISSN: 1530-888X. DOI: 10.1162/neco_a_00858.

[20] G. G. Rigatos y S. G. Tzafestas, «Feed-Forward Neural Networks Using Hermite Polynomial Activation Functions,» en *Advances in Artificial Intelligence*. Springer Berlin Heidelberg, 2006, págs. 323-333, ISBN: 9783540341185. DOI: 10.1007/11752912_33.

[21] A. Verma y M. Kumar, «Numerical solution of Bagley–Torvik equations using Legendre artificial neural network method,» *Evolutionary Intelligence*, vol. 14, n.º 4, págs. 2027-2037, sep. de 2020, ISSN: 1864-5917. DOI: 10.1007/s12065-020-00481-x.

[22] Y. Yang, M. Hou y J. Luo, «A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods,» *Advances in Difference Equations*, vol. 2018, n.º 1, dic. de 2018, ISSN: 1687-1847. DOI: 10.1186/s13662-018-1927-x.

[23] K. Diethelm y J. Ford, «Numerical Solution of the Bagley-Torvik Equation,» *BIT Numerical Mathematics*, vol. 42, n.º 3, págs. 490-507, sep. de 2002, ISSN: 1572-9125. DOI: 10.1023/a:1021973025166.

[24] S. S. Ray y R. Bera, «Analytical solution of the Bagley Torvik equation by Adomian decomposition method,» *Applied Mathematics and Computation*, vol. 168, n.º 1, págs. 398-410, sep. de 2005, ISSN: 0096-3003. DOI: 10.1016/j.amc.2004.09.006.

[25] G. W. Leibniz, C. Huygens y G. F. A. d. L'Hôpital, *Leibnizens gesammelte Werke, Leibnizens mathematische Schriften. Erste Abtheilung. Band II. Briefwechsel zwischen Leibniz, Huygens van Zulichem und dem Marquis de l'Hospital, aus den Handschriften der Königlichen Bibliothek zu Hannover*, G. H. Pertz, ed. Berlin: A. Asher & Comp., 1850, vol. 3, Mathematik ; 1,2, pág. 343, Digitalisiert von der Österreichischen Nationalbibliothek, Google, am 26.03.2015. dirección: https://data.onb.ac.at/rep/10A136D2.

[26] M. Lazarević, M. Rapaić y T. Šekara, «Introduction to Fractional Calculus with Brief Historical Background,» en ene. de 2014, págs. 3-16, ISBN: 978-960-474-348-3.

[27] L. Euler, «De progressionibus transcendentibus seu quarum termini generales algebraice dari nequeunt,» On transcendental progressions, that is, those whose general terms cannot be given algebraically, latín, *Commentarii academiae scientiarum Petropolitanae*, Opera Omnia: Series 1, vol. 5, págs. 36-57, 1738, Traducido del latín al inglés por S. G. Langton, University of San Diego, 1999. dirección: https://scholarlycommons.pacific.edu/euler-works/19/.

[28] N. H. Asmar, *Partial differential equations with Fourier series and boundary value problems*, 2. ed. Upper Saddle River, NJ: Pearson Prentice-Hall, 2005, [Rev. ed. of: Partial differential equations and boundary value problems], ISBN: 0131480960.

[29] B. R. Munson, ed., *Fundamentals of fluid mechanics*, 7. ed. Hoboken, NJ: Wiley, 2013, Includes indexes, ISBN: 9781118399712.

[30] I. Siddique y D. Vieru, «Stokes Flows of a Newtonian Fluid with Fractional Derivatives and Slip at the Wall,» *International Review of Physics (IREPHY)*, vol. 11, n.º 1, 2017, ISSN: 2533-2287. dirección: https://www.praiseworthyprize.org/jsm/index.php?journal=irephy&page=article&op=view&path%5B%5D=7269.

[31] H. Schlichting y K. Gersten, *Boundary-Layer Theory*. Springer Berlin Heidelberg, 2017, ISBN: 9783662529195. DOI: 10.1007/978-3-662-52919-5.

[32] F. Mainardi, «An historical perspective on fractional calculus in linear viscoelasticity: Short survey,» *Fractional Calculus and Applied Analysis*, vol. 15, n.º 4, págs. 712-717, sep. de 2012, ISSN: 1314-2224. DOI: 10.2478/s13540-012-0048-6.

[33] M. Fukunaga y N. Shimizu, «Fractional derivative models of viscoelastic materials for large extension,» en *ICFDA'14 International Conference on Fractional Differentiation and Its Applications 2014*, IEEE, jun. de 2014, págs. 1-5. DOI: 10.1109/icfda.2014.6967439.

[34] A. A. Zafar, G. Kudra y J. Awrejcewicz, «An Investigation of Fractional Bagley–Torvik Equation,» *Entropy*, vol. 22, n.º 1, pág. 28, dic. de 2019, ISSN: 1099-4300. DOI: 10.3390/e22010028.

[35] A. Ben-Israel, *Generalized Inverses, Theory and Applications* (Springer eBook Collection), Second Edition, T. N. E. Greville, ed. New York, NY: Springer New York, 2003, ISBN: 9780387216348.

[36] C. D. Vinodbhai y S. Dubey, «Numerical solution of neutral delay differential equations using orthogonal neural network,» *Scientific Reports*, vol. 13, n.º 1, feb. de 2023, ISSN: 2045-2322. DOI: 10.1038/s41598-023-30127-8.

[37] J. Wang, S. Lu, S.-H. Wang e Y.-D. Zhang, «A review on extreme learning machine,» *Multimedia Tools and Applications*, vol. 81, n.º 29, págs. 41 611-41 660, mayo de 2021, ISSN: 1573-7721. DOI: 10.1007/s11042-021-11007-7.

[38] I. Lagaris, A. Likas y D. Fotiadis, «Artificial neural networks for solving ordinary and partial differential equations,» *IEEE Transactions on Neural Networks*, vol. 9, n.º 5, págs. 987-1000, 1998, ISSN: 1045-9227. DOI: 10.1109/72.712178.

[39] K. Udaya Mohanan, S. Cho y B.-G. Park, «Optimization of the structural complexity of artificial neural network for hardware-driven neuromorphic computing application,» *Applied Intelligence*, vol. 53, n.º 6, págs. 6288-6306, jul. de 2022, ISSN: 1573-7497. DOI: 10.1007/s10489-022-03783-y.

[40] Y. Wang y Q. Ren, *Deep Learning-Based Forward Modeling and Inversion Techniques for Computational Physics Problems*. CRC Press, mayo de 2023, ISBN: 9781003397830. DOI: 10.1201/9781003397830.

[41] D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization,» dic. de 2014. DOI: 10.48550/ARXIV.1412.6980. arXiv: 1412.6980 [cs.LG].

[42] J. Nocedal, *Numerical optimization* (Springer series in operations research and financial engineering), Second edition, S. J. Wright, ed. New York, NY: Springer, 2006, ISBN: 9780387400655.

[43] S. Ruder, «An overview of gradient descent optimization algorithms,» sep. de 2016. DOI: 10.48550/ARXIV.1609.04747. arXiv: 1609.04747 [cs.LG].

[44] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain.,» *Psychological Review*, vol. 65, n.º 6, págs. 386-408, 1958, ISSN: 0033-295X. DOI: 10.1037/h0042519.

[45] R. Penrose, «A generalized inverse for matrices,» *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, n.º 3, págs. 406-413, jul. de 1955, ISSN: 1469-8064. DOI: 10.1017/s0305004100030401.

[46] T. Ji, J. Hou y C. Yang, «Numerical solution of the Bagley–Torvik equation using shifted Chebyshev operational matrix,» *Advances in Difference Equations*, vol. 2020, n.º 1, nov. de 2020, ISSN: 1687-1847. DOI: 10.1186/s13662-020-03110-0.

[47] S. ESMAEILI, «THE NUMERICAL SOLUTION OF THE BAGLEY-TORVIK EQUATION BY EXPONENTIAL INTEGRATORS,» *Scientia Iranica*, sep. de 2017, ISSN: 2345-3605. DOI: 10.24200/sci.2017.4503.

[48] I. Podlubny, «Matrix approach to discrete fractional calculus,» *Fractional Calculus & Applied Analysis (FCAA)*, vol. 3, ene. de 2000. dirección: http://www.tuke.sk/podlubny/papers.html.

Capítulo A

Códigos

A continuación se presentan la mayoría de los códigos utilizados. Se emplearon otros códigos, pero principalmente de forma complementaria, ya sea para extraer los datos tras el entrenamiento o para graficar los resultados. Estos no se incluyen, dado que no son de interés particular.

A.1 | Método de la matriz de Podlubny

A continuación se presenta el código utilizado para replicar el método de la matriz de Podlubny mostrada en [48]:

```
import numpy as np
import matplotlib.pyplot as plt

def fractional_weights(alpha, N):
    omega = np.zeros(N+1)
    omega[0] = 1.0
    for j in range(1, N+1):
        omega[j] = omega[j-1] * ((alpha - (j-1)) / j) * (-1.0)
    return omega

def fractional_matrix(alpha, N, h):
    omega = fractional_weights(alpha, N)
    B = np.zeros((N+1, N+1))
    for i in range(N+1):
        B[i, :i+1] = omega[:i+1][::-1]
    return B / (h**alpha)

def bagleytorvik(A, B, C, x0, y0, inicio, final, N):
    """
    Resuelve la ecuaci\'on A D^2 x + B D^1.5 x + C x = f(t),
    con x(0)=x0, x'(0)=y0, en [inicio, final] con N pasos.
    """
    h = (final - inicio)/N
    t = np.linspace(inicio, final, N+1)

    f = np.zeros_like(t)
    f[(t >= 0) & (t <= 1)] = 8.0

    B2 = fractional_matrix(2.0, N, h)
    B15 = fractional_matrix(1.5, N, h)
    M = A * B2 + B * B15 + C * np.eye(N+1)

    f = f.copy()
    M[0,:] = 0; M[0,0] = 1; f[0] = x0
    M[1,:] = 0; M[1,1] = 1; f[1] = y0

    x = np.linalg.solve(M, f)
```

```
38
39      fig, ax = plt.subplots(figsize=(10,6))
40      ax.plot(t, x, label="x(t)")
```

Listing A.1: Código en Python para el método de la matriz de Podlubny

A.2 | Métodos de primer orden

A continuación el código general usado para el entrenamiento de las redes con los métodos de primer orden:

```
1  def solve_tensorflow_method(method_name, learning_rate, epochs, A_full, f_full,
       X_basis):
2      """
3      Construcci\'on redes neuronales de primer orden para las diferentes familias de
           polinomios.
4
5      Args:
6          method_name: Nombre del optimizador ('Adam', 'SGD', etc.)
7          learning_rate: Tasa de aprendizaje
8          epochs: N\'umero de \'epocas
9          A_full, f_full: Sistema de ecuaciones
10         X_basis: Funciones base
11
12     Returns:
13         tuple: (y_solution, training_time, final_loss, success, history)
14     """
15
16     A_tensor = tf.constant(A_full, dtype=tf.float32)
17     f_tensor = tf.constant(f_full, dtype=tf.float32)
18
19     w = tf.Variable(tf.random.normal([len(X_basis)], stddev=0.01), dtype=tf.float32)
20
21     if method_name == 'SGD':
22         optimizer = tf.optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
23     elif method_name == 'Adam':
24         optimizer = tf.optimizers.Adam(learning_rate=learning_rate)
25     elif method_name == 'AdamW':
26         optimizer = tf.optimizers.AdamW(learning_rate=learning_rate,
               weight_decay=1e-4)
27     elif method_name == 'RMSprop':
28         optimizer = tf.optimizers.RMSprop(learning_rate=learning_rate)
29     elif method_name == 'Adagrad':
30         optimizer = tf.optimizers.Adagrad(learning_rate=learning_rate)
31     else:
32         optimizer = tf.optimizers.Adam(learning_rate=learning_rate)
33
34     @tf.function
35     def loss_fn():
36         residual = tf.linalg.matvec(A_tensor, w) - f_tensor
37         return tf.reduce_mean(tf.square(residual))
38
39     # Callbacks personalizados para monitoreo
40     class LossHistory:
41         def __init__(self):
42             self.losses = []
43             self.best_loss = float('inf')
44             self.patience_counter = 0
45             self.patience = 50
46
```

```
        def update(self, loss_val):
            self.losses.append(loss_val)
            if loss_val < self.best_loss:
                self.best_loss = loss_val
                self.patience_counter = 0
            else:
                self.patience_counter += 1

        def should_stop(self):
            return self.patience_counter >= self.patience

    start_time = time.time()
    history = LossHistory()

    for epoch in range(epochs):
        with tf.GradientTape() as tape:
            loss = loss_fn()

        gradients = tape.gradient(loss, [w])
        optimizer.apply_gradients(zip(gradients, [w]))

        loss_val = float(loss)
        history.update(loss_val)

        if history.should_stop():
            print(f"      Early stopping {epoch}")
            break

    training_time = time.time() - start_time

    w_final = w.numpy()
    y_solution = X_basis.T @ w_final

    return y_solution, training_time, history.best_loss, True, history.losses
```

Listing A.2: Código en Python para HerNN, LaNN y LeNN con métodos de entrenamiento de primer orden

A.3 | Métodos de segundo orden

A continuación el código general usado para el entrenamiento de las redes con los métodos de primer orden:

```
def setup(poly_type, n_neurons, t_vals, h, f_rhs):
    """
    Construcci\'on redes neuronales de segundo orden para las diferentes familias de
        polinomios.

    Args:
        poly_type: Tipo de polinomio
        n_neurons: N\'umero de neuronas
        t_vals: Dominio (valores de t)

    Returns:
        tuple: (X_basis, A_full, f_full)
    """

    if poly_type == 'Hermite':
        basis_func = hermite_basis
        deriv_func = hermite_deriv
        window = 60
```

```
18
19      elif poly_type == 'Laguerre':
20          basis_func = laguerre_basis
21          deriv_func = laguerre_deriv
22          window = 40
23
24      elif poly_type == 'Legendre':
25          basis_func = legendre_basis
26          deriv_func = legendre_deriv
27          window = 100
28
29      else:
30          raise ValueError(f"Tipo de polinomio no reconocido: {poly_type}")
31
32      print(f"    Sistema {poly_type}: {n_neurons} neuronas")
33
34      X_basis = np.array([basis_func(i, t_vals) for i in range(n_neurons + 2)])
35      X_deriv2 = np.array([deriv_func(i, t_vals, 2) for i in range(n_neurons + 2)])
36      X_frac = np.array([fractional_derivative(X_basis[i], h, alpha, window)
37                         for i in range(n_neurons + 2)])
38
39      A_matrix = (A * X_deriv2 + B * X_frac + C * X_basis).T
40
41      IC_matrix = np.array([
42          [basis_func(i, 0) for i in range(n_neurons + 2)],
43          [deriv_func(i, 0, 1) for i in range(n_neurons + 2)]
44      ])
45
46      A_full = np.vstack([A_matrix, IC_matrix])
47      f_full = np.concatenate([f_rhs, [0.0, 0.0]])
48
49      return X_basis, A_full, f_full
50
51  def solve_traditional_method(method_name, A_full, f_full, X_basis):
52      """
53      Construcci\'on redes neuronales de primer orden para las diferentes familias de
            polinomios.
54
55      Args:
56          method_name: Nombre del optimizador ('Adam', 'SGD', etc.)
57          A_full, f_full: Sistema de ecuaciones
58          X_basis: Funciones base
59
60      Returns:
61          tuple: (y_solution, training_time, result.fun)
62      """
63
64      def objective(w):
65          residual = A_full @ w - f_full
66          return np.sum(residual**2)
67
68      def gradient(w):
69          residual = A_full @ w - f_full
70          return 2 * A_full.T @ residual
71
72      def hessian(w):
73          return 2 * A_full.T @ A_full
74
75      w0 = np.random.randn(len(X_basis)) * 0.01
76
77      start_time = time.time()
78
79      try:
```

```
        if method_name == 'BFGS':
            result = minimize(objective, w0, method='BFGS', jac=gradient,
                              options={'maxiter': 1000, 'disp': False})
        elif method_name == 'L-BFGS-B':
            result = minimize(objective, w0, method='L-BFGS-B', jac=gradient,
                              options={'maxiter': 1000, 'disp': False})
        elif method_name == 'Newton-CG':
            result = minimize(objective, w0, method='Newton-CG', jac=gradient,
                hess=hessian,
                              options={'maxiter': 1000, 'disp': False})
        else:
            result = minimize(objective, w0, method='BFGS', jac=gradient)

        training_time = time.time() - start_time

        if result.success:
            y_solution = X_basis.T @ result.x
            return y_solution, training_time, result.fun, True
        else:
            return None, training_time, float('inf'), False

    except Exception as e:
        training_time = time.time() - start_time
        return None, training_time, float('inf'), False
```

Listing A.3: Código en Python para HerNN, LaNN y LeNN con métodos de entrenamiento de segundo orden

A.4 | ELM

A continuación se presentan las tres redes construidas para usar el entrenamiento de ELM:

A.4.1 | HerNN

```
def solve_elm_hermite(n_neurons, t_vals, h, f_rhs):
    window_hermite = 60

    X_basis = np.array([hermite_basis(i, t_vals) for i in range(n_neurons + 2)])
    X_deriv2 = np.array([hermite_deriv(i, t_vals, 2) for i in range(n_neurons + 2)])
    X_frac = np.array([fractional_derivative(X_basis[i], h, alpha, window_hermite)
                       for i in range(n_neurons + 2)])

    A_matrix = (A * X_deriv2 + B * X_frac + C * X_basis).T
    IC_matrix = np.array([[hermite_basis(i, 0) for i in range(n_neurons + 2)],
                          [hermite_deriv(i, 0, 1) for i in range(n_neurons + 2)]])

    A_full = np.vstack([A_matrix, IC_matrix])
    f_full = np.concatenate([f_rhs, [0.0, 0.0]])

    w = np.linalg.lstsq(A_full, f_full, rcond=None)[0]
    x_elm_hermite = w @ X_basis

    return x_elm_hermite
```

Listing A.4: Código en Python para HerNN

A.4.2 | LaNN

```python
def solve_elm_laguerre(n_neurons, t_vals, h, f_rhs):
    window_laguerre = 40

    X_basis = np.array([laguerre_basis(i, t_vals) for i in range(n_neurons + 2)])
    X_deriv2 = np.array([laguerre_deriv(i, t_vals, 2) for i in range(n_neurons + 2)])
    X_frac = np.array([fractional_derivative(X_basis[i], h, alpha, window_laguerre)
                       for i in range(n_neurons + 2)])

    A_matrix = (A * X_deriv2 + B * X_frac + C * X_basis).T
    IC_matrix = np.array([[laguerre_basis(i, 0) for i in range(n_neurons + 2)],
                          [laguerre_deriv(i, 0, 1) for i in range(n_neurons + 2)]])

    A_full = np.vstack([A_matrix, IC_matrix])
    f_full = np.concatenate([f_rhs, [0.0, 0.0]])

    w = np.linalg.lstsq(A_full, f_full, rcond=None)[0]
    x_elm_laguerre = w @ X_basis

    return x_elm_laguerre
```

Listing A.5: Código en Python para HerNN

A.4.3 | LeNN

```python
def solve_elm_legendre(n_neurons, t_vals, h, f_rhs):
    window_legendre = 100

    X_basis = np.array([legendre_basis(i, t_vals) for i in range(n_neurons + 2)])
    X_deriv2 = np.array([legendre_deriv(i, t_vals, 2) for i in range(n_neurons + 2)])
    X_frac = np.array([fractional_derivative(X_basis[i], h, alpha, window_legendre)
                       for i in range(n_neurons + 2)])

    A_matrix = (A * X_deriv2 + B * X_frac + C * X_basis).T
    IC_matrix = np.array([[legendre_basis(i, 0) for i in range(n_neurons + 2)],
                          [legendre_deriv(i, 0, 1) for i in range(n_neurons + 2)]])

    A_full = np.vstack([A_matrix, IC_matrix])
    f_full = np.concatenate([f_rhs, [0.0, 0.0]])

    w = np.linalg.lstsq(A_full, f_full, rcond=None)[0]
    x_elm_legendre = w @ X_basis

    return x_elm_legendre
```

Listing A.6: Código en Python para HerNN