



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Trabajo Terminal I

“Desarrollo de una aplicación web para el cálculo y graficación de series de Fourier”

2025-A075

QUE PARA OBTENER EL GRADO DE:

Ingeniero en Sistemas Computacionales

PRESENTA:

Sebastián Morales Palacios

DIRECTORES:

**Lic. Jesús Alfredo Martínez Nuño
M. en C. Gisela González Albarrán**

Ciudad de México
Noviembre 2024

Resumen

“Desarrollo de una aplicación web para el cálculo y graficación de series de Fourier”

Resumen - Dentro del presente trabajo terminal se lleva a cabo el desarrollo del prototipo de una aplicación web diseñada para calcular y graficar el desarrollo en series de Fourier de funciones periódicas. El desarrollo en serie puede ser tanto en forma trigonométrica como en forma exponencial compleja. Los módulos principales que contiene la aplicación son:

1. **Módulo para introducir la función:** ya sea definida en una sola función o a trozos, además de su periodo. Además, se tienen opciones para ingresar extensiones de funciones de medio rango pares, impares.
2. **Módulo para el cálculo de coeficientes:** donde se realizarán los cálculos correspondientes a los coeficientes mediante integración simbólica usando el sistema de álgebra computacional Maxima [9].
3. **Módulo de graficación de funciones periódicas y coeficientes de la serie:** donde se podrá ver la aproximación de la serie a la función mientras se incrementa el número de términos de la expansión.

La aplicación pretende ser de utilidad como herramienta de apoyo para estudiantes y docentes en el tema del análisis de Fourier.

Palabras Clave - aplicación web, análisis de Fourier, cálculo, matemáticas avanzadas para la ingeniería.

Correo de Contacto -
smoralesp1700@alumno.ipn.mx

Índice general

Resumen	III
1. Introducción	1
1.1. Motivación	1
1.2. Planteamiento del Problema	1
1.3. Justificación	2
1.4. Solución Propuesta	3
1.5. Objetivos	4
1.5.1. Objetivo General	4
1.5.2. Objetivos Específicos	4
1.6. Metodología de la Investigación	4
1.7. Estructura del documento técnico	5
2. Estado del Arte	6
2.1. Herramientas y Tecnologías Actuales	7
2.1.1. WolframAlpha	8
2.1.1.1. Prueba Wolfram Alpha	9
2.1.2. Symbolab	10
2.1.2.1. Prueba Symbolab	11
2.1.3. MATLAB	11
2.1.3.1. Prueba Matlab	12
2.1.4. Maple	13
2.1.4.1. Prueba Maple	14
2.1.5. Maxima	15
2.1.5.1. Prueba Maxima	16
2.1.6. Geogebra / Desmos	18
2.1.6.1. Prueba Geogebra	19
2.1.6.2. Prueba Desmos	20
2.1.7. Python	21
2.1.7.1. Sympy / Matplotlib	21
2.1.7.2. Prueba Python con Sympy y Matplotlib	21
2.1.7.3. Manim	22
2.1.7.4. Prueba Python con Manim	22
2.2. Comparativa del Funcionamiento de las Herramientas	24

3. Marco Teórico	26
3.1. Origen e Historia	26
3.1.1. El problema de la cuerda oscilante	26
3.1.1.1. D'Alambert y Euler	27
3.1.1.2. Bernoulli	28
3.1.2. La ecuación de calor	30
3.1.2.1. Fourier	30
3.2. Series de Fourier	33
3.2.1. Funciones Periódicas	33
3.2.2. Frecuencia	35
3.2.3. Armónicos	37
3.2.4. Funciones Pares e Impares	37
3.2.5. Funciones suaves y funciones suaves a trozos	39
3.2.6. Funciones Ortogonales	40
3.2.6.1. Ortogonalidad del seno y del coseno	41
3.2.7. Serie Trigonométrica de Fourier	41
3.2.7.1. Series de Fourier de una función con período T	41
3.2.7.2. Criterios para funciones pares e impares	43
3.2.8. Extensiones de Medio Intervalo	44
3.2.8.1. Extensión Par (Serie de Cosenos)	44
3.2.8.2. Extensión Impar (Serie de Senos)	45
3.2.8.3. Extensión Periódica	45
3.2.9. Serie Compleja de Fourier	46
3.2.9.1. Serie Compleja de Fourier de una función con período T	47
3.2.9.2. Serie Compleja de Fourier de 0 a T	48
3.2.10. Fenómeno de Gibbs	48
3.3. Aplicaciones Web	49
3.3.1. Componentes de una aplicación web	51
3.3.1.1. HTML	51
3.3.1.2. Canvas	51
3.3.1.3. CSS	51
3.3.1.4. JavaScript	51
3.3.2. NodeJS	52
3.3.3. Maxima	53
3.3.4. Angular	53
3.3.4.1. TypeScript	53
3.3.4.2. Tailwind CSS	54
3.3.5. API	54
3.3.5.1. Protocolo HTTP	54
3.3.5.2. REST	55
3.3.6. Interfaz gráfica de usuario	55
3.3.6.1. Principios de Usabilidad	55
3.3.7. Seguridad en Aplicaciones Web	57
3.3.7.1. Protocolo HTTPS	57
3.3.7.2. Certificado SSL/TLS	57

4. Análisis y Diseño del Sistema	58
4.1. Metodología de Desarrollo	58
4.1.1. ¿Por qué prototipos para este trabajo?	59
4.2. Análisis de Requerimientos	59
4.2.1. Requerimientos Funcionales	60
4.2.2. Requerimientos No Funcionales	64
4.3. Análisis de Riesgos	66
4.3.1. Identificación de Riesgos	66
4.3.2. Jerarquización de Riesgos	66
4.4. Estimación de Costos	66
4.4.1. Método de Estimación de Costos	66
4.4.2. Detalle de los Costos Estimados	66
4.5. Diseño del Sistema	66
4.5.1. Arquitectura General del Sistema	66
4.5.1.1. Arquitectura Cliente-Servidor	66
4.5.1.2. Flujo de Datos	66
4.5.2. Diseño de la Interfaz de Usuario	66
4.5.2.1. Estructura de la Interfaz	66
4.5.2.2. Prototipo de la Interfaz	66
5. Implementación (Primer Prototipo)	67
A. Cálculos	68
A.1. Forma Trigonométrica	68
A.2. Forma Exponencial Compleja	69
B. Códigos	71
B.1. Código en Matlab para graficar la serie de Fourier trigonométrica	71
B.2. Código en Matlab para graficar la serie de Fourier compleja . . .	72
B.3. Código en Maple para la serie de Fourier trigonométrica	73
B.4. Código en Maple para la serie de Fourier Compleja	75
B.5. Código en Maxima para la serie de Fourier trigonométrica	76
B.6. Código en Maxima para la serie de Fourier compleja	78
B.7. Código en Python usando matplotlib y sympy para la serie de Fourier trigonométrica	79
B.8. Código en Python usando matplotlib y sympy para la serie de Fourier compleja	81
Bibliografía	93

Índice de figuras

2.1. Gráfica de los coeficientes de Fourier calculados en el Apendice A <i>Fuente: Elaboración propia</i>	7
2.2. Logotipo de WolframAlpha	8
2.3. Calculo de una serie trigonométrica de Fourier en WolframAlpha	9
2.4. Calculo de una serie trigonométrica de Fourier en WolframAlpha	10
2.5. Logotipo de Symbolab	10
2.6. Calculo de una serie trigonométrica de Fourier en Symbolab	11
2.7. Logotipo de Matlab	12
2.8. Graficación de una serie trigonométrica de Fourier en Matlab	12
2.9. Graficación de una serie compleja de Fourier en Matlab	13
2.10. Logotipo de Maple	13
2.11. Cálculos de coeficientes trigonométricos de Fourier en Maple	14
2.12. Grafica de la serie de Fourier trigonométrica en Maple	14
2.13. Cálculo del coeficiente complejo de Fourier en Maple	15
2.14. Grafica de la serie de Fourier compleja en Maple	15
2.15. Logotipo de Maxima	16
2.16. Cálculos de coeficientes trigonométricos de Fourier en Maxima	17
2.17. Grafica de la serie de Fourier trigonométrica en Maxima	17
2.18. Cálculos de coeficientes compleja de Fourier en Maxima	18
2.19. Grafica de la serie de Fourier compleja en Maxima	18
2.20. Logotipo de Geogebra	19
2.21. Logotipo de Desmos	19
2.22. Gráfica de la serie trigonométrica de Fourier en Geogebra	20
2.23. Gráfica de la serie trigonométrica de Fourier en Geogebra	20
2.24. Logo de Python	21
2.25. Gráfica de la serie trigonométrica y/o exponencial de Fourier en Python con Sympy y Matplotlib	22
2.26. Gráfica de la serie trigonométrica de Fourier en Python con Manim	23
2.27. Gráfica de la serie compleja de Fourier en Python con Manim	23
3.1. Ilustración del problema de la cuerda vibrante	28
3.2. Ilustración de 3 armónicos y sus nodos de la cuerda al vibrar	29
3.3. Ilustración del problema de transferencia de calor en una barra de longitud ℓ	31
3.4. Gráfica de una función periódica $f(t)$ con periodo T	34
3.5. Gráfica del área bajo la curva de $f(t)$ entre los intervalos desde a hasta $a + T$ y desde b hasta $b + T$	35

3.6. Gráfica de las funciones trigonométricas $\sin(t)$ y $\cos(t)$	35
3.7. Función $\sin(t)$ con diferentes frecuencias	36
3.8. Relación de la frecuencia angular entre la función $\sin(t)$ con el círculo unitario.	36
3.9. Una función coseno y la misma función coseno desplazada hacia la izquierda por un ángulo φ	37
3.10. Función $f(t) = t^2$, una función par.	38
3.11. Función $f(t) = t^3$, una función impar	38
3.12. Ejemplo de los teoremas f y g de funciones pares e impares	39
3.13. Ejemplo de una función suave	39
3.14. Ejemplo de una función suave a trozos completamente continua y una función a trozos con discontinuidades (saltos).	40
3.15. Gráfica de una función periódica $f(t)$ con periodo T , repitiéndose en intervalos de $T, 2T$, y $3T$	42
3.16. Extensión par en cosenos de una función $f(t)$	44
3.17. Extensión impar en senos de una función $f(t)$	45
3.18. Extensión periódica de medio rango de una función $f(t)$	46
3.19. Aproximación de una función $f(t) = t$ con $-\pi < xt < \pi$ en sus 50 primeros armónicos	49
3.20. Diagrama de una arquitectura cliente - servidor	50
3.21. Cómo procesa node.js las peticiones entrantes utilizando el bucle de eventos	52
3.22. Arquitectura general de una API.	54
4.1. Etapas de la metodología por prototipos.	58

Índice de tablas

2.1. Comparación de software para cálculos matemáticos y visualización de datos	25
4.1. Ventajas/Desventajas de la metodología de Prototipos Evolutivos	59
4.2. Requerimiento funcional No. 1	60
4.3. Requerimiento funcional No. 2	61
4.4. Requerimiento funcional No. 3	61
4.5. Requerimiento funcional No. 4	62
4.6. Requerimiento funcional No. 5	62
4.7. Requerimiento funcional No. 6	63
4.8. Requerimiento funcional No. 7	63
4.9. Requerimiento funcional No. 8	64
4.10. Tabla de requerimientos no funcionales	65

Índice de códigos

1.	Código en Matlab para graficar la serie de Fourier trigonométrica de A.1.	72
2.	Código en Matlab para graficar la serie de Fourier compleja de A.2.	73
3.	Código en Maple para calcular y graficar la serie de Fourier trigonométrica de A.1.	74
4.	Código en Maple para calcular y graficar la serie de Fourier compleja de A.2.	76
5.	Código en Maxima para calcular y graficar la serie de Fourier trigonométrica de A.1.	77
6.	Código en Maxima para calcular y graficar la serie de Fourier compleja de A.2.	79
7.	Código en Python con matplotlib y sympy para graficar la serie de Fourier trigonométrica de A.1.	80
8.	Código en Python con matplotlib y sympy para calcular y graficar la serie de Fourier compleja de A.2.	82
9.	Código en Python con Manim para graficar la serie de Fourier trigonométrica de A.1.	87
10.	Código en Python con Manim para graficar la serie de Fourier compleja de A.1.	92

CAPÍTULO 1

Introducción

En este capítulo, examinaremos el problema que se abordará, la necesidad e impulso para hacerlo, así como los objetivos previstos para este proyecto y nuestras propuestas de solución para esta problemática.

1.1. Motivación

El área de estudio de las matemáticas es pilar en la formación de un ingeniero, proporcionando las herramientas esenciales para modelar y resolver problemas complejos en diversas áreas de la ingeniería. A través de mi formación como ingeniero en la Escuela Superior de Cómputo, al enfrentar el estudio del análisis de Fourier, especialmente al resolver ejercicios sobre series de Fourier, encontré que había una falta de medios óptimos para verificar los resultados de manera directa y eficiente. A diferencia de otros cálculos matemáticos, donde las calculadoras especializadas permiten comprobaciones rápidas y fiables, las series de Fourier requerían un proceso mucho más tedioso. Para asegurar la corrección de los cálculos, era necesario utilizar software de resolución matemática para obtener los coeficientes y, posteriormente, otro software de graficación para visualizar si los resultados correspondían efectivamente a la función dada. Este desafío se ampliaba aún más cuando se presentaban variaciones mínimas en los ejercicios, como cambiar el intervalo de la función o alternar entre series trigonométricas y complejas, o al aplicar extensiones pares o impares. Cada una de estas variaciones obligaba a repetir todos los pasos desde el inicio, lo que no solo consumía tiempo valioso, sino que también complicaba la gestión del tiempo disponible, especialmente cuando se tiene una carga académica intensa. Este tiempo podría utilizarse mejor en comprender los conceptos subyacentes y explorar en profundidad el por qué y el cómo de los fenómenos analizados mediante estas series.

1.2. Planteamiento del Problema

Las series de Fourier, desde su concepción por Jean-Baptiste Joseph Fourier a principios del siglo XIX [1], han desempeñado un rol crucial en el análisis y la comprensión de señales y fenómenos periódicos. Las series de Fourier son parte esencial en campos como la ingeniería eléctrica, la física teórica y el procesamiento

de señales, imágenes y audio, permitiendo descomponer funciones periódicas en sumas de senos y cosenos, lo que facilita su análisis y manipulación. Estas herramientas matemáticas no solo han impulsado avances significativos en la ciencia y tecnología, transformando nuestra interacción con el mundo, sino que también son cruciales en la enseñanza de los fundamentos teóricos de la ingeniería. Esta capacidad de simplificar señales complejas en componentes básicos no solo mejora el análisis, sino que también facilita la síntesis de nuevas tecnologías que se adaptan a necesidades y entornos cambiantes.

Este amplio espectro de aplicaciones destaca la importancia crítica de las series de Fourier no solo en la investigación avanzada, sino también en la formación académica de futuros ingenieros y científicos. Sin embargo, a pesar de su prevalencia en el currículo educativo, la implementación práctica de este análisis en entornos de aprendizaje a menudo revela áreas de oportunidad dentro de las herramientas disponibles, lo que impacta directamente en la eficacia con la que los estudiantes pueden aplicar y profundizar su comprensión de estos conceptos esenciales. La necesidad de una herramienta más eficiente, como se mencionó en la motivación, es crucial para superar estos desafíos y mejorar la experiencia educativa y profesional en el análisis de series de Fourier.

1.3. Justificación

A pesar de la indiscutible importancia del análisis de Fourier en diversas áreas de la ciencia y la ingeniería, el acceso a herramientas que integren de manera eficiente el cálculo y la visualización de las series de Fourier aún presenta importantes oportunidades de mejora. Las herramientas actuales, aunque avanzadas y robustas, suelen fragmentar el proceso, obligando a los usuarios a utilizar distintos programas o plataformas para realizar los cálculos y la representación gráfica de los resultados. Esta división entre herramientas de cálculo y visualización añade una capa de complejidad y tiempo, especialmente para aquellos usuarios que, además de realizar cálculos precisos, requieren visualizar los resultados de manera rápida y efectiva.

En este contexto, surge la propuesta de desarrollar un prototipo de aplicación web que combine ambas funcionalidades en una sola interfaz, facilitando el proceso de resolver y graficar las series de Fourier de manera práctica, intuitiva y accesible desde cualquier dispositivo con acceso a Internet. La creación de una plataforma que integre estos aspectos tiene el potencial de mejorar significativamente la experiencia del usuario al permitirle resolver ecuaciones y obtener representaciones gráficas de las series de Fourier en un solo entorno, eliminando la necesidad de conocimientos técnicos avanzados o el uso de múltiples herramientas.

El desarrollo de esta aplicación tiene el potencial de ser un aporte valioso para diversas comunidades, como estudiantes, ingenieros y profesionales del área de las ciencias exactas, que requieren manipular y visualizar series de Fourier en sus trabajos. La integración de funcionalidades de cálculo simbólico con visualización gráfica interactiva permitirá un mayor grado de experimentación

y aprendizaje, reduciendo el tiempo y esfuerzo necesarios para comprender y analizar las transformaciones de Fourier.

Además, este proyecto representa una oportunidad para integrar y aplicar los conocimientos y habilidades adquiridos a lo largo de mi formación en ingeniería en sistemas. Al abordar tanto aspectos de programación, como de matemáticas avanzadas y principios de ingeniería de software, este trabajo terminal no solo me permite abordar una necesidad práctica, sino que también sirve como un medio para demostrar la capacidad de aplicar teoría y técnicas de ingeniería en un contexto real. De esta manera, no solo se busca desarrollar una herramienta útil para otros, sino también consolidar y exhibir competencias en áreas clave de mi carrera.

1.4. Solución Propuesta

La solución que se plantea en este proyecto es el desarrollo de una aplicación web que permita el cálculo y graficación de series de Fourier de manera integrada, eficiente y accesible, cubriendo la brecha existente en las herramientas actuales, que tienden a separar el cálculo matemático de la visualización gráfica en una sola interfaz. La solución propuesta unificará el cálculo de los resultados con la graficación interactiva de las mismas en una sola plataforma, diseñada para ser simple de usar, práctica y accesible desde cualquier navegador web, para usuarios que deseen analizar y visualizar series de Fourier, eliminando la necesidad de usar múltiples programas para resolver y graficar las series de Fourier.

La aplicación web contará con las siguientes características clave:

- **Cálculo automático de series de Fourier para diferentes tipos de funciones:**
 - La aplicación calculará **series de Fourier para funciones continuas** en un intervalo o para **funciones definidas a trozos**. El usuario podrá ingresar funciones matemáticas, incluso aquellas con discontinuidades o que estén definidas por partes en distintos intervalos, y la aplicación manejará estos casos automáticamente.
 - Se implementará tanto la **serie trigonométrica** como la **serie exponencial compleja**. La serie trigonométrica descompondrá la función en términos de senos y cosenos, mientras que la versión compleja lo hará en términos de exponentiales imaginarios, lo cual es útil en el contexto de análisis más avanzado y en aplicaciones de ingeniería.
- **Extensiones de medio rango:**
 - La herramienta permitirá calcular **extensiones de medio rango** para funciones en un medio intervalo. Esto incluirá la posibilidad de obtener:
 - **Serie de senos** para funciones impares.
 - **Serie de cosenos** para funciones pares.

- Serie completa para funciones periódicas en un intervalo definido, permitiendo extender la función para construir series de Fourier en un medio rango específico.
- Visualización gráfica interactiva:
 - Una vez calculados los coeficientes de Fourier, la aplicación generará una gráfica interactiva que mostrará la aproximación de la serie de Fourier a la función original. El usuario podrá ajustar el número de términos de la serie para observar cómo mejora la aproximación conforme se incluyen más términos en la suma.
- Interfaz intuitiva y amigable:
 - La aplicación ofrecerá una interfaz sencilla y accesible, en la que el usuario podrá ingresar las funciones, definir los intervalos y elegir el tipo de serie de Fourier que desea calcular. La experiencia estará diseñada para minimizar la curva de aprendizaje, permitiendo a usuarios sin experiencia en programación obtener resultados rápidamente.

1.5. Objetivos

1.5.1. Objetivo General

Desarrollar un prototipo de una aplicación web capaz de calcular las series de Fourier en su forma trigonométrica o exponencial compleja de una función continua en un intervalo o definida a trozos, así como ser capaz de obtener la extensión par o impar de dicha función y, finalmente, graficar tanto la función original como la función expandida como una serie de Fourier y poder añadir o eliminar términos de dicha forma para apreciar como esta se aproxima a la función original.

1.5.2. Objetivos Específicos

- Implementar una interfaz de usuario que permita ingresar la función continua o a trozos, seleccionar intervalos y definir el tipo de serie de Fourier (trigonométrica o exponencial) o tipo de expansión (par o impar).
- Implementar los módulos para el cálculo de coeficientes de la serie trigonométricas y serie exponencial compleja, adaptándose a funciones continuas o definidas a trozos.
- Desarrollar un módulo de visualización que permita graficar tanto la función original como la aproximación de la serie de Fourier. Este módulo debería incluir opciones para añadir o eliminar términos y visualizar cómo estas modificaciones afectan la aproximación a la función original

1.6. Metodología de la Investigación

Para el desarrollo de este proyecto se utilizará la metodología por prototipos...

1.7. Estructura del documento técnico

Al final voy a detallar como es que está estructurado todo el documento del TT

CAPÍTULO 2

Estado del Arte

Para la correcta comprensión del trabajo presente, se necesita conocer el estado actual de las herramientas y tecnologías disponibles para el cálculo y la visualización de series de Fourier, así como los estudios y proyectos previos que abordan la implementación de soluciones similares. En este sentido, se revisarán diversas plataformas de uso común que permiten realizar estos procesos de manera separada, además de calcular y/o graficar la serie de Fourier para la función $f(x) = x$ en el intervalo de $-\pi$ a π utilizando cada una de las herramientas previamente descritas. El cálculo se realizará en su forma trigonométrica y, en los casos en que la herramienta lo permita, también se obtendrá la forma exponencial compleja. Asimismo, se graficará la serie de Fourier en las plataformas que lo permitan, lo que nos permitirá comparar tanto el proceso como los resultados obtenidos en cada herramienta.

Esta comparación servirá para identificar las capacidades, ventajas y limitaciones de cada una de las plataformas en el contexto del cálculo y la visualización de series de Fourier, evaluando también su facilidad de uso y precisión en la representación gráfica.

- **Forma Trigonométrica:**

- Coeficientes:

$$a_0 = 0, \quad a_n = 0, \quad b_n = \frac{2(-1)^{n+1}}{n}$$

- Serie trigonométrica de Fourier para $f(x) = x$:

$$f(x) = 2 \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} \sin(nx)$$

- **Forma Exponencial Compleja:**

- Coeficiente complejo:

$$c_n = \frac{i(-1)^n}{n}, \quad \text{para } n \neq 0.$$

- Serie exponencial compleja de Fourier para $f(x) = x$:

$$f(x) = \sum_{\substack{n=-\infty \\ n \neq 0}}^{\infty} \frac{i(-1)^n}{n} e^{inx}$$

(Los calculos para llegar a estos coeficientes se encuentran desarrollados en el Apendice A .)

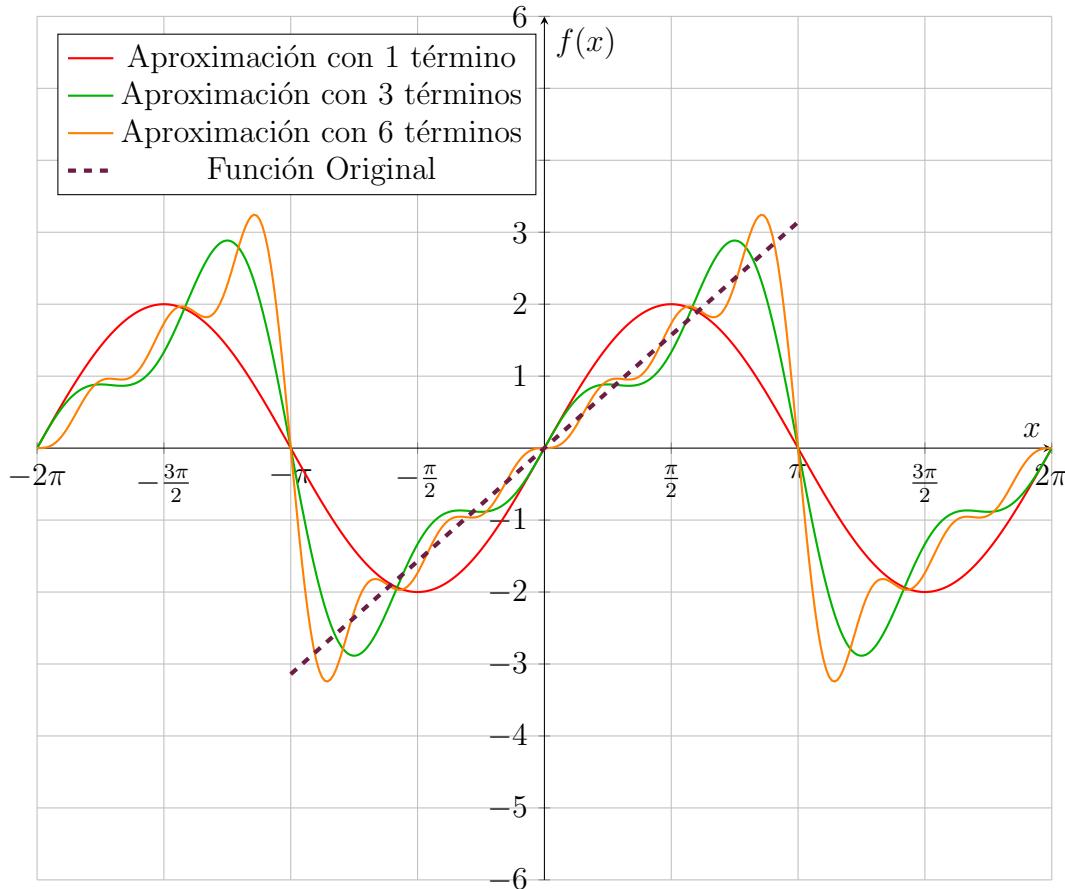


Figura 2.1: Gráfica de los coeficientes de Fourier calculados en el Apendice A
Fuente: Elaboración propia

Podemos ver en la 2.1, la gráfica de la función y sus aproximaciones de Fourier, donde estas aproximaciones son equivalentes entre la serie trigonométrica y la serie exponencial. Esta revisión permitirá contextualizar la propuesta de una aplicación web que integre ambas funcionalidades en una sola plataforma.

2.1. Herramientas y Tecnologías Actuales

Las series de Fourier al formar parte importante en diversas áreas de aprendizaje como la ingeniería, la física y las matemáticas aplicadas, ha impulsado el desarrollo de numerosas soluciones tecnológicas para su cálculo y visualización. En consecuencia, hoy en día disponemos de una gran variedad de plataformas y herramientas que facilitan estos procesos, permitiendo resolver y graficar series de Fourier de forma parcial. En esta sección, se presenta un análisis de las herramientas más significativas para esto, así como sus características, funcionalidades y limitaciones en cuanto a la incorporación de ambas capacidades en una única plataforma.

2.1.1. WolframAlpha

Wolfram Alpha es una aplicación comercial que proporciona respuestas tanto a preguntas como cálculos basados en el motor de Mathematica, un potente software para cálculos simbólicos y numéricos [2].



Figura 2.2: Logotipo de WolframAlpha. *Fuente:* [2]

En contraste con otros motores de búsqueda, en vez de ofrecer una lista de sitios web o documentos, proporciona respuestas precisas y exhaustivas basándose en los conceptos introducidos en su motor de búsqueda. Entre algunas de sus capacidades, este potente software tiene funciones propias para resolver y graficar series de Fourier [3]:

- `FourierSeries[exp, t, n]`
- `FourierTrigSeries[exp, t, n]`
- `FourierSinSeries[exp, t, n]`
- `FourierCosSeries[exp, t, n]`

Estas funciones nos permiten obtener expansión de la serie de Fourier, ya sea compleja, trigonométrica, en senos o cosenos de una función `exp` con periodo de 2π donde esta puede ser de un solo trozo o definida a varios trozos, para esto se usaría la función `Piecewise[val1,cond1,val2,cond2,...]` [4], en términos de la variable `t` hasta el `n`-ésimo término. Además nos mostrará una gráfica estática de una aproximación con `n` términos. También tiene funciones para calcular los coeficientes de la serie [2]:

- `FourierSeriesCoefficient[exp, t, n]`
- `FourierSinCoefficient[exp, t, n]`
- `FourierCosCoefficient[exp, t, n]`

Estas funciones nos permiten calcular el `n`-ésimo coeficiente de la serie de Fourier exponencial compleja, de senos o de cosenos, de una función `exp` con periodo de 2π , recordando que puede ser a trozos o de un solo trozo.

Podemos observar que las funciones propias de Wolfram Alpha para problemas que impliquen series de Fourier limitan el periodo en el que las funciones

matemáticas son definidas, limitándolo en 2π , para estos casos se podría resolver el coeficiente directamente usando la función de `Integrate[f, x, xmin, xmax]` [4] para calcular individualmente cada coeficiente, ya sea a_0, a_n y b_n para las series trigonométricas o c_0 y c_n para la serie compleja, pero así no nos proporciona la pequeña gráfica que si nos da al usar las funciones para expandir la serie, además de que la gráfica que se proporciona no es interactiva, y si queremos verla con más detalle debemos pagar su suscripción.

2.1.1.1. Prueba Wolfram Alpha

Para nuestra primer prueba, calcularemos la serie de Fourier trigonométrica de la función calculada en el Apendice A, para hacerlo usaremos la función `FourierTrigSeries(exp, t, n)` de WolframAlpha, que nos permite calcular la serie trigonométrica de Fourier de la función `exp` respecto a la variable `t` obteniendo la serie hasta el término `n`.

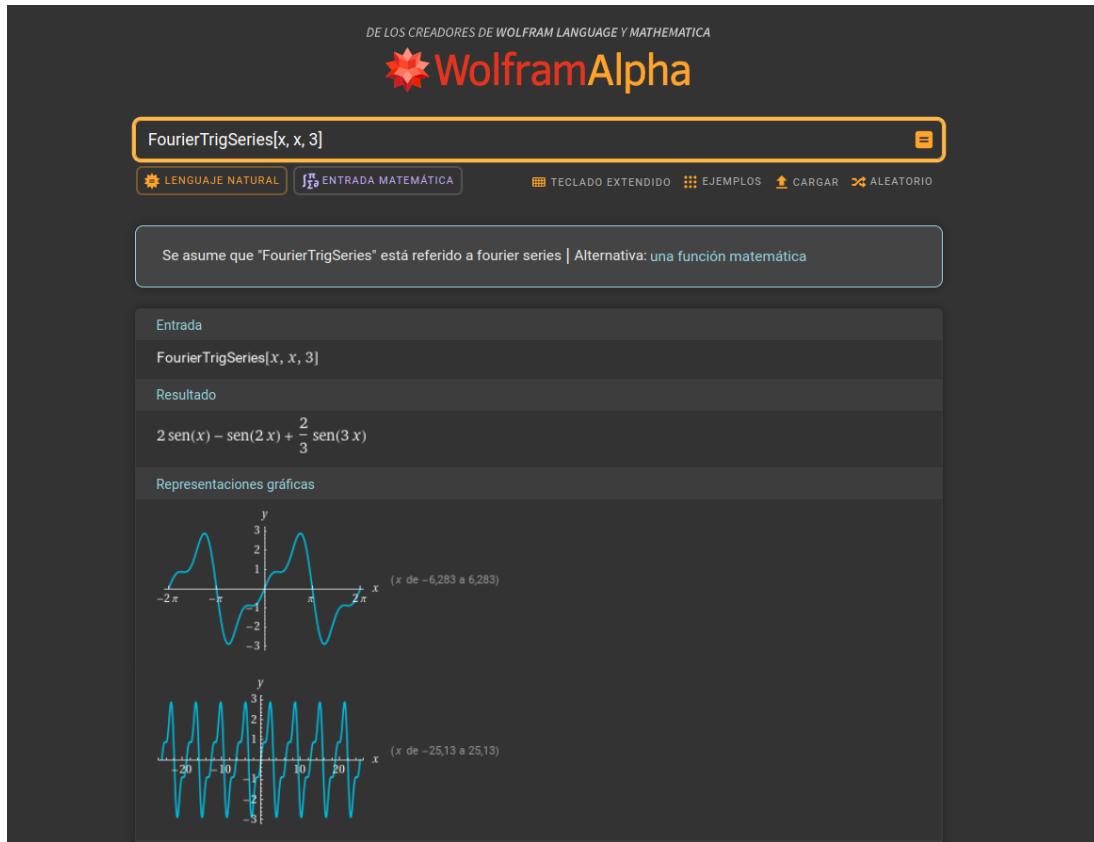


Figura 2.3: Calculo de una serie trigonométrica de Fourier en WolframAlpha

Ahora usaremos la función de `FourierSeries(exp, t, n)` de WolframAlpha, esta función no permite calcular la serie exponencial compleja de Fourier de la función `exp` respecto a la variable `t` obteniendo la serie hasta el término `n`.

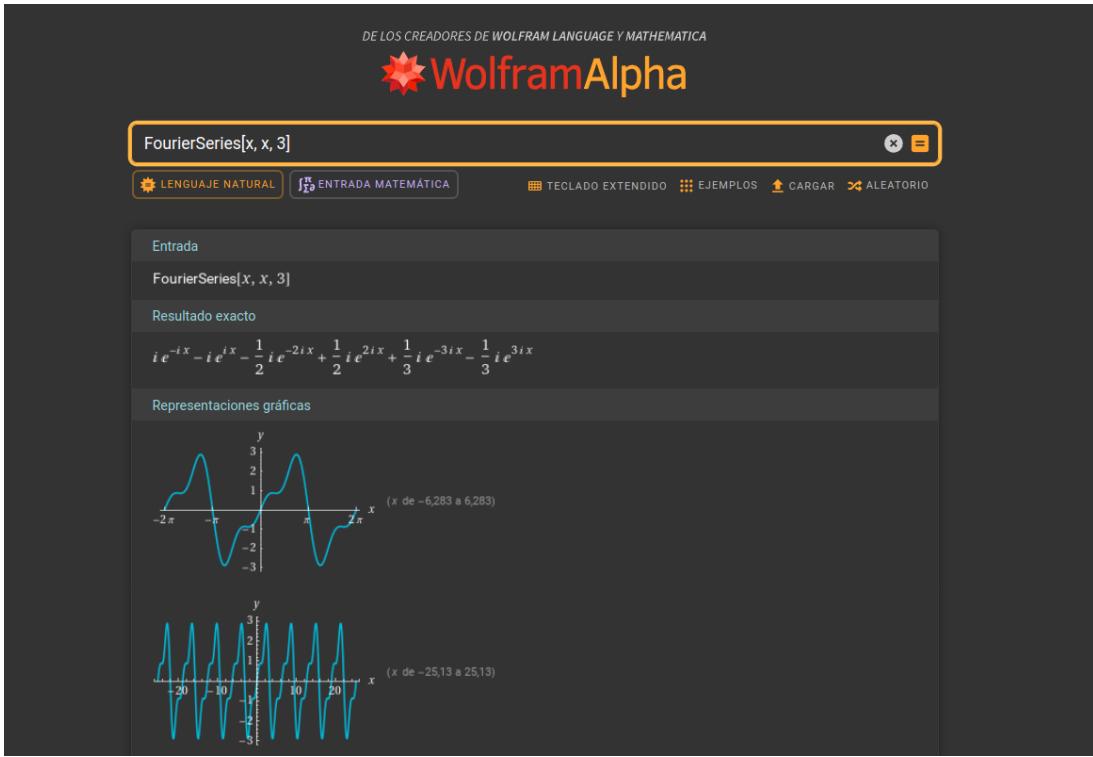


Figura 2.4: Calculo de una serie trigonométrica de Fourier en WolframAlpha

Wolfram nos devuelve la expansión de cada serie en su forma mas reducida, ademas de dos pequeñas gráficas de nuestra aproximación vista desde no nos devuelve los coeficientes de la serie ni tampoco su expresión final en notación de serie.

2.1.2. Symbolab

Symbolab es una calculadora digital que ayuda a resolver problemas matemáticos, como ecuaciones, álgebra o cálculo. Se trata de una herramienta que cuyo principal propósito es ayudar a sus usuarios a aprender matemáticas, ya que ofrece resoluciones paso a paso y conocimientos impulsados por inteligencia artificial. [5].



Figura 2.5: Logotipo de Symbolab. Fuente: [5]

Symbolab se centra en la enseñanza de procedimientos matemáticos mediante explicaciones detalladas. Su interfaz es especialmente útil para estudiantes, ya que

presenta guías paso a paso para temas como derivadas, integrales y ecuaciones algebraicas complejas, lo cual es ideal para el aprendizaje autodidacta. Este cuenta con una función para resolver series de Fourier en su forma trigonométrica denominada **fourier series (f)**, $[-L, L]$ [5] en donde f es una función definida en el intervalo de $[-L, L]$. A pesar de contar con funciones para definir funciones matemáticas a trozos y funciones para hacer gráficas, no es posible unificarlas en la misma aplicación, ademas de no contar con funciones para otras series de Fourier como extensiones de medio rango o exponencial compleja, pero, al igual que en Wolfram Alpha, se pueden calcular individualmente los coeficientes con la función **integral from a to b of x** [5].

2.1.2.1. Prueba Symbolab

Para nuestra siguiente prueba, calcularemos la serie de Fourier trigonométrica de la función calculada en el Apéndice A, para hacerlo usaremos la función **fourier series (f)**, $[-L, L]$ de Symbolab, que nos permite calcular la serie trigonométrica de Fourier de la función f respecto a la variable obteniendo desde el intervalo $-L$ a L .

The screenshot shows the Symbolab interface with the search bar containing "fourier series x, [-pi, pi]". The left sidebar has categories like Pre-algebra, Algebra, Precalculus, Calculus, Functions, Matrices and vectors, Trigonometry, Statistics, Chemistry, and Conversions. The main area shows the input "Serie de Fourier x, [-pi, pi]" and the resulting formula: $\sum_{n=1}^{\infty} \frac{2(-1)^n \sin(nx)}{n}$. Below the formula, there's a "Pasos de solución" (Solution Steps) section with several mathematical steps and formulas. On the right, there's a note: "¡Guardar en el cuaderno!" (Save to notebook) and "Iniciar sesión" (Log in).

Figura 2.6: Calculo de una serie trigonométrica de Fourier en Symbolab

Symbolab nos dará la expresión en forma de serie, y solo podremos hacerlo para la serie trigonométrica, ademas de que no nos muestra ninguna gráfica.

2.1.3. MATLAB

MATLAB es un entorno de programación y un lenguaje de alto nivel comercial ampliamente utilizado en ingeniería, física y matemáticas aplicadas para el análisis numérico, la visualización de datos y el desarrollo de algoritmos. Su robusta caja de herramientas, especialmente la *Signal Processing Toolbox* y la *Symbolic Math Toolbox*, facilitan el cálculo y la graficación de series de Fourier [6].



Figura 2.7: Logotipo de Matlab. Fuente: [6]

A pesar de que MATLAB cuenta con funciones especiales para el análisis de Fourier como la transformada de Fourier (`fourier(f)`) o la Transformada rápida de Fourier (`fft(f)`) no cuenta con funciones específicas para series de Fourier, sin embargo, podemos definir nuestros coeficientes para hacer el gráfico de ambas series.

2.1.3.1. Prueba Matlab

Para la prueba con Matlab, primero ejecutaremos el código en Apendice B, que nos graficará la serie de Fourier a partir de sus coeficientes trigonométricos 2.8.

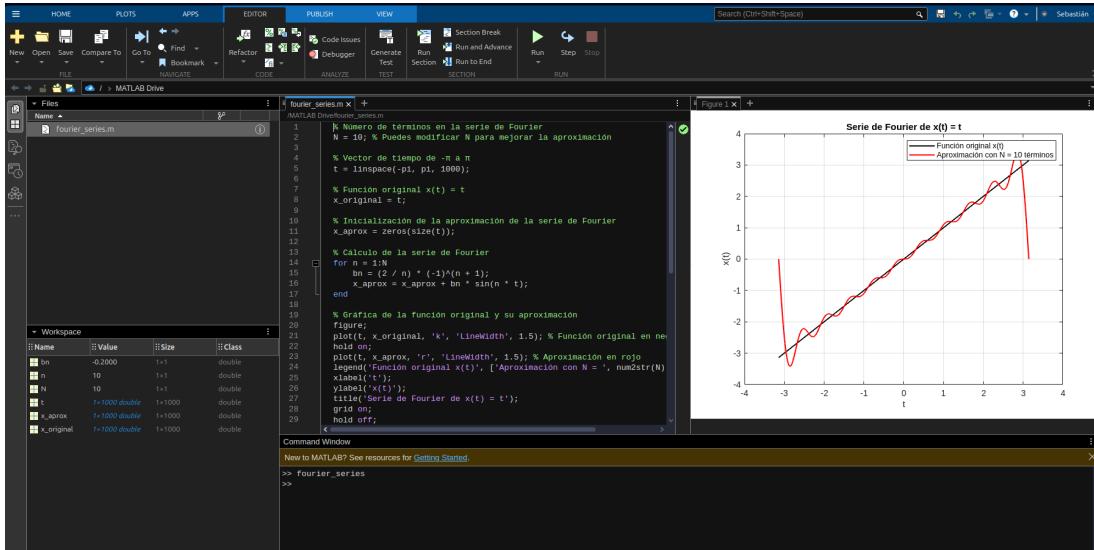


Figura 2.8: Graficación de una serie trigonométrica de Fourier en Matlab

Ahora usamos el código en Apendice B para graficar la misma serie pero ahora usando su coeficiente complejo 2.9.

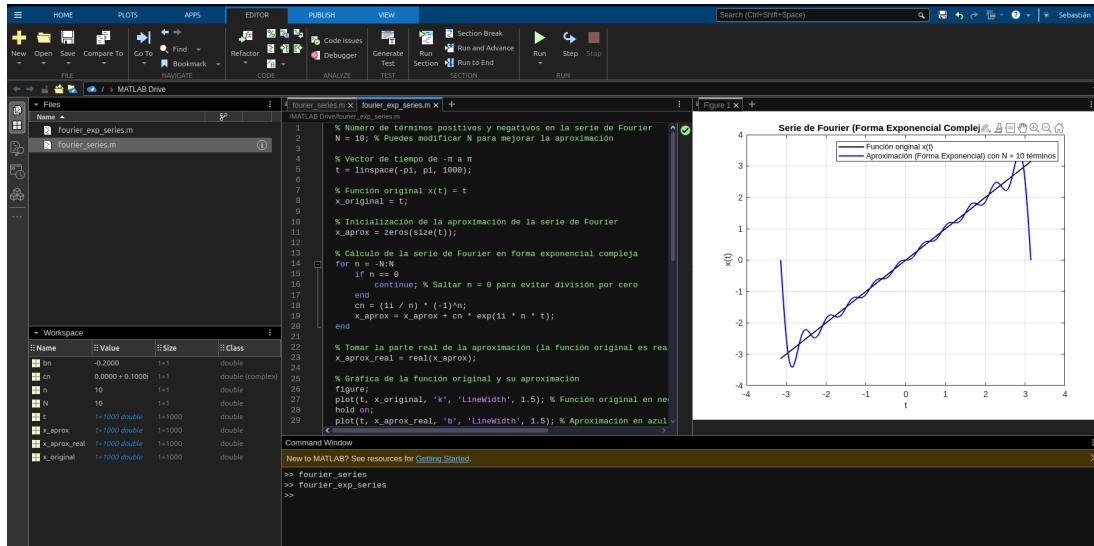


Figura 2.9: Graficación de una serie compleja de Fourier en Matlab

Matlab es capaz de graficar muy buenas aproximaciones de ambas series en una gráfica que nos brinda cierto nivel de interactividad pero todo dependerá de nuestros cálculos y de que tanto detalle plasmemos en nuestro código para aumentar las funcionalidades

2.1.4. Maple

Maple, desarrollado por Maplesoft, es un Sistema de Álgebra Computacional comercial que permite realizar cálculos matemáticos simbólicos y numéricos, resolver ecuaciones, analizar datos y crear visualizaciones gráficas en 2D y 3D. Maple utiliza el paradigma REPL (Read-Eval-Print Loop), un entorno interactivo donde cada línea de código ingresada es leída, evaluada y su resultado es impreso inmediatamente, facilitando así la experimentación y el desarrollo incremental de cálculos complejos. [7].



Figura 2.10: Logotipo de Maple. Fuente: [7]

Si bien este CAS cuenta con funciones dedicadas al cálculo de series de Fourier, éstas vienen definidas en paquetes desarrollados por la comunidad de Maplesoft [8], así que resulta más eficiente y conveniente usar las funciones de integración (`int(f, x = a..b)`) calcula simbólicamente la integral de `f`, que depende de la variable `x` desde el punto `a` hasta el `b`) para calcular los coeficientes, y para expandir series (`seq(m, n = a..b)`) en donde `m` es la función a sumar, sobre la variable `n` desde el punto `a` hasta el `b`) las funciones de expansión, además de poder declarar una función a trozos con la función `piecewise(cond-1, f-1, cond-2, f-2, ..., cond-n, f-n, f-otherwise)`.

2.1.4.1. Prueba Maple

Para la prueba con Maple, primero ejecutaremos el código en Apendice B, que, primeramente, calculará los coeficientes de la serie trigonométrica 2.11.

```

C:\Users\USER\Desktop\TTcto\Maple\Trig_con_grafica.mw - [Server 3] - Maple 2024
Archivo (F) Editar (E) Ver (V) Insertar (I) Formato (R) Evaluar (A) Herramientas (T) Ventana (W) Ayuda (H)
Iniciar sesión
Al Formula Assistant
Paletas Libre de trabajo
▶ Favoritos
▶ Expresión
▶ Cálculo
▶ Símbolos Comunes
▼ Variables
Variable Valor
oo
Matriz
Unidades
Trazo
Griego
Componentes
Término | Función | Serie Función | Simplificada | x := -1/2..1/2];
fune := x
T := 2 π
series_cosec_cose := cos(n-x)
series_zine_cose := sin(n-x)
a0 := 0
an := 0
bn := -2(-1)^n/n
a0_simp := 0
an_simp := 0
bn_simp := -2(-1)^n/n
Coeff_A0 := 0
Coeff_An := 0
Coeff_Bn := -2(-1)^n/n
n1 := 1
n2 := 5
lista_An := [0, 0, 0, 0]
lista_Bn := [2 sin(x), -sin(2 x), 2 sin(3 x), -sin(4 x), 2 sin(5 x)]
lista_completa := [0, 0, 0, 0], [2 sin(x), -sin(2 x), 2 sin(3 x), -sin(4 x), 2 sin(5 x)]
serie_final := [0, 0, 0, 0]
serie_factor := [0, 0, 0, 0]
serie_funcion := 2 sin(x) - sin(2 x) + 2 sin(3 x) - sin(4 x) + 2 sin(5 x)
serie_funcion_simplificada := 2(48 cos(x)^4 - 30 cos(x)^3 - 16 cos(x)^2 + 13) sin(x)/15
Atajos para el Editor de Matemáticas
Evaluar: Enter
Evaluar en Linea: Alt+Enter
Toggle Math/Text: F5
Cambiar a Matemática Ejecutable: Shift+F5
Fracción de Salida / Superíndice: →
Asignaciones: a:=b
Funciones: f:=x->x^2
Ecucciones: x=y
Comando Completo: Esc
Navegar Marcadores de Posición: Tab

```

Figura 2.11: Cálculos de coeficientes trigonométricos de Fourier en Maple

Posteriormente, este código nos dará una gráfica estática con los valores que le establecimos Apendice B 2.12.

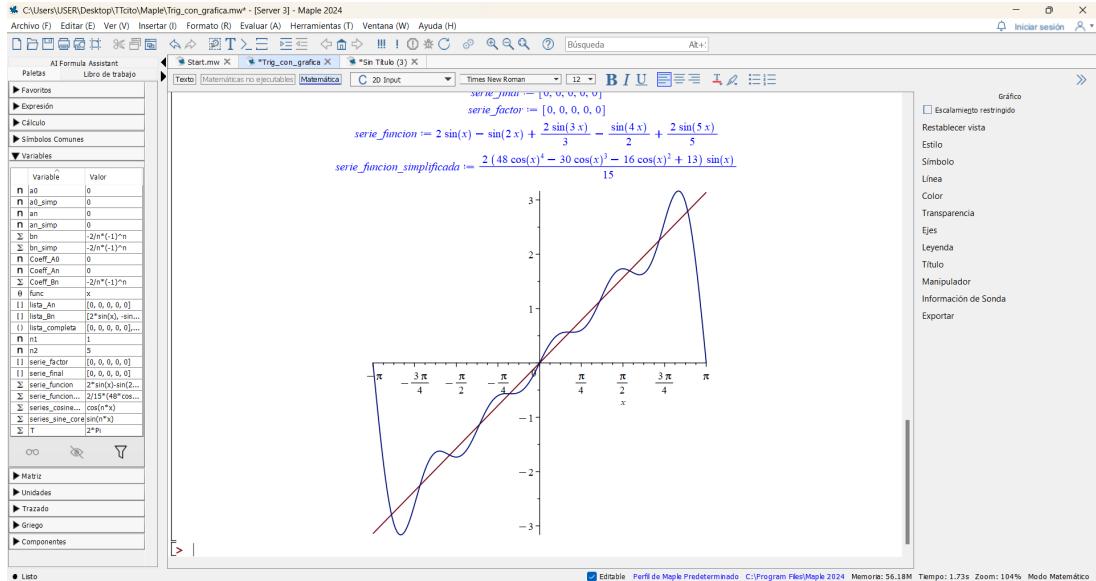


Figura 2.12: Grafica de la serie de Fourier trigonométrica en Maple

Podemos ver que puede resolver la serie trigonométrica sin problema alguno, ahora veamos la misma función pero desarrollada en una serie exponencial compleja Apendice B.

2.1. HERRAMIENTAS Y TECNOLOGÍAS ACTUALES

15

```

# Graficamos la función y la serie aproximada
plot([func,serie_funcion_simplificada],x = -T/2..T/2);

func := x
T := 6
series_core := e^(1/3)*x
cn := 31((-1)^(n+1))/pi*n
c0 := 0
c0_simp := 0
cn_simp := 31((-1)^n)/pi*n
Coeff_0 := 0
Coeff_n := 31((-1)^n)/pi*n
n := 1
n1 := 1
n2 := 5
lista_completa := [
-31e^(1/3*x)/pi, 31e^(1/3*x)/2, -1e^(1/3*x)/pi, 31e^(41/3*x)/4, -31e^(51/3*x)/5, 31e^(-1/3*x)/2, -31e^(-11/3*x)/4, 31e^(-21/3*x)/5
]
serie_funcion := -31e^(1/3*x)/pi + 31e^(21/3*x)/2 - 1e^(1/3*x)/pi + 31e^(41/3*x)/4 - 31e^(51/3*x)/5
serie_funcion_simplificada := -1/840( -2520e^(1/3*x) + 1260e^(21/3*x) - 840e^(31/3*x) + 630e^(41/3*x) - 504e^(51/3*x) + 420e^(11/3*x) - 360e^(21/3*x) + 315e^(31/3*x) - 280e^(41/3*x) + 252e^(51/3*x))

Warning, unable to evaluate 1 of the 2 functions to numeric values in the region; complex values were detected

```

Figura 2.13: Cálculo del coeficiente complejo de Fourier en Maple

De igual modo, este código nos dará una gráfica estática con los valores que le establecimos Apéndice B 2.12.

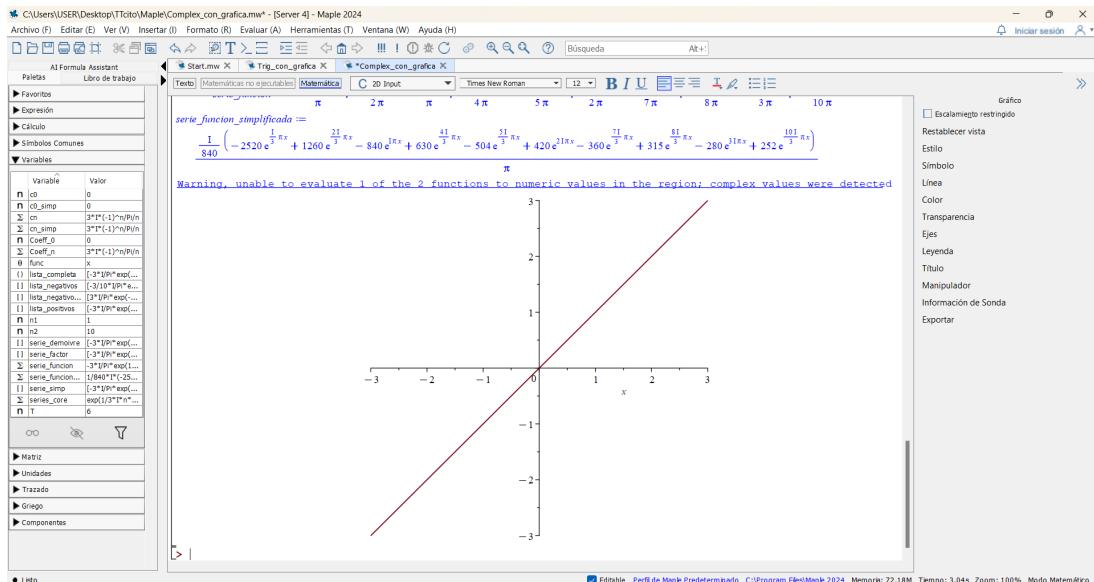


Figura 2.14: Grafica de la serie de Fourier compleja en Maple

El detalle está en que no hay forma de graficar nuestra función serie compleja ya que no puede hacer las cancelaciones entre los números imaginarios al expandir la serie, por lo tanto, la función la mantiene en el dominio de los números complejos y no la puede graficar en el plano real. Una alternativa sería graficar la serie trigonométrica que obtuvimos en 2.12 ya que son equivalentes.

2.1.5. Maxima

Maxima es un Sistema de Álgebra Computacional de código abierto y gratuito, derivado del sistema Macsyma desarrollado en la década de 1980. Permite

realizar cálculos simbólicos y numéricos avanzados, como derivadas, integrales, simplificación de expresiones algebraicas, resolución de ecuaciones y generación de gráficos en 2D y 3D. Maxima emplea el paradigma REPL (Read-Eval-Print Loop), proporcionando un entorno interactivo donde cada comando ingresado se procesa de inmediato, lo que facilita la verificación y corrección rápida de cálculos [9].



Figura 2.15: Logotipo de Maxima. *Fuente:* [9]

Si bien, este CAS al igual que Maple, cuenta con módulo que importa funciones especiales para el análisis de Fourier, este si es propio de Maxima. Sin embargo, igual que en Maple, resulta más práctico trabajar con las funciones propias de Maxima para integrar (`integrate(expr, x, a, b)`) Calcula simbólicamente la integral de `expr` con límites en `a` y `b`) así como su función para expandir una función en serie (`makelist(expr, i, i_min, i_max, step)`) Genera una lista evaluando `expr` para cada valor de `i` desde `i_min` hasta `i_max`, incrementando `i` en `step` en cada iteración).

2.1.5.1. Prueba Maxima

Para la prueba con Maxima, ejecutaremos el código en Apendice B, que primero calculará los coeficientes de la serie trigonométrica 2.11.

2.1. HERRAMIENTAS Y TECNOLOGÍAS ACTUALES

17

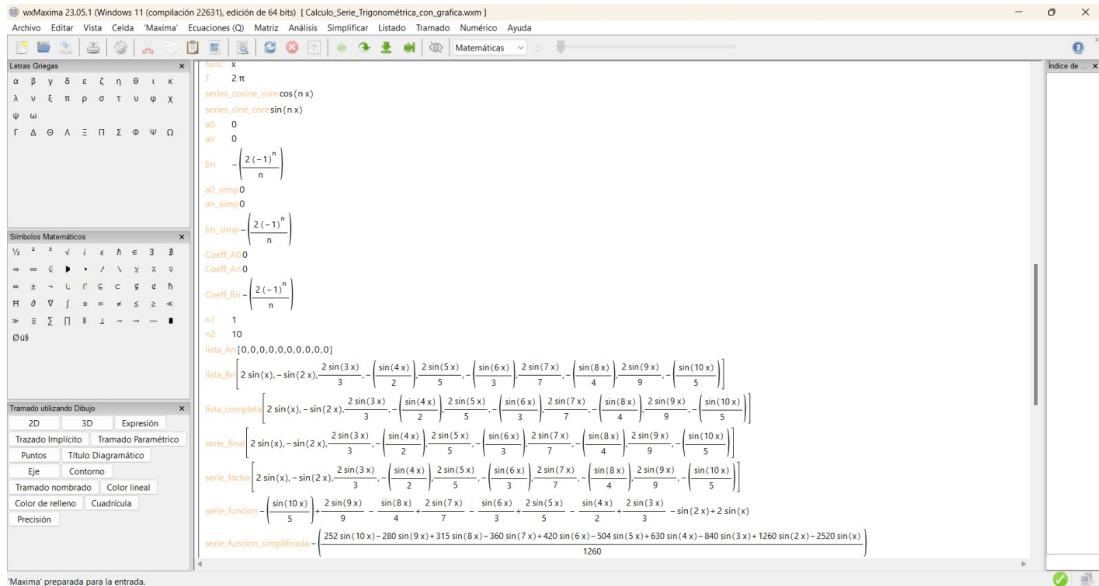


Figura 2.16: Cálculos de coeficientes trigonométricos de Fourier en Maxima

Posteriormente, este código nos dará una gráfica dinámica con los valores que le establecimos.

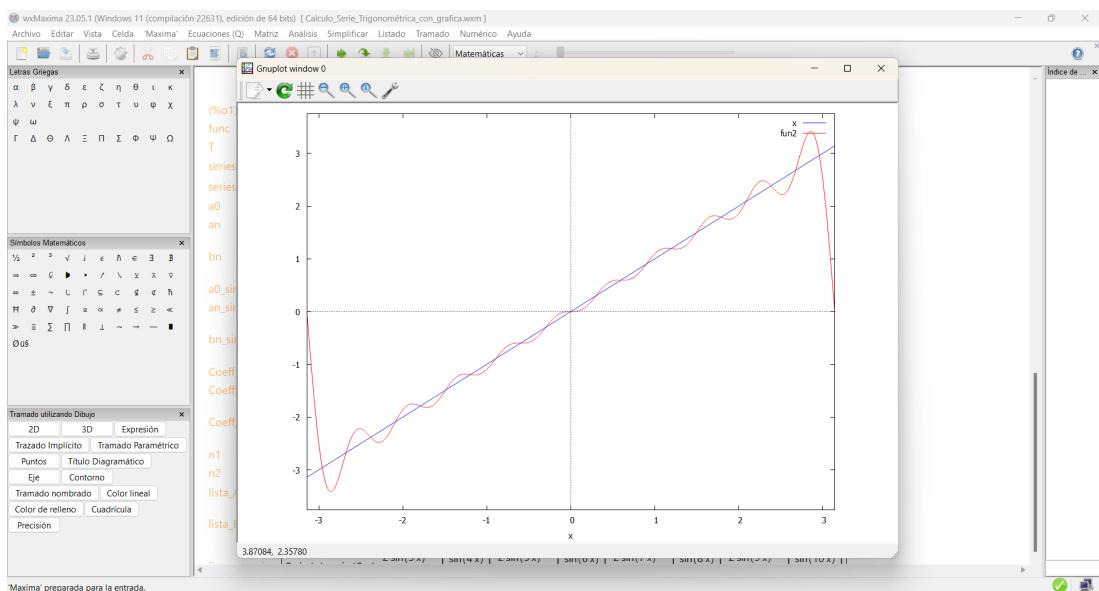


Figura 2.17: Grafica de la serie de Fourier trigonométrica en Maxima

Podemos ver que sin problema alguno se construye la gráfica, ahora, probemos haciendo el problema de la misma función pero para la serie compleja Apéndice B.

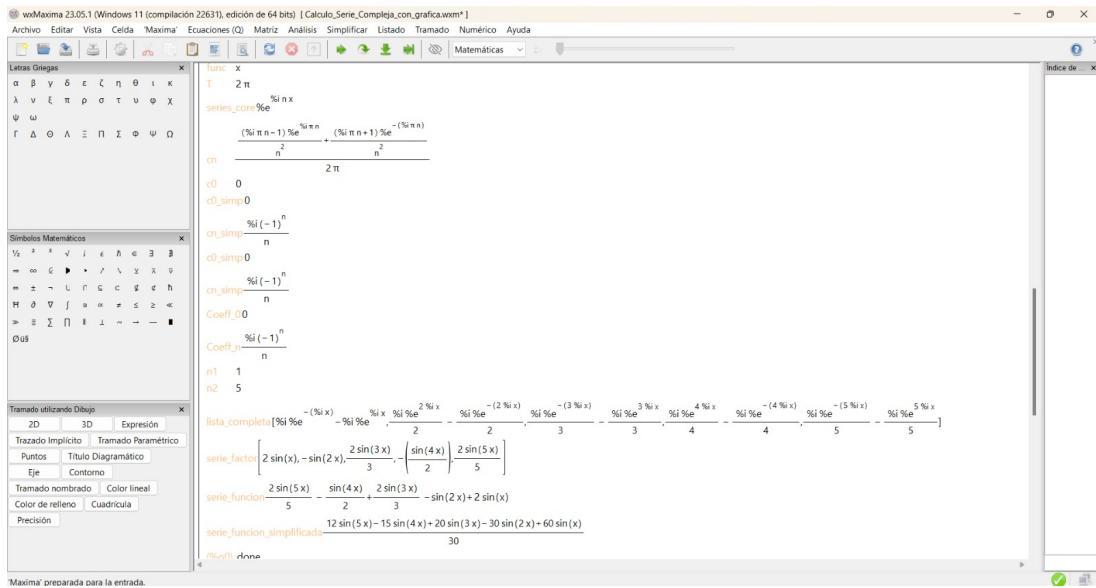


Figura 2.18: Cálculos de coeficientes compleja de Fourier en Maxima

De igual modo, este código nos dará una gráfica dinámica con los valores que le establecimos.

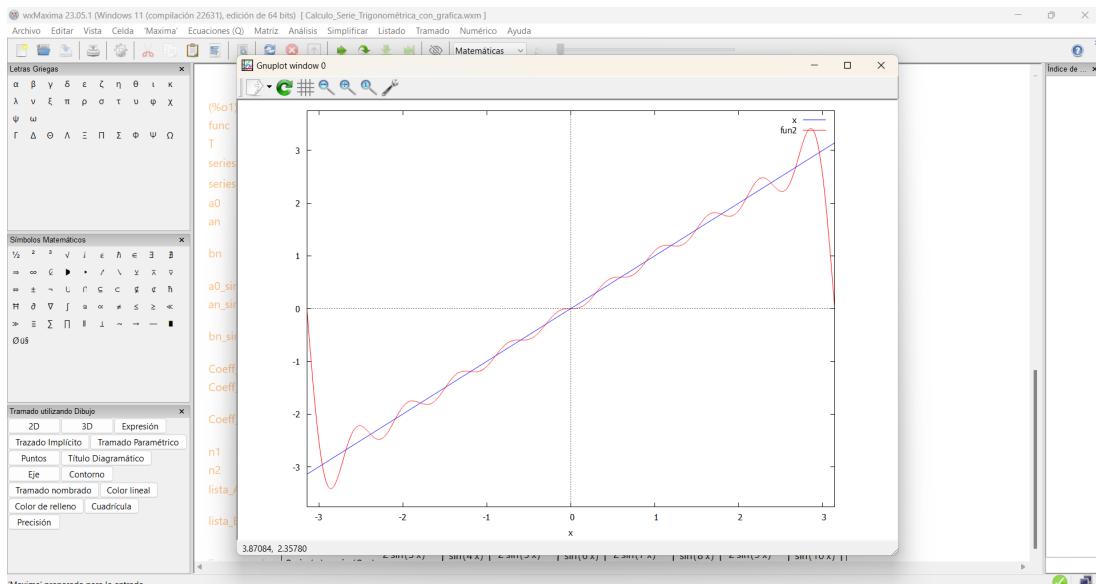


Figura 2.19: Grafica de la serie de Fourier compleja en Maxima

Podemos observar que sin problemas podemos hacer el gráfico ya que la función `demoivre()` se encarga de separar la parte real e imaginaria y con esto, hacer la gráfica equivalente. A pesar de que Maxima no cuenta con una función para declarar funciones a trozos, podemos calcular las integrales por separado o usar matrices (`matrix (fila_1, ..., fila_n)`), pero esto se verá más adelante.

2.1.6. Geogebra / Desmos

GeoGebra y Desmos son dos herramientas educativas digitales, gratuitas y de código abierto, utilizadas para la enseñanza y el aprendizaje de las matemáticas.

Ambas permiten a los usuarios graficar funciones, trabajar conceptos algebraicos y geométricos de manera interactiva, facilitando la visualización y comprensión de diversos temas matemáticos. Estas presentan algunas diferencias: GeoGebra ofrece una suite más completa que incluye módulos para álgebra, geometría, cálculo y estadística, lo que la hace especialmente útil para crear construcciones geométricas dinámicas y realizar análisis más complejos [10]. Por otro lado, Desmos se destaca por su interfaz intuitiva y facilidad de uso, enfocándose principalmente en la gráfica de funciones y proporcionando herramientas sencillas para crear visualizaciones rápidas y efectivas, lo que la hace ideal para estudiantes y educadores que buscan una plataforma accesible y potente para la enseñanza de conceptos fundamentales [11].

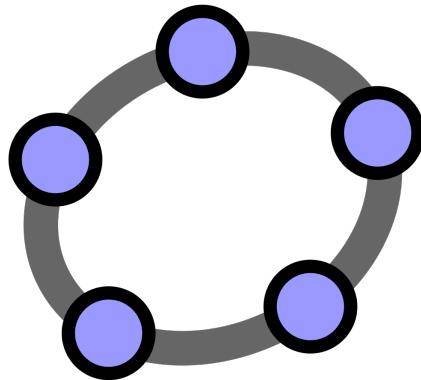


Figura 2.20: Logotipo de Geogebra. *Fuente:* [2]



Figura 2.21: Logotipo de Desmos. *Fuente:* [11]

Si bien, no tienen funciones específicas para el análisis de Fourier, ambas pueden hacer las gráficas de sumas correspondientes a los coeficientes. A continuación mostraremos la gráfica de la serie de Fourier de A.1 con ambas herramientas.

2.1.6.1. Prueba Geogebra

Para la prueba con Geogebra, crearemos un slider desde 1 hasta el valor que queramos con la única condición de que este de saltos de 1 en 1, es decir, sea un slider de números enteros, también usamos la función de `Suma(Expression, Variable, Valor Inicial, Valor Final)` en donde `Expresión` será la función de la serie, `Variable` es el n-ésimo término en la serie, `Valor Inicial` será 1 y `Valor Final` será el valor que toma el slider.

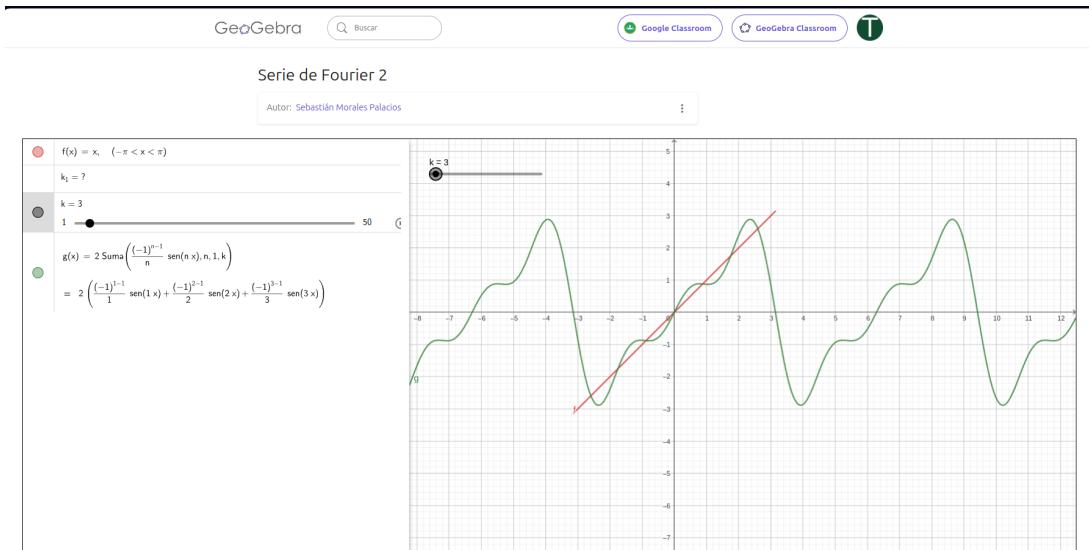


Figura 2.22: Gráfica de la serie trigonométrica de Fourier en Geogebra

Geogebra nos dará una gráfica completamente dinámica pero será un tanto pesada, ya que al añadir más de 5 términos la página comenzará a tener problemas con el dinamismo.

2.1.6.2. Prueba Desmos

Para Desmos será algo más sencillo que en Geogebra, ya que solo tendremos que computar la expresión de la serie sin definir funciones, al terminar de computar la serie nos generará en automático el slider, solo

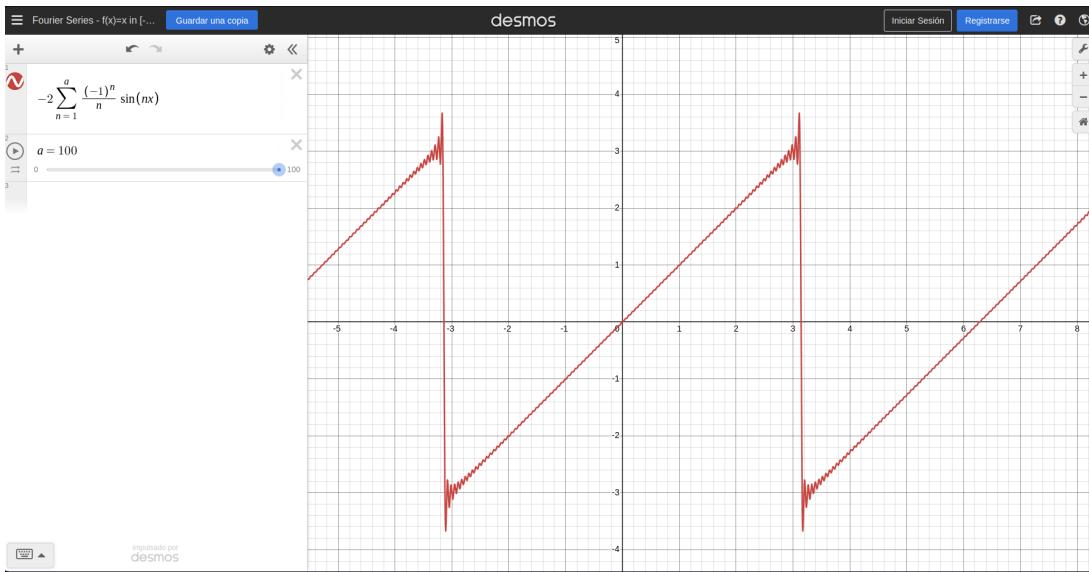


Figura 2.23: Gráfica de la serie trigonométrica de Fourier en Geogebra

A diferencia de geogebra, la gráfica tendrá un todo un tanto más simple pero será mucho más rápida, ya que podremos hacer una serie con 100 términos y no perderá fluidez como lo haría geogebra.

2.1.7. Python

Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). Los desarrolladores utilizan Python porque es eficiente y fácil de aprender, además de que se puede ejecutar en muchas plataformas diferentes. El software Python se puede descargar gratis, se integra bien a todos los tipos de sistemas y aumenta la velocidad del desarrollo [12].



Figura 2.24: Logo de Python

La versatilidad de python permiten que se creen librerías para todo tipo de tareas con él, una de ellas es la de hacer cálculos matemáticos y creación de gráficos, de los cuales, podemos tomar funciones específicas para ayudarnos en el calculo de series de Fourier.

2.1.7.1. Sympy / Matplotlib

SymPy y Matplotlib son dos bibliotecas de Python utilizadas en el dentro de las matemáticas y la visualización de datos. SymPy es una biblioteca de matemáticas simbólicas que permite realizar manipulaciones algebraicas, resolver ecuaciones, derivar e integrar funciones [13]. Por otro lado, Matplotlib es una biblioteca de visualización que permite crear gráficos estáticos, interactivos y animados [14]. Al combinar SymPy con Matplotlib, es posible pueden realizar cálculos simbólicos complejos y representar visualmente los resultados de manera clara y efectiva.

2.1.7.2. Prueba Python con Sympy y Matplotlib

Para este caso, utilizaremos las funciones de integración numérica de Sympy para calcular los coeficientes de la serie trigonométrica y usando las funciones de graficación de Matplotlib, crearemos un gráfico para visualizar la función original con su aproximación de Fourier, para esto usamos el código en Apéndice B.

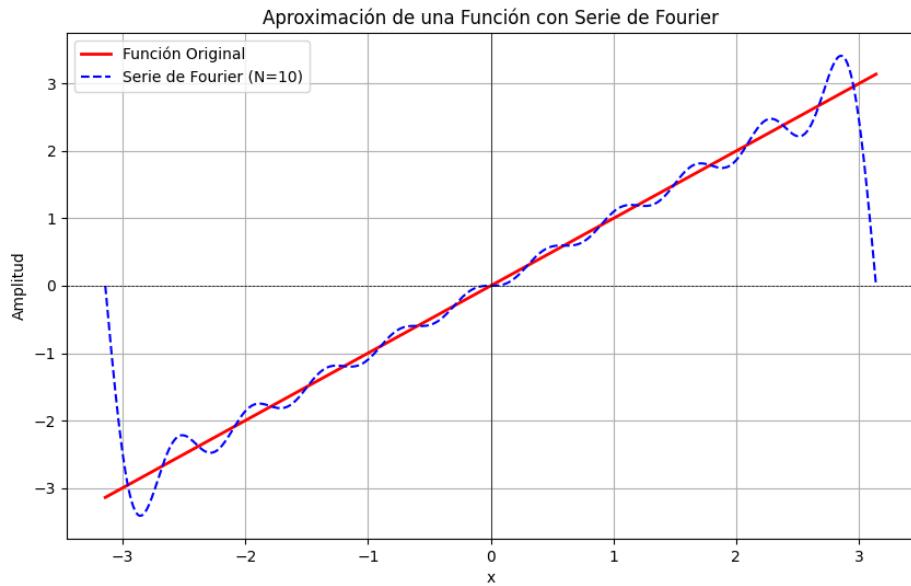


Figura 2.25: Gráfica de la serie trigonométrica y/o exponencial de Fourier en Python con Sympy y Matplotlib

Ahora, si hacemos unos cambios en el código para que calcule el coeficiente de la serie exponencial compleja , obtendremos exactamente la misma gráfica en 2.25, notando que Sympy es capaz de simplificar la parte imaginaria de la serie exponencial compleja para poder ver la parte real resultante en la gráfica.

2.1.7.3. Manim

Manim es una biblioteca de Python de código abierto diseñada para la creación de animaciones matemáticas de alta calidad. Desarrollada inicialmente por Grant Sanderson para su canal de YouTube ”3Blue1Brown”, Manim permite crear animaciones dinámicas y precisas de manera intuitiva y atractiva. La herramienta ofrece un control detallado sobre cada aspecto de la animación [15].

2.1.7.4. Prueba Python con Manim

Para poder usar a manim, debemos definir toda la expresión de la serie trigonométrica y usar las funciones que nos ofrece manim para hacer la animación como se detalla en el código en Apendice B.

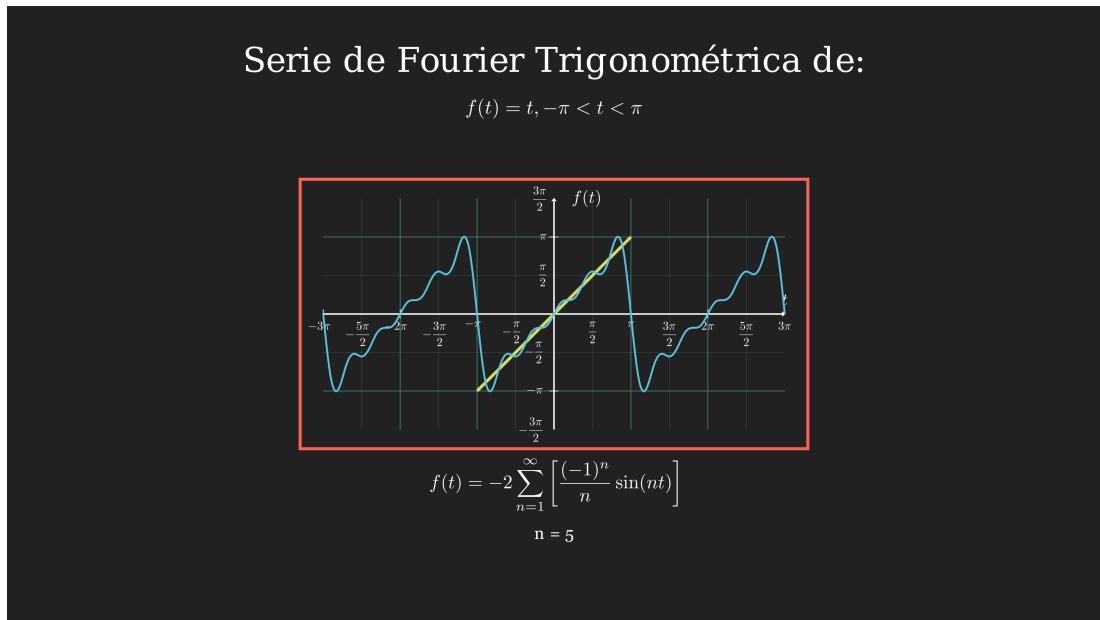


Figura 2.26: Gráfica de la serie trigonométrica de Fourier en Python con Manim

Ahora, ejecutamos el código en Apéndice B para la serie compleja y obtenemos la siguiente salida.

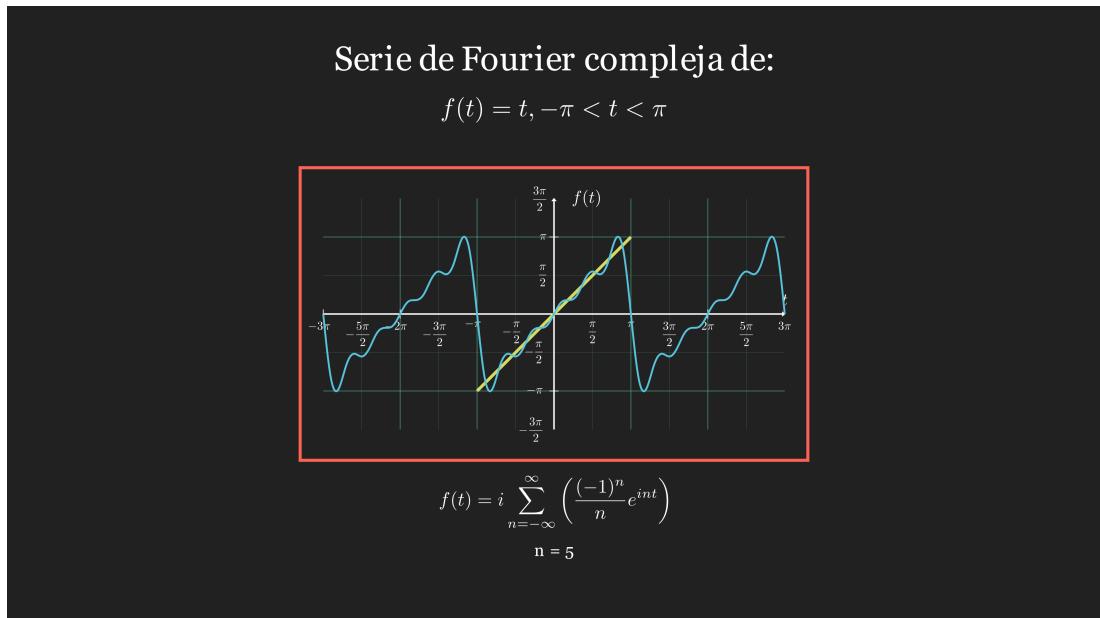


Figura 2.27: Gráfica de la serie compleja de Fourier en Python con Manim

Manim genera gráficas impecables con animaciones fluidas y limpias, sin embargo debemos recordar que los coeficientes deben ser calculados previamente ademas de que tardará bastante tiempo en renderizar el video o imagen dependiendo de la cantidad de cálculo que tenga que resolver o la calidad de la imagen.

2.2. Comparativa del Funcionamiento de las Herramientas

SOFTWARE	VENTAJAS	DESVENTAJAS	PRECIO
Wolfram Alpha	Funciones propias para series de Fourier; Integrando, puede obtener cualquier coeficiente en su forma más simplificada.	Las funciones de series de Fourier se limitan en su periodo; Gráfica no intuitiva	Desde MXN \$1,200.00 anuales para estudiantes.
Symbolab	Interfaz amigable; Función dedicada para la serie trigonométrica	Puede calcular coeficientes si integramos por aparte; No muestra gráfica	Planes básicos gratuitos; suscripción mensual para funciones avanzadas.
Matlab	Potente para realizar cálculos y visualización dinámica	Requiere conocimientos previos de programación en Matlab; licencia cara para profesionales.	Desde USD\$99 anuales para estudiantes.
Maple	Puede calcular cualquier coeficiente y graficar la serie trigonométrica	Interfaz limitada y menos intuitiva; Poco flexible al usarlo; Requiere conocimientos en REPL	Planes de estudiantes y profesionales, precio varía.
Maxima	Permite calcular cualquier coeficiente y expandirlo en serie además de mostrar una gráfica medianamente interactiva además de ser extremadamente liviano	Interfaz limitada y menos intuitiva; no tan popular, menos soporte comunitario; Requiere conocimientos en REPL	Gratis, de código abierto.

Continúa en la siguiente página

Tabla 2.1 – continuación

SOFTWAI	VENTAJAS	DESVENTAJAS	PRECIO
Geogebra	Gráficas extremadamente dinámicas e interactivas; Puede graficar cualquier serie trigonométrica.	Requiere realizar calcular los coeficientes; Limitado a la serie trigonométrica.; Pesado al hacer los cálculos para graficar	Gratis, de código abierto.
Desmos	Gráficas extremadamente dinámicas e interactivas; Puede graficar cualquier serie trigonométrica; Liviano.	Requiere cálculos previos de los coeficientes; Limitado a serie trigonométrica.	Gratis, de código abierto.
Matplotlib / Sympy	Permite calcular series exponenciales y complejas, además de mostrar una gráfica medianamente interactiva	Requiere conocimientos de Python y programación avanzada; menos intuitivo para principiantes.	Gratis, de código abierto.
Manim	Puede graficar cualquier serie de Fourier mientras conocemos sus coeficientes y expresión	Curva de aprendizaje alta; requiere conocimientos avanzados de Python y Manim; Requiere calcular la serie previamente.	Gratis, de código abierto.

Tabla 2.1: Comparación de software para cálculos matemáticos y visualización de datos

De esta tabla podemos observar que si bien, hay opciones que para cálculo simbólico sumamente potentes que nos permiten calcular los coeficientes, éstas se ven limitadas en la visualización de generar una gráfica dinámica e intuitiva, mientras que las que ofrecen una mejor visualización de las series en las gráficas no siempre pueden hacer los cálculos o algunas no tienen forma de graficar cuando la serie presenta números complejos, sin mencionar que la mayoría requiere de conocimientos en programación especializado en el lenguaje de la herramienta.

CAPÍTULO 3

Marco Teórico

En este capítulo se presentará la teoría necesaria para el desarrollo de nuestra calculadora de series de Fourier. Comenzaremos explorando la historia y evolución de las series de Fourier, se detallarán las fórmulas matemáticas para comprender la descomposición de funciones periódicas en sumas de senos y cosenos, resaltando los conceptos de coeficientes de Fourier. Además, se presentan las tecnologías necesarias para la elaboración del proyecto.

3.1. Origen e historia de las series de Fourier

Uno de los problemas del que se ocuparon los matemáticos del siglo XVIII es el que se conoce con el nombre del *problema de la cuerda vibrante*. Este problema fue estudiado por D'Alambert y Euler (usando el método de propagación de las ondas) y un poco más tarde, concretamente en 1753, por Daniel Bernoulli. La solución dada por este difería de la proporcionada por los anteriores y consistió básicamente en expresar la solución del problema como superposición (en general infinita) de ondas sencillas.

Las ideas de Bernoulli fueron aplicadas y perfeccionadas por Fourier, en 1807, en el estudio de problemas relacionados con la conducción del calor. Quedaron plasmadas por escrito en el libro clásico *Théorie analytique de la chaleur*, publicado en 1822. Los razonamientos realizados por Fourier en este libro plantearon de manera inmediata numerosas controversias y cuestiones que han tenido una influencia significativa en la historia de la Matemática [17].

3.1.1. El problema de la cuerda oscilante

Uno de los problemas más interesantes que abordaron los científicos del siglo XVIII, y que aparece con frecuencia en problemas físicos relacionados con procesos oscilatorios, es el conocido como *problema de la cuerda vibrante*. Este se puede describir en su forma más elemental de la siguiente manera: Supongamos que tenemos una cuerda flexible y tensa, cuyos extremos están fijos, convenientemente, en los puntos $(0, 0)$ y $(\ell, 0)$ sobre el eje horizontal. Si la cuerda se tira de modo que su forma inicial corresponde a la curva definida por $y = f(x)$, y luego se suelta, la pregunta es: ¿Cuál será el movimiento resultante de la cuerda? Los desplazamientos de la cuerda siempre se encuentran en un mismo plano, y el vector desplazamiento es perpendicular en cualquier momento. Para describir

este movimiento se utiliza una función $u(x, t)$, donde $u(x, t)$ representa el desplazamiento vertical de la cuerda en la posición x (con $0 \leq x \leq \ell$) y en el instante t (con $t \geq 0$). El problema es determinar $u(x, t)$ a partir de $f(x)$ [18].

3.1.1.1. D'Alambert y Euler

El primer matemático que propuso un modelo adecuado para este problema fue Jean Le Rond D'Alambert [19]. Bajo diversas hipótesis (asumiendo, por ejemplo, que las vibraciones son "pequeñas"), en 1747 D'Alambert dedujo la ecuación de onda en la siguiente forma:

$$\frac{\partial^2 u(x, t)}{\partial t^2} = \frac{\partial^2 u(x, t)}{\partial x^2}, \quad 0 < x < \ell, \quad t > 0 \quad (3.1)$$

en donde:

- $u(x, t)$: Es la función que describe el desplazamiento de la onda en función de la posición x y el tiempo t .
- $\frac{\partial^2 u(x, t)}{\partial t^2}$: Segunda derivada parcial de $u(x, t)$ respecto al tiempo t , que representa la aceleración del desplazamiento de la onda.
- $\frac{\partial^2 u(x, t)}{\partial x^2}$: Segunda derivada parcial de $u(x, t)$ respecto a la posición x , que describe la curvatura espacial de la onda.

y esta debe satisfacer las condiciones siguientes:

$$\begin{aligned} u(x, 0) &= f(x), \quad 0 \leq x \leq \ell \\ \frac{\partial u(x, 0)}{\partial t} &= 0, \quad 0 \leq x \leq \ell \\ u(0, t) &= u(\ell, t) = 0, \quad t \geq 0. \end{aligned} \quad (3.2)$$

En donde la primera ecuación establece la posición inicial de la cuerda, mientras que la segunda indica que la velocidad inicial de la cuerda es cero (recordando que, una vez desplazada a la posición $f(x)$, la cuerda es liberada). La última condición expresa que, para cualquier tiempo, los extremos de la cuerda permanecen fijos. En la figura 3.1 La variable $u = u(x, t)$ mide el desplazamiento sobre la vertical a tiempo $t > 0$ en la posición $x \in [0, \ell]$.

Y apartir de esto, construyó la solución [20]:

$$u(x, t) = F(x + t) + G(x - t) \quad (3.3)$$

Que, estando sujeta a las condiciones de frontera 3.2 se podía reducir a [20]:

$$u(x, t) = F(x + t) + F(x - t) \quad (3.4)$$

siempre y cuando la función F fuese periódica, impar y diferenciable en todas partes. D'Alambert demostraba con esto que la ecuación de onda admitía "un número infinito de soluciones".

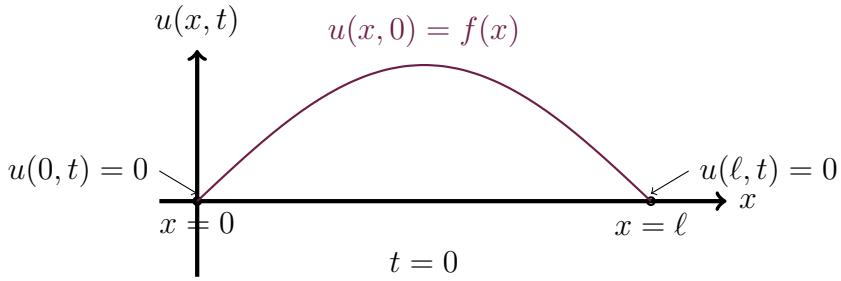


Figura 3.1: Ilustración del problema de la cuerda vibrante. *Fuente: Elaboración propia*

Euler entra en escena al criticar varias de las suposiciones que D'Alambert hiciera para encontrar su solución. Una de particular interés es la de suponer que la forma inicial de la curva o cuerda era continua y diferenciable “como una anguila”. Esta suposición para Euler, le quitaba generalidad a la solución de D' Alambert pues omitía funciones, notablemente la que representa una cuerda al ser pulsada, como se muestra en la figura 3.1. Euler derivó una ecuación de onda ligeramente más general que la de D'Alembert 3.1:

$$\frac{1}{c^2} \frac{\partial^2 u(x, t)}{\partial t^2} = \frac{\partial^2 u(x, t)}{\partial x^2} \quad (3.5)$$

con la solución [20]:

$$u(x, t) = F(x + ct) + G(x - ct) \quad (3.6)$$

que con las mismas condiciones de frontera resulta en [20]:

$$u(x, t) = F(x + ct) + F(x - ct) \quad (3.7)$$

Euler argumentaba, a diferencia de D'Alembert, que la función F quedaba determinada por la posición y velocidad inicial de la cuerda. Si denotamos por $w(x)$ y $v(x)$ a éstas, respectivamente, entonces la solución puede representarse como:

$$u(x, t) = \frac{1}{2} \left(w(x + ct) + w(x - ct) + \frac{1}{c} \int_{x-ct}^{x+ct} v(s) ds \right) \quad (3.8)$$

Las funciones w y v podían ser arbitrarias (en el sentido que podían representar cualquier curva que pudiera “dibujarse a mano”) e incluía, implícitamente, a la curva “triangular” de la figura. Como dicen los autores [21] a D'Alembert no le quitaba el sueño sacrificar realismo físico por pureza matemática; Euler, por el contrario, usaba la realidad física para contraargumentar la generalidad de las soluciones obtenidas por el francés aunque su interés no fuera tanto realismo físico sino la generalidad de las soluciones admitidas por la ecuación de onda.

3.1.1.2. Bernoulli

Otra manera de obtener la solución del problema 3.1, distinta a la de sus distinguidos interlocutores, fue propuesta por Daniel Bernoulli en 1753 [21] La

idea clave es obtener la solución de 3.1 como superposición de ondas más sencillas, concretamente aquellas que son de la forma [22]:

$$u(x) = \alpha \sin \frac{\pi x}{\ell} + \beta \sin \frac{2\pi x}{\ell} + \gamma \sin \frac{3\pi x}{\ell} + \delta \sin \frac{4\pi x}{\ell} + \dots \quad (3.9)$$

que incorporaban el hecho experimental de que cualquier modo de vibración de una cuerda puede obtenerse superponiendo modos vibratorios simples representados por cada término. Estas funciones representan, para $n = 1$, el modo fundamental, y para $n > 1$, sus armónicos. De esta manera, cualquier vibración de la cuerda puede describirse como una superposición de estos armónicos, como se puede ver en la figura 3.2.

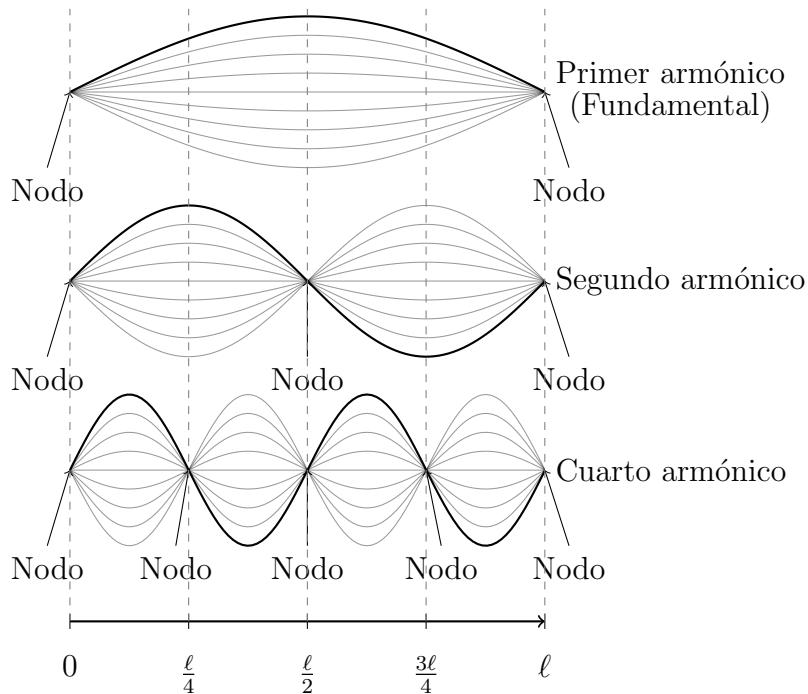


Figura 3.2: Ilustración de 3 armónicos y sus nodos de la cuerda al vibrar. *Fuente: Elaboración propia*

Si la solución propuesta por Bernouilli fuese correcta, ello obligaría a que

$$u(x, 0) = \sum_{n=1}^{\infty} C_n \sin(nx) \quad (3.10)$$

y por tanto a que

$$f(x) = \sum_{n=1}^{\infty} C_n \sin(nx), \quad \forall x \in [0, \ell], \quad (3.11)$$

para “una adecuada elección de los coeficientes C_n ”.

Pero esta fórmula no es correcta del todo pues le falta, a cada término, el producto por la función coseno (función del tiempo). Más aún, Bernoulli fue incapaz de mostrar cómo podían obtenerse los coeficientes $\alpha, \beta, \gamma, \delta, \dots$ cosa que sí hizo Euler un poco después con el método de integrar la expansión después

de multiplicarla por senos o cosenos. Es claro que la solución de Bernoulli no es tan general como la de D'Alembert pero algo que aún el gran Euler no pudo realmente entender es que la solución de Bernoulli permitía funciones iniciales más generales que, a decir de otros autores, fue tema de críticas por parte de Euler a la propuesta de D'Alembert [22]. Bernoulli y Euler mantuvieron una correspondencia epistolar productiva, y se puede afirmar que ambos dependían del otro para realizar su trabajo: Bernoulli necesitaba de Euler para orientar sus estudios matemáticos, mientras que Euler dependía de Bernoulli para comprender los fenómenos físicos que sustentaban su investigación matemática. A pesar de que partían de estudios matemáticos comunes, al final no lograron resolver ni juntos, ni de manera individual, ni desde una perspectiva experimental, donde la matemática jugara un papel metodológico predominante, ni desde el punto de vista matemático donde la física del problema guiara y definiera la teoría. Euler nunca consideró especialmente relevante el trabajo matemático de Daniel Bernoulli, ya que sus métodos eran algo imprecisos, aunque con gran intuición física. Por su parte, Bernoulli tampoco valoraba mucho los estudios matemáticos de Euler, pues estaban distantes de los experimentos. Bernoulli incluso llegó a comentar que Euler "tomaba sus pruebas de la naturaleza y no de algún principio de análisis". [22]

3.1.2. La ecuación de calor

Hubo que esperar 54 años hasta que las ideas de Bernoulli fueron tomadas en cuenta por el barón Jean Baptiste-Joseph Fourier, matemático y físico francés, quien, entre otras actividades acompañó a Napoleón, en calidad e científico, en la campaña de éste en Egipto. Allí, como secretario del Instituto de Egipto", hizo gala de gran competencia en diversos asuntos administrativos. [1]

3.1.2.1. Fourier

Al regresar a Francia, y como profesor de Análisis de la Escuela Politécnica, Fourier se interesó por la teoría de la conducción del calor en los cuerpos sólidos. En 1807 envió un artículo a la Academia de Ciencias de París, que trataba sobre dicho tema. Más concretamente, Fourier consideró una varilla delgada de longitud dada ℓ , cuyos extremos se mantienen a 0° centígrados y cuya superficie lateral está aislada. Si la distribución inicial de temperatura en la varilla viene dada por una función $f(x)$ (se supone que la temperatura de la varilla en cada sección transversal de la misma es constante), ¿Cuál será la temperatura de cualquier punto x de la varilla en el tiempo t ? Suponiendo que la varilla satisface condiciones físicas apropiadas, Fourier demostró que si $u(x, t)$ representa la temperatura en la sección x y en el tiempo t , entonces la función u debe tener la siguiente forma:

$$\frac{\partial u(x, t)}{\partial t} = \alpha^2 \frac{\partial^2 u(x, t)}{\partial x^2}, \quad 0 < x < \ell, \quad 0 < t < \infty \quad (3.12)$$

en donde:

- $\frac{\partial u}{\partial t}(x, t)$: Representa la derivada parcial de u con respecto al tiempo t , es decir, cómo cambia la temperatura $u(x, t)$ en el punto x a lo largo del tiempo t .

- $u(x, t)$: Es la función que describe la temperatura en un punto x en el espacio y en un tiempo t . Depende tanto de la posición espacial x como del tiempo t .
- α^2 : Es el coeficiente de difusión térmica, que depende del material en cuestión. Este coeficiente es constante y está relacionado con la capacidad del material para difundir el calor. La unidad de α es metros cuadrados por segundo (m^2/s).
- $\frac{\partial^2 u}{\partial x^2}(x, t)$: Representa la derivada parcial segunda de u con respecto a la posición x , es decir, cómo cambia la pendiente (o curvatura) de la temperatura a lo largo del espacio. Esta cantidad indica cómo el calor se distribuye espacialmente.

Esta ecuación modela la difusión del calor en un medio a lo largo del tiempo. La derivada en el tiempo ($\frac{\partial u}{\partial t}$) está relacionada con la derivada segunda en el espacio ($\frac{\partial^2 u}{\partial x^2}$), lo que refleja que el cambio en la temperatura con el tiempo depende de cómo está distribuido el calor en el espacio. y esta debe satisfacer las siguientes condiciones:

$$\begin{aligned} u(0, t) &= u(\ell, t) = 0, \quad 0 \leq t \leq \infty \\ u(x, 0) &= f(x), \quad 0 \leq x \leq \ell. \end{aligned} \tag{3.13}$$

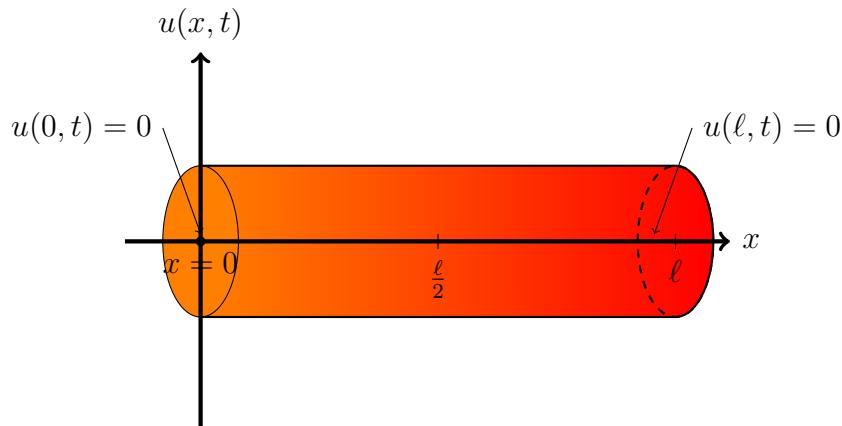


Figura 3.3: Ilustración del problema de transferencia de calor en una barra de longitud π . *Fuente: Elaboración propia*

La primera condición en 3.12 es una Ecuación en Derivadas Parciales de segundo orden, conocida con el nombre de Ecuación del Calor. La segunda significa que la temperatura, en los extremos de la varilla, se mantiene a 0° centígrados en cualquier tiempo, mientras que la última relación representa la distribución inicial de temperatura en la varilla considerada.

Partiendo de las ideas de Bernoulli, para la ecuación de ondas, Fourier buscó las soluciones más sencillas que puede presentar la ecuación del calor: aquellas que son de la forma:

$$u(x, t) = X(x)P(t) \tag{3.14}$$

Imponiendo la condición de que tales funciones satisfagan, formalmente, dicha ecuación, obtenemos, como en el caso de la ecuación de ondas, los dos problemas siguientes de ecuaciones diferenciales ordinarias [23]:

$$\begin{aligned} X''(x) + \mu X(x) &= 0, \quad x \in (0, \ell), \quad X(0) = X(\ell) = 0 \\ P'(t) + \mu P(t) &= 0, \quad t > 0 \end{aligned} \quad (3.15)$$

Utilizando el método de separación de variables [24], Fourier propuso que la solución a la ecuación del calor podría expresarse como el producto de dos funciones, una dependiente de la posición $X(x)$ y otra del tiempo $P(t)$, como se indica en (3.13). Esto descompone la ecuación en dos ecuaciones diferenciales ordinarias: una espacial (3.15) y una temporal (3.15). Fourier resolvió cada una de estas ecuaciones por separado. La solución espacial $X(t)$ resulta en una serie de funciones sinusoidales que satisfacen las condiciones de frontera en los extremos de la varilla, mientras que la solución temporal $P(t)$ resulta en exponentes negativos que representan la disipación de calor en el tiempo. Así, disponemos de un procedimiento que nos permite calcular infinitas “soluciones elementales” de la ecuación del calor, a saber, las funciones de la forma $b_n v_n$, donde v_n se define como:

$$v(x, t) = \sin\left(\frac{n\pi}{\ell}x\right) e^{-\alpha^2 \frac{n^2 \pi^2}{\ell^2} kt} \quad (3.16)$$

Es trivial que, si la distribución inicial de temperatura, f , es algún múltiplo de $\sin(nx)$ (o una combinación lineal finita de funciones de este tipo), entonces la solución buscada de (3.16) es un múltiplo adecuado de v_n (respectivamente, una adecuada combinación lineal de funciones de esta forma).

Ahora bien, $f(x)$ no es, en general, de la forma justo mencionada, pero, y aquí demostró Fourier, como Bernouilli, una enorme intuición, ¿Será posible obtener la solución $u(x, t)$ de (3.16), para cualquier $f(x)$ dada, como superposición de las anteriores soluciones sencillas v_n ? Es decir, ¿Será posible elegir adecuadamente los coeficientes b_n tal que la única solución de (3.16) sea de la forma:

$$u(x, t) = \sum_{n=1}^{\infty} b_n \sin\left(\frac{n\pi}{\ell}x\right) e^{-\alpha^2 \frac{n^2 \pi^2}{\ell^2} kt} \quad (3.17)$$

Fourier afirmó en su artículo que esto era correcto. Observemos que nuevamente llegamos a que, entonces, se debe satisfacer la relación (3.9). Esto plantea la misma cuestión para dos problemas completamente distintos, el problema (3.1) y el problema (3.12), en donde los coeficientes b_n toman la forma:

$$b_n = \frac{2}{\ell} \int_0^{\ell} f(x) \sin\left(\frac{n\pi x}{\ell}\right) dx \quad (3.18)$$

Los coeficientes b_n se determinan en función de la distribución inicial de temperatura $f(x)$ a través de la expresión en (3.18), lo que permite que la solución refleje la distribución inicial específica de la varilla. Así, la solución general toma la forma de una serie de Fourier, como se observa en (3.17), donde

cada término de la serie combina una función sinusoidal en el espacio con un decaimiento exponencial en el tiempo.

El artículo Fourier fue evaluado por Lagrange, Laplace y Legendre y fue rechazado por la Academia Francesa, principalmente debido a la forma en que dedujo la ecuación del calor y por la falta de rigor en sus conclusiones (según la opinión de los académicos mencionados). Sin embargo, los miembros de dicha institución reconocían la relevancia de los problemas relacionados con la propagación del calor, y los resultados teóricos que Fourier presentó mostraban gran concordancia con varios experimentos realizados previamente [17].

Debido a esto, la Academia estableció un premio sobre el tema. Dicho premio fue otorgado a Fourier en 1812, pero, a pesar de esto, los académicos continuaron criticando la falta de rigor, de modo que, aunque ganó el premio, Fourier no logró publicar su trabajo en la famosa serie “*Mémoires*” de la Academia Francesa. Con gran perseverancia, Fourier continuó trabajando en el tema, y en 1822 publicó su célebre libro *Théorie Analytique de la Chaleur*, Firmin Didot, Père et Fils, 1822, París, donde incluyó gran parte de su artículo de 1812 casi sin modificaciones. Este libro es actualmente una de las obras clásicas en matemáticas [17].

Dos años después, obtuvo el puesto de Secretario de la Academia Francesa, lo que le permitió finalmente publicar su artículo en la serie “*Mémoires*” [17].

3.2. Series de Fourier

Luego de que las soluciones para la ecuación de onda y de calor fueron desarrolladas, Joseph Fourier propuso un nuevo método para expresar funciones periódicas en función de sinusoides. Las bases de lo que hoy conocemos como series de Fourier fueron establecidas por esta propuesta.

A continuación, se describen los conceptos matemáticos en los cuales se basan las series de Fourier, ofreciendo una base firme para entenderlas y aplicarlas en el estudio de funciones que se repiten periódicamente.

3.2.1. Funciones Periódicas

Una función es llamada *función periódica* si existe una constante $T > 0$ tal que

$$f(t) = f(t \pm T) \quad (3.19)$$

para todo valor de t en el dominio de definición de $f(t)$, donde la constante T se denomina *periodo* de la función. El periodo también se puede definir como el tiempo transcurrido entre dos puntos equivalentes de una función [25].

Las funciones periódicas aparecen en muchas aplicaciones de matemáticas para problemas de física e ingeniería. Es evidente que la suma, diferencia, producto o cociente de dos funciones con período T también será una función con período T [25].

Si graficamos una función periódica $f(t)$ en un intervalo cerrado $a \leq t \leq a \pm T$, podemos obtener la gráfica completa de $f(t)$ repitiendo periódicamente la porción de la gráfica correspondiente a $a \leq t \leq a \pm T$ 3.4.

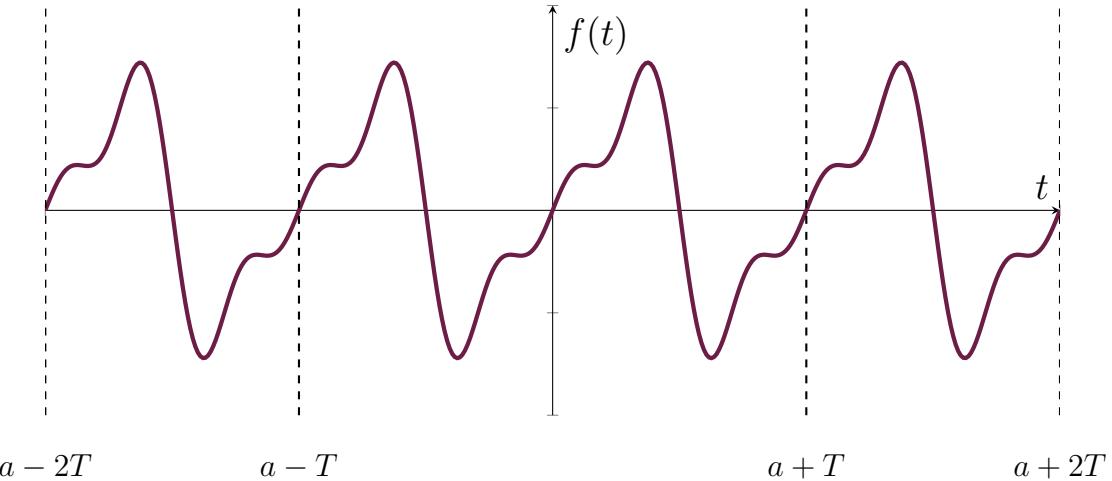


Figura 3.4: Gráfica de una función periódica $f(t)$ con periodo T . Fuente: *Elaboración propia*

Si T es un período de la función $f(t)$, entonces los números $2T, 3T, 4T, \dots$ también son períodos. Esto se deduce fácilmente al inspeccionar la gráfica de una función periódica o a partir de la serie de igualdades [26]

$$f(t) = f(t \pm T) = f(t \pm 2T) = f(t \pm 3T) = \dots = f(t \pm nT), \quad n = 0, \pm 1, \pm 2, \dots \quad (3.20)$$

que se obtiene mediante el uso repetido de la condición (3.19). Así, si T es un período, también lo es nT , donde n es cualquier entero positivo, es decir, si existe un período, no es único [26]. Además, podemos definir como el *periodo fundamental* T_0 de $f(t)$ a el valor positivo más pequeño de T para el cual se satisface (3.19). [27]

Ahora, consideremos la siguiente propiedad de cualquier función $f(t)$ con período T :

Si $f(t)$ es integrable en cualquier intervalo de longitud T , entonces es integrable en cualquier otro intervalo de la misma longitud, y el valor de la integral es el mismo. Es decir,

$$\int_a^{a+T} f(t) dx = \int_b^{b+T} f(t) dt \quad (3.21)$$

para cualquier a y b .

Esta propiedad es una consecuencia directa de la interpretación del área bajo la curva como la integral. De hecho, cada integral de la ecuación anterior representa el área entre la curva $f(t)$, el eje t , y las líneas verticales en los puntos extremos del intervalo. Las áreas sobre el eje t se consideran positivas y las áreas bajo el eje t como negativas. En este caso, las áreas representadas por ambas integrales son iguales debido a la periodicidad de $f(t)$ [25] 3.5.

A partir de ahora, cuando afirmemos que una función $f(t)$ de período T es integrable, queremos decir que es integrable en un intervalo de longitud T . Esto implica, a partir de la propiedad recién demostrada, que $f(t)$ es integrable en

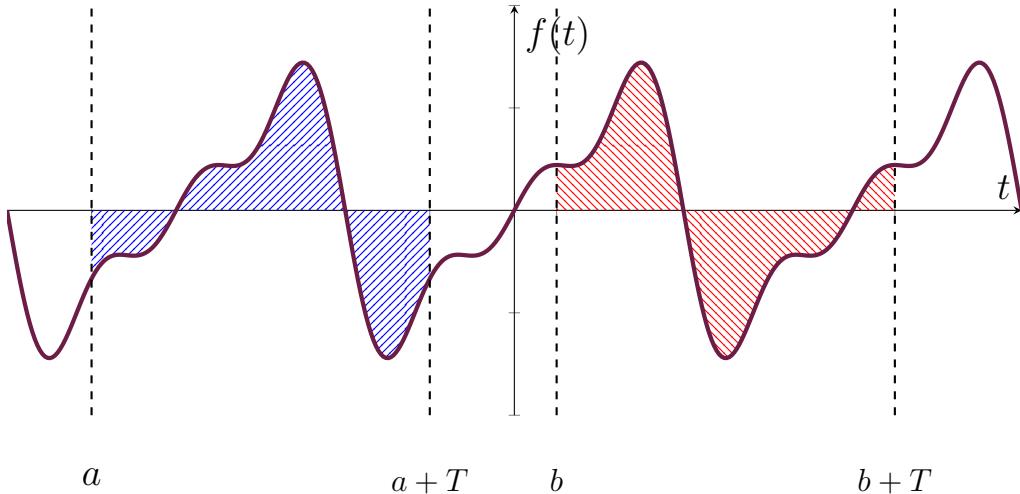


Figura 3.5: Gráfica del área bajo la curva de $f(t)$ entre los intervalos desde a hasta $a + T$ y desde b hasta $b + T$. *Fuente: Elaboración propia*

cualquier intervalo de longitud finita [25].

Las funciones periódicas más representativas son las funciones trigonométricas $\sin(t)$ y $\cos(t)$, ambas con periodo $T = 2\pi$ (Figura 3.6). Por lo tanto, de acuerdo con (3.19), se tiene que $\sin(t + 2\pi) = \sin(t)$ y $\cos(t + 2\pi) = \cos(t)$ [27].

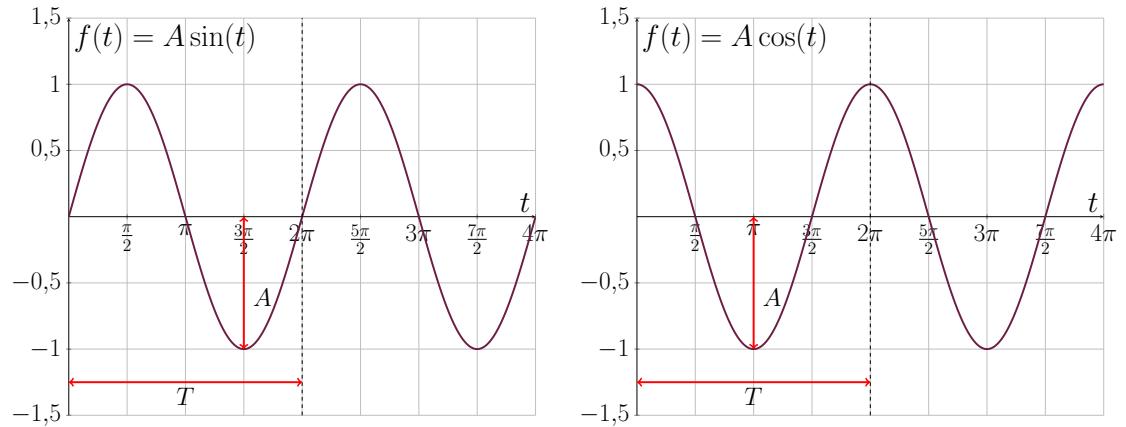


Figura 3.6: Gráfica de las funciones trigonométricas $\sin(t)$ y $\cos(t)$. *Fuente: Elaboración propia*

3.2.2. Frecuencia

La relación entre la *frecuencia* f y el *periodo* T de una función oscilatoria está dada por

$$f = \frac{1}{T} \quad (3.22)$$

lo cual indica el número de oscilaciones de la función por unidad de tiempo. La unidad de medida de la frecuencia es el hertz (Hz). Un hertz quiere decir que una función oscila una vez por segundo [27]. La Figura 3.7 muestra una función senoidal con tres frecuencias diferentes.

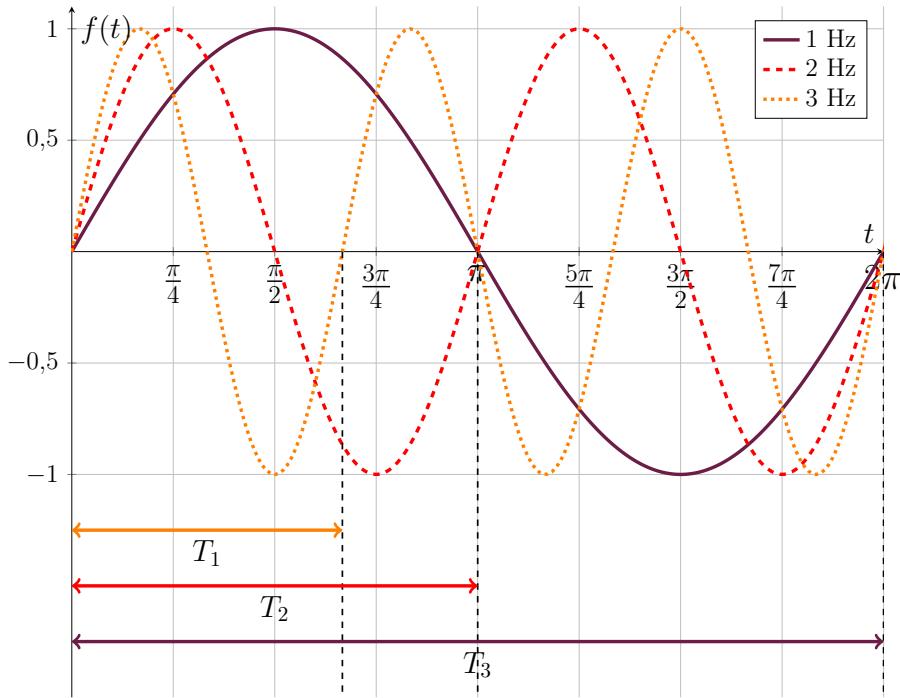


Figura 3.7: Función $\sin(t)$ con diferentes frecuencias. Fuente: Elaboración propia

La *frecuencia angular*, usualmente denotada por ω_0 , se define como la razón de cambio del desplazamiento angular θ durante la oscilación (o rotación) [27]:

$$\frac{d\theta}{dt} = \omega_0 = 2\pi f = \frac{2\pi}{T}. \quad (3.23)$$

La frecuencia angular se mide comúnmente en radianes por segundo (rad/s). La Figura 3.8 ejemplifica la relación de una onda senoidal en el intervalo $[0, 2\pi]$ con una frecuencia angular ω_0 . [27]

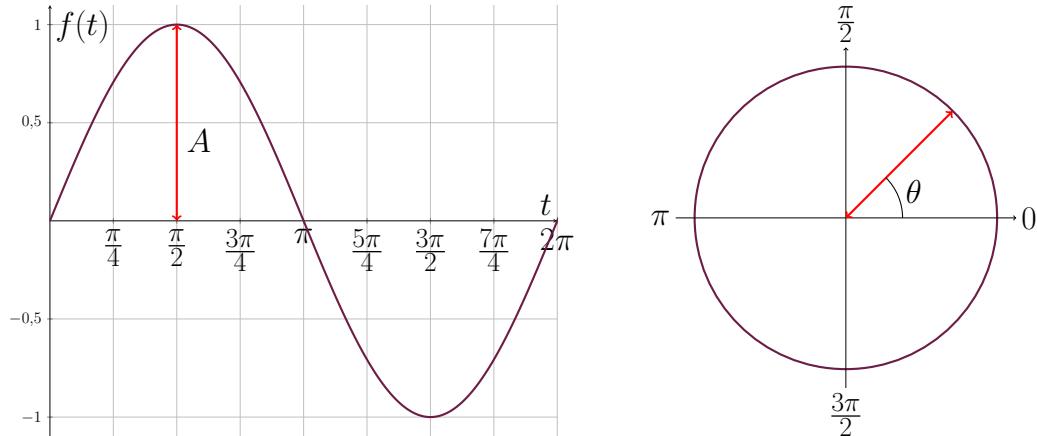


Figura 3.8: Relación de la frecuencia angular entre la función $\sin(t)$ con el círculo unitario. Fuente: Elaboración propia

3.2.3. Armónicos

La función periódica de mayor importancia es:

$$f(t) = A \sin(\omega_0 t + \varphi) \quad \text{o} \quad f(t) = A \cos(\omega_0 t + \varphi) \quad (3.24)$$

Cabe señalar que, dado que las funciones seno y coseno solo se diferencian por un deslizamiento de fase, este movimiento podría modelarse usando la función coseno o la función seno, en donde A , ω_0 , y φ son constantes. En una función armónica, la *amplitud* A representa el valor máximo de la oscilación. La *frecuencia angular* ω_0 determina la rapidez con la que se realiza la oscilación y viene dada por $\frac{2\pi}{T}$, recordando que T es el periodo de la función, y la *fase inicial* φ indica el desplazamiento de la función en el tiempo al inicio de la observación, es decir, indica con qué desfase comienza. [28].

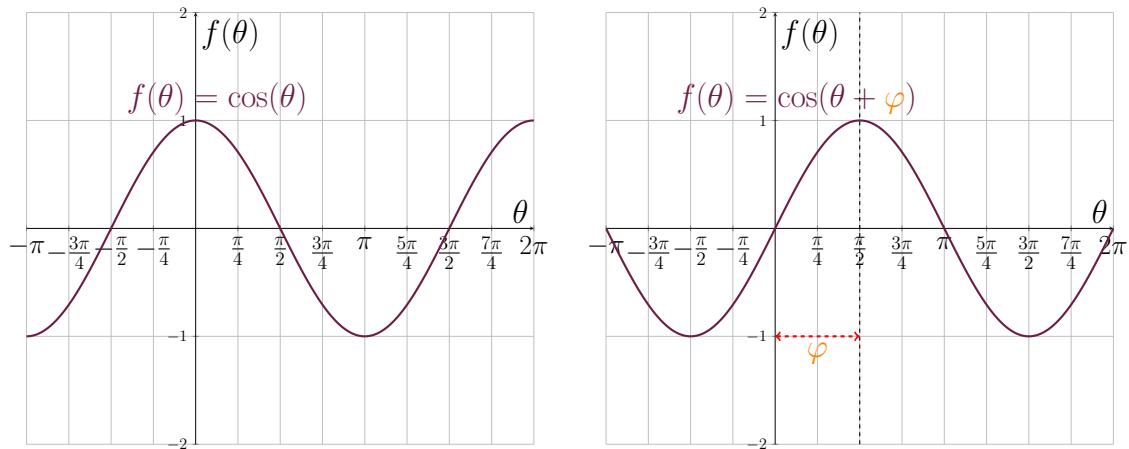


Figura 3.9: Una función coseno y la misma función coseno desplazada hacia la izquierda por un ángulo φ . Fuente: Elaboración propia

3.2.4. Funciones Pares e Impares

Más adelante se verá como es que el hecho de que una función $f(t)$ sea par o impar puede ahorrar diversos cálculos, comenzamos definiendo que una función es par si se cumple que: [29]

$$f(t) = f(-t), \quad -L < t < L \quad (3.25)$$

En la Figura 3.10 se muestra un ejemplo de una función para en donde se puede observar la definición (3.25)

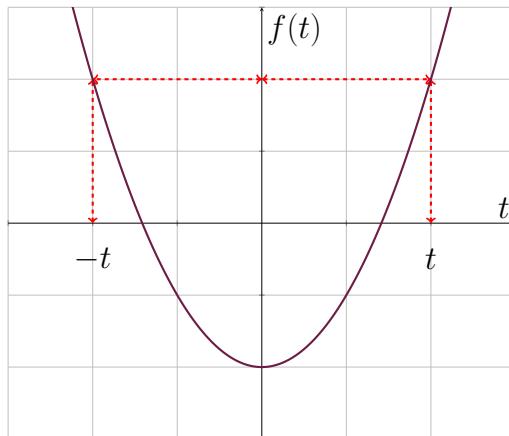


Figura 3.10: Función $f(t) = t^2$, una función par. *Fuente: Elaboración propia*

Ahora, decimos que una función es impar si se cumple que: [29]

$$f(t) = -f(-t), \quad -L < t < L \quad (3.26)$$

En la Figura 3.11 se muestra un ejemplo de una función para en donde se puede observar la definición (3.26)

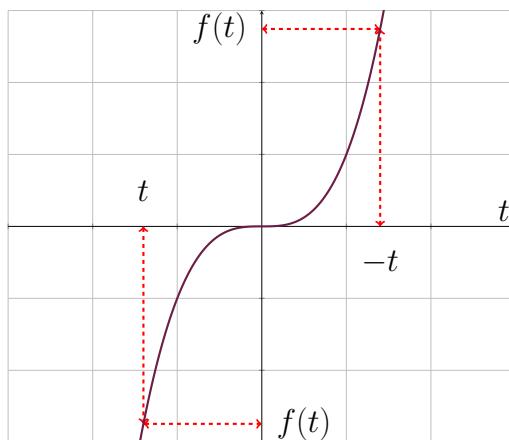


Figura 3.11: Función $f(t) = t^3$, una función impar. *Fuente: Elaboración propia*

Podemos decir entonces que en un intervalo simétrico $-L < t < L$, la gráfica de una función par tiene simetría respecto al eje y, mientras que la gráfica de una función impar tiene simetría en relación con el origen. Además, las funciones pares e impares cumplen con los siguientes teoremas [29]:

- a) El producto de dos funciones pares es par.
- b) El producto de dos funciones impares es par.
- c) El producto de una función par y una impar es impar.
- d) La suma (resta) de dos funciones pares es par.
- e) La suma (resta) de dos funciones impares es impar.

f) Si f es par, entonces $\int_{-a}^a f(t) dt = 2 \int_0^a f(t) dt$.

g) Si f es impar, entonces $\int_{-a}^a f(t) dt = 0$.

En la Figura 3.12 podemos observar como es que se cumplen los teoremas f y g, ya que en la primera el área de la función desde $-a$ hasta 0 es el negativo a su contraparte que se encuentra desde 0 hasta a , lo que significa que el área conjunta desde $-a$ hasta a se anulará dando 0, por otra parte, la segunda imagen nos muestra que el área desde $-a$ hasta 0 es la misma que la que hay desde 0 hasta a , lo que implica que el área desde $-a$ hasta a es lo doble de el área desde 0 hasta $\pm a$.

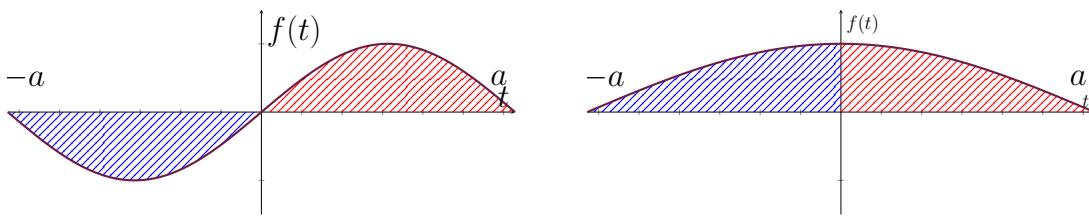


Figura 3.12: Ejemplo de los teoremas f y g de funciones pares e impares *Fuente: Elaboración propia*

3.2.5. Funciones suaves y funciones suaves a trozos

Una función $f(t)$ se dice que es *suave* en el intervalo $[a, b]$ si tiene una derivada *continua* en $[a, b]$. En términos geométricos, esto significa que la dirección de la tangente cambia de forma *continua*, sin saltos, mientras se desplaza a lo largo de la curva $f(t)$ [ver Figura 3.13]. Así, el gráfico de una función suave es una curva suave sin “saltos”. [25]

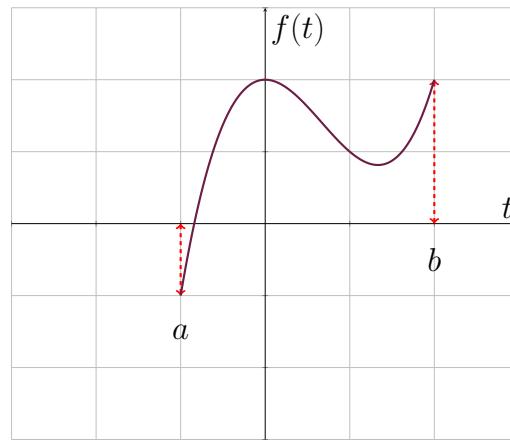


Figura 3.13: Ejemplo de una función suave. *Fuente: Elaboración propia*

La función $f(t)$ se dice que es *suave a trozos* en el intervalo $[a, b]$ si tanto $f(t)$ como su derivada son continuas en $[a, b]$, o si tienen solamente un número finito de discontinuidades de salto en $[a, b]$. Es fácil ver que el gráfico de una función suave por partes es o bien una curva continua o una curva discontinua que puede tener un número finito de saltos (en las cuales la derivada tiene saltos). A medida

que nos acercamos a cualquier discontinuidad o salto (desde un lado o el otro), la dirección de la tangente se aproxima a una posición límite definida, ya que la derivada sólo puede tener discontinuidades de salto. Las figuras 3.14 ilustran las gráficas de funciones suaves a trozos continuas y discontinuas. [25]

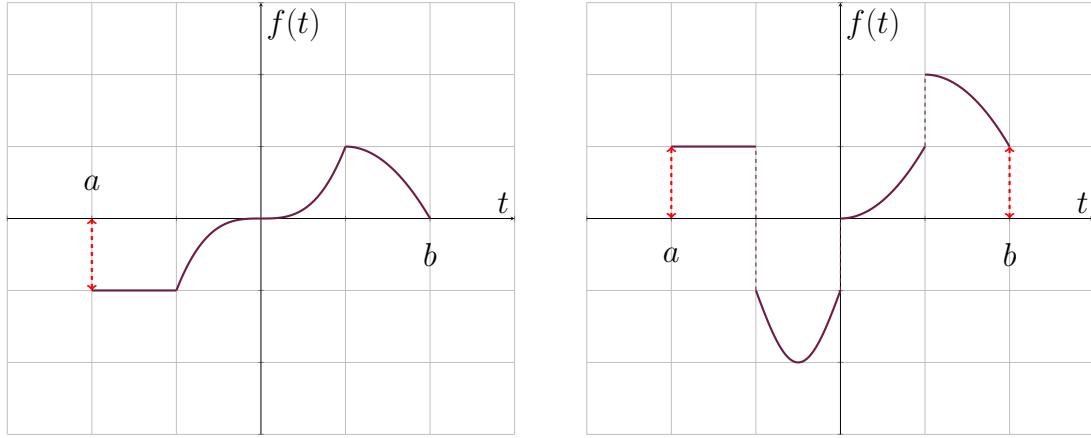


Figura 3.14: Ejemplo de una función suave a trozos completamente continua y una función a trozos con discontinuidades (saltos). *Fuente: Elaboración propia*

Entonces, una función continua o discontinua $f(t)$ definida en todo el eje t se dice que es *suave a trozos* si, en cualquier intervalo finito, se comporta de manera que su derivada es continua o tiene pocas discontinuidades. Esto significa que la función puede tener ciertos "saltos."° cambios bruscos en su forma, pero entre estos puntos, su comportamiento es regular y suave. Este concepto es útil para funciones periódicas, que se repiten en ciclos. Toda función $f(t)$ suave a trozos (ya sea continua o discontinua) está acotada, lo que implica que no puede crecer indefinidamente y tiene una derivada acotada en todas partes, excepto en sus .°esquinaz° puntos de discontinuidad (es decir, en los puntos donde la función cambia abruptamente o tiene saltos), donde la derivada $f'(t)$ no existe. [25]

3.2.6. Funciones Ortogonales

En álgebra lineal, el producto escalar (también conocido como producto interno o producto punto) entre dos vectores **a** y **b** se basa en la proyección de un vector sobre el otro. Esto permite medir cuánto el vector **a** apunta en la misma dirección que el vector **b**. Si ambos vectores apuntan en direcciones similares, el producto escalar es positivo; en cambio, si apuntan en direcciones opuestas, el producto escalar será negativo. Cuando los vectores son ortogonales, su producto escalar resulta en cero, lo cual indica que no se pueden representar uno en función del otro. [30]

El concepto de ortogonalidad de funciones es análogo al de perpendicularidad de vectores; es decir, funciones ortogonales proporcionan información mutuamente excluyente. Por lo tanto, se puede decir que dos funciones diferentes $f_1(t)$ y $f_2(t)$ son ortogonales en el intervalo $a < t < b$ cuando su producto interno es cero:

$$\langle f(t)_1, (t)_2 \rangle = \int_a^b f_1(t) f_2(t) dt = 0. \quad (3.27)$$

En general, se dice que el conjunto de funciones de valor real $\{\phi_0(t), \phi_1(t), \phi_2(t), \dots, \phi_n(t)\}$ es ortogonal en el intervalo $[a, b]$ si [29]

$$\langle \phi_n(t), \phi_m(t) \rangle = \int_a^b \phi_n(t) \phi_m(t) dt = 0, \quad m \neq n. \quad (3.28)$$

3.2.6.1. Ortogonalidad del seno y del coseno

Ahora, consideremos que tenemos conjuntos de funciones senoidales y cosenoidales recordando que $\omega_0 = \frac{2\pi}{T}$, a través del cálculo integral podemos encontrar que:

$$\int_{-T/2}^{T/2} \cos(m\omega_0 t) dt = 0, \quad \text{para } m \neq 0 \quad (3.29)$$

$$\int_{-T/2}^{T/2} \sin(m\omega_0 t) dt = 0, \quad \text{para todo valor de } m \quad (3.30)$$

$$\int_{-T/2}^{T/2} \cos(m\omega_0 t) \cos(n\omega_0 t) dt = \begin{cases} 0, & m \neq n \\ T/2, & m = n \neq 0 \end{cases} \quad (3.31)$$

$$\int_{-T/2}^{T/2} \sin(m\omega_0 t) \sin(n\omega_0 t) dt = \begin{cases} 0, & m \neq n \\ T/2, & m = n \neq 0 \end{cases} \quad (3.32)$$

$$\int_{-T/2}^{T/2} \sin(m\omega_0 t) \cos(n\omega_0 t) dt = 0, \quad \text{para todo valor de } m \text{ y } n \quad (3.33)$$

Estas relaciones demuestran que el conjunto de funciones $\{1, \cos \omega_0 t, \cos 2\omega_0 t, \dots, \cos n\omega_0 t, \dots, \sin \omega_0 t, \sin 2\omega_0 t, \dots, \sin n\omega_0 t, \dots\}$ son ortogonales en el intervalo $-\frac{T}{2} < t < \frac{T}{2}$.

3.2.7. Serie Trigonométrica de Fourier

Como ya vimos anteriormente, las series de Fourier, fueron introducidas por el matemático francés Jean Baptiste Joseph Fourier con el objetivo de aproximar funciones periódicas utilizando un conjunto de funciones ortogonales como $\{1, \cos \omega_0 t, \cos 2\omega_0 t, \dots, \cos n\omega_0 t, \dots, \sin \omega_0 t, \sin 2\omega_0 t, \dots, \sin n\omega_0 t, \dots\}$ [29].

La idea fundamental detrás de las series de Fourier es descomponer una función periódica en términos de una suma infinita de funciones básicas, senos y cosenos, cuyas frecuencias son múltiplos de la frecuencia de la función original. La ortogonalidad de las funciones seno y coseno, usadas como "bases", es fundamental porque permite descomponer la función original en componentes que representan diferentes armónicos o frecuencias sin que estos interfieran entre sí, asegurando que cada término de la serie tenga una contribución bien definida. Esta descomposición permite analizar las propiedades de una función o señal y facilita la síntesis de objetos o fenómenos.

3.2.7.1. Series de Fourier de una función con período T

Si $f(t)$ es una función periódica de periodo T 3.15, se puede expresar mediante una serie trigonométrica como:

$$f(t) = \frac{a_0}{2} + a_1 \cos(\omega_0 t) + a_2 \cos(2\omega_0 t) + \dots + b_1 \sin(\omega_0 t) + b_2 \sin(2\omega_0 t) + \dots \quad (3.34)$$

o de forma general como [26]:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)] \quad (3.35)$$

donde $\omega_0 = \frac{2\pi}{T}$ es la frecuencia angular fundamental y los coeficientes a_0 , a_n y b_n son los coeficientes de Fourier dados por (3.36) [26]:

$$a_0 = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) dt, \quad a_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cos(n\omega_0 t) dt, \quad b_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \sin(n\omega_0 t) dt \quad (3.36)$$

La ecuación (3.35) describe cómo una función periódica arbitraria puede representarse como una suma infinita de componentes sinusoidales con diferentes frecuencias. El componente de la frecuencia $\omega_n = n\omega_0$ se denomina el n -ésimo armónico de la función periódica, donde este n -ésimo componente os dice cuanto de la n -ésimo frecuencia está presente o aporta a la función original.

Otra forma muy común de representar una función $f(t)$ en una serie de Fourier es cuando el periodo T se encuentra definido en un intervalo de $-p$ a p , que es completamente equivalente a cuando está definida sobre $-\frac{T}{2}$ a $\frac{T}{2}$, para este caso, la fórmula de la serie sigue siendo la misma establecida previamente en (3.35), lo que cambia levemente es como se definen nuestros coeficientes: [29]

$$a_0 = \frac{1}{p} \int_{-p}^p f(t) dt, \quad a_n = \frac{1}{p} \int_{-p}^p f(t) \cos(n\omega_0 t) dt, \quad b_n = \frac{1}{p} \int_{-p}^p f(t) \sin(n\omega_0 t) dt \quad (3.37)$$

donde también cambia un poco como se define nuestra frecuencia fundamental $\omega_0 = \frac{\pi}{p}$. [29]:

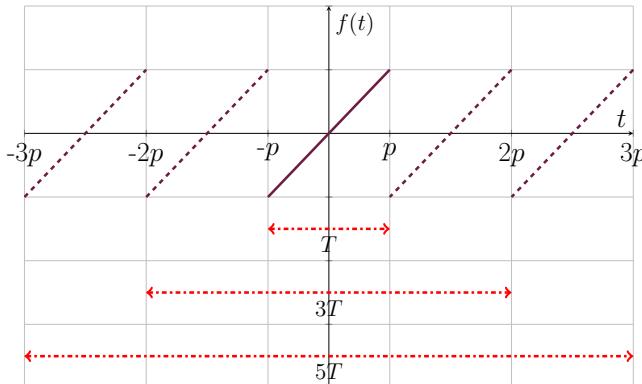


Figura 3.15: Gráfica de una función periódica $f(t)$ con periodo T , repitiéndose en intervalos de T , $2T$, y $3T$. Fuente: Elaboración propia

3.2.7.2. Criterios para funciones pares e impares

Como vimos anteriormente, el que una función sea par o impar podrá ahorrar potenciales cálculos, en primera, podemos decir que si nuestra función $f(t)$ es una función par, entonces los únicos coeficientes que será necesario calcular serán los coeficientes a_0 y a_n (3.36). Pero, ¿qué pasa con el coeficiente b_n ? Recordemos que el coeficiente b_n en la serie de Fourier está dado por la integral:

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(\sin(n\omega_0 t)) dt$$

Dado que $f(t)$ es par y $\sin(\sin(n\omega_0 t))$ es una función impar, al multiplicarlos obtenemos una función impar [29], como ya vimos anteriormente. Entonces, esto implica que cuando integremos esta función impar resultante sobre un intervalo simétrico alrededor del origen (como $[-\frac{T}{2}, \frac{T}{2}]$), la integral resultante será cero:

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin(\sin(n\omega_0 t)) dt = 0$$

Por lo tanto, todos los coeficientes b_n se cancelan automáticamente cuando $f(t)$ es una función par, ya que la contribución de los términos de seno se anula debido a la simetría impar del integrando. Esto simplifica la serie de Fourier, ya que solo es necesario calcular los coeficientes a_0 , correspondiente al promedio de la función y a_n , correspondiente a los términos de coseno, que son funciones pares.

Algo similar sucede cuando nuestra función $f(t)$ es una función impar, en este caso el único coeficiente que sobrevive es el coeficiente b_n . Sabemos que el coeficiente a_n en la serie de Fourier se calcula mediante la integral:

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos(n\omega_0 t) dt$$

Dado que $f(t)$ es una función impar y $\cos(n\omega_0 t)$ es una función par, el producto $f(t) \cos(n\omega_0 t)$ será una función impar. Esto es porque la multiplicación de una función impar con una función par produce una función impar. [29]

Al integrar una función impar en un intervalo simétrico alrededor del origen, como $[-T/2, T/2]$, la integral resultará en cero. Esto implica que:

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos(n\omega_0 t) dt = 0$$

Por un razonamiento análogo, el coeficiente a_0 , que representa el promedio de la función y se calcula mediante la integral:

$$a_0 = \frac{2}{T} \int_{-T/2}^{T/2} f(t) dt$$

también será igual a cero, ya que la función $f(t)$ es impar y, al integrarla sobre un intervalo simétrico, la integral se anula.

Por lo tanto, cuando $f(t)$ es una función impar, todos los coeficientes a_n y a_0 se cancelan automáticamente, ya que la contribución de los términos de coseno y

del promedio de la función se anula debido a la simetría impar del integrando. Así, en este caso, solo es necesario calcular los coeficientes b_n , correspondientes a los términos de seno, los cuales son funciones impares y, por lo tanto, sobreviven en la serie de Fourier.

Entonces, podemos decir que la serie de Fourier para una función $f(t)$ cuando esta función es **par** es:

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(n\omega_0 t) \quad (3.38)$$

Y la serie de Fourier para una función $f(t)$ cuando esta función es **ímpar** es:

$$f(t) = \sum_{n=1}^{\infty} b_n \sin(n\omega_0 t), \quad (3.39)$$

donde los coeficientes de Fourier cuando se mantienen igual 3.36, solo es cuestión de aplicar lo previamente visto para saber cuando calcular los necesarios:

3.2.8. Extensiones de Medio Intervalo

En la sección anterior observamos que para desarrollar una función $f(t)$ en una serie de Fourier debía estar declarada en un intervalo simétrico $-\frac{T}{2}, \frac{T}{2}$, sin embargo, en muchos casos nos interesa representar una función definida para $0 < x < T$, es decir, comenzando en 0, mediante una serie trigonométrica estas, se conocen como *extensiones de medio rango*, recordando que $\omega_0 = \frac{2\pi}{T}$ es la frecuencia angular fundamental. De aquí existen 3 posibilidades, asumir que la función a expandir en serie de Fourier es par, ímpar o periódica [31].

3.2.8.1. Extensión Par (Serie de Cosenos)

La extensión par hace que la función sea simétrica respecto al eje y (Figura 3.16), permitiendo extender el análisis a un intervalo completo de manera simétrica.

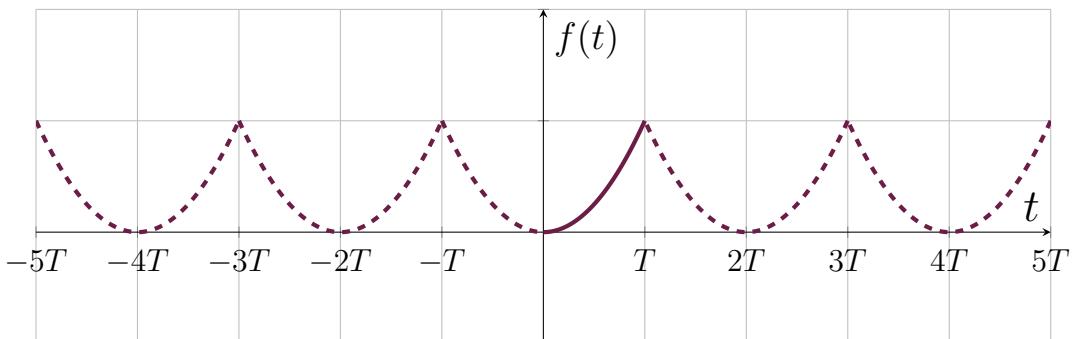


Figura 3.16: Extensión par en cosenos de una función $f(t)$. Fuente: Elaboración propia

Para la extensión par, la serie de Fourier se define de la misma manera que cuando tenemos una función par $f(t)$ (3.38):

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(n\omega_0 t) \quad (3.40)$$

Pero los coeficientes se verán afectados de la siguiente forma: [31]

$$a_0 = \frac{2}{T} \int_0^T f(t) dt, \quad a_n = \frac{2}{T} \int_0^T f(t) \cos(n\omega_0 t) dt, \quad (3.41)$$

3.2.8.2. Extensión Impar (Serie de Senos)

La extensión impar hace que la función sea simétrica respecto al origen (Figura 3.17), permitiendo analizarla en un intervalo completo con una continuidad en la transición de signo.

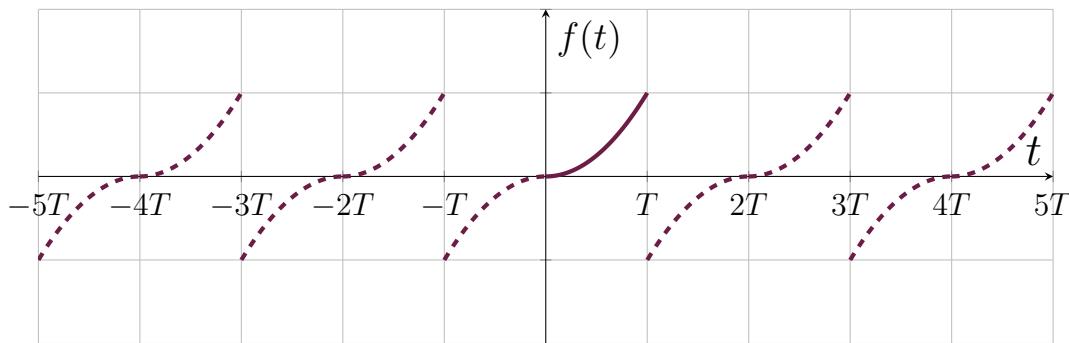


Figura 3.17: Extensión impar en senos de una función $f(t)$. Fuente: Elaboración propia

Para la extensión impar, la serie de Fourier se define de la misma manera que cuando tenemos una función impar $f(t)$ (3.39):

$$f(t) = \sum_{n=1}^{\infty} b_n \sin(n\omega_0 t), \quad (3.42)$$

Pero el coeficiente se verá afectado de la siguiente forma: [31]

$$b_n = \frac{2}{T} \int_0^T f(t) \sin(n\omega_0 t) dt \quad (3.43)$$

3.2.8.3. Extensión Periódica

La extensión periódica replica la función en intervalos sucesivos, creando una función periódica que se extiende indefinidamente en ambos sentidos (Figura 3.18).

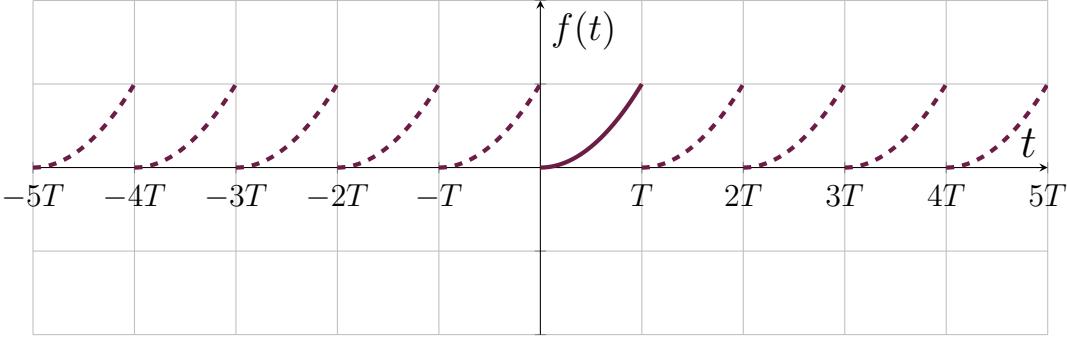


Figura 3.18: Extensión periódica de medio rango de una función $f(t)$. *Fuente: Elaboración propia*

Para la extensión periódica, al no ser par ni impar, requerirá de los 3 coeficientes, y a diferencia de las extensiones par e impar: [31]

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)], \quad (3.44)$$

$$a_0 = \frac{2}{T} \int_0^T f(t) dt, \quad a_n = \frac{2}{T} \int_0^T f(t) \cos(2n\omega_0 t) dt, \quad b_n = \frac{2}{T} \int_0^T f(t) \sin(n\omega_0 t) dt \quad (3.45)$$

3.2.9. Serie Compleja de Fourier

Una forma compacta de expresar a la serie trigonométrica de Fourier (3.35) es definiría en términos de una exponencial compleja. Para lograr esto, se representan a las funciones de seno y coseno usando la notación de Euler $e^{i\theta} = \cos(\theta) + i\sin(\theta)$, donde $i^2 = -1$. [32]

$$\cos t = \frac{e^{it} + e^{-it}}{2}, \quad \sin t = \frac{e^{it} - e^{-it}}{2i} \quad (3.46)$$

Al utilizar (3.46) para reemplazar $\cos(n\omega_0 t)$ y $\sin(n\omega_0 t)$ en (3.46), la serie de Fourier de una función f puede escribirse como

$$\begin{aligned} f(t) &= \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \frac{e^{i(n\omega_0 t)} + e^{-i(n\omega_0 t)}}{2} + b_n \frac{e^{i(n\omega_0 t)} - e^{-i(n\omega_0 t)}}{2i} \right] \\ &= \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[\frac{1}{2} (a_n - ib_n) e^{i(n\omega_0 t)} + \frac{1}{2} (a_n + ib_n) e^{-i(n\omega_0 t)} \right] \\ &= c_0 + \sum_{n=1}^{\infty} c_n e^{i(n\omega_0 t)} + \sum_{n=1}^{\infty} c_{-n} e^{-i(n\omega_0 t)} \end{aligned} \quad (3.47)$$

donde $c_0 = \frac{1}{2}a_0$, $c_n = \frac{1}{2}(a_n - ib_n)$ y $c_{-n} = \frac{1}{2}(a_n + ib_n)$. Los símbolos a_n , b_n son los coeficientes de la definición (3.35). Cuando la función $f(t)$ es real, c_n y c_{-n} son complejos conjugados y pueden escribirse también en términos de las funciones exponenciales complejas:

$$c_0 = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) dt \quad (3.48)$$

$$\begin{aligned} c_n &= \frac{2}{T} (a_n - ib_n) = \frac{2}{T} \left(\frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cos(n\omega_0 t) dt - i \frac{T}{2} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \sin(n\omega_0 t) dt \right) \\ &= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) (\cos(n\omega_0 t) - i \sin(n\omega_0 t)) dt \\ &= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{-i(n\omega_0 t)} dt \end{aligned} \quad (3.49)$$

$$\begin{aligned} c_{-n} &= \frac{2}{T} (a_n + ib_n) = \frac{2}{T} \left(\frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cos(n\omega_0 t) dt + i \frac{T}{2} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \sin(n\omega_0 t) dt \right) \\ &= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) (\cos(n\omega_0 t) + i \sin(n\omega_0 t)) dt \\ &= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) e^{i(n\omega_0 t)} dt \end{aligned} \quad (3.50)$$

Puesto que los subíndices de coeficientes y exponentes se encuentran en el rango de todo el conjunto de enteros no negativos... $-3, -2, -1, 0, 1, 2, 3, \dots$, podemos escribir los resultados de (3.47), (3.48), (3.49) y (3.50) de manera más compacta al sumar tanto enteros negativos como no negativos. En otras palabras, es posible utilizar *una* suma y *una* integral que defina todos los coeficientes c_0 , c_n y c_{-n} . [29]

3.2.9.1. Serie Compleja de Fourier de una función con período T

Si $f(t)$ es una función periódica de periodo T 3.15, se puede expresar mediante una serie exponencial compleja como: [26]

$$f(t) = c_0 + \sum_{\substack{n=-\infty \\ n \neq 0}}^{\infty} c_n e^{in\omega_0 t} \quad (3.51)$$

recordando que $\omega_0 = \frac{2\pi}{T}$ es la frecuencia angular fundamental y los coeficientes c_0 y c_n están dados por:

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-in\omega_0 t} dt \quad c_0 = \frac{1}{T} \int_{-T/2}^{T/2} f(t) dt \quad (3.52)$$

Se calcula el coeficiente c_0 por aparte del coeficiente general c_n ya que en varios casos, el coeficiente c_n presenta una indeterminación en $n = 0$, por lo tanto,

calculamos el coeficiente por aparte para evitar este problema.

Recordemos que otra forma en la que nos podemos encontrar la función $f(t)$ es que esté definida sobre un intervalo $[-p, p]$ 3.15, recordamos que la frecuencia fundamental toma la forma de $\omega_0 = \frac{\pi}{p}$ y los coeficientes ahora tienen la siguiente forma

$$c_n = \frac{1}{2p} \int_{-p}^p f(t) e^{-in\omega_0 t} dt \quad c_0 = \frac{1}{2p} \int_{-p}^p f(t) dt \quad (3.53)$$

3.2.9.2. Serie Compleja de Fourier de 0 a T

Así como con la serie trigonométrica, también se puede obtener una extensión de medio intervalo usando la serie exponencial compleja, con la diferencia de que solo podemos obtener la extensión de medio rango periódica 3.18 de una función definida en el intervalo 0 a T , ya que las extensiones par e impar se obtienen únicamente con componentes trigonométricos. Esta serie tiene la misma forma de (3.51), pero los coeficientes quedan de la siguiente forma [27]

$$c_n = \frac{1}{T} \int_0^T f(t) e^{-in\omega_0 t} dt \quad c_0 = \frac{1}{T} \int_0^T f(t) dt \quad (3.54)$$

3.2.10. Fenómeno de Gibbs

Cuando una función dada se aproxima mediante una serie de Fourier, ya sea trigonométrica o exponencial compleja, habrá un error considerable en la vecindad de la discontinuidad, no importa cuantos términos se quieran emplear. Este efecto se conoce como el fenómeno de Gibbs. Para ilustrar este fenómeno se puede ver en la Figura 3.19 el resultado de aproximar una onda cuadrada por una serie finita de Fourier [33]

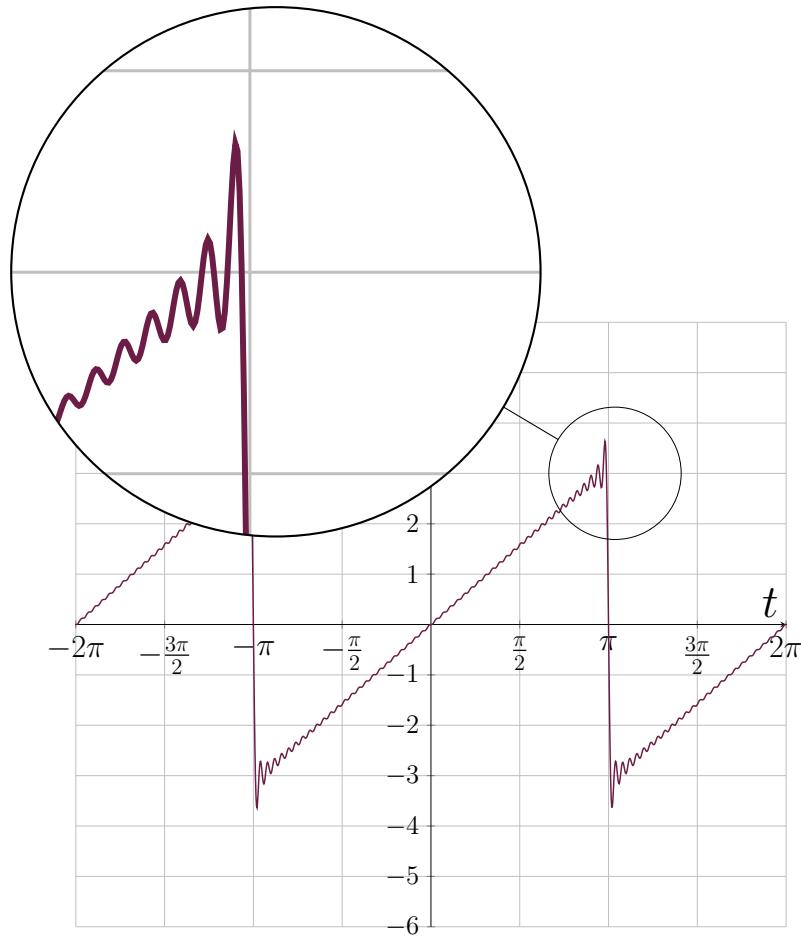


Figura 3.19: Aproximación de una función $f(t) = t$ con $-\pi < t < \pi$ en sus 50 primeros armónicos. *Fuente: Elaboración propia*

3.3. Aplicaciones Web

Una aplicación web es un software que se ejecuta desde un navegador web, como Google Chrome o Firefox. Estas aplicaciones no necesitan instalarse para poder ser utilizadas, ya que al ejecutarse sobre el navegador solo requiere de conexión a una red de internet, sino que se ejecutan en un servidor y se transmiten a través de Internet al navegador de nuestro dispositivo. Las aplicaciones en línea ofrecen numerosas ventajas, y prácticamente todas las grandes compañías las incorporan en sus propuestas para los usuarios [35]. Algunas de las ventajas de las aplicaciones web son;

- **Accesibilidad:** Las aplicaciones web pueden ser accedidas desde cualquier navegador y desde una variedad de dispositivos personales y empresariales. Grupos de trabajo en distintas ubicaciones pueden interactuar con documentos compartidos, sistemas de gestión de contenidos y otros servicios empresariales a través de aplicaciones web basadas en suscripciones.
- **Desarrollo eficiente:** El desarrollo de aplicaciones web es relativamente sencillo y rentable para las empresas, permitiendo a equipos pequeños realizar ciclos de desarrollo cortos. Esto hace que las aplicaciones web

sean una forma eficiente y asequible de desarrollar software. Además, dado que la misma versión de la aplicación funciona en todos los navegadores y dispositivos modernos, no es necesario crear múltiples versiones para diversas plataformas.

- **Simplicidad para el usuario:** Los usuarios no necesitan descargar las aplicaciones web, lo cual facilita su acceso sin ocupar espacio en el disco duro o requerir mantenimiento. Las aplicaciones web reciben actualizaciones de software y seguridad de forma automática, lo que garantiza que siempre estén actualizadas y reduce el riesgo de vulnerabilidades de seguridad.
- **Escalabilidad:** Las empresas que emplean aplicaciones web pueden añadir nuevos usuarios de manera flexible, sin necesidad de infraestructura adicional ni hardware costoso. Además, la mayoría de los datos de las aplicaciones web se almacenan en la nube, evitando así la inversión en capacidad de almacenamiento adicional para ejecutar las aplicaciones.

Las aplicaciones web están basadas en una arquitectura cliente-servidor, en la cual el código se divide en dos partes: scripts del lado del cliente y scripts del lado del servidor [35].



Figura 3.20: Diagrama de una arquitectura cliente - servidor. Fuente: [36]

- **Arquitectura del lado del cliente:** El script del lado del cliente se encarga de gestionar la funcionalidad de la interfaz de usuario, que incluye elementos como botones y menús desplegables. Cuando el usuario final accede a la aplicación web mediante un enlace, el navegador carga el script del lado del cliente y renderiza tanto los elementos visuales como el texto necesario para la interacción con el usuario. Por ejemplo, el usuario puede visualizar contenidos, reproducir videos o completar la información de un formulario de contacto. Acciones como hacer clic en el botón de enviar se envían al servidor como solicitudes del cliente.
- **Arquitectura del lado del servidor:** El script del lado del servidor se encarga del procesamiento de la información. Este servidor gestiona las solicitudes realizadas por el cliente y devuelve una respuesta. Estas solicitudes pueden incluir acciones como la obtención de datos adicionales, la edición de datos o el almacenamiento de nuevos datos. Por ejemplo, si el

usuario hace clic en un botón de “Leer más”, el servidor le envía contenido adicional. Si hace clic en “Enviar”, el servidor guarda los datos del usuario en una base de datos. En ciertos casos, el servidor completa la solicitud generando y enviando una página HTML completa al cliente, lo cual se denomina renderizado del lado del servidor.

3.3.1. Componentes de una aplicación web

3.3.1.1. HTML

HTML son las siglas en inglés de lenguaje de etiquetas de hipertexto (HyperText Markup Language) es el lenguaje estándar fundamental para estructurar y presentar contenido en la web, siendo la base sobre la cual se construyen todas las páginas web. Su función principal es organizar el contenido de manera semántica, lo que significa que emplea etiquetas específicas para definir la estructura de los distintos elementos, como encabezados, párrafos, listas, imágenes y enlaces, permitiendo que los navegadores interpreten y muestren adecuadamente el contenido al usuario. Esta estructuración semántica facilita la accesibilidad, ya que herramientas de asistencia, como los lectores de pantalla, pueden entender y transmitir mejor la información a personas con discapacidades visuales o cognitivas, y además optimiza el SEO (Search Engine Optimization), ayudando a los motores de búsqueda a indexar y clasificar el contenido de manera más precisa y efectiva. [37]

3.3.1.2. Canvas

El elemento <canvas> en HTML es una herramienta que permite dibujar gráficos, crear animaciones y renderizar imágenes en tiempo real a través de scripting. El elemento <canvas> es sólo un contenedor de gráficos. Es necesario usar JavaScript para crear los gráficos. Canvas cuenta con varios métodos para dibujar trazados, cuadros, círculos, texto así como agregar imágenes. Canvas también es compatible con todos los principales navegadores [38].

3.3.1.3. CSS

CSS son las siglas en inglés de hojas de estilo en cascada (Cascading Style Sheets) es básicamente un lenguaje que maneja el diseño y presentación de una aplicación web, es decir, cómo lucen cuando un usuario las visita. Funciona junto con el lenguaje HTML que se encarga del contenido básico de los sitios. Se les denomina hojas de estilo “en cascada” porque se puede tener varias y una de ellas con las propiedades heredadas (o en cascada) de otras. En general, una plantilla básica resulta suficiente para el funcionamiento de una aplicación web. Sin embargo, si se desea dar una apariencia llamativa para el usuario, se necesita implementar CSS y qeu ayudará a maximizar el impacto del contenido del sitio [39].

3.3.1.4. JavaScript

JavaScript es un lenguaje de programación que permite brindar interactividad a las aplicaciones web. Desde actualizar fuentes de redes sociales a mostrar

animaciones y mapas interactivos, las funciones de JavaScript pueden mejorar la experiencia del usuario de un sitio web. Como lenguaje de scripting del lado del servidor, se trata de una de las principales tecnologías de la World Wide Web. Por ejemplo, al navegar por Internet, en cualquier momento en el que vea un carrusel de imágenes, un menú desplegable “click-to-show” (clic para mostrar), o cambien de manera dinámica los elementos de color en una página web, estará viendo los efectos de JavaScript [40].

3.3.2. NodeJS

Node.js es un entorno de ejecución de un solo hilo, de código abierto y multiplataforma diseñado para desarrollar aplicaciones de red y del lado del servidor rápidas y escalables de JavaScript (de ahí su terminación en .js haciendo alusión al lenguaje JavaScript). Está escrito en C/C++ y Javascript y emplea una arquitectura de E/S basada en eventos y sin bloqueos, lo que la vuelve eficaz y apropiada para aplicaciones en tiempo real [41].

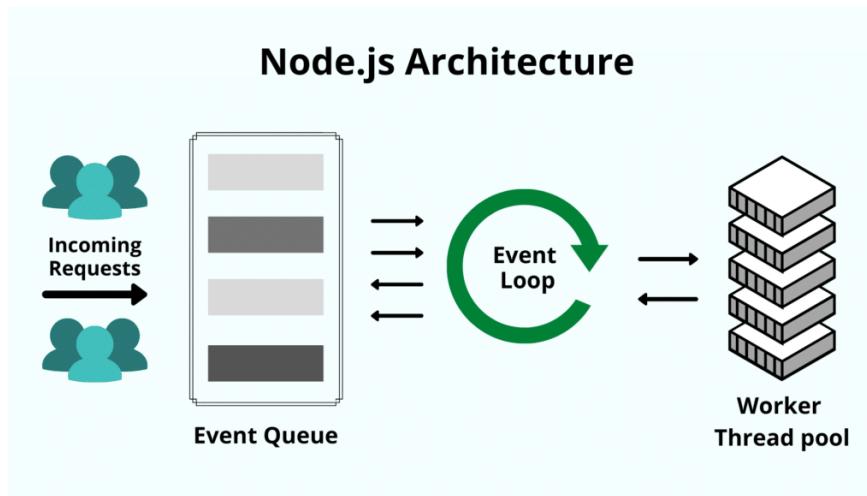


Figura 3.21: Cómo procesa node.js las peticiones entrantes utilizando el bucle de eventos. *Fuente:* [41]

Node.js utiliza una arquitectura de “Single Threaded Event Loop” 3.21 que maneja múltiples solicitudes mediante un único bucle de eventos y un pool de hilos limitado para operaciones de E/S bloqueantes. A diferencia del modelo multihilo de lenguajes como Java, donde cada solicitud se asigna a un hilo individual del pool, Node.js coloca las solicitudes entrantes en una cola y las procesa de manera no bloqueante, asignando hilos del pool de trabajadores solo cuando es necesario. Esta arquitectura reduce el consumo de recursos y memoria, permitiendo una ejecución más rápida de tareas ligeras y haciendo que Node.js sea ideal para aplicaciones en tiempo real. Sin embargo, para tareas que requieren un procesamiento intensivo de datos, los lenguajes multihilo como Java son más adecuados [41].

3.3.3. Maxima

Maxima es un Sistema de Álgebra Computacional (CAS, por sus siglas en inglés: Computer Algebra System) diseñado en Lisp, de código abierto diseñado para realizar manipulaciones simbólicas y numéricas de expresiones matemáticas. Originalmente desarrollado a partir de una versión de Macsyma del MIT en 1982, Maxima ha evolucionado gracias a la contribución de una comunidad de desarrolladores y usuarios que continúan mejorándolo y ampliando sus capacidades [9].

Se destaca por ser muy rápido y ligero, lo que permite realizar manipulaciones simbólicas y numéricas de manera eficiente, incluyendo simplificación, derivadas, integrales y resolución de ecuaciones algebraicas y diferenciales. Además, maneja operaciones avanzadas con matrices y vectores, genera gráficos bidimensionales y tridimensionales, y ofrece un lenguaje de programación propio para automatizar cálculos y crear scripts personalizados [9].

3.3.4. Angular

Angular es un Framework de JavaScript de código abierto escrito en TypeScript. Su objetivo principal es desarrollar aplicaciones de una sola página. Google se encarga del mantenimiento y constantes actualizaciones de mejoras para este framework. En el ámbito del desarrollo de software, un framework se define como una estructura de soporte tanto conceptual como tecnológica, compuesta generalmente por artefactos o módulos de software específicos. Estos componentes sirven como base para la organización y creación de aplicaciones. En otras palabras, un framework actúa como una plantilla o esquema fundamentado en tecnología que simplifica significativamente el trabajo. De este modo, se minimizan los posibles errores de programación. Así, un framework es un conjunto de herramientas y módulos reutilizables para distintos proyectos, lo que facilita el desarrollo en diversos aspectos, mejorando el tiempo de ejecución, el esfuerzo requerido y la organización general del proceso.

3.3.4.1. TypeScript

TypeScript es un superset de JavaScript que añade tipado estático y características avanzadas al lenguaje de programación. Desarrollado por Microsoft, TypeScript se compila a JavaScript, lo que permite utilizar sus funcionalidades en cualquier entorno que soporte JavaScript. Al incorporar tipos estáticos, TypeScript facilita la detección temprana de errores durante el desarrollo, mejora la legibilidad del código y facilita el mantenimiento de proyectos a gran escala. Además, ofrece soporte para programación orientada a objetos y otras mejoras que potencian la productividad y la robustez de las aplicaciones web. Gracias a estas ventajas, TypeScript se ha convertido en una herramienta popular en el desarrollo de aplicaciones modernas, complementando y extendiendo las capacidades de JavaScript [42].

3.3.4.2. Tailwind CSS

Tailwind CSS es un framework de CSS que permite a construir interfaces de usuario personalizadas de manera rápida y eficiente. A diferencia de los frameworks tradicionales que proporcionan componentes predefinidos, Tailwind ofrece clases de utilidad que pueden combinarse directamente en el HTML sin necesidad de escribir CSS personalizado. Esto facilita la creación de diseños responsivos y altamente personalizables, promoviendo la consistencia y reduciendo la cantidad de código CSS necesario. Además, Tailwind incluye características como el modo oscuro, diseños flexibles y un sistema de personalización extensivo que permite adaptar el framework a las necesidades específicas de cada proyecto [43].

3.3.5. API

Una API, por sus siglas en inglés de interfaz de programación de aplicaciones (application programming interface), es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar los sistemas de software de las aplicaciones. Al compartir únicamente la información necesaria y ocultar detalles internos, las API mejoran la seguridad del sistema [44].

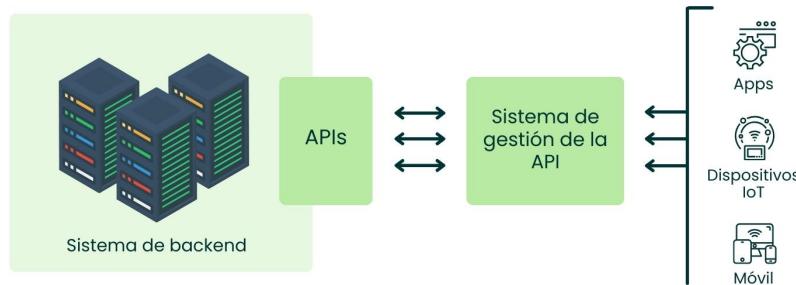


Figura 3.22: Arquitectura general de una API. Fuente: [44]

Suele considerarse como el contrato entre el usuario y el proveedor de información, en el cual se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta). Se puede considerar la API como el mediador entre los usuarios o clientes y los recursos o servicios web que quieren obtener. Con ellas, las empresas pueden compartir recursos e información mientras conservan la seguridad, el control y la autenticación, lo cual les permite definir los accesos de cada usuario [44].

3.3.5.1. Protocolo HTTP

El HTTP, que viene de las siglas en inglés de Protocolo de Transferencia de Hiper Texto (HyperText Transfer Protocol) es el protocolo esencial para la transmisión de información en la World Wide Web, facilitando la comunicación entre servidores, clientes y proxies mediante un lenguaje común. Establecido en 1999 por el World Wide Web Consortium y la Internet Engineering Task Force,

define las reglas de sintaxis y semántica para las peticiones y respuestas, operando principalmente a través de los puertos 80 y 8080. Al ser un protocolo sin estado, no guarda registro de interacciones anteriores, pero utiliza cookies para almacenar información de visitas previas en el cliente. Su funcionamiento se basa en un esquema de petición-respuesta, permitiendo una comunicación eficiente y flexible entre el usuario (como navegadores o rastreadores web) y los servidores web [45].

3.3.5.2. REST

REST significa Representational State Transfer, este no es un protocolo ni un estándar, sino un conjunto de restricciones vinculadas a la arquitectura. REST define un conjunto de funciones como GET, PUT, DELETE, etc. que los clientes pueden usar para acceder a los datos del servidor. Los clientes y servidores intercambian datos utilizando el protocolo HTTP.

La característica principal de las API REST es la falta de estado (statelessness). Esto significa que los servidores no guardan los datos del cliente entre las solicitudes. Las solicitudes del cliente al servidor son similares a las URL que escribes en tu navegador para visitar un sitio web. La respuesta del servidor es solo datos en formato plano, sin la representación gráfica típica de una página web [44].

3.3.6. Interfaz gráfica de usuario

Las Interfaces Gráficas de Usuario (GUI) son omnipresentes en nuestra vida cotidiana, ya sea al utilizar una computadora o un celular entre otros dispositivos. La eficacia de una GUI es crucial para determinar si un producto será competitivo o no. Un producto puede fracasar si el usuario no logra completar una acción, como una transacción económica, si no comprende la secuencia de pasos requeridos, si no encuentra fácilmente cómo realizar una acción necesaria, como hacer una compra, o si no encuentra atractivo el diseño de la aplicación. [46]

3.3.6.1. Principios de Usabilidad

Los principios de usabilidad web son la base de cualquier página web para que sea “user friendly”. O lo que es lo mismo, es un tipo de diseño centrado en el usuario para conseguir mejorar la experiencia del mismo. La usabilidad se refiere a la facilidad con la que los usuarios interactúan con una herramienta con el fin de lograr un propósito específico. Así pues, la usabilidad web indica hasta qué punto un sitio web resulta sencillo de utilizar. Esto significa que para que una aplicación web [47].

Los principios de usabilidad, son diez fundamentos que facilitan el diseño de productos más aceptados por los usuarios al enfocarse en sus necesidades y comportamientos [47]:

1. **Visibilidad del estado del sistema.** El usuario siempre debe de estar informado de lo que está pasando en la aplicación web y ofrecerle una respuesta en el menor tiempo posible. Ejemplos de esto son las barras de carga o notificaciones sobre el navegador.

2. **Relación entre el sistema y el mundo real.** El sistema tiene que “hablar” el lenguaje del usuario usando palabras, frases inclusive símbolos con los que el esté familiarizado y que pueda reconocer con facilidad. Para esto la información debe mostrarse en un orden lógico y usando las palabras y símbolos correctos, sin darle oportunidad al usuario de equivocarse.
3. **Control y libertad del usuario.** Los usuarios pueden equivocarse al realizar una acción dentro de la aplicación, así que este debe tener la oportunidad de corregir su error y no sentirse frustrado por ello. Claro ejemplo de esto es un botón de deshacer.
4. **Consistencia y estándares.** Las aplicaciones deben de seguir los estándares y convenios establecidos para iconos o colores, por ejemplo un ícono con líneas verticales indica un menú, o el color verde y rojo se asocian con aceptar y cancelar, respectivamente.
5. **Prevención de errores.** La aplicación debe de prevenir cualquier posible error que el usuario pueda cometer, y en caso de que este cometa uno, debe de disponer de diversas herramientas para corregirlo, limitar caracteres o entradas no validas por el sistema es un ejemplo de esto.
6. **Reconocer antes que recordar.** La aplicación debe ayudar al usuario a no tener que memorizar acciones u objetos para que pueda usarla fácilmente, usar botones asignados y personalizados para acciones específicas es un ejemplo de esto.
7. **Flexibilidad y eficiencia de uso.** La aplicación debe de estar preparada para recibir a todo tipo de usuarios, si alguien inexperto en el sitio o en el tema del mismo se pueden brindar ayudas como ventanas con mensajes o tutoriales.
8. **Diseño estético y minimalista.** La aplicación no debe de tener información innecesaria que distraiga al usuario y perturbe su experiencia al usarla.
9. **Ayudar a los usuarios a reconocer, diagnosticar y corregir los errores.** En caso de que se produzca un error o excepción dentro de la aplicación, los mensajes de error deben ser comprensibles para el usuario, como pasar de un ERROR 404 a un mensaje más entendible como ERROR DEL SERVIDOR.
10. **Ayuda y documentación.** Es preciso que la aplicación cuente con un manual de funcionamiento, esto ayuda al usuario a que le sea más fácil usar la aplicación. Este manual debe ser fácil de localizar, definir los pasos claramente y no de manera extensa.

El cumplir con estos principios de usabilidad ayudan a que el sitio cuente con un tráfico más recurrente, es decir, aumentamos las posibilidades de que un usuario, después de utilizar la aplicación, la vuelva a usar en un futuro, así como que más personas la usen, demás de que se disminuye el porcentaje de rebote, que no es otra cosa que conseguir que el tiempo de estancia del usuario en la aplicación sea más alto y que explore más partes del sitio.

3.3.7. Seguridad en Aplicaciones Web

La seguridad en aplicaciones web es fundamental para proteger la integridad, confidencialidad y disponibilidad de los datos manejados, así como para garantizar la confianza de los usuarios. Algunas medidas que se implementan en una aplicación web para implementar dicha seguridad son las siguientes.

3.3.7.1. Protocolo HTTPS

El protocolo de transferencia de hipertexto seguro, por sus siglas en inglés, (HyperText Transfer Protocol Secure) es una versión segura del protocolo HTTP que utiliza el SSL/TLS para cifrado y autenticación. HTTPS está especificado por RFC 2818 [48] y utiliza el puerto 443 o el 8443 de forma predeterminada en lugar del puerto 80/8080 de HTTP [49].

El protocolo HTTPS hace posible que los usuarios del sitio web transmitan datos confidenciales como números de tarjetas de crédito, información bancaria y credenciales de inicio de sesión de forma segura a través de Internet. Por esta razón, HTTPS es especialmente importante para asegurar actividades en línea como compras, banca y trabajo remoto. Sin embargo, HTTPS se está convirtiendo rápidamente en el protocolo estándar para cualquier sitio web, ya sea que este intercambie o no datos confidenciales con los usuarios [49].

HTTPS se diferencia de HTTP al agregar cifrado, autenticación e integridad al protocolo original. Mediante SSL/TLS, HTTPS cifra los datos para protegerlos de interceptaciones y ataques de intermediarios, utilizando criptografía de clave pública para establecer una conexión segura entre el servidor y el navegador. Además, HTTPS autentica la identidad del servidor mediante certificados digitales emitidos por autoridades de certificación confiables, garantizando que los documentos provienen de una fuente legítima. También asegura la integridad de los datos mediante firmas digitales que verifican que el contenido no ha sido alterado durante la transmisión. Estas mejoras hacen que HTTPS sea un protocolo mucho más seguro para navegar y realizar transacciones en la web en comparación con HTTP [49].

3.3.7.2. Certificado SSL/TLS

El certificado SSL (Secure Sockets Layer) es una pieza clave para establecer una conexión segura entre el servidor y el cliente. Este establece un enlace cifrado entre un servidor y un cliente. Esto permite que información confidencial, como los datos de la tarjeta de crédito, se transmita de forma segura a través de Internet. El certificado contiene una clave pública que autentica la identidad del sitio web y permite la transferencia de datos cifrados mediante criptografía asimétrica o de clave pública. La clave privada correspondiente se mantiene en secreto en el servidor [50].

CAPÍTULO 4

Análisis y Diseño del Sistema

En este capítulo se evaluaron los datos del marco teórico para seleccionar los componentes más adecuados para el proyecto. También se realizó un análisis detallado de los requerimientos y riesgos asociados al desarrollo del sistema, asegurando un diseño eficiente y viable para su implementación.

4.1. Metodología de Desarrollo

La metodología a usar La metodología propuesta para el desarrollo del proyecto es Prototipos Evolutivos, en la cual se emplearán dos iteraciones mínimas. Esta metodología es adecuada para proyectos en los que la comprensión completa de los requisitos puede surgir gradualmente y en donde es importante recibir retroalimentación continua [51]. Al utilizar prototipos, se permite una aproximación más ágil al desarrollo, ya que se pueden generar soluciones tempranas que serán ajustadas en función de los comentarios de los usuarios y evaluaciones constantes.

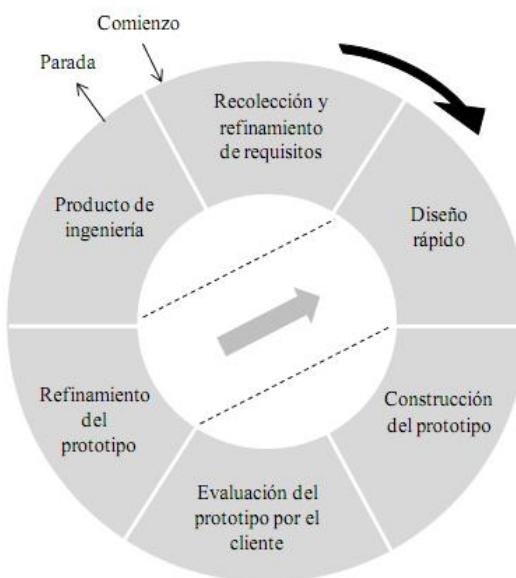


Figura 4.1: tapas de la metodología por prototipos. *Fuente: [51]*

El uso de prototipos evolutivos permite abordar los requerimientos del sistema de manera iterativa. Cada prototipo no es desecharlo sino que se refina

progresivamente hasta obtener una versión completa del sistema. Esto asegura que el producto final no solo cumpla con los requisitos establecidos inicialmente, sino también con las expectativas del cliente o usuario.

La retroalimentación temprana que se recibe desde las primeras etapas del proyecto es invaluable para identificar mejoras y posibles problemas antes de que se conviertan en obstáculos mayores. Esto permite un ahorro significativo de tiempo y recursos, ya que se pueden realizar correcciones antes de que los problemas se propaguen en el ciclo de desarrollo.

A continuación, se muestra una tabla comparativa de ventajas/desventajas de utilizar esta metodología.

Ventajas	Desventajas
Permite obtener retroalimentación temprana y frecuente del cliente, lo que ayuda a ajustar los requerimientos de manera rápida.	Puede generar una dependencia excesiva de los prototipos, llevando a malentendidos sobre la funcionalidad final del producto.
Facilita la detección temprana de errores o problemas de diseño, lo que reduce costos de desarrollo en fases posteriores.	Requiere recursos adicionales (tiempo y esfuerzo) para desarrollar y revisar los prototipos.
Mejora la comprensión de los requisitos por parte del equipo de desarrollo y el cliente, reduciendo malentendidos.	Si no se maneja adecuadamente, puede desviarse el proyecto del alcance original, añadiendo características no planificadas (creeping scope).
Aumenta la satisfacción del cliente al ver un producto tangible en las primeras fases del desarrollo.	El prototipo puede ser confundido con el producto final, lo que puede generar expectativas irreales respecto a tiempos y calidad.
Fomenta la iteración y mejora continua del producto, asegurando que se ajuste mejor a las necesidades reales del cliente.	El uso excesivo de prototipos puede ralentizar el desarrollo del producto final si las iteraciones no están bien gestionadas.

Tabla 4.1: Ventajas/Desventajas de la metodología de Prototipos Evolutivos

4.1.1. ¿Por qué prototipos para este trabajo?

Está perrona

4.2. Análisis de Requerimientos

Los requerimientos funcionales y no funcionales descritos en este documento son fundamentales para el desarrollo y éxito de la aplicación web de cálculo

y visualización de Series de Fourier. Estos requerimientos definen no solo las funcionalidades que la aplicación debe cumplir, sino también las expectativas de rendimiento, seguridad y usabilidad. A través de su análisis, se asegura que el sistema propuesto sea capaz de resolver la problemática planteada de manera eficaz, proporcionando a los usuarios una herramienta confiable y eficiente para el análisis de series matemáticas.

4.2.1. Requerimientos Funcionales

Los requerimientos funcionales de un sistema describen lo que el sistema debe hacer. Describen las interacciones entre el sistema y su ambiente, en forma independiente a su implementación. El ambiente incluye al usuario y cualquier otro sistema externo con el cual interactúa el sistema.

Requerimiento	Ingreso de Funciones Matemáticas
Identificador	RF1
Prioridad de desarrollo	Alta
Entrada	Funciones matemáticas ingresadas por el usuario usando MathQuill
Salida	Funciones ingresadas en formato compatible con Maxima
Descripción	El usuario debe poder ingresar funciones matemáticas definidas en un solo trozo o en múltiples trozos usando una notación matemática intuitiva. Debe soportar funciones trigonométricas, polinomios, exponenciales y cualquier otra función que Maxima pueda interpretar.
Precondición	El usuario debe estar en el área de ingreso de funciones.
Postcondición	Las funciones son ingresadas correctamente y listas para su procesamiento.

Tabla 4.2: Requerimiento funcional No. 1

Requerimiento	Selección de Parámetros
Identificador	RF2
Prioridad de desarrollo	Alta
Entrada	Parámetros seleccionados por el usuario (tipo de serie, período T)

Salida	Parámetros listos para el cálculo de coeficientes
Descripción	Los usuarios deben poder seleccionar entre las diferentes representaciones de Series de Fourier y especificar el período T de la función, ya sea de $-T/2$ a $T/2$ o de 0 a T.
Precondición	El usuario debe haber ingresado una función matemática y estar en el área de selección de parámetros.
Postcondición	Los parámetros seleccionados están listos para el cálculo de coeficientes.

Tabla 4.3: Requerimiento funcional No. 2

Requerimiento	Cálculo de Coeficientes
Identificador	RF3
Prioridad de desarrollo	Alta
Entrada	Función matemática y parámetros seleccionados por el usuario
Salida	Coeficientes de la serie de Fourier calculados
Descripción	La aplicación debe calcular automáticamente los coeficientes de la serie de Fourier según la función ingresada y los parámetros especificados, enviando la información a Maxima y recibiendo los resultados.
Precondición	El usuario debe haber ingresado una función y seleccionado los parámetros necesarios.
Postcondición	Los coeficientes de la serie de Fourier se calculan y se devuelven al sistema.

Tabla 4.4: Requerimiento funcional No. 3

Requerimiento	Interactividad en la Gráfica
Identificador	RF4
Prioridad de desarrollo	Media
Entrada	Control del usuario sobre la gráfica (acercar, alejar, cantidad de términos n)

Salida	Gráfica interactiva modificada según la selección del usuario
Descripción	El usuario debe poder acercar, alejar y desplazar la gráfica, así como seleccionar la cantidad de términos n de la suma parcial a visualizar.
Precondición	El usuario debe haber ingresado la función y los parámetros.
Postcondición	La gráfica es ajustada interactivamente según la entrada del usuario.

Tabla 4.5: Requerimiento funcional No. 4

Requerimiento	Visualización de las Gráficas
Identificador	RF5
Prioridad de desarrollo	Alta
Entrada	Función original y términos de la Serie de Fourier
Salida	Gráficas de la función original y las sumas parciales de la Serie de Fourier
Descripción	El usuario debe poder visualizar la gráfica de la función original y las sumas parciales de la Serie de Fourier en un lienzo canvas, que se actualizará en tiempo real a medida que se agregan o quitan términos.
Precondición	El usuario debe haber ingresado la función y los parámetros necesarios.
Postcondición	La gráfica se actualiza en tiempo real según los términos seleccionados.

Tabla 4.6: Requerimiento funcional No. 5

Requerimiento	Visualización de resultados
Identificador	RF6
Prioridad de desarrollo	Alta
Entrada	Función matemática y parámetros seleccionados

Salida	Expresión matemática de la serie de Fourier y coeficientes individuales
Descripción	La aplicación debe mostrar la serie de Fourier y los valores individuales de los coeficientes en notación matemática, utilizando MathQuill.
Precondición	El usuario debe haber ingresado la función y los parámetros necesarios.
Postcondición	Se muestran la serie de Fourier y los coeficientes en notación matemática.

Tabla 4.7: Requerimiento funcional No. 6

Requerimiento	Manejo de Errores
Identificador	RF7
Prioridad de desarrollo	Media
Entrada	Función ingresada por el usuario, parámetros seleccionados
Salida	Mensajes de error claros en caso de problemas con la función o los parámetros
Descripción	La aplicación debe detectar errores en la entrada de funciones o parámetros no válidos y mostrar mensajes de error claros. También debe notificar si las integrales no convergen o si Maxima no puede procesar la función.
Precondición	El usuario ha ingresado una función o parámetros y ha intentado procesar los datos.
Postcondición	Se muestra un mensaje de error claro si ocurre algún problema.

Tabla 4.8: Requerimiento funcional No. 7

Requerimiento	Optimización de Resultados
Identificador	RF8
Prioridad de desarrollo	Baja
Entrada	Parámetros y resultados de coeficientes previos

Salida	Optimización de cálculo y presentación de resultados mejorada
Descripción	La aplicación debe optimizar el cálculo de los coeficientes para mejorar el rendimiento y la experiencia del usuario, manejando grandes cantidades de términos de manera eficiente.
Precondición	El usuario ha ingresado parámetros complejos o grandes cantidades de términos.
Postcondición	Los resultados optimizados se calculan y presentan rápidamente.

Tabla 4.9: Requerimiento funcional No. 8

4.2.2. Requerimientos No Funcionales

Los requerimientos no funcionales describen una restricción sobre el sistema que limita nuestras elecciones en la construcción de una solución al problema. Describen atributos sólo del sistema o del ambiente del sistema que no están relacionados directamente con los requisitos funcionales. Los requisitos no funcionales incluyen restricciones cuantitativas, tipo de plataforma.

Nombre	Descripción
Rendimiento y Escalabilidad	El sistema debe calcular los coeficientes de Fourier en un tiempo razonable, incluso para funciones con múltiples trozos o con un alto número de términos en la serie. La visualización gráfica debe ser fluida y responsive, incluso al agregar o quitar muchos términos de la serie. Las interacciones en el lienzo canvas deben realizarse sin latencia perceptible.
Interfaz de Usuario Amigable e Intuitiva	La interfaz debe ser intuitiva y fácil de usar, con botones y controles claramente identificados. Los mensajes de error y advertencia deben ser comprensibles, evitando el uso de jerga técnica complicada. La notación matemática debe ser clara y precisa, usando MathQuill para entradas y visualización de resultados.
Manejo de la Concurrencia	La aplicación debe soportar múltiples usuarios concurrentes enviando cálculos al backend sin generar bloqueos ni conflictos en las llamadas a Maxima.

Portabilidad Compatibilidad	y	La aplicación debe ser compatible con los principales navegadores modernos (Chrome, Firefox, Edge). Aunque no es recomendable usar la aplicación en dispositivos móviles, el diseño debe ser responsive para ajustarse a diferentes tamaños de pantalla, incluidos dispositivos móviles y tabletas.
Modularidad Extensibilidad Código	y del	El frontend debe estar diseñado para facilitar la migración a frameworks como Angular en el futuro. El código debe ser modular y seguir buenas prácticas de desarrollo para facilitar la adición de nuevas características. Aunque el sistema no requiere una base de datos en su versión actual, el backend debe ser fácilmente extensible para permitir el almacenamiento de sesiones o configuraciones del usuario en el futuro.
Seguridad Aplicación	de la	Las peticiones a la API deben estar protegidas contra inyecciones de código. Implementar seguridad básica en el backend para evitar solicitudes maliciosas o repetitivas que puedan afectar el rendimiento del servicio.
Mantenibilidad Código	de	La aplicación debe tener una guía de usuario clara para explicar cómo ingresar funciones, seleccionar series y usar la interfaz. El código debe estar bien documentado para que futuros desarrolladores puedan entender la lógica y estructura de la aplicación fácilmente.

Tabla 4.10: Tabla de requerimientos no funcionales

4.3. Análisis de Riesgos

4.3.1. Identificación de Riesgos

4.3.2. Jerarquización de Riesgos

4.4. Estimación de Costos

4.4.1. Método de Estimación de Costos

4.4.2. Detalle de los Costos Estimados

4.5. Diseño del Sistema

4.5.1. Arquitectura General del Sistema

4.5.1.1. Arquitectura Cliente-Servidor

4.5.1.2. Flujo de Datos

4.5.2. Diseño de la Interfaz de Usuario

4.5.2.1. Estructura de la Interfaz

4.5.2.2. Prototipo de la Interfaz

CAPÍTULO 5

Implementación (Primer Prototipo)

En este capítulo vamos a juntar todo lo primero

APÉNDICE A

Cálculos

Dada la función $f(x) = x$, vamos a calcular los coeficientes correspondientes a su serie de Fourier en el intervalo $[-\pi, \pi]$.

A.1. Forma Trigonométrica

La serie trigonométrica de Fourier para una función $f(x)$ está dada por la siguiente expresión:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\omega_0 x) + b_n \sin(n\omega_0 x))$$

Donde los coeficientes a_0 , a_n y b_n se definen como:

$$\begin{aligned} a_0 &= \frac{2}{T} \int_{-T/2}^{T/2} f(x) dx \\ a_n &= \frac{2}{T} \int_{-T/2}^{T/2} f(x) \cos(n\omega_0 x) dx \\ b_n &= \frac{2}{T} \int_{-T/2}^{T/2} f(x) \sin(n\omega_0 x) dx \end{aligned}$$

Donde $\omega_0 = \frac{2\pi}{T}$, y en este caso, $T = 2\pi$, por lo que $\omega_0 = 1$.

Calculamos cada uno de los coeficientes para $f(x) = x$:

- El coeficiente a_0 es:

$$a_0 = \frac{2}{2\pi} \int_{-\pi}^{\pi} x dx = \frac{1}{\pi} \left[\frac{x^2}{2} \right]_{-\pi}^{\pi} = \frac{1}{\pi} \left(\frac{\pi^2}{2} - \frac{(-\pi)^2}{2} \right) = 0$$

Por lo tanto, $a_0 = 0$.

- El coeficiente a_n es:

Dado que $x \cos(nx)$ es una función impar, ya que x es impar y $\cos(nx)$ es par, su integral en un intervalo simétrico es cero:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} x \cos(nx) dx = 0$$

- El coeficiente b_n es:

Dado que $x \sin(nx)$ es una función par (producto de dos funciones impares), podemos calcular:

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} x \sin(nx) dx = \frac{2}{\pi} \int_0^{\pi} x \sin(nx) dx$$

Aplicamos integración por partes, tomando $u = x$ y $dv = \sin(nx) dx$, lo que nos da $du = dx$ y $v = -\frac{1}{n} \cos(nx)$. Entonces:

$$\int_0^{\pi} x \sin(nx) dx = -\frac{x}{n} \cos(nx) \Big|_0^{\pi} + \frac{1}{n} \int_0^{\pi} \cos(nx) dx$$

Evaluamos el primer término:

$$-\frac{\pi}{n} \cos(n\pi) + 0 = -\frac{\pi}{n}(-1)^n$$

La integral restante es:

$$\frac{1}{n} \int_0^{\pi} \cos(nx) dx = \frac{1}{n} \left[\frac{\sin(nx)}{n} \right]_0^{\pi} = \frac{1}{n^2}(0 - 0) = 0$$

Por lo tanto, el coeficiente b_n es:

$$b_n = \frac{2}{\pi} \left(-\frac{\pi}{n}(-1)^n \right) = \frac{2(-1)^{n+1}}{n}$$

Entonces, la serie trigonométrica de Fourier para $f(x) = x$ es:

$$f(x) = 2 \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} \sin(nx)$$

A.2. Forma Exponencial Compleja

La serie exponencial compleja de Fourier está dada por la siguiente expresión:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\omega_0 x}$$

Donde los coeficientes c_n se definen como:

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(x) e^{-in\omega_0 x} dx$$

Calculamos el coeficiente c_n para $n \neq 0$:

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} x e^{-inx} dx$$

Aplicamos integración por partes con $u = x$ y $dv = e^{-inx} dx$, obteniendo $du = dx$ y $v = \frac{e^{-inx}}{-in}$. Entonces:

$$c_n = \frac{1}{2\pi} \left(\left[x \frac{e^{-inx}}{-in} \right]_{-\pi}^{\pi} - \int_{-\pi}^{\pi} \frac{e^{-inx}}{-in} dx \right)$$

El primer término es:

$$\left[x \frac{e^{-inx}}{-in} \right]_{-\pi}^{\pi} = \frac{\pi e^{-in\pi} - (-\pi)e^{in\pi}}{-in} = \frac{2\pi(-1)^n}{-in}$$

El segundo término es cero, ya que:

$$\int_{-\pi}^{\pi} e^{-inx} dx = 0$$

Por lo tanto:

$$c_n = \frac{1}{2\pi} \left(\frac{2\pi(-1)^n}{-in} \right) = \frac{i(-1)^n}{n}$$

Para $n = 0$:

$$c_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} x dx = 0$$

Así que la serie exponencial compleja de Fourier para $f(x) = x$ es:

$$f(x) = \sum_{\substack{n=-\infty \\ n \neq 0}}^{\infty} \frac{i(-1)^n}{n} e^{inx}$$

Ambas representaciones son equivalentes y proporcionan la expansión de Fourier correcta para la función dada.

APÉNDICE B

Códigos

B.1. Código en Matlab para graficar la serie de Fourier trigonométrica

```
1      % Número de términos en la serie de Fourier
2      N = 10; % Puedes modificar N para mejorar la
            %>   aproximación
3
4      % Vector de tiempo de -pi a pi
5      t = linspace(-pi, pi, 1000);
6
7      % Función original x(t) = t
8      x_original = t;
9
10     % Inicialización de la aproximación de la serie
            %>   de Fourier
11     x_aprox = zeros(size(t));
12
13     % Cálculo de la serie de Fourier en forma
            %>   trigonométrica
14     for n = 1:N
15         bn = (2 * (-1)^(n+1)) / n; % Coeficientes de seno
16         x_aprox = x_aprox + bn * sin(n * t);
17     end
18
19     % Gráfica de la función original y su
            %>   aproximación
20     figure;
21     plot(t, x_original, 'k', 'LineWidth', 1.5); %
            %>   Función original en negro
22     hold on;
23     plot(t, x_aprox, 'b', 'LineWidth', 1.5); %
            %>   Aproximación en azul
24     legend('Función original x(t)', ['Aproximación
            %>   (Forma Trigonométrica) con N = ', num2str(N),
            %>   ' términos']);
```

```

25 xlabel('t');
26 ylabel('x(t)');
27 title('Serie de Fourier (Forma Trigonométrica) de
28 → x(t) = t');
29 grid on;
30 hold off;

```

Código 1: Código en Matlab para graficar la serie de Fourier trigonométrica de A.1. *Fuente: Elaboración propia*

B.2. Código en Matlab para graficar la serie de Fourier compleja

```

1 % Número de términos positivos y negativos en la
2 → serie de Fourier
3 N = 10; % Puedes modificar N para mejorar la
4 → aproximación
5
6 % Vector de tiempo de -pi a pi
7 t = linspace(-pi, pi, 1000);
8
9 % Función original x(t) = t
10 x_original = t;
11
12 % Inicialización de la aproximación de la serie
13 → de Fourier
14 x_aprox = zeros(size(t));
15
16 % Cálculo de la serie de Fourier en forma
17 → exponencial compleja
18 for n = -N:N
19 if n == 0
20 continue; % Saltar n = 0 para evitar división por
21 → cero
22 end
23 cn = (1i / n) * (-1)^n;
24 x_aprox = x_aprox + cn * exp(1i * n * t);
25 end
26
27 % Tomar la parte real de la aproximación (la
28 → función original es real)
29 x_aprox_real = real(x_aprox);
30
31 % Gráfica de la función original y su
32 → aproximación
33 figure;

```

```

27      plot(t, x_original, 'k', 'LineWidth', 1.5); %
28      ↳ Función original en negro
29      hold on;
30      plot(t, x_aprox_real, 'b', 'LineWidth', 1.5); %
31      ↳ Aproximación en azul
32      legend('Función original x(t)', ['Aproximación
33      ↳ (Forma Exponencial) con N = ', num2str(N), '
34      ↳ términos']);
35      xlabel('t');
36      ylabel('x(t)');
37      title('Serie de Fourier (Forma Exponencial
38      ↳ Compleja) de x(t) = t');
39      grid on;
40      hold off;

```

Código 2: Código en Matlab para graficar la serie de Fourier trigonométrica de A.2. Fuente: *Elaboración propia*

B.3. Código en Maple para la serie de Fourier trigonométrica

```

1      # Declaramos n como entero
2      assume(n, integer);
3
4      # Definimos la función
5      func := x;
6
7      # Periodo de la serie de Fourier
8      T := 2*Pi;
9
10     # Núcleos de las series de Fourier
11     series_cosine_core := cos(n*Pi*x/(T/2));
12     series_sine_core := sin(n*Pi*x/(T/2));
13
14     # Coeficientes de Fourier
15     a0 := (1/(T/2)) * int(func, x = -T/2..T/2);
16     an := (1/(T/2)) * int(func * series_cosine_core, x
17     ↳ = -T/2..T/2);
18     bn := (1/(T/2)) * int(func * series_sine_core, x =
19     ↳ = -T/2..T/2);
20
21     # Simplificamos los coeficientes
22     a0_simp := simplify(a0);
23     an_simp := simplify(an);
24     bn_simp := simplify(bn);

```

```

24      # Factorizamos los coeficientes
25      Coeff_A0 := factor(a0_simp);
26      Coeff_An := factor(an_simp);
27      Coeff_Bn := factor(bn_simp);

28
29      # Definimos el rango de n positivo y negativo
30      n1 := 1;
31      n2 := 5;

32
33      # Creamos la lista de An
34      lista_An := [seq(simplify(subs(n = i, Coeff_An *
35                                ↵ series_cosine_core)), i = n1..n2)];

36
37      # Creamos la lista de Bn
38      lista_Bn := [seq(simplify(subs(n = i, Coeff_Bn *
39                                ↵ series_sine_core)), i = n1..n2)];

40
41      # Sumamos los coeficientes An y Bn
42      lista_completa := lista_An, lista_Bn;

43      # Crear la serie final añadiendo A0 al principio
44      ↵ de la lista completa
45      serie_final := [Coeff_A0/2, lista_completa];
46      serie_final := simplify(lista_completa);

47      # Factorizamos la serie final
48      serie_factor := factor(serie_final);

49      # Suma de todos los coeficientes en una sola
50      ↵ variable
51      serie_funcion := Coeff_A0/2 + add(lista_An[i], i =
52      ↵ 1 .. nops(lista_An)) + add(lista_Bn[i], i = 1
53      ↵ .. nops(lista_Bn));

54
55      # Simplificamos la expresión para obtener la
56      ↵ forma más compacta
57      serie_funcion_simplificada :=
58      ↵ simplify(serie_funcion);

59
60      # Graficamos la función y la serie aproximada
61      plot([func, serie_funcion_simplificada], x =
62      ↵ -T/2..T/2);

```

Código 3: Código en Maple para calcular y graficar la serie de Fourier trigonométrica de A.1. Fuente: *Elaboración propia*

B.4. Código en Maple para la serie de Fourier Compleja

```

1          # Declaramos n como entero
2          assume(n, integer);

3
4          # Definimos la función
5          func := x;

6
7          # Periodo de la serie de Fourier
8          T := 2*Pi;

9
10         # Núcleos de las series de Fourier
11         series_cosine_core := cos(n*Pi*x/(T/2));
12         series_sine_core := sin(n*Pi*x/(T/2));

13
14         # Coeficientes de Fourier
15         a0 := (1/(T/2)) * int(func, x = -T/2..T/2);
16         an := (1/(T/2)) * int(func * series_cosine_core, x
17           = -T/2..T/2);
18         bn := (1/(T/2)) * int(func * series_sine_core, x =
19           = -T/2..T/2);

20         # Simplificamos los coeficientes
21         a0_simp := simplify(a0);
22         an_simp := simplify(an);
23         bn_simp := simplify(bn);

24         # Factorizamos los coeficientes
25         Coeff_A0 := factor(a0_simp);
26         Coeff_An := factor(an_simp);
27         Coeff_Bn := factor(bn_simp);

28
29         # Definimos el rango de n positivo y negativo
30         n1 := 1;
31         n2 := 5;

32
33         # Creamos la lista de An
34         lista_An := [seq(simplify(subs(n = i, Coeff_An *
35           series_cosine_core)), i = n1..n2)];

36         # Creamos la lista de Bn
37         lista_Bn := [seq(simplify(subs(n = i, Coeff_Bn *
38           series_sine_core)), i = n1..n2)];

39         # Sumamos los coeficientes An y Bn
40         lista_completa := lista_An, lista_Bn;

```

```

41
42      # Crear la serie final añadiendo A0 al principio
        # de la lista completa
43      serie_final := [Coeff_A0/2, lista_completa];
44      serie_final := simplify(lista_completa);

45
46      # Factorizamos la serie final
47      serie_factor := factor(serie_final);

48
49      # Suma de todos los coeficientes en una sola
        # variable
50      serie_funcion := Coeff_A0/2 + add(lista_An[i], i =
        # 1 .. nops(lista_An)) + add(lista_Bn[i], i = 1
        # .. nops(lista_Bn));

51
52      # Simplificamos la expresión para obtener la
        # forma más compacta
53      serie_funcion_simplificada :=
        # simplify(serie_funcion);

54
55      # Graficamos la función y la serie aproximada
56      plot([func, serie_funcion_simplificada], x =
        # -T/2..T/2);

```

Código 4: Código en Maple para calcular y graficar la serie de Fourier compleja de A.2. *Fuente: Elaboración propia*

B.5. Código en Maxima para la serie de Fourier trigonométrica

```

1      declare(n, integer);
2      func: x;
3      /*func: ((3*x**3)-2*x+3);*/
4      T: 2*%pi;
5      series_cosine_core: cos((n*%pi*x)/((T/2)));
6      series_sine_core: sin((n*%pi*x)/((T/2)));

7
8      a0: (1/(T/2)) * integrate((func), x ,-(T/2),
        # (T/2));
9      an: (1/(T/2)) * integrate((func) *
        # series_cosine_core, x ,-(T/2), (T/2));
10     bn: (1/(T/2)) * integrate((func) *
        # series_sine_core, x ,-(T/2), (T/2));

11    a0_simp: ratsimp(a0);
12    an_simp: ratsimp(an);

```

```

14      bn_simp: ratsimp(bn);
15
16      Coeff_A0: factor(a0_simp);
17      Coeff_An: factor(an_simp);
18      Coeff_Bn: factor(bn_simp);
19
20      /* Definimos el rango de n positivo y negativo */
21      n1 : 1;
22      n2 : 10;
23
24      /* Creamos la lista de An */
25      lista_An : makelist(subst(n=i, Coeff_An *
26                                ↳ series_cosine_core), i, n1, n2);
27
28      /* Creamos la lista de Bn */
29      lista_Bn : makelist(subst(n=i, Coeff_Bn *
30                                ↳ series_sine_core), i, n1, n2);
31
32      /* Sumamos los coeficientes An + Bn */
33      lista_completa : lista_An + lista_Bn;
34
35      /* Crear la serie final añadiendo A0 al principio
36      ↳ de la lista completa */
37      /*serie_final: cons(Coeff_A0/2,
38                        ↳ lista_completa);*/
39
40      serie_final: ratsimp(lista_completa);
41
42      serie_factor: factor(serie_final);
43
44      /* Suma de todos los coeficientes en una sola
45      ↳ variable */
46      serie_funcion : Coeff_A0/2 + sum(lista_An[i], i,
47                                         1, length(lista_An)) + sum(lista_Bn[i], i, 1,
48                                         length(lista_Bn));
49
49      /* Simplificamos la expresión para obtener la
49      ↳ forma más compacta */
50      serie_funcion_simplificada :
51          ratsimp(serie_funcion);
52
53      /* Graficamos la expresión */
54      plot2d([func, serie_funcion_simplificada], [x,
55             -T/2, T/2]);
56
57      kill(all);

```

Código 5: Código en Maxima para calcular y graficar la serie de Fourier trigonométrica de A.1. *Fuente: Elaboración propia*

B.6. Código en Maxima para la serie de Fourier compleja

```

1      declare(n, integer);
2      func: x;
3      /*func: ((3*x**3)-2*x+3);*/
4      T: 2*%pi;
5      series_cosine_core: cos((n*%pi*x)/((T/2)));
6      series_sine_core: sin((n*%pi*x)/((T/2)));
7
8      a0: (1/(T/2)) * integrate((func), x ,-(T/2),
9      ↵ (T/2));
10     an: (1/(T/2)) * integrate((func) *
11       ↵ series_cosine_core, x ,-(T/2), (T/2));
12     bn: (1/(T/2)) * integrate((func) *
13       ↵ series_sine_core, x ,-(T/2), (T/2));
14
15
16     a0_simp: ratsimp(a0);
17     an_simp: ratsimp(an);
18     bn_simp: ratsimp(bn);
19
20     /* Definimos el rango de n positivo y negativo */
21     n1 : 1;
22     n2 : 10;
23
24     /* Creamos la lista de An */
25     lista_An : makelist(subst(n=i, Coeff_An *
26       ↵ series_cosine_core), i, n1, n2);
27
28     /* Creamos la lista de Bn */
29     lista_Bn : makelist(subst(n=i, Coeff_Bn *
30       ↵ series_sine_core), i, n1, n2);
31
32     /* Sumamos los coeficientes An + Bn */
33     lista_completa : lista_An + lista_Bn;
34
35     /* Crear la serie final añadiendo A0 al principio
36       ↵ de la lista completa */
37     /*serie_final: cons(Coeff_A0/2,
38       ↵ lista_completa);*/

```

B.7. CÓDIGO EN PYTHON USANDO MATPLOTLIB Y SYMPY PARA LA SERIE DE FOURIER

```
35         serie_final: ratsimp(lista_completa);
36
37         serie_factor: factor(serie_final);
38
39         /* Suma de todos los coeficientes en una sola
39            ↵ variable */
40         serie_funcion : Coeff_A0/2 + sum(lista_An[i], i,
41             ↵ 1, length(lista_An)) + sum(lista_Bn[i], i, 1,
42             ↵ length(lista_Bn));
43
44         /* Simplificamos la expresión para obtener la
44            ↵ forma más compacta */
45         serie_funcion_simplificada :
46             ↵ ratsimp(serie_funcion);
47
48         /* Graficamos la expresión */
49         plot2d([func, serie_funcion_simplificada], [x,
50             ↵ -T/2, T/2]);
51
52         kill(all);
```

Código 6: Código en Maxima para calcular y graficar la serie de Fourier compleja de A.2. Fuente: *Elaboración propia*

B.7. Código en Python usando matplotlib y sympy para la serie de Fourier trigonométrica

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import sympy as sp
4
5 # Definimos las variables simbólicas
6 x = sp.symbols('x')
7
8 # Pedimos al usuario que ingrese la función
9 user_function = input("Ingresa la función a aproximar (en
10    ↵ términos de x, por ejemplo: sin(x), cos(x), x**2,
11    ↵ etc.): ")
12 function = sp.sympify(user_function)
13
14 # Definimos el intervalo y el número de términos de la
14    ↵ serie de Fourier
15 L = np.pi # Longitud del intervalo
16 N = 10 # Número de términos en la serie de Fourier
```

```

15
16      # Calculamos los coeficientes a0, an, bn
17      a0 = (1 / (2 * L)) * sp.integrate(function, (x, -L, L))
18      an = lambda n: (1 / L) * sp.integrate(function * sp.cos(n
19          * np.pi * x / L), (x, -L, L))
20      bn = lambda n: (1 / L) * sp.integrate(function * sp.sin(n
21          * np.pi * x / L), (x, -L, L))

22
23      # Calculamos la serie de Fourier
24      t = np.linspace(-L, L, 1000)
25      fourier_series = a0.evalf()

26
27      for n in range(1, N + 1):
28          fourier_series += an(n).evalf() * np.cos(n * np.pi * t /
29              L) + bn(n).evalf() * np.sin(n * np.pi * t / L)

30
31      # Convertimos la función original a una función NumPy
32          # para graficar
33      f_original = sp.lambdify(x, function, modules='numpy')

34
35      # Graficamos la función original y su serie de Fourier
36      plt.figure(figsize=(10, 6))
37      plt.plot(t, f_original(t), label='Función Original',
38          color='red', linewidth=2)
39      plt.plot(t, fourier_series, label='Serie de Fourier
40          (N=10)', color='blue', linestyle='--')
41      plt.title('Aproximación de una Función con Serie de
42          Fourier')
43      plt.xlabel('x')
44      plt.ylabel('Amplitud')
45      plt.axhline(0, color='black', linewidth=0.5,
46          linestyle='--')
47      plt.axvline(0, color='black', linewidth=0.5,
48          linestyle='--')
49      plt.grid()
50      plt.legend()

51
52      # Guarda la gráfica como archivo
53      plt.savefig('serie_fourier.png')
54      print("La gráfica se ha guardado como
55          'serie_fourier.png'.")

```

Código 7: Código en Python con matplotlib y sympy para graficar la serie de Fourier trigonométrica de A.1. Fuente: Elaboración propia

B.8. Código en Python usando matplotlib y sympy para la serie de Fourier compleja

```

1      import numpy as np
2      import matplotlib.pyplot as plt
3      import sympy as sp
4
5      # Definimos las variables simbólicas
6      x = sp.symbols('x')
7
8      # Pedimos al usuario que ingrese la función
9      user_function = input("Ingresa la función a aproximar (en
10      ↪ términos de x, por ejemplo: sin(x), cos(x), x**2,
11      ↪ etc.): ")
12      function = sp.sympify(user_function)
13
14      # Definimos el intervalo y el número de términos de la
15      ↪ serie de Fourier
16      L = np.pi # Longitud del intervalo
17      N = 10 # Número de términos en la serie de Fourier
18
19      # Calculamos los coeficientes cn para la serie de Fourier
20      ↪ en su forma exponencial compleja
21      cn = lambda n: (1 / (2 * L)) * sp.integrate(function *
22      ↪ sp.exp(-1j * n * sp.pi * x / L), (x, -L, L))
23
24      # Calculamos la serie de Fourier en su forma exponencial
25      ↪ compleja
26      t = np.linspace(-L, L, 1000)
27      fourier_series_complex = 0
28
29      for n in range(-N, N + 1):
30          fourier_series_complex += cn(n).evalf() * np.exp(1j * n *
31          ↪ np.pi * t / L)
32
33      # Convertimos la función original a una función NumPy
34      ↪ para graficar
35      f_original = sp.lambdify(x, function, modules='numpy')
36
37      # Graficamos la función original y su serie de Fourier
38      plt.figure(figsize=(10, 6))
39      plt.plot(t, f_original(t), label='Función Original',
40      ↪ color='red', linewidth=2)
41      plt.plot(t, fourier_series_complex.real, label='Serie de
42      ↪ Fourier Compleja (N=10)', color='blue',
43      ↪ linestyle='--')
```

```

33     plt.title('Aproximación de una Función con Serie de
34         ↪ Fourier Compleja')
35     plt.xlabel('x')
36     plt.ylabel('Amplitud')
37     plt.axhline(0, color='black', linewidth=0.5,
38         ↪ linestyle='--')
39     plt.axvline(0, color='black', linewidth=0.5,
40         ↪ linestyle='--')
41     plt.grid()
42     plt.legend()
43
44
45     # Guarda la gráfica como archivo
46     plt.savefig('serie_fourier_compleja.png')
47     print("La gráfica se ha guardado como
48         ↪ 'serie_fourier_compleja.png'.")

```

Código 8: Código en Python con matplotlib y sympy para calcular y graficar la serie de Fourier compleja de A.2. *Fuente: Elaboración propia*

Código en Python usando manim para la serie de Fourier trigonométrica

```

1      # Importamos todas la funciones de manim
2      from manim import *
3
4
5      # Creamos una escena
6      class Prueba(Scene):
7          def construct(self):
8
9              #Definimos un color de fondo para todo el lienzo
10             self.camera.background_color = "#212121"
11
12             x_labels = [
13                 "-3\pi",                      # -3pi
14                 "-\frac{5\pi}{2}",            # -5pi/2
15                 "-2\pi",                      # -2pi
16                 "-\frac{3\pi}{2}",            # -3pi/2
17                 "-\pi",                       # -pi
18                 "-\frac{\pi}{2}",             # -pi/2
19                 "0",                          # Blank
20                 "\frac{\pi}{2}",              # pi/2
21                 "\pi",                        # pi
22                 "\frac{3\pi}{2}",             # 3pi/2
23                 "2\pi",                       # 2pi
24                 "\frac{5\pi}{2}",             # 5pi/2
25                 "3\pi"                         # 3pi
26             ]

```

B.8. CÓDIGO EN PYTHON USANDO MATPLOTLIB Y SYMPY PARA LA SERIE DE FOURIER

```
27     y_labels = [
28         "-\frac{3\pi}{2}",      # -3pi/2
29         "-\pi",               # -pi
30         "-\frac{\pi}{2}",     # -pi/2
31         "0",                  # Blank
32         "\frac{\pi}{2}",      # pi/2
33         "\pi",                # pi
34         "\frac{3\pi}{2}"       # 3pi/2
35     ]
36
37     # Creamos los ejes
38     ejes=Axes(
39         x_range = (-3*PI, 3*PI, PI), # Rango del eje x
40         # → (inicio, fin, de cuanto en cuanto avanza)
41         x_length=(6),             # Tamaño del eje x
42         y_range = (-3*PI/2, 3*PI/2, PI), # Rango del eje y
43         # → (inicio, fin, de cuanto en cuanto avanza)
44         y_length=(3),             # Tamaño del eje y
45
46     # Configuramos los ejes.
47     axis_config={
48         "include_numbers": False,    # Los
49         # → ejes se numerarán
50         "font_size": 13,           # Tamaño
51         # → de los números
52         "tip_width": 0.05,        # Ancho de
53         # → la punta
54         "tip_height": 0.03,       # Alto de
55         # → la punta
56         "tick_size":0.06,         # Tamaño
57         # → de las lineas de los ejes
58         "color": WHITE,          # Color de
59         # → los ejes
60         "stroke_width": 1,        # Grosor
61         # → de los ejes
62         "line_to_number_buff": SMALL_BUFF # → Espacio entre las lineas y los
63         # → números
64     }
65 )
66
67     x_tex_labels = VGroup(*[
68         MathTex(t,
69             # → font_size=15).next_to(ejes.x_axis.n2p(x),DOWN*0.4)
70             # → if x >= 0 else
71             # Shift pi<0 labels to left
72             MathTex(t,
73                 # → font_size=15).next_to(ejes.x_axis.n2p(x),DOWN*0.4).shift(LL
74                 # → of radius*0.4)
```

```

61     for t,x in zip(x_labels,np.arange(-3*PI,
62                     ↵ 3*PI+PI/2, PI/2)) if t != "0"
63     # Ignore 0 value
64   ])
65
66   y_tex_labels = VGroup(*[
67     MathTex(t,
68       ↵ font_size=15).next_to(ejes.y_axis.n2p(x),LEFT*0.4)
69       ↵ if x >= 0 else
70     # Shift pi<0 labels to left
71     MathTex(t,
72       ↵ font_size=15).next_to(ejes.y_axis.n2p(x),LEFT*0.4).shift(LEFT*0.0)
73
74   for t,x in zip(y_labels,np.arange(-3*PI/2,
75                     ↵ 3*PI/2+PI/2, PI/2)) if t != "0"
76   # Ignore 0 value
77   ])
78
79
80   # Creamos el plano de números
81   plano_numeros = NumberPlane(
82     x_range = (-3*PI, 3*PI, PI), # Rango del eje x
83       ↵ (inicio, fin, de cuanto en cuanto avanza)
84     x_length=(6),           # Tamaño del eje x
85     y_range = (-3*PI/2, 3*PI/2, PI), # Rango del eje y
86       ↵ (inicio, fin, de cuanto en cuanto avanza)
87     y_length=(3),           # Tamaño del eje y
88     faded_line_ratio=2,      # Dentro de cada recuadro
89       ↵ de unidad, habrá dos de un color mas opaco
90
91   # Configuración de los ejes
92   axis_config={
93     "include_numbers": False,    # Los ejes no
94       ↵ se numerarán
95     "tip_width": 0.05,
96     "tip_height": 0.03,
97     "tick_size": 0.06,
98     "color": WHITE
99   },
100
101   # Configuración del plano
102   background_line_style={
103     "stroke_color": TEAL,      # Color de las
104       ↵ líneas del plano
105     "stroke_width": 1,         # Grosor de las
106       ↵ líneas
107     "stroke_opacity": 0.6      # Opacidad de las
108       ↵ líneas
109   }
110 ).set_z_index(-1)

```

B.8. CÓDIGO EN PYTHON USANDO MATPLOTLIB Y SYMPY PARA LA SERIE DE FOURIER

```
97
98         #ejes.height = 6
99         #ejes.width = 9
100
101     # Título en la parte superior
102     titulo = Text("Serie de Fourier Trigonométrica
103             ↪ de:", font_size=32)
104     fun = MathTex(r"f(t)= t, -\pi <t< \pi",
105             ↪ font_size=25)
106
107     # Agrupamos ambos textos en un solo grupo y lo
108             ↪ posicionamos en la parte superior
109     titulo.next_to(fun, UP)
110     encabezado = VGroup(titulo, fun)
111     encabezado.to_edge(UP).set_z_index(1)
112
113     # Etiquetas de ejes
114     eje_x =
115             ↪ ejes.get_x_axis_label("t").next_to(ejes.x_axis.get_end(),
116             ↪ UP*0.1).scale(0.45).set_color(WHITE)
117     eje_y =
118             ↪ ejes.get_y_axis_label("f(t)").next_to(ejes.y_axis.get_end(),
119             ↪ RIGHT*0.1).scale(0.45).set_color(WHITE)
120
121     ejes_etiquetas = VGroup(eje_x, eje_y)
122
123
124     # Función f(t) en la parte inferior
125     funcion_label = MathTex(
126             r"f(t)={-2}\sum_{n=1}^{\infty}\left [
127             ↪ \frac{(-1)^{n}}{n} \sin(nt) \right ]",
128             font_size=24
129             )
130     #funcion_label.set_color_by_tex("n", YELLOW)
131     funcion_label.next_to(ejes, DOWN*1.5)
132
133
134     # Usando dos rectángulos creamos una ventana para
135             ↪ dibujar dentro todo nuestro plano
136     rec1 = Rectangle(height=80, width=40)
137     rec2 = Rectangle(height=3.5, width=6.6)
138     ventana = Cutout(rec1, rec2, fill_opacity=1,
139             ↪ color="#212121", stroke_color=RED)
140
141
142     # Creamos un contador n para mostrar la cantidad
143             ↪ de términos que se añaden a la suma
144     i_value_text = Text("n = 0", color=WHITE,
145             ↪ font_size=16)
146     i_value_text.next_to(funcion_label, DOWN)
```

```

133      # Creamos la función a graficar en el intervalo
134      # establecido
135      func_Original=ejes.plot(
136          lambda x: x, x_range=(-PI,PI), color="#d7da63",
137          )
138
139      # Definimos la función serie compleja
140      def funcion_Serie_Trig(self, x, i):
141          # Inicialmente valdrá 0
142          val=0
143
144          # El coeficiente a0 es 0
145          a0 = 0
146
147          # En cada iteración sumará un elemento k positivo
148          # y uno negativo
149          for k in range(1,i+1):
150              try:
151                  val += (((-1)**(k)) / k) * np.sin(k * x)
152
153          # Si algún termino es indeterminado se sumará un
154          # 0
155          except ZeroDivisionError:
156              val += 0
157
158          # EL valor de la suma será multiplicado por su
159          # coeficiente
160          return a0 - (2) * val
161
162
163          self.play(Write(encabezado))
164          self.play(Create(ventana))
165          self.play(Create(ejes), Create(plano_numeros),
166          # Create(x_tex_labels), Create(y_tex_labels))
167          self.play(Write(ejes_etiquetas))
168          self.play(Create(func_Original))
169          self.wait(3)
170
171
172          # Se crean las aproximaciones en la Serie
173          for j in range (0,6):
174              i_value_text.set_text(f"n = {j}")
175              fsC_0=ejes.plot(
176                  lambda x: funcion_Serie_Trig(self, x, j),
177                  x_range=(-3*PI,3*PI), color=BLUE, stroke_width
178                  = 2.5
179                  )
180
181          if j==0:

```

B.8. CÓDIGO EN PYTHON USANDO MATPLOTLIB Y SYMPY PARA LA SERIE DE FOURIER

```

173         self.play(Create(fsC_0), Write(funcion_label),
174             Write(i_value_text))
175     old_graph=fsC_0
176     else:
177         self.play(Transform(old_graph, fsC_0),
178             Transform(i_value_text, Text(f"n = {j}" ,
179             color=WHITE,
180             font_size=16).next_to(funcion_label, DOWN)))
181
182     plano = VGroup(encabezado, ejes, ejes_etiquetas,
183                     ventana, func_Original, old_graph)
184     plano.move_to(ORIGIN)
185     self.wait(2)

```

Código 9: Código en Python con Manim para graficar la serie de Fourier trigonométrica de A.1. Fuente: *Elaboración propia*

Código en Python usando Manim para la serie de Fourier compleja

```

1      # Importamos todas la funciones de manim
2      from manim import *
3
4      # Creamos una escena
5      class Prueba(Scene):
6          def construct(self):
7
8              #Definimos un color de fondo para todo el lienzo
9              self.camera.background_color = "#212121"
10
11             x_labels = [
12                 "-3\\pi",                      # -3pi
13                 "-\\frac{5\\pi}{2}",           # -5pi/2
14                 "-2\\pi",                      # -2pi
15                 "-\\frac{3\\pi}{2}",           # -3pi/2
16                 "-\\pi",                        # -pi
17                 "-\\frac{\\pi}{2}",            # -pi/2
18                 "0",                            # Blank
19                 "\\frac{\\pi}{2}",            # pi/2
20                 "\\pi",                          # pi
21                 "\\frac{3\\pi}{2}",           # 3pi/2
22                 "2\\pi",                         # 2pi
23                 "\\frac{5\\pi}{2}",           # 5pi/2
24                 "3\\pi"                         # 3pi
25             ]
26
27             y_labels = [
28                 "-\\frac{3\\pi}{2}",           # -3pi/2
29                 "-\\pi",                      # -pi

```

```

30      "-\\frac{\\pi}{2}",      # -pi/2
31      "0",                  # Blank
32      "\\frac{\\pi}{2}",     # pi/2
33      "\\pi",                # pi
34      "\\frac{3\\pi}{2}"     # 3pi/2
35
36  ]
37
38
39  # Creamos los ejes
40  ejes=Axes(
41      x_range = (-3*PI, 3*PI, PI), # Rango del eje x
42      #<-- (inicio, fin, de cuanto en cuanto avanza)
43      x_length=(6),             # Tamaño del eje x
44      y_range = (-3*PI/2, 3*PI/2, PI), # Rango del eje y
45      #<-- (inicio, fin, de cuanto en cuanto avanza)
46      y_length=(3),             # Tamaño del eje y
47
48  # Configuramos los ejes.
49  axis_config={
50      "include_numbers": False,          # Los
51      #<-- ejes se numerarán
52      "font_size": 13,                  # Tamaño
53      #<-- de los números
54      "tip_width": 0.05,               # Ancho de
55      #<-- la punta
56      "tip_height": 0.03,              # Alto de
57      #<-- la punta
58      "tick_size":0.06,               # Tamaño
59      #<-- de las líneas de los ejes
60      "color": WHITE,                 # Color de
61      #<-- los ejes
62      "stroke_width": 1,               # Grosor
63      #<-- de los ejes
64      "line_to_number_buff": SMALL_BUFF #
65      #<-- Espacio entre las líneas y los
66      #<-- números
67  }
68
69
70  x_tex_labels = VGroup(*[
71      MathTex(t,
72          font_size=15).next_to(ejes.x_axis.n2p(x),DOWN*0.4)
73          #<-- if x >= 0 else
74      # Shift pi<0 labels to left
75      MathTex(t,
76          font_size=15).next_to(ejes.x_axis.n2p(x),DOWN*0.4).shift(LEFT*0.0

```

B.8. CÓDIGO EN PYTHON USANDO MATPLOTLIB Y SYMPY PARA LA SERIE DE FOURIER

```
64         for t,x in zip(x_labels,np.arange(-3*PI,
65                         ↵ 3*PI+PI/2, PI/2)) if t != "0"
66             # Ignore 0 value
67         ])
68
69         y_tex_labels = VGroup(*[
70             MathTex(t,
71                 ↵ font_size=15).next_to(ejes.y_axis.n2p(x),LEFT*0.4)
72                 ↵ if x >= 0 else
73             # Shift pi<0 labels to left
74             MathTex(t,
75                 ↵ font_size=15).next_to(ejes.y_axis.n2p(x),LEFT*0.4).shift(LEFT*0.4)
76         for t,x in zip(y_labels,np.arange(-3*PI/2,
77                         ↵ 3*PI/2+PI/2, PI/2)) if t != "0"
78             # Ignore 0 value
79         ])
80
81
82         # Creamos el plano de números
83         plano_numeros = NumberPlane(
84             x_range = (-3*PI, 3*PI, PI), # Rango del eje x
85                 ↵ (inicio, fin, de cuanto en cuanto avanza)
86             x_length=(6),           # Tamaño del eje x
87             y_range = (-3*PI/2, 3*PI/2, PI), # Rango del eje y
88                 ↵ (inicio, fin, de cuanto en cuanto avanza)
89             y_length=(3),           # Tamaño del eje y
90             faded_line_ratio=2,      # Dentro de cada recuadro
91                 ↵ de unidad, habrá dos de un color mas opaco
92
93         # Configuración de los ejes
94         axis_config={
95             "include_numbers": False,    # Los ejes no
96                 ↵ se numerarán
97             "tip_width": 0.05,
98             "tip_height": 0.03,
99             "tick_size": 0.06,
100            "color": WHITE
101        },
102
103         # COnfiguración del plano
104         background_line_style={
105             "stroke_color": TEAL,      # Color de las
106                 ↵ líneas del plano
107             "stroke_width": 1,        # Grosor de las
108                 ↵ líneas
109             "stroke_opacity": 0.6     # Opacidad de las
110                 ↵ líneas
111         }
112     ).set_z_index(-1)
```

```

100
101      # Título en la parte superior
102      titulo = Text("Serie de Fourier compleja de:",
103                      font_size=32)
103      fun = MathTex(r"f(t)= t, -\pi < t < \pi",
104                      font_size=32)
104
105      # Agrupamos ambos textos en un solo grupo y lo
106      # posicionamos en la parte superior
106      titulo.next_to(fun, UP)
107      encabezado = VGroup(titulo, fun)
108      encabezado.to_edge(UP).set_z_index(1)
109
110      # Etiquetas de ejes
111      eje_x =
112          eje_x.get_x_axis_label("t").next_to(ejes.x_axis.get_end(),
113          UP*0.1).scale(0.45).set_color(WHITE)
112      eje_y =
113          eje_y.get_y_axis_label("f(t)").next_to(ejes.y_axis.get_end(),
114          RIGHT*0.1).scale(0.45).set_color(WHITE)
114
115      ejes_etiquetas = VGroup(eje_x, eje_y)
116
116      # Función f(t) en la parte inferior
117      funcion_label = MathTex(
118          r"f(t)= i\sum_{n=-\infty}^{\infty}\left(\frac{(-1)^n}{n} e^{int}\right)",
119          font_size=24
120      )
121      #funcion_label.set_color_by_tex("n", YELLOW)
122      funcion_label.next_to(ejes, DOWN*2.4)
123
124      # Usando dos rectangulos creamos una ventana para
125      # dibujar dentro todo nuestro plano
125      rec1 = Rectangle(height=80, width=40)
126      rec2 = Rectangle(height=3.8, width=6.6)
127      ventana = Cutout(rec1, rec2, fill_opacity=1,
128                      color="#212121", stroke_color=RED)
128
129      # Creamos un contador n para mostrar la cantidad
130      # de términos que se añaden a la suma
130      i_value_text = Text("n = 0", color=WHITE,
131                      font_size=16)
131      i_value_text.next_to(funcion_label, DOWN)
132
133
134      # Definimos la función serie compleja
134      def funcion_Serie_Compleja(self, x, i):

```

B.8. CÓDIGO EN PYTHON USANDO MATPLOTLIB Y SYMPY PARA LA SERIE DE FOURIER

```
136      # Inicialmente valdrá 0
137      val=0
138
139      # Como el coeficiente c0 no es indeterminado, lo
140      # calculamos aparte
140      c0 = 0
141
142      # En cada iteración sumará un elemento k positivo
143      # y uno negativo
143      for k in range(1,i+1):
144          try:
145
146              val += (((-1)**(k)) / k) * (np.exp(complex(0, (k
147              * x))))
147              val += (((-1)**(-k)) / (-k)) * (np.exp(complex(0,
148              (-k) * x))))
149
149      # Si algún término es indeterminado se sumará un
150      # 0
150      except ZeroDivisionError:
151          val += 0
152
153      # El valor de la suma será multiplicado por su
154      # coeficiente
154      return complex(0,1) * val + c0
155
156      # Creamos la función a graficar en el intervalo
157      # establecido
157      func_Original=ejes.plot(
158          lambda x: x, x_range=(-PI,PI), color="#d7da63",
159          )
160
161      self.play(Write(encabezado))
162      self.play(Create(ventana))
163      self.play(Create(ejes), Create(plano_numeros),
164      Create(x_tex_labels), Create(y_tex_labels))
164      self.play(Write(ejes_etiquetas))
165      self.play(Create(func_Original))
166
167
168      # Se crean las aproximaciones en la Serie
169      for j in range (0,6):
170          i_value_text.set_text(f"n = {j}")
171          fsC_0=ejes.plot(
172              lambda x: funcion_Serie_Compleja(self, x, j),
173              x_range=(-3*np.pi,3*np.pi), color=BLUE,
173              stroke_width = 2.5
173          )
```

```
174     if j==0:  
175         self.play(Create(fsC_0), Write(funcion_label),  
176             Write(i_value_text))  
177         old_graph=fsC_0  
178     else:  
179         self.play(Transform(old_graph, fsC_0),  
180             Transform(i_value_text, Text(f"n = {j}"),  
181             color=WHITE,  
182             font_size=16).next_to(funcion_label, DOWN)))  
183  
184     plano = VGroup(encabezado, ejes, ejes_etiquetas,  
185                     ventana, func_Original, old_graph)  
186     plano.move_to(ORIGIN)  
187     self.wait(2)
```

Código 10: Código en Python con Manim para graficar la serie de Fourier compleja de A.1. *Fuente: Elaboración propia*

Bibliografía

- [1] J.M. Almira. *Fourier: un matemático al servicio de la física*. Genios de las matemáticas. RBA, 2017. ISBN: 9788447387755.
- [2] Wolfram: Computation Meets Knowledge. *Wolfram Mathematica: Modern Technical Computing*. <https://www.wolfram.com/mathematica/>. Accedido: 31 de Octubre de 2024. 2024.
- [3] Wolfram — Alpha: Computational Intelligence. *Fourier Analysis—Wolfram Language Documentation*. <https://reference.wolfram.com/language/guide/FourierAnalysis.html>. Accedido: 17 Junio, 2024. 2024.
- [4] Wolfram Language — System Documentation Center. *Piecewise: Function defined in pieces for different conditions—Wolfram Documentation*. <https://reference.wolfram.com/language/ref/Piecewise.html>. Accedido: 31 de Octubre de 2024. 2024.
- [5] Solucionador matemático Symbolab - calculadora paso a paso. *Solucionador matemático Symbolab - calculadora paso a paso*. <https://es.symbolab.com>. Accedido: 31 de Octubre de 2024. 2024.
- [6] MathWorks. *MathWorks - Creadores de MATLAB y Simulink - MATLAB y Simulink*. <https://la.mathworks.com/>. Accedido: 14 Junio, 2024. 2024.
- [7] Engineering Maplesoft - Software for Mathematics Online Learning. *Computer Algebra Systems (CAS) - Maplesoft*. https://www.maplesoft.com/documentation_center/. Accedido: 3 Noviembre, 2024. 2024.
- [8] R. J. Lopez. *Classroom Tips and Techniques: Teaching Fourier Series with Maple - Part 3*. <https://www.maplesoft.com/applications/download.aspx?id=4885/TeachingFourierSerieswithMaple-Part3.pdf>. Accedido: 3 Noviembre, 2024. Maplesoft - Software for Mathematics, Online Learning, Engineering, 2024.
- [9] maxima. *Maxima, a Computer Algebra System. Version 5.47.0*. <https://maxima.sourceforge.io/>. Accedido: 27 Octubre, 2024. 2023.
- [10] GeoGebra. *GeoGebra - the world's favorite, free math tools used by over 100 million students and teachers*. <https://www.geogebra.org>. Accedido: 14 Junio, 2024. 2024.
- [11] Desmos. *Desmos — Let's learn together*. <https://www.desmos.com>. Accedido: 14 Junio, 2024. 2024.

- [12] Inc. Amazon Web Services. *¿Qué es Python? - Explicación del lenguaje Python - AWS.* <https://aws.amazon.com/es/what-is/python/>. Accedido: 31 de Octubre de 2024. 2024.
- [13] SymPy Development Team. *Sympy Documentation.* Accessed: 2023-10-01. 2023. URL: <https://docs.sympy.org/latest/>.
- [14] Matplotlib Development Team. *Matplotlib Documentation.* Accessed: 2023-10-01. 2023. URL: <https://matplotlib.org/stable/contents.html>.
- [15] Manim Community. *Manim Community.* <https://www.manim.community>. Accedido: 14 Junio, 2024. 2024.
- [16] Ettore Messina. *Fourier Series in Python.* <https://computationalmindset.com/en/mathematics/fourier-series-in-python.html>. Accedido: 3 Octubre, 2024. 2024.
- [17] A. Cañada. *Una perspectiva histórica de las series de Fourier: de las ecuaciones de ondas y del calor a los operadores compactos y autoadjuntos.* <https://www.nieuwarchief.nl/serie5/pdf/naw5-2000-01-3-242.pdf>. Accedido: 23 Octubre, 2024. 2000.
- [18] H. F. Weinberger. *Ecuaciones diferenciales en derivadas parciales con métodos de variable compleja y de transformadas integrales.* Editorial Reverté, 1970. ISBN: 9788429151602.
- [19] O. L. Campo Bedoya. *Ecuación de D'Alembert, de la cuerda vibrante, bajo la teoría de Lie.* <https://repositorio.unal.edu.co/handle/unal/52162>. Accedido: 23 Octubre, 2024. 2014.
- [20] D. Chamorro. *Lección n°5: Ecuación de Ondas.* https://www.amarun.org/images/amarun/materiales/EDP/edp/Leccion_5.pdf. Accedido: 23 Octubre, 2024. 2015.
- [21] I. Grattan-Guinness. *Daniel Bernoulli and the varieties of mechanics.* <https://www.nieuwarchief.nl/serie5/pdf/naw5-2000-01-3-242.pdf>. Accedido: 23 Octubre, 2024. 2000.
- [22] E. Garber. *The Language of Physics: The Calculus and the Development of Theoretical Physics in Europe, 1750-1914.* Springer, 1999. ISBN: 978-1-4612-7272-4.
- [23] J. T. Brereton. *FOURIER SERIES: SOLVING THE HEAT EQUATION.* <https://math.berkeley.edu/~jbrere/heatequation>. Accedido: 23 Octubre, 2024. 2015.
- [24] J. Feldman. *Solution of the Heat Equation by Separation of Variables.* <https://personal.math.ubc.ca/~feldman/m267/heatSln.pdf>. Accedido: 23 Octubre, 2024. 2007.
- [25] G. P. Tolstov. *Fourier Series.* Prentice-Hall, Inc., 1962. ISBN: 9780486633176.
- [26] H. P. Hsu. *Analisis de Fourier.* Addison-Wesley Iberoamericana, 1987. ISBN: 9685000476.
- [27] W. Gómez Flores. *Introducción al Análisis de Fourier.* https://www.tamps.cinvestav.mx/~wgomez/documentos/analisis_de_fourier.pdf. Accedido: 26 Octubre, 2024. Cinvestav, 2024.

- [28] W. Moebs. *15.1 Movimiento armónico simple - Física universitaria volumen 1 — OpenStax*. <https://openstax.org/books/fsica-universitaria-volumen-1/pages/15-1-movimiento-armonico-simple>. Accedido: 26 Octubre, 2024.
- [29] D. G. Zill. *Matemáticas Avanzadas para la Ingeniería*. McGraw-Hill, 2011. ISBN: 9786071507723.
- [30] D. Poole. *Linear Algebra, A Modern Introduction 3rd ed.* Cengage Learning, 2011. ISBN: 9780538735452.
- [31] C. F. Cruz Fierro. *Ecuaciones Diferenciales / Matemáticas 5 / Series de Fourier*. <https://cruzfierro.com/cursos/2011v/matematicas5/apuntes5.pdf>. Accedido: 27 Octubre, 2024.
- [32] Matemáticas con Luz. *Serie Compleja de Fourier*. <https://www.youtube.com/watch?v=3F4XReHr3i8>. Accedido: 27 Octubre, 2024. Youtube, 2020.
- [33] C. J. Carrillo González. *Fundamentos del Análisis de Fourier*. https://grupo_ene.webs.uvigo.es/wordpress/publicaciones/Apuntes_Fourier.pdf. Accedido: 27 Octubre, 2024. Youtube, 2003.
- [34] Diego Alejandro Jiménez Daza. «Herramientas digitales para la enseñanza de las matemáticas en la educación básica». Disponible en <https://repository.ucc.edu/server/api/core/bitstreams/c3880d08-528b-40cb-823b-6571c73b8132/content>. Trabajo de grado. Universidad Cooperativa de Colombia.
- [35] ¿Qué es una aplicación web? <https://aws.amazon.com/es/what-is/web-application/>. Accedido: 27 Octubre, 2024. Amazon Web Services, Inc., 2023.
- [36] Introducción a la Arquitectura web. <https://wiki.uqbar.org/wiki/articles/ui-web-intro-arquitectura.html>. Accedido: 29 Octubre, 2024. uqbar-wiki, 2023.
- [37] MDN Web Docs. *HTML: HyperText Markup Language — MDN*. https://www.w3schools.com/graphics/canvas_intro.asp. Accedido: 27 Octubre, 2024.
- [38] W3Schools.com. *W3Schools Online Web Tutorials*. <https://developer.mozilla.org/en-US/docs/Web/HTML>. Accedido: 29 Octubre, 2024.
- [39] W3Schools.com. *CSS: Cascading Style Sheets — MDN*. <https://developer.mozilla.org/en-US/docs/Web/HTML>. Accedido: 29 Octubre, 2024.
- [40] MDN Web Docs. *JavaScript — MDN*. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Accedido: 29 Octubre, 2024.
- [41] Kinsta®. *Qué es Node.js y por qué debería usarlo*. <https://kinsta.com/es/base-de-conocimiento/que-es-node-js/>. Accedido: 2 Noviembre, 2024.
- [42] TypeScript: JavaScript With Syntax For Types. *The starting point for learning TypeScript*. <https://www.typescriptlang.org/docs/>. Accedido: 29 Octubre, 2024.

- [43] Documentation - Tailwind CSS. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.* <https://tailwindcss.com/docs/>. Accedido: 29 Octubre, 2024.
- [44] Dazzet. *Qué es una API, cómo funciona y para qué sirve - Dazzet.* <https://dazzet.co/que-es/api/>. Accedido: 2 Noviembre, 2024.
- [45] Editorial Etecé. *HTTP - Concepto, para qué sirve y cómo funciona.* <https://concepto.de/http/>. Accedido: 2 Noviembre, 2024.
- [46] M. C. Albornoz. *Diseño de interfaz gráfica de usuario.* <https://sedici.unlp.edu.ar/handle/10915/41578>. Accedido: 27 Octubre, 2024. SEDICI, 2014.
- [47] Semrush Blog. *Principios de usabilidad web de Jacob Nielsen y el diseño UX.* <https://es.semrush.com/blog/usabilidad-web-principios-jakob-nielsen/>. Accedido: 2 Noviembre, 2024. 2022.
- [48] IETF Datatracker. *RFC 2818: HTTP Over TLS.* <https://datatracker.ietf.org/doc/html/rfc2818>. Accedido: 2 Noviembre, 2024.
- [49] SSL.com. *¿Qué es HTTPS? - SSL.com.* <https://www.ssl.com/es/preguntas-frecuentes/que-es-https/>. Accedido: 2 Noviembre, 2024.
- [50] SSL.com. *Que es SSL /TLS: Una guía detallada - SSL.com.* <https://www.ssl.com/es/article/what-is-ssl-tls-an-in-depth-guide/>. Accedido: 2 Noviembre, 2024.
- [51] MathWorks. *Metodologías de desarrollo.* <https://woodyweb.wordpress.com/2015/08/24/metodologias-de-desarrollo/>. Accedido: 22 Oct, 2024. 2015.