UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA ELECTRICA
DIVISIÓN DE ESTUDIOS DE POSGRADO

Selección Automática de los Parámetros Óptimos de Métodos de Pronóstico de Series de Tiempo usando Cómputo en Paralelo Basado en Hilos.

TESIS
Que para obtener el grado de
MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Presenta
Ing. Jesús de la Torre Bucio

Director de tesis
Dr. Antonio Ramos Paz

Co-Director de Tesis
Dr. J. Aurelio Medina Ríos

Morelia, Michoacán Agosto 2022

SELECCIÓN AUTOMÁTICA DE LOS PARÁMETROS ÓPTIMOS DE MÉTODOS DE PRONÓSTICOS DE SERIES DE TIEMPO USANDO CÓMPUTO EN PARALELO BASADO EN HILOS

Los Miembros del Jurado de Examen de Grado aprueban la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de **Jesús de la Torre Bucio.**

Dr. Juan Anzurez Marín
Presidente del Jurado

Dr. Antonio Ramos Paz
Director de Tesis

Dr. J. Aurelio Medina Rios
Co-director

Dr. Roberto Tapia Sánchez
Vocal

Dr. Fernando Martínez Cárdenas
Revisor Externo (ITM)

Dr. J. Aurelio Medina Rios
Jefe de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas)

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Agosto 2022

"Todo esfuerzo tiene su recompenza, chuchín"

José Jesús de la Torre Aguilera

Agradecimientos

Quiero agradecer a CONACYT por considerarme como becario. Esto fue de gran ayuda para continuar constantemente con mis estudios de posgrado.

Agradezco a todos los profesores de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica de la Universidad Michoacana de San Nicolás de Hidalgo por darme las bases para concluir con mi desarrollo en el posgrado.

Quiero agradecer a mi asesor el Dr. Antonio Ramos Paz por ayudarme y asesorarme en el desarrollo de mi tesis. Le agradezco la paciencia y la excelente atención que me brindo para desarrollar el trabajo de tesis. Debo admitir que es grato ser asesorado por una persona tan capacitada.

Quiero agradecer a mis compañeros del posgrado Barcenas, Aarón, Mario, Yañez, Víctor y Sergio por ayudarme a progresar en el proceso. Gracias por su paciencia, ayuda y especialmente por su convivencia, pude tener ese gran escalón para concluir con éxito.

Quiero agradecer a mis padres, hermanos y abuelita por ser un gran apoyo en todo este proceso de estudios.

Finalmente, agradezco a quien lee este apartado y más de mi tesis.

Resumen

Hoy en día es de vital importancia conocer cuándo y cómo ocurrirá un evento o surgirá una necesidad de modo que se tomen las mejores decisiones. El pronóstico es una herramienta importante para determinar la incertidumbre de los eventos, pudiéndose planificar y prevenir eventos antes de que sucedan. El pronóstico puede ser usado en diversas áreas. Por ejemplo, marketing, producción, eventos meteorológicos, etc. En el caso de los sistemas eléctricos, se puede utilizar en la planeación, operación del sistema o mantenimiento. En sí, puede utilizarse en las áreas donde se disponga de una variable que se registra en el tiempo. Es importante considerar el pronóstico en los sistemas eléctricos porque ayudan a la planificación futura de las instalaciones de producción de electricidad y para garantizar que las instalaciones existentes puedan satisfacer los picos de demanda. Actualmente, se dispone de software que realizan el pronóstico, pero carecen de una característica, el uso de una gran cantidad de datos genera un conflicto directamente con el tiempo de cómputo, ya que, entre más cantidad de datos se analice, más tiempo tarda en realizarse el pronóstico. Además, cada método de pronóstico requiere de un parámetro, el cual está influenciado con qué tan rápido va a responder el pronóstico ante los cambios de los datos históricos. Si no se tiene los conocimientos del método al momento de variar el parámetro, se consumirá tiempo para obtener los resultados que se desean. En este trabajo de tesis se presenta una serie de algoritmos que por medio del cómputo en paralelo basado en hilos seleccionarán el parámetro óptimo a través de una búsqueda exhaustiva, con el fin de obtener un parámetro que obtenga el menor error en el pronóstico y con ayuda de múltiples procesadores optimice el tiempo de cómputo. Esto permite usar a cualquier conjunto de datos históricos univariables utilizando los métodos de pronóstico conocidos como promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados, y Holt-Winters.

Palabras clave: Series de tiempo, Métodos de Pronóstico, Búsqueda Exhaustiva, Cómputo en paralelo, Threads.

Abstract

Nowadays it is of vital importance to know when and how an event will occur or a need will arise so that the best decisions can be made. Forecasting is an important tool to determine the uncertainty of events, so that events can be planned and prevented before they happen. Forecasting can be used in several areas. For example, marketing, production, weather events, etc. In the case of electrical systems, it can be used in planning, system operation or maintenance. In itself, it can be used in areas where a variable that is recorded over time is available. It is important to consider forecasting in power systems because it aids in the future planning of electricity production facilities and to ensure that existing facilities can meet peak demand. Currently, software is available that performs forecasting, but they lack a feature. The use of a large amount of data directly conflicts with the computation time, since the more data that is analyzed, the longer it takes to perform the forecast. In addition, each forecasting method requires a parameter, which is influenced by how quickly the forecast will respond to changes in the historical data. If you do not have the knowledge of the method, it will take time to vary the parameter to obtain the desired results. In this thesis work we present a series of algorithms that by means of parallel computation based on threads will select the optimal parameter through an exhaustive search, in order to obtain a parameter that obtains the lowest error in the forecast and with the help of multiple processors optimizes the computation time. This allows the use of any univariate historical data set using the well-known forecasting methods such as moving averages, weighted moving averages, exponential smoothing, least squares, and Holt-Winters.

Keywords: Time Series, Forecast Methods, Exhaustive Search, Parallel Computing, POSIX Threads.

Lista de Publicaciones

J. De La Torre Bucio, A. R. Paz and J. A. M. Ríos, "Analysis of Forecasting Methods of Time-Series with Increasing Trends," 2021 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), 2021, pp. 1-6, doi: 10.1109/ROPEC53248.2021.9668024.

J. De La Torre Bucio, A. R. Paz and J. A. M. Ríos, "Exhaustive Search Applied to Time Series Forecasting Methods Using Parallel Processing," artículo enviado para su revisión en The 2022 IEEE Autumn Meeting on Power, Electronics and Computing (ROPEC 2022).

Contenido

Lista de Abreviaturas

AHW	Holt-Winters Aditivo.
CPU	Unidad Central de Procesamiento.
ES	Suavización Exponencial.
HW	Holt-Winters.
LS	Mínimos Cuadrados.
MA	Promedios Móviles.
MAE	Error Promedio Absoluto.
MAPE	Error Porcentual Absoluto Medio.
MD	Memoria Distribuida.
ME	Error Promedio.
MHW	Holt-Winters Multiplicativo.
MIMD	Múltiples Instrucciones Múltiples Datos.
MISD	Múltiples Instrucciones Datos Únicos.
MPE	Error Porcentual Medio.
MSE	Error Cuadrático Medio.
SM	Memoria Compartida.
SIMD	Instrucción Simple Múltiples Datos.
SISD	Instrucción Única Datos Únicos.
TW	Ventana de Entrenamiento.
VW	Ventana de Validación.
WMA	Promedios Móviles Ponderados.

Lista de símbolos

$\hat{Y}_t$	Holt-Winters Aditivo.
Y_t	Valor real de la serie de tiempo en el periodo t.
e_t	Error del pronóstico en el periodo t.
n	Cantidad de datos de la serie de tiempo.
n_{wma}	Cantidad de términos de la serie de tiempo para el método de WMA.
υ	Cantidad de parámetros a interpolar.
$f_1(x)$	Variable dependiente de x de la interpolación lineal.
x	Valores intermedios de la interpolación lineal.
$f(x_0)$	Valor del eje vertical del punto 0.
$f(x_1)$	Valor del eje vertical del punto 1.
x_0	Valor del eje horizontal del punto 0.
x_1	Valor del eje horizontal del punto 1.
ρ	Define el grado de la función del método de pronóstico de LS.
ρ_{max}	Valor del orden máximo de la función de LS.
P	Número de procesadores.
γ	Parámetro del método de pronóstico de ES, define la constante de suavizamiento.
δ	Parámetros a determinar del método de LS.
ϕ	Parámetro del método de HW, define la constante de suavizamiento de nivel.
ψ	Parámetro del método de HW, define la constante de suavizamiento de tendencia.

s	Parámetro inicial del método de HW que representa la duración de la estacionalidad.
L_s	Valor inicial de nivel.
b_s	Valor inicial de tendencia.
S_s	Valor inicial estacional.
L_t	Representa el nivel de la serie de tiempo.
b_t	Es la tendencia de la serie de tiempo.
S_t	Componente estacional de la serie de tiempo.
γ	Cantidad de parámetros que se asignará a cada elemento de procesamiento.
n_{param}	Cantidad de parámetros totales.
T_s	Tiempo secuencial.
T_P	Tiempo usando P cantidad de procesadores.
α	Parámetro del método de pronóstico de MA, se basa en los últimos términos a promediar.
β	Parámetro del método de pronóstico de WMA, establece los pesos que llevan los últimos n términos.
ω	Parámetro del método de HW, define la constante de suavizamiento de estacionalidad.
m	Parámetro inicial del método de HW que representa los periodos t que hay por delante.

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Antecedentes

Desde épocas muy remotas, el pronóstico se ha utilizado para una infinidad de actividades, por ejemplo, saber cada cuánto tiempo se mantiene el sol en el día, cuánto tiempo dura cada estación del año, el clima que habrá en cada estación del año, etc. A medida que va creciendo la necesidad de la eficiencia de los métodos de pronóstico, se desarrollan distintas técnicas predictivas.

Hoy en día, el pronóstico es fundamental en varios ámbitos. Para el caso de los sistemas eléctricos, ayuda a planificar y anticipar eventos futuros que afectan a la red eléctrica. La importancia de los pronósticos cae principalmente para el operador del sistema eléctrico, ya que este mantiene un equilibrio continuo entre la energía que se genera y la que se consume [Monteiro, 2009]. Por ejemplo, se puede aplicar el pronóstico en la demanda eléctrica como lo muestra [Hyndman and Fan, 2010, Hamid and Rahman, 2010, Moraes et al., 2013, Niu et al., 2010], en los sistemas de energía renovable, donde [Jurj et al., 2018] muestra los distintos métodos de pronóstico que puede ser aplicado a estas fuentes de energía, en [Snegirev et al., 2017, Moustris et al., 2016] se presenta aplicaciones del pronóstico a las fuentes de energía solares y en [Szczupak et al., 2014] se muestra pronósticos aplicados a las fuentes de energía eólicas. También, el pronóstico puede aplicarse a la planeación en caso de que se tenga alteraciones de la energía como lo explica [Shan et al., 2017] o el impacto del acceso a gran escala de vehículos eléctricos a la red eléctrica urbana como describe [Yong et al., 2020]. En sí, el pronóstico puede aplicarse a cualquier evento de interés que sea incontrolable.

Es importante que la lectura de los datos y los cálculos de cada método de pronóstico sean precisos. Por esta razón, el uso de una herramienta computacional para la aplicación

de los métodos de pronóstico es esencial para la seguridad y confiabilidad de los datos obtenidos por el pronóstico. Existen estudios en los cuales se enfocan en el desarrollo de software para la aplicación de los métodos de pronóstico tales como [Manzano et al., 2019]. Además, se tiene una gran diversidad de software que pueden aplicar el pronóstico de las series de tiempo. Por ejemplo, Minitab, Excel, POM-QM, R, SPSS, Matlab, etc. Estos software comparten una característica, al trabajarse con una serie de tiempo de gran cantidad de datos, nos lleva a gran tiempo computacional, además, el empleo de métodos de pronóstico requiere definir uno o más parámetros iniciales con el fin de obtener un pronóstico con menor error. Estos parámetros (dependiendo del método que se aplique), varían dentro de un rango, seleccionando a través de prueba y error. Si no se tiene la experiencia necesaria, este tipo de pruebas puede tomar tiempo. Siendo necesario establecer un algoritmo que realice la selección del parámetro óptimo a partir de la búsqueda exhaustiva en el pronóstico de las series de tiempo. En la literatura, se ha realizado pocos estudios donde aplican la búsqueda exhaustiva en los pronósticos. Por ejemplo, en [Kalos, 2005] presenta un método para seleccionar automáticamente la arquitectura óptima de redes neuronales feedforward para construir modelos de series de tiempo no lineales, utilizando un método heurístico para hacer una búsqueda exhaustiva y aplicarlo a los pronósticos diarios de precios de energía. También en [Song et al., 2019] muestra un modelo avanzado de control predictivo en un motor eléctrico de una turbina eólica industrial y emplea un método de búsqueda exhaustiva para buscar el candidato de salida de control que proporciona el mínimo valor de la función objetivo. Actualmente, no se encuentran estudios donde realice la selección del parámetro óptimo a partir de una búsqueda exhaustiva en los métodos de pronóstico de series de tiempo como promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados y Holt-Winters.

La desventaja que puede tener al considerar la búsqueda exhaustiva en la selección del parámetro óptimo en los métodos de pronóstico es el esfuerzo computacional, ya que este, por la gran cantidad de combinaciones que hace, no es tan viable en su tiempo de ejecución [Utans et al., 1995, Skolpadungket et al., 2009, Yizhen et al., 2011]. Este problema se puede resolver de diversas maneras, en este caso, se hace uso del cómputo en paralelo con el fin de optimizar el tiempo de ejecución.

El cómputo paralelo ha existido de una forma u otra durante muchas décadas. En sus principios generalmente se limitaba a los profesionales que tenían acceso a máquinas grandes y costosas. Hoy en día, todo es diferente, ya que la mayor parte de las computadoras tienen unidades de procesamiento central, o CPU con varios núcleos. Pudiéndose tener mayor

acceso a estas tecnologías que permiten el cómputo en paralelo [Fatica and Ruetsch, 2014]. Existen una diversidad de arquitecturas de computadoras que permiten el desarrollo de programas paralelos, en este caso, hacemos enfoque a las computadoras que contienen múltiples elementos de procesamiento. Específicamente se planea trabajar con la plataforma Pthreads. Para el caso del cómputo paralelo en la plataforma Threads, hay pocos casos de estudio en la literatura para el pronóstico de las series de tiempo. Se tiene [Maity et al., 2013] que aplica el pronóstico de series de tiempo en modelos meteorológicos y [Barbhuiya and Liang, 2012] que aplica el procesamiento en paralelo a pronósticos del tiempo. Siendo un área importante de investigación.

Con base en lo establecido anteriormente, se muestra la necesidad de la herramienta del cómputo en paralelo basado en hilos para aplicarlo en la selección automática del parámetro óptimo por medio de la búsqueda exhaustiva de los métodos de pronóstico de series de tiempo propuestos en este trabajo de tesis, ya que, es la aportación que se pretende realizar.

1.2. Objetivos

1.2.1. Objetivo general

Optimizar el método de pronóstico de series de tiempo de promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados, y Holt-Winters, en el lenguaje de programación C haciendo uso del cómputo paralelo basado en hilos por medio de la plataforma Pthreads para obtener el pronóstico en series de tiempo univariables basándose en la selección del parámetro óptimo a partir de una búsqueda exhaustiva.

1.2.2. Objetivos particulares

1. Optimizar el método de pronóstico de series de tiempo de promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados, y Holt-Winters, a través de la selección del parámetro óptimo por medio de una búsqueda exhaustiva.

2. La selección del parámetro óptimo de la propuesta de optimización de los métodos de pronóstico de series de tiempo reducirá al mínimo la métrica de error MAPE.

3. El uso del cómputo paralelo basado en hilos en la plataforma Pthreads permitirá obtener menores tiempos de cómputo en la propuesta de optimización de los métodos de pronóstico de series de tiempo.

4. En cada caso de estudio, realizar una comparación entre los métodos de pronóstico de series de tiempo optimizados por medio de la métrica MAPE para observar cuál de ellos obtiene mayor aproximación a la ventana de validación de la serie de tiempo.

5. En cada caso de estudio, realizar una comparación de la optimización del tiempo de cómputo de cada método de pronóstico utilizando distinta cantidad de elementos de procesamiento. Esto a través de la métrica *speed up*.

1.3. Hipótesis

La aplicación del cómputo en paralelo basado en hilos en la selección automática de los parámetros óptimos que tiene cada método de pronóstico de series de tiempo permitirá obtener un pronóstico confiable, disminuyendo el tiempo de ejecución en comparación de los programas secuenciales.

1.4. Justificación

Basándonos en lo expuesto de los antecedentes, se observa que existen distintos trabajos de investigación en dónde se ha realizado el pronóstico de series de tiempo, sin embargo, son pocos los estudios donde usan del cómputo en paralelo basado en hilos para el pronóstico, además, no existen estudios del uso de la búsqueda exhaustiva de parámetros para la selección del parámetro óptimo de los métodos de pronóstico de series de tiempo propuestos en este trabajo. Siendo un área que se pretende optimizar. El pronóstico es una herramienta primordial en los sistemas eléctricos, ya que, permiten la planificación futura de las instalaciones de generación de electricidad y para garantizar que las instalaciones existentes puedan satisfacer los picos de demanda. Siendo así, se propone el diseño de una serie de algoritmos paralelos que realizan la selección automática del parámetro óptimo a través de una búsqueda exhaustiva en sus parámetros, permitiendo obtener un pronóstico con el mínimo error de los métodos de pronóstico de series de tiempo de promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados, y Holt-Winters, haciendo uso del cómputo en paralelo basado en hilos para optimizar el tiempo de ejecución. Esta contribución no solo permitirá realizar la planeación, operación y mantenimiento en los sistemas eléctricos; sino también podrá aplicarse a cualquier serie de tiempo univariable. Además, los algoritmos paralelos propuestos en este trabajo de tesis pueden trasladarse a otras plataformas debido al lenguaje de programación donde están desarrollados.

1.5. Metodología

La primera etapa se asocia a la investigación para conocer los diferentes tipos y características de los métodos de pronóstico de series de tiempo. Además de conocer sobre los patrones de series de tiempo y cómo puede usarse este conjunto de datos para el pronóstico. La segunda etapa trata sobre la selección de un conjunto de métodos de pronóstico de series de tiempo tradicionales. En este caso, se consideraron el método de promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados y Holt-Winters. Asimismo, se realiza su implementación en el lenguaje de programación C. La tercera etapa está asociada a la programación en C de los algoritmos que disponen los métodos de pronóstico de series de tiempo propuestos, pero considerando un algoritmo que realice la selección automática del parámetro óptimo a partir de una búsqueda exhaustiva. La cuarta etapa consiste en hacer la investigación del cómputo en paralelo, las arquitecturas que dispone y metodologías para el desarrollo de programas paralelos. Además, de conocer cómo utilizar el cómputo en paralelo basado en hilos, sus características, funciones y estados que tienen los hilos. En la quinta etapa, se hizo el algoritmo paralelo en C que realiza la selección automática del parámetro óptimo a partir de una búsqueda exhaustiva del método de pronóstico de series de tiempo de promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados y Holt-Winters, haciendo uso del cómputo en paralelo basado en hilos. La sexta etapa se aplicó los algoritmos paralelos a tres casos de estudio con el fin de mostrar una comparación del tiempo de ejecución usando distinta cantidad de elementos de procesamiento por medio del *speed up* y comparar los resultados obtenidos de los métodos de pronóstico de series de tiempo usando su parámetro óptimo en el pronóstico. Por último, en la séptima etapa, se establecieron las conclusiones y trabajos futuros. En la Figura 1.1 se muestra un diagrama generalizado de la metodología propuesta.

1.6. Aportación Original

Diseño y aplicación de la optimización del método de pronóstico de series de tiempo de promedios móviles, promedios móviles ponderados, suavización exponencial, mínimos cuadrados, y Holt-Winters, en el lenguaje de programación C haciendo uso del cómputo paralelo basado en hilos por medio de la plataforma Pthreads para obtener el pronóstico en series de tiempo univariables basándose en la selección del parámetro óptimo a partir de una búsqueda exhaustiva.

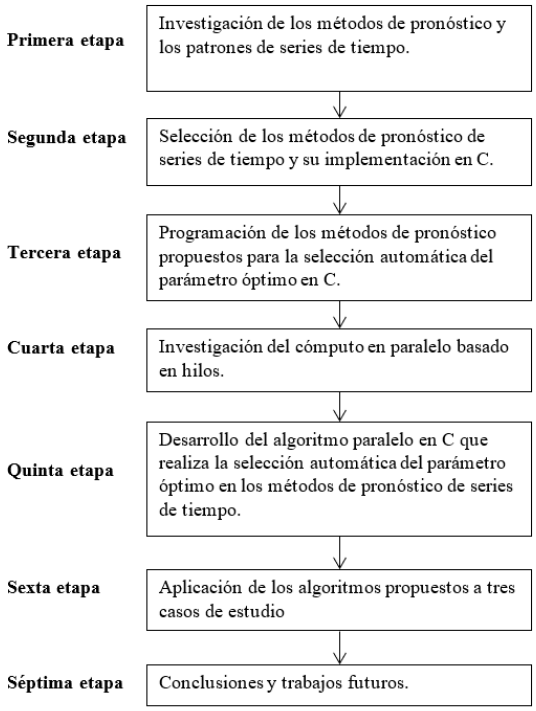

Figura 1.1: Diagrama generalizado de la metodología.

1.7. Descripción de los Capítulos

El Capítulo 2 presenta una parte introductoria para mostrar conceptos como: métodos básicos de pronóstico, series de tiempo, categoría de los métodos de pronóstico y la métrica de error para el pronóstico. Además, se presentan las características, formulación, algoritmos, ventajas y desventajas de cada método de pronóstico utilizado en este trabajo de tesis.

El Capítulo 3 presenta una parte introductoria para mostrar la importancia y conceptos básicos del cómputo en paralelo. Además, presenta las arquitecturas para el cómputo en paralelo, metodología de diseño de algoritmos paralelos, granularidad, conceptos básicos para el uso de los hilos y la métrica para la velocidad de procesamiento.

En el Capítulo 4 se presenta la propuesta del cómputo en paralelo basado en hilos de los algoritmos que realizan la selección automática del parámetro óptimo de los métodos de pronóstico de series de tiempo propuestos en esta tesis. Se muestra el algoritmo, diagramas de flujo, y parámetros que se usaron para su desarrollo.

En el Capítulo 5 se realizan tres casos de estudio que presentan la ventaja del cómputo en paralelo basado en hilos en la aplicación de la selección automática del parámetro óptimo para los métodos de pronóstico de series de tiempo de MA, WMA, ES, LS y HW. También, se muestra las características del equipo de cómputo donde se hicieron las pruebas, la

metodología que se sugirió para aplicar el speed up, el empleo de la interpolación lineal para generar datos artificiales en la serie de tiempo y la manera que se hace la comparación de los métodos de pronóstico de series de tiempo.

En el Capítulo 6 se presentan las conclusiones asociadas al trabajo de tesis y las recomendaciones a trabajos futuros sobre el pronóstico de series de tiempo.

Capítulo 2

Métodos de Pronóstico de Series de Tiempo

2.1. Introducción

Por lo general, existe un lapso de tiempo entre el conocimiento de un evento y la ocurrencia del mismo. Este tiempo de espera es el principal motivo para planificar y pronosticar el evento. Si el tiempo de espera es muy corto (cercano a cero), no es necesario la planificación. Siendo inevitable un tiempo largo de espera y disponer de factores identificables del evento para que la planificación pueda desempeñar un papel importante. En estas situaciones, el pronóstico es necesario para determinar cuándo y cómo va a ocurrir el evento, y así, se puedan tomar las mejores decisiones. Siendo relevante el pronóstico para lograr una planificación eficiente [Hanke and Wichern, 2006, Wheelwright et al., 1998].

En términos generales, los modelos de pronóstico se pueden clasificar en métodos cuantitativos y cualitativos. Para el caso de los *métodos cualitativos*, estos no requieren de alguna manipulación de datos, ya que, solamente necesita del juicio de quien pronostica para aplicar el método. Por lo general, se aplican cuando existe una escasez de información de los datos o si la información a la que se accede no es clara. Con estos métodos cualitativos es de vital importancia la experiencia. En el caso de los *métodos cuantitativos*, estos no requieren elementos de juicio; son modelos matemáticos que disponen de datos históricos para poder aplicar el método que se requiera. Hay algunos modelos cuantitativos que requieren una manipulación más sofisticada de los datos que otros modelos [Hanke and Wichern, 2006]. De este modo, en este trabajo de tesis, se considera un conjunto de métodos cuantitativos que se basan en la aplicación de series de tiempo, debi-

do a que tienen conocimiento de datos históricos para tener una perspectiva del futuro [Wheelwright et al., 1998, Hanke and Wichern, 2006].

Esta sección pretende proporcionar los conceptos básicos para el análisis de series de tiempo, el uso de los métodos de pronóstico a estas series de tiempo y la métrica para evaluar la eficiencia del método de pronóstico. Además, en el Apéndice A, se presenta los diagramas de flujo que corresponde a los algoritmos secuenciales de cada método de pronóstico de serie de tiempo propuestos en esta tesis.

2.1.1. Series de tiempo

Las *series de tiempo* son un conjunto de datos capturados de manera secuencial a lo largo de incrementos sucesivos de tiempo [Hanke and Wichern, 2006]. Por ejemplo, las cifras de ventas mensuales, los precios de las acciones diarias, las tasas de interés semanales, las ganancias anuales, las temperaturas máximas diarias, la producción anual de cultivos y las mediciones del electrocardiógrafo [Wheelwright et al., 1998]. La forma más eficiente de presentar las series de tiempo es gráficamente. De esta manera, se puede revelar cualquier patrón que exista a lo largo del tiempo. Las gráficas que representan el comportamiento de los datos, siempre tendrán dos dimensiones. Por ejemplo, en la Figura 2.1 se muestra una serie de tiempo obtenida del entorno de R. Esta serie proporciona los totales mensuales de los pasajeros de una aerolínea estadounidense desde 1949 a 1960, donde en el eje horizontal muestra la frecuencia de las mediciones o también conocido como periodo de tiempo, mientras que en el eje vertical representa la variable de interés.

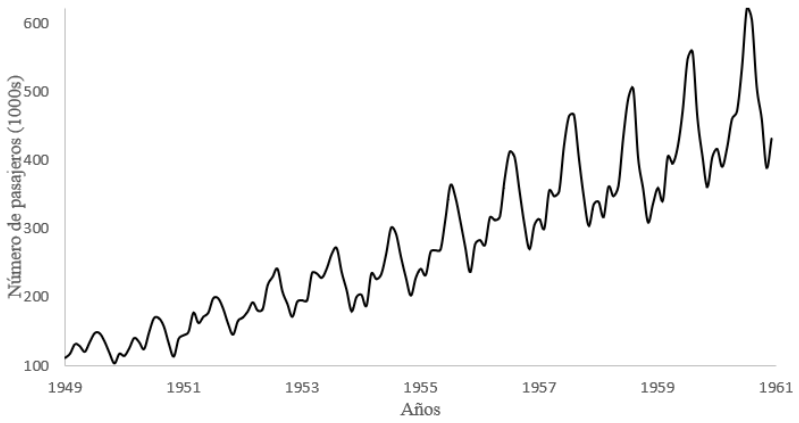

Figura 2.1: Total mensual de pasajeros de una aerolínea de EE. UU. (1949-1960) en miles.

Los modelos de series de tiempo pueden ser de manera *univariable* y *multivariable*. Cuando se habla de una serie de tiempo univariable, se trata del análisis de un conjunto de

datos que describen una sola variable. En el caso de una serie de tiempo multivariable, es cuando se analizan varias series de tiempo a la vez. En este trabajo de tesis se consideran únicamente los métodos de pronóstico de series de tiempo univariable.

Una serie de tiempo univariable es un conjunto de datos de una variable Y, cuyo subíndice está relacionado con una observación. Siendo así, se tiene que Y_t es el valor de una serie de tiempo en el periodo t. Si hay n observaciones en la serie de tiempo, se denota como:

$$Y_t \quad \text{donde} \quad t = 1, 2, 3, \ldots, n \tag{2.1}$$

Es importante que los datos que dispone la serie de tiempo sean confiables, representativos y que mantengan la frecuencia con las que se realizan las observaciones. Generalmente, los datos se toman en intervalos equidistantes de tiempo, conociéndose como *series de tiempo discretas*. Por ejemplo, se dispone de datos capturados cada minuto, hora, día, mes, etc. Si los datos se pueden generar y observar de forma continua, disponiéndose de un número infinito de observaciones, se trata del caso de las *series de tiempo continuas*. La mayoría del conjunto de datos que se disponen para su estudio se observan en tiempo discreto. Por lo tanto, en este trabajo de tesis se hace enfoque al estudio de variables discretas capturadas en intervalos regulares de tiempo [Hanke and Wichern, 2006, Casimiro, 2009].

2.1.2. Patrones en series de tiempo

De manera general, se pueden definir cuatro tipos diferentes de patrones en series de tiempo: horizontal, tendencia, estacional y cíclico [Wheelwright et al., 1998].

Un patrón de *tendencia*, es cuando hay un aumento o disminución a largo plazo del valor de los datos [Wheelwright et al., 1998, Hanke and Wichern, 2006]. La existencia de un patrón *horizontal*, es cuando el conjunto de datos de una serie de tiempo fluctúa a través de un nivel constante o medio (también conocida como "estacionaria") y además contienen una varianza constante [Wheelwright et al., 1998, Hanke and Wichern, 2006]. El patrón *estacional*, es cuando la serie de tiempo está influenciada por factores estacionales, como lo son los semestres, año, mes, o días de la semana. Las series estacionales a veces también se denominan "periódicas", aunque no se repiten exactamente en cada período [Wheelwright et al., 1998, Hanke and Wichern, 2006]. Cuando las observaciones suben y bajan en periodos distintos, se les conoce como patrones *cíclicos*. Los patrones cíclicos y estacionales podrían crear confusiones debido a que presentan comportamientos muy similares. La distinción que hay entre ellos, es que, en el caso de los patrones estacionales, tiene duraciones constantes y se repite periódicamente, mientras que en los patrones cíclicos

varía en duración [Wheelwright et al., 1998, Hanke and Wichern, 2006].

2.1.3. Ventana de entrenamiento y validación

La finalidad de los métodos de pronóstico es predecir observaciones futuras, pero puede que las predicciones no sean tan eficientes al considerar ciertos parámetros. Para obtener el pronóstico con menor error de cada método, es necesario que los datos totales de la serie de tiempo se separen en un conjunto de "inicialización" o mejor conocido como *ventana de entrenamiento* (TW, del inglés *training window*) y en un conjunto de "prueba" o mejor conocido como *ventana de validación* (VW, del inglés *validation window*). El conjunto de datos más grande suele ser la TW. Este conjunto de datos, como su nombre lo dice, nos permite entrenar el método de pronóstico. Es utilizado para estimar cualquier parámetro y para inicializar el método. En el caso de la VW, este conjunto de datos no se usó en el ajuste del modelo, aplicándose para conocer qué tan eficiente es el método con base a la métrica de error [Wheelwright et al., 1998]. Es decir, teniendo la serie de tiempo, se dividen los datos en la TW y VW. Con la TW se forma el modelo, realizando el pronóstico. Posteriormente, con la ventana de validación, se compara qué tan eficiente es el método.

2.1.4. Medición de la precisión del pronóstico

La eficiencia de los métodos de pronóstico de series de tiempo recae en el tipo de comportamiento que tenga el conjunto de datos donde se aplica el pronóstico. La eficiencia de cada método de pronóstico es calculada de forma cuantitativa a través de métricas de error que se basan en la diferencia del valor real de la serie de tiempo en un periodo t y el pronóstico en el mismo periodo t. Con esto, se puede calcular qué método es más eficiente con base al menor valor que obtenga la métrica [Hanke and Wichern, 2006, Wheelwright et al., 1998].

Existen métricas que pueden utilizarse para evaluar la eficiencia de los métodos de pronóstico de series de tiempo. Estas métricas son: Error promedio (ME, del inglés *mean error*), error promedio absoluto (MAE, del inglés *mean absolute error*), error cuadrático medio (MSE, del inglés *mean squared error*), error porcentual absoluto medio (MAPE, del inglés *mean absolute porcentage error*) y el error porcentual medio (MPE, del inglés *mean porcentage error*). Con el fin de interpretar los resultados de manera más sencilla, se usa la métrica MAPE para representar sus resultados porcentualmente [Hanke and Wichern, 2006, Wheelwright et al., 1998].

La métrica MAPE se calcula mediante:

$$MAPE\,\% = \frac{1}{n}\sum_{t=1}^{n}\left(\frac{|Y_t - \hat{Y}_t|}{Y_t}\cdot 100\right) \tag{2.2}$$

donde n en este caso es la cantidad de datos que dispone la VW, $\hat{Y}_t$ es el pronóstico en el periodo t y Y_t es el valor real en el periodo t.

2.2. Métodos de Pronóstico de Series de Tiempo

El pronóstico es necesario para poder determinar cuándo va a ocurrir un evento, de esta forma, se pueden tomar acciones apropiadas en el momento y en la aplicación que se requiera [Wheelwright et al., 1998]. Es importante destacar que, en este trabajo de tesis, se describe cada método de pronóstico de series de tiempo en dos partes: modelo y pronóstico. En el caso del *modelo*, se trata de la aplicación del método de pronóstico a los datos que conforma la TW, de esta manera, se entrena el método de pronóstico que está sometido a un parámetro para formar el modelo. Con base al modelo, se pueden realizar pruebas para ver qué tan certero es el parámetro seleccionado. En el caso del *pronóstico*, se usa el modelo para realizar el pronóstico. Este pronóstico tiene las mismas dimensiones que el conjunto de datos que está conformado por la VW.

En los algoritmos de los métodos de pronóstico, se presentan dos conceptos que son relevantes para la descripción de los métodos, los cuales son: frecuencia y periodos. Los *periodos* es la escala de tiempo de referencia, esta puede establecerse ya sea por día, mes, bimestre, trimestre, semestre, anual, etc. La *frecuencia* es la cantidad de datos que dispone cada periodo.

A continuación, se presentan los métodos de pronóstico de series de tiempo considerados en este trabajo de tesis. En cada método se muestran sus características, formulación y el algoritmo para su aplicación.

2.2.1. Promedios móviles

El método de pronóstico basado en los promedios móviles (MA, del inglés *moving averages*) promedia los valores más recientes con el fin de suavizar las fluctuaciones. El método de MA especifica el valor de un parámetro, α, el cual se utiliza para promediar los datos más recientes. En otras palabras, en cada nuevo punto de datos, se incluye en el promedio al estar disponible y se elimina el punto de datos más antiguo [Hanke and Wichern, 2006]. El método de pronóstico de MA considera pesos iguales en cada dato de la serie de tiempo y no se comporta adecuadamente cuando se tiene series de tiempo que contienen patrones de tendencia o estacionalidad. Siendo así, es de gran importancia determinar el tamaño de α. Cuanto menor sea el valor de α, mayor peso se les dará a los periodos recientes,

siendo deseable cuando existen cambios repentinos en el nivel de la serie de tiempo. Por el contrario, cuanto mayor sea el valor de α se les dará menor peso a los periodos más recientes, siendo deseable cuando hay fluctuaciones amplias y poco frecuentes en la serie [Hanke and Wichern, 2006].

El modelo y pronóstico del método de MA se calcula mediante:

$$\hat{Y}_{t+1} = \frac{Y_t + Y_{t-1} + \ldots + Y_{t-\alpha+1}}{\alpha} \tag{2.3}$$

donde $\hat{Y}_{t+1}$ es el valor del modelo o pronósticos para el siguiente periodo t, Y_t es el valor real en el periodo t y α es el número de términos a promediar.

Para el modelo de MA, Los valores de los periodos, t, menores a α son cero, siendo así se tiene que:

$$\hat{Y}_1,\ \hat{Y}_2, \ldots,\ \hat{Y}_\alpha = 0 \tag{2.4}$$

donde $\hat{Y}_1,\ \hat{Y}_2, \ldots,\ \hat{Y}_\alpha$ son valores del modelo y α es el número de términos a promediar. Para el valor de los periodos t mayores a α, se calculan mediante (2.3) hasta llegar a un periodo $t = n$. Para el pronóstico de MA, se usa (2.3) en un periodo $t = n + 1$.

2.2.2. Promedios móviles ponderados

El método de pronóstico basado en los promedios móviles ponderados (WMA, del inglés *weighted moving averages*) es considerado una versión mejorada del método de MA. Este implementa unas constantes, β, donde asigna un peso a los datos inmediatos más recientes de la serie de tiempo [Hansun and Kristanda, 2017]. En este método, la idea para obtener el pronóstico de un periodo es considerar n_{wma} cantidad de datos $(\beta_1, \beta_2, \ldots, \beta_{n_{wma}})$. A diferencia del método de pronóstico de MA, este aplica mayor peso a los n_{wma} datos más recientes, y el peso disminuye para los datos más antiguos. Es importante que la suma de los pesos, β, sea igual al 100 %. El método de WMA tiene mejor aproximación en los patrones de tendencia que el método de MA. Esto se debe porque el método de MA asigna pesos idénticos a todos los datos de la serie de tiempo.

El modelo y pronóstico del método de WMA se calcula mediante:

$$\hat{Y}_{t+1} = (\beta_1 \cdot Y_t) + (\beta_2 \cdot Y_{t-1}) + \ldots + (\beta_{n_{wma}} \cdot Y_{t-n_{wma}+1}) \tag{2.5}$$

donde $\hat{Y}_{t+1}$ es el valor del modelo o pronóstico para el siguiente periodo t, Y_t es el valor real en el periodo t y $\beta_1 + \beta_2 + \ldots + \beta_{n_{wma}}$ son los pesos aplicados.

Para formar el modelo de WMA, el valor del periodo, t, menor a n_{wma} son cero

$$\hat{Y}_1,\ \hat{Y}_2, \ldots,\ \hat{Y}_{n_{wma}} = 0 \tag{2.6}$$

donde $\hat{Y}_1,\ \hat{Y}_2, \ldots,\ \hat{Y}_{n_{wma}}$ son valores del modelo considerando n_{wma} cantidad de términos. Para el valor de los periodos, t, mayores a n_{wma}, se calculan mediante (2.5) hasta llegar a un periodo $t = n$. Para calcular el pronóstico de WMA, se usa (2.5) en un periodo $t = n+1$.

2.2.3. Suavización exponencial

El método de pronóstico de suavización exponencial (ES, del inglés *exponential smoothing*) proporciona un promedio móvil con un peso exponencial a todos los valores observados con anterioridad. Este método se enfoca en suavizar los valores pasados más recientes en una forma exponencialmente decreciente, teniendo todos los datos un peso sobre el pronóstico. El periodo más reciente tiene un peso, γ, y los demás periodos decrecen en su magnitud, mientras más alejados están del momento actual, su peso es cada vez menor. Además, el peso de cada dato no decae de manera lineal, sino exponencial, es por eso la razón del nombre del método [Hanke and Wichern, 2006].

Por otra parte, γ es el peso que recibe el último dato que se ha obtenido, mientras mayor sea este peso mayor será su influencia en el pronóstico y podrá alterar de manera más significativa. Es deseable cuando se requiere una respuesta rápida a un cambio en el patrón de los datos de la serie. Si el valor de γ es pequeño, entonces los cambios no serán tan drásticos y la curva se suavizará, siendo deseable cuando se desea predicciones estables y que suavicen las variaciones aleatorias. El método de ES, es apropiado cuando se tiene series de tiempo que presentan patrones de tendencia no predecibles tanto ascendentes como descendentes en el tiempo [Hanke and Wichern, 2006].

El modelo y pronóstico del método de ES se calcula mediante:

$$\hat{Y}_{t+1} = \gamma Y_t + (1-\gamma)\hat{Y}_t \tag{2.7}$$

donde $\hat{Y}_{t+1}$ es el valor del modelo o pronóstico para el siguiente periodo, t, γ es la constante de suavizamiento, Y_t es el valor real de la serie de tiempo en el periodo t y $\hat{Y}_t$ es el valor del modelo el periodo t.

En el modelo de ES, su valor del primer periodo siempre será cero y el valor del segundo periodo es el primer dato de la serie de tiempo, siendo así se tiene:

$$\hat{Y}_1 = 0 \tag{2.8}$$

$$\hat{Y}_2 = Y_1 \tag{2.9}$$

donde $\hat{Y}_1$ es el valor del modelo en el primer periodo, $\hat{Y}_2$ es el valor del modelo del segundo periodo y Y_1 es el primer dato de la serie de tiempo. Ahora, cuando se tiene un valor de

$t > 2$, se obtienen los valores del modelo usando (2.7) hasta llegar al valor de $t = n$. Para el caso del pronóstico de ES, se calcula mediante (2.7) en un periodo de $t = n + 1$.

2.2.4. Mínimos cuadrados

El método de pronóstico de mínimos cuadrados (LS, del inglés *Least Squares*) tiene como función principal dar una aproximación por medio de una función polinomial de orden ρ a un conjunto de puntos x_t y y_t. Considerando que n es el número total de datos de la serie, se tiene que $t = 1, 2, 3, ..., n$, y que x_t y y_t es el tiempo y la variable a analizar, respectivamente.

El modelo del método LS se forma a través de una función polinomial de orden ρ

$$\hat{Y}_t = \delta_0 + \delta_1 x_t + \delta_2 x_t^2, \ldots + \delta_\rho x_t^p \tag{2.10}$$

donde $\hat{Y}_t$ es el valor del modelo en el periodo t; $x_t, \ldots, x_t^p$ son el conjunto de valores que representan el tiempo; $\delta_0, \ldots, \delta_\rho$ son los parámetros a determinar y ρ es el orden del polinomio. Con base en (2.10), se puede llegar a una aproximación de un conjunto de datos x_t y y_t) por medio de una función polinomial de cualquier orden, determinando las incógnitas $\delta_0, \ldots, \delta_\rho$ [Lay, 2012]. Para obtener los valores de $\delta_0, \ldots, \delta_\rho$ se realiza mediante:

$$Ad = b \tag{2.11}$$

donde

$$A = \begin{bmatrix} n & \sum_{t=1}^n x_t & \cdots & \sum_{t=1}^n x_t^\rho \\ \sum_{t=1}^n x_t & \sum_{t=1}^n x_t^2 & \cdots & \sum_{t=1}^n x_t^{\rho+1} \\ \sum_{t=1}^n x_t^2 & \sum_{t=1}^n x_t^3 & \cdots & \sum_{t=1}^n x_t^{\rho+2} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{t=1}^n x_t^\rho & \sum_{t=1}^n x_t^{\rho+1} & \cdots & \sum_{t=1}^n x_t^{\rho+\rho} \end{bmatrix}, \; d = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \\ \vdots \\ \delta_\rho \end{bmatrix} \text{ y } b = \begin{bmatrix} \sum_{t=1}^n y_t \\ \sum_{t=1}^n x_t y_t \\ \sum_{t=1}^n x_t^2 y_t \\ \vdots \\ \sum_{t=1}^n x_t^\rho y_t \end{bmatrix}$$

En (2.11) los coeficientes del sistema A y b se obtienen a través de los datos de x_t y y_t. Para el cálculo del vector de incógnitas, d, se resuelve el sistema $A^{-1}b = d$ invirtiendo la matriz A a través del método de Gauss-Jordan y su resultado multiplicado por el vector b. Teniendo el valor de las incógnitas, se puede realizar el pronóstico de LS mediante (2.10) en un periodo $t = n + 1$.

2.2.5. Holt-Winters

El método de pronóstico de series de tiempo de Holt-Winters (HW) es un método que se enfoca en los patrones de tendencia y estacionalidad. Existen dos tipos de modelos de HW:

Una versión aditiva conocida como Holt-Winters aditivo (AHW, del inglés *aditive Holt-Winters*) que supone que los efectos estacionales son de tamaño constante y una versión multiplicativa conocida como Holt-Winters multiplicativo (MHW, del inglés *multiplicative Holt-Winters*) que supone que los efectos estacionales varían proporcionalmente a la tendencia [Chatfield, 1978, Wheelwright et al., 1998]. El método de HW realiza su modelo y pronóstico a partir de tres parámetros: nivel (ϕ), tendencia (ψ) y estacionalidad (ω). A medida que se dispone de cada nueva observación, los parámetros se actualizan mediante un suavizado exponencial. Luego, se pueden producir pronósticos para cualquier m pasos por delante [Chatfield, 1978, Wheelwright et al., 1998].

Para el modelo del método de pronóstico de series de tiempo de HW, su valor de los periodos t menores a s van a ser cero, siendo así se tiene:

$$\hat{Y}_t = 0 \quad \text{donde} \quad t = 1, 2, \ldots, s-1 \tag{2.12}$$

Posteriormente, se hace el cálculo de los valores iniciales de nivel, tendencia y estacionalidad. Para el caso de MHW, sus valores iniciales se calculan mediante:

$$L_s = \frac{1}{s}(Y_1 + Y_2 + \ldots + Y_s) \tag{2.13}$$

$$b_s = \frac{1}{s}\left[\frac{Y_{s+1} - Y_1}{s} + \frac{Y_{s+2} - Y_2}{s} + \ldots + \frac{Y_{s+s} - Y_s}{s}\right] \tag{2.14}$$

$$S_1 = \frac{Y_1}{L_s},\ S_2 = \frac{Y_2}{L_s}, \ldots,\ S_s = \frac{Y_s}{L_s} \tag{2.15}$$

Para el caso del método de AHW sus valores iniciales se calculan mediante:

$$L_s = \frac{1}{s}(Y_1 + Y_2 + \ldots + Y_s) \tag{2.16}$$

$$b_s = \frac{1}{s}\left[\frac{Y_{s+1} - Y_1}{s} + \frac{Y_{s+2} - Y_2}{s} + \ldots + \frac{Y_{s+s} - Y_s}{s}\right] \tag{2.17}$$

$$S_1 = Y_1 - L_s,\ S_2 = Y_2 - L_s,\ \ldots,\ S_s = Y_s - L_s \tag{2.18}$$

donde s es la longitud de tiempo de estacionalidad, L_s es el valor inicial del nivel en el periodo $t = s$, Y_s valor real de la serie de tiempo en el periodo $t = s$ y S_s es el valor estacional en el periodo $t = s$.

Posteriormente, en el modelo se realizan los cálculos del nivel, tendencia y estacionalidad a partir del periodo $t = s + 1$. Para el caso del método de MHW se tiene las siguientes

ecuaciones:

$$\text{Nivel} \quad : L_t = \phi \frac{Y_t}{S_{t-s}} + (1-\phi)(L_{t-1} + b_{t-1}) \tag{2.19}$$

$$\text{Tendencia} \quad : b_t = \psi(L_t - L_{t-1}) + (1-\psi)b_{t-1} \tag{2.20}$$

$$\text{Estacionalidad} \quad : S_t = \omega \frac{Y_t}{L_t} + (1-\omega)S_{t-s} \tag{2.21}$$

$$\text{Pronóstico} \quad : \hat{Y}_{t+m} = (L_t + b_t m)S_{t-s+m} \tag{2.22}$$

Para el caso del método de AHW se tiene las siguientes ecuaciones:

$$\text{Nivel} \quad : L_t = \phi(Y_t - S_{t-s}) + (1-\phi)(L_{t-1} + b_{t-1}) \tag{2.23}$$

$$\text{Tendencia} \quad : b_t = \psi(L_t - L_{t-1}) + (1-\psi)b_{t-1} \tag{2.24}$$

$$\text{Estacionalidad} \quad : S_t = \omega(Y_t - L_t) + (1-\omega)S_{t-s} \tag{2.25}$$

$$\text{Pronóstico} \quad : \hat{Y}_{t+m} = L_t + b_t m + S_{t-s+m} \tag{2.26}$$

donde L_t es el valor suavizado para el nivel de la serie en el periodo t. ϕ constante de suavización exponencial para el nivel, Y_t valor real de la serie de tiempo en el periodo t, b_{t-1} componente de tendencia de la serie de tiempo para el periodo t, ψ constante de suavización exponencial para la tendencia, S_t componente estacional de la serie de tiempo para el periodo t, ω constante de suavización exponencial para la estacionalidad, s es la longitud de tiempo de estacionalidad, m son los periodos futuros a predecir y $\hat{Y}_{t+m}$ es el pronóstico para el periodo $t+m$.

2.3. Conclusiones

En este capítulo se presentan los conceptos básicos para conocer sobre las series de tiempo y cómo pueden ser utilizadas para el pronóstico. Se muestran los distintos patrones de series de tiempo, sus tipos de representación (univariable y multivariable), la importancia de la confiabilidad de los datos y la clasificación de la disponibilidad de los datos (series de tiempo discretas y continuas).

Se exponen conceptos básicos para comprender el uso de los métodos de pronóstico, como lo es: la categoría de los métodos (cualitativos y cuantitativos), el conjunto de datos que se usan para su aplicación (TW y VW) y la métrica para medir cuantitativamente la eficiencia.

Se presentan las características, formulación y algoritmos de cinco métodos de pronóstico para series de tiempo. Además de conceptos (modelo, pronóstico, *periodos* y $frecuencia$) que son necesarios para el desarrollo de cada diagrama de flujo que se presenta en el Anexo

A. Siendo así, se considera un conjunto de métodos de pronóstico cuantitativos que se basan en la aplicación de series de tiempo, donde estas series representan una sola variable y son capturadas de forma equidistante en el tiempo.

Capítulo 3

Cómputo en Paralelo Basado en Hilos

3.1. Introducción

Desde hace tiempo se ha aprovechado el uso de las computadoras porque permiten realizar tareas más fácilmente y en menor tiempo. Estas son ejecutadas gracias al procesador que es el cerebro del sistema, siendo de gran importancia la rapidez que este disponga para ejecutar cada acción. Conforme pasa el tiempo, se ha trabajado en el aumento de la velocidad de los procesadores, disminuyendo el tamaño de los transistores. De esta manera, se puede aumentar la velocidad de los transistores, pero ocasiona un aumento en su consumo de energía, provocando que el circuito integrado se caliente y deje de ser seguro. Siendo así, en vez de construir procesadores monolíticos cada vez más rápidos y complejos, la industria ha decidido colocar múltiples procesadores completos, relativamente simples, solamente en un chip para ejecutar las tareas en el menor tiempo posible [Pacheco, 2011].

Es importante considerar que un programa tradicional es secuencial. Este al ser compilado y ejecutado en una computadora multiprocesador no aprovechará las ventajas que dispone el sistema. Siendo relevante la consideración de plataformas para comunicar la repartición de tareas en los múltiples procesadores.

El *procesamiento en paralelo* o también conocido como *cómputo en paralelo*, es una herramienta que puede dividir grandes procesos computacionales en pequeñas tareas aplicándose de manera simultánea. El resultado esperado es la ejecución de programas de manera más rápida en comparación con su ejecución de forma secuencial. Una herramienta para el cómputo en paralelo es a través de hilos (threads), donde el lenguaje de programación C dispone de una librería llamada POSIX Threads (Pthread) que aprovecha de

los múltiples procesadores que dispone la máquina. POSIX threads puede usarse en sistemas operativos similares a UNIX, por ejemplo, LINUX y Mac OS X, solaris, HPUX, etc [Pacheco, 2011]. En la Figura 3.1 se muestra un diagrama de la aplicación de los hilos de manera simultánea. En el Apéndice B se presentan los comandos principales de la librería Pthread que se tomaron en cuenta para el desarrollo de este trabajo de tesis.

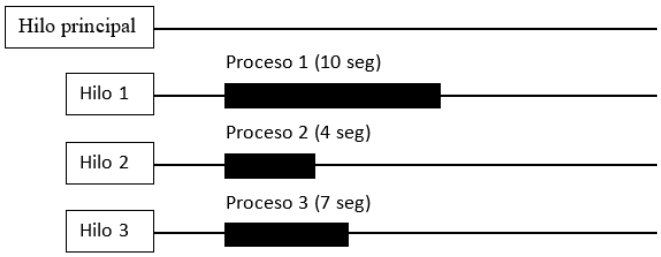

Figura 3.1: Ejemplo del cómputo en paralelo basado en hilos.

Este capítulo presenta conceptos básicos de las distintas arquitecturas, diseño de programación y granularidad del cómputo en paralelo. Además de mostrar conceptos para entender el uso de los hilos y sus transiciones de estados. Por último, se presenta la métrica de velocidad para el empleo de multiprocesadores.

3.2. Arquitecturas para el cómputo en paralelo

En el cómputo paralelo existe una diversidad de arquitecturas que pueden ser utilizadas. Siendo la *taxonomía de Flynn* la que clasifica la arquitectura para el cómputo en paralelo, basándose en la organización del flujo de instrucciones y de datos simultáneos que son tratados por el sistema durante la ejecución del programa. Esta clasificación se divide en SISD, SIMD, MISD y MIMD [Aguilar et al., 2004].

La clasificación SISD por sus siglas en inglés "single instruction - single data", se caracteriza por poseer un único procesador. Esta clasificación no contiene paralelismo, siendo un sistema monoprocesador. Representa a la mayoría de las computadoras secuenciales [Aguilar et al., 2004]. La clasificación SIMD por sus siglas en inglés "simple instruction - multiple data", se caracteriza por contener una única entrada de instrucciones sobre múltiples flujos de datos. Se emplea en procesadores de matrices y en procesadores vectoriales. Es muy usada en el procesamiento de multimedia [Aguilar et al., 2004]. La clasificación MISD por sus siglas en inglés "multiple instruction - single data", se caracteriza por poseer múltiples flujos de instrucciones. Muchos autores consideran que esta clase no corresponde

a un modo de funcionamiento realista [Aguilar et al., 2004]. La clasificación *MIMD* por sus siglas en inglés "multiple instruction - multiple data", se caracteriza por emplear varios tipos de control, teniendo instrucciones y datos corrientes e independientes. Dispone de multiprocesadores en paralelo que pueden ejecutar su propia secuencia de instrucciones. Los procesadores pueden trabajar en una parte distinta del problema y se caracterizan por la habilidad de comunicar datos a otros procesadores. La Taxonomía de Flynn no distingue la organización de la memoria interna de los computadores paralelos, dejando por fuera alguna clase de máquinas. Para el caso de la organización de la memoria interna existen dos tipos de organización de memoria: *Memoria compartida* (SM, del inglés *Shared Memory*) y *memoria distribuida* (MD, inglés *Memory Distributed*) [Aguilar et al., 2004].
En los sistemas SM la memoria se comparte a todos los procesadores del equipo [Pacheco, 2011]. Los sistemas de MD son multicomputadores conectados a una misma red de interconexión, donde todos los procesos son ejecutados en procesadores independientes y la información es enviada y recibida por medio de mensajes.

3.3. Metodología de diseño de algoritmos en paralelo

En la programación paralela, una de las figuras más importantes fue *Ian Foster*, donde su libro [Foster, 1995] propone un modelo para el diseño de la programación paralela. Este se divide en cuatro etapas: partición, comunicación, aglomeración y mapeo. En la Figura 3.2 se muestra un diagrama de la metodología de Ian Foster.

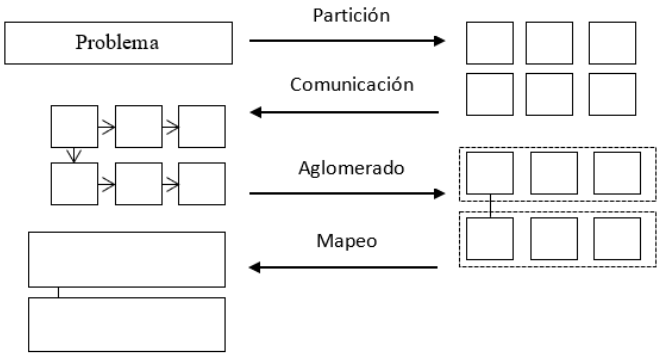

Figura 3.2: Metodología de Ian Foster.

La etapa de la *partición* está enfocada a descomponer el algoritmo en varias tareas. Siendo así, se ignoran aspectos como el número de procesadores, organización de la memoria, etc. Al realizarse la partición se especifica la mejor manera de dividir el problema y encontrar las "oportunidades del paralelismo". En la segunda etapa, trata de identificar

la *comunicación* entre los procesadores para coordinar la ejecución de tareas, buscando la relación y las dependencias de datos. En la etapa de *aglomeración* (proceso inverso de la partición), trata de juntar las tareas que se puedan ejecutar de manera simultánea sin afectar al sistema en general. Teniendo definidas las dependencias de datos, la comunicación y el orden de las tareas, se aplica la aglomeración. En la última etapa, que es el *mapeo*, asigna las tareas a cada procesador. Es decir, ya que se tiene el conjunto de bloques aglomerados, el mapeo asigna a cada hilo la tarea que debe ejecutar [Foster, 1995].

3.4. Granularidad del paralelismo

La *granularidad del paralelismo* es el tamaño promedio de las tareas que se ejecutan con referencia al número de instrucciones, número de palabras de memorias usadas y la duración del tiempo de ejecución. Siendo la cantidad de procesamiento que una tarea hace antes de necesitar comunicarse con otra tarea. Teniendo una relación entre la computación y la comunicación [Aguilar et al., 2004].

La granularidad entre más pequeña sea implica más comunicación y cambios de contexto entre las tareas; si la granularidad es grande, implica menos comunicación, pero puede que potenciales tareas concurrentes queden agrupadas y por consiguiente ejecutadas secuencialmente. En la Tabla 3.1 se presenta los distintos tipos de granularidad con referencia al tipo de código, modo paralelo y arquitectura del equipo [Aguilar et al., 2004].

Tabla 3.1: Granularidad en el cómputo paralelo.

Tipo de código	**Granularidad**	**Modo Paralelo**	**Arquitectura**
Programas	Grueso	Concurrente	MIMD
Rutinas	Mediano-Fuerte	Concurrente	MIMD
Instrucciones	Mediano-Fino	Concurrente-Paralelo	SIMD
Operadores y Bits	Fino	Paralelo	SIMD y Hadware

3.5. Plataforma Pthreads

Un *hilo* es el proceso que representa una tarea aplicada simultáneamente dentro de una máquina multiprocesador, como el mostrado en la Figura 3.3. Es decir, mediante múltiples procesadores se puede disponer de diversos hilos compartiendo un espacio de memoria para la ejecución de distintas tareas, donde cada hilo de ejecución representa el trabajo de un procesador [Butenhof, 1997].

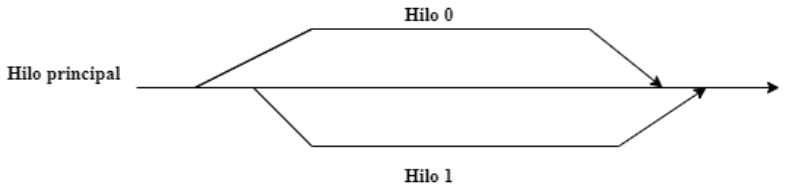

Figura 3.3: Diagrama de la ejecución de los hilos.

Para el uso de hilos, es necesario conocer el algoritmo y sus partes que son *asíncronas*. Los algoritmos asíncronos son aquellos que suceden de manera independiente a menos que exista alguna dependencia impuesta. En pocas palabras, son operaciones cualesquiera que pueden proceder independientemente unas de las otras. Entonces, las secciones de código que son asíncronas, son las tareas que se envían a los hilos [Butenhof, 1997].

Es importante considerar que los hilos requieren proteger sus datos debido a que trabajan en una SM y requiere comunicar datos de manera independiente. Siendo así, la *sincronización* es la manera en que todos los hilos cooperan para ejecutar una tarea. Con base en la biblioteca pthreads, el objetivo principal de la sincronización es proporcionar un mecanismo para evitar que colisionan inestablemente. La utilización del mutex en la biblioteca pthreads funciona como "seguro", ya que este evita que las tareas de cada hilo ocurran al mismo tiempo. Si no se utiliza la sincronización en la comunicación de los datos, daría como resultado el caos y no la comunicación [Butenhof, 1997]. La Figura 3.4 muestra un diagrama del funcionamiento del mutex en pthread.

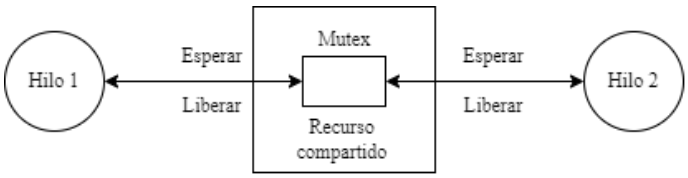

Figura 3.4: Diagrama del mutex.

3.5.1. Transiciones de estado de los hilos

Los hilos se encuentran en distintos estados conforme avanza la ejecución del programa. Con la finalidad de comprender la programación de los hilos, se generaliza a cuatro estados principales: listo, corriendo, obstruido y terminado. En la Tabla 3.2 se muestra la descripción de cada estado y en la Figura 3.5 se muestra esquemáticamente los estados.

Tabla 3.2: Estados de los hilos. Fuente: [Butenhof, 1997].

Estado	Significado
Listo	El hilo puede ejecutarse, pero está esperando un procesador. Es posible que haya comenzado recientemente, que se haya desbloqueado o que otro hilo lo haya reemplazado.
Corriendo	El hilo se está ejecutando actualmente; en un multiprocesador puede haber más de un hilo en ejecución en el proceso.
Obstruido	El hilo no puede ejecutarse porque está esperando algo; por ejemplo, puede estar esperando bloquear un mutex.
Terminado	El hilo ha terminado al regresar de su función de inicio.

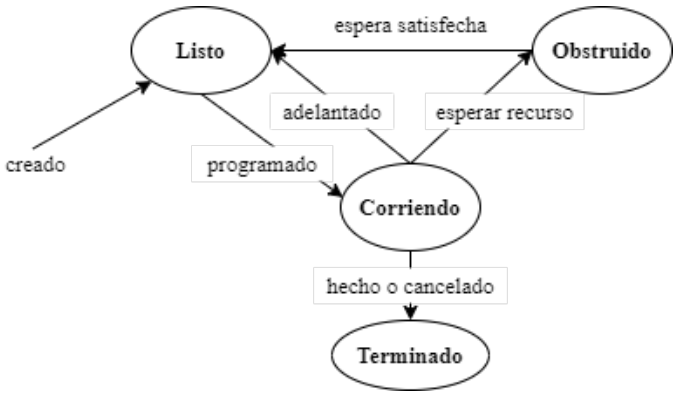

Figura 3.5: Transiciones de estado de los hilos. Fuente:[Butenhof, 1997].

3.6. Métrica para la velocidad de procesamiento

En términos ideales, el rendimiento de los procesadores va en aumento linealmente conforme crece el número de procesadores. Por ejemplo, si se usan dos procesadores en un sistema, el rendimiento debería duplicarse. Prácticamente, esto no sucede, ya que a partir de una cantidad de procesadores se provoca un efecto de saturación, ocasionando una degradación en el rendimiento del sistema [Aguilar et al., 2004].

En términos de arquitectura de computadoras, existe la aceleración del rendimiento, que es la relación entre el tiempo de ejecución de un procesador secuencial y el tiempo de ejecución de múltiples procesadores [Aguilar et al., 2004]. El objetivo de la aceleración es medir de manera cuantitativa el tiempo que tarda en ejecutarse los programas utilizando múltiples procesadores.

Matemáticamente, la aceleración puede ser definida como:

$$Speed\ up = \frac{T_s}{T_p} \tag{3.1}$$

donde T_s es el tiempo secuencial conocido y T_p es el tiempo ejecutando P procesadores.

3.7. Conclusiones

En este capítulo se muestran conceptos básicos para comprender las ventajas que tiene el cómputo paralelo y su funcionamiento en general. Presentando características de la arquitectura SISD, SIMD, MISD y MIMD (ya sea en SM o MD). Se muestra una metodología del diseño de programación de Ian Foster para identificar las distintas etapas que requieren los programas para dividirse en la mayor cantidad de tareas. Además, mediante la granularidad, se conoce qué arquitectura es la adecuada para realizar el cómputo en paralelo, ya que su objetivo es reducir el tiempo de ejecución de los programas. También, se presenta una sección dedicada a los hilos, donde su introducción describe el concepto de hilos, asincronía y la importancia de la sincronización. Además, la sección explica los cuatro estados generales de los hilos al momento de su ejecución. Finalmente, uno de los objetivos de este trabajo de tesis es medir el tiempo de ejecución usando distintos procesadores, se presenta la métrica para medir el tiempo de paralelización por medio del *Speed up*.

En este trabajo de tesis, se considera una arquitectura MIMD-SM, y por medio de la librería pthread, se vinculan los programas paralelos a los procesadores. La repartición de las tareas asíncronas se hará considerando una granularidad mediano-fuerte por la necesidad de paralelizar funciones.

Capítulo 4

Propuesta de Paralelización de los Métodos de Pronóstico de Series de tiempo

4.1. Introducción

En el Capítulo 2 se dio una descripción básica de los algoritmos de los métodos de pronóstico de series de tiempo de MA, WMA, ES y HW (tanto su parte multiplicativa como aditiva). Estos algoritmos presentan una característica en común. Seleccionando un parámetro, realiza el modelo, pronóstico y evaluación del método de pronóstico. Aplicar el pronóstico de esta manera ocasiona cierta incertidumbre al usuario, ya que, necesita del conocimiento y experiencia del método de pronóstico para establecer el parámetro que presente menor error en el pronóstico. Esto se debe por la cantidad de parámetros que pueden aplicar los métodos de pronóstico de series de tiempo. Por ello, se propone la búsqueda exhaustiva de los parámetros en cada método de pronóstico de series de tiempo para identificar cuál parámetro es el que presenta menor error en el pronóstico. Esto ocasiona un esfuerzo computacional alto, reflejándose directamente en el tiempo que tarda en ejecutarse el programa. Para su mejora, se sugiere el cómputo en paralelo usando múltiples elementos de procesamiento y, a través de la plataforma POSIX Threads, hacer el código de programación que permite la paralelización.

Con base a la metodología del diseño de la programación paralela de Ian Foster, la partición se aplicó en los parámetros, ya que la ejecución de cada método de pronóstico es independiente con el uso de cada parámetro. La distribución equivalente de la cantidad de parámetros en los elementos de procesamiento está asociada con la aglomeración y,

finalmente, el mapeo se encarga de distribuir los valores de los parámetros en cada elemento de procesamiento. Por el diseño que se propone de la programación paralela de Ian Foster, se consideró una granularidad mediano-fuerte, ya que se aplica el método de pronóstico de series de tiempo a cada parámetro que dispone la búsqueda exhaustiva.

Este capítulo presenta la descripción de los algoritmos que realizan la selección automática del parámetro óptimo en los métodos de pronóstico de series de tiempo de MA, WMA, ES, LS y HW por medio del cómputo en paralelo basado en hilos para su reducción del tiempo computacional. El código en C que ejecuta la propuesta de paralelización de los métodos de pronóstico de series de tiempo se encuentra en el Apéndice C - Apéndice H.

4.2. Propuesta de paralelización

En la propuesta de paralelización, se consideraron secciones semejantes para el desarrollo de los algoritmos, con el fin de tener mayor facilidad en su programación. Siendo así, se tiene cinco secciones principales: datos iniciales, lectura de serie de tiempo, distribución de datos de TW y VW, división de tareas y ficheros de salida. Mostrándose en la Figura 4.1 un diagrama de flujo generalizado de los algoritmos que realizan la selección automática del parámetro óptimo de los métodos de pronóstico de series de tiempo de MA, WMA, ES, LS, y HW, por medio del cómputo en paralelo basado en hilos.

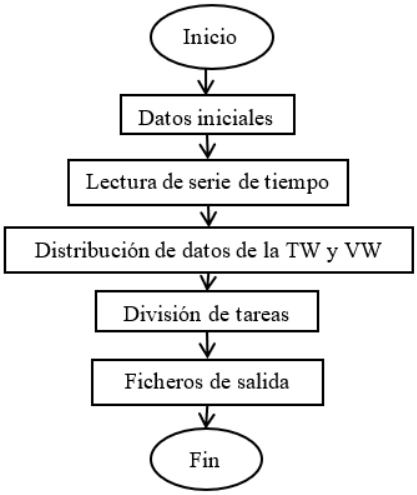

Figura 4.1: Diagrama de flujo generalizado de los algoritmos que realizan la selección automática del parámetro óptimo de los métodos de pronóstico de series de tiempo de MA, WMA, ES, LS, y HW, por medio del cómputo en paralelo basado en hilos.

La parte que representa los datos iniciales trata de la descripción de la sintaxis que lleva el ingreso de datos dentro de la terminal de Linux para ejecutar cada algoritmo

paralelo. La sección correspondiente a la lectura de la serie de tiempo, toma como base los datos iniciales para leer la serie de tiempo que se encuentra contenida en un fichero, para después almacenar los datos dentro de un arreglo y poder manipular la información en el programa. En el fragmento de código que representa la distribución de datos de la TW y VW, se encarga de repartir los datos del arreglo definido en la sección de lectura de serie de tiempo para formar la TW y VW. La parte asociada a la división de tareas, se encarga de repartir el trabajo que le corresponde a cada hilo. Debido a que cada método de pronóstico dispone de distintos parámetros, esta sección tiene variaciones y condiciones distintas en cada método de pronóstico de series de tiempo. Finalmente, se tiene la sección de los ficheros de salida, esta arroja tres archivos asociados a la serie de tiempo, parámetros y pronóstico del método utilizado.

A continuación, se describe con más detalle las secciones que conforman la Figura 4.1, mostrando las similitudes y diferencias entre los algoritmos paralelos propuestos.

4.2.1. Datos iniciales

El primer paso de la Figura 4.1 es definir los valores iniciales, y estos se ingresan dentro de la terminal de Linux. Los valores iniciales son necesarios para que el programa en C permite leer la serie de tiempo que está contenida en un fichero y ayudar al desarrollo del algoritmo paralelo. Es importante mencionar que antes de establecer estos valores, se tuvieron que realizar observaciones de la serie de tiempo para conocer la escala de tiempo en que se necesita realizar el pronóstico. Hecho esto, se tendrá en consideración la $frecuencia$ y la cantidad de *periodos* para poder hacer el pronóstico de la serie de tiempo.

Los valores iniciales constan de cinco argumentos (a excepción del método de MA que consta de cuatro argumentos). A continuación, se muestra la sintaxis de los datos iniciales establecidos en la terminal de Linux

```
./[ejecutable][fichero.txt][frecuencia][periodos][saltos][procesadores]
```

donde

- el primer argumento es el nombre del ejecutable, siendo las siglas que se definió en cada método de pronóstico de series de tiempo;
- el segundo argumento es el nombre del fichero con una extensión ".txt" donde está contenida la serie de tiempo;
- el tercer argumento es la $frecuencia$ que se escogió para realizar el pronóstico de la serie de tiempo;

- el cuarto argumento se trata de la cantidad de *periodos* que se escogió para realizar el pronóstico de la serie de tiempo;

- el quinto argumento son los saltos que se requieren para aplicar la búsqueda exhaustiva para la selección del parámetro óptimo del método de pronóstico de series de tiempo de WMA, ES, MHW y AHW. En el caso del método de pronóstico de series de tiempo de LS, este argumento se trata del orden máximo del polinomio de la función (ρ_{max}). Para el caso del método de pronóstico de series de tiempo de MA no ingresa este valor inicial;

- el quinto argumento es la cantidad de procesadores a utilizar.

4.2.2. Lectura de serie de tiempo

El segundo paso de la Figura 4.1, es la lectura de la serie de tiempo. Para todos los métodos de pronóstico, se requiere que el formato de la serie de tiempo sea un vector columna dentro de un fichero, donde sus datos son los capturados secuencialmente en la serie de tiempo. Se realiza de esta manera debido a que es un formato estándar para la aplicación de los métodos de pronóstico de series de tiempo y para la búsqueda de cualquier serie de tiempo. Para todos los métodos de pronóstico, se tiene el mismo código de programación para la lectura de la serie de tiempo.

4.2.3. Distribución de datos de la ventana de entrenamiento y validación

El tercer paso de la Figura 4.1, es la distribución de la TW y VW. Para los métodos de pronóstico de series de tiempo de MA, WMA, ES y LS el algoritmo paralelo procede a tomar la serie de tiempo establecida en la Sección 4.2.2 y con base a los datos iniciales, forma una matriz de dimensiones ($frecuencia$ x No. *periodos*-1) para la TW, siendo los datos que corresponden desde el primer hasta el penúltimo periodo de la serie de tiempo. Para el caso de la VW, se define un vector con una cantidad de datos igual a la $frecuencia$, siendo los datos del último periodo de la serie de tiempo.

Para el caso de los métodos de HW es diferente, ya que este no distribuye sus datos de la TW de la misma manera. Esto se debe al algoritmo que dispone el método de pronóstico [Wheelwright et al., 1998]. En este caso, se trabaja con un vector en vez de una matriz.

4.2.4. División de tareas

La división de tareas corresponde a la cuarta parte de la Figura 4.1. Para el algoritmo paralelo de todos los métodos de pronóstico de series de tiempo utilizados en este trabajo de tesis, el mapeo se realiza al distribuir la cantidad de parámetros disponibles (n_{params}) en los elementos de procesamiento (P). Se considera de esta manera para repartir equivalentemente las tareas a cada hilo, requiriendo analizar el rango de inicio-final para aplicar los métodos de pronóstico de series de tiempo con base a los parámetros asignados.

Para asignar la cantidad de parámetros que ejecutará cada hilo se realiza mediante:

$$\sigma = \frac{n_{param}}{P} \tag{4.1}$$

donde n_{param} es la cantidad de parámetros totales y P es el número de elementos de procesamiento. De esta manera, el algoritmo establece un rango de parámetros basándose en el número de elementos de procesamiento.

Otra consideración para la distribución de datos, es analizar los valores y cantidad de parámetros que puede aplicar el algoritmo paralelo de cada método de pronóstico de series de tiempo. A continuación, se presenta una descripción de los valores que disponen los parámetros en los métodos de pronóstico de MA, WMA, ES, LS y HW. Además, se presenta su diagrama de flujo para la división de tareas.

4.2.4.1. Promedios móviles

En el método de pronóstico de series de tiempo de MA, se considera el parámetro, α, el cual define el número de términos a promediar de (2.3). Este puede variar desde $\alpha = 1,\ 2, \ldots, No.periodos - 2$. Para la división de tareas, se toman los valores de α y se distribuyen en grupos equivalentes a cada hilo. En la Figura 4.2 se muestra el diagrama de flujo que hace la división de parámetros del método de pronóstico de series de tiempo de MA aplicando el cómputo en paralelo basado en hilos y en la Tabla 4.1 se presenta la descripción del algoritmo.

Tabla 4.1: Algoritmo del cómputo en paralelo basado en hilos aplicando la distribución de parámetros del método de pronóstico de series de tiempo de MA.

1. Inicio.
2. Asignar la cantidad de parámetros que ejecutará cada hilo.
3. Mapeo a cada elemento del proceso.
4. Se asignan la cantidad de parámetros a ejecutar.
5. Se forma el modelo con base a los datos de la TW.
6. Realiza el pronóstico con el parámetro asignado.
7. Realiza el MAPE de la posición i
8. Realiza el MAPE promedio del parámetro asignado.
9. Selecciona el MAPE mínimo entre todos los parámetros usados.
10. Con base al MAPE mínimo seleccionado, se selecciona el parámetro óptimo y se escoge el pronóstico realizado con ese parámetro.
11. Fin

4.2.4.2. Promedios móviles ponderados

En el método de pronóstico de series de tiempo de WMA, dispone del parámetro, β, y este se encuentra asociado al valor de los datos de la serie de tiempo (n_{wma}). El peso β se aplica a los n_{wma} datos más recientes, donde su valor puede variar de $n_{wma} = 1$, . . . , no. *Periodos* - 2. Siendo así, la cantidad de pesos depende del valor de n_{wma} ($\beta_1, \beta_2, \ldots, \beta_{n_{wma}}$). Los pesos aplicados son distintos entre sí, ya que cumplen con dos condiciones: la suma entre ellos debe ser igual al 100 % y se aplica un peso mayor a los datos más recientes. Para su división de tareas, se usan ”combinaciones´´ y se reparten a los elementos de procesamiento gracias a (4.1).

El código en C de la selección automatica del parámetro óptimo por medio del cómputo en paralelo basado en hilos para el método de pronóstico de WMA, se consideran combinaciones de β con un valor de $n_{wma} <= 5$, ya que, el algoritmo difícilmente selecciona el parámetro óptimo con una cantidad mayor de combinaciones. En la Figura 4.3 se muestra el diagrama de flujo que realiza la división de parámetros del método de pronóstico de series de tiempo de WMA aplicando el cómputo en paralelo basado en hilos y en la Tabla 4.2 se presenta la descripción del algoritmo.

Tabla 4.2: Algoritmo del cómputo en paralelo basado en hilos aplicando la distribución de parámetros del método de pronóstico de series de tiempo de WMA.

1. Inicio.
2. Declara el valor de n_{wma}.
3. Se definen los pesos β con base a los n_{wma} términos.
4. Verifica que la suma de los pesos sea del 100 %
5. Verifica que los términos para los recientes son mayores
6. Asignar la cantidad de parámetros que ejecutará cada hilo.
7. Mapeo a cada elemento del proceso.
8. Se asignan la cantidad de parámetros a ejecutar.
9. Se forma el modelo con base a los datos de la TW.
10. Realiza el pronóstico con el parámetro asignado .
11. Realiza el MAPE de la posición i
12. Realiza el MAPE promedio del parámetro asignado.
13. Selecciona el MAPE mínimo entre todos los parámetros usados.
14. Con base al MAPE mínimo seleccionado, se selecciona el parámetro óptimo y se escoge el pronóstico realizado con ese parámetro.
15. Fin

4.2.4.3. Suavización exponencial

En el método de pronóstico de series de tiempo de ES, su parámetro, γ, es la constante de suavizamiento de (2.7) y puede variar desde $0 < \gamma < 1$. La variación de γ depende de los saltos establecidos en los datos iniciales. Para la distribución de tareas, todos los valores que puede tener α, se distribuyen en los elementos de procesamiento. En la Figura 4.4 se presenta el diagrama de flujo que realiza distribución de parámetros del método de pronóstico de series de tiempo de ES usando el cómputo en paralelo basado en hilos y en la Tabla 4.3 se presenta la descripción del algoritmo.

Tabla 4.3: Algoritmo del cómputo en paralelo basado en hilos aplicando la distribución de parámetros del método de pronóstico de series de tiempo de ES.

1. Inicio.
2. Asignar la cantidad de parámetros que ejecutará cada hilo.
3. Mapeo a cada elemento del proceso.
4. Se asignan la cantidad de parámetros a ejecutar.
5. Se forma el modelo con base a los datos de la TW.
6. Realiza el pronóstico con el parámetro asignado.
7. Realiza el MAPE de la posición i
8. Realiza el MAPE promedio del parámetro asignado.
9. Selecciona el MAPE mínimo entre todos los parámetros usados.
10. Con base al MAPE mínimo seleccionado, se selecciona el parámetro óptimo y se escoge el pronóstico realizado con ese parámetro.
11. Fin

4.2.4.4. Mínimos cuadrados

En el método de pronóstico de LS, su parámetro, ρ, define el orden de la función polinomial de (2.10), donde sus variaciones van desde $\rho = 1,\ 2, \ldots, No.Periodos - 1$. El código en C del cómputo en paralelo basado en hilos para la selección automática del parámetro óptimo del método de pronóstico de series de tiempo de LS, el orden máximo, ρ_{max}, se establece en los valores iniciales del algoritmo. Siendo así, el orden de la función polinomial tendrá sus variaciones desde $\rho = 1, 2, \ldots, \rho_{max}$.

La división de tareas se realiza con base al parámetro ρ_{max} y la cantidad de elementos de procesamiento. Por ejemplo, si se tiene $\rho_{max} = 5$ y se dispone de cinco elementos de procesamiento, entonces se ejecutaría un parámetro, ρ, por cada hilo. En la Figura 4.5, se presenta el diagrama de flujo que hace distribución de parámetros del método de pronóstico de series de tiempo de LS usando el cómputo en paralelo basado en hilos y en la Tabla 4.4 se presenta la descripción del algoritmo.

Tabla 4.4: Algoritmo del cómputo en paralelo basado en hilos aplicando la distribución de parámetros del método de pronóstico de series de tiempo de LS.

1. Inicio.
2. Asigna la cantidad de parámetros que ejecutará cada hilo.
3. Mapeo a cada elemento del proceso.
4. Se forma la matriz A.
5. Invierte la matriz A.
6. Se forma el vector b.
7. Se forma el vector d.
8. Se forma el modelo con base a los datos de la TW.
9. Realiza el pronóstico con el parámetro asignado.
10. Realiza el MAPE de la posición i.
11. Realiza el MAPE promedio del parámetro asignado.
12. Selecciona el MAPE mínimo entre todos los parámetros usados.
13. Con base al MAPE mínimo seleccionado, se selecciona el parámetro óptimo y se escoge el pronóstico realizado con ese parámetro.
14. Fin

4.2.4.5. Holt-Winters

Finalmente, se tiene los métodos de pronóstico de series de tiempo de HW, el cual se conforma de dos tipos: MHW y AHW. Ambos tienen el mismo desarrollo en su diagrama de flujo, entonces esta sección describe el algoritmo como el método de HW sin necesidad de explicar particularmente el método de pronóstico de series de tiempo de MHW o AHW. El método de pronóstico de series de tiempo de HW, está conformado de tres parámetros (ϕ, ψ y ω). Estos, al igual que el método de pronóstico de ES, sus valores pueden variar desde $0 < \phi, \psi, \omega < 1$.

La distribución de tareas depende de los saltos que tenga los parámetros ϕ, ψ, y ω con base a lo establecido en los valores iniciales. Siendo estos valores los que se distribuyen equivalentemente en los elementos de procesamiento que se requieran. En la Figura 4.6 se presenta el diagrama de flujo que realiza distribución de parámetros de los métodos de pronóstico de series de tiempo de HW usando el cómputo en paralelo basado en hilos y en la Tabla 4.5 se presenta la descripción del algoritmo.

Tabla 4.5: Algoritmo del cómputo en paralelo basado en hilos aplicando la distribución de parámetros del método de pronóstico de series de tiempo de HW.

1. Inicio.
2. Asigna la cantidad de parámetros que ejecutará cada hilo.
3. Mapeo a cada elemento del proceso.
4. Calcula los valores iniciales L_s, b_s y S_s.
5. Calcula el elemento t de L_t.
6. Calcula el elemento t de b_t.
7. Calcula el elemento t de S_t.
8. Se forma el modelo en el periodo i.
9. Realiza el pronóstico en el periodo i
10. Realiza el MAPE de la posición i.
11. Realiza el MAPE promedio del parámetro asignado.
12. Selecciona el MAPE mínimo entre todos los parámetros usados.
13. Con base al MAPE mínimo seleccionado, se selecciona el parámetro óptimo y se escoge el pronóstico realizado con ese parámetro.
14. Fin

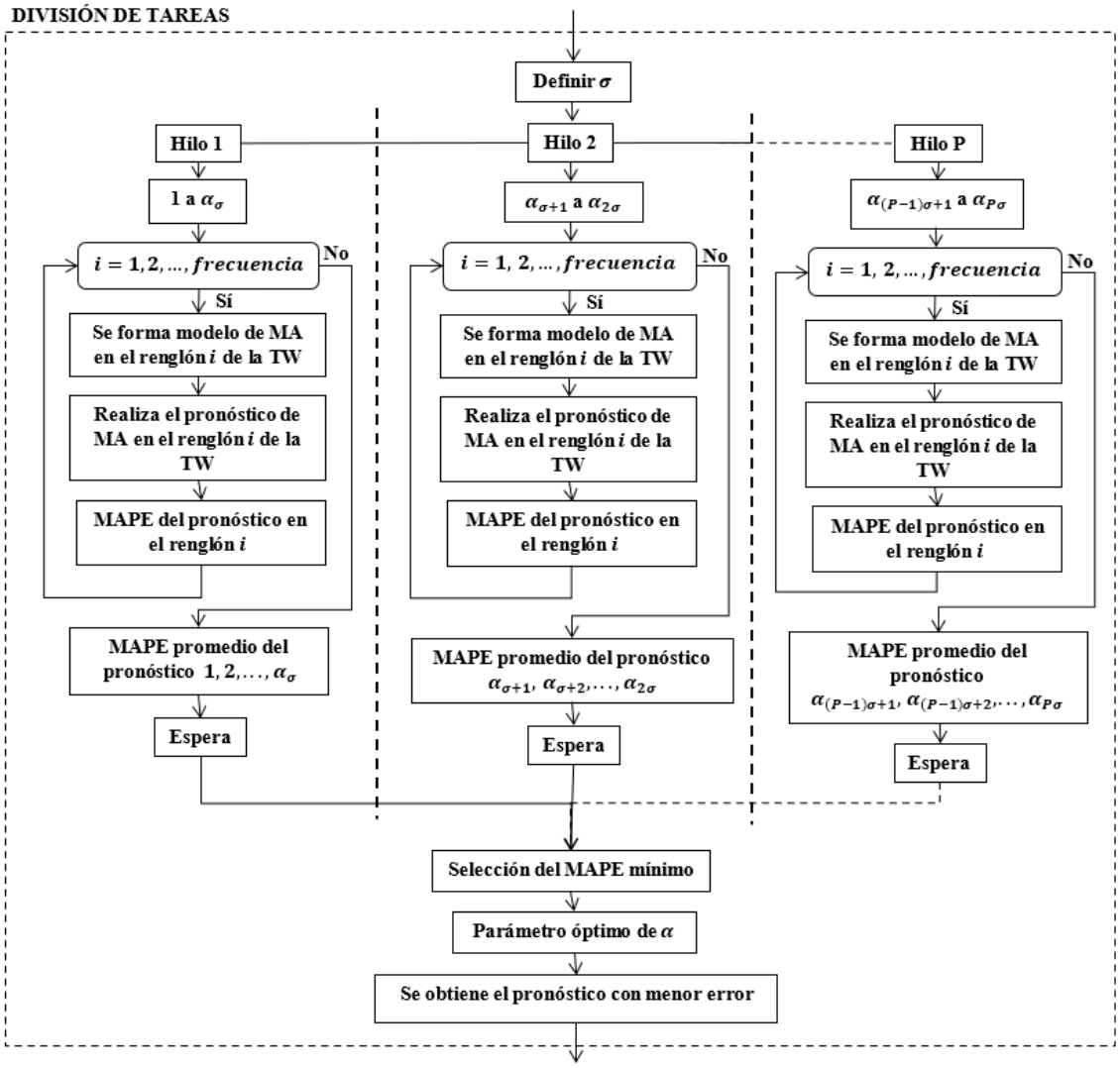

Figura 4.2: Diagrama de flujo para la distribución de parámetros del método de pronóstico de series de tiempo de MA haciendo uso del cómputo en paralelo basado en hilos.

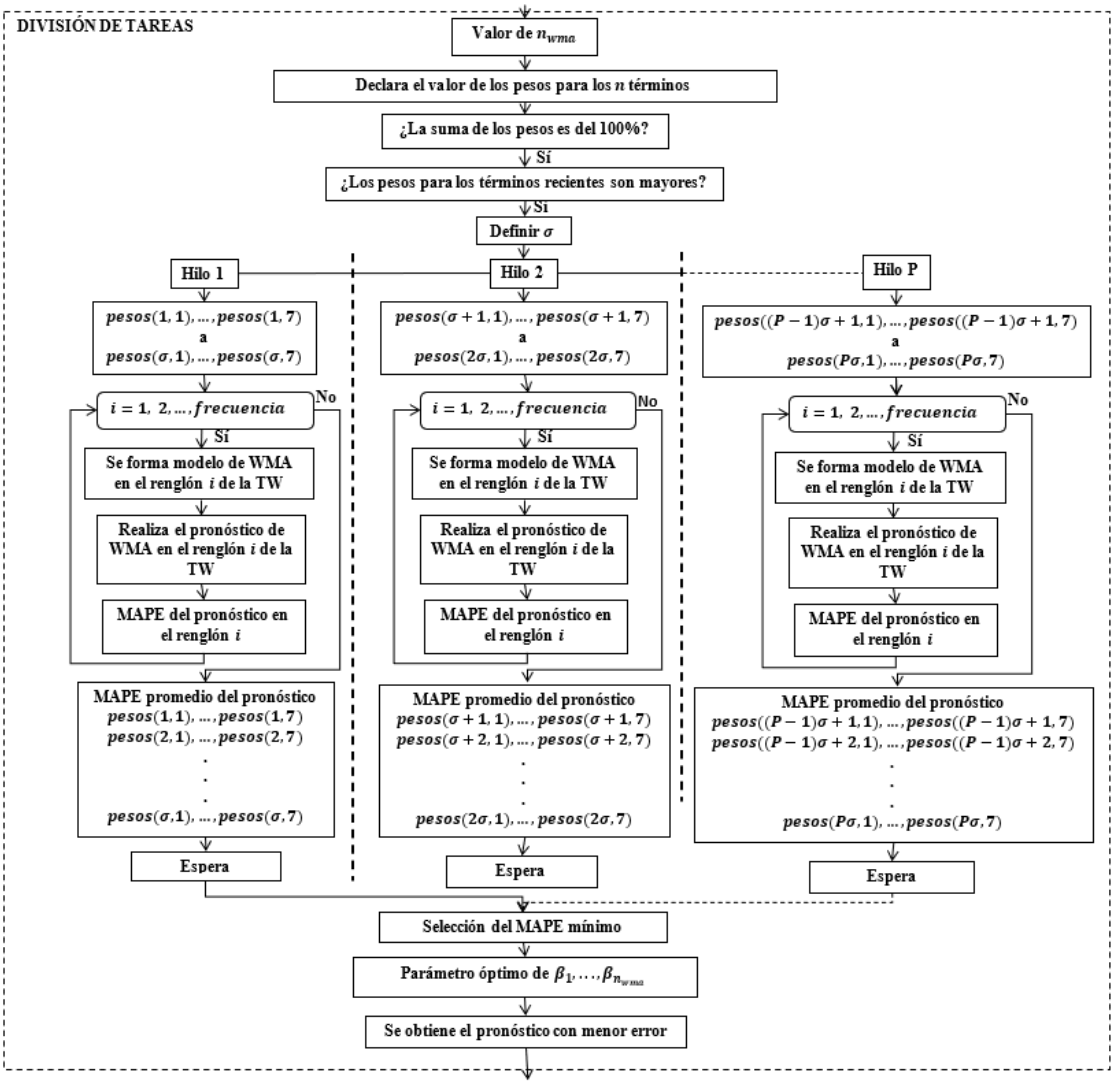

Figura 4.3: Diagrama de flujo para la distribución de parámetros del método de pronóstico de series de tiempo de WMA haciendo uso del cómputo en paralelo basado en hilos.

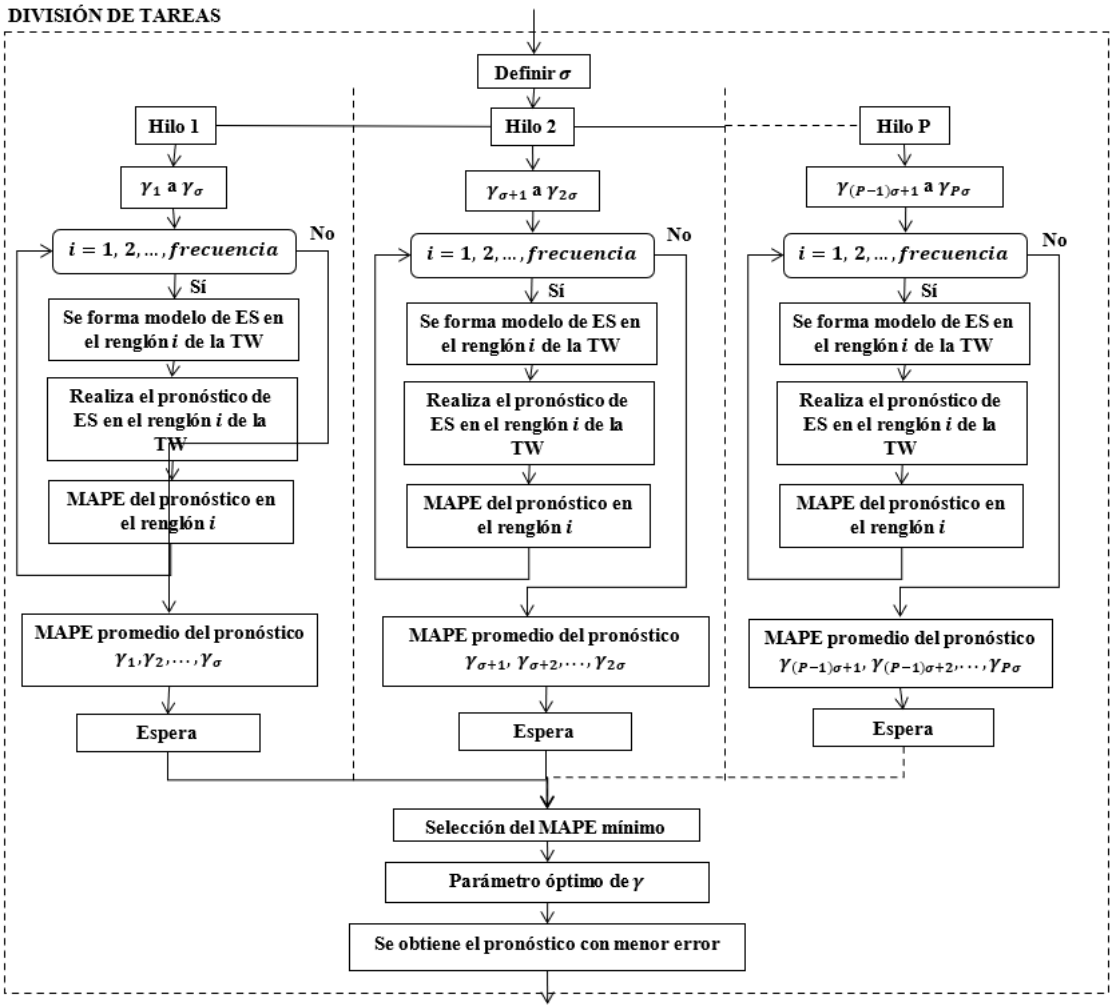

Figura 4.4: Diagrama de flujo para la distribución de parámetros del método de pronóstico de series de tiempo de ES haciendo uso del cómputo en paralelo basado en hilos.

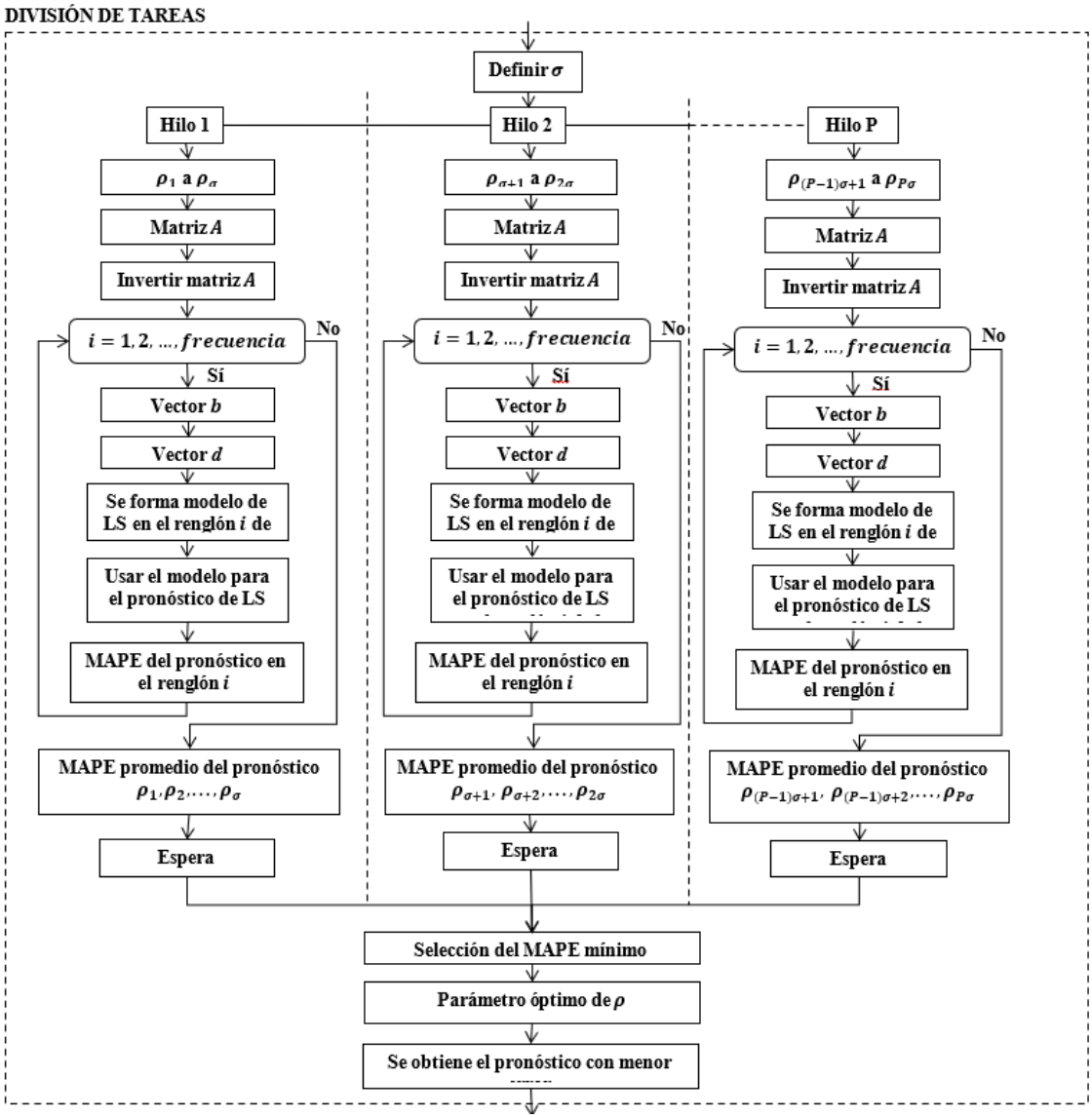

Figura 4.5: Diagrama de flujo para la distribución de parámetros del método de pronóstico de series de tiempo de LS haciendo uso del cómputo en paralelo basado en hilos.

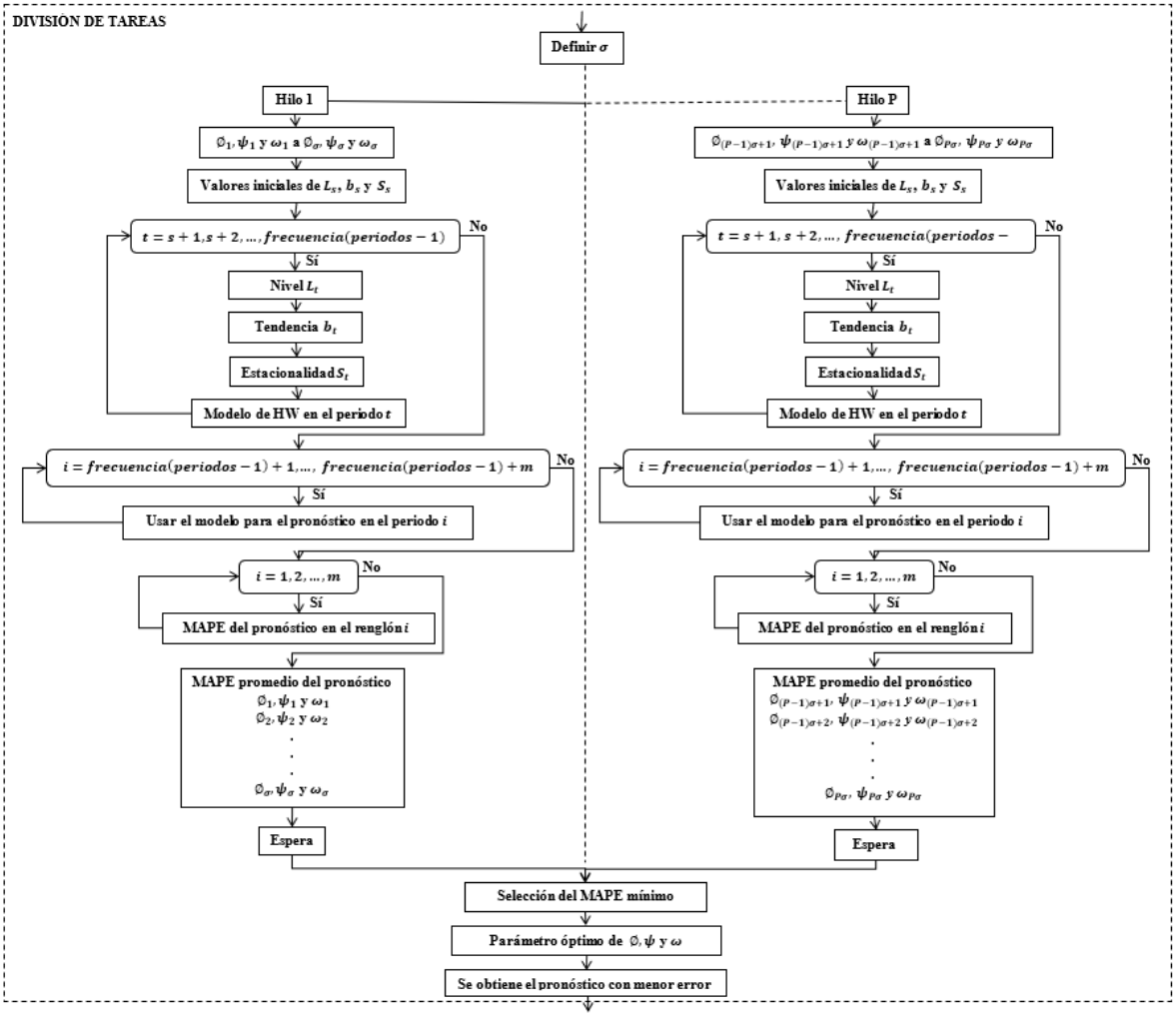

Figura 4.6: Diagrama de flujo para la distribución de parámetros del método de HW haciendo uso del cómputo en paralelo basado en hilos.

4.2.5. Ficheros de salida

El último paso de la Figura 4.1 son los ficheros de salida. En esta sección se genera tres archivos para acceder a la información del algoritmo y método de pronóstico. Debido a que se puede trabajar con grandes cantidades de datos, es necesario tener facilidad para acceder a la información, repartiéndose en archivos asociados a serie de tiempo, parámetros y pronóstico. Dentro de los tres archivos, se muestra información acerca del algoritmo, ya que el usuario debe verificar si todos los datos son los correctos. Básicamente, estos datos son los que se ingresaron en la sección de datos iniciales y los obtenidos en la división de tareas. En la Figura 4.7, se muestra la información que presenta cada fichero del algoritmo.

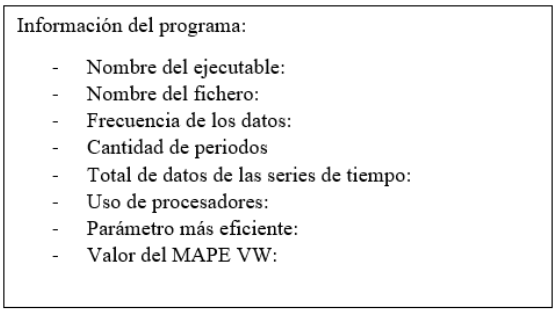

Figura 4.7: Datos iniciales del método de pronóstico de series de tiempo.

El primer archivo que genera el algoritmo se llama "series de tiempo" y muestra la estructura que tiene la serie de tiempo, TW y VW que se utilizó para el desarrollo del algoritmo. Se presenta la serie de tiempo y TW distribuida de forma matricial; y la VW acomodada de forma vectorial. El segundo archivo se llama "parámetros". Se genera este fichero para conocer el comportamiento del pronóstico ante los parámetros usados en la búsqueda exhaustiva, mostrándo los resultados asociados al MAPE de TW y VW. Por último, se tiene el fichero asociado al pronóstico. Lo que muestra es el pronóstico utilizando el parámetro óptimo, siendo el que presenta menor error en el MAPE de la VW.

4.3. Conclusiones

En el capítulo se presenta la propuesta del cómputo en paralelo basado en hilos para la selección automática del parámetro óptimo de los métodos de pronóstico de series de tiempo de MA, WMA, ES, LS y HW. Debido a que el algoritmo paralelo de cada método de pronóstico tiene secciones semejantes, se mostró las similitudes y diferencias entre los métodos de pronóstico de series de tiempo. Se presentó el algoritmo paralelo dividido en cinco secciones: datos iniciales, lectura de serie de tiempo, distribución de datos de TW y VW, división de tareas y ficheros de salida; donde principalmente la sección correspondiente a la división de tareas explica cómo se realiza el cómputo en paralelo basado en hilos a partir de la aplicación del método de pronóstico usando un rango de parámetros por cada hilo. El diseño del algoritmo paralelo se hizo mediante la metodología de Ian Foster, considerando la división de los parámetros para su distribución equivalente en cada elemento de procesamiento, ya que los métodos de pronóstico de series de tiempo pueden trabajar simultáneamente por cada valor de parámetro establecido. Debido a esto, se consideró una granularidad medio-fuerte por el paralelismo de funciones. Considerando la arquitectura de MIMD con SM, se hace el manejo de hilos por medio de la plataforma de Pthreads

Capítulo 5

Casos de Estudio

5.1. Introducción

En este capítulo se presentan tres casos de estudio que muestran la aplicación de los algoritmos propuestos. Cada caso de estudio muestra una descripción del problema, la manera en la que se utilizó la interpolación lineal, comparación de resultados de pronóstico, *Speed up* y la reducción del tiempo de cómputo.

Antes de entrar con los casos de estudio, se presenta las características del equipo de cómputo donde se realizaron las pruebas, la metodología que se realizó para la aplicación del *Speed up*, una descripción básica del método de interpolación para las series de tiempo y el método de pronóstico que nos permitirá comparar los resultados obtenidos.

5.1.1. Características del equipo usado para el cómputo en paralelo

El equipo donde se hicieron las pruebas del cómputo en paralelo es una computadora HP con una CPU intel(R) Xeon(R) E5405 que dispone de ocho elementos de procesamiento y una velocidad de 2.00 GHz. Se hacen las pruebas en el sistema operativo Xubuntu, que es un sistema de distribución Linux basado en Ubuntu. En la Tabla 5.1, se presentan las especificaciones de la CPU.

Tabla 5.1: Especificaciones de la CPU.

Arquitectura: x86_64
Modo(s) de operación de las CPUs: 32-bit, 64-bit
Orden de los bytes: Little Endian
CPU(s): 8
Lista de la(s) CPU(s) en línea: 0-7
Hilos(s) de procesamiento por núcleo: 1
Núcleo(s) por <<socket>>: 4
<<Socket(s)>>: 2
Modo(s) NUMA: 1
ID de fabricante: GenuineIntel
Familia de CPU: 6
Modelo: 23
Nombre del modelo: Intel(R) Xeon(R) CPU E5405 @ 2.00GHz
Revisión: 6
CPU MHz: 2000.068
BogoMIPS: 4000.13
Virtualización: VT-x
Caché L1d: 32k
Caché L1i: 32k
Caché L2: 6144k
CPU(s) del modo NUME 0: 0-7

5.1.2. Medición del tiempo de ejecución del cómputo en paralelo

El tiempo de ejecución de cada programa se mide utilizando el comando `time`, fijándose en la terminal de Linux antes de declarar el ejecutable. El uso de `time` establece el tiempo que duró el programa en finalizar. Es necesario conocer el tiempo de ejecución, ya que mediante a esto, se puede aplicar el *Speed up*.

Los recursos que dispone una computadora no son constantes, por ello, el hacer una sola medición del tiempo no sería confiable. Siendo así, se establece una metodología que propone realizar diez mediciones de tiempo por cada método de pronóstico de series de tiempo usando $P = 1, \ldots, 8$ elementos de procesamiento. Posteriormente, en cada conjunto de mediciones, se eliminan la medición mayor y menor del tiempo, sobrando ocho mediciones. Después, se calcula el promedio de las mediciones restantes. Con esto, se tiene

el promedio de tiempo de ejecución utilizando $P = 1, \ldots, 8$ elementos de procesamiento, pudiéndose aplicar el *speed up* y obtener resultados confiables.

5.1.3. Datos artificiales en las series de tiempo

El cómputo en paralelo es eficiente para ejecutar grandes cantidades de cálculos en tiempos de ejecución relativamente cortos. Por este motivo, no es apropiado trabajar con series de tiempo pequeñas, ya que no refleja el cómputo paralelo en el tiempo de ejecución. En la literatura, no es accesible la información a series de tiempo de grandes cantidades de datos, siendo necesario generar datos artificiales para mostrar el funcionamiento de los algoritmos propuestos en este trabajo de tesis. Esto se hace por medio de un método numérico de interpolación lineal, ya que da una aproximación de los υ valores intermedios entre dos puntos por medio de una recta. La ventaja de la interpolación lineal en las series de tiempo, es que genera artificialmente una serie mayor de datos, conservando las características de la serie de tiempo.

La interpolación lineal se calcula mediante:

$$f_1(x) = f(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) \tag{5.1}$$

donde $f_1(x)$ es variable dependiente de x, $x = 1, 2, \ldots, \upsilon$ son los valores intermedios, υ son la cantidad de puntos que se van a interpolar, x_0 y $f(x_0)$ son los valores del punto 0 (punto inicial) y x_1 y $f(x_1)$ son los valores del punto 1 (punto final).

5.1.4. Comparación de métodos de pronóstico de series de tiempo

La métrica MAPE no brinda una buena base sobre la comparación de los métodos de pronóstico, ya que los resultados son relativos. Por ejemplo, considerando el método de pronóstico de series de tiempo de MA y WMA, el método de MA obtuvo en su pronóstico un MAPE de 10 % y el método de WMA obtuvo en su pronóstico un MAPE de 5 %. Si se compara ambos resultados, se puede llegar a la conclusión que el método de WMA obtuvo una mejor aproximación en el pronóstico debido a que su MAPE fue menor, pero este resultado no brinda información si representa un "buen pronóstico" [Wheelwright et al., 1998].

Una base para la comparación entre los métodos de pronóstico de series de tiempo es definir un modelo de pronóstico simple contra los cuales se puede comparar el desempeño de métodos más sofisticados. Para este caso, se utiliza el método de pronóstico de series de tiempo de Naive, con el fin de usarlo como referencia en la evaluación de otros métodos de pronóstico. Para aplicar el método de Naive, se considera el último valor del periodo t

disponible en la TW [Wheelwright et al., 1998]. Siendo así, el método de Naive se calcula mediante:

$$\hat{Y}_{t+1} = Y_t \tag{5.2}$$

donde $\hat{Y_{t+1}}$ el valor del pronóstico en el siguiente periodo t y Y_t es el valor real de la serie de tiempo en el periodo t.

5.2. Caso de Estudio 1: Cantidad de pasajeros de una aerolínea estadounidense

El primer caso de estudio trata del pronóstico anual de la cantidad de pasajeros mensuales que dispone una aerolínea estadounidense. La captura de los datos se realizó mensualmente en los años 1949-1960, siendo el conjunto de datos mostrado en la Figura 2.1. Esta serie de tiempo presenta dos tipos de patrones: estacionalidad creciente y tendencia creciente.

La serie de tiempo de la Figura 2.1 está constituida por 144 datos, donde su $frecuencia$ = 12 y No. $periodos$ = 12, tratándose de una serie de tiempo relativamente corta. Para extender la cantidad de datos por medio de (5.1), se considera un valor de $\upsilon_{period} = 1$ para los periodos y un valor de $\upsilon_{frec} = 718$ para la frecuencia. Con esto, se extiende a una serie de tiempo de, 181930 datos donde su $frecuencia$ = 7910 y $No.\ periodos$ = 23. En la Figura 5.1 se muestra la serie de tiempo extendida de la Figura 2.1 usando la interpolación lineal.

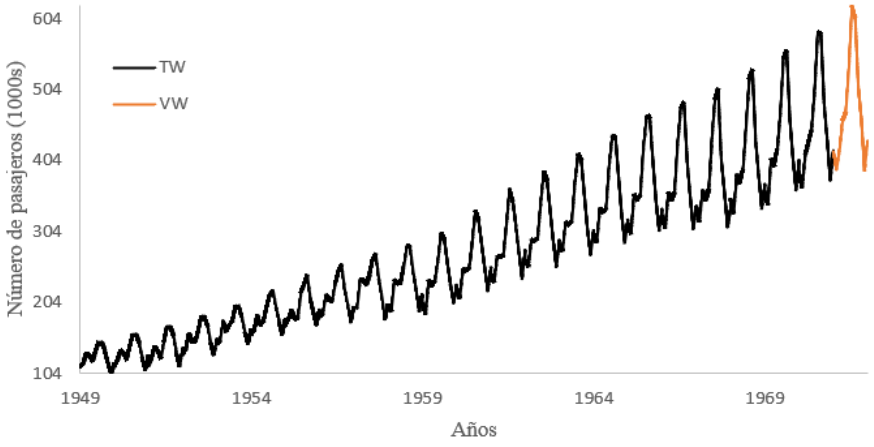

Figura 5.1: Serie de tiempo extendida por medio de interpolación lineal de la cantidad de pasajeros de una aerolínea estadounidense (1949-1960).

Para aplicar el pronóstico en este caso de estudio 1, se define los saltos que va a realizar

la búsqueda exhaustiva de los parámetros. En la Tabla 5.2 se muestra los saltos definidos en cada método de pronóstico para su prueba.

Tabla 5.2: Saltos establecidos en los parámetros para el pronóstico anual de la cantidad de pasajeros de una aerolínea estadounidense.

	MA	**WMA**	**ES**	**LS**	**MHW**	**AHW**
Saltos	—	2	0.0001	8	0.05	0.05

La Figura 5.2 ilustra una comparación del pronóstico con menor error de cada método, tomando solamente la VW de la serie de tiempo de la Figura 5.1. Se puede observar que es difícil identificar qué método presenta mejor aproximación, debido a que el resultado de los pronósticos tienen valores cercanos entre sí. Para validar cuantitativamente los resultados mostrados en la Figura 5.2, la Tabla 5.3 muestra la comparación de los métodos de pronóstico utilizando la métrica MAPE. Se aprecia que el método de pronóstico de LS es el que presentó una mayor aproximación a la VW, teniendo un MAPE de 1.7959 %. Es decir, el método de pronóstico de series de tiempo de LS con un parámetro de $\rho = 2$ obtuvo un menor error en el pronóstico, siendo el método que obtuvo mejores resultados. Además, se presenta un buen pronóstico, ya que su MAPE en la VW fue menor que el método de pronóstico series de tiempo de Naive, donde se tiene un MAPE de 5.0012 %.

Tabla 5.3: Comparación del pronóstico de los métodos de pronóstico de series de tiempo para los datos de la Figura 5.1 usando la métrica MAPE.

Pronóstico			
Método	**Búsqueda exhaustiva**	**Mejor parámetro**	**MAPE VW**
Naive	$---$	$---$	5.0012 %
MA	$\alpha = 1, 2, \ldots, 21$	$\alpha = 1$	5.0012 %
WMA	Considerando un valor de $n_{wma} = 5$.	$\beta_1 = 100\,\%$	5.0012 %
ES	$0 < \gamma < 1$ incrementos 0.0001	$\gamma = 1$	5.0012 %
LS	$\rho = 1, 2, \ldots, 8$	$\rho = 2$	1.7959 %
MHW	$0 < \phi$, ψ y $\omega < 1$ incrementos 0.05	$\phi = 0$, $\psi = 0$ y $\omega = 1$	2.6607 %
AWH	$0 < \phi$, ψ y $\omega < 1$ incrementos 0.05	$\phi = 0$, $\psi = 0$ y $\omega = 1$	3.6153 %

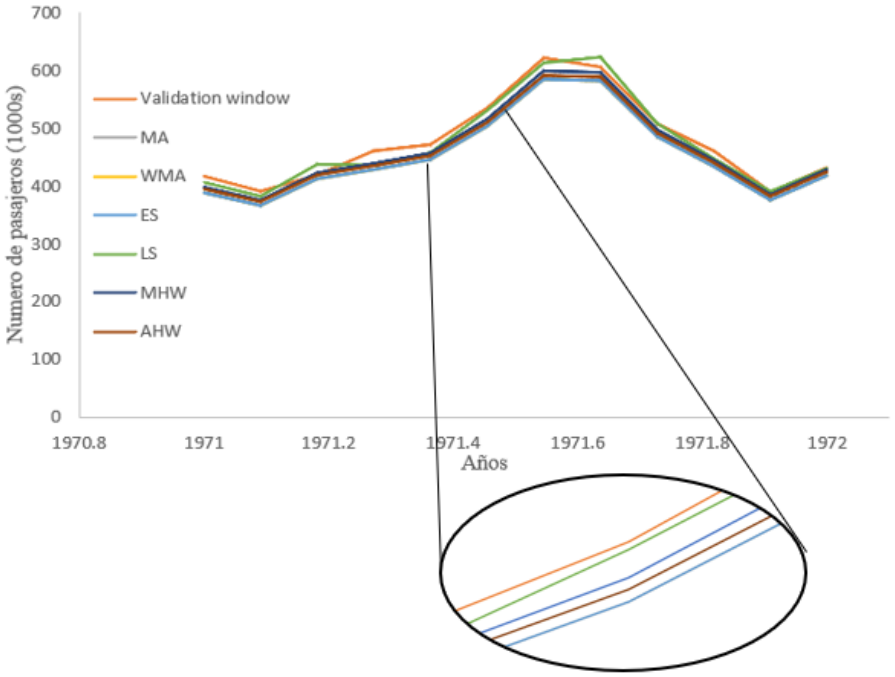

Figura 5.2: Comparación del pronóstico de los métodos de pronóstico de series de tiempo de los datos secuenciales de la Figura 5.1 haciendo una aplicación en la VW.

En la Figura 5.3 se muestra gráficamente el *Speed up* del cómputo en paralelo basado en hilos aplicado al pronóstico de la Figura 5.1. Se observa en el método de WMA, ES, AHW y MWH, la optimización del *Speed up* considerando 8 elementos de procesamiento. Además, los métodos de LS y MA presentan una optimización menor que los otros métodos. Siendo aproximadamente su *Speed up* de 1.5.

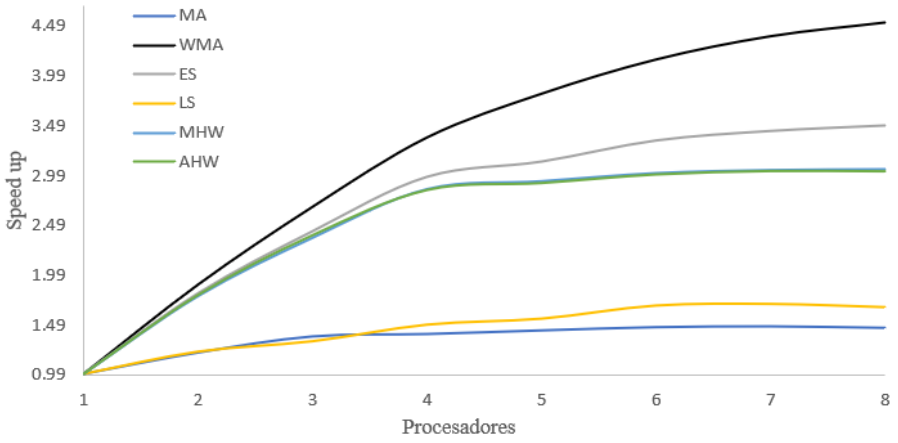

Figura 5.3: *Speed up* del cómputo en paralelo basado en hilos aplicado al pronóstico de la Figura 5.1.

En la Tabla 5.4 se muestra el *Speed up* expresado de manera cuantitativa, observándose

que, para los métodos de MA, LS y AHW al considerar más de 7 elementos de procesamiento, el tiempo de ejecución ya no presenta eficiencia en el cómputo en paralelo basado en hilos. Esto se debe porque el valor del *Speed up* disminuye al emplear 8 elementos de procesamiento. Para el caso del método de WMA, ES y MHW al considerar 8 elementos de procesamiento se sigue presentando eficiencia en los tiempos de ejecución, siendo su *Speed up* de 4.5277, 3.4987 y 3.0547 respectivamente.

Tabla 5.4: Resultados cuantitativos del *Speed up* en el cómputo en paralelo basado en hilos aplicado al pronóstico de los datos de la Figura 5.1.

Speed up						
Elementos de Procesamiento	**Métodos**					
	MA	**WMA**	**ES**	**LS**	**MHW**	**AHW**
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.2164	1.8966	1.8115	1.2232	1.7810	1.7913
3	1.3803	2.6773	2.4352	1.3273	2.3661	2.3898
4	1.4049	3.3736	2.9822	1.4936	2.8548	2.8433
5	1.4416	3.8124	3.1356	1.5537	2.9344	2.9130
6	1.4745	4.1561	3.3477	1.6854	3.0156	2.9989
7	1.4828	4.3898	3.4434	1.7016	3.0462	3.0330
8	1.4676	4.5277	3.4987	1.6689	3.0547	3.0305

Con referencia a los valores mostrados en la Tabla 5.4, en la Tabla 5.5 se presenta porcentualmente la reducción de tiempo de cada método de pronóstico usando diferentes cantidades de elementos de procesamiento. Se observa que la reducción máxima de tiempo de cómputo fue de 32.56 % en el método de MA, 77.91 % en el método de WMA, 71.42 % en el método de ES, 41.23 % en el método de LS, 67.26 % en el método de MHW y 67.03 % en el método de AHW. Con esto, se muestra que el método de WMA presenta mejor optimización del tiempo de ejecución, debido a que con 8 procesadores el *Speed up* es 4.52 veces más rápido que su versión secuencial, lo cual representa una reducción del 77.91 %.

Tabla 5.5: Comparación de la reducción de tiempo por medio del *Speed up* del cómputo en paralelo basado en hilos del Caso de Estudio 1.

Elementos de procesamiento	**MA**	**WMA**	**ES**	**LS**	**MHW**	**AHW**
1	00.00 %	00.00 %	00.00 %	00.00 %	00.00 %	0.000 %
2	17.79 %	47.27 %	44.80 %	18.25 %	43.85 %	44.18 %
3	27.55 %	62.65 %	58.94 %	24.66 %	57.78 %	58.16 %
4	28.82 %	70.36 %	66.47 %	33.05 %	64.97 %	64.83 %
5	30.63 %	73.77 %	68.11 %	35.63 %	65.92 %	65.67 %
6	32.18 %	75.94 %	70.13 %	40.67 %	66.84 %	66.66 %
7	32.56 %	77.22 %	70.96 %	41.23 %	67.17 %	67.03 %
8	31.86 %	77.91 %	71.42 %	40.08 %	67.26 %	67.00 %

El primer caso de estudio presenta el pronóstico anual de la cantidad de pasajeros que tiene una aerolínea estadounidense. Debido a que la serie de tiempo no podría reflejar la ventaja del cómputo en paralelo por disponer de poca cantidad de datos, se extendió la serie de tiempo mediante la interpolación lineal. Se presentaron mejores resultados en el pronóstico con el método de LS, pero menor eficiencia en el uso del cómputo en paralelo. Por otra parte, el método de pronóstico de WMA mostró mayor eficiencia en el cómputo en paralelo, pero no obtuvo los mejores resultados en el pronóstico, ya que sus valores en el pronóstico fueron similares al método de Naive.

5.3. Caso de Estudio 2: Demanda eléctrica por balance del sistema interconectado nacional mexicano zona occidente

El segundo caso de estudio trata del pronóstico semestral del mes de enero-junio de la demanda eléctrica por balance del sistema interconectado nacional mexicano zona occidente. La captura de los datos se realizó en los años 2018-2021, siendo el conjunto de datos mostrado en la Figura 5.4. Esta serie de tiempo fue obtenida por medio de [CENACE, 2022], donde se tiene los datos capturados cada hora de los meses de enero-junio. Además, la serie de tiempo presenta patrones estacionales y cíclicos.

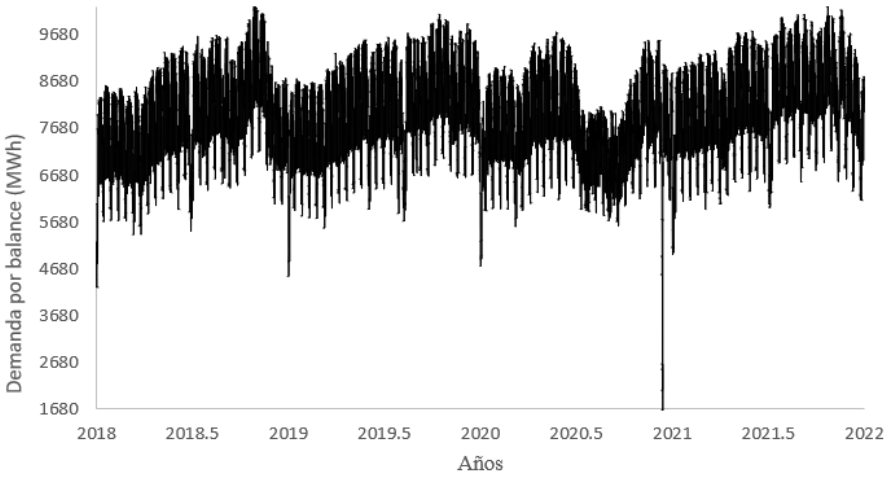

Figura 5.4: Serie de tiempo de la demanda eléctrica por balance del sistema interconectado nacional mexicano zona occidente 2018-2021.

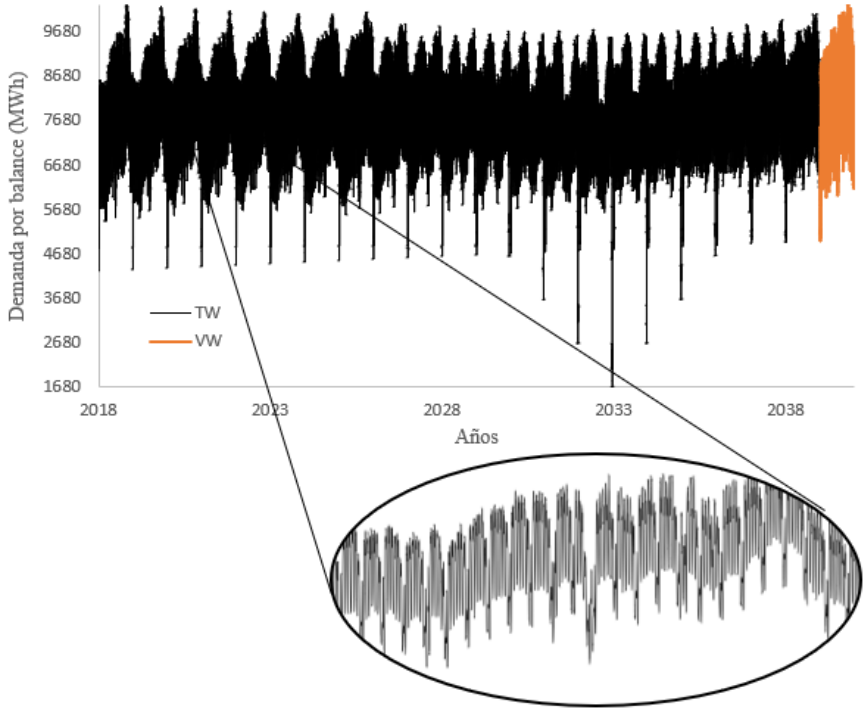

Figura 5.5: Serie de tiempo extendida por medio de la interpolación lineal de la demanda eléctrica por balance del sistema interconectado nacional mexicano zona occidente 2018-2021.

La serie de tiempo de la Figura 5.4 está constituida por, 17376 datos, donde la $frecuencia =$ 4344 y No. *periodos* =4, tratándose de una serie de tiempo relativamente corta. Para extender la cantidad de datos por medio de (5.1), se considera un valor de $\upsilon_{period} = 6$ para los periodos y un valor de $\upsilon_{frec} = 1$ para la frecuencia. Con esto, se extiende a una serie de tiempo de 191114 datos donde su $frecuencia = 8687$ y $No.\ periodos = 22$. En la Figura 5.5 se muestra la serie de tiempo extendida de la Figura 5.4 usando la interpolación

lineal.

Tabla 5.6: Saltos establecidos en los parámetros para el pronóstico semestral de la demanda eléctrica por balance del sistema interconectado nacional mexicano zona occidente.

	MA	**WMA**	**ES**	**LS**	**MHW**	**AHW**
Saltos	—	2	0.0001	8	0.05	0.05

Para aplicar el pronóstico en este caso de estudio 2, se define los saltos que va a realizar la búsqueda exhaustiva de los parámetros. En la Tabla 5.6 se muestra los saltos definidos en cada método de pronóstico para su prueba.

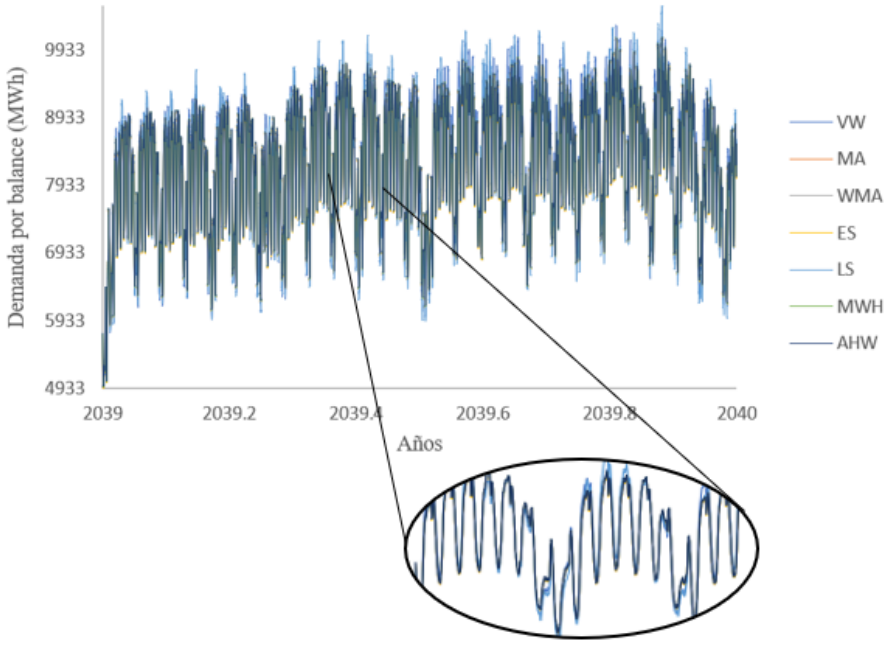

Figura 5.6: Comparación del pronóstico de los métodos de pronóstico de series de tiempo de los datos secuenciales de la Figura 5.5 haciendo una aplicación en la VW.

La Figura 5.6 la comparación gráfica del pronóstico con menor error de cada método, tomando solamente la VW de la serie de tiempo de la Figura 5.5. Se puede observar que de igual manera que el Caso de Estudio 1, se tiene dificultad en identificar qué método presenta mejor aproximación en el pronóstico. Para validar cuantitativamente los resultados mostrados en la Figura 5.6, la Tabla 5.7 se muestra la comparación de los métodos de pronóstico empleando la métrica MAPE. Se aprecia que el método de pronóstico de MHW es el que presentó una mayor aproximación a la VW, teniendo un MAPE de 1.5171 %. Es decir, el método de pronóstico de series de tiempo de MHW con un valor en sus parámetros de $\phi = 0$, $\psi = 0$ y $\omega = 1$ se obtuvo un menor error en el pronóstico, siendo el método que obtuvo mejores resultados. Además, se presenta un buen pronóstico, ya que, su MAPE en

la VW fue menor que el método de pronóstico series de tiempo de Naive, donde se tiene un MAPE de 1.6265 %.

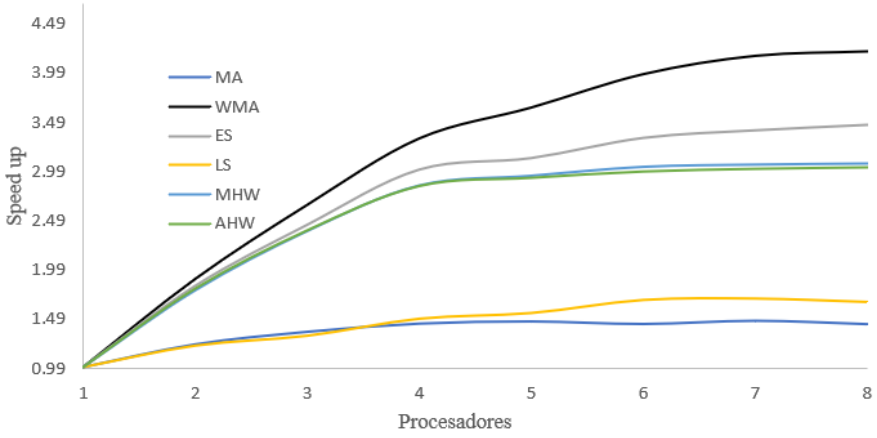

Figura 5.7: *Speed up* del cómputo en paralelo basado en hilos aplicado al pronóstico de la Figura 5.5.

Tabla 5.7: Comparación del pronóstico de los métodos de pronóstico de series de tiempo para los datos de la Figura 5.5 usando la métrica MAPE.

Pronóstico			
Método	**Búsqueda exhaustiva**	**Mejor parámetro**	**MAPE VW**
Naive	$---$	$---$	1.6265 %
MA	$\alpha = 1, 2, \ldots, 20$	$\alpha = 1$	1.6265 %
WMA	Considerando un valor de $n_{wma} = 7$.	$\beta_1 = 100\,\%$	1.6265 %
ES	$0 < \gamma < 1$ incrementos 0.0001	$\gamma = 1$	1.6265 %
LS	$\rho = 1, 2, \ldots, 8$	$\rho = 5$	1.7245 %
MHW	$0 < \phi$, ψ y $\omega < 1$ incrementos 0.05	$\phi = 0$, $\psi = 0$ y $\omega = 1$	1.5171 %
AWH	$0 < \phi$, ψ y $\omega < 1$ incrementos 0.05	$\phi = 0$, $\psi = 0$ y $\omega = 1$	1.5279 %

En la Figura 5.7 se muestra gráficamente el *Speed up* del cómputo en paralelo basado en hilos aplicado al pronóstico de la Figura 5.5. Se observa en el método de WMA, ES, AHW y MHW, la optimización del *speed up* considerando 8 elementos de procesamiento. Además, los métodos de LS y MA presentan una optimización menor en el tiempo de cómputo que los otros métodos. Se observa que hay un comportamiento similar que en el Caso de Estudio 1. Con excepción en los métodos de MHW y AHW, ya que, estos tienen

una pequeña diferencia entre sí al usar más de 5 elementos de procesamiento.

En la Tabla 5.8 se muestra el *Speed up* de manera cuantitativa, observándose que, para los métodos de MA y LS al considerar más de 7 elementos de procesamiento, el tiempo de ejecución ya no presenta eficiencia en el cómputo en paralelo basado en hilos. Esto se debe porque el valor del *Speed up* disminuye al emplear 8 elementos de procesamiento. Para el caso del método de WMA, ES, MHW y AHW al considerar 8 elementos de procesamiento aún presentan eficiencia en los tiempos de ejecución, siendo un *Speed up* de 4.217, 3.4709, 3.0723 y 3.0351, respectivamente.

Tabla 5.8: Resultados cuantitativos del *Speed up* en el cómputo en paralelo basado en hilos aplicado al pronóstico de los datos de la Figura 5.5.

Speed up						
Elementos de Procesamiento	**Métodos**					
	MA	**WMA**	**ES**	**LS**	**MHW**	**AHW**
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.2340	1.8979	1.8284	1.2209	1.7815	1.7995
3	1.3630	2.6529	2.4513	1.3223	2.3847	2.3931
4	1.4471	3.3300	3.0144	1.4976	2.8487	2.8450
5	1.4693	3.6448	3.1320	1.5564	2.9475	2.9288
6	1.4433	3.9851	3.3378	1.6907	3.0385	2.9928
7	1.4764	4.1716	3.4149	1.7048	3.0612	3.0203
8	1.4416	4.2170	3.4709	1.6699	3.0723	3.0351

Con referencia a los valores mostrados en la Tabla 5.8, en la Tabla 5.9 se presenta porcentualmente la reducción de tiempo de cada método de pronóstico usando el cómputo en paralelo basado en hilos. Se observa que la reducción máxima de tiempo de cómputo fue de 32.27 % en el método de MA, 76.29 % en el método de WMA, 71.19 % en el método de ES, 41.34 % en el método de LS, 67.45 % en el método de MHW y 67.05 % en el método de AHW. Con esto, se muestra que el método de WMA presenta mejor optimización del tiempo de ejecución, debido a que con 8 procesadores el *Speed up* es 4.21 veces más rápido que su versión secuencial, lo cual representa una reducción del 76.29 %.

Tabla 5.9: Comparación de la reducción de tiempo por medio del *Speed up* cómputo en paralelo basado en hilos del Caso de estudio 2.

Elementos de procesamiento	**MA**	**WMA**	**ES**	**LS**	**MHW**	**AHW**
1	00.00 %	00.00 %	00.00 %	00.00 %	00.00 %	00.00 %
2	18.96 %	47.31 %	45.31 %	18.10 %	43.87 %	44.43 %
3	26.63 %	62.30 %	59.21 %	24.38 %	58.07 %	58.21 %
4	30.90 %	69.97 %	66.83 %	33.23 %	64,90 %	64.85 %
5	31.94 %	72.56 %	68.07 %	35.75 %	66.07 %	65.86 %
6	30.71 %	74.91 %	70.04 %	40.85 %	67.09 %	66.59 %
7	32.27 %	76.03 %	70.72 %	41.34 %	67.33 %	66.89 %
8	30.63 %	76.29 %	71.19 %	40.12 %	67.45 %	67.05 %

El segundo caso de estudio presenta el pronóstico semestral de enero-junio de la demanda eléctrica por balance del sistema interconectado nacional mexicano zona occidente. Debido a que la serie de tiempo no podría reflejar la ventaja del cómputo paralelo en algunos métodos de pronóstico, se decidió extender la serie de tiempo mediante interpolación lineal. Se presentaron mejores resultados en el pronóstico con el método de MHW, y fue de los métodos de pronóstico que mostró mayor eficiencia en el uso del cómputo en paralelo. Por otra parte, el método de pronóstico de WMA presentó mayor eficiencia en el *Speed up*, pero no obtuvo los mejores resultados en el pronóstico. Aun así, la aproximación que tuvo fue bastante buena, teniendo un MAPE en la VW de 1.6265 % considerando un peso $\beta_1 = 100\,\%$, siendo valores del pronóstico similares al método de Naive.

5.4. Caso de Estudio 3: Número de nacimientos de la ciudad de New York

El tercer caso de estudio trata del pronóstico anual del número de nacimientos de la ciudad de New York. La captura de datos se realizó mensualmente en los años 1946-1959, siendo el conjunto de datos mostrado en la Figura 5.8. Esta serie de tiempo fue obtenida por medio de R, donde presenta patrones de estacionalidad, tendencia y cíclico.

La serie de tiempo de la Figura 5.8 está constituida por 168 datos, donde la $frecuencia = 12$ y No. *periodos* =14, tratándose de una serie de tiempo relativamente corta. Para extender la cantidad de datos por medio de (5.1), se considera un valor de $\upsilon_{period} = 1$ para

los periodos y un valor de $v_{frec} = 718$ para la frecuencia. Con esto, se extiende a una serie de tiempo de 213570 datos, donde su $frecuencia = 7910$ y $No.\ periodos = 27$. En la Figura 5.9 se muestra la serie de tiempo extendida de la Figura 5.8 usando la interpolación lineal.

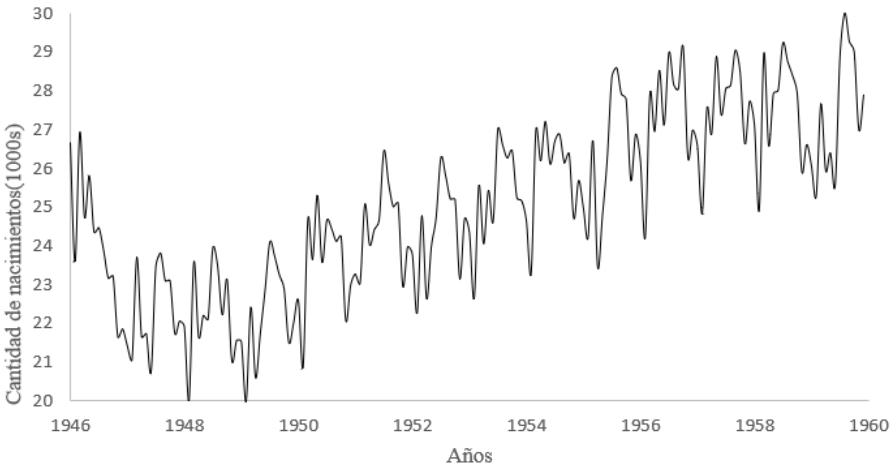

Figura 5.8: Serie de tiempo del número de nacimientos de la ciudad de New York desde 1946-1959.

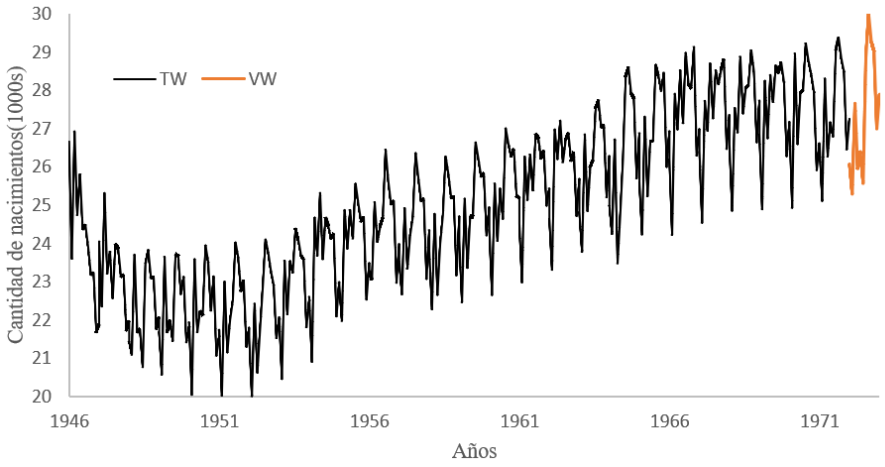

Figura 5.9: Serie de tiempo extendida por medio de la interpolación lineal del número de nacimientos de la ciudad de New York desde 1946-1959.

Para aplicar el pronóstico en este caso de estudio 3, se define los saltos que va a realizar la búsqueda exhaustiva de los parámetros. En la Tabla 5.10 se muestra los saltos definidos en cada método de pronóstico para su prueba.

Tabla 5.10: Saltos establecidos en los parámetros para el pronóstico anual del número de nacimientos de la ciudad de New York.

	MA	**WMA**	**ES**	**LS**	**MHW**	**AHW**
Saltos	—	2	0.0001	8	0.05	0.05

La Figura 5.10 ilustra una comparación del pronóstico con menor error de cada método, tomando solamente la VW de la serie de tiempo de la Figura 5.9. Se puede observar que es más fácil de identificar qué métodos presentan un pronóstico más cercano a la VW, siendo el caso del método de MHW el que presenta menor aproximación en el pronóstico. Para validar cuantitativamente los resultados mostrados en la Figura 5.10, la Tabla 5.11 muestra la comparación de los métodos de pronóstico utilizando la métrica MAPE. Se aprecia que los métodos de pronóstico de MA, WMA y ES, son los que presentaron una mayor aproximación a la VW, teniendo un MAPE de 1.8692 %. Es decir, los métodos de pronóstico de series de tiempo de MA, WMA y ES, con un valor en su parámetro de $\alpha = 1$, $\beta_1 = 100\,\%$ y $\gamma = 1$, respectivamente, se obtuvo un menor error en el pronóstico, siendo los métodos que obtuvieron mejores resultados. Además, se presenta un buen pronóstico, ya que su MAPE en la VW es similar al método de pronóstico de series de tiempo de Naive.

Tabla 5.11: Comparación del pronóstico de los métodos de pronóstico de series de tiempo para los datos de la Figura 5.9 usando la métrica MAPE.

Pronóstico			
Método	**Búsqueda exhaustiva**	**Mejor parámetro**	**MAPE VW**
Naive	$---$	$---$	1.8692 %
MA	$\alpha = 1, 2, \ldots, 25$	$\alpha = 1$	1.8692 %
WMA	Considerando un valor de $n_{wma} = 5$.	$\beta_1 = 100\,\%$	1.8692 %
ES	$0 < \gamma < 1$ incrementos 0.0001	$\gamma = 1$	1.8692 %
LS	$\rho = 1, 2, \ldots, 8$	$\rho = 3$	2.8084 %
MHW	$0 < \phi$, ψ y $\omega < 1$ incrementos 0.05	$\phi = 0{,}05$, $\psi = 0$ y $\omega = 1$	10.3139 %
AWH	$0 < \phi$, ψ y $\omega < 1$ incrementos 0.05	$\phi = 0$, $\psi = 0{,}05$ y $\omega = 1$	3.1177 %

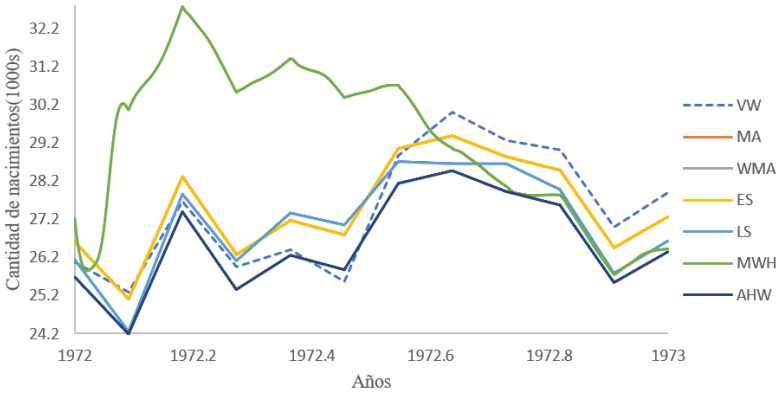

Figura 5.10: Comparación del pronóstico de los métodos de pronóstico de series de tiempo de los datos secuenciales de la Figura 5.9 haciendo una aplicación en la VW.

En la Figura 5.11 se muestra gráficamente el *Speed up* del cómputo en paralelo basado en hilos aplicado al pronóstico de la Figura 5.9. Se observa en el método de WMA, ES, AHW y MHW, la optimización del *Speed up* considerando 8 elementos de procesamiento. Además, los métodos de LS y MA presentan una optimización menor que los otros métodos de pronóstico.

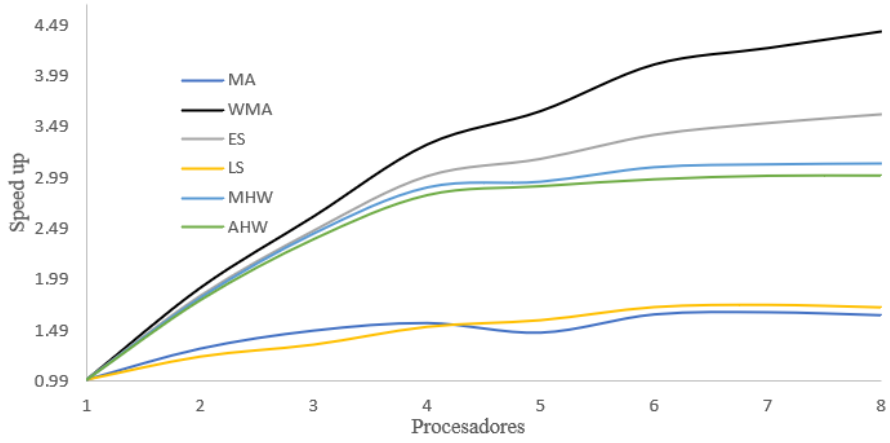

Figura 5.11: *Speed up* del cómputo en paralelo basado en hilos aplicado al pronóstico de la Figura 5.9.

En la Tabla 5.12 se muestra el *Speed up* expresado de manera cuantitativa, observándose que, para los métodos de MA y LS al considerar más de 7 elementos de procesamiento, el tiempo de ejecución ya no presenta eficiencia en el cómputo en paralelo basado en hilos. Esto se debe porque el valor del *Speed up* disminuye al emplear 8 elementos de procesamiento. Para el caso del método de WMA, ES, MHW y AHW al considerar 8 elementos de procesamiento se sigue presentando eficiencia en los tiempos de ejecución, siendo un *Speed up* de 4.4286, 3.6195, 3.1270 y 3.0081, respectivamente.

Con referencia a los valores mostrados en la Tabla 5.12, en la Tabla 5.13 se presenta porcentualmente la reducción de tiempo de cada método de pronóstico usando diferentes cantidades de elementos de procesamiento. Se observa que la reducción máxima de tiempo de cómputo fue de 38.29 % en el método de MA, 77.42 % en el método de WMA, 72.37 % en el método de ES, 42.69 % en el método de LS, 68.02 % en el método de MHW y 66.76 % en el método de AHW. Con esto, se muestra que el método de WMA presenta mejor optimización del tiempo de ejecución, debido a que con 8 procesadores el *Speed up* es 4.42 veces más rápido que su versión secuencial, lo cual representa una reducción del 77.42 %.

El tercer caso de estudio presenta el pronóstico anual del número de nacimientos de la ciudad de New York. Debido a que la serie de tiempo no podría reflejar la ventaja del cómputo en paralelo por disponer de poca cantidad de datos, se extendió la serie de tiempo mediante interpolación lineal. Se presentaron mejores resultados en el pronóstico con el método de MA, WMA, ES, donde se mostró mayor eficiencia en el uso del cómputo en paralelo con los métodos de WMA y ES.

Tabla 5.12: Resultados cuantitativos del *Speed up* en el cómputo en paralelo basado en hilos aplicado al pronóstico de los datos de la Figura 5.9.

Speed up						
Elementos de procesamiento	**Métodos**					
	MA	**WMA**	**ES**	**LS**	**MHW**	**AHW**
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.3096	1.9035	1.8272	1.2292	1.8043	1.781
3	1.4896	2.6077	2.4712	1.3485	2.4376	2.3825
4	1.5655	3.3159	3.0094	1.5262	2.8915	2.8154
5	1.4693	3.6448	3.1795	1.5922	2.9475	2.9033
6	1.6517	4.1037	3.4168	1.7228	3.0903	2.9712
7	1.6731	4.2653	3.5331	1.7450	3.1183	3.0060
8	1.6433	4.4286	3.6195	1.7198	3.1270	3.0081

Tabla 5.13: Comparación de la reducción de tiempo por medio del *Speed up* cómputo en paralelo basado en hilos del Caso de estudio 3.

Elementos de procesamiento	**MA**	**WMA**	**ES**	**LS**	**MHW**	**AHW**
1	0.00 %	00.00 %	00.00 %	00.00 %	00.00 %	00.00 %
2	20.88 %	47.46 %	45.27 %	18.65 %	44.58 %	43.85 %
3	30.66 %	61.65 %	59.53 %	25.84 %	58.98 %	58.03 %
4	34.35 %	69.84 %	66.77 %	34.48 %	65.42 %	64.48 %
5	38.18 %	73.26 %	68.55 %	37.19 %	66.67 %	65.56 %
6	37.14 %	75.63 %	70.73 %	41.95 %	67.64 %	66.34 %
7	38.29 %	76.56 %	71.70 %	42.69 %	67.93 %	66.73 %
8	36.74 %	77.42 %	72.37 %	41.85 %	68.02 %	66.76 %

5.5. Conclusiones

En este capítulo se presenta la comparación del tiempo de ejecución y pronóstico de los algoritmos propuestos para los métodos de MA, WMA, ES, LS y HW. Debido a la falta de acceso en la literatura para obtener series de tiempo confiables que representarán los sistemas eléctricos, se consideraron dos series de tiempo que son conocidas en la estadística. Además, se disponía de series de tiempo relativamente cortas, siendo necesario el uso de un método de interpolación para generar de manera artificial una serie mayor de datos, conservando los patrones que caracterizan a cada serie de tiempo. Se realizaron pruebas en series de tiempo que tuvieran patrones semejantes entre sí, con el fin de realizar una comparación de la eficiencia de la búsqueda exhaustiva en los parámetros de los métodos de pronóstico. En este caso, se consideraron series de tiempo con estacionalidad. Para el caso del pronóstico, se aplicó el método de pronóstico de series de tiempo de Naive para proporcionar una medida de referencia. Siendo así, el primer caso de estudio, considerando la serie de tiempo que representa a la cantidad de pasajeros de una aerolínea estadounidense, el mejor pronóstico se obtuvo con el método de LS teniendo un MAPE de 1.7959 %. El mejor *Speed up* se obtuvo con el método de WMA, reduciendo hasta 77.91 % el tiempo de ejecución. El segundo caso de estudio, considerando la serie de tiempo que representa a la demanda eléctrica por balance del sistema interconectado nacional mexicano zona occidente, el mejor pronóstico se obtuvo con el método de MHW, teniendo un MAPE de 1.5171 %. El mejor *Speed up* se obtuvo con el método de WMA, reduciendo hasta 76.28 % el tiempo

de ejecución. El tercer caso de estudio, considerando la serie de tiempo que representa al número de nacimientos de la ciudad de New York, el mejor pronóstico se obtuvo con el método de MA, WMA, ES, teniendo un MAPE de 1.8692 %. El mejor *Speed up* se obtuvo con el método de WMA, reduciendo hasta 77.42 % el tiempo de ejecución.

Se presenta la ventaja de la aplicación de la búsqueda exhaustiva para la selección del parámetro óptimo en los métodos de pronóstico de series de tiempo aplicado en los casos de estudio, ya que muestran buenas aproximaciones en su pronóstico cuando se trabaja con el parámetro adecuado. Se observa en todos los métodos de pronóstico la ventaja del cómputo en paralelo basado en hilos, teniendo una reducción en el tiempo de ejecución.

Capítulo 6

Conclusiones y Trabajos Futuros

6.1. Conclusiones Generales

Una vez finalizado el trabajo de investigación se obtienen las siguientes conclusiones:

- Los algoritmos desarrollados son útiles para la aplicación en diversas áreas. Por ejemplo, en los sistemas eléctricos de potencia, pronóstico de la demanda eléctrica, comportamiento de las reservas energéticas, conocer el comportamiento de variables meteorológicas, etc.

- Las series de tiempo de grandes dimensiones no son accesibles en la literatura. Por eso, se usa series de tiempo accesibles, pero estas no son lo suficientemente grandes para reflejar la ventaja del cómputo en paralelo basado en hilos. Siendo así, se consideró el uso de la interpolación lineal, ya que genera de manera artificial una serie mayor de datos, conservando la característica de la serie. Esto se debe a que solamente se está generando valores intermedios y no valores fuera de rango. Se utiliza para simular una serie de tiempo que dispone de gran cantidad de datos. No se genera datos de manera trivial, se emplea un método de interpolación, lo cual garantiza que no haya incongruencia en los datos.

- Se realizaron pruebas en series de tiempo que tuvieran patrones semejantes entre sí, con el fín de comparar la eficiencia de la búsqueda exhaustiva en los parámetros de los métodos de pronóstico.

- Este trabajo de tesis propone una serie de algoritmos que realizan una búsqueda exhaustiva en los parámetros que dispone los métodos de pronóstico de MA, WMA, ES, LS y HW para la selección del parámetro óptimo. Se propuso de esta manera, ya que, cada método de pronóstico tiene un rango de parámetros diferentes y uno

de esos parámetros va a presentar menor error en el pronóstico, garantizando que se obtiene un pronóstico cercano a la realidad. Este algoritmo requiere de esfuerzos computacionales altos, afectando directamente en el tiempo computacional. Por este motivo se sugiere el cómputo en paralelo basado en hilos.

- Se presentaron relativamente buenos resultados con la metodología propuesta en este trabajo de tesis. Esto se debe a que en los casos de estudio se obtuvo un error en el pronóstico igual o menor que el método de pronóstico de series de tiempo de Naive.

- Se aplica la métrica de error de pronóstico MAPE debido a que se trata de una métrica que expresa el error como un porcentaje del valor real, siendo una manera de interpretación sencilla para el usuario.

- Se usa una granularidad media-fuerte, considerando la paralelización en cuanto a funciones. Por este motivo, la arquitectura MIMD de SM presenta ser la mejor opción para el cómputo en paralelo en los algoritmos propuestos.

- Es importante observar la eficiencia que se tiene al considerar P cantidad de procesadores. Debido a que estos no mejoran el tiempo de manera ideal y dependiendo de su diseño de paralelismo y granularidad, se muestra qué tan eficiente es el cómputo en paralelo basado en hilos.

- El *speed up* es la relación del tiempo de ejecución de los programas secuenciales y los programas paralelos. El uso de múltiples procesadores presenta ventajas mientras el *speed up* siga aumentando.

- En promedio, se obtuvo en el pronóstico un MAPE de 2.8319 % en los métodos de MA, WMA y ES, 2.1096 % método de LS, 4.8305 % en el método de MHW y, 2.7536 % del método de AHW.

- En promedio, se tuvo una reducción del tiempo de ejecución9954 de 34.37 % en el método de MA, 77.2 % en el método de WMA, 71.64 % en el método de ES, 41.82 % método de LS, 66.57 % del método de MHW y, 67.053 % del método de AHW.

6.2. Trabajos Futuros

A continuación, se propone algunos trabajos futuros que pueden desarrollarse para extender el alcance de esta tesis. Entre los posibles trabajos futuros se destacan:

- Optimización de los métodos de pronóstico de MA, WMA, LS, ES, LS y HW por medio de reducción del gradiente para la selección automática del parámetro con menor error en el pronóstico.
- Usar las técnicas de simulación Montecarlo para el pronóstico de series de tiempo.
- Paralelización de los métodos de pronóstico ARMA, ARIMA, TBATS y Montecarlo.
- Aplicación del procesamiento del cómputo en paralelo al pronóstico de series de tiempo multivariable.
- Aplicación de un algoritmo que analice los valores disponibles de las series de tiempo y modifique los valores atípicos por medio de algún método de interpolación.
- Aplicación de los algoritmos propuestos en este trabajo de tesis a series de tiempo que estén sometidas a distintas técnicas de interpolación con el objeto de ver cuál es su efecto en la precisión del pronóstico.

Referencias

[Aguilar et al., 2004] Aguilar, J., Leiss, E., et al. (2004). Introducción a la computación paralela. *Venezuela: Gráficas Quinteto*.

[Barbhuiya and Liang, 2012] Barbhuiya, S. and Liang, Y. (2012). A multi-threaded programming strategy for parallel weather forecast model using c#. In *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, pages 319–324.

[Butenhof, 1997] Butenhof, D. R. (1997). *Programming with POSIX threads*. Addison-Wesley Professional.

[Casimiro, 2009] Casimiro, M. P. G. (2009). Análisis de series temporales: Modelos arima. *Universidad del País Basco*, 1(1):1–169.

[CENACE, 2022] CENACE (28 de Febrero de 2022). `https://www.cenace.gob.mx/Paginas/SIM/MetricasErroresPron.aspx`.

[Chatfield, 1978] Chatfield, C. (1978). The holt-winters forecasting procedure. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 27(3):264–279.

[Fatica and Ruetsch, 2014] Fatica, M. and Ruetsch, G. (2014). *CUDA Fortran for scientists and engineers*. Elsevier.

[Foster, 1995] Foster, I. (1995). *Designing and building parallel programs: concepts and tools for parallel software engineering*. Addison-Wesley.

[Hamid and Rahman, 2010] Hamid, M. A. and Rahman, T. A. (2010). Short term load forecasting using an artificial neural network trained by artificial immune system learning algorithm. In *2010 12th International Conference on Computer Modelling and Simulation*, pages 408–413.

[Hanke and Wichern, 2006] Hanke, J. E. and Wichern, D. W. (2006). *Pronósticos en los negocios*. Pearson educación.

[Hansun and Kristanda, 2017] Hansun, S. and Kristanda, M. B. (2017). Performance analysis of conventional moving average methods in forex forecasting. In *2017 International Conference on Smart Cities, Automation Intelligent Computing Systems (ICON-SONICS)*, pages 11–17.

[Hyndman and Fan, 2010] Hyndman, R. J. and Fan, S. (2010). Density forecasting for long-term peak electricity demand. *IEEE Transactions on Power Systems*, 25(2):1142–1153.

[Jurj et al., 2018] Jurj, D. I., Micu, D. D., and Muresan, A. (2018). Overview of electrical energy forecasting methods and models in renewable energy. In *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)*, pages 0087–0090.

[Kalos, 2005] Kalos, A. (2005). Automated heuristic growing of neural networks for non-linear time series models. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 1, pages 320–325 vol. 1.

[Lay, 2012] Lay, D. C. (2012). *Álgebra lineal y sus aplicaciones*. Pearson.

[Maity et al., 2013] Maity, S., Bonthu, S., Sasmal, K., and Warrior, H. (2013). Role of parallel computing in numerical weather forecasting models.

[Manzano et al., 2019] Manzano, M., Ayala, C., Gómez, C., and López Cuesta, L. (2019). A software service supporting software quality forecasting. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 130–132.

[Monteiro, 2009] Monteiro, C y Keko, H. y. B. R. y. M. V. y. B. A. y. W. J. . . y. C. G. y. o. (2009). Una guía rápida para la previsión de energía eólica: estado del arte 2009. Technical report, Argonne National Lab. (ANL), Argonne, IL (Estados Unidos).

[Moraes et al., 2013] Moraes, L., Flauzino, R., Araujo, M., and Batista, O. (2013). A fuzzy methodology to improve time series forecast of power demand in distribution systems. pages 1–5.

[Moustris et al., 2016] Moustris, K. P., Kavvadias, K. A., Kokkosis, A. I., and Paliatsos, A. G. (2016). One day-ahead forecasting of mean hourly global solar irradiation for energy management systems purposes using artificial neural network modeling. In *Me-*

diterranean Conference on Power Generation, Transmission, Distribution and Energy Conversion (MedPower 2016), pages 1–6.

[Niu et al., 2010] Niu, D., Shi, H., Li, J., and Wei, Y. (2010). Research on short-term power load time series forecasting model based on bp neural network. In *2010 2nd International Conference on Advanced Computer Control*, volume 4, pages 509–512.

[Pacheco, 2011] Pacheco, P. (2011). *An introduction to parallel programming*. Morgan Kaufmann.

[Shan et al., 2017] Shan, B., Jia, D., Zhang, L., Cao, F., and Sun, Y. (2017). Analysis of energy demand forecasting model in the context of electric power alteration. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 798–801.

[Skolpadungket et al., 2009] Skolpadungket, P., Dahal, K., and Harnpornchai, N. (2009). Forecasting stock returns using variable selections with genetic algorithm and artificial neural-networks. In *2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA)*, volume 1, pages 186–189.

[Snegirev et al., 2017] Snegirev, D. A., Eroshenko, S. A., Valiev, R. T., and Khalyasmaa, A. I. (2017). Algorithmic realization of short-term solar power plant output forecasting. In *2017 IEEE II International Conference on Control in Technical Systems (CTS)*, pages 228–231.

[Song et al., 2019] Song, D. R., Li, Q. A., Cai, Z., Li, L., Yang, J., Su, M., and Joo, Y. H. (2019). Model predictive control using multi-step prediction model for electrical yaw system of horizontal-axis wind turbines. *IEEE Transactions on Sustainable Energy*, 10(4):2084–2093.

[Szczupak et al., 2014] Szczupak, J., Crombez, S., and Pinto, L. (2014). Long term wind production modelling and forecast. In *2014 14th International Conference on Environment and Electrical Engineering*, pages 204–209.

[Utans et al., 1995] Utans, J., Moody, J., Rehfuss, S., and Siegelmann, H. (1995). Input variable selection for neural networks: application to predicting the u.s. business cycle. In *Proceedings of 1995 Conference on Computational Intelligence for Financial Engineering (CIFEr)*, pages 118–122.

[Wheelwright et al., 1998] Wheelwright, S., Makridakis, S., and Hyndman, R. J. (1998). *Forecasting: methods and applications*. John Wiley & Sons.

[Yizhen et al., 2011] Yizhen, L., Wenhua, Z., Ling, L., Jun, W., and Gang, L. (2011). The forecasting of shanghai index trend based on genetic algorithm and back propagation artificial neural network algorithm. In *2011 6th International Conference on Computer Science Education (ICCSE)*, pages 420–424.

[Yong et al., 2020] Yong, C., YingDa, J., Gang, X., JiaJia, C., DaYu, Q., and XiMing, Z. (2020). Load forecasting of electric vehicles based on monte carlo method. In *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, pages 1203–1206.

Apéndice A

Algoritmos Secuenciales de los Métodos de Pronóstico de Series de Tiempo

Promedios móviles

La Figura A.1 presenta el diagrama de flujo del método de pronóstico de MA. Este es descrito en seis fases, donde, la primera fase trata del análisis de la serie de tiempo para establecer los valores de $frecuencia$ y $periodos$. La Fase II define la TW y VW, donde la TW es una matriz que tiene dimensiones ($frecuencia\ x\ No.\ periodos - 1$) y la VW es un vector que tiene la misma cantidad de datos que la $frecuencia$. En la Fase III establece el valor del parámetro, α, donde sus variaciones van desde $\alpha = 1, 2, \ldots,\ No.\ periodos - 2$. La fase IV forma el modelo del método de MA con respecto al parámetro, α, ya establecido. En la Fase V, se aplica el pronóstico de series de tiempo, siendo un vector que tiene la misma dimensión que la VW. Por último, se tiene la fase de evaluación del método de pronóstico, donde se realiza el MAPE en cada renglón del vector de pronóstico, aplicando (2.2). Finalmente, estos datos se promedian para obtener un valor escalar del MAPE.

Promedios móviles ponderados

En la Figura A.2 se presenta el diagrama de flujo del método de pronóstico de WMA. Este algoritmo tiene su distribución en seis fases: valores iniciales, estructura de TW y VW, establecimiento de parámetros, modelo, pronóstico y evaluación del método de pronóstico. La Fase I establece los valores iniciales de $periodos$ y $frecuencia$. La Fase II separa los datos de la serie de tiempo en la TW y VW, siendo hasta este punto un algoritmo similar

al método de MA. La Fase III establece un valor n_{wma} para determinar la cantidad de términos y, a partir de esto, establecer la cantidad de pesos. Es importante considerar que la suma de los pesos debe ser del 100 % y la aplicación de estos a los datos más recientes debe ser mayor. En la Fase IV, se realiza la aplicación del método de WMA formando el modelo. En la fase V, se aplica el pronóstico, siendo un vector que tiene la misma dimensión que la VW. Finalmente, se tiene la Fase VI, en esta fase se aplica la métrica MAPE.

Suavización exponencial

El algoritmo del método de pronóstico de ES es similar al de MA. La diferencia se tiene en la variación del parámetro en la Fase III, ya que el valor del parámetro γ debe estar entre 0 y 1. A demás, en la Fase IV el modelo se realiza a partir de (2.7) - (2.9), y para el pronóstico, únicamente se utiliza (2.7). El método de pronóstico ES, a diferencia de MA y WMA, requiere de su modelo para poder realizar el pronóstico.

Mínimos cuadrados

La Figura A.3 muestra el diagrama de flujo del método de pronóstico de LS. Este método dispone de ocho fases. En la Fase I que corresponde a los valores iniciales, se establece la $frecuencia$ y los *periodos*. En la Fase II se forma el conjunto de datos que contiene la TW y VW, siendo hasta este punto un algoritmo similar al método de MA. En la Fase III, establece el orden de ρ en (2.10), donde sus variaciones pueden ir desde $\rho = 1, 2, \ldots,\ No.\ periodos - 1$. En la Fase IV, hace el cálculo de la matriz A y por medio del método de Gauss-Jordan se invierte la misma. La Fase V calcula el vector de incógnitas b, obteniendo los parámetros $\delta_0, \ldots, \delta_\rho$ por cada renglón que dispone la TW. En la Fase VI, se tiene el modelo de cada renglón de la TW. La Fase VII usa el modelo para hacer el pronóstico (caso similar al método de ES). Finalmente, la fase VIII usa la métrica MAPE para la evaluación del método de pronóstico de series de tiempo.

Holt-Winters

Los algoritmos del método de MHW y AHW tienen el mismo desarrollo en el diagrama de flujo. Entonces, para una explicación más concisa, se describe el algoritmo como el método de HW. La Figura A.4 presenta el diagrama de flujo del método de pronóstico de HW. El diagrama muestra siete fases, donde la primera fase corresponde a los valores iniciales, realizando un análisis gráfico para establecer la $frecuencia$ y *periodos* de la serie de tiempo. A demás, el método de MHW requiere de valores iniciales como lo son s y m,

siendo su valor el mismo que el establecido de la $frecuencia$. La Fase II, que presenta la distribución de los datos de la TW y VW, usa los valores establecidos de $frecuencia$ y *periodos* para separar la serie de tiempo en el conjunto de datos que corresponde a TW y VW, donde la TW (a diferencia de los métodos anteriores) es un vector que tiene una dimensión de ($frecuencia$*(No. *periodos*-1)). En la Fase III, calcula los valores iniciales para el método MHW o AHW. En la Fase IV, establece los parámetros ϕ, ψ y ω donde sus valores están en un rango entre 0 - 1. La Fase V realiza el modelo para el pronóstico, esta parte calcula L_t, b_t y S_t. Posteriormente, se forma el modelo de pronóstico. La fase VI aplica el pronóstico a los m datos por delante. Por último, se tiene la fase de evaluación del método de pronóstico, donde se hace el MAPE en cada renglón del vector de pronóstico.

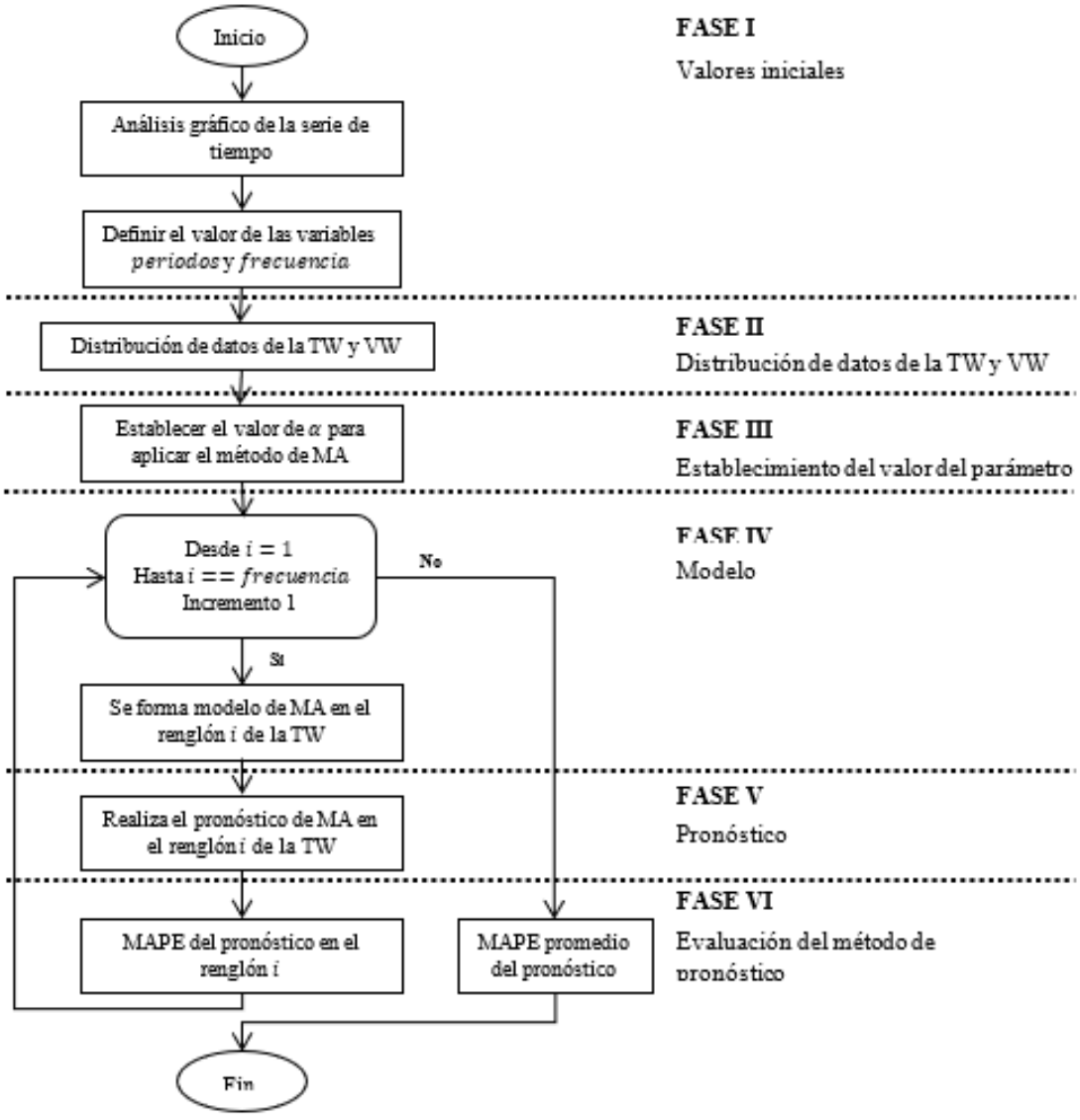

Figura A.1: Diagrama de Flujo del método de MA secuencial.

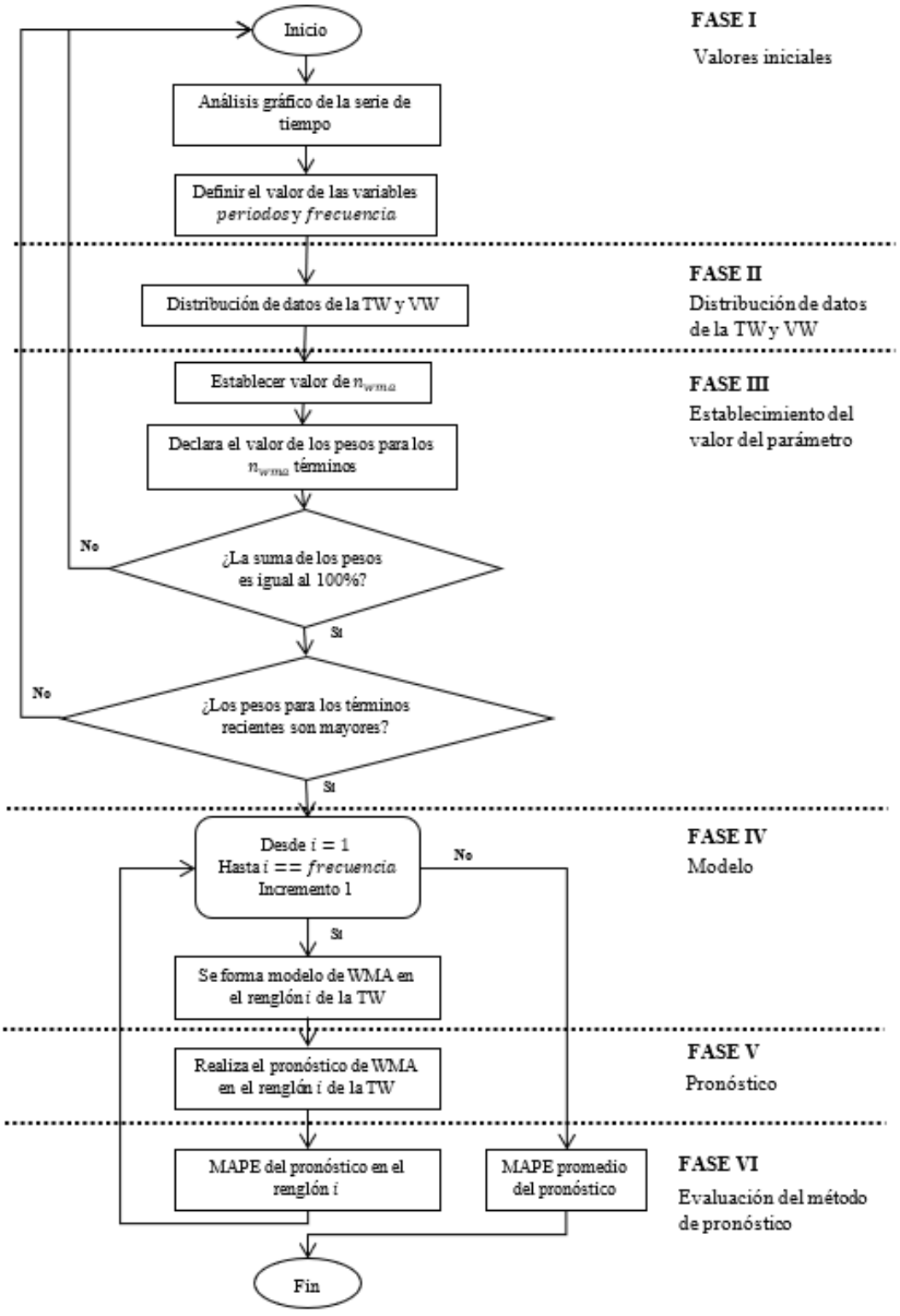

Figura A.2: Diagrama de Flujo del método de WMA secuencial.

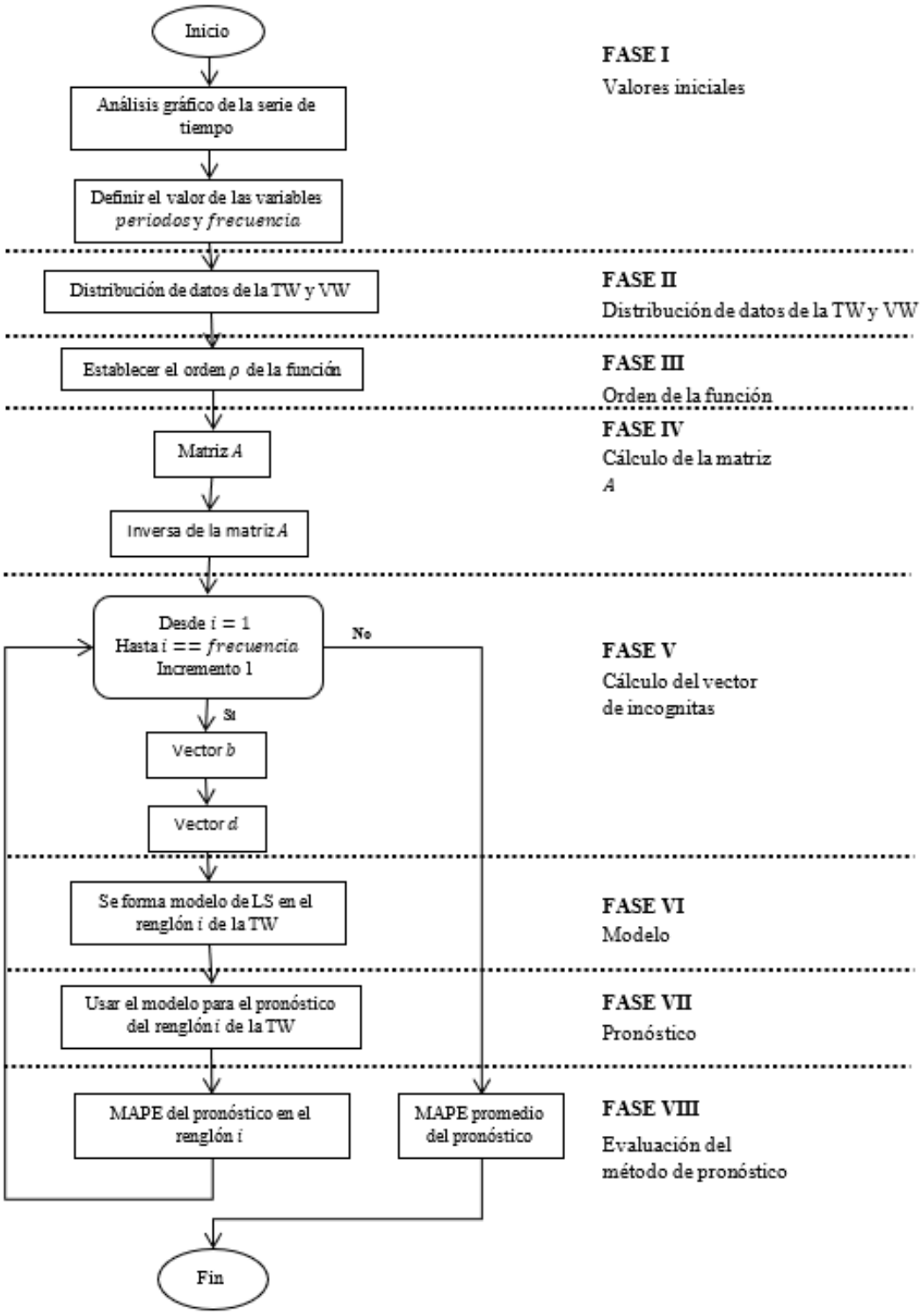

Figura A.3: Diagrama de Flujo del método de LS secuencial.

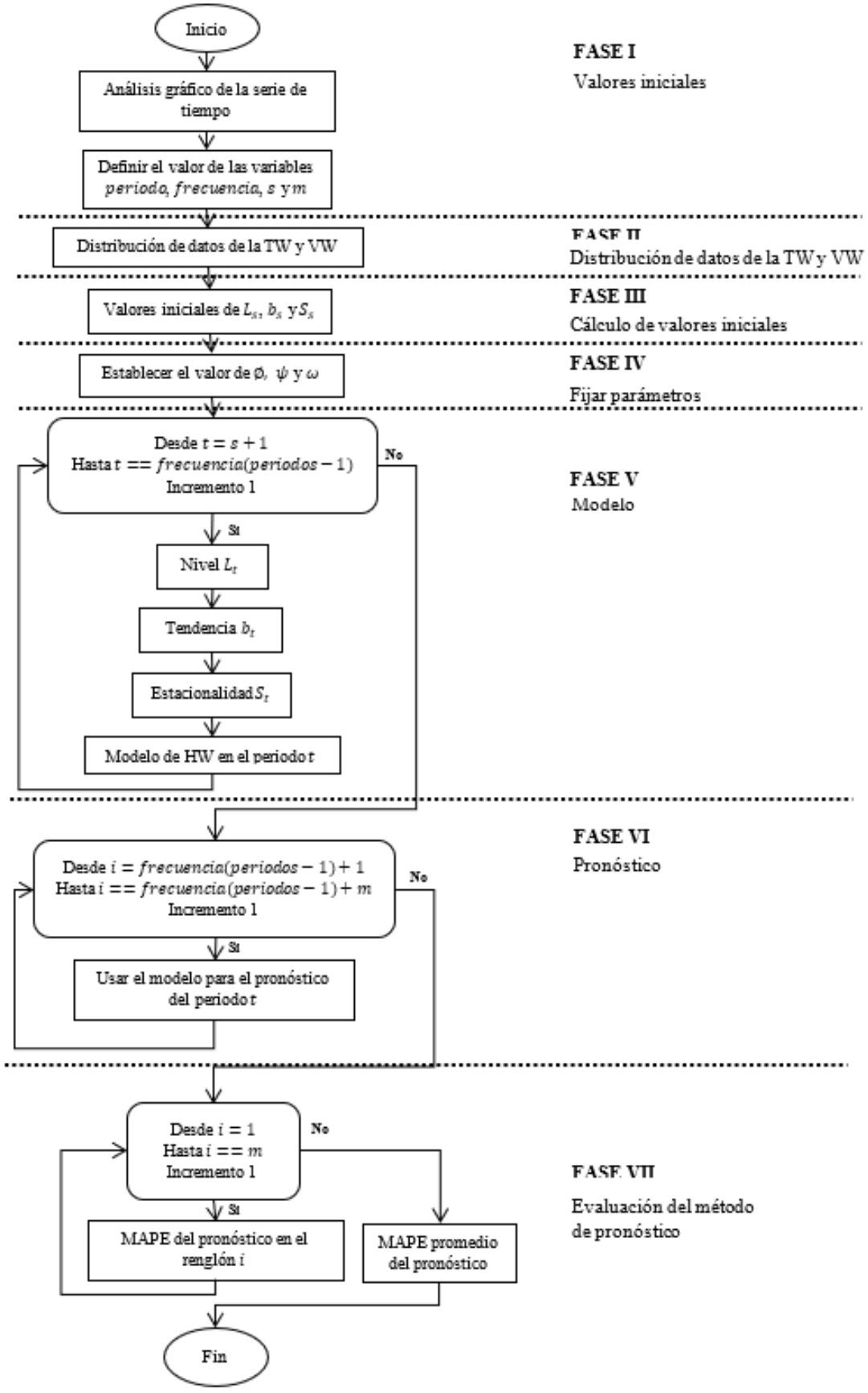

Figura A.4: Diagrama de Flujo del método de MH secuencial.

Apéndice B

Comandos principales de la librería Pthread

En el Capítulo 3 se dio una breve introducción sobre los conceptos básicos de los hilos y el cómo lo interpreta el hardware de la computadora multiprocesador. A continuación, se muestra cómo se representa los hilos en código de programación, presentándose en la Tabla B.1 algunas operaciones de referencia para el desarrollo de programas en la plataforma Pthreads.

Tabla B.1: Comandos principales para el uso de hilos.

`-lpthread`
`#include <pthread.h>`
`pthread_t <nombre identificador>;`

`int pthread_create(pthread_t* <nombre identificador>,`
`const pthread_attr_t* <nombre atributo>,`
`void* (*inicio rutina)(void*), void* arg_p);`

`int pthread_join(pthread_t* <nombre del identificador>,` `void** ret_val_p );`
`pthread_exit;`
`pthread_mutex_t <nombre identificador mutex>;`
`int pthread_mutex_lock(pthread_mutex_t *mutex);`
`pthread_mutex_unlock(pthread_mutex_t *mutex);`
`pthread_mutex_init(pthread_mutex_t *mutex,` `pthread_mutexattr_t *attr);`
`int pthread_mutex_destroy(pthread_mutex_t *mutex);`

Hilos

Para la compilación de los programas que trabajan con hilos, es necesario el uso del comando `-lpthread` en la terminal de C. Este comando le dice al compilador que se requiere vincular a la biblioteca pthreads. Se muestra a continuación la sintaxis:

```
gcc <nombre del archivo.c> -o <nombre del ejecutable> -lpthread
```

Pasando al código de programación, es necesario establecer la utilización de los hilos, esto se hace por medio de la librería `#include <pthread.h>`, siendo un archivo de encabezado de pthreads que declara las diversas funciones, constantes, tipos, etc., para la paralelización [Pacheco, 2011].

En la programación, el hilo de ejecución debe disponer de un identificador, en este caso se hace uso de `pthread_t`. Este se utiliza para crear el identificador; si el identificador es necesario solo dentro de una función, o si la función no regresará hasta que el hilo esté terminado, puede declarar el identificador con la clase de almacenamiento automático [Butenhof, 1997, Pacheco, 2011]. La sintaxis de `pthread_t` es:

```
pthread_t <nombre del identificador del hilo>;
```

Se emplea la función `pthread_create` para iniciar los hilos. Al llamarse este comando, se genera un hilo, o, en otras palabras, se bifurca el hilo principal. Dependiendo de las llamadas que se realicen a pthread_create darán como resultado múltiples hilos o bifurcaciones. La sintaxis de pthread_create es [Pacheco, 2011]:

```
int pthread_create( pthread_t* < nombre del identificador >,
                    const pthread_attr_t* < nombre del atributo >,
                    void* (*inicio de rutina)(void*), void* arg_p);
```

En el comando pthread_create, el primer argumento se trata de un puntero que va dirigido a `pthread_t`. Es importante que se considere que `pthread_t` debe declararse antes de su llamada en pthread_create. El segundo argumento se trata de los atributos que dispone el hilo, en este caso no es necesario su uso, siendo declarado el argumento `NULL` en la llamada a la función. El tercer argumento se trata de la función que va a ejecutar el hilo en paralelo, y el último argumento es un puntero que se pasa al inicio de la función que se ejecuta en paralelo [Pacheco, 2011].

El comando `pthread_join` se utiliza para esperar a que se complete el hilo asociado a `pthread_t`. La sintaxis de `pthread_join` es [Pacheco, 2011]:

`int pthread_join(pthread_t* < nombre del identificador >, void** ret_val_p );`

El primer argumento de `pthread_join`, es el puntero que va dirigido a `pthread_t`, y el segundo argumento se puede usar para recibir cualquier valor de retorno que sea calculado por el hilo.

Finalmente, se tiene la operación `pthread_exit` que termina la llamada del hilo. Se coloca al final de la función que va a ejecutar cada hilo en paralelo.

Mutex

Para el caso de la sincronización, pthreads proporciona exclusión mutua utilizando una forma especial del semáforo de Edsger Dijkstra, llamado mutex [Butenhof, 1997]. El uso del mutex requiere de un identificador, este se realiza a través de `pthread_mutex_t`.

```
pthread_mutex_t <nombre del identificador del mutex>;
```

Para asegurar que un hilo pueda comunicar datos de manera independiente, es necesario bloquear su mutex con `pthread_mutex_lock` para que pueda leer o escribir sobre las variables que lo requiera [Butenhof, 1997]. La operación requiere únicamente como argumento el identificador del mutex.

```
pthread_mutex_lock(pthread_mutex_t *mutex);
```

En el caso de desbloquear la utilización del mutex y el programa pueda correr de manera adecuada, es necesario usar la operación `pthread_mutex_unlock`. Al igual que `pthread_mutex_lock`, solamente requiere como argumento el identificador del mutex, siendo su sintaxis la siguiente [Butenhof, 1997]:

```
pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Los mutex deben inicializarse explícitamente, esto se ejecuta a través de `pthread_mutex_init`, donde su primer argumento especifica el identificador del mutex y como segundo argumento los atributos de tipo mutex. El comportamiento predeterminado de los atributos de tipo mutex es la privacidad para cada proceso, de modo que solo sea visible entre los hilos dentro de un proceso. A continuación, se muestra la sintaxis de `pthread_mutex_init` [Butenhof, 1997]

```
int pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t *attr);
```

Para asegurar el liberar cualquier estructura de datos relacionada con pthread. Para el caso del mutex, se tiene la operación de destrucción de mutex `pthread_mutex_destroy`. Esta operación únicamente requiere como argumento el identificador del mutex [Butenhof, 1997].

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Apéndice C

Código en C método MA

A continuación, se presenta el código de programación en C que realiza la selección del parámetro óptimo del método de pronóstico de series de tiempo de MA usando el cómputo en paralelo basado en hilos.

```
1  ////////////////////////////////////////////////////////////////////////////
2  // UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
3  // DIVISION DE ESTUDIOS DE POSGRADO DE INGENIERIA ELECTRICA
4  // AREA: SISTEMAS ELECTRICOS DE POTENCIA
5  // METODO DE PRONOSTICO: PROMEDIOS MOVILES
6  // PROGRAMADOR : JESUS DE LA TORRE BUCIO
7  // PARALELO
8  ////////////////////////////////////////////////////////////////////////////
9
10 ////////////////////////////////////////////////////////////////////////////
11 //LIBRERIAS
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <pthread.h>
16
17 ////////////////////////////////////////////////////////////////////////////
18 //VARIABLES GLOBALES
19 //VARIABLES DEL MAIN
20 long double datos, repeat, ndatatw;
21 // DIMENSIONES DE LA SERIE DE TIEMPO
22 int period, frec, core, pos_ft_tw;
23 //VARIABLES DEL MAPE
24 long double ref_tw, ref_vw;
25 // MEMORIA DINAMICA
26 long double  *st, **time_series, **tw, *vw, *MAPE_tw, *MAPE_vw, **global_ft_vw;
27 //ARCHIVOS DE ENTRADA Y SALIDA
28 FILE *f_in, *f_ts, *f_ft_vw, *f_param;
29 char *name_in, *name_ejecutable;
30 // MUTEX
31 pthread_mutex_t mutex;
32
33 ////////////////////////////////////////////////////////////////////////////
34 // ESTRUCTURA
35 typedef struct{
36     int ini;
37     int fin;
38 }args_threads;
39
40 ////////////////////////////////////////////////////////////////////////////
41 //FUNCIONES
```

```
42  void allocate_memory(void){// SE GENERA LA MEMORIA DINAMICA DE LAS VARIABLES GLOBALES
43      st = malloc(datos * sizeof(long double));
44      time_series = malloc(frec * sizeof(long double*));
45      tw = malloc(frec * sizeof(long double*));
46      vw = malloc(frec * sizeof(long double));
47      MAPE_tw = malloc(repeat * sizeof(long double));
48      MAPE_vw = malloc(repeat * sizeof(long double));
49      global_ft_vw = malloc(frec * sizeof(long double*));
50      if (st == NULL){printf("No se guardo espacio de memoria de st \n"); exit(1);}
51      if (time_series == NULL){printf("No se guardo espacio de memoria al array \"timeseries\"
            \n"); exit(1);}
52      if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n"); exit(1);}
53      if (vw == NULL){printf("No se guardo espacio de memoria al array \"vw\" \n"); exit(1);}
54      if (MAPE_tw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_tw\" \n");
            exit(1);}
55      if (MAPE_vw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_vw\" \n");
            exit(1);}
56      if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"global_ft_vw
            \" \n"); exit(1);}
57      for (int i = 0; i < frec; i++){
58              time_series[i] = malloc(period * sizeof(long double));
59              tw[i] = malloc((period-1) * sizeof(long double));
60              global_ft_vw[i] = malloc(repeat * sizeof(long double));
61              if (time_series == NULL){printf("No se guardo espacio de memoria al array \"
                    timeseries\" \n"); exit(1);}
62              if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n");
                    exit(1);}
63              if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    global_ft_vw\" \n"); exit(1);}
64      }
65  }
66  void read_data(void){// SE ESTABLECE TODA LA SERIE DE TIEMPO EN UN ARRAY LLAMADO "st"
67      f_in = fopen(name_in, "r");
68      if(f_in == NULL){ printf("\n Error al abrir el archivo de lectura de datos llamado %s \n"
            , name_in); exit(1);}
69      for (int i = 0; i < datos ; i++){
70              fscanf(f_in, "%Lf", &st[i]);
71      }
72      if(fclose(f_in) != 0){printf("\n no se ha podido cerrar el fichero %s \n", name_in); exit
            (1);}
73  }
74  void tw_vw(void){// SE ESTABLECE LA SERIE DE TIEMPO DE LA VENTANA DE VALIDACION Y
        ENTRENAMIENTO
75      int cont, pos = 0;
76      for(int i = 0; i < period ; i++){
77              cont = 0;
78              for(int j = 0; j < frec ; j++){
79                      time_series[j][i] = st[j + pos];
80                      cont = cont + 1;
81                      if(i < period-1){
82                              tw[j][i] = time_series[j][i];
83                      } else{
84                              vw[j] = time_series[j][i];
85                      }
86              }
87              pos = pos + cont;
88      }
89      //printf("\n serie de tiempo\n");
90      //for(int i = 0; i < frec ; i++){
91      //      for(int j = 0; j < period ; j++){
92      //              printf("%Lf ", time_series[i][j]);
93      //      }
94      //      printf("\n");
95      //}
96      //printf("\n Ventana de entrenamiento\n");
97      //for(int i = 0; i < frec ; i++){
```

```
98      //      for(int j = 0; j < period-1 ; j++){
99      //              printf("%Lf ", tw[i][j]);
100     //      }
101     //      printf("\n");
102     //}
103     //printf("\n Ventana de validacion\n");
104     //for(int i = 0; i < frec ; i++){
105     //      printf("%Lf \n", vw[i]);
106     //}
107 }
108 void free_memory(void){// SE LIBERA MEMORIA DINAMIECA
109     for (int i = 0 ; i < frec ; i++){
110             free(time_series[i]);
111             free(tw[i]);
112             free(global_ft_vw[i]);
113     }
114     free(global_ft_vw);
115     free(time_series);
116     free(tw);
117     free(st);
118     free(vw);
119     free(MAPE_vw);
120     free(MAPE_tw);
121 }
122 void *MA(void *rank){
123 //****** ESTABLECE LOS PARAMETROS DE INICIO Y FINAL DE CADA HILO
124     args_threads *thread_data = (args_threads*)rank;
125     int start = thread_data -> ini;
126     int end = thread_data -> fin;
127     //printf("Inicio %d Final %d", start, end);
128     //printf(" \n inicio del hilo %d fin del hilo %d \n", start,end);
129     for (int n = start; n < end; n++){
130             int alpha = n + 1;
131             // MEMORIA DINAMICA PARA EL RPONOSTICO
132             long double **ft_tw, *ft_vw;
133             ft_tw = malloc(frec * sizeof(long double*));
134             ft_vw = malloc(frec * sizeof(long double));
135             if (ft_vw == NULL){printf("No se guardo espacio de memoria al array \"ft_vw\" \n"
                    ); exit(1);}
136             if (ft_tw == NULL){printf("No se guardo espacio de memoria al array \"ft_tw\" \n"
                    ); exit(1);}
137             for (int i = 0; i < frec; i++){
138                     ft_tw[i] = malloc((period-1) * sizeof(long double));
139                     if (ft_tw == NULL){printf("No se guardo espacio de memoria al array \"
                            ft_tw\" \n"); exit(1);}
140             }
141             // MEMORIA DINAMICA VENTANA DE VALIDACION
142             long double *error_rest_vw, *error_abs_vw, *error_div_vw;
143             long double cont_vw, sum_vw;
144             error_rest_vw = malloc((frec) * sizeof(long double));
145             error_abs_vw = malloc((frec) * sizeof(long double));
146             error_div_vw = malloc((frec) * sizeof(long double));
147             if (error_rest_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    error_rest_vw\" \n"); exit(1);}
148             if (error_abs_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    error_abs_vw\" \n"); exit(1);}
149             if (error_div_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    error_div_vw\" \n"); exit(1);}
150             // MEMORIA DINAMICA VENTANA DE ENTRENAMIENTO
151             long double *error_rest_tw, *error_abs_tw, *error_div_tw;
152             long double cont_tw, cont1, sum_tw, alphaFrec = alpha*frec;
153             int pos_tw;
154             error_rest_tw = malloc((ndatatw-alphaFrec) * sizeof(long double));
155             error_abs_tw = malloc((ndatatw-alphaFrec) * sizeof(long double));
156             error_div_tw = malloc((ndatatw-alphaFrec) * sizeof(long double));
157             if (error_rest_tw == NULL){printf("No se guardo espacio de memoria al array \"
```

```
                    error_rest_tw\"␣\n"); exit(1);}
158             if (error_abs_tw == NULL){printf("No␣se␣guardo␣espacio␣de␣memoria␣al␣array␣\"
                    error_abs_tw\"␣\n"); exit(1);}
159             if (error_div_tw == NULL){printf("No␣se␣guardo␣espacio␣de␣memoria␣al␣array␣\"
                    error_div_tw\"␣\n"); exit(1);}
160 //************** METODO DE PRONOSTICO
161             long double sum;
162             for (int i = 0; i < frec; i++){
163                     // PRONOSTICO VENTANA DE ENTRENAMIENTO
164                     for (int j = 0 ; j < period-1 ; j++){
165                             if(j < alpha){
166                                     ft_tw[i][j] = 0;
167                             } else{
168                                     sum = 0;
169                                     for (int k = 0; k < alpha ; k++){
170                                             sum = sum + tw[i][j - 1 - k];
171                                     }
172                                     ft_tw[i][j] = sum / alpha;
173                             }
174                     }
175                     // PRONOSTICO VENTANA DE VALIDACION
176                     sum = 0;
177                     for(int j = period-alpha-1; j < period-1; j++){
178                             sum = sum + tw[i][j];
179                     }
180                     ft_vw[i] = sum/alpha;
181             }
182             //printf("\n pronostico ventana de entrenamiento para alpha = %d\n", alpha);
183             //for (int i = 0; i < frec; i++){
184             //      for (int j = 0 ; j < period-1 ; j++){
185             //              printf("%Lf ", ft_tw[i][j]);
186             //      }
187             //      printf("\n");
188             //}
189             /*printf("\n pronostico ventana de validacion para alpha = %d\n ", alpha);
190             for (int i = 0; i < frec; i++){
191                     printf("%Lf\n", ft_vw[i]);
192             }*/
193 //************** MAPE
194             // MAPE VENTANA DE ENTRENAMIENTO
195             cont_tw = 0; sum_tw = 0; pos_tw = 0;
196             //printf("\ncon alpha = %d\n",alpha);
197             for(int i = alpha; i < period-1 ; i++){
198                     cont1 = 0;
199                     for(int j = 0; j < frec ; j++){
200                             // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
201                             error_rest_tw[pos_tw+j] = tw[j][i] - ft_tw[j][i];
202                             // VALOR ABSOLUTO
203                             if(error_rest_tw[pos_tw+j] < 0){
204                                     error_abs_tw[pos_tw+j] = error_rest_tw[pos_tw+j] * -1;
205                             } else{
206                                     error_abs_tw[pos_tw+j] = error_rest_tw[pos_tw+j];
207                             }
208                             //printf("%Lf \n", error_abs_tw[pos_tw+j]);
209                             // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
210                             error_div_tw[pos_tw+j] = (error_abs_tw[pos_tw+j] / tw[j][i])*100;
211                             // SE REALIZA EL PROMEDIO
212                             cont_tw = cont_tw + 1;
213                             sum_tw = sum_tw + error_div_tw[pos_tw+j];
214                             cont1 = cont1 +1;
215                     }
216                     pos_tw = pos_tw + cont1;
217             }
218             //printf("\n El mape de la ventana de entrenamiento con alpha = %d es %Lf \n",
                    alpha, MAPE_tw[n]);
219             // VENTANA DE VALIDACION
```

```
220             cont_vw = 0; sum_vw = 0;
221             for(int i = 0; i < frec ; i++){
222                     // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
223                     error_rest_vw[i] = vw[i] - ft_vw[i];
224                     // VALOR ABSOLUTO
225                     if(error_rest_vw[i] < 0){
226                             error_abs_vw[i] = error_rest_vw[i] * -1;
227                     } else{
228                             error_abs_vw[i] = error_rest_vw[i];
229                     }
230                     // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
231                     error_div_vw[i] = (error_abs_vw[i] / vw[i])*100;
232                     // SE REALIZA EL PROMEDIO
233                     cont_vw = cont_vw + 1;
234                     sum_vw = sum_vw + error_div_vw[i];
235             }
236             //printf("\n el MAPE de la ventana de validacion con alpha = %d es %lf\n", alpha,
                        MAPE_vw[n]);
237             pthread_mutex_lock(&mutex);//BLOQUEA EL DATO CON EL MUTEX
238             MAPE_tw[n] = sum_tw/cont_tw;// SE CALCULA EL MAPE DE VENTANA DE ENTRENAMIENTO
239             MAPE_vw[n] = sum_vw/cont_vw;// SE CALCULA EL MAPE DE VENTANA DE VALIDACION
240             for (int i = 0; i < frec; i++){
241                             global_ft_vw[i][n] = ft_vw[i];
242             }
243             //MEJORES RESULTADOS
244             if(n > 0){
245                     if(MAPE_vw[n]< ref_vw ){
246                             ref_tw = MAPE_tw[n];
247                             ref_vw = MAPE_vw[n];
248                             pos_ft_tw = n;
249                     }
250             } else{
251                     ref_tw = MAPE_tw[n];
252                     pos_ft_tw = n;
253                     ref_vw = MAPE_vw[n];
254             }
255             pthread_mutex_unlock(&mutex);//DESBLOQUEA EL DATO CON EL MUTEX
256             // LIBERACION DE MEMORIA DINAMICA
257             free(error_rest_tw);
258             free(error_abs_tw);
259             free(error_div_tw);
260             free(error_rest_vw);
261             free(error_abs_vw);
262             free(error_div_vw);
263             for (int i = 0 ; i < frec ; i++){
264             free(ft_tw[i]);
265             }
266             free(ft_tw);
267             free(ft_vw);
268
269     }
270     pthread_exit(NULL);
271 }
272 void distribute(void){
273     // ALMACENA MEMORIA EN VECTORES DINAMICOS
274     int *amount_data = malloc(core * sizeof(int));// ALMACENA CANTIDAD DE DATOS A LEER
275     args_threads *data_sent = malloc(core * sizeof(args_threads));// ARGUMENTOS PARA LOS
            HILOS
276     pthread_t *th = malloc(core * sizeof(pthread_t));// VARIABLE DONDE ALMACENA LA CANTIDAD
            DE HILOS
277     if (amount_data == NULL){printf("No se guardo espacio de memoria al array \"amount_data\"
            \n"); exit(1);}
278     if (data_sent == NULL){printf("No se guardo espacio de memoria al array \"data_sent\" \n"
            ); exit(1);}
279     if (th == NULL){printf("No se guardo espacio de memoria al array \"th\" \n"); exit(1);}
280     pthread_mutex_init(&mutex,NULL); //SE INICIALIZA EL MUTEX
```

```
281     // DIVISION
282     div_t dist = div(repeat, core);
283     //printf("\n El valor entero es: %d \n", x.quot);
284     //printf("\n El residuo es: %d \n", x.rem);
285     // CANTIDAD DE DATOS A LEER POR CADA HILO
286     for (int i = 0; i < core; i++){
287             amount_data[i] = dist.quot;//numero entero
288     }
289     for (int i = 0; i < dist.rem; i++){
290             amount_data[i] = amount_data[i] + 1;//residuo
291     }
292     /*printf("\n Datos de finales de cada nucleo \n");
293     for (int i = 0 ; i < core ; i++){
294             printf("%d \n", amount_data[i]);
295     }*/
296     // DECLARA LOS PARAMETROS EN EL VECTOR DE ARGUMENTOS
297     for (int i = 0; i < core; i++){
298             if(i == 0){
299                     data_sent[i].ini = 0;
300                     data_sent[i].fin = amount_data[i];
301             } else{
302                     data_sent[i].ini = data_sent[i - 1].fin;
303                     data_sent[i].fin = data_sent[i].ini + amount_data[i];
304             }
305     }
306     // CREACION DE HILOS
307     for (int i = 0; i < core; i++){
308             pthread_create(&th[i], NULL, MA, &data_sent[i]);
309     }
310     for (int i= 0; i < core; i++){
311             pthread_join(th[i], NULL);
312     }
313     pthread_mutex_destroy(&mutex); //SE INICIALIZA EL MUTEX
314     // LIBERAR MEMORIA
315     free(amount_data);
316     free(data_sent);
317     free(th);
318 }
319 void best_parameter(void){
320     // MEMORIA DINAMICA
321     long double *error, *error_abs, *vec_min_error,ref_min_error;
322     int pos_min_error, pos, cont;
323     error = malloc(repeat * sizeof(long double));
324     error_abs = malloc(repeat * sizeof(long double));
325     vec_min_error = malloc(repeat * sizeof(long double));
326     if (error == NULL){printf("No se guardo espacio de memoria al array \"error\" \n"); exit
            (1);}
327     if (error_abs == NULL){printf("No se guardo espacio de memoria al array \"error_abs\" \n"
            ); exit(1);}
328     if (vec_min_error == NULL){printf("No se guardo espacio de memoria al array \"
            vec_min_error\" \n"); exit(1);}
329     // SE REALIZA EL VECTOR QUE CONTIENE LOS ERRORES ENTRE MAPE_TW Y MAPE_VW
330     printf("\n VECTOR ERROR ABSOLUTO \n");
331     for(int i = 0; i < repeat; i++){
332             error[i] = MAPE_tw[i] - MAPE_vw[i];
333             if(error[i] < 0){
334                     error_abs[i] = error[i] * -1;
335             } else{
336                     error_abs[i] = error[i];
337             }
338             printf("MAPE_tw = %Lf, MAPE_vw = %Lf y error = %Lf\n",MAPE_tw[i], MAPE_vw[i],
                    error_abs[i]);
339     }
340     for(int i = 0; i < repeat; i++){
341             for(int j = 0; j < repeat; j++){
342                     if(i == 0){
```

```
343                             if(j == 0){
344                                     ref_min_error = error_abs[j];
345                                     pos_min_error = j;
346                             } else{
347                                     if(ref_min_error < error_abs[j]){
348                                             ref_min_error = error_abs[j];
349                                             pos_min_error = j;
350                                     }
351                             }
352                             vec_min_error[i] = pos_min_error;
353                     } else{
354                             for(int k = 0; k < pos; k++){
355                                     if(j != vec_min_error[k]){
356                                             cont = 0;
357                                             if(cont == 0){
358                                                     ref_min_error = error_abs[j];
359                                                     pos_min_error = j;
360                                                     cont = cont + 1;
361                                             } else{
362                                                     if(ref_min_error < error_abs[j]){
363                                                             ref_min_error = error_abs[j];
364                                                             pos_min_error = j;
365                                                     }
366                                             }
367                                     }
368                             }
369                     }
370             }
371     }
372     free(error);
373     free(error_abs);
374     free(vec_min_error);
375 }
376 void output_file(void){
377 //***** FICHERO SERIE DE TIEMPO
378     f_ts = fopen("MA_time_series.txt", "w");
379     if(f_ts == NULL){ printf("\n Error al abrir el archivo que va a contener la serie de
          tiempo \n"); exit(1);}
380     fprintf(f_ts, "\n");
381     fprintf(f_ts, "***** METODO DE PRONOSTICO MA PARA SERIES DE TIEMPO *****\n");
382     fprintf(f_ts, "\n");
383     fprintf(f_ts, " INFORMACION DEL METODO MA:\n");
384     fprintf(f_ts, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
385     fprintf(f_ts, "- NOMBRE DEL FICHERO: %s\n", name_in);
386     fprintf(f_ts, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
387     fprintf(f_ts, "- CANTIDAD DE PERIODOS: %d\n", period);
388     fprintf(f_ts, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
389     fprintf(f_ts, "- USO DE PROCESADORES: %d\n", core);
390     fprintf(f_ts, "- PARAMETRO MAS EFICIENTE: %d\n", pos_ft_tw + 1);
391     fprintf(f_ts, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
392     fprintf(f_ts, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
393     fprintf(f_ts, "\n");
394     fprintf(f_ts, "\n");
395     fprintf(f_ts, " SERIE DE TIEMPO\n");
396     fprintf(f_ts, "\n");
397     fprintf(f_ts, "\n");
398     for (int i = 0 ; i < frec ; i++){
399             for(int j = 0 ; j < period ; j++){
400                     fprintf(f_ts, "%Lf ",  time_series[i][j]);
401             }
402             fprintf(f_ts, "\n");
403     }
404     fprintf(f_ts, "\n");
405     fprintf(f_ts, "\n");
406     fprintf(f_ts, " VENTANA DE ENTRENAMIENTO\n");
407     fprintf(f_ts, "\n");
```

```
408       fprintf(f_ts, "\n");
409       for (int i = 0 ; i < frec ; i++){
410               for(int j = 0 ; j < period-1 ; j++){
411                       fprintf(f_ts, "%Lf ",  tw[i][j]);
412               }
413               fprintf(f_ts, "\n");
414       }
415       fprintf(f_ts, "\n");
416       fprintf(f_ts, "\n");
417       fprintf(f_ts, " VENTANA DE VALIDACION\n");
418       fprintf(f_ts, "\n");
419       fprintf(f_ts, "\n");
420       for(int j = 0 ; j < frec ; j++){
421                       fprintf(f_ts, "%Lf \n",vw[j]);
422       }
423       if(fclose(f_ts) != 0){printf("\n no se ha podido cerrar el archivo que contiene la serie
              de tiempo \n"); exit(1);}
424   //***** FICHERO PARAMETRO
425       f_param = fopen("MA_parameters.txt", "w");
426       if(f_param == NULL){ printf("\n Error al abrir el archivo que va a contener los
              parametros \n"); exit(1);}
427       fprintf(f_param, "\n");
428       fprintf(f_param, "***** METODO DE PRONOSTICO MA PARA SERIES DE TIEMPO *****\n");
429       fprintf(f_param, "\n");
430       fprintf(f_param, " INFORMACION DEL METODO MA:\n");
431       fprintf(f_param, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
432       fprintf(f_param, "- NOMBRE DEL FICHERO: %s\n", name_in);
433       fprintf(f_param, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
434       fprintf(f_param, "- CANTIDAD DE PERIODOS: %d\n", period);
435       fprintf(f_param, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
436       fprintf(f_param, "- USO DE PROCESADORES: %d\n", core);
437       fprintf(f_param, "- PARAMETRO MAS EFICIENTE: %d\n", pos_ft_tw + 1);
438       fprintf(f_param, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
439       fprintf(f_param, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
440       fprintf(f_param, "\n");
441       fprintf(f_param, "\n");
442       fprintf(f_param, "\n");
443       fprintf(f_param, "PARAMETRO          MAPE                  MAPE \n");
444       fprintf(f_param, "                  ENTRENAMIENTO          VALIDACION \n");
445       fprintf(f_param, "\n");
446       fprintf(f_param, "\n");
447       for(int i = 0; i < repeat; i++){
448               fprintf(f_param, "%d           %Lf                %Lf \n", i+1, MAPE_tw[i], MAPE_vw
                      [i]);
449       }
450       if(fclose(f_param) != 0){printf("\n no se ha podido cerrar el archivo que contiene los
              parametros \n"); exit(1);}
451   //***** FICHERO PRONOSTICO DE LA VENTANA DE VALIDACION
452       f_ft_vw = fopen("MA_forecasting.txt", "w");
453       if(f_ft_vw == NULL){ printf("\n Error al abrir el archivo que contienen el pronostico de
              la ventana de entrenamiento \n"); exit(1);}
454       fprintf(f_ft_vw, "\n");
455       fprintf(f_ft_vw, "***** METODO DE PRONOSTICO MA PARA SERIES DE TIEMPO *****\n");
456       fprintf(f_ft_vw, "\n");
457       fprintf(f_ft_vw, " INFORMACION DEL METODO MA:\n");
458       fprintf(f_ft_vw, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
459       fprintf(f_ft_vw, "- NOMBRE DEL FICHERO: %s\n", name_in);
460       fprintf(f_ft_vw, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
461       fprintf(f_ft_vw, "- CANTIDAD DE PERIODOS: %d\n", period);
462       fprintf(f_ft_vw, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
463       fprintf(f_ft_vw, "- USO DE PROCESADORES: %d\n", core);
464       fprintf(f_ft_vw, "- PARAMETRO MAS EFICIENTE: %d\n", pos_ft_tw + 1);
465       fprintf(f_ft_vw, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
466       fprintf(f_ft_vw, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
467       fprintf(f_ft_vw, "\n");
468       fprintf(f_ft_vw, "\n");
```

```
469       fprintf(f_ft_vw, "\n");
470       for (int i = 0 ; i < frec ; i++){
471                         fprintf(f_ft_vw, "%Lf\n", global_ft_vw[i][pos_ft_tw]);
472       }
473       if(fclose(f_ft_vw) != 0){printf("\n no se ha podido cerrar el archivo que contiene el
              pronostico de la ventana de entrenamiento \n"); exit(1);}
474   }
475   ////////////////////////////////////////////////////////////////////////////
476   //FUNCION PRINCIPAL
477   int main(int arcg, char *argv[])
478   {
479   //******VARIABLES
480       name_ejecutable = argv[0];
481       name_in = argv[1];
482       frec = atoi(argv[2]);
483       period = atoi(argv[3]);
484       core = atoi(argv[4]);
485       datos = (frec*period);
486       ndatatw = (period-1)*frec;
487       repeat = period - 2;
488
489   //******FUNCIONES
490       printf("\n");
491       printf("***** METODO DE PRONOSTICO MA PARA SERIES DE TIEMPO *****\n");
492       printf("\n");
493       printf(" INFORMACION DEL METODO MA:\n");
494       printf("- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
495       printf("- NOMBRE DEL FICHERO: %s\n", name_in);
496       printf("- FRECUENCIA DE LOS DATOS: %d\n", frec);
497       printf("- CANTIDAD DE PERIODOS: %d\n", period);
498       printf("- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
499       printf("- USO DE PROCESADORES: %d\n", core);
500       printf("\n Reservando Memoria Dinamica...\n");
501       allocate_memory();
502       printf("\n Leyendo serie de tiempo...\n");
503       read_data();
504       printf("\n Ordenando serie de tiempo...\n");
505       tw_vw();
506       if (period > 2){
507               if(core <= repeat){
508                       printf("\n Inicio del programa...\n");
509                       distribute();
510                       printf("\n Salida de archivos...\n");
511                       output_file();
512               } else{
513                       printf("\n La cantidad de parametros a procesar son: %d. Por lo tanto no
                            se permite elegir una cantidad mayor de procesadores \n",period-2);
514               }
515       } else{
516               printf("\n La cantidad de periodos debe ser mayor a 2 \n");
517       }
518       printf("\n Liberar memoria Dinamica...\n");
519       free_memory();
520       return 0;
521   }
```

Apéndice D

Código en C método WMA

A continuación, se presenta el código de programación en C que realiza la selección del parámetro óptimo del método de pronóstico de series de tiempo de WMA usando el cómputo en paralelo basado en hilos.

```
1  ////////////////////////////////////////////////////////////////////////////
2  // UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
3  // DIVISION DE ESTUDIOS DE POSGRADO DE INGENIERIA ELECTRICA
4  // AREA: SISTEMAS ELECTRICOS DE POTENCIA
5  // METODO DE PRONOSTICO: PROMEDIOS MOVILES PONDERADOS
6  // PROGRAMADOR : JESUS DE LA TORRE BUCIO
7  // COMPUTO EN        PARALELO
8  ////////////////////////////////////////////////////////////////////////////
9
10 ////////////////////////////////////////////////////////////////////////////
11 //LIBRERIAS
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <pthread.h>
16
17 ////////////////////////////////////////////////////////////////////////////
18 //VARIABLES GLOBALES
19 //VARIABLES DEL MAIN
20 long double datos;
21 // DIMENSIONES DE LA SERIE DE TIEMPO
22 int period, frec, values_weight, weight_count = 5, core, best, steps;
23 //VARIABLES DEL MAPE
24 long double ref_tw, ref_vw, *pos_ft_tw;
25 // MEMORIA DINAMICA
26 long double  *st, **time_series, **tw, *vw, **pesos, *MAPE_tw, *MAPE_vw, **global_ft_vw;
27 //ARCHIVOS DE ENTRADA Y SALIDA
28 FILE *f_in, *f_ts, *f_ft_tw, *f_ft_vw, *f_param;
29 char *name_in, *name_ejecutable;
30 // MUTEX
31 pthread_mutex_t mutex;
32
33 ////////////////////////////////////////////////////////////////////////////
34 // ESTRUCTURA
35 typedef struct{
36     int ini;
37     int fin;
38 }args_threads;
39 ////////////////////////////////////////////////////////////////////////////
40 //FUNCIONES
41 void allocate_memory(void){// SE GENERA LA MEMORIA DINAMICA DE LAS VARIABLES GLOBALES
```

```
42      pesos = malloc(values_weight * sizeof(long double*));
43      if (pesos == NULL){printf("No se guardo espacio de memoria de pesos \n"); exit(1);}
44      for(int i = 0; i < values_weight; i++){
45              pesos[i] = malloc(weight_count * sizeof(long double));
46              if (pesos == NULL){printf("No se guardo espacio de memoria al array \"pesos\" \n"
                    ); exit(1);}
47      }
48      st = malloc(datos * sizeof(long double));
49      time_series = malloc(frec * sizeof(long double*));
50      tw = malloc(frec * sizeof(long double*));
51      vw = malloc(frec * sizeof(long double));
52      pos_ft_tw = malloc(weight_count * sizeof(long double));
53      global_ft_vw = malloc(frec * sizeof(long double*));
54      MAPE_tw = malloc(values_weight * sizeof(long double));
55      MAPE_vw = malloc(values_weight * sizeof(long double));
56      if (st == NULL){printf("No se guardo espacio de memoria de st \n"); exit(1);}
57      if (time_series == NULL){printf("No se guardo espacio de memoria al array \"timeseries\"
            \n"); exit(1);}
58      if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n"); exit(1);}
59      if (vw == NULL){printf("No se guardo espacio de memoria al array \"vw\" \n"); exit(1);}
60      if (pos_ft_tw == NULL){printf("No se guardo espacio de memoria al array \"pos_ft_tw\" \n"
            ); exit(1);}
61      if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"global_ft_vw
            \" \n"); exit(1);}
62      if (MAPE_tw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_tw\" \n");
            exit(1);}
63      if (MAPE_vw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_vw\" \n");
            exit(1);}
64      for (int i = 0; i < frec; i++){
65              time_series[i] = malloc(period * sizeof(long double));
66              tw[i] = malloc((period-1) * sizeof(long double));
67              global_ft_vw[i] = malloc(values_weight * sizeof(long double));
68              if (time_series == NULL){printf("No se guardo espacio de memoria al array \"
                    timeseries\" \n"); exit(1);}
69              if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n");
                    exit(1);}
70              if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    global_ft_vw\" \n"); exit(1);}
71      }
72  }
73  void read_data(void){// SE ESTABLECE TODA LA SERIE DE TIEMPO EN UN ARRAY LLAMADO "st"
74      f_in = fopen(name_in, "r");
75      if(f_in == NULL){ printf("\n Error al abrir el archivo de lectura de datos llamado %s \n"
            , name_in); exit(1);}
76      for (int i = 0; i < datos ; i++){
77              fscanf(f_in, "%Lf", &st[i]);
78      }
79      if(fclose(f_in) != 0){printf("\n no se ha podido cerrar el fichero %s \n", name_in); exit
            (1);}
80  }
81  void definition_cant_parameters(void){
82      int a, pos2 = 0, pos3 = 0, pos4 = 0, pos5= 0;
83      // dos pesos
84      for(int b = 0; b <= 50; b+= steps){
85              a = 100 - b;
86              if(a > b){
87                      pos2++;
88              }
89      }
90      //tres pesos
91      for(int c = 0; c <= 33; c+= steps){
92              for(int b = 0; b <= 50; b+= steps){
93                      a = 100 - b - c;
94                      if((a > b)&&(b > c)){
95                              pos3++;
96                      }
```

```
97              }
98      }
99      // cuatro pesos
100     for(int d = 0; d <= 25; d+= steps){
101             for(int c = 0; c <= 33; c+= steps){
102                     for(int b = 0; b <= 50; b+= steps){
103                             a = 100 - b - c - d;
104                             if((a > b)&&(b > c)&&(c > d)){
105                                     pos4++;
106                             }
107                     }
108             }
109     }
110     // cinco pesos
111     for(int e = 0; e <= 20; e+= steps){
112             for(int d = 0; d <= 25; d+= steps){
113                     for(int c = 0; c <= 33; c+= steps){
114                             for(int b = 0; b <= 50; b+= steps){
115                                     a = 100 - b - c - d - e;
116                                     if((a > b)&&(b > c)&&(c > d)&&(d > e)){
117                                             pos5++;
118                                     }
119                             }
120                     }
121             }
122     }
123     //printf("%d %d %d %d \n", pos2, pos3, pos4, pos5);
124     values_weight = pos2+ pos3+ pos4+ pos5;
125     //printf("%d \n", values_weight);
126 }
127 void weight(void){
128     int a, pos2 = 0, pos3 = 0, pos4 = 0, pos5= 0;
129     // dos pesos
130     for(int b = 0; b <= 50; b+= steps){
131             a = 100 - b;
132             if(a > b){
133                     pesos[pos2][0] = a;
134                     pesos[pos2][1] = b;
135                     pos2++;
136             }
137     }
138     //tres pesos
139     for(int c = 0; c <= 33; c+= steps){
140             for(int b = 0; b <= 50; b+= steps){
141                     a = 100 - b - c;
142                     if((a > b)&&(b > c)){
143                             pesos[pos3 + pos2][0] = a;
144                             pesos[pos3 + pos2][1] = b;
145                             pesos[pos3 + pos2][2] = c;
146                             pos3++;
147                     }
148             }
149     }
150     // cuatro pesos
151     for(int d = 0; d <= 25; d+= steps){
152             for(int c = 0; c <= 33; c+= steps){
153                     for(int b = 0; b <= 50; b+= steps){
154                             a = 100 - b - c - d;
155                             if((a > b)&&(b > c)&&(c > d)){
156                                     pesos[pos4 + pos3 + pos2][0] = a;
157                                     pesos[pos4 + pos3 + pos2][1] = b;
158                                     pesos[pos4 + pos3 + pos2][2] = c;
159                                     pesos[pos4 + pos3 + pos2][3] = d;
160                                     pos4++;
161                             }
162                     }
```

```
163             }
164     }
165     // cinco pesos
166     for(int e = 0; e <= 20; e+= steps){
167             for(int d = 0; d <= 25; d+= steps){
168                     for(int c = 0; c <= 33; c+= steps){
169                             for(int b = 0; b <= 50; b+= steps){
170                                     a = 100 - b - c - d - e;
171                                     if((a > b)&&(b > c)&&(c > d)&&(d > e)){
172                                             pesos[pos5 + pos4 + pos3 + pos2][0] = a;
173                                             pesos[pos5 + pos4 + pos3 + pos2][1] = b;
174                                             pesos[pos5 + pos4 + pos3 + pos2][2] = c;
175                                             pesos[pos5 + pos4 + pos3 + pos2][3] = d;
176                                             pesos[pos5 + pos4 + pos3 + pos2][4] = e;
177                                             pos5++;
178                                     }
179                             }
180                     }
181             }
182     }
183     /*for(int i = 0; i < values_weight; i++){
184             for(int j = 0; j < weight_count ; j++){
185                     printf("%Lf ", pesos[i][j]);
186             }
187             printf("\n");
188     }*/
189 }
190 void tw_vw(void){// SE ESTABLECE LA SERIE DE TIEMPO DE LA VENTANA DE VALIDACION Y
        ENTRENAMIENTO
191     int cont, pos = 0;
192     for(int i = 0; i < period ; i++){
193             cont = 0;
194             for(int j = 0; j < frec ; j++){
195                     time_series[j][i] = st[j + pos];
196                     cont = cont + 1;
197                     if(i < period-1){
198                             tw[j][i] = time_series[j][i];
199                     } else{
200                             vw[j] = time_series[j][i];
201                     }
202             }
203             pos = pos + cont;
204     }
205     /*printf("\n serie de tiempo\n");
206     for(int i = 0; i < frec ; i++){
207             for(int j = 0; j < period ; j++){
208                     printf("%Lf ", time_series[i][j]);
209             }
210             printf("\n");
211     }
212     printf("\n Ventana de entrenamiento\n");
213     for(int i = 0; i < frec ; i++){
214             for(int j = 0; j < period-1 ; j++){
215                     printf("%Lf ", tw[i][j]);
216             }
217             printf("\n");
218     }
219     printf("\n Ventana de validacion\n");
220     for(int i = 0; i < frec ; i++){
221             printf("%Lf \n", vw[i]);
222     }*/
223 }
224 void free_memory(void){// SE LIBERA MEMORIA DINAMIECA
225     for (int i = 0 ; i < frec ; i++){
226             free(time_series[i]);
227             free(tw[i]);
```

```
228         }
229         free(time_series);
230         free(tw);
231         for (int i = 0 ; i < values_weight; i++){
232                 free(pesos[i]);
233         }
234         free(pesos);
235         free(st);
236         free(vw);
237         free(MAPE_tw);
238         free(MAPE_vw);
239     }
240     void *WMA(void *rank){
241     //****** ESTABLECE LOS PARAMETROS DE INICIO Y FINAL DE CADA HILO
242         args_threads *thread_data = (args_threads*)rank;
243         int start = thread_data -> ini;
244         int end = thread_data -> fin;
245         //printf(" \n inicio del hilo %d fin del hilo %d \n", start,end);
246         int cont, dim_weight, pos;
247         // INICIA CON LOS RENGLONES DE LOS PESOS
248         for(int i = start; i < end; i++){
249                 cont = 0;
250                 // SE CUENTA LA CANTIDAD DE PESOS DISPONIBLES
251                 for(int j = 0; j < weight_count; j++){
252                         if(pesos[i][j] != 0){
253                                 cont = cont + 1;
254                         }
255                 }
256                 //printf("\n%d\n", cont);
257                 // EVALUA EN OTRA VARIABLE
258                 dim_weight = cont;
259                 // SE GENERA LA MEMORIA DINAMICA DEL ARREGLO
260                 long double *weight;
261                 weight = malloc(dim_weight * sizeof(long double));
262                 if (weight == NULL){printf("No se guardo espacio de memoria al array \"weight\" \
                        n"); exit(1);}
263                 // MEMORIA DINAMICA PARA EL RPONOSTICO
264                 long double **ft_tw, *ft_vw;
265                 ft_tw = malloc(frec * sizeof(long double*));
266                 ft_vw = malloc(frec * sizeof(long double));
267                 if (ft_vw == NULL){printf("No se guardo espacio de memoria al array \"ft_vw\" \n"
                        ); exit(1);}
268                 if (ft_tw == NULL){printf("No se guardo espacio de memoria al array \"ft_tw\" \n"
                        ); exit(1);}
269                 for (int j = 0; j < frec; j++){
270                         ft_tw[j] = malloc((period-1) * sizeof(long double));
271                         if (ft_tw == NULL){printf("No se guardo espacio de memoria al array \"
                                ft_tw\" \n"); exit(1);}
272                 }
273                 // MEMORIA DINAMICA VENTANA DE ENTRENAMIENTO
274                 long double *error_rest_tw, *error_abs_tw, *error_div_tw;
275                 long double cont_tw, cont1, sum_tw, dim_error = ((period-1)-dim_weight)*frec;
276                 int pos_tw;
277                 error_rest_tw = malloc(dim_error * sizeof(long double));
278                 error_abs_tw = malloc(dim_error * sizeof(long double));
279                 error_div_tw = malloc(dim_error * sizeof(long double));
280                 if (error_rest_tw == NULL){printf("No se guardo espacio de memoria al array \"
                        error_rest_tw\" \n"); exit(1);}
281                 if (error_abs_tw == NULL){printf("No se guardo espacio de memoria al array \"
                        error_abs_tw\" \n"); exit(1);}
282                 if (error_div_tw == NULL){printf("No se guardo espacio de memoria al array \"
                        error_div_tw\" \n"); exit(1);}
283                 // MEMORIA DINAMICA VENTANA DE VALIDACION
284                 long double *error_rest_vw, *error_abs_vw, *error_div_vw;
285                 long double cont_vw, sum_vw;
286                 int pos_vw;
```

```
287             error_rest_vw = malloc(frec * sizeof(long double));
288             error_abs_vw = malloc(frec* sizeof(long double));
289             error_div_vw = malloc(frec * sizeof(long double));
290             if (error_rest_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    error_rest_vw\" \n"); exit(1);}
291             if (error_abs_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    error_abs_vw\" \n"); exit(1);}
292             if (error_div_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    error_div_vw\" \n"); exit(1);}
293             // ASIGNA LOS VALORES AL NUEVO ARREGLO PARA COMENZAR CON EL PRONOSTICO
294             //printf("\n Los pesos son:      \n");
295             // SE CAMBIA EL ORDEN DE LOS PESOS DE MENOR A MAYOR
296             pos = dim_weight-1;
297             for(int j = 0; j < dim_weight; j++){
298                     weight[pos] = pesos[i][j] / 100;
299                     pos = pos-1;
300             }
301             /*for(int j = 0; j < dim_weight; j++){
302                     printf("%Lf \n", weight[j]);
303             }*/
304 //************* METODO DE PRONOSTICO DE PROMEDIOS MOVILES PONDERADOS
305             long double sum = 0;
306             cont = 0;
307             for(int j = 0; j < frec; j++){
308                     // PRONOSTICO VENTANA DE ENTRENAMIENTO
309                     for(int k = dim_weight; k < period-1 ; k++ ){
310                             for(int l = 0; l < dim_weight; l++){
311                                     sum = sum + (tw[j][k - dim_weight + l] * weight[l]);
312                             }
313                             ft_tw[j][k] = sum;
314                             sum = 0;
315                     }
316                     // PRONOSTICO VENTANA DE VALIDACION
317                     for(int k = 0; k < dim_weight; k++){
318                             sum = sum + (tw[j][(period-1) - dim_weight + k] * weight[k]);
319                     }
320                     ft_vw[j] = sum;
321                     sum = 0;
322             }
323             /*printf("\n PRONOSTICO VENTANA DE ENTRENAMIENTO \n");
324             for(int j = 0; j < frec; j++){
325                     for(int k = 0; k < period-1 ; k++ ){
326                             printf("%Lf ", ft_tw[j][k]);
327                             }
328                             printf("\n");
329             }
330             printf("\n PRONOSTICO VENTANA DE VALIDACION \n");
331             for(int k = 0; k < frec; k++){
332                     printf("%Lf \n", ft_vw[k]);
333             }*/
334             free(weight);
335
336 //************** MAPE
337             // VENTANA DE ENTRENAMIENTO
338             cont_tw = 0; sum_tw = 0; pos_tw = 0;
339             //printf("\ncon pesos = %d\n",i);
340             for(int j = dim_weight; j < period-1 ; j++){
341                     cont1 = 0;
342                     for(int k = 0; k < frec ; k++){
343                             // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
344                             error_rest_tw[pos_tw+k] = tw[k][j] - ft_tw[k][j];
345                             // VALOR ABSOLUTO
346                             if(error_rest_tw[pos_tw+k] < 0){
347                                     error_abs_tw[pos_tw+k] = error_rest_tw[pos_tw+k] * -1;
348                             } else{
349                                     error_abs_tw[pos_tw+k] = error_rest_tw[pos_tw+k];
```

```
350                             }
351                             //printf("%Lf \n", error_abs_tw[pos_tw+k]);
352                             // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
353                             error_div_tw[pos_tw+k] = (error_abs_tw[pos_tw+k] / tw[k][j])*100;
354                             // SE REALIZA EL PROMEDIO
355                             cont_tw = cont_tw + 1;
356                             sum_tw = sum_tw + error_div_tw[pos_tw+k];
357                             cont1 = cont1 +1;
358                     }
359                     pos_tw = pos_tw + cont1;
360             }
361             //printf("\n El mape de la ventana de entrenamiento con alpha = %d es %Lf \n",i,
                    MAPE_tw[i]);
362             // VENTANA DE VALIDACION
363             cont_vw = 0; sum_vw = 0;
364             for(int j = 0; j < frec ; j++){
365                     // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
366                     error_rest_vw[j] = vw[j] - ft_vw[j];
367                     // VALOR ABSOLUTO
368                     if(error_rest_vw[j] < 0){
369                             error_abs_vw[j] = error_rest_vw[j] * -1;
370                     } else{
371                             error_abs_vw[j] = error_rest_vw[j];
372                     }
373                     // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
374                     error_div_vw[j] = (error_abs_vw[j] / vw[j])*100;
375                     // SE REALIZA EL PROMEDIO
376                     cont_vw = cont_vw + 1;
377                     sum_vw = sum_vw + error_div_vw[j];
378             }
379             //printf("\n el MAPE de la ventana de validacion con alpha = %d es %Lf\n", i,
                    MAPE_vw[i]);
380             //MEJORES RESULTADOS
381             pthread_mutex_lock(&mutex);//BLOQUEA EL DATO CON EL MUTEX
382             MAPE_tw[i] = sum_tw/cont_tw;// SE CALCULA EL MAPE VENTANA DE ENTRENAMIENTO
383             MAPE_vw[i] = sum_vw/cont_vw;//SE CALCULA EL MAPE VENTANA DE VALIDACION
384             for (int j = 0; j < frec; j++){
385                             global_ft_vw[j][i] = ft_vw[j];
386             }
387             if(i > 0){
388                     if(MAPE_vw[i]< ref_vw ){
389                             ref_tw = MAPE_tw[i];
390                             ref_vw = MAPE_vw[i];
391                             pos_ft_tw[0] = pesos[i][0];
392                             pos_ft_tw[1] = pesos[i][1];
393                             pos_ft_tw[2] = pesos[i][2];
394                             pos_ft_tw[3] = pesos[i][3];
395                             pos_ft_tw[4] = pesos[i][4];
396                             best = i;
397                     }
398             } else{
399                     ref_tw = MAPE_tw[i];
400                     pos_ft_tw[0] = pesos[i][0];
401                     pos_ft_tw[1] = pesos[i][1];
402                     pos_ft_tw[2] = pesos[i][2];
403                     pos_ft_tw[3] = pesos[i][3];
404                     pos_ft_tw[4] = pesos[i][4];
405                     ref_vw = MAPE_vw[i];
406                     best = i;
407             }
408             pthread_mutex_unlock(&mutex);//DESBLOQUEA EL DATO CON EL MUTEX
409             //LIBERAR MEMORIA
410             free(error_rest_tw);
411             free(error_abs_tw);
412             free(error_div_tw);
413             free(error_rest_vw);
```

```
414                 free(error_abs_vw);
415                 free(error_div_vw);
416                 for (int i = 0 ; i < frec ; i++){
417                         free(ft_tw[i]);
418                 }
419                 free(ft_tw);
420                 free(ft_vw);
421         }
422         //printf("\nFin de hilo\n");
423         pthread_exit(NULL);
424 }
425 void distribute(void)
426 {
427         // ALMACENA MEMORIA EN VECTORES DINAMICOS
428         int *amount_data = malloc(core * sizeof(int));// ALMACENA CANTIDAD DE DATOS A LEER
429         args_threads *data_sent = malloc(core * sizeof(args_threads));// ARGUMENTOS PARA LOS
                HILOS
430         pthread_t *th = malloc(core * sizeof(pthread_t));// VARIABLE DONDE ALMACENA LA CANTIDAD
                DE HILOS
431         if (amount_data == NULL){printf("No se guardo espacio de memoria al array \"amount_data\"
                \n"); exit(1);}
432         if (data_sent == NULL){printf("No se guardo espacio de memoria al array \"data_sent\" \n"
                ); exit(1);}
433         if (th == NULL){printf("No se guardo espacio de memoria al array \"th\" \n"); exit(1);}
434         // DIVISION
435         div_t dist = div(values_weight, core);
436         //printf("\n El valor entero es: %d \n", x.quot);
437         //printf("\n El residuo es: %d \n", x.rem);
438         // CANTIDAD DE DATOS A LEER POR CADA HILO
439         for (int i = 0; i < core; i++){
440                 amount_data[i] = dist.quot;//numero entero
441         }
442         for (int i = 0; i < dist.rem; i++){
443                 amount_data[i] = amount_data[i] + 1;//residuo
444         }
445         /*printf("\n Datos de finales de cada nucleo \n");
446         for (int i = 0 ; i < core ; i++){
447                 printf("%d \n", amount_data[i]);
448         }*/
449         // DECLARA LOS PARAMETROS EN EL VECTOR DE ARGUMENTOS
450         for (int i = 0; i < core; i++){
451                 if(i == 0){
452                         data_sent[i].ini = 0;
453                         data_sent[i].fin = amount_data[i];
454                 } else{
455                         data_sent[i].ini = data_sent[i - 1].fin;
456                         data_sent[i].fin = data_sent[i].ini + amount_data[i];
457                 }
458         }
459         // CREACION DE HILOS
460         pthread_mutex_init(&mutex,NULL); //SE INICIALIZA EL MUTEX
461         for (int i = 0; i < core; i++){
462                 pthread_create(&th[i], NULL, WMA, &data_sent[i]);
463         }
464         for (int i= 0; i < core; i++){
465                 pthread_join(th[i], NULL);
466         }
467         pthread_mutex_destroy(&mutex); //SE INICIALIZA EL MUTEX
468         // LIBERAR MEMORIA
469         free(amount_data);
470         free(data_sent);
471         free(th);
472 }
473 void output_file(void)
474 {
475 //***** FICHERO SERIE DE TIEMPO
```

```
476       f_ts = fopen("WMA_time_series.txt", "w");
477       if(f_ts == NULL){ printf("\n Error al abrir el archivo que va a contener la serie de
              tiempo \n"); exit(1);}
478       fprintf(f_ts, "\n");
479       fprintf(f_ts, "***** METODO DE PRONOSTICO WMA PARA SERIES DE TIEMPO *****\n");
480       fprintf(f_ts, "\n");
481       fprintf(f_ts, " INFORMACION DEL METODO WMA:\n");
482       fprintf(f_ts, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
483       fprintf(f_ts, "- NOMBRE DEL FICHERO: %s\n", name_in);
484       fprintf(f_ts, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
485       fprintf(f_ts, "- CANTIDAD DE PERIODOS: %d\n", period);
486       fprintf(f_ts, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
487       fprintf(f_ts, "- USO DE PROCESADORES: %d\n", core);
488       fprintf(f_ts, "- PARAMETRO MAS EFICIENTE: %Lf, %Lf, %Lf, %Lf, %Lf y %Lf\n",
              pos_ft_tw[0], pos_ft_tw[1], pos_ft_tw[2], pos_ft_tw[3], pos_ft_tw[4], pos_ft_tw[5],
              pos_ft_tw[6]);
489       fprintf(f_ts, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
490       fprintf(f_ts, "- VALOR DE MAPE EN VW: %Lf \n\n\n", ref_vw);
491       fprintf(f_ts, "\n");
492       fprintf(f_ts, "\n");
493       fprintf(f_ts, " SERIE DE TIEMPO\n");
494       fprintf(f_ts, "\n");
495       fprintf(f_ts, "\n");
496       for (int i = 0 ; i < frec ; i++){
497                for(int j = 0 ; j < period ; j++){
498                         fprintf(f_ts, "%Lf ",  time_series[i][j]);
499                }
500                fprintf(f_ts, "\n");
501       }
502       fprintf(f_ts, "\n");
503       fprintf(f_ts, "\n");
504       fprintf(f_ts, " VENTANA DE ENTRENAMIENTO\n");
505       fprintf(f_ts, "\n");
506       fprintf(f_ts, "\n");
507       for (int i = 0 ; i < frec ; i++){
508                for(int j = 0 ; j < period-1 ; j++){
509                         fprintf(f_ts, "%Lf ",  tw[i][j]);
510                }
511                fprintf(f_ts, "\n");
512       }
513       fprintf(f_ts, "\n");
514       fprintf(f_ts, "\n");
515       fprintf(f_ts, " VENTANA DE VALIDACION\n");
516       fprintf(f_ts, "\n");
517       fprintf(f_ts, "\n");
518       for(int j = 0 ; j < frec ; j++){
519                         fprintf(f_ts, "%Lf \n",vw[j]);
520       }
521       if(fclose(f_ts) != 0){printf("\n no se ha podido cerrar el archivo que contiene la serie
              de tiempo \n"); exit(1);}
522   //***** FICHERO PARAMETRO
523       f_param = fopen("WMA_parameters.txt", "w");
524       if(f_param == NULL){ printf("\n Error al abrir el archivo que va a contener los
              parametros \n"); exit(1);}
525       fprintf(f_param, "\n");
526       fprintf(f_param, "***** METODO DE PRONOSTICO WMA PARA SERIES DE TIEMPO *****\n");
527       fprintf(f_param, "\n");
528       fprintf(f_param, " INFORMACION DEL METODO WMA:\n");
529       fprintf(f_param, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
530       fprintf(f_param, "- NOMBRE DEL FICHERO: %s\n", name_in);
531       fprintf(f_param, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
532       fprintf(f_param, "- CANTIDAD DE PERIODOS: %d\n", period);
533       fprintf(f_param, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
534       fprintf(f_param, "- USO DE PROCESADORES: %d\n", core);
535       fprintf(f_param, "- PARAMETRO MAS EFICIENTE: %Lf, %Lf, %Lf, %Lf, %Lf y %Lf\n",
              pos_ft_tw[0], pos_ft_tw[1], pos_ft_tw[2], pos_ft_tw[3], pos_ft_tw[4], pos_ft_tw[5],
```

```
            pos_ft_tw[6]);
536         fprintf(f_param, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
537         fprintf(f_param, "- VALOR DE MAPE EN VW: %Lf \n\n\n", ref_vw);
538         fprintf(f_param, "\n");
539         fprintf(f_param, "\n");
540         fprintf(f_param, "PARAMETRO              MAPE                  MAPE \n");
541         fprintf(f_param, "                  ENTRENAMIENTO          VALIDACION \n\n\n");
542         for(int i = 0; i < values_weight; i++){
543                 fprintf(f_param, " %Lf, %Lf, %Lf, %Lf Y %Lf           %Lf               %Lf \n",
                        pesos[i][0], pesos[i][1], pesos[i][2], pesos[i][3], pesos[i][4],  MAPE_tw[i
                        ], MAPE_vw[i]);
544         }
545         if(fclose(f_param) != 0){printf("\n no se ha podido cerrar el archivo que contiene los
                parametros \n"); exit(1);}
546     //***** FICHERO PRONOSTICO DE LA VENTANA DE VALIDACION
547         f_ft_vw = fopen("WMA_forecasting.txt", "w");
548         if(f_ft_vw == NULL){ printf("\n Error al abrir el archivo que contienen el pronostico de
                la ventana de entrenamiento \n"); exit(1);}
549         fprintf(f_ft_vw, "\n");
550         fprintf(f_ft_vw, "***** METODO DE PRONOSTICO WMA PARA SERIES DE TIEMPO *****\n");
551         fprintf(f_ft_vw, "\n");
552         fprintf(f_ft_vw, " INFORMACION DEL METODO WMA:\n");
553         fprintf(f_ft_vw, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
554         fprintf(f_ft_vw, "- NOMBRE DEL FICHERO: %s\n", name_in);
555         fprintf(f_ft_vw, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
556         fprintf(f_ft_vw, "- CANTIDAD DE PERIODOS: %d\n", period);
557         fprintf(f_ft_vw, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
558         fprintf(f_ft_vw, "- USO DE PROCESADORES: %d\n", core);
559         fprintf(f_ft_vw, "- PARAMETRO MAS EFICIENTE: %Lf, %Lf, %Lf, %Lf, %Lf, %Lf y %Lf\n",
                pos_ft_tw[0], pos_ft_tw[1], pos_ft_tw[2], pos_ft_tw[3], pos_ft_tw[4], pos_ft_tw[5],
                pos_ft_tw[6]);
560         fprintf(f_ft_vw, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
561         fprintf(f_ft_vw, "- VALOR DE MAPE EN VW: %Lf \n\n\n", ref_vw);
562         fprintf(f_ft_vw, "\n");
563         fprintf(f_ft_vw, "\n");
564         for (int i = 0 ; i < frec ; i++){
565                         fprintf(f_ft_vw, "%Lf\n",  global_ft_vw[i][best]);
566         }
567         if(fclose(f_ft_vw) != 0){printf("\n no se ha podido cerrar el archivo que contiene el
                pronostico de la ventana de entrenamiento \n"); exit(1);}
568     }
569     ////////////////////////////////////////////////////////////////////////////
570     //FUNCION PRINCIPAL
571     int main(int arcg, char *argv[])
572     {
573     //******VARIABLES
574         name_ejecutable = argv[0];
575         name_in = argv[1];
576         frec = atoi(argv[2]);
577         period = atoi(argv[3]);
578         steps = atoi(argv[4]);
579         core = atoi(argv[5]);
580         datos = (frec*period);
581     //******FUNCIONES
582         printf("\n");
583         printf("***** METODO DE PRONOSTICO WMA PARA SERIES DE TIEMPO *****\n");
584         printf("\n");
585         printf(" INFORMACION DEL METODO WMA:\n");
586         printf("- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
587         printf("- NOMBRE DEL FICHERO: %s\n", name_in);
588         printf("- FRECUENCIA DE LOS DATOS: %d\n", frec);
589         printf("- CANTIDAD DE PERIODOS: %d\n", period);
590         printf("- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
591         printf("- USO DE PROCESADORES: %d\n", core);
592         printf("\n Reservando Memoria...\n");
593         definition_cant_parameters();
```

```
594       allocate_memory();
595       printf("\n Leyendo serie de tiempo...\n");
596       read_data();
597       printf("\n Leyendo los pesos...\n");
598       weight();
599       printf("\n Ordenando serie de tiempo...\n");
600       tw_vw();
601   //******SELECCION DE PESOS
602       if (period > 3){
603               if(values_weight > core){
604                       printf("\n Inicio del programa...\n");
605                       distribute();
606                       printf("\n Salida de archivos...\n");
607                       output_file();
608               } else{
609                       printf("\nLa cantidad de procesadores no puede ser mayor al numero de
                                parametros ingresados\n");
610               }
611       } else{
612               printf("\n EL NUMERO DE PERIODOS ES MENOR A 3, NO ES POSIBLE APLICAR EL METODO DE
                        PROMEDIOS MOVILES PONDERADOS \n");
613       }
614       printf("\n Liberar memoria...\n");
615       free_memory();
616       return 0;
617   }
```

Apéndice E

Código en C método ES

A continuación, se presenta el código de programación en C que realiza la selección del parámetro óptimo del método de pronóstico de series de tiempo de ES usando el cómputo en paralelo basado en hilos.

```
1  ////////////////////////////////////////////////////////////////////////////
2  // UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
3  // DIVISION DE ESTUDIOS DE POSGRADO DE INGENIERIA ELECTRICA
4  // AREA: SISTEMAS ELECTRICOS DE POTENCIA
5  // METODO DE PRONOSTICO: SUAVIZACION EXPONENCIAL
6  // PROGRAMADOR : JESUS DE LA TORRE BUCIO
7  // PARALELO
8  ////////////////////////////////////////////////////////////////////////////
9
10 ////////////////////////////////////////////////////////////////////////////
11 //LIBRERIAS
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <pthread.h>
16
17 ////////////////////////////////////////////////////////////////////////////
18 //VARIABLES GLOBALES
19 //VARIABLES DEL MAIN
20 long double datos, steps;
21 //VARIABLES DEL MAPE
22 long double ref_tw, ref_vw, pos_ft_tw;
23 // DIMENSIONES DE LA SERIE DE TIEMPO
24 int period, frec, params = 11, params_count, core, best;
25 // MEMORIA DINAMICA
26 long double *st, **time_series, **tw, *vw, *MAPE_tw, *MAPE_vw, *best_ft_vw, *parameters, **
       global_ft_vw, *parameters;
27 //ARCHIVOS DE ENTRADA Y SALIDA
28 FILE *f_in, *f_ts, *f_ft_vw, *f_param, *f_parameters;
29 char *name_in, *name_ejecutable;
30 // MUTEX
31 pthread_mutex_t mutex;
32
33 ////////////////////////////////////////////////////////////////////////////
34 // ESTRUCTURA
35 typedef struct{
36     int ini;
37     int fin;
38 }args_threads;
39 ////////////////////////////////////////////////////////////////////////////
40 //FUNCIONES
```

```
41  void allocate_memory(void){// SE GENERA LA MEMORIA DINAMICA DE LAS VARIABLES GLOBALES
42      st = malloc(datos * sizeof(long double));
43      time_series = malloc(frec * sizeof(long double*));
44      tw = malloc(frec * sizeof(long double*));
45      vw = malloc(frec * sizeof(long double));
46      MAPE_tw = malloc(params_count * sizeof(long double));
47      MAPE_vw = malloc(params_count * sizeof(long double));
48      parameters = malloc(params_count * sizeof(long double));
49      global_ft_vw = malloc(frec * sizeof(long double*));
50      if (st == NULL){printf("No se guardo espacio de memoria de st \n"); exit(1);}
51      if (time_series == NULL){printf("No se guardo espacio de memoria al array \"timeseries\"
            \n"); exit(1);}
52      if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n"); exit(1);}
53      if (vw == NULL){printf("No se guardo espacio de memoria al array \"vw\" \n"); exit(1);}
54      if (MAPE_tw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_tw\" \n");
            exit(1);}
55      if (MAPE_vw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_vw\" \n");
            exit(1);}
56      if (parameters == NULL){printf("No se guardo espacio de memoria al array \"parameters\" \
            n"); exit(1);}
57      if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"global_ft_vw
            \" \n"); exit(1);}
58      for (int i = 0; i < frec; i++){
59              time_series[i] = malloc(period * sizeof(long double));
60              tw[i] = malloc((period-1) * sizeof(long double));
61              global_ft_vw[i] = malloc(params_count * sizeof(long double));
62              if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n");
                    exit(1);}
63              if (time_series == NULL){printf("No se guardo espacio de memoria al array \"
                    timeseries\" \n"); exit(1);}
64              if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    global_ft_vw\" \n"); exit(1);}
65      }
66  }
67  void read_data(void){// SE ESTABLECE TODA LA SERIE DE TIEMPO EN UN ARRAY LLAMADO "st"
68      f_in = fopen(name_in, "r");
69      if(f_in == NULL){ printf("\n Error al abrir el archivo de lectura de datos llamado %s \n"
            , name_in); exit(1);}
70      for (int i = 0; i < datos ; i++){
71              fscanf(f_in, "%Lf", &st[i]);
72      }
73      if(fclose(f_in) != 0){printf("\n no se ha podido cerrar el fichero %s \n", name_in); exit
            (1);}
74  }
75  void parametros(void){
76      for (int n = 0; n < params_count; n++){
77              parameters[n] = n*steps;
78      }
79  }
80  void tw_vw(void){// SE ESTABLECE LA SERIE DE TIEMPO DE LA VENTANA DE VALIDACION Y
        ENTRENAMIENTO
81      int cont, pos = 0;
82      for(int i = 0; i < period ; i++){
83              cont = 0;
84              for(int j = 0; j < frec ; j++){
85                      time_series[j][i] = st[j + pos];
86                      cont = cont + 1;
87                      if(i < period-1){
88                              tw[j][i] = time_series[j][i];
89                      } else{
90                              vw[j] = time_series[j][i];
91                      }
92              }
93              pos = pos + cont;
94      }
95  }
```

```
96  void free_memory(void){// SE LIBERA MEMORIA DINAMICA
97      for (int i = 0 ; i < frec ; i++){
98              free(time_series[i]);
99              free(tw[i]);
100             free(global_ft_vw[i]);
101     }
102     free(st);
103     free(time_series);
104     free(tw);
105     free(vw);
106     free(parameters);
107     free(MAPE_vw);
108     free(MAPE_tw);
109     free(global_ft_vw);
110 }
111 void *ES(void *rank){
112     if (period > 2){
113             //****** ESTABLECE LOS PARAMETROS DE INICIO Y FINAL DE CADA HILO
114             args_threads *thread_data = (args_threads*)rank;
115             int start = thread_data -> ini;
116             int end = thread_data -> fin;
117             //printf(" \n inicio del hilo %d fin del hilo %d \n", start,end);
118             for(int alpha = start; alpha < end; alpha++){
119                     long double **ft_tw, *ft_vw;
120                     // MEMORIA DINAMICA PRONOSTICO
121                     ft_tw = malloc(frec * sizeof(long double*));
122                     ft_vw = malloc(frec * sizeof(long double));
123                     if (ft_tw == NULL){printf("No se guardo espacio de memoria al array \"
                            ft_tw\" \n"); exit(1);}
124                     if (ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                            ft_vw\" \n"); exit(1);}
125                     for (int i = 0; i < frec; i++){
126                             ft_tw[i] = malloc((period-1) * sizeof(long double));
127                             if (ft_tw == NULL){printf("No se guardo espacio de memoria al
                                    array \"ft_tw\" \n"); exit(1);}
128                     }
129                     // MEMORIA DINAMICA MAPE VENTANA DE ENTRENAMIENTO
130                     long double *error_rest_tw, *error_abs_tw, *error_div_tw;
131                     long double cont_tw, cont1, sum_tw;
132                     int pos_tw;
133                     error_rest_tw = malloc((frec*(period-2)) * sizeof(long double));
134                     error_abs_tw = malloc((frec*(period-2)) * sizeof(long double));
135                     error_div_tw = malloc((frec*(period-2)) * sizeof(long double));
136                     if (error_rest_tw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_rest_tw\" \n"); exit(1);}
137                     if (error_abs_tw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_abs_tw\" \n"); exit(1);}
138                     if (error_div_tw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_div_tw\" \n"); exit(1);}
139                     // MEMORIA DINAMICA
140                     long double *error_rest_vw, *error_abs_vw, *error_div_vw;
141                     long double cont_vw, sum_vw;
142                     error_rest_vw = malloc((frec) * sizeof(long double));
143                     error_abs_vw = malloc((frec) * sizeof(long double));
144                     error_div_vw = malloc((frec) * sizeof(long double));
145                     if (error_rest_vw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_rest_vw\" \n"); exit(1);}
146                     if (error_abs_vw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_abs_vw\" \n"); exit(1);}
147                     if (error_div_vw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_div_vw\" \n"); exit(1);}
148                     // PRONOSTICO VENTANA DE ENTRENAMIENTO
149                     for (int i = 0; i < frec; i++){
150                             ft_tw[i][0] = 0;
151                             ft_tw[i][1] = tw[i][0];
152                             for (int j = 2 ; j < period-1; j++){
```

```
153                                     ft_tw[i][j] = (parameters[alpha] * tw[i][j - 1])+((1 -
                                            parameters[alpha]) * ft_tw[i][j - 1]);
154                             }
155                     }
156                     // PRONOSTICO VENTANA DE VALIDACION
157                     for (int i = 0; i < frec ; i++){
158                             ft_vw[i]= (parameters[alpha] * tw[i][period - 2])+((1 -
                                    parameters[alpha]) * ft_tw[i][period - 2]);
159                     }
160             //****** MAPE
161                     // VENTANA DE ENTRENAMIENTO
162                     cont_tw = 0; sum_tw = 0; pos_tw = 0;
163                     //printf("\ncon alpha = %d\n",alpha);
164                     for(int i = 1; i < period-1 ; i++){
165                             cont1 = 0;
166                             for(int j = 0; j < frec ; j++){
167                                     // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
168                                     error_rest_tw[pos_tw+j] = tw[j][i] - ft_tw[j][i];
169                                     // VALOR ABSOLUTO
170                                     if(error_rest_tw[pos_tw+j] < 0){
171                                             error_abs_tw[pos_tw+j] = error_rest_tw[pos_tw+j]
                                                    * -1;
172                                     } else{
173                                             error_abs_tw[pos_tw+j] = error_rest_tw[pos_tw+j];
174                                     }
175                                     //printf("%Lf \n", error_abs_tw[pos_tw+j]);
176                                     // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
177                                     error_div_tw[pos_tw+j] = (error_abs_tw[pos_tw+j] / tw[j][
                                            i])*100;
178                                     // SE REALIZA EL PROMEDIO
179                                     cont_tw = cont_tw + 1;
180                                     sum_tw = sum_tw + error_div_tw[pos_tw+j];
181                                     cont1 = cont1 +1;
182                             }
183                             pos_tw = pos_tw + cont1;
184                     }
185                     // VENTANA DE VALIDACION
186                     cont_vw = 0; sum_vw = 0;
187                     for(int i = 0; i < frec ; i++){
188                             // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
189                             error_rest_vw[i] = vw[i] - ft_vw[i];
190                             // VALOR ABSOLUTO
191                             if(error_rest_vw[i] < 0){
192                                     error_abs_vw[i] = error_rest_vw[i] * -1;
193                             } else{
194                                     error_abs_vw[i] = error_rest_vw[i];
195                             }
196                             // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
197                             error_div_vw[i] = (error_abs_vw[i] / vw[i])*100;
198                             // SE REALIZA EL PROMEDIO
199                             cont_vw = cont_vw + 1;
200                             sum_vw = sum_vw + error_div_vw[i];
201                     }
202                     // SELECCION DE LOS MEJORES RESULTADOS
203                     pthread_mutex_lock(&mutex);//BLOQUEA EL DATO CON EL MUTEX
204                     MAPE_tw[alpha] = sum_tw/cont_tw;// SE CALCULA EL MAPE VENTANA DE
                            ENTRENAMIENTO
205                     MAPE_vw[alpha] = sum_vw/cont_vw;// SE CALCULA EL MAPE VENTANA DE
                            VALIDACION
206                     for (int i = 0; i < frec; i++){
207                             global_ft_vw[i][alpha] = ft_vw[i];
208                     }
209                     if(alpha > 0){
210                             if(MAPE_vw[alpha]< ref_vw ){
211                                     ref_tw = MAPE_tw[alpha];
212                                     pos_ft_tw = parameters[alpha];
```

```
213                                     ref_vw = MAPE_vw[alpha];
214                                     best = alpha;
215                             }
216                     } else{
217                             ref_tw = MAPE_tw[alpha];
218                             pos_ft_tw = parameters[alpha];
219                             ref_vw = MAPE_vw[alpha];
220                             best = alpha;
221                     }
222                     pthread_mutex_unlock(&mutex);//DESBLOQUEA EL DATO CON EL MUTEX
223                     // LIBERACION DE MEMORIA DINAMICA
224                     free(error_rest_vw);
225                     free(error_abs_vw);
226                     free(error_div_vw);
227                     free(error_rest_tw);
228                     free(error_abs_tw);
229                     free(error_div_tw);
230                     for (int i = 0 ; i < frec ; i++){
231                             free(ft_tw[i]);
232                     }
233                     free(ft_tw);
234                     free(ft_vw);
235             }
236     } else{
237             printf("\n La cantidad de periodos debe ser mayor a 2 \n");
238     }
239     pthread_exit(NULL);
240 }
241 void distribute(void){
242     // ALMACENA MEMORIA EN VECTORES DINAMICOS
243     int *amount_data = malloc(core * sizeof(int));// ALMACENA CANTIDAD DE DATOS A LEER
244     args_threads *data_sent = malloc(core * sizeof(args_threads));// ARGUMENTOS PARA LOS
            HILOS
245     pthread_t *th = malloc(core * sizeof(pthread_t));// VARIABLE DONDE ALMACENA LA CANTIDAD
            DE HILOS
246     if (amount_data == NULL){printf("No se guardo espacio de memoria al array \"amount_data\"
            \n"); exit(1);}
247     if (data_sent == NULL){printf("No se guardo espacio de memoria al array \"data_sent\" \n"
            ); exit(1);}
248     if (th == NULL){printf("No se guardo espacio de memoria al array \"th\" \n"); exit(1);}
249     // DIVISION
250     div_t dist = div(params_count, core);
251     //printf("\n El valor entero es: %d \n", x.quot);
252     //printf("\n El residuo es: %d \n", x.rem);
253     // CANTIDAD DE DATOS A LEER POR CADA HILO
254     for (int i = 0; i < core; i++){
255             amount_data[i] = dist.quot;//numero entero
256     }
257     for (int i = 0; i < dist.rem; i++){
258             amount_data[i] = amount_data[i] + 1;//residuo
259     }
260     /*printf("\n Datos de finales de cada nucleo \n");
261     for (int i = 0 ; i < core ; i++){
262             printf("%d \n", amount_data[i]);
263     }*/
264     // DECLARA LOS PARAMETROS EN EL VECTOR DE ARGUMENTOS
265     for (int i = 0; i < core; i++){
266             if(i == 0){
267                     data_sent[i].ini = 0;
268                     data_sent[i].fin = amount_data[i];
269             } else{
270                     data_sent[i].ini = data_sent[i - 1].fin;
271                     data_sent[i].fin = data_sent[i].ini + amount_data[i];
272             }
273     }
274     // CREACION DE HILOS
```

```
275     pthread_mutex_init(&mutex,NULL); //SE INICIALIZA EL MUTEX
276     for (int i = 0; i < core; i++){
277             pthread_create(&th[i], NULL, ES, &data_sent[i]);
278     }
279     for (int i= 0; i < core; i++){
280             pthread_join(th[i], NULL);
281     }
282     pthread_mutex_destroy(&mutex); //SE FINALIZA EL MUTEX
283     // LIBERAR MEMORIA
284     free(amount_data);
285     free(data_sent);
286     free(th);
287 }
288 void output_file(void){
289 //***** FICHERO SERIE DE TIEMPO
290     f_ts = fopen("ES_time_series.txt", "w");
291     if(f_ts == NULL){ printf("\n Error al abrir el archivo que va a contener la serie de
          tiempo \n"); exit(1);}
292     fprintf(f_ts, "\n");
293     fprintf(f_ts, "***** METODO DE PRONOSTICO ES PARA SERIES DE TIEMPO *****\n");
294     fprintf(f_ts, "\n");
295     fprintf(f_ts, " INFORMACION DEL METODO ES:\n");
296     fprintf(f_ts, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
297     fprintf(f_ts, "- NOMBRE DEL FICHERO: %s\n", name_in);
298     fprintf(f_ts, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
299     fprintf(f_ts, "- CANTIDAD DE PERIODOS: %d\n", period);
300     fprintf(f_ts, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
301     fprintf(f_ts, "- USO DE PROCESADORES: %d\n", core);
302     fprintf(f_ts, "- PARAMETRO MAS EFICIENTE: %Lf\n", pos_ft_tw);
303     fprintf(f_ts, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
304     fprintf(f_ts, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
305     fprintf(f_ts, "\n");
306     fprintf(f_ts, "\n");
307     fprintf(f_ts, " SERIE DE TIEMPO\n");
308     fprintf(f_ts, "\n");
309     fprintf(f_ts, "\n");
310     for (int i = 0 ; i < frec ; i++){
311             for(int j = 0 ; j < period ; j++){
312                     fprintf(f_ts, "%Lf ",  time_series[i][j]);
313             }
314             fprintf(f_ts, "\n");
315     }
316     fprintf(f_ts, "\n");
317     fprintf(f_ts, "\n");
318     fprintf(f_ts, " VENTANA DE ENTRENAMIENTO\n");
319     fprintf(f_ts, "\n");
320     fprintf(f_ts, "\n");
321     for (int i = 0 ; i < frec ; i++){
322             for(int j = 0 ; j < period-1 ; j++){
323                     fprintf(f_ts, "%Lf ",  tw[i][j]);
324             }
325             fprintf(f_ts, "\n");
326     }
327     fprintf(f_ts, "\n");
328     fprintf(f_ts, "\n");
329     fprintf(f_ts, " VENTANA DE VALIDACION\n");
330     fprintf(f_ts, "\n");
331     fprintf(f_ts, "\n");
332     for(int j = 0 ; j < frec ; j++){
333                     fprintf(f_ts, "%Lf \n",vw[j]);
334     }
335     if(fclose(f_ts) != 0){printf("\n no se ha podido cerrar el archivo que contiene la serie
          de tiempo \n"); exit(1);}
336 //***** FICHERO PARAMETRO
337     f_param = fopen("ES_parameters.txt", "w");
338     if(f_param == NULL){ printf("\n Error al abrir el archivo que va a contener los
```

```
            parametros␣\n"); exit(1);}
339         fprintf(f_param, "\n");
340         fprintf(f_param, "*****␣METODO␣DE␣PRONOSTICO␣ES␣PARA␣SERIES␣DE␣TIEMPO␣*****\n");
341         fprintf(f_param, "\n");
342         fprintf(f_param, "␣INFORMACION␣DEL␣METODO␣ES:\n");
343         fprintf(f_param, "␣NOMBRE␣DEL␣EJECUTABLE:␣%s\n", name_ejecutable);
344         fprintf(f_param, "␣NOMBRE␣DEL␣FICHERO:␣%s\n", name_in);
345         fprintf(f_param, "␣FRECUENCIA␣DE␣LOS␣DATOS:␣%d\n", frec);
346         fprintf(f_param, "␣CANTIDAD␣DE␣PERIODOS:␣%d\n", period);
347         fprintf(f_param, "␣TOTAL␣DE␣DATOS␣DE␣LA␣SERIE␣DE␣TIEMPO:␣%Lf\n", datos);
348         fprintf(f_param, "␣USO␣DE␣PROCESADORES:␣%d\n", core);
349         fprintf(f_param, "␣PARAMETRO␣MAS␣EFICIENTE:␣%Lf\n", pos_ft_tw);
350         fprintf(f_param, "␣VALOR␣DE␣MAPE␣EN␣TW:␣%Lf\n", ref_tw);
351         fprintf(f_param, "␣VALOR␣DE␣MAPE␣EN␣VW:␣%Lf\n", ref_vw);
352         fprintf(f_param, "\n");
353         fprintf(f_param, "\n");
354         fprintf(f_param, "\nPARAMETRO␣␣␣MAPE␣ENTRENAMIENTO␣␣␣MAPE␣VALIDACION␣\n");
355         for(int i = 0;  i < params_count; i++){
356                 fprintf(f_param, "%Lf␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%Lf␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣%Lf\n", parameters[i],
                        MAPE_tw[i],MAPE_vw[i]);
357         }
358         if(fclose(f_param) != 0){printf("\n␣no␣se␣ha␣podido␣cerrar␣el␣archivo␣que␣contiene␣los␣
                parametros␣\n"); exit(1);}
359     //***** FICHERO PRONOSTICO DE LA VENTANA DE VALIDACION
360         f_ft_vw = fopen("ES_forecasting.txt", "w");
361         if(f_ft_vw == NULL){ printf("\n␣Error␣al␣abrir␣el␣archivo␣que␣contienen␣el␣pronostico␣de␣
                la␣ventana␣de␣entrenamiento␣\n"); exit(1);}
362         fprintf(f_ft_vw, "\n");
363         fprintf(f_ft_vw, "*****␣METODO␣DE␣PRONOSTICO␣ES␣PARA␣SERIES␣DE␣TIEMPO␣*****\n");
364         fprintf(f_ft_vw, "\n");
365         fprintf(f_ft_vw, "␣INFORMACION␣DEL␣METODO␣ES:\n");
366         fprintf(f_ft_vw, "␣NOMBRE␣DEL␣EJECUTABLE:␣%s\n", name_ejecutable);
367         fprintf(f_ft_vw, "␣NOMBRE␣DEL␣FICHERO:␣%s\n", name_in);
368         fprintf(f_ft_vw, "␣FRECUENCIA␣DE␣LOS␣DATOS:␣%d\n", frec);
369         fprintf(f_ft_vw, "␣CANTIDAD␣DE␣PERIODOS:␣%d\n", period);
370         fprintf(f_ft_vw, "␣TOTAL␣DE␣DATOS␣DE␣LA␣SERIE␣DE␣TIEMPO:␣%Lf\n", datos);
371         fprintf(f_ft_vw, "␣USO␣DE␣PROCESADORES:␣%d\n", core);
372         fprintf(f_ft_vw, "␣PARAMETRO␣MAS␣EFICIENTE:␣%Lf\n", pos_ft_tw);
373         fprintf(f_ft_vw, "␣VALOR␣DE␣MAPE␣EN␣TW:␣%Lf\n", ref_tw);
374         fprintf(f_ft_vw, "␣VALOR␣DE␣MAPE␣EN␣VW:␣%Lf\n", ref_vw);
375         fprintf(f_ft_vw, "\n");
376         fprintf(f_ft_vw, "\n");
377         for (int i = 0 ; i < frec ; i++){
378                 fprintf(f_ft_vw, "%Lf\n",  global_ft_vw[i][best]);
379         }
380         if(fclose(f_ft_vw) != 0){printf("\n␣no␣se␣ha␣podido␣cerrar␣el␣archivo␣que␣contiene␣el␣
                pronostico␣de␣la␣ventana␣de␣entrenamiento␣\n"); exit(1);}
381     }
382     ////////////////////////////////////////////////////////////////////////////
383     // Funcion principal
384     int main(int arcg, char *argv[]){
385     //******VARIABLES
386         long double param_ope;
387         name_ejecutable = argv[0];
388         name_in = argv[1];
389         frec = atoi(argv[2]);
390         period = atoi(argv[3]);
391         steps = atof(argv[4]);
392         core = atoi(argv[5]);
393         datos = (frec*period);
394         param_ope = 1/steps;
395         params_count = param_ope + 2;
396     //******FUNCIONES
397         printf("\n");
398         printf("*****␣METODO␣DE␣PRONOSTICO␣ES␣PARA␣SERIES␣DE␣TIEMPO␣*****\n");
399         printf("\n");
```

```
400         printf("␣INFORMACION␣DEL␣METODO␣ES:\n");
401         printf("−␣NOMBRE␣DEL␣EJECUTABLE:␣%s\n", name_ejecutable);
402         printf("−␣NOMBRE␣DEL␣FICHERO:␣%s\n", name_in);
403         printf("−␣FRECUENCIA␣DE␣LOS␣DATOS:␣%d\n", frec);
404         printf("−␣CANTIDAD␣DE␣PERIODOS:␣%d\n", period);
405         printf("−␣TOTAL␣DE␣DATOS␣DE␣LA␣SERIE␣DE␣TIEMPO:␣%Lf\n", datos);
406         printf("−␣USO␣DE␣PROCESADORES:␣%d\n", core);
407         printf("\n␣Reservando␣Memoria...\n");
408         allocate_memory();
409         printf("\n␣Leyendo␣serie␣de␣tiempo...\n");
410         read_data();
411         printf("\n␣Leyendo␣los␣parametros...\n");
412         parametros();
413         printf("\n␣Ordenando␣serie␣de␣tiempo...\n");
414         tw_vw();
415         if (period > 3){
416                 printf("\n␣Inicio␣del␣programa...\n");
417                 distribute();
418                 printf("\n␣Salida␣de␣archivos...\n");
419                 output_file();
420         } else{
421                 printf("\n␣EL␣NUMERO␣DE␣PERIODOS␣ES␣MENOR␣A␣3,␣NO␣ES␣POSIBLE␣APLICAR␣EL␣METODO␣DE
                        ␣SUAVIZACION␣EXPONENCIAL␣\n");
422         }
423         printf("\n␣Liberar␣memoria...\n");
424         free_memory();
425         return 0;
426 }
```

Apéndice F

Código en C método LS

A continuación, se presenta el código de programación en C que realiza la selección del parámetro óptimo del método de pronóstico de series de tiempo de LS usando el cómputo en paralelo basado en hilos.

```
1  ////////////////////////////////////////////////////////////////////////////
2  // UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
3  // DIVISION DE ESTUDIOS DE POSGRADO DE INGENIERIA ELECTRICA
4  // AREA: SISTEMAS ELECTRICOS DE POTENCIA
5  // METODO DE PRONOSTICO: MINIMOS CUADRADOS
6  // PROGRAMADOR : JESUS DE LA TORRE BUCIO
7  // PARALELO
8  ////////////////////////////////////////////////////////////////////////////
9
10 ////////////////////////////////////////////////////////////////////////////
11 // LIBRERIAS
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <pthread.h>
16
17 ////////////////////////////////////////////////////////////////////////////
18 //VARIABLES GLOBALES
19 //VARIABLES DEL MAIN
20 long double datos;
21 //VARIABLES DEL MAPE
22 long double ref_tw, ref_vw;
23 // DIMENSIONES DE LA SERIE DE TIEMPO
24 int period, frec, orden_max, pos_ft_tw, core;
25 // MEMORIA DINAMICA
26 long double  *st, **time_series, **tw, *vw, *MAPE_tw, *MAPE_vw, **global_ft_vw;
27 //ARCHIVOS DE ENTRADA Y SALIDA
28 FILE *f_in, *f_ts, *f_ft_tw, *f_ft_vw, *f_param;
29 char *name_in, *name_ejecutable;
30 // MUTEX
31 pthread_mutex_t mutex;
32
33 ////////////////////////////////////////////////////////////////////////////
34 // ESTRUCTURA
35 typedef struct{
36     int ini;
37     int fin;
38 }args_threads;
39 ////////////////////////////////////////////////////////////////////////////
40 //FUNCIONES
41 void allocate_memory(void)
```

```
42  {// SE GENERA LA MEMORIA DINAMICA DE LAS VARIABLES GLOBALES
43      st = malloc(datos * sizeof(long double));
44      time_series = malloc(frec * sizeof(long double*));
45      tw = malloc(frec * sizeof(long double*));
46      vw = malloc(frec * sizeof(long double));
47      MAPE_tw = malloc((orden_max-1)  * sizeof(long double));
48      MAPE_vw = malloc((orden_max-1)  * sizeof(long double));
49      global_ft_vw = malloc(frec * sizeof(long double*));
50      if (st == NULL){printf("No se guardo espacio de memoria de st \n"); exit(1);}
51      if (time_series == NULL){printf("No se guardo espacio de memoria al array \"timeseries\"
            \n"); exit(1);}
52      if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n"); exit(1);}
53      if (vw == NULL){printf("No se guardo espacio de memoria al array \"vw\" \n"); exit(1);}
54      if (MAPE_tw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_tw\" \n");
            exit(1);}
55      if (MAPE_vw == NULL){printf("No se guardo espacio de memoria al array \"MAPE_vw\" \n");
            exit(1);}
56      if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"global_ft_vw
            \" \n"); exit(1);}
57      for (int i = 0; i < frec; i++){
58              tw[i] = malloc((period-1) * sizeof(long double));
59              time_series[i] = malloc(period * sizeof(long double));
60              global_ft_vw[i] = malloc(orden_max * sizeof(long double));
61              if (time_series == NULL){printf("No se guardo espacio de memoria al array \"
                    timeseries\" \n"); exit(1);}
62              if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n");
                    exit(1);}
63              if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    global_ft_vw\" \n"); exit(1);}
64      }
65  }
66  void read_data(void)
67  {// SE ESTABLECE TODA LA SERIE DE TIEMPO EN UN ARRAY LLAMADO "st"
68      f_in = fopen(name_in, "r");
69      if(f_in == NULL){ printf("\n Error al abrir el archivo de lectura de datos llamado %s \n"
            , name_in); exit(1);}
70      for (int i = 0; i < datos ; i++){
71              fscanf(f_in, "%Lf", &st[i]);
72              //printf("%Lf \n",st[i]);
73      }
74      if(fclose(f_in) != 0){printf("\n no se ha podido cerrar el fichero %s \n", name_in); exit
            (1);}
75
76  }
77  void tw_vw(void)
78  {// SE ESTABLECE LA SERIE DE TIEMPO DE LA VENTANA DE VALIDACION Y ENTRENAMIENTO
79      int cont, pos = 0;
80      for(int i = 0; i < period ; i++){
81              cont = 0;
82              for(int j = 0; j < frec ; j++){
83                      time_series[j][i] = st[j + pos];
84                      cont = cont + 1;
85                      if(i < period-1){
86                              tw[j][i] = time_series[j][i];
87                      } else{
88                              vw[j] = time_series[j][i];
89                      }
90              }
91              pos = pos + cont;
92      }
93  }
94  void free_memory(void)
95  {// SE LIBERA MEMORIA DINAMIECA
96      for (int i = 0 ; i < frec ; i++){
97              free(time_series[i]);
98              free(tw[i]);
```

```
99              free(global_ft_vw[i]);
100     }
101     free(global_ft_vw);
102     free(time_series);
103     free(tw);
104     free(st);
105     free(vw);
106     free(MAPE_vw);
107     free(MAPE_tw);
108 }
109
110 void *LS(void *rank)
111 {
112 //****** ESTABLECE LOS PARAMETROS DE INICIO Y FINAL DE CADA HILO
113     args_threads *thread_data = (args_threads*)rank;
114     int start = thread_data -> ini;
115     int end = thread_data -> fin;
116     //printf(" \n inicio del hilo %d fin del hilo %d \n", start,end);
117     for(int n = start; n < end; n++){
118             if (period-1 > n){
119                     // MEMORIA DINAMICA PARA EL RPONOSTICO
120                     long double **ft_tw, *ft_vw;
121                     ft_tw = malloc(frec * sizeof(long double*));
122                     ft_vw = malloc(frec * sizeof(long double));
123                     if (ft_tw == NULL){printf("No se guardo espacio de memoria al array \"
                            ft_tw\" \n"); exit(1);}
124                     if (ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                            ft_vw\" \n"); exit(1);}
125                     for(int i = 0; i < frec; i++){
126                             ft_tw[i] = malloc((period-1) * sizeof(long double));
127                             if (ft_tw == NULL){printf("No se guardo espacio de memoria al
                                    array \"ft_tw\" \n"); exit(1);}
128                     }
129                     // MEMORIA DINAMICA VENTANA DE VALIDACION
130                     long double *error_rest_vw, *error_abs_vw, *error_div_vw;
131                     long double cont_vw, sum_vw;
132                     error_rest_vw = malloc((frec) * sizeof(long double));
133                     error_abs_vw = malloc((frec) * sizeof(long double));
134                     error_div_vw = malloc((frec) * sizeof(long double));
135                     if (error_rest_vw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_rest_vw\" \n"); exit(1);}
136                     if (error_abs_vw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_abs_vw\" \n"); exit(1);}
137                     if (error_div_vw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_div_vw\" \n"); exit(1);}
138                     // MEMORIA DINAMICA VENTANA DE ENTRENAMIENTO
139                     long double *error_rest_tw, *error_abs_tw, *error_div_tw;
140                     long double cont_tw, cont1, sum_tw;
141                     int pos_tw;
142                     error_rest_tw = malloc((frec*(period-1)) * sizeof(long double));
143                     error_abs_tw = malloc((frec*(period-1)) * sizeof(long double));
144                     error_div_tw = malloc((frec*(period-1)) * sizeof(long double));
145                     if (error_rest_tw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_rest_tw\" \n"); exit(1);}
146                     if (error_abs_tw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_abs_tw\" \n"); exit(1);}
147                     if (error_div_tw == NULL){printf("No se guardo espacio de memoria al
                            array \"error_div_tw\" \n"); exit(1);}
148                     int g_g = n + n;
149                     int g_1 = n + 1;
150                     int sum_g = g_g + g_1;
151                     //***** METODO DE PRONOSTICO
152                     // VARIABLES
153                     int cont, pos;
154                     long double mul, sum_mc, pivote, aux, ft_pos, suma;
155                     long double x[period-1], pot_x[period-1][g_g], sum_x[g_g + 1], A[g_1][g_1
```

```
        ], y[period-1], pot_y[period-1][g_1], sum_y[g_1], b[g_1], I[g_1][g_1
        ], incognitas[g_1];
156                 for (int i = 0; i < period-1; i++){
157                         // SE FORMA EL VECTOR X
158                         x[i] = i + 1;
159                 //      printf("%Lf \n", x[i]);
160                         // SE FORMA LA PRIMERA COLUMNA DEL ARRAY pot_x
161                         pot_x[i][0] = x[i];
162                 }
163                 // SE FORMA COMPLETAMENTE EL ARRAY pot_x
164                 for (int j = 1; j < g_g; j++){// COLUMNAS
165                         for (int  k = 0; k < period-1; k++){// RENGLONES
166                                 mul = 1;
167                                 for (int l = 0; l < (j + 1) ; l++){
168                                         mul = mul * pot_x[k][0] ;
169                                 }
170                                 pot_x[k][j] = mul;
171                         }
172                 }
173                 // SE FORMA EL ARRAY sum_x
174                 sum_x[0] = period-1;
175                 for (int j = 1; j < g_g + 1 ; j++){
176                         sum_mc = 0;
177                         for (int k = 0; k < period-1; k++){
178                                 sum_mc =  sum_mc + pot_x[k][j - 1];
179                         }
180                         sum_x[j] = sum_mc;
181                 }
182                 // SE FORMA EL ARRAY A
183                 cont = 0;
184                 for (int j = 0; j < g_1 ; j++){
185                         for (int k = 0; k < g_1; k++){
186                                 A[j][k] = sum_x[k + cont];
187                         }
188                         cont++;
189                 }
190                 // SE INVIERTE A POR MEDIO DEL METODO DE GAUSS-JORDAN
191                 // FORMACION DE LA MATRIZ IDENTIDAD
192                 for (int j = 0 ; j < g_1 ; j++){
193                         for(int k = 0 ; k < g_1; k++){
194                                 if(j == k){
195                                         I[j][k] = 1;
196                                 } else{
197                                         I[j][k] = 0;
198                                 }
199                         }
200                 }
201                 // REDUCCION POR RENGLONES
202                 for (int j = 0 ; j < g_1; j++){
203                         // SE MUEVE A TRAVEZ DE LA DIAGONAL
204                         pivote = A[j][j];
205                         for (int l = 0 ; l < g_1 ; l++){
206                                 // DIVISION DEL PIVOTE PARA QUE TENGA VALOR DE 1
207                                 //Y APLICAR LA OPERACION DE TODA LA FILA
208                                 A[j][l] = A[j][l] / pivote;
209                                 I[j][l] = I[j][l] / pivote;
210                         }
211                         for (int k = 0; k < g_1 ; k++){
212                         // CONVIERTE EN CERO TODA LA COLUMNA A ESEPCION DEL PIVOTE
213                                 if (j != k){
214                                 // SIGNIFICA QUE EL VALOR ESTA FUERA DE LA DIAGONAL
215                                         aux = A[k][j];
216                                         for (int l = 0 ; l < g_1 ; l++){
217                                                 A[k][l] = A[k][l] - aux * A[j][l];
218                                                 I[k][l] = I[k][l] - aux * I[j][l];
219                                         }
```

```
220                                     }
221                             }
222                     }
223                     // SE COMIENZA LOS CICLOS PARA EL VECTOR DE INCOGNITAS
224                     for (int j = 0 ; j < frec; j++){
225                             for (int k = 0; k < period-1; k++){
226                                     // SE FORMA EL VECTOR Y
227                                     y[k] = tw [j][k];
228                                     // SE LLENA LA PRIMERA COLUMNA DE LA MATRIZ pot_y
229                                     pot_y[k][0] = y[k];
230                             }
231                             // SE TERMINA DE FORMAR LA MATRIZ pot_y
232                             pos = 0;
233                             for (int k = 1; k < g_1; k++){// COLUMNA
234                                     for (int l = 0; l < period-1 ; l++){// RENGLON
235                                             pot_y[l][k] = pot_x[l][pos] * pot_y[l][0];
236                                     }
237                                     pos++;
238                             }
239                             // SE APLICA LA SUMATORIA A CADA COLUMNA DE LA MATRIZ pot_y
240                             for (int k = 0; k < g_1 ; k++){
241                                     sum_mc = 0;
242                                     for (int l = 0; l < period-1; l++){
243                                             sum_mc =  sum_mc + pot_y[l][k];
244                                     }
245                                     sum_y[k] = sum_mc;
246                                     // CAMBIO DE VARIABLE
247                                     b[k] = sum_y[k];
248                             }
249                             // PRODUCTO DE A*b PARA OBTENER EL VECTOR DE INCOGNITAS
250                             for (int k = 0 ; k < g_1 ; k++){
251                                     suma = 0;
252                                     for (int l = 0; l < g_1; l++){
253                                             suma = suma + (I[k][l] * b[l]) ;
254                                     }
255                                     incognitas[k] = suma;
256                             }
257                             // PRONOSTICO VENTANA DE ENTRENAMIENTO
258                             ft_pos = 0;
259                             for(int k = 0; k < period -1; k++){
260                                     for(int l = 0; l < g_1; l++){
261                                             if(l == 0){
262                                                     ft_pos= incognitas[l];
263                                             } else{
264                                                     mul = 1;
265                                                     for(int m = 0; m < l ; m++){
266                                                             mul = mul * x[k];
267                                                     }
268                                                     ft_pos = ft_pos + (incognitas[l] * mul);
269                                             }
270                                     }
271                                     ft_tw[j][k] = ft_pos;
272                             }
273                             // PRONOSTICO VENTANA DE VALIDACION
274                             // se calcula el nuevo punto para el pronostico
275                             ft_pos = 0;
276                             for (int k = 0; k < g_1 ; k++){
277                                     if(k == 0){
278                                             ft_pos = ft_pos + incognitas[k];
279                                     } else{
280                                             mul = 1;
281                                             for (int l = 0; l < k; l++){
282                                                     mul = mul * (period);
283                                             }
284                                             ft_pos = ft_pos + (incognitas[k] * mul);
285                                     }
```

```
286                             }
287                             ft_vw[j] = ft_pos;
288                     }
289             //****** MAPE
290                     // VENTANA DE ENTRENAMIENTO
291                     // MEMORIA DINAMICA
292                     cont_tw = 0; sum_tw = 0; pos_tw = 0;
293                     for(int i = 0; i < period-1 ; i++){
294                             cont1 = 0;
295                             for(int j = 0; j < frec ; j++){
296                                     // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
297                                     error_rest_tw[pos_tw+j] = tw[j][i] - ft_tw[j][i];
298                                     // VALOR ABSOLUTO
299                                     if(error_rest_tw[pos_tw+j] < 0){
300                                             error_abs_tw[pos_tw+j] = error_rest_tw[pos_tw+j]
                                                    * -1;
301                                     } else{
302                                             error_abs_tw[pos_tw+j] = error_rest_tw[pos_tw+j];
303                                     }
304                                     //printf("%Lf \n", error_abs_tw[pos_tw+j]);
305                                     // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
306                                     error_div_tw[pos_tw+j] = (error_abs_tw[pos_tw+j] / tw[j][
                                            i])*100;
307                                     // SE REALIZA EL PROMEDIO
308                                     cont_tw = cont_tw + 1;
309                                     sum_tw = sum_tw + error_div_tw[pos_tw+j];
310                                     cont1 = cont1 +1;
311                             }
312                             pos_tw = pos_tw + cont1;
313                     }
314                     // VENTANA DE VALIDACION
315                     cont_vw = 0; sum_vw = 0;
316                     for(int i = 0; i < frec ; i++){
317                             error_rest_vw[i] = vw[i] - ft_vw[i];// SE REALIZA LA RESTA DEL
                                    VALOR REAL Y EL PRONOSTICADO
318                             if(error_rest_vw[i] < 0){// VALOR ABSOLUTO
319                                     error_abs_vw[i] = error_rest_vw[i] * -1;
320                             } else{
321                                     error_abs_vw[i] = error_rest_vw[i];
322                             }
323                             error_div_vw[i] = (error_abs_vw[i] / vw[i])*100;// LA DIVISION
                                    DEL VALOR ABSOLUTO Y EL VALOR REAL
324                             // SE REALIZA EL PROMEDIO
325                             cont_vw = cont_vw + 1;
326                             sum_vw = sum_vw + error_div_vw[i];
327                     }
328                     //printf("\n el MAPE de la ventana de validacion con alpha = %d es %Lf\n
                            ", alpha, MAPE_vw[alpha-1]);
329                     // SELECCION DE LOS MEJORES RESULTADOS
330                     pthread_mutex_lock(&mutex);//BLOQUEA EL DATO CON EL MUTEX
331                     MAPE_tw[n - 1] = sum_tw/cont_tw;// MAPE VENTANA DE ENTRENAMIENTO
332                     MAPE_vw[n - 1] = sum_vw/cont_vw;// MAPE VENTANA DE VALIDACION
333                     for (int i = 0; i < frec; i++){
334                             global_ft_vw[i][n-1] = ft_vw[i];
335                     }
336                     if(n > 1){
337                             if(MAPE_vw[n-1]< ref_vw ){
338                                     ref_tw = MAPE_tw[n-1];
339                                     ref_vw = MAPE_vw[n-1];
340                                     pos_ft_tw = n;
341                             }
342                     } else{
343                             ref_tw = MAPE_tw[n-1];
344                             pos_ft_tw = n;
345                             ref_vw = MAPE_vw[n-1];
346                     }
```

```
347                         pthread_mutex_unlock(&mutex);//DESBLOQUEA EL DATO CON EL MUTEX
348                         // LIBERACION DE MEMORIA DINAMICA
349                         free(error_rest_tw);
350                         free(error_abs_tw);
351                         free(error_div_tw);
352                         free(error_rest_vw);
353                         free(error_abs_vw);
354                         free(error_div_vw);
355                         for (int i = 0 ; i < frec ; i++){
356                         free(ft_tw[i]);
357                         }
358                         free(ft_tw);
359                         free(ft_vw);
360                 } else{
361                         printf("\n La cantidad de periodos de la serie de tiempo debe ser mayor a
                                 %d \n", n);
362                         MAPE_tw[n-1] = 0;
363                         MAPE_vw[n-1] = 0;// checar esto
364                 }
365         }
366         pthread_exit(NULL);
367 }
368 void distribute(void)
369 {
370         // ALMACENA MEMORIA EN VECTORES DINAMICOS
371         int *amount_data = malloc(core * sizeof(int));// ALMACENA CANTIDAD DE DATOS A LEER
372         args_threads *data_sent = malloc(core * sizeof(args_threads));// ARGUMENTOS PARA LOS
                HILOS
373         pthread_t *th = malloc(core * sizeof(pthread_t));// VARIABLE DONDE ALMACENA LA CANTIDAD
                DE HILOS
374         if (amount_data == NULL){printf("No se guardo espacio de memoria al array \"amount_data\"
                \n"); exit(1);}
375         if (data_sent == NULL){printf("No se guardo espacio de memoria al array \"data_sent\" \n"
                ); exit(1);}
376         if (th == NULL){printf("No se guardo espacio de memoria al array \"th\" \n"); exit(1);}
377         // DIVISION
378         div_t dist = div(orden_max, core);
379         //printf("\n El valor entero es: %d \n", x.quot);
380         //printf("\n El residuo es: %d \n", x.rem);
381         // CANTIDAD DE DATOS A LEER POR CADA HILO
382         for (int i = 0; i < core; i++){
383                 amount_data[i] = dist.quot;//numero entero
384         }
385         for (int i = 0; i < dist.rem; i++){
386                 amount_data[i] = amount_data[i] + 1;//residuo
387         }
388         /*printf("\n Datos de finales de cada nucleo \n");
389         for (int i = 0 ; i < core ; i++){
390                 printf("%d \n", amount_data[i]);
391         }*/
392         // DECLARA LOS PARAMETROS EN EL VECTOR DE ARGUMENTOS
393         for (int i = 0; i < core; i++){
394                 if(i == 0){
395                         data_sent[i].ini = 1;
396                         data_sent[i].fin = amount_data[i];
397                 } else{
398                         data_sent[i].ini = data_sent[i - 1].fin;
399                         data_sent[i].fin = data_sent[i].ini + amount_data[i];
400                 }
401         }
402         // CREACION DE HILOS
403         pthread_mutex_init(&mutex,NULL); //SE INICIALIZA EL MUTEX
404         for (int i = 0; i < core; i++){
405                 pthread_create(&th[i], NULL, LS, &data_sent[i]);
406         }
407         for (int i= 0; i < core; i++){
```

```
408             pthread_join(th[i], NULL);
409     }
410     pthread_mutex_destroy(&mutex); //SE INICIALIZA EL MUTEX
411     // LIBERAR MEMORIA
412     free(amount_data);
413     free(data_sent);
414     free(th);
415 }
416 void output_file(void)
417 {
418 //***** FICHERO SERIE DE TIEMPO
419     f_ts = fopen("LS_time_series.txt", "w");
420     if(f_ts == NULL){ printf("\n Error al abrir el archivo que va a contener la serie de
            tiempo \n"); exit(1);}
421     fprintf(f_ts, "\n");
422     fprintf(f_ts, "***** METODO DE PRONOSTICO LS PARA SERIES DE TIEMPO *****\n");
423     fprintf(f_ts, "\n");
424     fprintf(f_ts, " INFORMACION DEL METODO LS:\n");
425     fprintf(f_ts, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
426     fprintf(f_ts, "- NOMBRE DEL FICHERO: %s\n", name_in);
427     fprintf(f_ts, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
428     fprintf(f_ts, "- CANTIDAD DE PERIODOS: %d\n", period);
429     fprintf(f_ts, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
430     fprintf(f_ts, "- USO DE PROCESADORES: %d\n", core);
431     fprintf(f_ts, "- PARAMETRO MAS EFICIENTE: %d\n", pos_ft_tw);
432     fprintf(f_ts, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
433     fprintf(f_ts, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
434     fprintf(f_ts, "\n");
435     fprintf(f_ts, "\n");
436     fprintf(f_ts, " SERIE DE TIEMPO\n");
437     fprintf(f_ts, "\n");
438     fprintf(f_ts, "\n");
439     for (int i = 0 ; i < frec ; i++){
440             for(int j = 0 ; j < period ; j++){
441                     fprintf(f_ts, "%Lf ",  time_series[i][j]);
442             }
443             fprintf(f_ts, "\n");
444     }
445     fprintf(f_ts, "\n");
446     fprintf(f_ts, "\n");
447     fprintf(f_ts, " VENTANA DE ENTRENAMIENTO\n");
448     fprintf(f_ts, "\n");
449     fprintf(f_ts, "\n");
450     for (int i = 0 ; i < frec ; i++){
451             for(int j = 0 ; j < period-1 ; j++){
452                     fprintf(f_ts, "%Lf ",  tw[i][j]);
453             }
454             fprintf(f_ts, "\n");
455     }
456     fprintf(f_ts, "\n");
457     fprintf(f_ts, "\n");
458     fprintf(f_ts, " VENTANA DE VALIDACION\n");
459     fprintf(f_ts, "\n");
460     fprintf(f_ts, "\n");
461     for(int j = 0 ; j < frec ; j++){
462                     fprintf(f_ts, "%Lf \n",vw[j]);
463     }
464     if(fclose(f_ts) != 0){printf("\n no se ha podido cerrar el archivo que contiene la serie
            de tiempo \n"); exit(1);}
465 //***** FICHERO PARAMETRO
466     f_param = fopen("LS_parameters.txt", "w");
467     if(f_param == NULL){ printf("\n Error al abrir el archivo que va a contener los
            parametros \n"); exit(1);}
468     fprintf(f_param, "\n");
469     fprintf(f_param, "***** METODO DE PRONOSTICO LS PARA SERIES DE TIEMPO *****\n");
470     fprintf(f_param, "\n");
```

```
471       fprintf(f_param, "␣INFORMACION␣DEL␣METODO␣LS:\n");
472       fprintf(f_param, "−␣NOMBRE␣DEL␣EJECUTABLE:␣%s\n", name_ejecutable);
473       fprintf(f_param, "−␣NOMBRE␣DEL␣FICHERO:␣%s\n", name_in);
474       fprintf(f_param, "−␣FRECUENCIA␣DE␣LOS␣DATOS:␣%d\n", frec);
475       fprintf(f_param, "−␣CANTIDAD␣DE␣PERIODOS:␣%d\n", period);
476       fprintf(f_param, "−␣TOTAL␣DE␣DATOS␣DE␣LA␣SERIE␣DE␣TIEMPO:␣%Lf\n", datos);
477       fprintf(f_param, "−␣USO␣DE␣PROCESADORES:␣%d\n", core);
478       fprintf(f_param, "−␣PARAMETRO␣MAS␣EFICIENTE:␣%d\n", pos_ft_tw);
479       fprintf(f_param, "−␣VALOR␣DE␣MAPE␣EN␣TW:␣%Lf\n", ref_tw);
480       fprintf(f_param, "−␣VALOR␣DE␣MAPE␣EN␣VW:␣%Lf\n", ref_vw);
481       fprintf(f_param, "\n");
482       fprintf(f_param, "\n");
483       fprintf(f_param, "\n");
484       fprintf(f_param, "\nPARAMETRO␣␣␣MAPE␣ENTRENAMIENTO␣␣␣MAPE␣VALIDACION␣\n");
485       for (int i = 1 ; i < orden_max  ; i++){
486                 fprintf(f_param, "␣␣␣␣%d␣␣␣␣␣", i);
487                 fprintf(f_param, "␣␣%Lf␣␣␣",MAPE_tw[i-1]);
488                 fprintf(f_param, "␣␣␣␣␣␣␣␣␣%Lf␣␣␣",MAPE_vw[i-1]);
489                 fprintf(f_param, "\n");
490       }
491       if(fclose(f_param) != 0){printf("\n␣no␣se␣ha␣podido␣cerrar␣el␣archivo␣que␣contiene␣los␣
              parametros␣\n"); exit(1);}
492   //***** FICHERO PRONOSTICO DE LA VENTANA DE VALIDACION
493       f_ft_vw = fopen("LS_forecasting.txt", "w");
494       if(f_ft_vw == NULL){ printf("\n␣Error␣al␣abrir␣el␣archivo␣que␣contienen␣el␣pronostico␣de␣
              la␣ventana␣de␣entrenamiento␣\n"); exit(1);}
495       fprintf(f_ft_vw, "\n");
496       fprintf(f_ft_vw, "*****␣METODO␣DE␣PRONOSTICO␣LS␣PARA␣SERIES␣DE␣TIEMPO␣*****\n");
497       fprintf(f_ft_vw, "\n");
498       fprintf(f_ft_vw, "␣INFORMACION␣DEL␣METODO␣LS:\n");
499       fprintf(f_ft_vw, "−␣NOMBRE␣DEL␣EJECUTABLE:␣%s\n", name_ejecutable);
500       fprintf(f_ft_vw, "−␣NOMBRE␣DEL␣FICHERO:␣%s\n", name_in);
501       fprintf(f_ft_vw, "−␣FRECUENCIA␣DE␣LOS␣DATOS:␣%d\n", frec);
502       fprintf(f_ft_vw, "−␣CANTIDAD␣DE␣PERIODOS:␣%d\n", period);
503       fprintf(f_ft_vw, "−␣TOTAL␣DE␣DATOS␣DE␣LA␣SERIE␣DE␣TIEMPO:␣%Lf\n", datos);
504       fprintf(f_ft_vw, "−␣USO␣DE␣PROCESADORES:␣%d\n", core);
505       fprintf(f_ft_vw, "−␣PARAMETRO␣MAS␣EFICIENTE:␣%d\n", pos_ft_tw);
506       fprintf(f_ft_vw, "−␣VALOR␣DE␣MAPE␣EN␣TW:␣%Lf\n", ref_tw);
507       fprintf(f_ft_vw, "−␣VALOR␣DE␣MAPE␣EN␣VW:␣%Lf\n", ref_vw);
508       fprintf(f_ft_vw, "\n");
509       fprintf(f_ft_vw, "\n");
510       fprintf(f_ft_vw, "\n");
511       for (int i = 0 ; i < frec ; i++){
512                 fprintf(f_ft_vw, "%Lf\n",  global_ft_vw[i][pos_ft_tw-1]);
513       }
514       if(fclose(f_ft_vw) != 0){printf("\n␣no␣se␣ha␣podido␣cerrar␣el␣archivo␣que␣contiene␣el␣
              pronostico␣de␣la␣ventana␣de␣entrenamiento␣\n"); exit(1);}
515   }
516   ////////////////////////////////////////////////////////////////////////////
517   // FUNCION PRINCIPAL
518   int main(int arcg, char *argv[])
519   {
520   //****** VARIABLES
521       name_ejecutable = argv[0];
522       name_in = argv[1];
523       frec = atoi(argv[2]);
524       period = atoi(argv[3]);
525       orden_max = atoi(argv[4])+1;
526       core = atoi(argv[5]);
527       datos = (frec*period);
528   //***** FUNCIONES
529       printf("\n");
530       printf("*****␣METODO␣DE␣PRONOSTICO␣LS␣PARA␣SERIES␣DE␣TIEMPO␣*****\n");
531       printf("\n");
532       printf("␣INFORMACION␣DEL␣METODO␣MA:\n");
533       printf("−␣NOMBRE␣DEL␣EJECUTABLE:␣%s\n", name_ejecutable);
```

```
534         printf("- NOMBRE DEL FICHERO: %s\n", name_in);
535         printf("- FRECUENCIA DE LOS DATOS: %d\n", frec);
536         printf("- CANTIDAD DE PERIODOS: %d\n", period);
537         printf("- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %Lf\n", datos);
538         printf("- USO DE PROCESADORES: %d\n", core);
539         printf("\n Reservando Memoria...\n");
540         allocate_memory();
541         printf("\n Leyendo serie de tiempo...\n");
542         read_data();
543         printf("\n Ordenando serie de tiempo...\n");
544         tw_vw();
545         if(core <= orden_max){
546                 printf("\n Inicio del programa...\n");
547                 distribute();
548                 printf("\n Salida de archivos...\n");
549                 output_file();
550         } else{
551                 printf("\n La cantidad de parametros a procesar son: %d. Por lo tanto no se
                        permite elegir una cantidad mayor de procesadores \n",orden_max);
552         }
553         printf("\n Liberar memoria...\n");
554         free_memory();
555                 return 0;
556 }
```

Apéndice G

Código en C método MHW

A continuación, se presenta el código de programación en C que realiza la selección del parámetro óptimo del método de pronóstico de series de tiempo de MHW usando el cómputo en paralelo basado en hilos.

```
1  ////////////////////////////////////////////////////////////////////////////
2  // UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
3  // DIVISION DE ESTUDIOS DE POSGRADO DE INGENIERIA ELECTRICA
4  // AREA: SISTEMAS ELECTRICOS DE POTENCIA
5  // METODO DE PRONOSTICO: HOLT-WINTERS MULTIPLICATIVO
6  // PROGRAMADOR : JESUS DE LA TORRE BUCIO
7  // COMPUTO EN PARALELO
8  ////////////////////////////////////////////////////////////////////////////
9
10 ////////////////////////////////////////////////////////////////////////////
11 //LIBRERIAS
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <pthread.h>
16
17 ////////////////////////////////////////////////////////////////////////////
18 //VARIABLES GLOBALES
19 // DIMENSIONES DE LA SERIE DE TIEMPO
20 int period, frec, s, m, params_count, core, pos_global_ft_vw;
21 // MEMORIA DINAMICA
22 long double  *st, *tw, *vw, **parameters, *MAPE_tw, *MAPE_vw, **global_ft_vw;
23 //VARIABLES DEL MAPE
24 long double ref_tw, ref_vw, *pos_ft_tw, steps ;
25 //ARCHIVOS DE ENTRADA Y SALIDA
26 FILE *f_in, *f_ts, *f_param, *f_ft_vw;
27 char *name_in, *name_ejecutable;
28 // MUTEX
29 pthread_mutex_t mutex;
30
31 ////////////////////////////////////////////////////////////////////////////
32 // ESTRUCTURA
33 typedef struct{
34     int ini;
35     int fin;
36 }args_threads;
37
38 ////////////////////////////////////////////////////////////////////////////
39 //FUNCIONES
40 void allocate_memory(void){// SE GENERA LA MEMORIA DINAMICA DE LAS VARIABLES GLOBALES
41     st = malloc((frec*period) * sizeof(long double));
```

```
42      tw = malloc((frec*(period-1)) * sizeof(long double));
43      vw = malloc(frec * sizeof(long double));
44      MAPE_tw = malloc(params_count * sizeof(long double));
45      MAPE_vw = malloc(params_count * sizeof(long double));
46      global_ft_vw = malloc(frec * sizeof(long double*));
47      pos_ft_tw = malloc(3 * sizeof(long double));
48      if (st == NULL){printf("No se guardo espacio de memoria de st \n"); exit(1);}
49      if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n"); exit(1);}
50      if (vw == NULL){printf("No se guardo espacio de memoria al array \"vw\" \n"); exit(1);}
51      if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"global_ft_vw
            \" \n"); exit(1);}
52      if (pos_ft_tw == NULL){printf("No se guardo espacio de memoria al array \"pos_ft_tw\" \n
            "); exit(1);}
53      for (int i = 0; i < frec; i++){
54              global_ft_vw[i] = malloc(params_count * sizeof(long double));
55              if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    global_ft_vw\" \n"); exit(1);}
56      }
57      parameters = malloc(params_count * sizeof(long double*));
58      if (parameters == NULL){printf("No se guardo espacio de memoria al array \"parameters\" \
            n"); exit(1);}
59      for (int i = 0 ; i < params_count; i++){
60              parameters[i] = malloc(3 * sizeof(long double));
61              if (parameters == NULL){printf("No se guardo espacio de memoria al array \"
                    parameters\" \n"); exit(1);}
62      }
63  }
64  void read_data(void){// SE ESTABLECE TODA LA SERIE DE TIEMPO EN UN ARRAY LLAMADO "st"
65      f_in = fopen(name_in, "r");
66      if(f_in == NULL){ printf("\n Error al abrir el archivo de lectura de datos llamado %s \n"
            , name_in); exit(1);}
67      for (int i = 0; i < (frec*period) ; i++){
68              fscanf(f_in, "%Lf", &st[i]);
69      }
70      if(fclose(f_in) != 0){printf("\n no se ha podido cerrar el fichero %s \n", name_in); exit
            (1);}
71      /*for (int i = 0; i < (frec*period); i++){
72              printf("%Lf \n", st[i]);
73      }*/
74  }
75  void tw_vw(void){// SE ESTABLECE LA SERIE DE TIEMPO DE LA VENTANA DE VALIDACION Y
        ENTRENAMIENTO
76      int pos_tw = 0, pos_vw = 0;
77      for(int i = 0; i < (frec*period) ; i++){
78              if(i < ((period-1)*frec)){
79                      tw[pos_tw] = st[i];
80                      pos_tw = pos_tw + 1;
81              } else{
82                      vw[pos_vw] = st[i];
83                      pos_vw = pos_vw +1;
84              }
85      }
86      /*printf("\n serie de tiempo\n");
87      for(int i = 0; i < (frec*period); i++){
88              printf("%Lf \n", st[i]);
89      }
90      printf("\n Ventana de entrenamiento\n");
91      for(int i = 0; i < (frec*(period-1)); i++){
92              printf("%Lf \n", tw[i]);
93      }
94      printf("\n Ventana de validacion\n");
95      for(int i = 0; i < frec ; i++){
96              printf("%Lf \n", vw[i]);
97      }*/
98  }
99
```

```
100 void parametros(void){
101     long double phi, psi, ommega;
102     phi = 0, psi = 0, ommega = 0;
103     for (int i = 0; i < params_count; i++){
104             if ((phi < 1+steps) && (phi >= 0)){
105                     parameters[i][0] = phi;
106                     parameters[i][1] = psi;
107                     parameters[i][2] = ommega;
108                     phi = phi + steps;
109             } else{
110                     phi = 0;
111                     parameters[i][0] = phi;
112                     phi = phi + steps;
113                     psi = psi + steps;
114                     if((psi < 1+steps) && (psi >= 0)){
115                             parameters[i][1] = psi;
116                             parameters[i][2] = ommega;
117                     } else{
118                             psi = 0;
119                             parameters[i][1] = psi;
120                             ommega = ommega + steps;
121                             parameters[i][2] = ommega;
122                     }
123             }
124     }
125 }
126 void free_memory(void){// SE LIBERA MEMORIA DINAMIECA
127     for (int i = 0 ; i < params_count ; i++){
128             free(parameters[i]);
129     }
130     free(parameters);
131     free(tw);
132     free(st);
133     free(vw);
134     free(MAPE_tw);
135     free(MAPE_vw);
136     for (int i = 0 ; i < frec ; i++){
137             free(global_ft_vw[i]);
138     }
139     free(global_ft_vw);
140     free(pos_ft_tw);
141 }
142 void *MHW(void *rank){
143 //****** ESTABLECE LOS PARAMETROS DE INICIO Y FINAL DE CADA HILO
144     args_threads *thread_data = (args_threads*)rank;
145     int start = thread_data -> ini;
146     int end = thread_data -> fin;
147     //printf(" \n inicio del hilo %d fin del hilo %d \n", start,end);
148     //VALORES INICIALES lt, Bt, St
149     long double sum_Ls = 0, sum_Bs = 0, Ls=0, Bs=0, *Ss;
150     Ss = malloc((s) * sizeof(long double));
151     if (Ss == NULL){printf("No se guardo espacio de memoria al array \"Ss\" \n"); exit(1);}
152     long double s_1 = s;
153     for (int i = 0; i < s; i++){
154             sum_Ls = sum_Ls + tw[i];//NIVEL
155             //printf("%Lf \n", sum_Ls);
156             sum_Bs = sum_Bs + ((tw[i+(s)]-tw[i])/s_1);//TENDENCIA
157             //printf("%Lf \n", sum_Bs);
158     }
159     Ls = (1/s_1)*(sum_Ls); // VALOR INICIAL DEL NIVEL
160     Bs = (1/s_1)*sum_Bs; // VALOR INICIAL DE TENDENCIA
161     //printf("%Lf \n", Ls);
162     //printf("%Lf \n", Bs);
163     for (int i = 0; i < s; i++){
164             Ss[i] = tw[i]/Ls;//VALORES INICIALES DE ESTACIONALIDAD
165             //printf("%Lf \n", Ss[i]);
```

```
166     }
167     for (int n = start; n < end; n++){
168             // MEMORIA DINAMICA
169                     int dim_MAPE = (frec*(period-1)) - s;
170                     //METODO DE HOLT WINTERS
171                             long double *Lt, *Bt, *St;
172                             Lt = malloc((frec*(period-1)) * sizeof(long double));
173                             Bt = malloc((frec*(period-1)) * sizeof(long double));
174                             St = malloc((frec*(period-1)) * sizeof(long double));
175                             if (Bt == NULL){printf("No se guardo espacio de memoria al array
                                    \"Bt\" \n"); exit(1);}
176                             if (Lt == NULL){printf("No se guardo espacio de memoria al array
                                    \"Lt\" \n"); exit(1);}
177                             if (St == NULL){printf("No se guardo espacio de memoria al array
                                    \"St\" \n"); exit(1);}
178                     // PRONOSTICO VENTANA DE ENTRENAMIENTO Y VALIDACION
179                             long double *ft_tw, *ft_vw;
180                             ft_tw = malloc((frec*(period-1)) * sizeof(long double));
181                             ft_vw = malloc(frec * sizeof(long double));
182                             if (ft_tw == NULL){printf("No se guardo espacio de memoria al
                                    array \"ft_tw\" \n"); exit(1);}
183                             if (ft_vw == NULL){printf("No se guardo espacio de memoria al
                                    array \"ft_vw\" \n"); exit(1);}
184                     // MAPE VENTANA DE ENTRENAMIENTO
185                             long double *error_rest_tw, *error_abs_tw, *error_div_tw;
186                             long double cont_tw, sum_tw;
187                             error_rest_tw = malloc(dim_MAPE * sizeof(long double));
188                             error_abs_tw = malloc(dim_MAPE * sizeof(long double));
189                             error_div_tw = malloc(dim_MAPE * sizeof(long double));
190                             if (error_rest_tw == NULL){printf("No se guardo espacio de
                                    memoria al array \"error_rest_tw\" \n"); exit(1);}
191                             if (error_abs_tw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_abs_tw\" \n"); exit(1);}
192                             if (error_div_tw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_div_tw\" \n"); exit(1);}
193                     // MAPE VENTANA DE VALIDACION
194                             long double *error_rest_vw, *error_abs_vw, *error_div_vw;
195                             long double cont_vw, sum_vw;
196                             error_rest_vw = malloc((frec) * sizeof(long double));
197                             error_abs_vw = malloc((frec) * sizeof(long double));
198                             error_div_vw = malloc((frec) * sizeof(long double));
199                             if (error_rest_vw == NULL){printf("No se guardo espacio de
                                    memoria al array \"error_rest_vw\" \n"); exit(1);}
200                             if (error_abs_vw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_abs_vw\" \n"); exit(1);}
201                             if (error_div_vw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_div_vw\" \n"); exit(1);}
202             // CALCULO DEL NIVEL, ESTACIONALIDAD Y TENDENCIA
203             long double alpha = parameters [n][0] , beta = parameters [n][1] , gamma =
                    parameters [n][2];
204             //printf("\n Los parametros son: alpha = %Lf, beta = %Lf y gamma= %Lf \n",
                    parameters [n][0] , parameters [n][1] , parameters [n][2]);
205             for (int i = 0; i < s-1; i++){
206                     Lt[i] = 0;
207                     Bt[i] = 0;
208             }
209             Lt[s-1] = Ls;
210             Bt[s-1] = Bs;
211             for (int i = 0; i < s; i++){
212                     St[i] = Ss[i];
213             }
214             for (int i = s ; i < (frec*(period-1)) ; i ++){//calculo de los demas vectores
215                     Lt[i] = (alpha * (tw[i] / St[i - s])) + ((1 - alpha) * (Lt[i - 1] + Bt[i
                            - 1]));
216                     Bt[i] = (beta * (Lt[i] - Lt[i - 1])) + ((1 - beta ) * Bt[i - 1]);
217                     St[i] = (gamma * (tw[i] / Lt[i])) + ((1 - gamma ) * St[i - s]);
```

```
218                 }
219                 //printf("\n nivel, tendencia y estacionalidad \n");
220                 //for(int i = 0; i < (frec*(period-1)); i++){
221                 //      printf("%Lf %Lf %Lf \n", Lt[i], Bt[i], St[i]);
222                 //}
223                 // PRONOSTICO VENTANA DE ENTRENAMIENTO
224                 for (int i = 0; i < dim_MAPE ; i++){
225                         ft_tw[i] = (Lt[i + (s-1)] + Bt[i + (s-1)]) * St[i];
226                         //printf("%Lf \n",ft_tw[i]);
227                 }
228                 // PRONOSTICO VENTANA DE VALIDACION
229                 int dim = (frec*(period-1)) - 1;
230                 int pos = 1;
231                 for (int i = 0; i < m ; i++){
232                         ft_vw[i] = (Lt[dim] + (Bt[dim] * (i+1))) * St[(dim-s) + i + 1];
233                         //printf("%Lf \n", ft_vw[i]);
234                 }
235                 //MAPE VENTA DE ENTRENAMIENTO
236                 cont_tw = 0; sum_tw = 0;
237                 for(int i = 0; i < dim_MAPE ; i++){
238                         // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
239                         error_rest_tw[i] = tw[i + s] - ft_tw[i];
240                         // VALOR ABSOLUTO
241                         if(error_rest_tw[i] < 0){
242                                 error_abs_tw[i] = error_rest_tw[i] * -1;
243                         } else{
244                                 error_abs_tw[i] = error_rest_tw[i];
245                         }
246                         // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
247                         error_div_tw[i] = (error_abs_tw[i] / tw[i + s])*100;
248
249                         // SE REALIZA EL PROMEDIO
250                         cont_tw = cont_tw + 1;
251                         sum_tw = sum_tw + error_div_tw[i];
252                 }
253                 //MAPE VENTANA DE VALIDACION
254                 cont_vw = 0; sum_vw = 0;
255                 for(int i = 0; i < frec ; i++){
256                         // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
257                         error_rest_vw[i] = vw[i] - ft_vw[i];
258                         // VALOR ABSOLUTO
259                         if(error_rest_vw[i] < 0){
260                                 error_abs_vw[i] = error_rest_vw[i] * -1;
261                         } else{
262                                 error_abs_vw[i] = error_rest_vw[i];
263                         }
264                         // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
265                         error_div_vw[i] = (error_abs_vw[i] / vw[i])*100;
266                         // SE REALIZA EL PROMEDIO
267                         cont_vw = cont_vw + 1;
268                         sum_vw = sum_vw + error_div_vw[i];
269                 }
270                 pthread_mutex_lock(&mutex);//BLOQUEA EL DATO CON EL MUTEX
271                 MAPE_tw[n] = sum_tw/cont_tw;// SE CALCULA EL MAPE DE VENTANA DE ENTRENAMIENTO
272                 MAPE_vw[n] = sum_vw/cont_vw;// SE CALCULA EL MAPE DE VENTANA DE VALIDACION
273                 for (int i = 0; i < frec; i++){
274                         global_ft_vw[i][n] = ft_vw[i];
275                 }
276                 //MEJORES RESULTADOS
277                 if(n > 0){
278                         if(MAPE_vw[n]< ref_vw ){
279                                 ref_tw = MAPE_tw[n];
280                                 ref_vw = MAPE_vw[n];
281                                 pos_ft_tw[0] = parameters[n][0];
282                                 pos_ft_tw[1] = parameters[n][1];
283                                 pos_ft_tw[2] = parameters[n][2];
```

```
284                                 pos_global_ft_vw = n;
285                         }
286                 } else{
287                         ref_tw = MAPE_tw[n];
288                         ref_vw = MAPE_vw[n];
289                         pos_ft_tw[0] = parameters[n][0];
290                         pos_ft_tw[1] = parameters[n][1];
291                         pos_ft_tw[2] = parameters[n][2];
292                         pos_global_ft_vw = n;
293                 }
294                 pthread_mutex_unlock(&mutex);//DESBLOQUEA EL DATO CON EL MUTEX
295                 // LIBERACION DE MEMORIA DINAMICA
296                 free(Lt);
297                 free(Bt);
298                 free(St);
299                 free(ft_tw);
300                 free(ft_vw);
301                 free(error_rest_vw);
302                 free(error_abs_vw);
303                 free(error_div_vw);
304                 free(error_rest_tw);
305                 free(error_abs_tw);
306                 free(error_div_tw);
307         }
308         free(Ss);
309         pthread_exit(NULL);
310 }
311 void distribute(void){
312         // ALMACENA MEMORIA EN VECTORES DINAMICOS
313         int *amount_data = malloc(core * sizeof(int));// ALMACENA CANTIDAD DE DATOS A LEER
314         args_threads *data_sent = malloc(core * sizeof(args_threads));// ARGUMENTOS PARA LOS
                HILOS
315         pthread_t *th = malloc(core * sizeof(pthread_t));// VARIABLE DONDE ALMACENA LA CANTIDAD
                DE HILOS
316         if (amount_data == NULL){printf("No se guardo espacio de memoria al array \"amount_data\"
                \n"); exit(1);}
317         if (data_sent == NULL){printf("No se guardo espacio de memoria al array \"data_sent\" \n"
                ); exit(1);}
318         if (th == NULL){printf("No se guardo espacio de memoria al array \"th\" \n"); exit(1);}
319         pthread_mutex_init(&mutex,NULL); //SE INICIALIZA EL MUTEX
320         // DIVISION
321         div_t dist = div(params_count, core);
322         //printf("\n El valor entero es: %d \n", x.quot);
323         //printf("\n El residuo es: %d \n", x.rem);
324         // CANTIDAD DE DATOS A LEER POR CADA HILO
325         for (int i = 0; i < core; i++){
326                 amount_data[i] = dist.quot;//numero entero
327         }
328         for (int i = 0; i < dist.rem; i++){
329                 amount_data[i] = amount_data[i] + 1;//residuo
330         }
331         //printf("\n Datos de finales de cada nucleo \n");
332         //for (int i = 0 ; i < core ; i++){
333         //      printf("%d \n", amount_data[i]);
334         //}
335         // DECLARA LOS PARAMETROS EN EL VECTOR DE ARGUMENTOS
336         for (int i = 0; i < core; i++){
337                 if(i == 0){
338                         data_sent[i].ini = 0;
339                         data_sent[i].fin = amount_data[i];
340                 } else{
341                         data_sent[i].ini = data_sent[i - 1].fin;
342                         data_sent[i].fin = data_sent[i].ini + amount_data[i];
343                 }
344         }
345         // CREACION DE HILOS
```

```
346     for (int i = 0; i < core; i++){
347             pthread_create(&th[i], NULL, MHW, &data_sent[i]);
348     }
349     for (int i= 0; i < core; i++){
350             pthread_join(th[i], NULL);
351     }
352     pthread_mutex_destroy(&mutex); //SE INICIALIZA EL MUTEX
353     // LIBERAR MEMORIA
354     free(amount_data);
355     free(data_sent);
356     free(th);
357 }
358 void output_file(void){
359 //***** FICHERO SERIE DE TIEMPO
360     f_ts = fopen("MHW_time_series.txt", "w");
361     if(f_ts == NULL){ printf("\n Error al abrir el archivo que va a contener la serie de
            tiempo \n"); exit(1);}
362     long double **time_series, **ventana_entrenamiento;
363     time_series = malloc(frec * sizeof(long double*));
364     ventana_entrenamiento = malloc(frec * sizeof(long double*));
365     for (int i = 0; i < frec; i++){
366             time_series[i] = malloc(period * sizeof(long double));
367             ventana_entrenamiento[i] = malloc((period-1) * sizeof(long double));
368     }
369     int cont, pos = 0;
370     for(int i = 0; i < period ; i++){
371             cont = 0;
372             for(int j = 0; j < frec ; j++){
373                     time_series[j][i] = st[j + pos];
374                     cont = cont + 1;
375                     if(i < period-1){
376                             ventana_entrenamiento[j][i] = time_series[j][i];
377                     }
378             }
379             pos = pos + cont;
380     }
381     fprintf(f_ts, "\n");
382     fprintf(f_ts, "***** METODO DE PRONOSTICO MHW PARA SERIES DE TIEMPO *****\n");
383     fprintf(f_ts, "\n");
384     fprintf(f_ts, " INFORMACION DEL METODO MHW:\n");
385     fprintf(f_ts, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
386     fprintf(f_ts, "- NOMBRE DEL FICHERO: %s\n", name_in);
387     fprintf(f_ts, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
388     fprintf(f_ts, "- CANTIDAD DE PERIODOS: %d\n", period);
389     fprintf(f_ts, "- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
390     fprintf(f_ts, "- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
391     fprintf(f_ts, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
392     fprintf(f_ts, "- USO DE PROCESADORES: %d\n", core);
393     fprintf(f_ts, "- PARAMETROS MAS EFICIENTES: Alpha = %Lf, Beta = %Lf y Gamma = %Lf\n",
            pos_ft_tw[0], pos_ft_tw[1],pos_ft_tw[2]);
394     fprintf(f_ts, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
395     fprintf(f_ts, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
396     fprintf(f_ts, "\n");
397     fprintf(f_ts, "\n");
398     fprintf(f_ts, " SERIE DE TIEMPO\n");
399     fprintf(f_ts, "\n");
400     fprintf(f_ts, "\n");
401     for (int i = 0 ; i < frec ; i++){
402             for(int j = 0 ; j < period ; j++){
403                     fprintf(f_ts, "%Lf ",  time_series[i][j]);
404             }
405             fprintf(f_ts, "\n");
406     }
407     fprintf(f_ts, "\n");
408     fprintf(f_ts, "\n");
409     fprintf(f_ts, " VENTANA DE ENTRENAMIENTO\n");
```

```
410       fprintf(f_ts, "\n");
411       fprintf(f_ts, "\n");
412       for (int i = 0 ; i < frec ; i++){
413                 for(int j = 0 ; j < period-1 ; j++){
414                           fprintf(f_ts, "%Lf ",  ventana_entrenamiento[i][j]);
415                 }
416                 fprintf(f_ts, "\n");
417       }
418       fprintf(f_ts, "\n");
419       fprintf(f_ts, "\n");
420       fprintf(f_ts, " VENTANA DE VALIDACION\n");
421       fprintf(f_ts, "\n");
422       fprintf(f_ts, "\n");
423       for(int j = 0 ; j < frec ; j++){
424                           fprintf(f_ts, "%Lf \n",vw[j]);
425       }
426       for (int i = 0 ; i < frec ; i++){
427                 free(time_series[i]);
428                 free(ventana_entrenamiento[i]);
429       }
430       free(time_series);
431       free(ventana_entrenamiento);
432       if(fclose(f_ts) != 0){printf("\n no se ha podido cerrar el archivo que contiene la serie
              de tiempo \n"); exit(1);}
433   //***** FICHERO PARAMETRO
434       f_param = fopen("MHW_parameters.txt", "w");
435       if(f_param == NULL){ printf("\n Error al abrir el archivo que va a contener los
              parametros \n"); exit(1);}
436       fprintf(f_param, "\n");
437       fprintf(f_param, "***** METODO DE PRONOSTICO MHW PARA SERIES DE TIEMPO *****\n");
438       fprintf(f_param, "\n");
439       fprintf(f_param, " INFORMACION DEL METODO MHW:\n");
440       fprintf(f_param, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
441       fprintf(f_param, "- NOMBRE DEL FICHERO: %s\n", name_in);
442       fprintf(f_param, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
443       fprintf(f_param, "- CANTIDAD DE PERIODOS: %d\n", period);
444       fprintf(f_param, "- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
445       fprintf(f_param, "- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
446       fprintf(f_param, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
447       fprintf(f_param, "- USO DE PROCESADORES: %d\n", core);
448       fprintf(f_param, "- PARAMETROS MAS EFICIENTES: Alpha = %Lf, Beta = %Lf y Gamma = %Lf\n",
              pos_ft_tw[0], pos_ft_tw[1],pos_ft_tw[2]);
449       fprintf(f_param, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
450       fprintf(f_param, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
451       fprintf(f_param, "\n");
452       fprintf(f_param, "\n");
453       fprintf(f_param, "\n");
454       fprintf(f_param, "PARAMETRO                              MAPE                MAPE \n");
455       fprintf(f_param, "                                  ENTRENAMIENTO          VALIDACION \n\n
              \n");
456       for(int i = 0; i < params_count; i++){
457                 fprintf(f_param, "alpha = %Lf, Beta = %Lf y Gamma = %Lf          %Lf              
                        %Lf \n",parameters[i][0],parameters[i][1],parameters[i][2], MAPE_tw[i],
                        MAPE_vw[i]);
458       }
459       if(fclose(f_param) != 0){printf("\n no se ha podido cerrar el archivo que contiene los
              parametros \n"); exit(1);}
460   //***** FICHERO PRONOSTICO DE LA VENTANA DE VALIDACION
461       f_ft_vw = fopen("MHW_forecasting.txt", "w");
462       if(f_ft_vw == NULL){ printf("\n Error al abrir el archivo que contienen el pronostico de
              la ventana de entrenamiento \n"); exit(1);}
463       fprintf(f_ft_vw, "\n");
464       fprintf(f_ft_vw, "***** METODO DE PRONOSTICO MHW PARA SERIES DE TIEMPO *****\n");
465       fprintf(f_ft_vw, "\n");
466       fprintf(f_ft_vw, " INFORMACION DEL METODO MHW:\n");
467       fprintf(f_ft_vw, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
```

```
468       fprintf(f_ft_vw, "- NOMBRE DEL FICHERO: %s\n", name_in);
469       fprintf(f_ft_vw, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
470       fprintf(f_ft_vw, "- CANTIDAD DE PERIODOS: %d\n", period);
471       fprintf(f_ft_vw, "- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
472       fprintf(f_ft_vw, "- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
473       fprintf(f_ft_vw, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
474       fprintf(f_ft_vw, "- USO DE PROCESADORES: %d\n", core);
475       fprintf(f_ft_vw, "- PARAMETROS MAS EFICIENTES: Alpha = %Lf, Beta = %Lf y Gamma = %Lf\n",
              pos_ft_tw[0], pos_ft_tw[1], pos_ft_tw[2]);
476       fprintf(f_ft_vw, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
477       fprintf(f_ft_vw, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
478       fprintf(f_ft_vw, "\n");
479       fprintf(f_ft_vw, "\n");
480       fprintf(f_ft_vw, "\n");
481       for (int i = 0 ; i < frec ; i++){
482                         fprintf(f_ft_vw, "%Lf\n",  global_ft_vw[i][pos_global_ft_vw]);
483       }
484       if(fclose(f_ft_vw) != 0){printf("\n no se ha podido cerrar el archivo que contiene el
              pronostico de la ventana de entrenamiento \n"); exit(1);}
485   }
486   ////////////////////////////////////////////////////////////////////////////
487   //FUNCION PRINCIPAL
488   int main(int arcg, char *argv[]){
489   //******VARIABLES
490       name_ejecutable = argv[0];
491       name_in = argv[1];
492       frec = atoi(argv[2]);
493       period = atoi(argv[3]);
494       steps = atof(argv[4]);
495       core = atoi(argv[5]);
496       s = frec;
497      m = frec;
498       params_count = (1/steps + 1) * (1/steps + 1) * (1/steps + 1) + 1;
499   //******FUNCIONES
500       printf("\n");
501       printf("***** METODO DE PRONOSTICO MHW PARA SERIES DE TIEMPO *****\n");
502       printf("\n");
503       printf(" INFORMACION DEL METODO MHW:\n");
504       printf("- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
505       printf("- NOMBRE DEL FICHERO: %s\n", name_in);
506       printf("- FRECUENCIA DE LOS DATOS: %d\n", frec);
507       printf("- CANTIDAD DE PERIODOS: %d\n", period);
508       printf("- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
509       printf("- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
510       printf("- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
511       printf("- USO DE PROCESADORES: %d\n", core);
512       printf("\n Reservando Memoria Dinamica...\n");
513       allocate_memory();
514       printf("\n Leyendo serie de tiempo...\n");
515       read_data();
516       printf("\n Ordenando serie de tiempo...\n");
517       tw_vw();
518       parametros();
519       printf("\n Inicio del programa...\n");
520       distribute();
521       printf("\n Salida de archivos...\n");
522       output_file();
523       printf("\n Liberar memoria Dinamica...\n");
524       free_memory();
525       return 0;
526   }
```

Apéndice H

Código en C método AHW

A continuación, se presenta el código de programación en C que realiza la selección del parámetro óptimo del método de pronóstico de series de tiempo de AHW usando el cómputo en paralelo basado en hilos.

```
1  ////////////////////////////////////////////////////////////////////////////
2  // UNIVERSIDAD MICHOACANA DE SAN NICOLAS DE HIDALGO
3  // DIVISION DE ESTUDIOS DE POSGRADO DE INGENIERIA ELECTRICA
4  // AREA: SISTEMAS ELECTRICOS DE POTENCIA
5  // METODO DE PRONOSTICO: HOLT-WINTERS ADITIVO
6  // PROGRAMADOR : JESUS DE LA TORRE BUCIO
7  // COMPUTO EN PARALELO
8  ////////////////////////////////////////////////////////////////////////////
9
10 ////////////////////////////////////////////////////////////////////////////
11 //LIBRERIAS
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <math.h>
15 #include <pthread.h>
16
17 ////////////////////////////////////////////////////////////////////////////
18 //VARIABLES GLOBALES
19 // DIMENSIONES DE LA SERIE DE TIEMPO
20 int period, frec, s, m, params_count, core, pos_global_ft_vw;
21 // MEMORIA DINAMICA
22 long double  *st, *tw, *vw, **parameters, *MAPE_tw, *MAPE_vw, **global_ft_vw;
23 //VARIABLES DEL MAPE
24 long double ref_tw, ref_vw, *pos_ft_tw, steps ;
25 //ARCHIVOS DE ENTRADA Y SALIDA
26 FILE *f_in, *f_ts, *f_param, *f_ft_vw;
27 char *name_in, *name_ejecutable;
28 // MUTEX
29 pthread_mutex_t mutex;
30
31 ////////////////////////////////////////////////////////////////////////////
32 // ESTRUCTURA
33 typedef struct{
34     int ini;
35     int fin;
36 }args_threads;
37
38 ////////////////////////////////////////////////////////////////////////////
39 //FUNCIONES
40 void allocate_memory(void){// SE GENERA LA MEMORIA DINAMICA DE LAS VARIABLES GLOBALES
41     st = malloc((frec*period) * sizeof(long double));
```

```
42      tw = malloc((frec*(period-1)) * sizeof(long double));
43      vw = malloc(frec * sizeof(long double));
44      MAPE_tw = malloc(params_count * sizeof(long double));
45      MAPE_vw = malloc(params_count * sizeof(long double));
46      global_ft_vw = malloc(frec * sizeof(long double*));
47      pos_ft_tw = malloc(3 * sizeof(long double));
48      if (st == NULL){printf("No se guardo espacio de memoria de st \n"); exit(1);}
49      if (tw == NULL){printf("No se guardo espacio de memoria al array \"tw\" \n"); exit(1);}
50      if (vw == NULL){printf("No se guardo espacio de memoria al array \"vw\" \n"); exit(1);}
51      if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"global_ft_vw
            \" \n"); exit(1);}
52      if (pos_ft_tw == NULL){printf("No se guardo espacio de memoria al array \"pos_ft_tw\" \n
            "); exit(1);}
53      for (int i = 0; i < frec; i++){
54              global_ft_vw[i] = malloc(params_count * sizeof(long double));
55              if (global_ft_vw == NULL){printf("No se guardo espacio de memoria al array \"
                    global_ft_vw\" \n"); exit(1);}
56      }
57      parameters = malloc(params_count * sizeof(long double*));
58      if (parameters == NULL){printf("No se guardo espacio de memoria al array \"parameters\" \
            n"); exit(1);}
59      for (int i = 0 ; i < params_count; i++){
60              parameters[i] = malloc(3 * sizeof(long double));
61              if (parameters == NULL){printf("No se guardo espacio de memoria al array \"
                    parameters\" \n"); exit(1);}
62      }
63  }
64  void read_data(void){// SE ESTABLECE TODA LA SERIE DE TIEMPO EN UN ARRAY LLAMADO "st"
65      f_in = fopen(name_in, "r");
66      if(f_in == NULL){ printf("\n Error al abrir el archivo de lectura de datos llamado %s \n"
            , name_in); exit(1);}
67      for (int i = 0; i < (frec*period) ; i++){
68              fscanf(f_in, "%Lf", &st[i]);
69      }
70      if(fclose(f_in) != 0){printf("\n no se ha podido cerrar el fichero %s \n", name_in); exit
            (1);}
71      /*for (int i = 0; i < (frec*period); i++){
72              printf("%Lf \n", st[i]);
73      }*/
74  }
75  void tw_vw(void){// SE ESTABLECE LA SERIE DE TIEMPO DE LA VENTANA DE VALIDACION Y
        ENTRENAMIENTO
76      int pos_tw = 0, pos_vw = 0;
77      for(int i = 0; i < (frec*period) ; i++){
78              if(i < ((period-1)*frec)){
79                      tw[pos_tw] = st[i];
80                      pos_tw = pos_tw + 1;
81              } else{
82                      vw[pos_vw] = st[i];
83                      pos_vw = pos_vw +1;
84              }
85      }
86      /*printf("\n serie de tiempo\n");
87      for(int i = 0; i < (frec*period); i++){
88              printf("%Lf \n", st[i]);
89      }
90      printf("\n Ventana de entrenamiento\n");
91      for(int i = 0; i < (frec*(period-1)); i++){
92              printf("%Lf \n", tw[i]);
93      }
94      printf("\n Ventana de validacion\n");
95      for(int i = 0; i < frec ; i++){
96              printf("%Lf \n", vw[i]);
97      }*/
98  }
99
```

```
100 void parametros(void){
101     long double phi, psi, ommega;
102     phi = 0, psi = 0, ommega = 0;
103     for (int i = 0; i < params_count; i++){
104             if ((phi < 1+steps) && (phi >= 0)){
105                     parameters[i][0] = phi;
106                     parameters[i][1] = psi;
107                     parameters[i][2] = ommega;
108                     phi = phi + steps;
109             } else{
110                     phi = 0;
111                     parameters[i][0] = phi;
112                     phi = phi + steps;
113                     psi = psi + steps;
114                     if((psi < 1+steps) && (psi >= 0)){
115                             parameters[i][1] = psi;
116                             parameters[i][2] = ommega;
117                     } else{
118                             psi = 0;
119                             parameters[i][1] = psi;
120                             ommega = ommega + steps;
121                             parameters[i][2] = ommega;
122                     }
123             }
124     }
125 }
126 void free_memory(void){// SE LIBERA MEMORIA DINAMIECA
127     for (int i = 0 ; i < params_count ; i++){
128             free(parameters[i]);
129     }
130     free(parameters);
131     free(tw);
132     free(st);
133     free(vw);
134     free(MAPE_tw);
135     free(MAPE_vw);
136     for (int i = 0 ; i < frec ; i++){
137             free(global_ft_vw[i]);
138     }
139     free(global_ft_vw);
140     free(pos_ft_tw);
141 }
142 void *AHW(void *rank){
143 //****** ESTABLECE LOS PARAMETROS DE INICIO Y FINAL DE CADA HILO
144     args_threads *thread_data = (args_threads*)rank;
145     int start = thread_data -> ini;
146     int end = thread_data -> fin;
147     //printf(" \n inicio del hilo %d fin del hilo %d \n", start,end);
148     //VALORES INICIALES lt, Bt, St
149     long double sum_Ls = 0, sum_Bs = 0, Ls=0, Bs=0, *Ss;
150     Ss = malloc((s) * sizeof(long double));
151     if (Ss == NULL){printf("No se guardo espacio de memoria al array \"Ss\" \n"); exit(1);}
152     long double s_1 = s;
153     for (int i = 0; i < s; i++){
154             sum_Ls = sum_Ls + tw[i];//NIVEL
155             //printf("%Lf \n", sum_Ls);
156             sum_Bs = sum_Bs + ((tw[i+(s)]-tw[i])/s_1);//TENDENCIA
157             //printf("%Lf \n", sum_Bs);
158     }
159     Ls = (1/s_1)*(sum_Ls); // VALOR INICIAL DEL NIVEL
160     Bs = (1/s_1)*sum_Bs; // VALOR INICIAL DE TENDENCIA
161     //printf("%Lf \n", Ls);
162     //printf("%Lf \n", Bs);
163     for (int i = 0; i < s; i++){
164             Ss[i] = tw[i]-Ls;//VALORES INICIALES DE ESTACIONALIDAD
165             //printf("%Lf \n", Ss[i]);
```

```
166     }
167     for (int n = start; n < end; n++){
168             // MEMORIA DINAMICA
169                     int dim_MAPE = (frec*(period-1)) - s;
170                     //METODO DE HOLT WINTERS
171                             long double *Lt, *Bt, *St;
172                             Lt = malloc((frec*(period-1)) * sizeof(long double));
173                             Bt = malloc((frec*(period-1)) * sizeof(long double));
174                             St = malloc((frec*(period-1)) * sizeof(long double));
175                             if (Bt == NULL){printf("No se guardo espacio de memoria al array
                                    \"Bt\" \n"); exit(1);}
176                             if (Lt == NULL){printf("No se guardo espacio de memoria al array
                                    \"Lt\" \n"); exit(1);}
177                             if (St == NULL){printf("No se guardo espacio de memoria al array
                                    \"St\" \n"); exit(1);}
178                     // PRONOSTICO VENTANA DE ENTRENAMIENTO Y VALIDACION
179                             long double *ft_tw, *ft_vw;
180                             ft_tw = malloc((frec*(period-1)) * sizeof(long double));
181                             ft_vw = malloc(frec * sizeof(long double));
182                             if (ft_tw == NULL){printf("No se guardo espacio de memoria al
                                    array \"ft_tw\" \n"); exit(1);}
183                             if (ft_vw == NULL){printf("No se guardo espacio de memoria al
                                    array \"ft_vw\" \n"); exit(1);}
184                     // MAPE VENTANA DE ENTRENAMIENTO
185                             long double *error_rest_tw, *error_abs_tw, *error_div_tw;
186                             long double cont_tw, sum_tw;
187                             error_rest_tw = malloc(dim_MAPE * sizeof(long double));
188                             error_abs_tw = malloc(dim_MAPE * sizeof(long double));
189                             error_div_tw = malloc(dim_MAPE * sizeof(long double));
190                             if (error_rest_tw == NULL){printf("No se guardo espacio de
                                    memoria al array \"error_rest_tw\" \n"); exit(1);}
191                             if (error_abs_tw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_abs_tw\" \n"); exit(1);}
192                             if (error_div_tw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_div_tw\" \n"); exit(1);}
193                     // MAPE VENTANA DE VALIDACION
194                             long double *error_rest_vw, *error_abs_vw, *error_div_vw;
195                             long double cont_vw, sum_vw;
196                             error_rest_vw = malloc((frec) * sizeof(long double));
197                             error_abs_vw = malloc((frec) * sizeof(long double));
198                             error_div_vw = malloc((frec) * sizeof(long double));
199                             if (error_rest_vw == NULL){printf("No se guardo espacio de
                                    memoria al array \"error_rest_vw\" \n"); exit(1);}
200                             if (error_abs_vw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_abs_vw\" \n"); exit(1);}
201                             if (error_div_vw == NULL){printf("No se guardo espacio de memoria
                                    al array \"error_div_vw\" \n"); exit(1);}
202             // CALCULO DEL NIVEL, ESTACIONALIDAD Y TENDENCIA
203             long double alpha = parameters [n][0] , beta = parameters [n][1] , gamma =
                    parameters [n][2];
204             //printf("\n Los parametros son: alpha = %Lf, beta = %Lf y gamma= %Lf \n",
                    parameters [n][0] , parameters [n][1] , parameters [n][2]);
205             for (int i = 0; i < s-1; i++){
206                     Lt[i] = 0;
207                     Bt[i] = 0;
208             }
209             Lt[s-1] = Ls;
210             Bt[s-1] = Bs;
211             for (int i = 0; i < s; i++){
212                     St[i] = Ss[i];
213             }
214             for (int i = s ; i < (frec*(period-1)) ; i ++){//calculo de los demas vectores
215                     Lt[i] = (alpha * (tw[i] - St[i - s])) + ((1 - alpha) * (Lt[i - 1] + Bt[i
                            - 1]));
216                     Bt[i] = (beta * (Lt[i] - Lt[i - 1])) + ((1 - beta ) * Bt[i - 1]);
217                     St[i] = (gamma * (tw[i] - Lt[i])) + ((1 - gamma ) * St[i - s]);
```

```
218                 }
219                 //printf("\n nivel, tendencia y estacionalidad \n");
220                 //for(int i = 0; i < (frec*(period-1)); i++){
221                 //      printf("%Lf %Lf %Lf \n", Lt[i], Bt[i], St[i]);
222                 //}
223                 // PRONOSTICO VENTANA DE ENTRENAMIENTO
224                 for (int i = 0; i < dim_MAPE ; i++){
225                         ft_tw[i] = Lt[i + (s-1)] + Bt[i + (s-1)] + St[i];
226                         //printf("%Lf \n",ft_tw[i]);
227                 }
228                 // PRONOSTICO VENTANA DE VALIDACION
229                 int dim = (frec*(period-1)) - 1;
230                 int pos = 1;
231                 for (int i = 0; i < m ; i++){
232                         ft_vw[i] = Lt[dim] + (Bt[dim] * (i+1)) + St[(dim-s) + i + 1];
233                         //printf("%Lf \n", ft_vw[i]);
234                 }
235                 //MAPE VENTA DE ENTRENAMIENTO
236                 cont_tw = 0; sum_tw = 0;
237                 for(int i = 0; i < dim_MAPE ; i++){
238                         // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
239                         error_rest_tw[i] = tw[i + s] - ft_tw[i];
240                         // VALOR ABSOLUTO
241                         if(error_rest_tw[i] < 0){
242                                 error_abs_tw[i] = error_rest_tw[i] * -1;
243                         } else{
244                                 error_abs_tw[i] = error_rest_tw[i];
245                         }
246                         // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
247                         error_div_tw[i] = (error_abs_tw[i] / tw[i + s])*100;
248
249                         // SE REALIZA EL PROMEDIO
250                         cont_tw = cont_tw + 1;
251                         sum_tw = sum_tw + error_div_tw[i];
252                 }
253                 //MAPE VENTANA DE VALIDACION
254                 cont_vw = 0; sum_vw = 0;
255                 for(int i = 0; i < frec ; i++){
256                         // SE REALIZA LA RESTA DEL VALOR REAL Y EL PRONOSTICADO
257                         error_rest_vw[i] = vw[i] - ft_vw[i];
258                         // VALOR ABSOLUTO
259                         if(error_rest_vw[i] < 0){
260                                 error_abs_vw[i] = error_rest_vw[i] * -1;
261                         } else{
262                                 error_abs_vw[i] = error_rest_vw[i];
263                         }
264                         // LA DIVISION DEL VALOR ABSOLUTO Y EL VALOR REAL
265                         error_div_vw[i] = (error_abs_vw[i] / vw[i])*100;
266                         // SE REALIZA EL PROMEDIO
267                         cont_vw = cont_vw + 1;
268                         sum_vw = sum_vw + error_div_vw[i];
269                 }
270                 pthread_mutex_lock(&mutex);//BLOQUEA EL DATO CON EL MUTEX
271                 MAPE_tw[n] = sum_tw/cont_tw;// SE CALCULA EL MAPE DE VENTANA DE ENTRENAMIENTO
272                 MAPE_vw[n] = sum_vw/cont_vw;// SE CALCULA EL MAPE DE VENTANA DE VALIDACION
273                 for (int i = 0; i < frec; i++){
274                         global_ft_vw[i][n] = ft_vw[i];
275                 }
276                 //MEJORES RESULTADOS
277                 if(n > 0){
278                         if(MAPE_vw[n]< ref_vw ){
279                                 ref_tw = MAPE_tw[n];
280                                 ref_vw = MAPE_vw[n];
281                                 pos_ft_tw[0] = parameters[n][0];
282                                 pos_ft_tw[1] = parameters[n][1];
283                                 pos_ft_tw[2] = parameters[n][2];
```

```
284                                 pos_global_ft_vw = n;
285                         }
286                 } else{
287                         ref_tw = MAPE_tw[n];
288                         ref_vw = MAPE_vw[n];
289                         pos_ft_tw[0] = parameters[n][0];
290                         pos_ft_tw[1] = parameters[n][1];
291                         pos_ft_tw[2] = parameters[n][2];
292                         pos_global_ft_vw = n;
293                 }
294                 pthread_mutex_unlock(&mutex);//DESBLOQUEA EL DATO CON EL MUTEX
295                 // LIBERACION DE MEMORIA DINAMICA
296                 free(Lt);
297                 free(Bt);
298                 free(St);
299                 free(ft_tw);
300                 free(ft_vw);
301                 free(error_rest_vw);
302                 free(error_abs_vw);
303                 free(error_div_vw);
304                 free(error_rest_tw);
305                 free(error_abs_tw);
306                 free(error_div_tw);
307         }
308         free(Ss);
309         pthread_exit(NULL);
310 }
311 void distribute(void){
312         // ALMACENA MEMORIA EN VECTORES DINAMICOS
313         int *amount_data = malloc(core * sizeof(int));// ALMACENA CANTIDAD DE DATOS A LEER
314         args_threads *data_sent = malloc(core * sizeof(args_threads));// ARGUMENTOS PARA LOS
                HILOS
315         pthread_t *th = malloc(core * sizeof(pthread_t));// VARIABLE DONDE ALMACENA LA CANTIDAD
                DE HILOS
316         if (amount_data == NULL){printf("No se guardo espacio de memoria al array \"amount_data\"
                \n"); exit(1);}
317         if (data_sent == NULL){printf("No se guardo espacio de memoria al array \"data_sent\" \n"
                ); exit(1);}
318         if (th == NULL){printf("No se guardo espacio de memoria al array \"th\" \n"); exit(1);}
319         pthread_mutex_init(&mutex,NULL); //SE INICIALIZA EL MUTEX
320         // DIVISION
321         div_t dist = div(params_count, core);
322         //printf("\n El valor entero es: %d \n", x.quot);
323         //printf("\n El residuo es: %d \n", x.rem);
324         // CANTIDAD DE DATOS A LEER POR CADA HILO
325         for (int i = 0; i < core; i++){
326                 amount_data[i] = dist.quot;//numero entero
327         }
328         for (int i = 0; i < dist.rem; i++){
329                 amount_data[i] = amount_data[i] + 1;//residuo
330         }
331         //printf("\n Datos de finales de cada nucleo \n");
332         //for (int i = 0 ; i < core ; i++){
333         //      printf("%d \n", amount_data[i]);
334         //}
335         // DECLARA LOS PARAMETROS EN EL VECTOR DE ARGUMENTOS
336         for (int i = 0; i < core; i++){
337                 if(i == 0){
338                         data_sent[i].ini = 0;
339                         data_sent[i].fin = amount_data[i];
340                 } else{
341                         data_sent[i].ini = data_sent[i - 1].fin;
342                         data_sent[i].fin = data_sent[i].ini + amount_data[i];
343                 }
344         }
345         // CREACION DE HILOS
```

```
346     for (int i = 0; i < core; i++){
347             pthread_create(&th[i], NULL, AHW, &data_sent[i]);
348     }
349     for (int i= 0; i < core; i++){
350             pthread_join(th[i], NULL);
351     }
352     pthread_mutex_destroy(&mutex); //SE INICIALIZA EL MUTEX
353     // LIBERAR MEMORIA
354     free(amount_data);
355     free(data_sent);
356     free(th);
357 }
358 void output_file(void){
359 //***** FICHERO SERIE DE TIEMPO
360     f_ts = fopen("AHW_time_series.txt", "w");
361     if(f_ts == NULL){ printf("\n Error al abrir el archivo que va a contener la serie de
            tiempo \n"); exit(1);}
362     long double **time_series, **ventana_entrenamiento;
363     time_series = malloc(frec * sizeof(long double*));
364     ventana_entrenamiento = malloc(frec * sizeof(long double*));
365     for (int i = 0; i < frec; i++){
366             time_series[i] = malloc(period * sizeof(long double));
367             ventana_entrenamiento[i] = malloc((period-1) * sizeof(long double));
368     }
369     int cont, pos = 0;
370     for(int i = 0; i < period ; i++){
371             cont = 0;
372             for(int j = 0; j < frec ; j++){
373                     time_series[j][i] = st[j + pos];
374                     cont = cont + 1;
375                     if(i < period-1){
376                             ventana_entrenamiento[j][i] = time_series[j][i];
377                     }
378             }
379             pos = pos + cont;
380     }
381     fprintf(f_ts, "\n");
382     fprintf(f_ts, "***** METODO DE PRONOSTICO AHW PARA SERIES DE TIEMPO *****\n");
383     fprintf(f_ts, "\n");
384     fprintf(f_ts, " INFORMACION DEL METODO AHW:\n");
385     fprintf(f_ts, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
386     fprintf(f_ts, "- NOMBRE DEL FICHERO: %s\n", name_in);
387     fprintf(f_ts, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
388     fprintf(f_ts, "- CANTIDAD DE PERIODOS: %d\n", period);
389     fprintf(f_ts, "- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
390     fprintf(f_ts, "- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
391     fprintf(f_ts, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
392     fprintf(f_ts, "- USO DE PROCESADORES: %d\n", core);
393     fprintf(f_ts, "- PARAMETROS MAS EFICIENTES: Alpha = %Lf, Beta = %Lf y Gamma = %Lf\n",
            pos_ft_tw[0], pos_ft_tw[1],pos_ft_tw[2]);
394     fprintf(f_ts, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
395     fprintf(f_ts, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
396     fprintf(f_ts, "\n");
397     fprintf(f_ts, "\n");
398     fprintf(f_ts, " SERIE DE TIEMPO\n");
399     fprintf(f_ts, "\n");
400     fprintf(f_ts, "\n");
401     for (int i = 0 ; i < frec ; i++){
402             for(int j = 0 ; j < period ; j++){
403                     fprintf(f_ts, "%Lf ",  time_series[i][j]);
404             }
405             fprintf(f_ts, "\n");
406     }
407     fprintf(f_ts, "\n");
408     fprintf(f_ts, "\n");
409     fprintf(f_ts, " VENTANA DE ENTRENAMIENTO\n");
```

```
410       fprintf(f_ts, "\n");
411       fprintf(f_ts, "\n");
412       for (int i = 0 ; i < frec ; i++){
413               for(int j = 0 ; j < period-1 ; j++){
414                       fprintf(f_ts, "%Lf ",  ventana_entrenamiento[i][j]);
415               }
416               fprintf(f_ts, "\n");
417       }
418       fprintf(f_ts, "\n");
419       fprintf(f_ts, "\n");
420       fprintf(f_ts, " VENTANA DE VALIDACION\n");
421       fprintf(f_ts, "\n");
422       fprintf(f_ts, "\n");
423       for(int j = 0 ; j < frec ; j++){
424                       fprintf(f_ts, "%Lf \n",vw[j]);
425       }
426       for (int i = 0 ; i < frec ; i++){
427               free(time_series[i]);
428               free(ventana_entrenamiento[i]);
429       }
430       free(time_series);
431       free(ventana_entrenamiento);
432       if(fclose(f_ts) != 0){printf("\n no se ha podido cerrar el archivo que contiene la serie
              de tiempo \n"); exit(1);}
433   //***** FICHERO PARAMETRO
434       f_param = fopen("AHW_parameters.txt", "w");
435       if(f_param == NULL){ printf("\n Error al abrir el archivo que va a contener los
              parametros \n"); exit(1);}
436       fprintf(f_param, "\n");
437       fprintf(f_param, "***** METODO DE PRONOSTICO AHW PARA SERIES DE TIEMPO *****\n");
438       fprintf(f_param, "\n");
439       fprintf(f_param, " INFORMACION DEL METODO AHW:\n");
440       fprintf(f_param, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
441       fprintf(f_param, "- NOMBRE DEL FICHERO: %s\n", name_in);
442       fprintf(f_param, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
443       fprintf(f_param, "- CANTIDAD DE PERIODOS: %d\n", period);
444       fprintf(f_param, "- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
445       fprintf(f_param, "- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
446       fprintf(f_param, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
447       fprintf(f_param, "- USO DE PROCESADORES: %d\n", core);
448       fprintf(f_param, "- PARAMETROS MAS EFICIENTES: Alpha = %Lf, Beta = %Lf y Gamma = %Lf\n",
              pos_ft_tw[0], pos_ft_tw[1],pos_ft_tw[2]);
449       fprintf(f_param, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
450       fprintf(f_param, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
451       fprintf(f_param, "\n");
452       fprintf(f_param, "\n");
453       fprintf(f_param, "\n");
454       fprintf(f_param, "PARAMETRO                              MAPE                MAPE \n");
455       fprintf(f_param, "                                ENTRENAMIENTO          VALIDACION \n\n
              \n");
456       for(int i = 0; i < params_count; i++){
457               fprintf(f_param, "alpha = %Lf, Beta = %Lf y Gamma = %Lf          %Lf              
                    %Lf \n",parameters[i][0],parameters[i][1],parameters[i][2], MAPE_tw[i],
                    MAPE_vw[i]);
458       }
459       if(fclose(f_param) != 0){printf("\n no se ha podido cerrar el archivo que contiene los
              parametros \n"); exit(1);}
460   //***** FICHERO PRONOSTICO DE LA VENTANA DE VALIDACION
461       f_ft_vw = fopen("AHW_forecasting.txt", "w");
462       if(f_ft_vw == NULL){ printf("\n Error al abrir el archivo que contienen el pronostico de
              la ventana de entrenamiento \n"); exit(1);}
463       fprintf(f_ft_vw, "\n");
464       fprintf(f_ft_vw, "***** METODO DE PRONOSTICO AHW PARA SERIES DE TIEMPO *****\n");
465       fprintf(f_ft_vw, "\n");
466       fprintf(f_ft_vw, " INFORMACION DEL METODO AHW:\n");
467       fprintf(f_ft_vw, "- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
```

```
468       fprintf(f_ft_vw, "- NOMBRE DEL FICHERO: %s\n", name_in);
469       fprintf(f_ft_vw, "- FRECUENCIA DE LOS DATOS: %d\n", frec);
470       fprintf(f_ft_vw, "- CANTIDAD DE PERIODOS: %d\n", period);
471       fprintf(f_ft_vw, "- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
472       fprintf(f_ft_vw, "- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
473       fprintf(f_ft_vw, "- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
474       fprintf(f_ft_vw, "- USO DE PROCESADORES: %d\n", core);
475       fprintf(f_ft_vw, "- PARAMETROS MAS EFICIENTES: Alpha = %Lf, Beta = %Lf y Gamma = %Lf\n",
              pos_ft_tw[0], pos_ft_tw[1],pos_ft_tw[2]);
476       fprintf(f_ft_vw, "- VALOR DE MAPE EN TW: %Lf\n", ref_tw);
477       fprintf(f_ft_vw, "- VALOR DE MAPE EN VW: %Lf\n", ref_vw);
478       fprintf(f_ft_vw, "\n");
479       fprintf(f_ft_vw, "\n");
480       fprintf(f_ft_vw, "\n");
481       for (int i = 0 ; i < frec ; i++){
482                         fprintf(f_ft_vw, "%Lf\n",  global_ft_vw[i][pos_global_ft_vw]);
483       }
484       if(fclose(f_ft_vw) != 0){printf("\n no se ha podido cerrar el archivo que contiene el
              pronostico de la ventana de entrenamiento \n"); exit(1);}
485   }
486   ////////////////////////////////////////////////////////////////////////////
487   //FUNCION PRINCIPAL
488   int main(int arcg, char *argv[]){
489   //******VARIABLES
490       name_ejecutable = argv[0];
491       name_in = argv[1];
492       frec = atoi(argv[2]);
493       period = atoi(argv[3]);
494       steps = atof(argv[4]);
495       core = atoi(argv[5]);
496       s = frec;
497      m = frec;
498       steps = atof(argv[1]);
499   //******FUNCIONES
500       printf("\n");
501       printf("***** METODO DE PRONOSTICO AHW PARA SERIES DE TIEMPO *****\n");
502       printf("\n");
503       printf(" INFORMACION DEL METODO AHW:\n");
504       printf("- NOMBRE DEL EJECUTABLE: %s\n", name_ejecutable);
505       printf("- NOMBRE DEL FICHERO: %s\n", name_in);
506       printf("- FRECUENCIA DE LOS DATOS: %d\n", frec);
507       printf("- CANTIDAD DE PERIODOS: %d\n", period);
508       printf("- VALOR DEL COMPONENTE ESTACIONAL: %d\n", s);
509       printf("- CANTIDAD DE PERIODOS POR DELANTE: %d\n", m);
510       printf("- TOTAL DE DATOS DE LA SERIE DE TIEMPO: %d\n", frec*period);
511       printf("- USO DE PROCESADORES: %d\n", core);
512       printf("\n Reservando Memoria Dinamica...\n");
513       allocate_memory();
514       printf("\n Leyendo serie de tiempo...\n");
515       read_data();
516       printf("\n Ordenando serie de tiempo...\n");
517       tw_vw();
518       parametros();
519       printf("\n Inicio del programa...\n");
520       distribute();
521       printf("\n Salida de archivos...\n");
522       output_file();
523       printf("\n Liberar memoria Dinamica...\n");
524       free_memory();
525       return 0;
526   }
```