



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Ingeniería en Inteligencia Artificial

Cómputo Paralelo - 6BV1

Castro Paez Ilse Yazbeth

Catonga Tecla Daniel Isaí

Olguin Castillo Victor Manuel

Padilla Sanchez Hariel

Fecha de entrega: 16 de mayo de 2025

Índice general

<i>Índice de figuras</i>	III
1. Introducción	1
2. Contexto de Serie de Fourier	2
2.1. Forma Compleja de la Serie de Fourier	2
2.2. Fórmula Trigonométrica de la Serie de Fourier	3
2.3. Aplicaciones de la Serie de Fourier	4
2.3.1. Aplicación en la ciencia	4
2.3.2. Aplicación en la tecnología	5
2.3.3. Aplicación en la ingeniería	6
2.3.4. Aplicación en la Inteligencia Artificial	6
2.3.5. Aplicación en el cómputo paralelo	7
2.4. Teoría de Procesos Hijos con la Llamada al Sistema fork() en Lenguaje C	8
2.4.1. Comunicación entre procesos padre e hijo	9
2.4.2. Aplicaciones prácticas de fork()	9
2.5. Teoría sobre Memoria Compartida en Lenguaje C	10
2.5.1. Implementación de memoria compartida POSIX	11
2.5.2. Implementación de memoria compartida System V	11
2.5.3. Consideraciones prácticas en el uso de memoria compartida	12
2.6. Teoría sobre Semáforos en Lenguaje C	13
2.6.1. Semáforos POSIX	13
2.6.2. Semáforos System V	14
2.6.3. Patrones de uso de semáforos	15
3. Soluciones individuales de series de Fourier	16
3.1. Función resuelta por Castro Paez Ilse Yazbeth	17
3.2. Función resuelta por Catonga Tecla Daniel Isaí	18
3.3. Función resuelta por Padilla Sanchez Hariel	22
3.4. Función resuelta por Olguin Castillo Víctor Manuel	23
3.5. Programa en C para el cálculo de la serie de Fourier	26
3.6. Análisis de la aproximación mediante serie de Fourier	31
3.7. Programa en C para el cálculo de la serie de Fourier usando hilos	32
4. Conclusiones	38

4.1. Castro Paez Ilse Yazbeth	39
4.2. Catonga Tecla Daniel Isaí	41
4.3. Padilla Sanchez Hariel	44
Anexos	48

Índice de figuras

2.1. Retrato caricaturesco de J. Fourier, atribuido a Julien Leopold Boilly	2
3.1. Cálculo de la función $f(x) = 9 - 3x - x^2$	17
3.2. Cálculo de a_0 para $f(x) = 6 - 2x$	18
3.3. Cálculo parcial de a_n para $f(x) = 6 - 2x$	19
3.4. Cálculo de a_n y desarrollo de b_n para $f(x) = 6 - 2x$	20
3.5. Cálculo de b_n para $f(x) = 6 - 2x$	21
3.6. Cálculo de a_0 y a_n para $f(x) = 6 - 4x$	22
3.7. Cálculo de b_n para $f(x) = 6 - 4x$	23
3.8. Cálculo de a_0	24
3.9. Cálculo de a_n	25
3.10. Cálculo de b_n	26
3.11. Librerías utilizadas en el programa	27
3.12. Función para calcular $b_n \sin(nx)$	27
3.13. Creación del semáforo	28
3.14. Creación de archivos vacíos	28
3.15. Generación de valores de x	28
3.16. Creación y sincronización de procesos	29
3.17. Suma de los resultados y generación de archivo	30
3.18. Escritura de resultados en archivo CSV	30
3.19. Impresión de la matriz de resultados	31
3.20. Cierre del programa y liberación de recursos	31
3.21. Grafica de resultados sobre los calculos en excel y el código C.	32
3.22. Fragmento de código donde se incluyen las bibliotecas y se definen las constantes utilizadas	33
3.23. Definición de la estructura utilizada para pasar parámetros a cada hilo.	34
3.24. Función que calcula cada fila de la matriz con base en el índice del hilo.	34
3.25. Fragmento del código que muestra la creación y sincronización de los hilos.	35
3.26. Código encargado de calcular la suma total de la serie de Fourier en cada punto.	35
3.27. Fragmento de código encargado de guardar los resultados en un archivo CSV.	36
3.28. Impresión de los resultados aproximados de $f(x)$ en la terminal.	36
3.29. Comparativa gráfica entre el valor real de la función y las aproximaciones de las fases 2, 3 y 4.	37

1

Introducción

Uno de los aspectos clave del análisis de Fourier es la obtención de los coeficientes que permiten reconstruir una función a partir de sus componentes armónicas. Estos coeficientes determinan la contribución de cada frecuencia en la representación de la función original y son esenciales para aplicaciones como el procesamiento de señales, la resolución de ecuaciones diferenciales y la compresión de datos.

En este proyecto, se estudia el cálculo de los coeficientes de Fourier para una función específica, analizando el procedimiento matemático necesario para su determinación. A través de este desarrollo, se busca comprender cómo el análisis de Fourier permite representar funciones de manera eficiente y cómo esta técnica se aplica en diferentes áreas del conocimiento.

Se incluye un marco teórico que explora las aplicaciones más relevantes del análisis de Fourier, destacando su impacto en la ciencia y la tecnología.

Este trabajo tiene como objetivo reforzar la comprensión de las series de Fourier y su utilidad en el estudio de fenómenos periódicos, proporcionando una base sólida para su aplicación en problemas reales. A través de la implementación práctica del cálculo de coeficientes de Fourier, se demuestra cómo esta herramienta facilita el análisis de funciones y contribuye a desarrollar soluciones más eficientes en diversas disciplinas científicas e ingenieriles.

El análisis de Fourier y el cálculo de sus coeficientes son herramientas fundamentales en las matemáticas, con un impacto significativo en diversas disciplinas, que abarcan desde la ingeniería hasta las ciencias sociales. Esta proyecto tiene como objetivo no solo desarrollar la teoría subyacente a este proceso, sino también profundizar en la importancia de aplicar las series de Fourier en la resolución de problemas complejos. A través de este enfoque, se pretende demostrar cómo estas series permiten transformar funciones en distintas áreas del conocimiento, facilitando su análisis y solución.

2

Contexto de Serie de Fourier

Según [2], el estudio de las series de Fourier se originó en los trabajos de Joseph Fourier sobre la propagación del calor en cuerpos sólidos. En 1807, presentó su memoria *Mémoire sur la propagation de la chaleur dans les corps solides*, donde analizaba la evolución de la temperatura en función del tiempo. Su trabajo fue clave para el desarrollo de la teoría del análisis de Fourier, que posteriormente consolidó en 1822 con la publicación de *Théorie analytique de la chaleur*, estableciendo los principios para la expansión en series de funciones trigonométricas que representan fenómenos periódicos.



Figura 2.1: Retrato caricaturesco de J. Fourier, atribuido a Julien L. Imagen tomada de [2]

Las Series de Fourier constituyen una herramienta fundamental en el análisis de funciones, permitiendo representar funciones periódicas como una suma infinita de senos y cosenos. Su estudio se basa en la construcción del espacio L^2 y en la determinación de los coeficientes que conforman la serie. Además, su aplicación se extiende a funciones no periódicas mediante técnicas de extensión y generalización a períodos arbitrarios [29].

2.1. Forma Compleja de la Serie de Fourier

La serie de Fourier puede expresarse en su forma compleja, lo que resulta conveniente en contextos que requieren trabajar con números complejos y simplifica el uso de la Transformada de Fourier [29]. En esta notación, una función $f(x)$ periódica con período T se representa como:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}, \quad (2.1)$$

Los coeficientes c_n se determinan mediante la integral:

$$c_n = \frac{1}{T} \int_T f(x) e^{-inx} dx \quad (2.2)$$

Esta forma compleja de la serie de Fourier permite una interpretación más directa en términos de números complejos y resulta fundamental para la formulación de la Transformada de Fourier [33, 29].

2.2. Fórmula Trigonométrica de la Serie de Fourier

El problema consiste en, dada una función f de período 2π , encontrar su representación en forma de una serie trigonométrica, como se describe en [2]. La Serie de Fourier para una función $f(x)$ en el intervalo $[-\pi, \pi]$ se expresa de la siguiente manera:

$$f(x) = A + \sum_{n=1}^{\infty} a_n \cos(nx) + b_n \sin(nx), \quad \forall x \in T \quad (2.3)$$

En esta ecuación, los coeficientes a_0 , a_n y b_n se determinan mediante integrales sobre el intervalo T . Esta representación permite expresar la función $f(x)$ como una suma infinita de funciones trigonométricas, lo cual facilita el análisis de funciones periódicas.

Los coeficientes a_0 , a_n y b_n se calculan mediante las siguientes integrales:

Coeficiente a_0 :

$$a_0 = \frac{2}{T} \int_T f(x) dx \quad (2.4)$$

Coeficientes a_n y b_n :

$$a_n = \frac{2}{T} \int_T f(x) \cos\left(\frac{2\pi n}{T}x\right) dx \quad (2.5)$$

y

$$b_n = \frac{2}{T} \int_T f(x) \sin\left(\frac{2\pi n}{T}x\right) dx \quad (2.6)$$

El procedimiento para obtener el coeficiente a_0 en la forma trigonométrica de la Serie de Fourier, a partir de la ecuación (2.3), es el siguiente. Al integrar esta expresión sobre el intervalo T , y aplicar las propiedades de ortogonalidad de los senos y cosenos, se obtiene:

$$\int_T f(x) dx = \int_T A dx + \int_T \sum_{n=1}^{\infty} a_n \cos(nx) + b_n \sin(nx) dx$$

$$\begin{aligned}
 &= A \cdot 2\pi + \sum_{n=1}^{\infty} a_n \underbrace{\int_T \cos(nx) dx}_0 + b_n \underbrace{\int_T \sin(nx) dx}_0 \\
 &= A \cdot 2\pi.
 \end{aligned} \tag{2.7}$$

Finalmente, tomando $A = \frac{a_0}{2}$, se obtiene el valor de a_0 como:

$$a_0 = \frac{1}{\pi} \int_T f(x) dx. \tag{2.8}$$

Para obtener los coeficientes a_n y b_n , el proceso es análogo. Consiste en multiplicar la ecuación (2.3) por $\cos(kx)$ y $\sin(kx)$, respectivamente, e integrar sobre el intervalo T . Para una derivación detallada de estos coeficientes y una discusión más profunda sobre las series de Fourier, se recomienda consultar [29].

2.3. Aplicaciones de la Serie de Fourier

El análisis de Fourier es una herramienta fundamental en matemáticas e ingeniería, utilizada para descomponer funciones en una suma infinita de senos y cosenos. En particular, la serie de Fourier permite representar funciones periódicas mediante combinaciones de funciones trigonométricas, lo que facilita su estudio y aplicación en diversas áreas del conocimiento [33].

A lo largo de los años, la serie de Fourier ha demostrado ser una herramienta esencial en la resolución de problemas donde la periodicidad y la descomposición en frecuencias juegan un papel clave. Su utilidad se extiende más allá del análisis matemático, permitiendo modelar fenómenos físicos y optimizar procesos en ingenierías.

Gracias a su capacidad para transformar funciones complejas en combinaciones de términos sinusoidales, esta técnica se ha convertido en un pilar en múltiples disciplinas.

Aplicación en la ciencia

La serie de Fourier es una herramienta matemática fundamental en las áreas de la ciencia como se muestra a continuación.

En mecánica cuántica, la serie de Fourier es fundamental para resolver la ecuación de Schrödinger en sistemas periódicos. Su aplicación permite describir la evolución temporal de partículas en potenciales periódicos, como en los cristales cuánticos y los superconductores [17]. Además, la transformada de Fourier es utilizada en la representación de estados cuánticos en diferentes bases, lo que facilita la simulación de sistemas cuánticos complejos en computación cuántica [30].

En astronomía, la serie de Fourier se emplea en la espectroscopía astronómica para descomponer la luz de estrellas y galaxias en componentes espectrales. Esto permite identificar la composición química de cuerpos celestes y analizar efectos como el desplazamiento Doppler en exoplanetas [8].

En la radioastronomía, la transformada de Fourier es utilizada en la síntesis de imágenes obtenidas por interferometría, lo que ha permitido la observación de agujeros negros y la detección de ondas gravitacionales [1].

En ciencia de materiales, la serie de Fourier se aplica en el estudio de estructuras cristalinas. Mediante la difracción de rayos X, los científicos pueden analizar patrones de interferencia que se traducen en series de Fourier para determinar la disposición atómica en materiales sólidos. Esto es esencial para el diseño de nuevos materiales con propiedades específicas, como superconductores o aleaciones de alta resistencia [13].

En la física, la serie de Fourier es ampliamente utilizada para resolver ecuaciones diferenciales que describen sistemas oscilatorios y ondulatorios. Por ejemplo, en mecánica cuántica, la transformada de Fourier permite representar funciones de onda en el espacio de momentos, lo que es esencial para estudiar partículas en potenciales periódicos, como en los cristales y superconductores [17].

En la química, la serie de Fourier se utiliza en espectroscopía para analizar las frecuencias de vibración de moléculas. Esto es clave para identificar compuestos químicos y estudiar sus propiedades dinámicas, como en la espectroscopía infrarroja y Raman [3].

Aplicación en la tecnología

La serie de Fourier es una herramienta matemática en áreas de la tecnología. Sirve para analizar señales en el dominio de la frecuencia y permite optimizar sistemas de telecomunicaciones, mejora la compresión de imágenes y audio, y desarrollar algoritmos en el procesamiento de señales digitales.

En el campo de las telecomunicaciones, la serie de Fourier sirve para la modulación y transmisión de señales. Se emplea en técnicas como la multiplexación por división de frecuencia (FDM) y la modulación por división de frecuencia ortogonal (OFDM), utilizadas en sistemas de comunicación inalámbrica como 4G, 5G y Wi-Fi [35].

Las series de Fourier también juegan un papel clave en el procesamiento de imágenes digitales. La transformada de Fourier se utiliza en la compresión de imágenes, como en el estándar JPEG, donde permite representar imágenes en términos de sus componentes de frecuencia y eliminar información redundante [15].

En el campo de la tecnología de audio, la serie de Fourier se utiliza para analizar y procesar señales de sonido. Por ejemplo, en la compresión de archivos de audio (como MP3), la transformada de Fourier permite descomponer la señal en sus componentes frecuenciales, lo que facilita la eliminación de frecuencias no perceptibles por el oído humano y, por tanto, reduce el

tamaño del archivo sin perder calidad apreciable [32].

En el ámbito de la tecnología médica, la serie de Fourier se aplica en dispositivos de diagnóstico por imagen, como los escáneres de resonancia magnética (MRI). Aquí, la transformada de Fourier convierte las señales captadas en el dominio de la frecuencia en imágenes detalladas del cuerpo humano, lo que permite diagnósticos más precisos y no invasivos [18].

En la tecnología de energía, la serie de Fourier es fundamental para el análisis de señales en sistemas de potencia. Permite identificar y corregir distorsiones armónicas en la red eléctrica, lo que mejora la calidad de la energía y previene fallos en equipos sensibles [26].

Aplicación en la ingeniería

La serie de Fourier es una herramienta matemática para diversas ramas de la ingeniería. Su capacidad para descomponer señales y analizar su comportamiento en el dominio de la frecuencia mejora el diseño y optimización de sistemas en ingeniería eléctrica, mecánica, civil y biomédica.

En ingeniería eléctrica, la serie de Fourier se utiliza para analizar y diseñar circuitos de corriente alterna (CA). La representación de señales en el dominio de la frecuencia facilita el estudio del comportamiento de filtros eléctricos, amplificadores y sistemas de comunicación [20]

En ingeniería mecánica, la serie de Fourier es fundamental en el estudio de vibraciones y el análisis modal de estructuras. Se utiliza para modelar y predecir el comportamiento dinámico de componentes mecánicos sometidos a cargas periódicas, como ejes rotativos, turbinas y motores [22]

En ingeniería biomédica, la serie de Fourier es esencial para el procesamiento de señales fisiológicas, como electrocardiogramas (ECG) y electroencefalogramas (EEG). Permite la eliminación de ruido en registros médicos y la detección de anomalías en señales biológicas [27]

En ingeniería de control, la serie de Fourier es fundamental para el análisis de sistemas dinámicos. Por ejemplo, en el diseño de controladores para robots o vehículos autónomos, la transformada de Fourier ayuda a analizar la respuesta en frecuencia del sistema, lo que permite optimizar su estabilidad y rendimiento [31].

En ingeniería civil, la serie de Fourier se utiliza para analizar cargas dinámicas en estructuras, como puentes y edificios. Al estudiar las frecuencias de las fuerzas aplicadas, los ingenieros pueden predecir cómo responderá la estructura a terremotos o vientos fuertes, lo que ayuda a mejorar su diseño y seguridad [12].

Aplicación en la Inteligencia Artificial

La serie de Fourier se ha utilizado en diversos campos, incluida la Inteligencia Artificial. Esto puede ser útil en aplicaciones como el reconocimiento de patrones, el procesamiento de imágenes y la optimización de modelos.

Las Redes Implícitas Neuronales (INR) utilizan perceptrones multicapa para representar funciones de alta frecuencia en dominios de baja dimensión. Recientemente, estas representaciones han logrado resultados destacados en tareas relacionadas con objetos y escenas 3D complejas [4].

El análisis armónico, que incluye las series de Fourier, se utiliza para comprender cómo las redes neuronales profundas aprenden tareas complejas. Por ejemplo, investigadores de la Universidad de Rice entrenaron una red neuronal profunda para reconocer flujos complejos de aire o agua y predecir sus cambios en el tiempo [11].

Además, las series de Fourier son esenciales en el procesamiento de señales dentro de la IA, permitiendo la eliminación de ruido, la detección de patrones y la mejora de la calidad de las señales [33]. Esta técnica es clave en aplicaciones como el reconocimiento de voz, la visión por computadora y la bioinformática, donde la extracción de características relevantes en el dominio de la frecuencia mejora la precisión de los modelos de aprendizaje automático.

En el análisis de series temporales, la serie de Fourier es empleada para descomponer datos en componentes frecuenciales, lo que ayuda a identificar tendencias y patrones periódicos. Esto es especialmente útil en aplicaciones de predicción, como el pronóstico del tiempo o la detección de anomalías en datos financieros [6].

En el aprendizaje profundo (deep learning), la transformada de Fourier se utiliza para acelerar operaciones de convolución en redes neuronales convolucionales (CNN). Aplicando la transformada rápida de Fourier (FFT), es posible reducir la complejidad computacional de estas operaciones, lo que permite entrenar modelos más grandes y complejos de manera eficiente [16].

En la optimización de modelos de IA, la serie de Fourier se utiliza para analizar y suavizar funciones de pérdida. Esto ayuda a identificar mínimos globales en lugar de mínimos locales, lo que mejora la convergencia de algoritmos de optimización como el descenso de gradiente [7].

Aplicación en el cómputo paralelo

El cómputo paralelo permite acelerar la ejecución de algoritmos dividiendo tareas en múltiples unidades de procesamiento. La serie de Fourier, junto con su transformada rápida (FFT), es ampliamente utilizada en la optimización de estos algoritmos, reduciendo el tiempo de cómputo en problemas como el análisis de señales, la dinámica de fluidos computacional (CFD) y la inteligencia artificial [24].

En visión por computadora, la serie de Fourier se utiliza para mejorar la eficiencia del procesamiento de imágenes mediante filtrado en el dominio de la frecuencia. La implementación de algoritmos FFT en GPU (Unidades de Procesamiento Gráfico) permite acelerar la detección de bordes, el reconocimiento de patrones y la compresión de imágenes [28].

En simulaciones numéricas, como en la dinámica de fluidos computacional (CFD), la serie de Fourier se emplea para resolver ecuaciones en diferencias finitas y para modelar turbulencias [10].

En la optimización de algoritmos paralelos, la serie de Fourier se emplea para diseñar técnicas de descomposición de problemas en subproblemas independientes que pueden resolverse en paralelo. Esto es especialmente útil en aplicaciones de aprendizaje automático distribuido, donde la FFT se utiliza para acelerar operaciones matriciales y de convolución en grandes redes neuronales [16].

En el análisis de big data, la transformada de Fourier se aplica en el procesamiento paralelo de grandes conjuntos de datos temporales o espaciales. Por ejemplo, en astronomía, la FFT se utiliza para analizar señales de telescopios de manera distribuida, lo que permite procesar petabytes de datos en tiempo récord [8].

2.4. Teoría de Procesos Hijos con la Llamada al Sistema fork() en Lenguaje C

En sistemas operativos basados en UNIX, la llamada al sistema fork() es un mecanismo fundamental para la creación de procesos. Esta función permite a un proceso existente (proceso padre) crear una copia casi idéntica de sí mismo (proceso hijo), estableciendo así un modelo de concurrencia que ha sido esencial en el desarrollo de aplicaciones multiproceso. La naturaleza de fork() establece que, tras su invocación, dos procesos separados continúan la ejecución del programa: el proceso original y su copia recién creada [39].

La singularidad de fork() radica en su comportamiento único de retorno: cuando es invocada correctamente, devuelve dos valores diferentes simultáneamente. En el proceso padre, retorna el identificador del proceso hijo (PID) recién creado, mientras que en el proceso hijo devuelve cero. Este mecanismo permite que ambos procesos determinen su rol y ejecuten código específico según su identidad. Si la llamada a fork() falla, retorna un valor negativo al proceso padre, indicando el error ocurrido [23].

Durante la creación del proceso hijo, el sistema operativo realiza una duplicación del espacio de memoria del proceso padre, incluyendo variables, descriptores de archivo y otros recursos. Sin embargo, esta duplicación implementa una estrategia de “copia en escritura”(copy-on-write), donde los segmentos de memoria realmente se duplican solo cuando uno de los procesos intenta modificarlos. Esta optimización mejora significativamente el rendimiento de la creación de procesos, especialmente en programas que realizan múltiples llamadas a fork() [5].

El sistema de procesos basado en fork() constituye un paradigma fundamental en la programación de sistemas paralelos y concurrentes en entornos UNIX. A diferencia de otros modelos de concurrencia, como los hilos, los procesos creados mediante fork() operan con espacios de memoria independientes, lo que proporciona un aislamiento natural entre las ejecuciones. Esta

característica resulta especialmente valiosa en aplicaciones donde la seguridad y la robustez son prioritarias, ya que un fallo en un proceso hijo no compromete la integridad del proceso padre [25].

Comunicación entre procesos padre e hijo

La comunicación entre procesos creados mediante `fork()` requiere mecanismos específicos de comunicación interproceso (IPC). Las tuberías (pipes) representan uno de los métodos más directos y eficientes para establecer canales de comunicación unidireccionales entre procesos relacionados. Mediante la llamada al sistema `pipe()`, se crean dos descriptores de archivo: uno para lectura y otro para escritura, permitiendo el intercambio de datos entre el proceso padre y sus hijos [39].

Los valores de retorno de `fork()` facilitan la implementación de estrategias de comunicación efectivas. El proceso padre, conociendo el PID del hijo, puede iniciar mecanismos de sincronización como señales (signals) para coordinar la ejecución. Por su parte, el proceso hijo puede utilizar el valor cero returned para adaptarse a su rol específico dentro del programa. Esta diferenciación permite diseñar arquitecturas de software donde cada proceso asume responsabilidades particulares dentro de una tarea global [40].

El concepto de jerarquía de procesos es fundamental en la programación con `fork()`. En sistemas UNIX, los procesos forman una estructura arbórea donde cada proceso, excepto el proceso `init` (PID 1), tiene un único parent. Esta organización jerárquica facilita operaciones como la recolección del estado de finalización de procesos hijos mediante las llamadas `wait()` y `waitpid()`, permitiendo al proceso parent monitorear y responder adecuadamente a la terminación de sus hijos [23].

La gestión adecuada de procesos hijos creados con `fork()` incluye la consideración de procesos "zombies" "huérfanos". Un proceso zombie ocurre cuando un proceso hijo termina pero su proceso parent no recoge su estado mediante `wait()`. Por otro lado, un proceso huérfano surge cuando el parent termina antes que el hijo, en cuyo caso el proceso `init` adopta automáticamente al hijo. Comprender estos escenarios es esencial para desarrollar aplicaciones multiproceso robustas que eviten fugas de recursos del sistema [25].

Aplicaciones prácticas de `fork()`

En servidores de red concurrentes, la llamada `fork()` es ampliamente utilizada para manejar múltiples conexiones simultáneas. El proceso principal acepta conexiones entrantes y crea un proceso hijo para atender cada cliente, permitiendo que el servidor continúe aceptando nuevas conexiones sin bloquearse. Este modelo, conocido como "prefork", ha sido implementado en servidores web como Apache HTTP Server, demostrando su eficacia en escenarios de alta concurrencia [19].

Los intérpretes de comandos o shells en sistemas UNIX utilizan `fork()` para ejecutar comandos externos. Cuando un usuario introduce un comando, el shell crea un proceso hijo mediante `fork()`

y luego utiliza exec() para reemplazar la imagen del proceso hijo con el programa solicitado. Esta combinación fork-exec constituye el paradigma fundamental para la ejecución de programas en sistemas UNIX, permitiendo al shell mantener su estado mientras ejecuta comandos externos [36].

En aplicaciones de procesamiento paralelo, fork() permite dividir tareas computacionalmente intensivas entre múltiples procesos. Cada proceso hijo puede trabajar independientemente en un subconjunto de los datos, aprovechando los sistemas multiprocesador modernos. Este enfoque se utiliza frecuentemente en análisis científicos, procesamiento de imágenes y otras aplicaciones que pueden beneficiarse de la paralelización de tareas [34].

La implementación de demonios (daemons) en sistemas UNIX típicamente involucra múltiples llamadas a fork(). Un demonio es un proceso que se ejecuta en segundo plano, independiente de cualquier terminal. El proceso de "daemonización" generalmente implica llamar a fork() dos veces: la primera para permitir que el padre original termine (desvinculándose de la terminal), y la segunda para evitar que el demonio se convierta en líder de sesión. Este patrón es fundamental en la implementación de servicios del sistema que operan continuamente en segundo plano [38].

2.5. Teoría sobre Memoria Compartida en Lenguaje C

La memoria compartida representa uno de los mecanismos más eficientes para la comunicación interproceso (IPC) en sistemas operativos UNIX y POSIX. A diferencia de otros métodos de IPC como tuberías o colas de mensajes, la memoria compartida permite que múltiples procesos accedan directamente a una región común del espacio de direcciones físicas, eliminando así la necesidad de copiar datos entre espacios de procesos. Esta característica la convierte en la opción preferida cuando se requiere un alto rendimiento en el intercambio de grandes volúmenes de datos [39].

En el estándar POSIX, la memoria compartida se implementa a través de un conjunto de funciones que incluyen `shm_open()`, `mmap()`, y `shm_unlink()`. Estas funciones permiten crear, mapear y destruir segmentos de memoria compartida con una interfaz consistente y portable. En sistemas UNIX tradicionales, las funciones `shmget()`, `shmat()`, `shmdt()` y `shmctl()` proporcionan funcionalidad similar pero con un enfoque basado en el System V IPC, que sigue siendo ampliamente utilizado en muchas implementaciones [23].

La creación de un segmento de memoria compartida en C implica un proceso de dos pasos: primero, la obtención de un identificador para el segmento mediante `shmget()` o la apertura de un objeto de memoria compartida con `shm_open()`; y segundo, la asociación de este segmento al espacio de direcciones del proceso mediante `shmat()` o `mmap()`. Este proceso establece una correspondencia entre un espacio de memoria física y las direcciones virtuales en los espacios de cada proceso participante [36].

El uso eficaz de la memoria compartida requiere una cuidadosa gestión para evitar condiciones de carrera y asegurar la coherencia de los datos. Dado que múltiples procesos pueden leer y

escribir simultáneamente en la región compartida, es necesario implementar mecanismos de sincronización como semáforos o mutexes para coordinar el acceso. Sin una sincronización adecuada, los procesos podrían observar estados intermedios o inconsistentes de los datos, lo que conduciría a comportamientos impredecibles en la aplicación [40].

Implementación de memoria compartida POSIX

La API de memoria compartida POSIX ofrece una interfaz moderna y portable para la gestión de regiones de memoria compartida entre procesos. La función `shm_open()` crea o abre un objeto de memoria compartida, devolviendo un descriptor de archivo que puede ser utilizado con `ftruncate()` para ajustar su tamaño. Posteriormente, `mmap()` establece la correspondencia entre este objeto y el espacio de direcciones virtuales del proceso, permitiendo el acceso directo a los datos compartidos [38].

Una ventaja significativa de la interfaz POSIX es su integración con el sistema de archivos. Los objetos de memoria compartida se representan como archivos en el sistema de archivos `/dev/shm`, facilitando su gestión y monitoreo. Esta característica también permite aplicar permisos de acceso estándar para controlar qué procesos pueden acceder al segmento compartido, proporcionando así un nivel adicional de seguridad y control [25].

La liberación de recursos en el enfoque POSIX se realiza mediante las funciones `munmap()` para desvincular el mapeo del espacio de direcciones del proceso, y `shm_unlink()` para eliminar el objeto de memoria compartida cuando ya no es necesario. Es crucial que las aplicaciones liberen adecuadamente estos recursos para evitar fugas de memoria y mantener la integridad del sistema, especialmente en programas de larga duración [23].

El modelo de memoria compartida POSIX ofrece compatibilidad con otras características del estándar POSIX, como los hilos `pthread` y mecanismos de sincronización como `mutex` y variables de condición. Esta coherencia en el diseño de la API facilita el desarrollo de aplicaciones que combinan múltiples técnicas de concurrencia y comunicación interproceso, adaptándose a las necesidades específicas de cada situación [9].

Implementación de memoria compartida System V

El mecanismo de memoria compartida System V, introducido originalmente en UNIX System V, proporciona una interfaz robusta que ha permanecido estable durante décadas. La función `shmget()` crea o accede a un segmento de memoria compartida identificado por una clave IPC, generalmente generada mediante la función `ftok()`. Este enfoque basado en claves permite que procesos no relacionados localicen y utilicen el mismo segmento compartido sin requerir un ancestro común [39].

Una característica distintiva de la memoria compartida System V es su integración con la infraestructura general de IPC del sistema, que incluye semáforos y colas de mensajes. Esta cohesión facilita la implementación de soluciones completas de comunicación interproceso utilizando un conjunto consistente de herramientas y conceptos. Además, las herramientas de administración

del sistema como ipcs e ipcrm permiten monitorear y gestionar los recursos de IPC, incluyendo segmentos de memoria compartida [36].

El modelo de persistencia de la memoria compartida System V difiere significativamente del enfoque POSIX. Los segmentos creados mediante shmget() persisten en el sistema hasta que son explícitamente eliminados con shmctl(IPC_RMID) o hasta que el sistema se reinicia. Esta persistencia puede ser ventajosa para servicios de larga duración, pero también requiere una gestión cuidadosa para evitar la acumulación de segmentos abandonados que consumen recursos del sistema [25].

La asignación y liberación de la memoria compartida en System V se gestiona mediante las funciones shmat() y shmdt(). La primera asocia el segmento al espacio de direcciones del proceso, permitiendo el acceso directo a los datos compartidos, mientras que la segunda desvincula el segmento cuando ya no es necesario. Es importante destacar que shmdt() no elimina el segmento del sistema, solo termina el acceso del proceso actual a ese segmento [40].

Consideraciones prácticas en el uso de memoria compartida

La alineación de datos en memoria compartida es un aspecto crítico que afecta tanto al rendimiento como a la corrección del programa. Las estructuras de datos que contienen tipos de diferentes tamaños pueden tener requisitos de alineación específicos que varían entre arquitecturas. Para garantizar la portabilidad, los desarrolladores deben utilizar técnicas como el relleno explícito de estructuras o emplear directivas de compilador como #pragma pack para controlar la alineación [14].

La sincronización de caché entre múltiples procesadores representa un desafío adicional en sistemas multicore. Cuando procesos ejecutándose en diferentes núcleos acceden a la misma región de memoria compartida, pueden ocurrir inconsistencias debido a las cachés locales de cada procesador. Los sistemas modernos implementan protocolos de coherencia de caché para mitigar este problema, pero los desarrolladores deben ser conscientes de estas implicaciones, especialmente en aplicaciones de alto rendimiento [21].

La gestión del ciclo de vida de los segmentos de memoria compartida es crucial para evitar fugas de recursos y comportamientos indefinidos. En aplicaciones de producción, es esencial implementar mecanismos robustos para la limpieza de recursos, incluso en casos de terminación anormal del programa. Esto puede incluir el uso de manejadores de señales o procesos monitores que aseguren la liberación adecuada de los segmentos compartidos cuando los procesos principales terminan inesperadamente [23].

El rendimiento de la memoria compartida depende significativamente del patrón de acceso y la localidad de los datos. Para maximizar la eficiencia, los desarrolladores deben diseñar sus estructuras de datos para favorecer patrones de acceso que minimicen los fallos de caché y la contención entre procesos. Esto incluye técnicas como el particionamiento de datos, el padding para evitar falsos compartidos (false sharing) y la agrupación de datos frecuentemente accedidos

juntos para mejorar la localidad espacial [14].

2.6. Teoría sobre Semáforos en Lenguaje C

Los semáforos son primitivas de sincronización fundamentales en sistemas operativos, introducidos originalmente por Edsger Dijkstra en 1965 como una solución elegante al problema de la exclusión mutua en entornos concurrentes. En esencia, un semáforo es un contador protegido que puede ser incrementado mediante la operación signal (o V) y decrementado mediante la operación wait (o P), con la restricción de que cuando el contador llega a cero, cualquier intento adicional de decrementarlo bloqueará al proceso hasta que otro proceso incremente el contador [40].

En la programación en C, existen dos interfaces principales para trabajar con semáforos: la interfaz POSIX (IEEE Std 1003.1) y la interfaz System V. Ambas implementaciones proporcionan funcionalidad similar pero con diferencias sintácticas y semánticas significativas. La interfaz POSIX, con funciones como `sem_open()`, `sem_wait()` y `sem_post()`, ofrece una API más moderna y portable, mientras que la interfaz System V con `semget()`, `semop()` y `semctl()` conserva su relevancia histórica y sigue siendo ampliamente utilizada [39].

Los semáforos se clasifican generalmente en dos tipos: semáforos binarios y semáforos contadores. Los semáforos binarios, también conocidos como mutex, solo pueden tomar los valores 0 y 1, y se utilizan principalmente para proteger secciones críticas de código garantizando el acceso exclusivo. Por otro lado, los semáforos contadores pueden tomar valores enteros no negativos arbitrarios y son útiles para gestionar recursos limitados, como conexiones a bases de datos o elementos en un buffer de tamaño fijo [23].

La correcta implementación y uso de semáforos es crucial para evitar problemas clásicos de concurrencia como interbloqueos (deadlocks), inanición (starvation) y condiciones de carrera (race conditions). Un interbloqueo puede ocurrir cuando dos o más procesos esperan indefinidamente por recursos que están siendo retenidos entre ellos, mientras que la inanición sucede cuando un proceso nunca obtiene acceso al recurso debido a la continua priorización de otros procesos. La programación defensiva con semáforos requiere un diseño cuidadoso y la aplicación de técnicas como la adquisición ordenada de recursos para evitar estos problemas [21].

Semáforos POSIX

La API de semáforos POSIX está diseñada para integrarse coherentemente con otras facilidades de concurrencia POSIX, como hilos (`pthreads`) y memoria compartida. Los semáforos POSIX se dividen en dos categorías: semáforos con nombre (named semaphores), creados mediante `sem_open()` y vinculados a un nombre en el sistema de archivos, y semáforos sin nombre (unnamed semaphores), inicializados con `sem_init()` en una región de memoria compartida entre procesos relacionados [9].

Las operaciones fundamentales sobre semáforos POSIX incluyen `sem_wait()`, que decremente

el valor del semáforo o bloquea si el valor es cero; `sem_post()`, que incrementa el valor del semáforo y potencialmente desbloquea procesos en espera; y `sem_trywait()`, que proporciona una versión no bloqueante de `sem_wait()`. Estas operaciones son atómicas, garantizando que no ocurrirán interrupciones durante su ejecución que puedan comprometer la integridad del semáforo [38].

Un aspecto valioso de los semáforos POSIX es su capacidad para implementar diversos patrones de sincronización. Por ejemplo, utilizando múltiples semáforos es posible construir barreras de sincronización, donde varios procesos esperan hasta que todos hayan alcanzado un punto determinado antes de continuar. También pueden implementarse soluciones al problema del productor-consumidor, donde un proceso produce datos que otro consume, requiriendo coordinación para evitar desbordamientos o subdesbordamientos en el buffer compartido [25].

La limpieza adecuada de los recursos de semáforos es esencial para evitar fugas. Para semáforos con nombre, `sem_close()` libera los recursos asociados en el proceso actual, mientras que `sem_unlink()` elimina el semáforo del sistema. Para semáforos sin nombre, `sem_destroy()` libera los recursos cuando el semáforo ya no es necesario. Estas operaciones deben realizarse meticulosamente, especialmente en aplicaciones de larga duración o en sistemas con recursos limitados [23].

Semáforos System V

Los semáforos System V ofrecen una interfaz robusta con capacidades avanzadas, aunque con una sintaxis más compleja que su contraparte POSIX. Una característica distintiva es que los semáforos System V se crean en conjuntos mediante la función `semget()`, permitiendo la manipulación eficiente de múltiples semáforos relacionados como una unidad. Cada conjunto de semáforos se identifica por una clave única, generalmente generada mediante la función `ftok()` [39].

La operación sobre semáforos System V se realiza principalmente a través de la función `semop()`, que permite ejecutar múltiples operaciones en diferentes semáforos del conjunto de forma atómica. Esta atomicidad a nivel de conjunto es particularmente valiosa para implementar protocolos de sincronización complejos, como la prevención de interbloqueos mediante la adquisición simultánea de múltiples recursos. Además, `semop()` admite la especificación de banderas como `SEM_UNDO`, que automáticamente revierte las operaciones si el proceso termina mientras aún mantiene recursos [36].

El control y administración de los semáforos System V se realiza mediante la función `semctl()`, que proporciona una amplia gama de operaciones como inicialización (`SETVAL`), consulta de valores (`GETVAL`), y eliminación de conjuntos de semáforos (`IPC_RMID`). Esta función también permite manipular atributos específicos del sistema IPC, como permisos de acceso y propietarios, facilitando la implementación de políticas de seguridad en entornos multiusuario [25].

Una consideración importante con los semáforos System V es su persistencia en el sistema. A diferencia de muchos otros recursos que se liberan automáticamente cuando un proceso termina, los conjuntos de semáforos System V persisten hasta que son explícitamente eliminados con semctl(IPC_RMID) o hasta que el sistema se reinicia. Esta característica puede ser beneficiosa para implementar servicios persistentes, pero también requiere una gestión diligente para evitar la acumulación de recursos huérfanos [40].

Patrones de uso de semáforos

El patrón de exclusión mutua es quizás el uso más común de los semáforos, garantizando que solo un proceso a la vez pueda acceder a un recurso compartido o ejecutar una sección crítica. La implementación consiste en inicializar un semáforo con valor 1, ejecutar sem_wait() antes de entrar a la sección crítica y sem_post() al salir. Este patrón es fundamental para mantener la consistencia de los datos compartidos y prevenir condiciones de carrera [21].

El problema clásico del productor-consumidor ilustra el uso de semáforos para coordinar procesos con roles complementarios. En esta solución, tres semáforos gestionan la sincronización: uno para la exclusión mutua en el acceso al buffer, otro para contar los espacios disponibles y un tercero para contar los elementos en el buffer. Este patrón se aplica ampliamente en sistemas con procesamiento por lotes, colas de mensajes y pipelines de procesamiento [37].

La implementación de barreras de sincronización representa otro patrón común, donde varios procesos deben alcanzar un punto específico antes de que cualquiera pueda continuar. Utilizando un semáforo combinado con un contador protegido, es posible crear un mecanismo donde N procesos esperan hasta que todos hayan llegado a la barrera. Este patrón es especialmente útil en computación paralela, donde diferentes etapas de un algoritmo requieren la finalización completa de etapas anteriores antes de continuar [34].

Los semáforos también pueden implementar soluciones al problema de los lectores-escritores, donde múltiples procesos pueden leer simultáneamente un recurso compartido, pero la escritura requiere acceso exclusivo. Esta implementación utiliza típicamente múltiples semáforos para rastrear el número de lectores activos y controlar el acceso de los escritores. La correcta gestión de estos semáforos es crucial para evitar la inanición de escritores, un problema común donde los escritores nunca obtienen acceso debido a un flujo continuo de lectores [37].

3

Soluciones individuales de series de Fourier

En este apartado se presentan las soluciones individuales de cada estudiante, desarrolladas a partir de las funciones propuestas por el profesor de la materia de Cómputo Paralelo. El objetivo principal de esta fase fue llevar a cabo el cálculo de la serie de Fourier correspondiente a cada función asignada, aplicando los métodos matemáticos adecuados para la obtención de los coeficientes y la expansión de la función en términos de senos y cosenos.

Cada integrante del equipo realizó los cálculos de manera independiente, asegurando un desarrollo detallado y estructurado de los coeficientes de Fourier. Para ello, se siguió el procedimiento estándar de integración para la obtención de los coeficientes a_0 , a_n y b_n , que determinan la representación trigonométrica de la función en términos de series de Fourier.

3.1. Función resuelta por Castro Paez Ilse Yazbeth

En este siguiente apartado, se presentan los cálculos para de la serie de Fourier de la función $f(x) = 9 - 3x - x^2$, correspondientes a la figura 3.1, donde se muestran detalladamente el desarrollo hasta obtener el resultado.

Funcióñ: $f(x) = \int_{-\pi}^{\pi} (9 - 3x - x^2) dx$

Identidad de ejes:

$$\begin{aligned} \text{Sen } n\pi &= 0 \\ \text{Sen } -n\pi &= -\text{Sen } n\pi \\ \text{Cos } -n\pi &= \text{Cos } n\pi \\ \text{Cos } n\pi &= (-1)^n \end{aligned}$$

$(-\pi, \pi) = [-\pi, \pi]$

Para $a_0 \rightarrow \frac{1}{\pi} \int_{-\pi}^{\pi} (9 - 3x - x^2) dx$

$$\begin{aligned} &= \frac{1}{\pi} \left(\int_{-\pi}^{\pi} 9 dx - \int_{-\pi}^{\pi} 3x dx - \int_{-\pi}^{\pi} x^2 dx \right) \\ \rightarrow \int_{-\pi}^{\pi} 9 dx + 9x \Big|_{-\pi}^{\pi} &= 9(\pi) - 9(-\pi) - 9\pi + 9\pi = 18\pi \\ \rightarrow \int_{-\pi}^{\pi} 3x dx = 3 \frac{x^2}{2} \Big|_{-\pi}^{\pi} &= 3 \left[\frac{\pi^2}{2} - \frac{(-\pi)^2}{2} \right] = 0 \\ \rightarrow \int_{-\pi}^{\pi} x^2 dx = 2 \frac{x^3}{3} \Big|_{-\pi}^{\pi} &= 2 \left[\frac{\pi^3}{3} - \frac{(-\pi)^3}{3} \right] = \frac{4\pi^3}{3} \\ &= \frac{1}{\pi} (18\pi - 0 - \frac{4\pi^3}{3}) = \frac{1}{\pi} (18\pi - \frac{4\pi^3}{3}) = 18 - \frac{4\pi^2}{3} \end{aligned}$$

Para $a_n \rightarrow \frac{1}{\pi} \int_{-\pi}^{\pi} (9 - 3x - x^2) \cos(nx) dx$

$$\begin{aligned} &= \frac{1}{\pi} \int_{-\pi}^{\pi} (9 - 3x - x^2) \cos(nx) dx \\ &= \frac{1}{\pi} \int_{-\pi}^{\pi} (9 \cos(nx) dx - 3x \cos(nx) dx - x^2 \cos(nx) dx) \\ \rightarrow \int_{-\pi}^{\pi} 9 \cos(nx) dx = 9 \left[\frac{\text{Sen}(nx)}{n} \right] \Big|_{-\pi}^{\pi} &= \left[\frac{9 \text{Sen}(n\pi)}{n} \right] - \left[\frac{9 \text{Sen}(-n\pi)}{n} \right] = \left[\frac{9 \text{Sen}(n\pi)}{n} \right] - \left[\frac{9 \text{Sen}(-n\pi)}{n} \right] = \frac{18 \text{Sen}(n\pi)}{n} = 0 \\ \rightarrow \int_{-\pi}^{\pi} 3x \cos(nx) dx = \int_{-\pi}^{\pi} uv - v du \Big|_{-\pi}^{\pi} &= 3x \left[\frac{\text{Sen}(nx)}{n} \right] \Big|_{-\pi}^{\pi} - \left[\frac{\text{Sen}(nx)}{n} \right] \Big|_{-\pi}^{\pi} = \left[3\pi \frac{\text{Sen}(\pi)}{n} - 3(-\pi) \frac{\text{Sen}(-\pi)}{n} \right] = \left[\frac{3\pi \text{Sen}(\pi)}{n} - 3(-\pi) \frac{\text{Sen}(-\pi)}{n} \right] = \left[\frac{3\pi \text{Sen}(\pi)}{n} - 3(-\pi) \frac{\text{Sen}(\pi)}{n} \right] = 0 \\ \rightarrow \int_{-\pi}^{\pi} x^2 \cos(nx) dx = \int_{-\pi}^{\pi} uv - v du \Big|_{-\pi}^{\pi} &= x^2 \left[\frac{\text{Sen}(nx)}{n} \right] \Big|_{-\pi}^{\pi} - \frac{2}{n} \int_{-\pi}^{\pi} \text{Sen}(nx) x dx = \text{Multiplicando por } -\frac{2}{n} \text{ el coeficiente de } x \text{ de } x = \frac{-2\pi^2(-1)^{n+1}}{n^2} = \frac{2\pi^2(-1)^{n+1}}{n^2} \\ \text{Juntando las integrales} \rightarrow \frac{1}{\pi} \left(\frac{2\pi^2(-1)^{n+1}}{n^2} \right) &= \frac{2(-1)^{n+1}}{n^2} \end{aligned}$$

Para $b_n \rightarrow \frac{1}{\pi} \int_{-\pi}^{\pi} (9 - 3x - x^2) \text{Sen}(nx) dx$

$$\begin{aligned} &= \frac{1}{\pi} \int_{-\pi}^{\pi} (9 - 3x - x^2) \text{Sen}(nx) dx \\ &= \frac{1}{\pi} \int_{-\pi}^{\pi} (9 \text{Sen}(nx) dx - 3x \text{Sen}(nx) dx - x^2 \text{Sen}(nx) dx) \\ \rightarrow \int_{-\pi}^{\pi} 9 \text{Sen}(nx) dx = 9 \left[\frac{-\text{Cos}(nx)}{n} \right] \Big|_{-\pi}^{\pi} &= -9 \left[\frac{\text{Cos}(n\pi)}{n} + \frac{\text{Cos}(-n\pi)}{n} \right] = -9 \left[\frac{\text{Cos}(n\pi)}{n} + \frac{\text{Cos}(n\pi)}{n} \right] = 0 \\ \rightarrow \int_{-\pi}^{\pi} 3x \text{Sen}(nx) dx = \int_{-\pi}^{\pi} uv - v du \Big|_{-\pi}^{\pi} &= 3x \left[\frac{-\text{Cos}(nx)}{n} \right] \Big|_{-\pi}^{\pi} - \left[\frac{-\text{Cos}(nx)}{n} \right] \Big|_{-\pi}^{\pi} = -3\pi \frac{\text{Cos}(\pi)}{n} + 3(-\pi) \frac{\text{Cos}(-\pi)}{n} = -3\frac{\pi \text{Cos}(\pi)}{n} + 3\frac{\pi \text{Cos}(\pi)}{n} = 0 \\ \rightarrow \int_{-\pi}^{\pi} x^2 \text{Sen}(nx) dx = \int_{-\pi}^{\pi} uv - v du \Big|_{-\pi}^{\pi} &= x^2 \left[\frac{-\text{Cos}(nx)}{n} \right] \Big|_{-\pi}^{\pi} - \frac{2}{n} \int_{-\pi}^{\pi} x \text{Cos}(nx) dx = \left[\frac{-\pi^2 \text{Cos}(\pi)}{n} + \frac{2\pi \text{Cos}(\pi)}{n} \right] + \left[\frac{2}{n} \cdot 0 \right] = 0 \\ \text{Juntando las integrales} \rightarrow \frac{1}{\pi} (0 - 0 - 0) &= 0 \end{aligned}$$

$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \text{Cos} \frac{n\pi}{\pi} x + b_n \text{Sen} \frac{n\pi}{\pi} x)$

$$\begin{aligned} &= \frac{18 - 2\pi^2}{2} + \sum_{n=1}^{\infty} \left(\frac{2(-1)^{n+1}}{n^2} \right) \left(\text{Cos} \frac{n\pi}{\pi} x \right) + (0) \left(\text{Sen} \frac{n\pi}{\pi} x \right) \\ &= 9 - \frac{\pi^2}{3} + \sum_{n=1}^{\infty} \frac{2(-1)^{n+1}}{n^2} \cdot \text{Cos}(nx) \end{aligned}$$

Figura 3.1: Cálculo de la función $f(x) = 9 - 3x - x^2$, resuelto por Castro Paez Ilse Yazbeth

3.2. Función resuelta por Catonga Tecla Daniel Isaí

En el siguiente apartado, se presentan los cálculos para la obtención de los coeficientes de Fourier de la función asignada, $f(x) = 6 - 2x$, correspondientes a las figuras 3.2 hasta 3.5. Se detallan las ecuaciones utilizadas y el procedimiento matemático seguido para su determinación.

Función: $f(x) = 6 - 2x$

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

Cof. de fourier

$$a_0 = \frac{1}{P} \int_{-P}^P f(x) dx$$

$$a_n = \frac{1}{P} \int_{-P}^P f(x) \cos(\frac{n\pi}{P}x) dx \quad P = \pi \Rightarrow (-\pi, \pi)$$

$$b_n = \frac{1}{P} \int_{-P}^P f(x) \sin(\frac{n\pi}{P}x) dx$$

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} (6 - 2x) dx$$

$$= \frac{1}{\pi} [6x - x^2]_{-\pi}^{\pi}$$

$$= \frac{1}{\pi} ([6\pi - \pi^2] - [-6\pi - (-\pi)^2])$$

$$= \frac{1}{\pi} (6\pi - \pi^2 + 6\pi + \pi^2)$$

$$= \frac{12\pi}{\pi}$$

$a_0 = 12$

Figura 3.2: Cálculo de a_0 para $f(x) = 6 - 2x$, resuelto por Catonga Tecla Daniel Isaí

$$\begin{aligned}
 G_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} (6 - 2x) \cos\left(\frac{n\pi}{\pi} x\right) dx \\
 &= \frac{1}{\pi} \int_{-\pi}^{\pi} 6 \cos(nx) dx - \frac{1}{\pi} \int_{-\pi}^{\pi} 2x \cos(nx) dx
 \end{aligned}$$

Por partes

$$\begin{aligned}
 \textcircled{1} \quad \frac{1}{\pi} \int_{-\pi}^{\pi} 6 \cos(nx) dx &\Rightarrow \frac{1}{\pi} \int_{-\pi}^{\pi} 6 \cos u \frac{du}{n} \\
 u &= nx \\
 du &= n dx \\
 \frac{du}{n} &= dx \\
 &= \frac{1}{n\pi} \left[6 \sin(nx) \right]_{-\pi}^{\pi} \\
 &= \frac{1}{n\pi} \left[6 \sin(\pi n) - 6 \sin(-\pi n) \right] \\
 \sin(n\pi) &= 0 \\
 \sin(-n\pi) &= -\sin(n\pi)
 \end{aligned}$$

$$\begin{aligned}
 \textcircled{2} \quad -\frac{1}{\pi} \int_{-\pi}^{\pi} 2x \cos(nx) dx &\quad u = x \quad \frac{du}{dx} = \cos(nx) \\
 &\quad v = \frac{\sin(nx)}{n} \quad \frac{dv}{dx} = \frac{\cos(nx)}{n} \\
 &= u \cdot v + \int \frac{\sin(nx)}{n} dx \\
 &= x \frac{\sin(nx)}{n} + \frac{1}{n} \int \sin(nx) dx \\
 &= x \frac{\sin(nx)}{n} + \frac{1}{n^2} \int \sin w dw \quad w = nx \quad dw = n dx \\
 &\quad \frac{dw}{n} = dx
 \end{aligned}$$

Figura 3.3: Desarrollo parcial del cálculo de los coeficientes a_n para la función $f(x) = 6 - 2x$, resuelto por Catonga Tecla Daniel Isaí. El resultado final aún no se muestra.

$$\begin{aligned}
 & -\frac{2}{\pi} \left[\frac{x \sin(nx)}{n} \Big|_0^\pi - \frac{1}{n^2} \cos(nx) \Big|_0^\pi \right]_{-\pi}^\pi \\
 & = 0 \\
 \underline{a_n} &= 0 + 0 = 0 \quad \cancel{\text{X}} \\
 \\
 b_n &= -\frac{1}{\pi} \int_{-\pi}^{\pi} (6 - 2x) \sin\left(\frac{n\pi}{\pi} x\right) dx \\
 &= \frac{1}{\pi} \int_{-\pi}^{\pi} 6 \sin(nx) dx - \frac{1}{\pi} \int_{-\pi}^{\pi} 2x \sin(nx) dx \\
 \textcircled{1} \\
 & \frac{6}{\pi} \int_{-\pi}^{\pi} \sin(nx) dx \Rightarrow \frac{6}{\pi} \int_{-\pi}^{\pi} \sin u \frac{du}{n} = \frac{6}{\pi n} \int_{-\pi}^{\pi} \sin u du \\
 & u = nx \\
 & \frac{du}{n} = dx \\
 & = -\frac{6}{\pi n} \left[\cos(nx) \right]_{-\pi}^{\pi} \\
 \\
 \textcircled{2} \\
 & -\frac{1}{\pi} \int_{-\pi}^{\pi} 2x \sin(nx) dx \quad u = 2x \quad du = \sin(nx) \\
 & \quad \quad \quad \quad \quad \quad \quad v = -\frac{1}{n} \cos(nx) \\
 & = -\frac{1}{\pi} \left[2x \left(-\frac{1}{n} \cos(nx) \right) - \int -\frac{1}{n} \cos(nx) 2 dx \right] \\
 & = \frac{1}{\pi} \left[-\frac{2x}{n} \cos(nx) + \frac{2}{n} \int \cos(nx) dx \right] \\
 & \Rightarrow \int \cos(nx) dx \quad z = nx \Rightarrow \int \cos z dz \quad \frac{dz}{dx} = \frac{ndz}{dx} \\
 & \int \cos z \frac{dz}{n} \Rightarrow \frac{1}{n} \int \cos z dz \\
 & \Rightarrow \frac{1}{n} \sin(nx)
 \end{aligned}$$

Figura 3.4: Cálculo de a_n y desarrollo parcial de b_n para la función $f(x) = 6 - 2x$, resuelto por Catonga Tecla Daniel Isai.

$$\begin{aligned}
 &= -\frac{1}{\pi} \left(-\frac{2x}{n} \cos(nx) + \frac{2}{n^2} \sin(nx) \right) \Big|_{-\pi}^{\pi} \\
 &= -\frac{2}{\pi n} \left[-x \cos(nx) + \frac{1}{n} \overset{\nearrow 0}{\cancel{\sin(nx)}} \right]_{-\pi}^{\pi} \\
 &= -\frac{2}{\pi n} (-x \cos(n\pi) + x \cos(-n\pi)) \\
 &= -\frac{2}{\pi n} (-\pi \cos(n\pi) + (-\pi) \cos(-n\pi)) \\
 &= -\frac{2}{\pi n} (-2\pi \cos(n\pi)) \\
 &= \frac{4}{n} \cos(n\pi) = \underbrace{\frac{4}{n} (-1)^n}_{\cancel{+}} \\
 f(x) &= \frac{12}{2} + \sum_{n=1}^{\infty} \left(0 \cdot \cos\left(\frac{n\pi}{\pi} x\right) + \frac{4}{n} (-1)^n \sin\left(\frac{n\pi}{\pi} x\right) \right) \\
 f(x) &= 6 + \sum_{n=1}^{\infty} \left(\frac{4}{n} (-1)^n \sin(nx) \right) \quad \cancel{\left[\text{sin}(nx) \right]}
 \end{aligned}$$

Figura 3.5: Cálculo de b_n para la función $f(x) = 6 - 2x$, mostrando la función reconstruida a partir de los coeficientes, resuelto por Catonga Tecla Daniel Isaí.

3.3. Función resuelta por Padilla Sanchez Haniel

En esta sección, se expone el procedimiento matemático para determinar los coeficientes de Fourier de la función $f(x) = 6 - 4x$. Se inicia con la ecuación general de la serie de Fourier y se procede al cálculo del coeficiente a_0 como se muestra en la figura 3.6. Para la obtención de a_n , se emplea integración por partes, evidenciando la cancelación de ciertos términos. Se incluyen además observaciones sobre las propiedades trigonométricas utilizadas en el análisis.

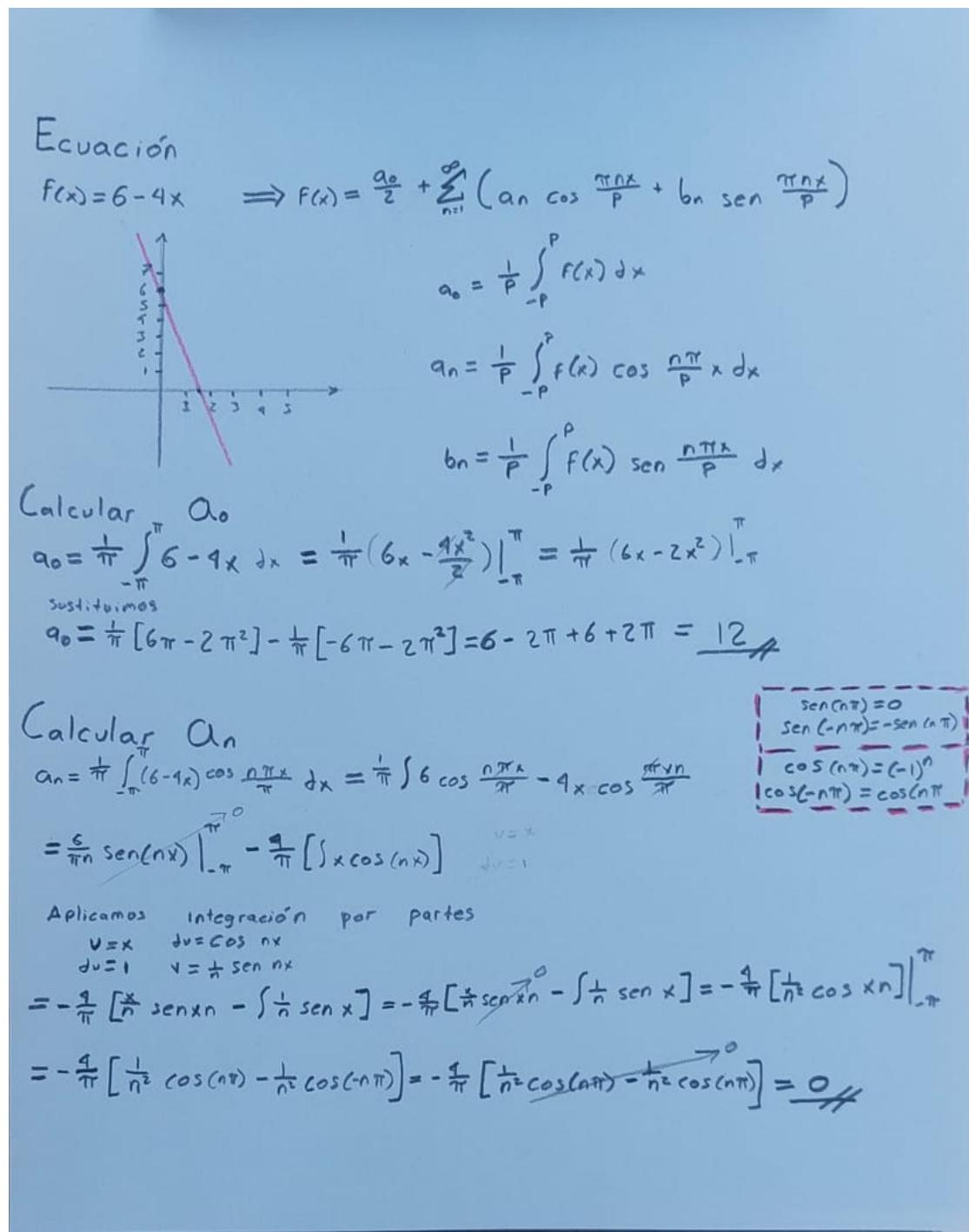


Figura 3.6: Cálculo de a_0 y a_n para la función $f(x) = 6 - 4x$, resuelto por Padilla Sánchez Haniel.

En el siguiente apartado, se presentan los cálculos para la obtención del coeficiente b_n de la serie

de Fourier de la función $f(x) = 6 - 4x$. Se muestra el desarrollo de la integral correspondiente y la aplicación del método de integración por partes para resolver términos específicos como se muestra en la figura 3.7. Finalmente, se obtiene una expresión general para b_n , la cual se utilizará en la reconstrucción de la función periódica.

Calcular b_n

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin \frac{n\pi x}{\pi} dx$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} (6 - 4x) \sin \frac{n\pi x}{\pi} dx = \frac{1}{\pi} \int_{-\pi}^{\pi} 6 \sin(n\pi x) dx - (4x) \sin(n\pi x) \Big|_{-\pi}^{\pi}$$

$$b_n = \frac{6}{\pi} \left[-\cos(n\pi x) \frac{1}{n} \right]_{-\pi}^{\pi} - \frac{4}{\pi} \int x \sin(n\pi x) dx$$

$\begin{array}{l} u=x \quad dv=\sin(n\pi x) \\ du=1 \quad v=-\frac{1}{n}\cos(n\pi x) \end{array}$

del lado Derecho se integra por partes

$$b_n = \frac{6}{\pi} \left[-\cos(n\pi) + \cos(-n\pi) \right] - \frac{4}{\pi} \left(-\frac{x}{n} \cos(n\pi) + \frac{1}{n} \int \cos(n\pi x) dx \right)$$

$$b_n = -\frac{4}{\pi} \left(-\frac{x}{n} \cos(n\pi) + \left[\frac{1}{n^2} \sin(n\pi x) \right]_{-\pi}^{\pi} \right) = -\frac{4}{\pi} \left(-\frac{\pi}{n} \cos(n\pi) - \frac{\pi}{n} \cos(n\pi) \right)$$

$$b_n = \frac{8}{n} \cos(n\pi) = \underline{\underline{\frac{8}{n} (-1)^n}}$$

$$f(x) = 6 - 4x = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos \frac{n\pi x}{\pi} + b_n \sin \frac{n\pi x}{\pi})$$

$$f(x) = \frac{12}{2} + \sum_{n=1}^{\infty} \left(\frac{8}{n} (-1)^n \cos \frac{n\pi x}{\pi} + \left(\frac{8}{n} (-1)^n \right) \sin \frac{n\pi x}{\pi} \right)$$

$$\underline{\underline{f(x) = 6 + \sum_{n=1}^{\infty} \left(\frac{8}{n} (-1)^n \cdot \sin \frac{n\pi x}{\pi} \right)}}$$

Figura 3.7: Cálculo de b_n para la función $f(x) = 6 - 4x$, resuelto por Padilla Sánchez Hariel.

3.4. Función resuelta por Olguín Castillo Víctor Manuel

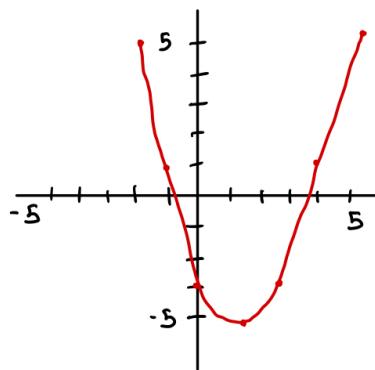
A continuación se calculan los coeficientes de Fourier asociados a la función $f(x) = x^2 - 3x - 3$

$$f(x) = x^2 - 3x - 3$$

$$\sin(n\pi) = 0$$

$$\sin(-n\pi) = -\sin(n\pi)$$

$$\cos(-n\pi) = \cos(n\pi)$$



Formulas

$$f(x) = \underbrace{a_0}_\text{offset} + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi}{P} x + b_n \sin \frac{n\pi}{P} x \right)$$

$$a_0 = \frac{1}{P} \int_{-P}^P f(x) dx \quad a_n = \frac{1}{P} \int_{-P}^P f(x) \cos \frac{n\pi}{P} x dx$$

$$b_n = \frac{1}{P} \int_{-P}^P f(x) \sin \frac{n\pi}{P} x dx$$

Resolviendo la ecuación

$$\begin{aligned}
 a_0 &= \frac{1}{\pi} \int_{-\pi}^{\pi} (x^2 - 3x - 3) dx = \frac{1}{\pi} \left[\frac{x^3}{3} - \frac{3x^2}{2} - 3x \right]_{-\pi}^{\pi} \\
 &= \frac{1}{\pi} \left[\frac{\pi^3}{3} - \frac{3\pi^2}{2} - 3\pi \right] - \frac{1}{\pi} \left[\frac{-\pi^3}{3} - \frac{3(-\pi)^2}{2} + 3(-\pi) \right] \\
 &= \frac{\pi^3}{3\pi} - \frac{3\pi^2}{2\pi} - \frac{3\pi}{\pi} + \frac{\pi^3}{3\pi} + \cancel{\frac{3\pi^2}{2\pi}} - \cancel{\frac{3\pi}{\pi}} = \frac{2\pi^2}{3} - 6
 \end{aligned}$$

Figura 3.8: Cálculo de a_0 , resuelto por Olguín Castillo.

$$\begin{aligned}
 a_n &= \frac{1}{\pi} \int_{-\pi}^{\pi} (x^2 - 3x - 3) \cos \frac{n\pi}{\pi} x \, dx \\
 &\Rightarrow \int_{-\pi}^{\pi} x^2 \cos(nx) \, dx = 2 \int_0^{\pi} x^2 \cos(nx) \, dx \\
 &\Rightarrow \int_{-\pi}^{\pi} \cos(nx) \, dx = 2 \int_0^{\pi} \cos(nx) \, dx \\
 a_n &= \frac{2}{\pi} \left[\int_0^{\pi} x^2 \cos(nx) \, dx - 3 \int_0^{\pi} \cos(nx) \, dx \right] \\
 \int_0^{\pi} x^2 \cos(nx) \, dx &= \frac{\sin(nx)}{n} \Big|_0^{\pi} = \frac{\sin(n\pi)}{n} - 0 = 0
 \end{aligned}$$

dv = $\cos(nx) \, dx$

$$\begin{aligned}
 a_n &= \frac{2}{\pi} \int x^2 \cos(nx) \, dx \\
 a_n &= \frac{x^2 \sin(nx)}{n} \Big|_0^{\pi} - \int_0^{\pi} \frac{2x \sin(nx)}{n} \, dx
 \end{aligned}$$

U = x^2 dv = $2x \, dx$
 V = $\frac{\sin(nx)}{n}$

$$\begin{aligned}
 a_n &= -\frac{2}{n} \int_0^{\pi} x \sin(nx) \, dx \\
 a_n &= \frac{x \cos(nx)}{n} \Big|_0^{\pi} + \int_0^{\pi} \frac{\cos(nx)}{n} \, dx
 \end{aligned}$$

U = x du = dx
 dv = $\sin(nx) \, dx$ V = $-\frac{\cos(nx)}{n}$

$$\begin{aligned}
 a_n &= \frac{x \cos(nx)}{n} \Big|_0^{\pi} = -\frac{\pi \cos(n\pi)}{n} \\
 a_n &= \frac{2\pi \cos(n\pi)}{n^2} = \frac{4 \cos(n\pi)}{n^2}
 \end{aligned}$$

$$\cos(n\pi) = (-1)^n$$

$$a_n = \frac{4(-1)^n}{n^2}$$

Figura 3.9: Cálculo de a_n , resuelto por Olguin Castillo.

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} (x^2 - 3x - 3) \sin(nx) dx$$

IMPAR = \emptyset
 PAR

$$b_n = -\frac{3}{\pi} \int_{-\pi}^{\pi} x \sin(nx) dx = \frac{3}{\pi} \cdot 2 \int_0^{\pi} x \sin(nx) dx$$

$$b_n = \frac{6}{n} \left[-\frac{\pi \cos(n\pi)}{n} \right] = \frac{6 \cos(n\pi)}{n}$$

$$b_n = \frac{6(-1)^n}{n}$$

Sustituyendo

$$f(x) = \frac{\pi^2}{3} - 3 + \sum_{n=1}^{\infty} \left[\frac{4(-1)^n}{n^2} \cdot \cos(nx) + \frac{6(-1)^n}{n} \cdot \sin(nx) \right]$$

Figura 3.10: Cálculo de b_n , resuelto por Olguín Castillo.

El método se basa en la expansión en series de Fourier usando simplificaciones algebraicas y las propiedades trigonométricas que permiten reducir la expresión final.

3.5. Programa en C para el cálculo de la serie de Fourier

El objetivo del programa es calcular una aproximación de la serie de Fourier para la función $f(x) = 64x$, utilizando un enfoque paralelo basado en la creación de procesos hijos, memoria compartida y semáforos en el sistema operativo Linux. Esta función es impar, por lo tanto, su serie de Fourier se puede expresar únicamente con términos en seno, en la forma:

$$f(x) \approx a_0 + \sum_{n=1}^{N} b_n \sin(nx)$$

donde $a_0 = 6$ y los coeficientes b_n están dados por:

$$b_n = \frac{8}{n} (-1)^n$$

Para este programa, se consideran valores de x desde -3.14 hasta 3.14 con un incremento de 0.15. Se calculan los primeros 10 términos de la serie de Fourier, y cada uno de estos es calculado por un proceso hijo independiente. Los resultados son almacenados en una matriz en memoria compartida, protegida por un semáforo para evitar conflictos en la escritura concurrente. Finalmente, se suman todos los términos y se genera un archivo CSV con los resultados. Para la implementación en C se hace uso de varias bibliotecas. La biblioteca `stdio.h` se emplea para las operaciones de entrada y salida estándar como la lectura y escritura en archivos. La

biblioteca `stdlib.h` se utiliza para funciones de control como `exit()` y la gestión de memoria dinámica. `unistd.h` proporciona el uso de `fork()`, fundamental para la creación de procesos hijos. Además, se incluyen `sys/ipc.h`, `sys/shm.h` y `sys/sem.h`, necesarias para trabajar con memoria compartida y semáforos bajo el esquema de IPC System V en Linux. Finalmente, `math.h` permite el uso de funciones matemáticas como `sin()` y `pow()`.

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<sys/ipc.h>
5 #include<sys/shm.h>
6 #include<sys/sem.h>
7 #include<sys/wait.h>
8 #include<math.h>
```

Figura 3.11: Fragmento de código que muestra las librerías utilizadas en el programa.

La función `funcion` tiene la responsabilidad de calcular cada término individual de la serie de Fourier. Recibe como parámetros el valor de n y el punto x , y devuelve el resultado de la expresión $\frac{8}{n}(-1)^n \sin(nx)$, que corresponde al coeficiente $b_n \sin(nx)$. Este valor es calculado por cada proceso hijo para los diferentes valores de x .

```

29
30 // definir función de Fourier
31 double funcion(int n, double x)
32 {
33     double seno;
34     seno=sin(n*x);
35     return (double )((8.0000/n)* pow(-1,n) * seno);
36 }
37
```

Figura 3.12: Fragmento de código de la función que calcula el término $b_n \sin(nx)$.

Para controlar el acceso concurrente a la memoria compartida, se definen dos funciones auxiliares: `down` y `up`. La función `down` realiza una operación de espera o bloqueo, decrementando el valor del semáforo. Esto indica que un proceso ha ingresado a la sección crítica. Por otro lado, la función `up` incrementa el semáforo, liberando así la sección crítica para que otro proceso pueda acceder. Estas funciones garantizan la exclusión mutua al momento de escribir en la memoria compartida.

La función `Crea_semaforo` encapsula el proceso de creación de un semáforo. Se basa en una clave generada por la función `ftok()` y lo crea con permisos de lectura y escritura para todos

los usuarios. Se inicializa con un valor de 1, lo cual significa que inicialmente la sección crítica está libre.

```

48
49 // Crear Semaforo
50 int Crea_semaforo(key_t llave,int valor_inicial)
51 {
52     int semid=semget(llave,1,IPC_CREAT|PERMISOS);
53     if(semid==-1)
54     {
55         return -1;
56     }
57     semctl(semid,0,SETVAL,valor_inicial);
58     return semid;
59 }
60

```

Figura 3.13: Fragmento de código que muestra la creación del semáforo.

La función `crearArchivo` se utiliza para crear archivos vacíos que posteriormente son usados por `ftok()` para generar claves únicas. Aunque no escriben información dentro del archivo, su existencia es necesaria para que `ftok()` funcione correctamente. Se crean archivos para la memoria compartida y el semáforo.

```

60
61 void crearArchivo(const char *nombreArchivo) {
62     FILE *archivo = fopen(nombreArchivo, "w");
63     if (archivo == NULL) {
64         perror("No se pudo crear el archivo");
65         exit(1);
66     }
67     fclose(archivo);
68 }
69

```

Figura 3.14: Fragmento de código que muestra la creación de archivos vacíos para la generación de claves.

Dentro de la función `main`, se calcula primero la cantidad de puntos de evaluación entre los valores de -3.14 y 3.14, considerando un incremento de 0.15. Esta cantidad determina el tamaño del arreglo de valores de x , así como el tamaño de la memoria compartida necesaria para guardar todos los resultados. Luego, se generan las claves para la memoria y el semáforo, y se crean ambos recursos. A continuación, se calcula el arreglo de valores de x y se almacena en un arreglo auxiliar.

```

// Generamos valores de paso
for (valor_actual = FIN; valor_actual <= INICIO; valor_actual += PASO) {
    if (indice < numero_elementos) {
        valores_paso[indice] = valor_actual;
        indice++;
    }
}

```

Figura 3.15: Cálculo de los valores de x que se usarán en la evaluación de los términos de la serie.

Posteriormente, se crea el primer proceso hijo que escribe directamente la constante $a_0 = 6$ en toda la primera fila de la matriz compartida. Después, se crean otros diez procesos hijos. Cada uno de estos se encarga de calcular los valores del término $b_n \sin(nx)$ para cada valor de x . Para evitar que múltiples procesos escriban al mismo tiempo, cada proceso llama a `down` antes de escribir y a `up` después de terminar de escribir.

```

117
118     // Primer hijo para n=0 valor constante de 6
119     hijos[0] = fork();
120     if (hijos[0] == 0) {
121         // Proceso hijo
122         for (int i = 0; i < numero_elementos; i++) {
123             matriz_resultados[0][i] = a0;
124         }
125         exit(0);
126     }
127
128     // Generamos los demás hijos valores de n
129     for (int n = 1; n < (COLUMNAS - 1); n++) {
130         hijos[n] = fork();
131         if (hijos[n] == 0) {
132             for (int i = 0; i < numero_elementos; i++) {
133                 double x = valores_paso[i];
134                 double bn = funcion(n, x);
135
136                 down(semaforo_general);
137                 matriz_resultados[n][i] = bn;
138                 // printf("n = %d, x = %.2f, bn = %.4f\n", n, x, bn);
139                 up(semaforo_general);
140             }
141             shmdt(matriz_resultados);
142             exit(0);
143         }
144     }

```

Figura 3.16: Código que muestra la creación de procesos hijos y el uso de semáforo para escritura sincronizada.

El proceso padre se encarga de esperar a que todos los procesos hijos terminen su ejecución. Para ello, utiliza múltiples llamadas a la función `wait()`. Una vez que todos los hijos han terminado, el padre realiza la suma de todos los términos calculados para cada valor de x . Esta suma se almacena en la última fila de la matriz compartida.

```

145     // Esperamos a que terminen los hijos
146     for (int i = 0; i < COLUMNAS - 1; i++) {
147         wait(NULL);
148     }
149
150
151     // Se calcula la suma de los resultados
152     for (int i = 0; i < numero_elementos; i++) {
153         double suma = 0.0;
154         for (int fila = 0; fila < COLUMNAS - 1; fila++) {
155             suma += matriz_resultados[fila][i];
156         }
157         matriz_resultados[11][i] = suma;
158     }
159

```

Figura 3.17: Código que suma los términos de la serie para cada valor de x y escribe los resultados en un archivo.

Finalmente, se abre un archivo CSV llamado `resultados.csv` y se imprimen en él todos los resultados, incluyendo el valor de x , cada término de la serie y la suma total.

```

160     FILE *archivo = fopen("resultados.csv", "w");
161     if (archivo == NULL) {
162         perror("Error al crear archivo CSV");
163         exit(1);
164     }
165
166
167     // Se imprime la matriz resultados
168     printf("Matriz Resultados:\n");
169     for (int j = 0; j < numero_elementos; j++) {
170         printf("%2f ", valores_paso[j]);
171         fprintf(archivo, "%2f", valores_paso[j]);
172         for (int i = 0; i < COLUMNAS; i++) {
173             fprintf(archivo, ",%4f", matriz_resultados[i][j]);
174             printf("%4f ", matriz_resultados[i][j]);
175         }
176         printf("\n");
177         fprintf(archivo, "\n");
178     }
179     fclose(archivo);
180

```

Figura 3.18: Código que muestra la escritura de los resultados en un archivo CSV.

Al finalizar la ejecución del programa, se imprime en la consola la matriz completa de resultados. En cada línea se muestra un valor de x , seguido por los términos a_0 y $b_n \sin(nx)$ calculados por cada proceso hijo para ese punto, así como la suma total de todos los términos, que representa la aproximación de $f(x)$ mediante la serie de Fourier. Esta visualización permite al usuario verificar que los cálculos se han realizado correctamente y apreciar el comportamiento de la aproximación en el dominio especificado.

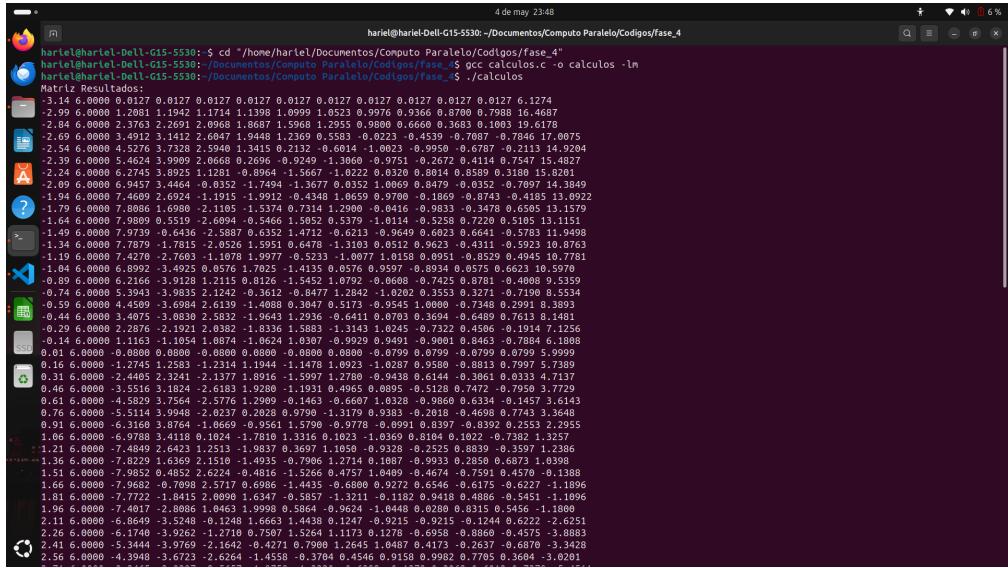


Figura 3.19: Código que muestra la impresión de la matriz de resultados en la consola.

Como parte del cierre correcto del programa, se realiza la desvinculación de la memoria compartida mediante `shmdt()`, y posteriormente se elimina del sistema con `shmctl()`. Asimismo, se elimina el semáforo con `semctl()`, liberando así todos los recursos del sistema utilizados por el programa.

```
181     // Desvinculamos la memoria compartida
182     if (shmctl(id_matriz, IPC_RMID, NULL) == -1) {
183         perror("Error al desvincular memoria compartida");
184         exit(1);
185     }
186     shmctl(id_matriz, IPC_RMID, NULL);
187     semctl(semaforo_general, 0, IPC_RMID);
```

Figura 3.20: Código que muestra el cierre del programa y la liberación de recursos.

3.6. Análisis de la aproximación mediante serie de Fourier

Se presenta un análisis sobre la aproximación mediante series de Fourier para la función lineal:

$$f(x) = 6 - 4x$$

El análisis se apoya en los cálculos de la serie de Fourier realizados por el estudiante Padilla Sanchez Haniel 3.6 y 3.7. La función $f(x)$ es lineal y se define en el intervalo $[-3, 14, 3, 14]$. La serie de Fourier para esta función se expresa como:

$$\text{Término}_n = \frac{8}{n} \cdot (-1)^n \cdot \sin(n \cdot x)$$

Una vez calculados los coeficientes a_0 y b_n , se procede a la evaluación de la serie en un rango de valores de x . De primera mano se realizaron cálculos en el intervalos sobre la ecuación

$$f(x) = 6 - 4x$$

esto nos dara una recta lineal, posterior a ello durante la fase 2 se realizaron los calculos de la serie de fourier en una hoja de excel, donde se implementaron las fórmulas necesarias para calcular los términos de la serie y finalmente se realizaron los calculos con el programa en C usando memoria compartida y semáforos para la sincronización de procesos como se muestra en 3.19.

Se juntaron los resultados obtenidos de la hoja de excel y del programa en C, y se graficaron en la figura 3.21. En esta gráfica se observa que la ecuación nos muestra una función lineal en la recta naranja, mientras que las series de fourier calculadas en el intervalo tanto como en el excel como se muestra pintada en azul, y el programa en C pintada en verde, nos muestran una aproximación a la función lineal, sin embargo, se puede observar que la serie de fourier tiene mucha similitud a la función sin embargo en la parte de los extremos se puede observar que la serie de fourier no logra aproximarse a la función lineal, esto es debido a que la serie de fourier es una aproximación de la función en un intervalo cerrado, y al ser una función lineal no tiene un periodo definido, por lo que la serie de fourier no logra aproximarse a la función en su totalidad.

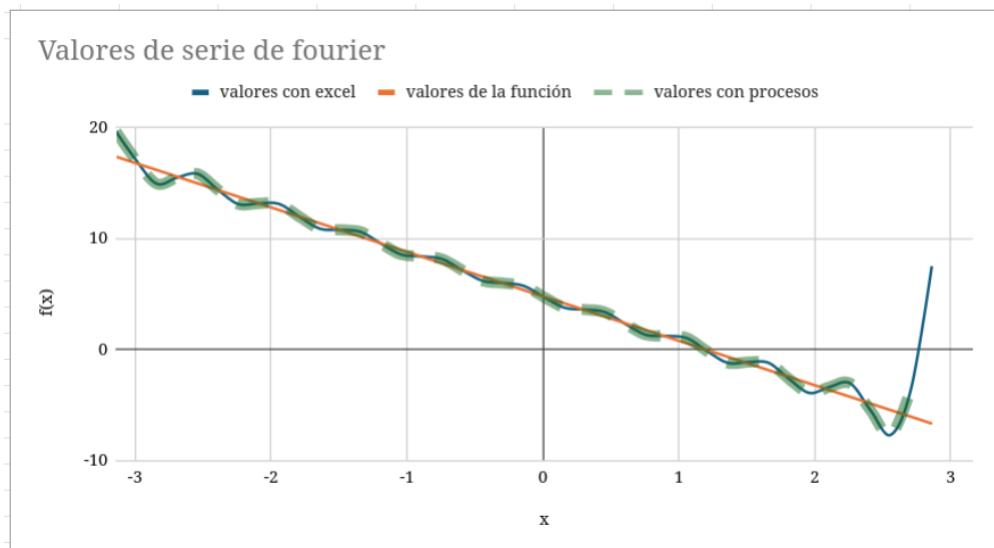


Figura 3.21: Grafica de resultados sobre los calculos en excel y el código C.

3.7. Programa en C para el cálculo de la serie de Fourier usando hilos

En esta fase se desarrolló un programa en lenguaje C que implementa el cálculo de la serie de Fourier para la función $f(x) = 6 - 4x$ utilizando programación paralela con hilos mediante la biblioteca pthread. A diferencia de las fases anteriores en las que se usaron procesos hijos y mecanismos de sincronización como semáforos, los hilos permiten una forma más sencilla de compartir memoria y comunicación entre tareas paralelas, ya que todos los hilos comparten el mismo espacio de direcciones dentro del proceso principal. Esto reduce la complejidad y el uso de estructuras adicionales para el intercambio de información.

El programa inicia con la inclusión de bibliotecas esenciales para su funcionamiento. Se utiliza `stdio.h` para las operaciones de entrada y salida como impresión por pantalla y escritura en

archivos. `stdlib.h` se usa para funciones relacionadas con la administración de memoria dinámica. `math.h` se emplea para funciones matemáticas como `sin()` y `pow()` que son necesarias para calcular los términos de la serie. Finalmente, `pthread.h` es crucial para manejar la creación, ejecución y sincronización de los hilos. La inclusión de estas bibliotecas asegura que el programa pueda realizar correctamente todas las operaciones necesarias para el cálculo paralelo de la serie de Fourier.

Las constantes definidas al inicio del programa delimitan el comportamiento del mismo. Se declara `INICIO` con un valor de 3.14 y `FIN` con -3.14, indicando que el intervalo de evaluación de la función se realiza simétricamente alrededor del cero. El paso, definido por la constante `PASO`, se establece en 0.15, determinando la resolución del muestreo de x . La constante `COLUMNAS` se define como 12, pues considera una fila para el valor a_0 , diez filas para los términos $b_n \sin(nx)$ desde $n = 1$ hasta $n = 10$, y una fila adicional para almacenar la suma total de todos los términos, que representa el valor aproximado de $f(x)$ en cada punto.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <pthread.h>
5
6 #define INICIO 3.14
7 #define FIN -3.14
8 #define PASO 0.15
9 #define COLUMNAS 12
10

```

Figura 3.22: Fragmento de código donde se incluyen las bibliotecas y se definen las constantes utilizadas.

La función principal del programa tiene como responsabilidad inicial determinar cuántos puntos se evaluarán en el intervalo definido. Esto se logra calculando el número de pasos de x a través de la fórmula:

$$\text{filas} = \frac{|\text{INICIO} - \text{FIN}|}{\text{PASO}} + 1$$

El resultado determina la cantidad de columnas de la matriz bidimensional donde se almacenarán los resultados, ya que cada columna representará un valor diferente de x .

Se define una estructura llamada `hilo_args_t` que encapsula la información que cada hilo necesita para realizar su cálculo de forma independiente. Esta estructura contiene un puntero a la matriz compartida de resultados, el índice del término que le corresponde al hilo (por ejemplo,

si se trata de a_0 , $b_1 \sin(x)$, $b_2 \sin(2x)$, etc.), el número total de filas y el valor del paso. Gracias a los hilos compartir memoria con el hilo principal, no se requiere el uso de memoria compartida explícita, lo que simplifica el diseño del programa.

```

9 |     typedef struct {
9 |     |         int n; // fila que calculara el hilo
10| } hilo_args_t;

```

Figura 3.23: Definición de la estructura utilizada para pasar parámetros a cada hilo.

Cada hilo ejecuta la función `calcular_fila`, que es responsable de llenar una fila específica de la matriz. Si el índice recibido es 0, el hilo escribe el valor constante $a_0 = 6$ en toda la fila. En caso contrario, se calcula el valor de b_n según la fórmula $b_n = \frac{8}{n}(-1)^n$ y se evalúa $b_n \sin(nx)$ para cada valor de x en el intervalo definido. La variable x se actualiza a partir del índice de la fila y el valor de paso, es decir, $x = \text{INICIO} - i \cdot \text{PASO}$. Este diseño permite que cada hilo trabaje de forma independiente sobre una porción distinta del resultado final.

```

23 | void *calcular_fila(void *arg) {
24 |     hilo_args_t *args = (hilo_args_t *)arg;
25 |     int n = args->n;
26 |
27 |     if (n == 0) {
28 |         for (int i = 0; i < numero_elementos; i++) {
29 |             matriz_resultados[0][i] = 6.0;
30 |         }
31 |     } else {
32 |         for (int i = 0; i < numero_elementos; i++) {
33 |             double x = valores_paso[i];
34 |             matriz_resultados[n][i] = funcion(n, x);
35 |         }
36 |     }
37 |
38 |     free(args); // liberar memoria del struct
39 |     pthread_exit(NULL);
40 |

```

Figura 3.24: Función que calcula cada fila de la matriz con base en el índice del hilo.

Una vez definida la función que ejecuta cada hilo, en la función principal se crean 12 hilos correspondientes a cada fila de la matriz. A cada hilo se le asigna una instancia de la estructura de parámetros, se inicializa con los valores necesarios y se invoca usando `pthread_create`. Posteriormente, se usa `pthread_join` para esperar la finalización de cada hilo antes de continuar, asegurando que todos los resultados estén correctamente calculados antes de proceder al siguiente paso.

```

49     pthread_t hilos[COLUMNAS - 1];
50
51     for (int n = 0; n < COLUMNAS - 1; n++) {
52         hilo_args_t *args = malloc(sizeof(hilo_args_t));
53         args->n = n;
54         if (pthread_create(&hilos[n], NULL, calcular_fila, args) != 0) {
55             perror("Error al crear hilo");
56             exit(EXIT_FAILURE);
57         }
58     }
59
60     for (int i = 0; i < COLUMNAS - 1; i++) {
61         pthread_join(hilos[i], NULL);
62     }
63

```

Figura 3.25: Fragmento del código que muestra la creación y sincronización de los hilos.

Después de que todos los hilos han terminado su ejecución, se calcula la suma total por columnas, almacenando el resultado en la última fila de la matriz. Esta fila representa la aproximación de la función $f(x)$ mediante la suma de los 11 términos de la serie de Fourier evaluados para cada valor de x . La matriz se recorre iterativamente sumando el valor correspondiente de cada fila (excepto la última) para cada columna, y el resultado se guarda en la posición correspondiente de la fila final.

```

63
64     // Calcular suma final en la ultima fila
65     for (int i = 0; i < numero_elementos; i++) {
66         double suma = 0.0;
67         for (int j = 0; j < COLUMNAS - 1; j++) {
68             suma += matriz_resultados[j][i];
69         }
70         matriz_resultados[COLUMNAS - 1][i] = suma;
71     }
72

```

Figura 3.26: Código encargado de calcular la suma total de la serie de Fourier en cada punto.

Una vez completados todos los cálculos, los resultados se escriben en un archivo CSV llamado `resultado_hilos.csv`. Cada fila del archivo corresponde a un término de la serie (incluyendo la suma total), y cada columna representa un valor de x . Este archivo puede ser utilizado posteriormente para generar gráficas comparativas o para su análisis en herramientas como Excel o Python.

```

72
73     // Imprimir resultado
74     FILE *archivo = fopen("resultados.csv", "w");
75     if (!archivo) {
76         perror("Error al abrir resultados.csv");
77         exit(EXIT_FAILURE);
78     }
79
80     printf("Matriz Resultados:\n");
81     for (int j = 0; j < numero_elementos; j++) {
82         printf("%.2f ", valores_paso[j]);
83         fprintf(archivo, "%.2f", valores_paso[j]);
84         for (int i = 0; i < COLUMNAS; i++) {
85             printf("%.4f ", matriz_resultados[i][j]);
86             fprintf(archivo, "%.\n4f", matriz_resultados[i][j]);
87         }
88         printf("\n");
89         fprintf(archivo, "\n");
90     }
91
92     fclose(archivo);

```

Figura 3.27: Fragmento de código encargado de guardar los resultados en un archivo CSV.

Además del archivo CSV, el programa también imprime los resultados en la terminal, mostrando para cada valor de x el valor aproximado de $f(x)$ obtenido mediante la suma de la serie de Fourier. Esta salida permite verificar rápidamente si los resultados parecen razonables sin necesidad de abrir el archivo de salida.

```

harel@harel-Dell-G15-5530: /Documentos/Computo Paralelo/Fases - reporte/Fase 1 - Cálculo Paralelo/Code
gcc fase4.c -o fase4
./fase4
          x      f(x)
 0.000000 6.000000
 0.125000 5.875000
 0.250000 5.750000
 0.375000 5.625000
 0.500000 5.500000
 0.625000 5.375000
 0.750000 5.250000
 0.875000 5.125000
 1.000000 5.000000
 1.125000 4.875000
 1.250000 4.750000
 1.375000 4.625000
 1.500000 4.500000
 1.625000 4.375000
 1.750000 4.250000
 1.875000 4.125000
 2.000000 4.000000
 2.125000 3.875000
 2.250000 3.750000
 2.375000 3.625000
 2.500000 3.500000
 2.625000 3.375000
 2.750000 3.250000
 2.875000 3.125000
 3.000000 3.000000
 3.125000 2.875000
 3.250000 2.750000
 3.375000 2.625000
 3.500000 2.500000
 3.625000 2.375000
 3.750000 2.250000
 3.875000 2.125000
 4.000000 2.000000
 4.125000 1.875000
 4.250000 1.750000
 4.375000 1.625000
 4.500000 1.500000
 4.625000 1.375000
 4.750000 1.250000
 4.875000 1.125000
 5.000000 1.000000
 5.125000 0.875000
 5.250000 0.750000
 5.375000 0.625000
 5.500000 0.500000
 5.625000 0.375000
 5.750000 0.250000
 5.875000 0.125000
 6.000000 0.000000

```

Figura 3.28: Impresión de los resultados aproximados de $f(x)$ en la terminal.

Finalmente, se observa una gráfica comparativa entre las tres fases implementadas: la primera con una función lineal, la segunda con los valores de excel, la tercera con procesos y la cuarta con hilos. En esta gráfica se representa el valor original de la función $f(x) = 6 - 4x$ y las aproximaciones obtenidas en cada fase. Se observa que, tanto los resultados generados en la fase 2 calculados por medio de excel, los valores de la fase 3 con procesos y los de la fase 4 con hilos, todos son iguales y así vez se aproximan a la función original.

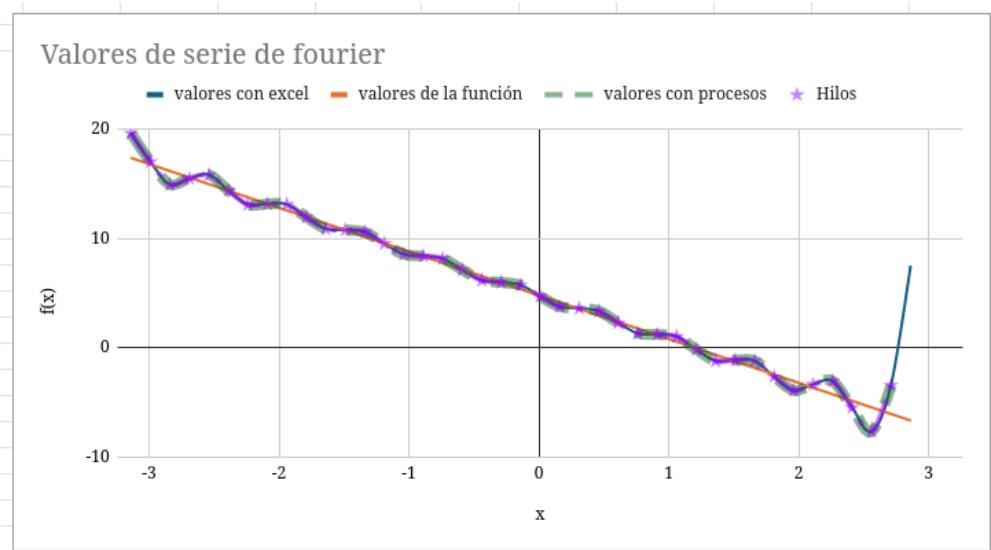


Figura 3.29: Comparativa gráfica entre el valor real de la función y las aproximaciones de las fases 2, 3 y 4.

4

Conclusiones

4.1. Castro Paez Ilse Yazbeth

Resolver la serie de Fourier fue un proceso laborioso pero muy interesante. Al tratarse de una función polinómica, al inicio parecía ser muy sencilla pero conforme avancé en los cálculos me percate de la importancia de aplicar correctamente cada uno de los pasos del desarrollo de Fourier.

Desde el planteamiento de la integral hasta la obtención de los coeficientes, cada una de las operaciones requería de precisión y un buen entendimiento sólido de cómo usar correctamente el método. En particular, resolver la integral de una función cuadrática presentó un reto interesante, ya que implicó manipular expresiones algebraicas de segundo grado y así verificar que los resultados fueran consistentes. Otro punto importante fue saber interpretar correctamente el resultado de la serie y cómo la combinación de términos trigonométricos podía aproximar la función original de forma periódica. A parte de los cálculos de la función, esta actividad me ayudó y me resultó bastante útil para poder comprender de mejor manera el contexto y la utilidad de las series de Fourier.

En el área de la inteligencia artificial juega un papel de gran importancia en el procesamiento de señales y de imágenes, especialmente en técnicas de filtrado y compresión de datos. Dentro del cómputo paralelo, su optimizada implementación permite que el proceso del análisis de grandes volúmenes de información sea más veloz, lo que facilita las aplicaciones en las telecomunicaciones y el procesamiento del audio.

Este ejercicio me dejó una mayor apreciación y un mayor aprendizaje en la profundidad matemática detrás del análisis de Fourier. Aunque al inició el hecho de hacer una transformación de una función en una serie infinita de términos trigonométricos puede resultar ser algo abstracta, al ir desarrollando los cálculos comprendí mucho mejor cómo es que funciona esta descomposición y su importancia en diversos campos.

Me permitió reforzar mis habilidades de integración y manipulación algebraica, lo cual es esencial para el correcto manejo de ecuaciones en aplicaciones más avanzadas. La manipulación igualmente desempeñó un papel de gran importancia en el proceso, desde la simplificación de expresiones hasta la ordenación de términos en las ecuaciones resultantes. Otra de las habilidades reforzadas fue la interpretación de los resultados obtenidos, ya que no basta solo con obtener una solución matemática, si no que es de gran importancia saber comprender su significado y sus aplicaciones.

Este ejercicio me permitió conectar y entender mejor los conceptos abstractos con aplicaciones prácticas al mundo real. La resolución de la serie de Fourier para esta función representó un reto matemático interesante y también reforzó mis habilidades fundamentales en la integración, álgebra y el modelado matemático.

Como parte de la segunda fase, se realizó la graficación de la función proporcional usando series de Fourier y una matriz en Excel. Para ello se generaron los datos correspondientes a los coeficientes de la serie. Usando fórmulas y herramientas gráficas de Excel, se pudo visualizar la aproximación de la serie a la función original.

Esta actividad reforzó mi comprensión teórica de las series de Fourier y me permitió observar cómo, al ir incrementando el número de términos de la serie, la aproximación se vuelve cada vez más precisa.

Uno de los aspectos más destacados de esta experiencia fue el aprendizaje y la aplicación de fórmulas y funciones más avanzadas en una hoja de cálculo. El uso de fórmulas para automatizar cálculos repetitivos ahorro tiempo y redujo la posibilidad de errores, lo cual es crucial en trabajos que utilizan un análisis numérico.

Un aspecto también importante fue la creación de tablas dinámicas y gráficos. Estas herramientas me permitieron visualizar los datos de una forma más clara y comprensible, lo cuál facilita la interpretación de los resultados para poder identificar patrones. Por ejemplo, al obtener los datos ya graficados, pude observar de manera inmediata como los datos se comportaban entre ellos.

A lo largo del proceso, me enfrenté a diversos desafíos los cuales pusieron a prueba mis habilidades. Uno de esos desafíos fue el manejo de grandes conjuntos de datos, para solucionar esto, aprendí a optimizar y utilizar funciones más eficientes que permiten procesar datos de manera más rápida.

La experiencia adquirida en esta fase no solo tiene aplicaciones académicas, sino también prácticas en el mundo profesional. Las hojas de cálculo son una herramienta esencial en campos como la ingeniería, la economía, la administración y las ciencias, donde el manejo y análisis de datos son fundamentales. Aprender a utilizarlas de manera eficiente me ha dado una ventaja competitiva y me ha preparado para enfrentar problemas más complejos en el futuro.

En conclusión, esta fase de trabajo con hojas de cálculo de Excel ha sido una experiencia muy importante que me ha permitido desarrollar habilidades técnicas, mejorar mi capacidad de organización y precisión, y enfrentar desafíos que han fortalecido mi pensamiento crítico y analítico. Aunque al principio el manejo de estas herramientas puede parecer abrumador, con práctica y dedicación es posible dominarlas y aprovechar todo su potencial.

4.2. Catonga Tecla Daniel Isaí

En esta primera fase, al resolver la ecuación para obtener los coeficientes de Fourier, me pareció interesante descubrir que existía una función específica para calcular estos coeficientes y, a partir de ellos, obtener la serie de Fourier correspondiente. Fue una experiencia buena para mí porque, en este caso, la ecuación a resolver no presentaba una gran dificultad, ya que se trataba de una función lineal. Esto facilitó considerablemente el proceso, permitiéndole comprender mejor la metodología sin enfrentar complicaciones excesivas.

Sin embargo, uno de los desafíos con los que me encontré fue el hecho de que, al no haber trabajado recientemente con cálculo diferencial e integral, en un principio olvidé algunos pasos clave, como la correcta aplicación de las reglas de integración y derivación. Esto hizo que tuviera que repasar varias fórmulas y recordar conceptos que no había utilizado en algún tiempo, lo que representó un reto adicional. A pesar de ello, logré superar estas dificultades y resolver correctamente la ecuación dada, que en este caso correspondía a la función $f(x) = 6 - 2x$.

Una de las cosas que aprendí de la transformada de Fourier es que permite expresar funciones mediante series trigonométricas. En particular, me llamó la atención su aplicación en la inteligencia artificial y el procesamiento de señales que es una materia que actualmente estoy cursando, por lo que me resultó útil conocer un poco más allá. En la inteligencia artificial, las series de Fourier son utilizadas para el análisis de datos, reducción de dimensionalidad y el reconocimiento de patrones en grandes volúmenes de información y esto es importante para mí ya que es un área en la que me quiero especializar. En el procesamiento de señales, juegan un papel crucial en la filtración y mejora de calidad de audio e imagen, así como en la transmisión eficiente de datos.

Además, este ejercicio me permitió reforzar mis habilidades matemáticas en integración y manipulación algebraica, lo que considero esencial para el correcto manejo de ecuaciones en aplicaciones más avanzadas. Me ayudó también a comprender la importancia de la precisión en los cálculos y la correcta interpretación de los resultados obtenidos.

En los cálculos realizados, es interesante observar cómo el número obtenido en cada iteración depende del valor de n , que representa el número de la iteración. A medida que aumentamos el valor de n , la aproximación de la función original se va refinando. Cada iteración nos proporciona un valor que se utiliza para calcular los armónicos correspondientes, los cuales son esenciales para reconstruir la función.

Este proceso iterativo nos permite generar una serie de armónicos que, al ser sumados, proporcionan una aproximación cada vez más precisa de la función que estamos tratando de resolver. Cuantos más términos se incluyan en la suma, mayor es la exactitud de la representación de la función original. De este modo, la serie de Fourier ofrece una herramienta poderosa para

aproximar funciones complejas con una precisión controlable.

Para el manejo de Excel no tuve mayores dificultades para construir la tabla de valores y calcular cada una de las iteraciones. Gracias a mi experiencia previa trabajando con tablas, el proceso fue relativamente sencillo y no presentó complicaciones. Además, Excel facilitó la organización de los datos y la obtención de los resultados de manera rápida y clara.

Por otro lado, la visualización gráfica de los resultados fue una herramienta clave para entender mejor el comportamiento de la serie de Fourier. Ver cómo la aproximación mejora a medida que se agregan más iteraciones me permitió apreciar de manera más tangible cómo los términos de la serie se ajustan progresivamente a la función original. La representación gráfica no solo ayudó a confirmar la precisión de la aproximación, sino que también facilitó la interpretación de los resultados, haciendo que el análisis fuera mucho más intuitivo.

Después de realizar los cálculos, pude comprender mejor por qué es útil utilizar cómputo paralelo para resolver este tipo de problemas. Cuando se trata de funciones más complejas, el uso de cómputo paralelo se vuelve esencial para realizar las iteraciones necesarias de manera eficiente. Esto permite aproximar la función original con la mayor precisión posible, reduciendo el tiempo de cálculo y optimizando los recursos al trabajar con múltiples procesos simultáneamente.

Además, el cómputo paralelo no solo mejora la eficiencia, sino que también facilita el manejo de grandes cantidades de datos o funciones de alta complejidad. A medida que se aumentan los términos en la serie de Fourier o se analizan funciones de mayor dimensión, la capacidad de distribuir los cálculos entre varios núcleos de procesamiento se convierte en una ventaja significativa. Esto no solo acelera el proceso de aproximación, sino que también permite abordar problemas más grandes y complejos que de otro modo serían difíciles de manejar con un enfoque secuencial.

El desarrollo de la serie de Fourier para esta función representó un ejercicio valioso para mejorar mis conocimientos en análisis matemático y su aplicación en la ciencia y tecnología. Me brindó una nueva perspectiva sobre la importancia de las series de Fourier en el análisis de funciones y cómo estas herramientas pueden utilizarse en la resolución de problemas reales. En futuras ocasiones, me gustaría explorar cómo la transformada de Fourier puede aplicarse a proyectos prácticos dentro de la inteligencia artificial y el procesamiento de datos, ya que considero que su impacto en estas áreas es de gran relevancia.

En cuanto a la experiencia con la programación en lenguaje C para implementar los procesos paralelos, representó un desafío estimulante que me permitió aplicar mis conocimientos de programación en un contexto matemático avanzado. Al principio, la implementación de la parallelización mediante OpenMP requirió un período de adaptación, ya que debí familiarizarme con las directivas específicas para dividir eficientemente las tareas de cálculo entre múltiples hilos de ejecución. Particularmente, la correcta sincronización de los procesos para evitar con-

diciones de carrera y garantizar resultados precisos fue uno de los aspectos más desafiantes pero enriquecedores del ejercicio.

La traducción de las ecuaciones matemáticas de la serie de Fourier a código C me brindó una perspectiva valiosa sobre cómo las abstracciones matemáticas pueden materializarse en implementaciones computacionales eficientes. Observé cómo el rendimiento mejoraba significativamente al distribuir el cálculo de los diferentes coeficientes y términos de la serie entre varios núcleos del procesador, lo que reafirmó la importancia del paralelismo en aplicaciones de cálculo intensivo. Además, este ejercicio me permitió profundizar en conceptos de programación de bajo nivel, como la gestión de memoria y optimización de bucles, que son cruciales para el procesamiento eficiente de grandes volúmenes de datos.

Este componente práctico de programación complementó perfectamente el análisis teórico, permitiéndome visualizar de manera tangible cómo los conceptos matemáticos abstractos pueden transformarse en soluciones computacionales concretas. La experiencia adquirida con la programación en C para el cálculo de series de Fourier ha fortalecido mi interés en la intersección entre las matemáticas avanzadas y la computación de alto rendimiento, áreas que considero fundamentales para mi desarrollo profesional en el campo de la inteligencia artificial y el procesamiento de señales.

4.3. Padilla Sanchez Haniel

En esta primera fase, el profesor a cargo de la materia nos asignó una función diferente a cada integrante del equipo. En mi caso, trabajé con la función lineal $f(x)=6-4x$. Al ser una función lineal, los cálculos se redujeron considerablemente en comparación con funciones de mayor grado, lo que facilitó la obtención de los coeficientes de Fourier. Sin embargo, aún encontré desafíos, ya que algunos conceptos de integración y álgebra que no había utilizado recientemente requirieron revisarlos para aplicarlos correctamente. Además, algunas expresiones trigonométricas eran nuevas para mí, y al no tomarlas en cuenta hubiera hecho que la resolución tomara más tiempo del necesario.

En cuanto al procedimiento de la serie de Fourier, aunque los conceptos ya los había estudiado en la materia de Análisis de Sistemas Digitales, volver a retomarlos fue algo que me gusto ya que hay veces que conceptos importantes no se les da continuidad para ver sus aplicaciones. Un aspecto interesante fue analizar la paridad de la función para determinar si podía simplificar los cálculos estableciendo ciertos coeficientes como cero. Sin embargo, en este caso, aunque algunos coeficientes resultaron ser cero, fue por la naturaleza de la función y no por su paridad (de la función $f(x) = 6 - 4x$ en general).

Más allá de los cálculos, este ejercicio me permitió reflexionar sobre la importancia del análisis de Fourier en diversas disciplinas. En la ciencia, es fundamental para modelar fenómenos físicos, como la propagación de ondas y la conducción del calor. En la ingeniería, tiene aplicaciones en el procesamiento de señales, la acústica y el análisis estructural. En el ámbito tecnológico, su uso es indispensable en la compresión de datos y el procesamiento de imágenes, lo cual me resulta especialmente interesante, ya que es un área con la que interactuamos frecuentemente. Particularmente en la inteligencia artificial, las series de Fourier juegan un papel crucial en el reconocimiento de patrones y la optimización de algoritmos de aprendizaje.

Una de las aplicaciones que más me llamó la atención al investigar sobre la serie de Fourier fue su uso en inteligencia artificial, particularmente en el ámbito de la salud ya que estoy haciendo un proyecto sobre biotecnología. Su aplicación se ve justamente en las señales biomédicas en concreto me interesaron las señales electromiográficas (EMG). Las cuales sirven para detectar la actividad eléctrica de nuestros músculos, y contiene una gran cantidad de información de una persona analizando en el dominio de la frecuencia mediante transformadas de Fourier, esto nos lleva a poder analizar y detectar patrones o mejorando modelos para la adaptación o personalización de prótesis u ortesis.

Al llevar a cabo los cálculos en la hoja de cálculo (en esta fase 2), pude apreciar la importancia de la organización y sistematización de los datos, no solo para esta práctica, sino para la carrera de Inteligencia Artificial en general. En esta disciplina, el manejo adecuado de los datos es fundamental. Si bien Excel no es la herramienta más óptima para trabajos de gran escala, resulta útil para cálculos simples y de menor complejidad, como los de esta práctica. Al principio, no recordaba bien cómo fijar filas y columnas para replicar automáticamente los cálculos, lo que me llevó a invertir tiempo en reorganizar la tabla. Sin embargo, una vez solucionado, pude visualizar mejor cómo variaban los coeficientes según los términos de la serie y comprender de manera más intuitiva la aproximación de Fourier a la función original. Además, el uso de la hoja de cálculo facilitó la detección de errores, ya que los valores atípicos o inconsistentes resaltaban de inmediato al graficar los resultados.

Desde un punto de vista más amplio, esta experiencia me ayudó a comprender mejor la relevancia del análisis de Fourier en múltiples disciplinas. En ingeniería, es fundamental para el análisis de circuitos eléctricos y la transmisión de señales, ya que permite descomponer señales complejas en componentes más simples. En física, es una herramienta esencial para describir fenómenos ondulatorios, como el comportamiento de las ondas sonoras o la propagación de señales electromagnéticas. En mi caso particular, el aprendizaje de Fourier se vincula directamente con mis intereses en inteligencia artificial y biotecnología, ya que el análisis de señales es una parte crucial en el procesamiento de datos biomédicos, como las señales electromiográficas (EMG).

A nivel personal, esta fase del proyecto me permitió desarrollar mayor confianza en mis habilidades matemáticas y en mi capacidad para resolver problemas de manera estructurada. Aunque al principio algunos conceptos parecían complejos, aplicarlos en un caso concreto me ayudó a notar cómo se entrelazan distintas áreas del conocimiento, desde el cálculo hasta la programación. Además, el uso de herramientas computacionales para realizar los cálculos me permitió comprobar cómo la combinación de teoría y tecnología facilita el análisis de problemas complejos.

En conclusión, la experiencia de resolver la serie de Fourier para una función algebraica me permitió reforzar mis conocimientos matemáticos, mejorar mi manejo de herramientas computacionales y comprender la aplicabilidad de estos conceptos en problemas reales. Más allá del ejercicio académico, este análisis me dejó una reflexión importante: el poder de las matemáticas para modelar el mundo que nos rodea y su papel fundamental en el desarrollo de tecnologías avanzadas.

El desarrollo de este proyecto (fase 3) permitió aplicar de manera práctica los conceptos fundamentales del cómputo paralelo utilizando procesos, memoria compartida y semáforos en un entorno basado en Linux. A través de la implementación de un programa en C que calcula los coeficientes de la serie de Fourier para la función $f(x)=6-4xf(x)=6-4x$, se logró dividir y distribuir el trabajo entre procesos hijos, lo cual favoreció la comprensión de la sincronización y la comunicación interprocesos mediante las herramientas que ofrece el sistema operativo, como `shmget`, `shmat`, `semget`, `semop`, entre otras.

Uno de los aprendizajes más relevantes fue entender la necesidad de controlar el acceso a la memoria compartida para evitar condiciones de carrera. En este proyecto se emplearon semáforos para asegurar que cada proceso hijo accediera de forma ordenada y segura a la región de memoria compartida, evitando así errores en el cálculo o la escritura concurrente de datos. La implementación y control de estos mecanismos representó un reto técnico importante, pero también una valiosa oportunidad para fortalecer habilidades en programación de sistemas y cómputo concurrente.

Además, este ejercicio permitió observar cómo una tarea computacional que puede parecer simple, como el cálculo de una serie matemática, se puede optimizar significativamente mediante la paralelización. La creación de procesos hijos que se encargan cada uno de una parte del trabajo refleja el potencial del procesamiento paralelo para mejorar el rendimiento, especialmente en aplicaciones de mayor escala o con cargas de trabajo intensivas.

Por otro lado, la experiencia también dejó en evidencia la importancia de una buena planificación en el diseño del algoritmo paralelo. Fue necesario definir con precisión la cantidad de términos a calcular, distribuirlos correctamente entre los procesos y sincronizar la escritura en la memoria compartida para obtener resultados coherentes y precisos. Este enfoque promueve una forma de pensar más estructurada y modular, aspectos esenciales en el desarrollo de software robusto y escalable.

En resumen, esta fase del proyecto no solo contribuyó al entendimiento de la teoría detrás del cómputo paralelo, sino que permitió vivenciarla a través de un caso práctico y funcional, afianzando los conocimientos y fortaleciendo competencias clave en programación concurrente, manejo de memoria compartida y control de sincronización. Esta experiencia representa un avance significativo en la formación académica y profesional en el área de sistemas operativos y programación paralela.

En esta última fase (fase 4) del proyecto trabajé en una versión del programa para calcular la serie de Fourier, pero usando hilos en lugar de procesos. A diferencia de las fases anteriores, esta implementación fue más sencilla en cuanto a manejo de memoria, ya que los hilos comparten el mismo espacio, lo cual facilita mucho el intercambio de datos sin necesidad de usar memoria compartida ni semáforos. Esto me permitió enfocar más la atención en la lógica del cálculo que en la sincronización entre procesos, haciendo que el programa fuera más limpio y eficiente.

Una de las cosas que más noté fue cómo se puede hacer lo mismo que en fases anteriores, pero de una forma más optimizada. Los resultados que obtuve fueron los mismos que en la Fase 3, lo cual demuestra que el uso de hilos no afecta el resultado final, pero sí mejora el rendimiento y reduce la complejidad del código. Aprendí también que, aunque usar hilos puede parecer más simple, sigue siendo importante tener bien organizada la información que se le pasa a cada uno, y cuidar bien cómo se escribe en la matriz final.

Esta fase me ayudó a entender mejor cómo funciona el paralelismo en C, sobre todo con hilos. Me quedó claro que usar procesos o hilos depende mucho del tipo de aplicación que estés desarrollando y de qué tanto control necesites sobre la ejecución y la memoria. En resumen, fue una forma de cerrar el proyecto viendo otra alternativa para resolver el mismo problema, y comparando su rendimiento y complejidad con lo que hicimos antes.

Gracias a este trabajo me sentí más confiado programando en C, y también comprendí mejor cómo aplicar cómputo paralelo para resolver problemas matemáticos como el cálculo de series de Fourier. Esta última fase reforzó todo lo que aprendí en las anteriores y me dejó muy satisfecho con el resultado final del proyecto.

Anexos

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<sys/ipc.h>
5 #include<sys/shm.h>
6 #include<sys/sem.h>
7 #include<sys/wait.h>
8 #include<math.h>
9
10 #define INICIO 3.14
11 #define FIN -3.14
12 #define PASO 0.15
13 #define PERMISOS 0666
14 #define ITERACION 10
15 #define COLUMNAS 12
16
17 // Permisos   Acceso del dueño    Grupo      Otros
18 // 0600       rw-           ---        ---
19 // 0666       rw-           rw-        rw-
20 // 0644       rw-           r--        r--
21 // 0777       rwx           rwx        rwx inseguro
22
23 // Estructura de archivo en memoria
24 // n          n1          n2          ...         suma de fila
25 // 6          0.02         ...         --- 
26 // 6          1.20         ...         --- 
27 // 6          2.37         ...         --- 
28 // ...         ...         ...         --- 
29
30 // definir funcion de Fourier
31 double funcion(int n, double x)
32 {
33     double seno;
34     seno=sin(n*x);
35     return (double )((8.0000/n)* pow(-1,n) * seno);
36 }
37
38 // Funciones para el semaforo
39 void down(int semid) {
40     struct sembuf op = {0, -1, 0};
```

```
41         semop(semid, &op, 1);
42     }
43
44     void up(int semid) {
45         struct sembuf op = {0, 1, 0};
46         semop(semid, &op, 1);
47     }
48
49 // Crear Semaforo
50 int Crea_semaforo(key_t llave,int valor_inicial)
51 {
52     int semid=semget(llave,1,IPC_CREAT|PERMISOS);
53     if(semid== -1)
54     {
55         return -1;
56     }
57     semctl(semid,0,SETVAL,valor_inicial);
58     return semid;
59 }
60
61 void crearArchivo(const char *nombreArchivo) {
62     FILE *archivo = fopen(nombreArchivo, "w");
63     if (archivo == NULL) {
64         perror("No se pudo crear el archivo");
65         exit(1);
66     }
67     fclose(archivo);
68 }
69
70
71 int main(int argc, char *argv[])
72 {
73     pid_t hijos[COLUMNAS];
74     int numero_elementos = (int)((INICIO - FIN) / PASO) + 1;
75     int a0 = 6;
76     double valores_paso[numero_elementos];
77     double valor_actual;
78     int indice = 0;
79     int tam_memoria = COLUMNAS * numero_elementos * sizeof(double );
80
81
82     crearArchivo("fourier");
83     crearArchivo("semaforo_general");
84     key_t clave_resultados = ftok("fourier", 66);
85     key_t clave_semaforo_general = ftok("semaforo_general", 66);
86
87 // Creamos el semaforo
88     int semaforo_general = Crea_semaforo(clave_semaforo_general, 1);
89
90     if (semaforo_general == -1) {
91         perror("Error al crear el semáforo");
```

```
92         exit(1);
93     }
94
95     // Creamos la memoria compartida
96     int id_matriz = shmget(clave_resultados, tam_memoria, IPC_CREAT|PERMISOS);
97     if (id_matriz == -1) {
98         perror("Error al crear memoria compartida");
99         exit(1);
100    }
101
102    // Puntero a la memoria compartida
103    double (*matriz_resultados)[numero_elementos];
104    matriz_resultados = (double (*)[numero_elementos]) shmat(id_matriz, NULL, 0);
105    if (matriz_resultados == (void *)-1) {
106        perror("Error al enlazar memoria");
107        exit(1);
108    }
109
110    // Generamos valores de paso
111    for (valor_actual = FIN; valor_actual <= INICIO; valor_actual += PASO) {
112        if (indice < numero_elementos) {
113            valores_paso[indice] = valor_actual;
114            indice++;
115        }
116    }
117
118    // Primer hijo para n=0 valor constante de 6
119    hijos[0] = fork();
120    if (hijos[0] == 0) {
121        // Proceso hijo
122        for (int i = 0; i < numero_elementos; i++) {
123            matriz_resultados[0][i] = a0;
124        }
125        exit(0);
126    }
127
128    // Generamos los demás hijos valores de n
129    for (int n = 1; n < (COLUMNAS - 1); n++) {
130        hijos[n] = fork();
131        if (hijos[n] == 0) {
132            for (int i = 0; i < numero_elementos; i++) {
133                double x = valores_paso[i];
134                double bn = funcion(n, x);
135
136                down(semaforo_general);
137                matriz_resultados[n][i] = bn;
138                // printf("n = %d, x = %.2f, bn = %.4f\n", n, x, bn);
139                up(semaforo_general);
140            }
141            shmdt(matriz_resultados);
142            exit(0);
143        }
144    }
145
146    // Esperar a que todos los hijos terminen
147    for (int n = 0; n < COLUMNAS; n++) {
148        if (hijos[n] != -1) {
149            waitpid(hijos[n], &status, 0);
150        }
151    }
152
153    // Limpiar memoria compartida
154    if (shmdt(matriz_resultados) == -1) {
155        perror("Error al desmontar memoria compartida");
156    }
157
158    // Salir del programa
159    exit(0);
160}
```

```
143         }
144     }
145
146     // Esperamos a que terminen los hijos
147     for (int i = 0; i < COLUMNAS - 1; i++) {
148         wait(NULL);
149     }
150
151     // Se calcula la suma de los resultados
152     for (int i = 0; i < numero_elementos; i++) {
153         double suma = 0.0;
154         for (int fila = 0; fila < COLUMNAS - 1; fila++) {
155             suma += matriz_resultados[fila][i];
156         }
157         matriz_resultados[11][i] = suma;
158     }
159
160     FILE *archivo = fopen("resultados.csv", "w");
161     if (archivo == NULL) {
162         perror("Error al crear archivo CSV");
163         exit(1);
164     }
165
166
167     // Se imprime la matriz resultados
168     printf("Matriz Resultados:\n");
169     for (int j = 0; j < numero_elementos; j++) {
170         printf("%.2f ", valores_paso[j]);
171         fprintf(archivo, "%.2f", valores_paso[j]);
172         for (int i = 0; i < COLUMNAS; i++) {
173             fprintf(archivo, ",%.4f", matriz_resultados[i][j]);
174             printf("%.4f ", matriz_resultados[i][j]);
175         }
176         printf("\n");
177         fprintf(archivo, "\n");
178     }
179     fclose(archivo);
180
181
182     // Desvinculamos la memoria compartida
183     if (shmctl(id_matriz, IPC_RMID, NULL) == -1) {
184         perror("Error al desvincular memoria compartida");
185         exit(1);
186     }
187     shmctl(id_matriz, IPC_RMID, NULL);
188     semctl(semaforo_general, 0, IPC_RMID);
189     return 0;
190 }
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <pthread.h>
5
6 #define INICIO 3.14
7 #define FIN -3.14
8 #define PASO 0.15
9 #define COLUMNAS 12
10
11 int numero_elementos;
12 double matriz_resultados[COLUMNAS][42];
13 double valores_paso[42]; // 43 elementos para el rango de -3.14 a 3.14 con paso de 0.15
14
15 double funcion(int n, double x) {
16     return (8.0 / n) * pow(-1, n) * sin(n * x);
17 }
18
19 typedef struct {
20     int n; // fila que calculara el hilo
21 } hilo_args_t;
22
23 void *calcular_fila(void *arg) {
24     hilo_args_t *args = (hilo_args_t *)arg;
25     int n = args->n;
26
27     if (n == 0) {
28         for (int i = 0; i < numero_elementos; i++) {
29             matriz_resultados[0][i] = 6.0;
30         }
31     } else {
32         for (int i = 0; i < numero_elementos; i++) {
33             double x = valores_paso[i];
34             matriz_resultados[n][i] = funcion(n, x);
35         }
36     }
37
38     free(args); // liberar memoria del struct
39     pthread_exit(NULL);
40 }
41
42 int main() {
43     int indice = 0;
44     for (double valor = FIN; valor <= INICIO; valor += PASO) {
45         valores_paso[indice++] = valor;
46     }
47     numero_elementos = indice;
48
49     pthread_t hilos[COLUMNAS - 1];
50 }
```

```
51     for (int n = 0; n < COLUMNAS - 1; n++) {
52         hilo_args_t *args = malloc(sizeof(hilo_args_t));
53         args->n = n;
54         if (pthread_create(&hilos[n], NULL, calcular_fila, args) != 0) {
55             perror("Error al crear hilo");
56             exit(EXIT_FAILURE);
57         }
58     }
59
60     for (int i = 0; i < COLUMNAS - 1; i++) {
61         pthread_join(hilos[i], NULL);
62     }
63
64     // Calcular suma final en la ultima fila
65     for (int i = 0; i < numero_elementos; i++) {
66         double suma = 0.0;
67         for (int j = 0; j < COLUMNAS - 1; j++) {
68             suma += matriz_resultados[j][i];
69         }
70         matriz_resultados[COLUMNAS - 1][i] = suma;
71     }
72
73     // Imprimir resultado
74     FILE *archivo = fopen("resultados.csv", "w");
75     if (!archivo) {
76         perror("Error al abrir resultados.csv");
77         exit(EXIT_FAILURE);
78     }
79
80     printf("Matriz Resultados:\n");
81     for (int j = 0; j < numero_elementos; j++) {
82         printf("%.2f ", valores_paso[j]);
83         fprintf(archivo, "%.2f", valores_paso[j]);
84         for (int i = 0; i < COLUMNAS; i++) {
85             printf("%.4f ", matriz_resultados[i][j]);
86             fprintf(archivo, ",%.4f", matriz_resultados[i][j]);
87         }
88         printf("\n");
89         fprintf(archivo, "\n");
90     }
91
92     fclose(archivo);
93     return 0;
94 }
```

Bibliografía

- [1] J. M. M. A. R. Thompson y G. W. Swenson, *Interferometry and Synthesis in Radio Astronomy*. Springer, 2017.
- [2] A. Arenas, Ó. Ciaurri, E. Labarga, L. Roncal y J. L. Varona, «Series de Fourier no trigonométricas: Una perspectiva familiar,» *Monográfico*, vol. 26, págs. 39-54, 2014. dirección: <https://www.unirioja.es/cu/jvarona/downloads/Fourier-familiar.pdf>.
- [3] C. Banwell y E. McCash, *Fundamentals of Molecular Spectroscopy*, 4th. McGraw-Hill, 1994.
- [4] N. Benbarka, N. Navab y B. Busam, «Seeing Behind Objects for 3D Multi-Object Tracking in RGB-D Sequences,» *arXiv preprint arXiv:2103.12091*, 2021. dirección: <https://arxiv.org/pdf/2103.12091.pdf>.
- [5] D. Bovet y M. Cesati, *Understanding the Linux Kernel*, 3rd. Sebastopol, CA: O'Reilly Media, 2005.
- [6] G. Box, G. Jenkins y G. Reinsel, *Time Series Analysis: Forecasting and Control*, 5th. Wiley, 2015.
- [7] S. Boyd y L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [8] R. Bracewell, *The Fourier Transform and Its Applications*, 3rd. McGraw-Hill, 2003.
- [9] D. Butenhof, *Programming with POSIX Threads*. Boston, MA: Addison-Wesley Professional, 1997.
- [10] A. Q. C. Canuto M. Y. Hussaini y T. A. Zang, *Spectral Methods: Fundamentals in Single Domains*. Springer, 2006.
- [11] J. Choi, C. Zhang, L. Anderegg, K. Willcox y K. Duraisamy, «Mathematics of Deep Learning for Fluid Dynamics,» *Annual Review of Fluid Mechanics*, vol. 55, págs. 575-608, 2023. dirección: <https://doi.org/10.1146/annurev-fluid-032822-025843>.
- [12] A. Chopra, *Dynamics of Structures: Theory and Applications to Earthquake Engineering*, 5th. Pearson, 2017.
- [13] B. Cullity y S. Stock, *Elements of X-Ray Diffraction*, 3rd. Pearson, 2014.
- [14] U. Drepper, «Every Computer Scientist Should Know About Memory,» *Red Hat, Inc.*, 2007.
- [15] R. C. Gonzalez y R. E. Woods, *Digital Image Processing*. Pearson, 2017.
- [16] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016.
- [17] D. Griffiths y D. Schroeter, *Introduction to Quantum Mechanics*, 3rd. Cambridge University Press, 2018.

- [18] E. Haacke, R. Brown, M. Thompson y R. Venkatesan, *Magnetic Resonance Imaging: Physical Principles and Sequence Design*, 2nd. Wiley, 2014.
- [19] B. Hall, *Beej's Guide to Network Programming: Using Internet Sockets*. Independently published, 2016.
- [20] S. Haykin y M. Moher, *Communication Systems*. Wiley, 2001.
- [21] M. Herlihy y N. Shavit, *The Art of Multiprocessor Programming*, Revised 1st. Burlington, MA: Morgan Kaufmann, 2012.
- [22] D. J. Inman, *Engineering Vibration*. Prentice-Hall, 2001.
- [23] M. Kerrisk, *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. San Francisco, CA: No Starch Press, 2010.
- [24] C. V. Loan, *Computational Frameworks for the Fast Fourier Transform*. SIAM, 1992.
- [25] R. Love, *Linux System Programming: Talking Directly to the Kernel and C Library*, 2nd. Sebastopol, CA: O'Reilly Media, 2013.
- [26] M. McGrath, D. Mueller y M. Samotyj, *Power Quality in Electrical Systems*. McGraw-Hill, 2017.
- [27] A. Metin, *Biomedical Signal Processing: A Conceptual Approach*. Academic Press, 2004.
- [28] K. Moreland y E. Angel, «The FFT on a GPU,» en *Proceedings of the ACM Workshop on Graphics Hardware*, 2003, págs. 112-119.
- [29] P. D. Nañez, «Series de Fourier,» Tutora: María del Mar Jiménez Sevilla, Trabajo Fin de Grado, Facultad de Ciencias Matemáticas, sep. de 2023.
- [30] M. A. Nielsen e I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
- [31] K. Ogata, *Modern Control Engineering*, 5th. Prentice Hall, 2010.
- [32] A. Oppenheim y R. Schafer, *Discrete-Time Signal Processing*, 3rd. Pearson, 2010.
- [33] A. V. Oppenheim y R. W. Schafer, *Discrete-Time Signal Processing*. Prentice Hall, 1999. dirección: https://research.iaun.ac.ir/pd/naghsh/pdfs/UploadFile_2230.pdf.
- [34] P. Pacheco, *An Introduction to Parallel Programming*. Burlington, MA: Morgan Kaufmann, 2011.
- [35] J. G. Proakis y M. Salehi, *Digital Communications*. McGraw-Hill, 2001.
- [36] K. Robbins y S. Robbins, *UNIX Systems Programming: Communication, Concurrency, and Threads*, 2nd. Upper Saddle River, NJ: Prentice Hall, 2003.
- [37] A. Silberschatz, P. Galvin y G. Gagne, *Operating System Concepts*, 10th. Hoboken, NJ: Wiley, 2018.
- [38] W. Stevens, B. Fenner y A. Rudoff, *UNIX Network Programming, Volume 2: Interprocess Communications*, 2nd. Boston, MA: Addison-Wesley Professional, 2005.
- [39] W. Stevens y S. Rago, *Advanced Programming in the UNIX Environment*, 3rd. Boston, MA: Addison-Wesley Professional, 2013.
- [40] A. Tanenbaum y H. Bos, *Modern Operating Systems*, 4th. Upper Saddle River, NJ: Pearson, 2015.