UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

Biblioteca

"DISEÑO Y ACELERACIÓN DE NUEVOS PROCEDIMIENTOS DE IDENTIFICACIÓN AUTOMATIZADA DE BLOQUEOS ATMOSFÉRICOS EN EL ATLÁNTICO NORTE"

TRABAJO FIN DE GRADO

Junio -2024

AUTOR: Víctor Hernández Sánchez

DIRECTOR/ES: José Antonio García Orza

Héctor Francisco Migallón Gomis

Biblioteca
UNIVERSITAS Miguel Hernández

"The most damaging phrase in the language is: 'It's always been done that way'."

- Grace M. Hopper, creadora del lenguaje *COBOL* y primera mujer *hacker*.

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

AGRADECIMIENTOS

A mis profesores, por su dedicación y su paciencia.

A todos mis compañeros, por los buenos momentos.

A mis dos tutores, por su ayuda, su colaboración y su comprensión.

A mis padres, por su paciencia y su amor y cariño incondicional.

Y a Paula, mi pareja, por ser mi mayor apoyo y por ser la pieza que completa mis puzles.

A todos ellos, gracias por formar parte de este proyecto.

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

RESUMEN

Un patrón de bloqueo atmosférico es un patrón anómalo de vientos en la alta troposfera que se caracteriza por la obstrucción del flujo típico debido a una fuerte deformación meridional localizada al oeste de los centros de presión que provocan el bloqueo. El resultado es un movimiento casi estacionario, o incluso retrógrado, en el área situada al este de esos centros de presión.

Un área extensa con elevadas presiones (una dorsal o un anticiclón cerrado) situada al norte de un área de bajas presiones, se denomina patrón de bloqueo "Rex". Uno de los dos centros de presión puede tener menor extensión e intensidad que el otro, pero en cualquier caso provocan al oeste una deformación del flujo que tiende a rodear a estos sitemas por el norte y/o por el sur, quedando al este la situación de bloqueo. Otro patrón relevante es el del bloqueo "Omega", formado por un sistema de altas presiones hacia el polo con dos bajas presiones colocadas más hacia el ecuador, una al este y otra al oeste del área de altas presiones. La deformación en el oeste es mucho más compleja y al este se tiene igualmente el bloqueo de los flujos atmosféricos.

Los trabajos sobre bloqueos y sus efectos sobre la Península Ibérica y el resto del continente europeo están basados en las diferencias de masa atmosférica que aparecen entre latitudes altas, medias y bajas, así como por los vientos muy reducidos del oeste (que pueden llegar a proceder del este), y están basados en el índice de bloqueo de Tibaldi y Molteni [14] o versiones adaptadas de él. Son métodos en los que se calculan gradientes meridionales de los campos de altura geopotencial a 500 hPa integrados en intervalos longitudinales.

En este trabajo se ha desarrollado un nuevo procedimiento basado en la identificación de los centros de presión usando la altura geopotencial a 500hPa como indicador y aplicando una serie de filtros y algoritmos de agrupación y selección. Además, se han estudiado técnicas de computación de altas prestaciones para la aceleración del código y una generación de mapas clara y muy visual, logrando así una caracterización más precisa, rápida y cómoda para el usuario.

Biblioteca
UNIVERSITAS Miguel Hernández

ÍNDICE

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

ÍNDICE DE TABLAS

Biblioteca
UNIVERSITAS Miguel Hernández

ÍNDICE DE FIGURAS

Biblioteca
UNIVERSITAS Miguel Hernández

ÍNDICE DE ALGORITMOS

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

1. Introducción

En este primer capítulo, se pone el foco en qué son los bloqueos atmosféricos y qué tipos existen, qué técnicas se emplean para su identificación, el estado actual de la investigación en este campo y los objetivos de este proyecto.

1.1. Motivación

De entre los diferentes fenómenos que ocurren en la atmósfera, encontramos los bloqueos: patrones de flujo atmosférico a gran escala, casi estacionarios y persistentes, que bloquean el flujo de aire típico del oeste y obliga a la circulación atmosférica a desviarse por sus lados norte y sur [3], provocando así en las zonas afectadas un crecimiento de temperaturas, recalentamiento del aire, reventones térmicos y gotas frías e intrusiones de polvo y de aire en el caso de las zonas próximas al continente Africano sobre Europa.

Se debe tener en cuenta que estos bloqueos se pueden generar en otras partes del mundo y se podrán provocar una gran variedad de otros fenómenos; en este trabajo se considerará específicamente la zona Euro-Atlántica.

Es por todo esto que el estudio de estas estructuras es importante, ya que un mayor conocimiento y comprensión de cómo se forman e interactúan y disponer de técnicas de cómo identificarlas y reconocer los patrones típicos que suelen aparecer, permitirá que seamos capaces de predecir cuándo y dónde ocurrirán, las posibles consecuencias que se producirán y las diversas alteraciones que se encuentran en las capas inferiores y superiores de la atmósfera.

Estas estructuras se pueden clasificar en dos tipos principales: los bloqueos Omega y los bloqueos Rex. Los primeros se caracterizan por presentar un área de altas presiones, junto con dos áreas de bajas presiones a ambos lados y en una latitud inferior, formando una estructura en forma de omega (Ω). Los segundos, por otro lado, presentan un área de altas presiones, junto con otra de bajas presiones en una latitud inferior y cercana en longitud al área máxima. En este caso, la estructura se

asemeja a un dipolo.

A la hora de identificar de forma precisa y correcta los bloqueos atmosféricos, no hay un buen método ni un programa claro o de referencia que se use. Sí aparecen diversos algoritmos, algunos adaptados o derivados de otros, pero con criterios demasiado arbitrarios y principalmente basados en la estadística. Ni siquiera hay un criterio único que sea común a los diferentes artículos de investigación que indagan sobre el tema.

Debido a esto, se ha considerado que un buen punto de partida para lograr avances en este campo es tratar de encontrar técnicas nuevas que logren mejores resultados, algoritmos basados en la identificación de procesos físicos, idear pruebas sólidas que confirmen el correcto funcionamiento y, finalmente, la creación de un programa informático usable y cómodo, capaz de facilitar y procesar toda esta información, en tiempos reducidos y con un resultado comprensible y fiable.

Biblioteca
UNIVERSITAS Miguel Hernández

1.2. Objetivos

El objetivo general del proyecto es la identificación automática de bloqueos atmosféricos de interés por sus efectos en la meteorología, desarrollando nuevos algoritmos e implantándolos en una solución software computacionalmente eficiente y acelerada mediante técnicas de paralelización. Así mismo, los objetivos específicos que se plantean para este trabajo se pueden agrupar en cuatro principales categorías: funcionalidad, optimización, escalabilidad e integración y comodidad.

- **Funcionalidad:** Se busca un marco teórico adecuado y robusto, donde todo esté bien estructurado, con criterios claros y con las mínimas arbitrariedades posibles. Además, se busca codificar algoritmos que sean capaces de llevar a la práctica estas suposiciones teóricas, con la máxima eficiencia posible. Los hitos de esta categoría son:

  - *Algoritmos de reconocimiento de bloqueos mediante codificación en lenguaje C.*
  - *Generación de archivos de resultados intuitivos.*
  - *Generación de mapas completos mediante el uso de Python.*
  - *Creación de animaciones y secuencias en bucle (gif) de los mapas generados para facilitar el análisis temporal.*

- **Optimización:** Se requiere que el programa tenga tiempos de ejecución asumibles y que generen una experiencia de usuario positiva. Los tres hitos fundamentales encontrados en este objetivo son:
  - *Código secuencial optimizado.*
  - *Pruebas rigurosas para la validación de resultados.*
  - *Paralelización para sistemas de memoria compartida, distribuida e híbridos, empleando una combinación de OpenMP y MPI.*
- **Escalabilidad:** Se requiere que, tanto el programa en su conjunto como ciertos componentes y elementos que hay en él, se puedan modificar, adaptar a nuevas situaciones, añadir restricciones adicionales o desplazar a otras zonas del planeta. Para ello, los hitos esenciales que se plantean son:
  - *Documentación del código correcta, completa y explicativa.*
  - *Alta modularización del código fuente.*
  - *Flexibilidad de algoritmos y variables.*
  - *Eficiencia en el uso de recursos.*
- **Integración y comodidad:** Se requiere un entorno simple que permita ser usado a partir de unas entradas sencillas, sin tener que preocuparse de mover archivos o de realizar pasos extra, todo integrado en un solo comando o selector de opciones. Para lograr esto, se plantean los siguientes hitos:
  - *Creación de un Command Line Interface funcional, cómodo y simple.*
  - *Lograr un programa "caja negra" para el usuario.*
  - *Sencillez de uso y fácil aprendizaje.*

1.3. Estado del arte

A la hora de analizar el estado actual del campo, se observa una tendencia en las técnicas, dentro de la gran disparidad de planteamientos teóricos y estudios. En general, la descripción de un fluido en movimiento se realiza de dos formas distintas: se consideran los campos del fluido en una zona del espacio y se estudia el estado de variables como velocidad, temperatura, presión o altura geopotencial en distintos instantes de tiempo; o se identifican partículas del fluido y se siguen en el tiempo, analizando su trayectoria y cómo las propiedades cambian al ir cambiando de posición. A la primera se le denomina descripción euleriana, y a la segunda descripción lagrangiana.

Las dos descripciones son complementarias y se utilizan en aplicaciones diferentes de la meteorología y se usan frecuentemente mapas en los que se representan campos meteorológicos. Para estudiar las situaciones de bloqueo, hay diversidad de criterios básicos en la elección de las variables utilizadas. Se utilizan los campos de velocidad, sobre los que se calcula la vorticidad, se utiliza la presión en superficie, o la altura geopotencial a presiones concretas. Sobre estos campos se busca aplicar algún tipo de técnica estadística como métodos de agrupación o clustering, técnicas de ponderaciones y árboles de decisiones o índices dinámicos.

Si se revisan los estudios previos para la identificación de bloqueos, no se encuentran investigaciones del método lagrangiano, ya que la mayoría de los estudios se basan en la descripción euleriana. Para nuestro caso, se ha decidido seguir la descripción euleriana por ser la más común y la que más se ha estudiado, así como por poder analizar los datos de forma más sencilla y rápida y tener más precisión por no tener que depender de derivadas, ya que, para trabajar con ellas, se deben realizar aproximaciones y, en consecuencia, se produce una pérdida de precisión.

Para el estudio y la investigación de los diferentes bloqueos de una forma euleriana, se encuentra el artículo *Occurrence and transition probabilities of omega and high-over-low blocking in the Euro-Atlantic region* [3], basado en el Índice de Bloqueo Instantáneo de Tibaldi y Molteni [14], el cual es un índice de bloqueo bi-

nario unidimensional que determina Longitudes Instantáneas Bloqueadas (Instant Blocked Longitudes *IBLs*) para cada instante de tiempo del conjunto de datos. Su funcionamiento se basa en identificar bloqueos en términos de gradiente de altura geopotencial con respecto a una Latitud de Bloqueo Central de Referencia (Central Reference Blocking Latitude *CRBL*). Usualmente se aplica con el framework FREVA [9], desarrollado como una solución de software para aplicar el anteriormente mencionado índice y realizar estudios y mediciones de altas prestaciones sobre modelos del sistema terrestre y su climatología derivada.

En este mismo estudio [3], se investigan las posibilidades de formación, cambio, transición y desaparición de los bloqueos. Se aplican técnicas de regresión para investigar cambios a largo plazo en las probabilidades, aplicando también una regresión multinominal a tres niveles que describe las diferentes tendencias de los bloqueos. Finalmente, se estudia cual de los dos modelos de Markov estudiados describen mejor las probabilidades de transición; uno de dos estados, bloqueo o no bloqueo, y otro de tres estados, no bloqueo, bloqueo Omega, bloqueo Rex.

Los diversos parámetros que aplican son un grid de 2.5^{o} latitud x 2.5^{o} longitud de espaciado entre valores y fijan el nivel de presión a 500hPa, ya que es el punto donde la divergencia del campo horizontal del viento es próximo a cero [12]. Además, como principales variables de estudio, usan las componentes horizontales del viento (U, V), que son las proyecciones del vector viento en un sistema de coordenadas cartesianas horizontales. Estas componentes permiten descomponer el viento en dos direcciones perpendiculares entre sí, lo que facilita su análisis y representación. La componente U representa la dirección Este-Oeste y la componente V representa la dirección Norte-Sur.

Para identificar patrones de bloqueo, aplican el método del trapezoide [10] [8]. Esta aproximación usa aspectos de cinemática de flujo para identificar los vórtices y también aplica teoría de vórtices para determinar las propiedades del bloqueo y clasificarlo. En una proyección regular de latitud y longitud, el bloqueo tipo Rex tendrá una forma más parecida a un rectángulo, a diferencia del Omega, que presentará una forma más trapezoidal y los dos sistemas de bajas presiones haciendo de base del trapecio.

Tras analizar este trabajo, podemos extraer que el método que se ha usado produce un buen resultado, ya que permite identificación y seguimiento de ambos eventos. Sin embargo, al ser métodos muy basados en la estadística, brindan curvas de probabilidad e inexactitudes, así como la posibilidad de casos no ideales o que se desvíen del estándar, pudiendo provocar resultados imprecisos. Además, en ningún momento se habla de software o aplicaciones usadas, así como tampoco se especifica tiempos o rendimiento de la generación, lo que dificulta mucho el uso para la comunidad. Es por esto que, junto al principal problema de la arbitrariedad y el análisis estadístico inexacto, se decide no avanzar por esta rama, buscando así procesos más exactos y deterministas.

Biblioteca
UNIVERSITAS Miguel Hernández

2. Marco Teórico

En esta sección se estudian los aspectos a tener en cuenta para poder sentar unas bases fundamentadas previas al diseño de los algoritmos y la implementación del software de identificación de bloqueos. Entre otras cosas, se observan las características clave seleccionadas para el modelo planteado, se discuten sobre los diferentes métodos de filtrado que existen y que han sido investigados en el trabajo y, finalmente, se describen los tipos de bloqueo existentes, así como sus principales características y cómo reconocerlos.

2.1. Análisis del problema

En este trabajo, se busca un procedimiento más cercano a los elementos meteorológicos que provocan el bloqueo: la detección de los centros de acción (áreas de altas presiones y áreas de bajas presiones) que ubicados en posiciones relativas específicas de unos respecto de otros crean las configuraciones en Omega y Rex. Para ello, deben identificarse los puntos donde hay mínimos y máximos en el campo Z, filtrarlos de acuerdo a ciertos criterios que los identifican y buscar las configuraciones ligadas a los bloqueos.

Al analizar este problema detenidamente, se detectan una serie de complicaciones y factores que pueden ser críticos. Se encuentran dificultades a nivel teórico para trazar el problema; tanto a la hora de seleccionar variables para el estudio, como por ejemplo si se debe usar la vorticidad con las componentes del viento o la altura geopotencial o, también, a qué altura de la atmósfera se analizarán los campos seleccionados.

Además, se identifica otro aspecto donde prestar especial atención y es en las decisiones de programación y los enfoques de los algoritmos, ya que dependiendo de qué filtros sean aplicados, con qué restricciones y cómo sean tratados los datos, se podrán obtener resultados con mayor o menor ruido y más o menos adecuados para el marco teórico propuesto.

2.2. Procesamiento inicial de los datos

El primer paso es obtener los datos necesarios. En el portal web de Copernicus [4], se pueden descargar los datos de Reanalysis ERA5 [7], que es la base de datos de mayor resolución espacial (0.25^{o}) y temporal (desde 1940 en intervalos de 1 hora) en 137 niveles de altura. De ellos se puede extraer una representación detallada y coherente de la atmósfera, facilitando así el estudio de la dinámica atmosférica y ... de las situaciones de bloqueo.

Para escoger los datos necesarios, se debe seleccionar la variable de interés y un nivel de presión. Los siguientes parámetros son año, mes, día y hora para fijar el momento concreto que queremos estudiar. Generalmente se descargan los datos de uno o varios días (hasta semanas) y los incrementos de tiempo que sean necesarios para un correcto seguimiento, dependiendo del caso y de la virulencia del bloqueo.

Finalmente, se puede acotar la zona geográfica, que por defecto está en el mapa del mundo completo. Para la descarga, se selecciona el formato NetCDF (Network Common Data Form). También se podría escoger GRIB (Gridded Binary); a pesar de ser ambos ficheros binarios, NetCDF es ampliamente usado en la investigación climática y ambiental debido a su flexibilidad para almacenar datos multidimensionales y complejos, su robusto soporte para metadatos y su amplia compatibilidad con diversas herramientas y lenguajes de programación científicos. Además de esto, se escoge usar NetCDF [15] ya que las librerías, tanto para Python como para C, están optimizadas y mejor preparadas, ahorrando así mucho tiempo y permitiendo mejoras significativas en la velocidad y comodidad del procesamiento de los datos.

Posteriormente a esto, se requerirá crear una cuenta si no estuviera ya creada e iniciar sesión para mandar la petición de descarga al servidor. Una vez esté lista, se podrá descargar el archivo si la solicitud es aceptada.

2.2.1. Altura geopotencial y sus niveles

Para analizar los bloqueos atmosféricos, la elección de la variable a investigar es crucial, ya que la forma de trabajar y los métodos a utilizar serán muy diferentes. Si se quiere tomar un enfoque distinto con respecto a los trabajos actuales, pero sin salir de las variables eulerianas, la elección de la vorticidad, calculada a partir de las componentes bidimensionales de la velocidad del aire puede ser descartada. No se debe prescindir únicamente por ser el método más habitual, también se encuentra un motivo más relevante para no usarla y es el propio cálculo de las vorticidades. Para poder extraerlas y realizar operaciones, es necesario calcular las derivadas de las componentes del viento y, además, introduce desviaciones y márgenes de error con esos cálculos, lo cual no es deseable para un modelo fiable y completamente funcional.

Estando descartada la vorticidad, se puede justificar el uso de la altura geopotencial como indicador por tres principales motivos: el primero es que, en ambos hemisferios, los valores y mediciones se realizan de la misma forma, cosa que no ocurre si se usa vorticidad, ya que para representar los giros en una u otra dirección, se deberá cambiar el signo de las componentes del viento. El segundo motivo es que su uso es generalizado y está extendido entre la comunidad de físicos que investigan sobre campos atmosféricos, por lo que facilita la comprensión visual. Finalmente, el tercer y último motivo es que los centros de acción de los máximos y los mínimos se representan con mucha claridad y se pueden asociar a los valores detectados con vorticidad, por lo que los bloqueos se pueden identificar de igual forma, con el añadido de obtener un campo más suavizado y sin posibles errores procedentes de las derivadas.

Una de las justificaciones que se encuentran para fijar la altura geopotencial a un nivel de 500hPa de presión es tratada en el trabajo de Schielicke [12], donde se explica que esta altura es perfecta ya que es un nivel en el que la divergencia del campo horizontal del viento es cercana a cero. Sin embargo, ya que se decide descartar la vorticidad, y, por tanto, el uso de las componentes del viento como variables de estudio (U, V), podemos justificar que para la altura geopotencial (Z) hay dos

motivos más importantes y destacables: el primero es que a ese nivel nos encontramos en la troposfera a una altura media, justo en la zona donde la mitad de la masa de la atmósfera se encuentra por debajo y la otra mitad por encima. Esto es muy interesante, ya que se pueden apreciar reflejados todos los procesos físicos influyentes, que se producen bien en las capas superiores o bien en las capas más inferiores. El segundo es que queda por encima justo de la gran mayoría de terrenos elevados y montañas de la superficie terrestre, por lo que el terreno pasa a no ser tan influyente ni determinante, evitando problemas de datos irreales o modificados incorrectamente.

2.2.2. Archivos NetCDF

Una vez han sido obtenido los datos y almacenados en ficheros con formato .nc (NetCDF), configurados para el uso de la altura geopotencial y a 500hPa de presión, antes de ser procesados, podemos utilizar el visualizador de Panoply [6]. Este software desarrollado por la NASA sirve para inspeccionar estos archivos y permite confirmar que la descarga de los datos es correcta y comprobar el formato que usan, para que después puedan ser procesados correctamente.

Para la configuración usada, cada variable tiene sus propios tipos y límites y es vital ser consciente de ello para tratar los datos correctamente.

Variable	**Tipo**	**Unidades**	**Tamaño**	**Extras**
Time	int	horas desde 01/01/1900	T	No
Latitude	float	grados al norte	721	No
Longitude	float	grados al este	1440	No
Z	short	m^2/s^2	T, 721, 1440	Scale factor, offset

Tabla 2.1: Distribución de las variables y sus características en el archivo NetCDF.

Tal como se aprecia en la tabla 2.1, se obtiene toda la información que se usará para aplicar en el modelo las entradas y sus tipos y también para tratar, almacenar y operar con los datos del archivo. La tabla corresponde a un mapa completo (con todas las latitudes y longitudes). La variable T será el número de instantes de tiempo: para 1 día completo de 6 en 6 horas, $T = 4$. Scale factor y offset son valores que

habrá que aplicar a Z para transformarlo de short a double de la siguiente manera: $(Z \times scale\_factor) + offset$; esto se usa para almacenar en short y ahorrar en espacio de almacenamiento y mejorar la eficiencia de cálculos que no necesitan ser realizados con valores en double.

2.2.2.1 Instalación de librería para C

Para la instalación de la herramienta NetCDF y el uso de su librería para C, se seguirá la guía del Anexo A, basada en el repositorio de Danwild [1]. Esta guía de su repositorio de GitHub se encontraba algo obsoleta y algunos enlaces a descargas habían sido migrados o modificados, por lo que se ha adaptado en este trabajo para su uso en la actualidad.

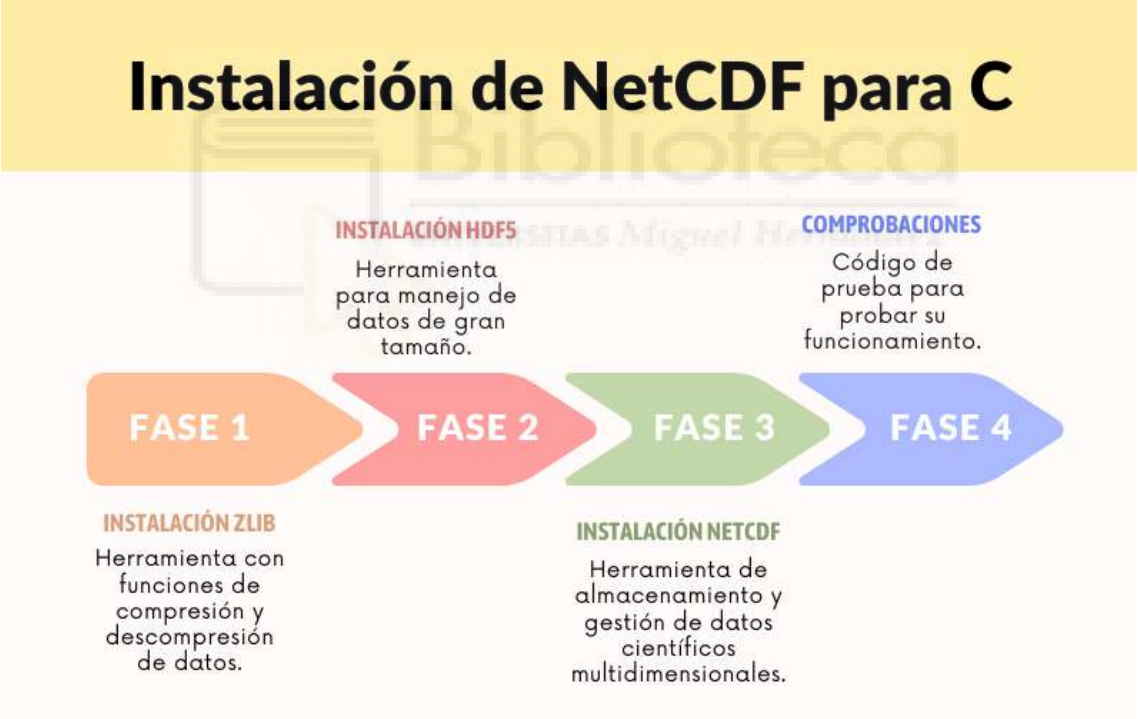

Figura 2.1: Fases para instalar la librería de netcdf.h en un sistema UNIX.

Tal como se resume en la figura 2.1, la guía tiene 3 fases de instalación y una última de comprobaciones. Esta guía está preparada para instalar todo en sistemas Linux basados en Debian, pero los pasos son similares para cualquier otro Sistema Operativo o versión, cambiando únicamente la forma de descarga e instalación.

2.3. Tratamiento y estandarización

A partir de los datos en crudo, se debe realizar una estandarización según el criterio usado para la latitud y la longitud. Hay que asegurarse de realizar una comprobación y, posteriormente, una conversión si fuera necesaria. Generalmente se utiliza el formato: Latitud: (90.00, -90.00) y Longitud: (-180.00, 180.00), siendo la coordenada (90.00, -180.00) el punto del extremo superior izquierdo [1]. En las ocasiones que encontremos los datos con formato: Latitud: (0, 180.00) y Longitud: (0, 360.00), se realizará la conversión siguiente:

$$Latitud\_Corregida = 90 - Latitud$$

$$Longitud\_Corregida = (Longitud + 180)\,\%360 - 180$$

Si se aplica esta fórmula en un bucle para cada elemento de las latitudes y las longitudes, se obtendrá la conversión, sin olvidar de realizar el cambio también en la matriz de Z, que tendrá dimensiones (latitud, longitud).

Una vez aplicada la conversión de los datos, se deberá comprobar si el rango de latitudes y longitudes de los datos de Z es para todo el globo o para una zona geográfica concreta, ya que, por defecto, se tendrá el mapa completo, pero en ciertas ocasiones se querrá solamente el mapa de una determinada zona. Para estos casos, hay dos opciones; la primera de ellas, es descargar el mapa directamente recortado y se procesa igual que si fuera completo, pero con los límites cambiados. Como segunda opción y más versátil, se podrá descargar el mapa a tamaño completo y posteriormente indicar el área que se requiera analizar. Al indicar esto, solo se procesará y tendrá en cuenta la zona marcada, pero en posteriores ejecuciones, podrá ser modificado a gusto del usuario.

[1] Teniendo en cuenta que se usará una representación de la superficie terrestre sobre una proyección rectangular plana, estando el Norte en la parte superior, como, por ejemplo, en la proyección de Mercator.

2.3.1. Zona de estudio

Si bien es cierto que la identificación de bloqueos atmosféricos puede realizarse para todo el globo, la zona de estudio debe acotarse, recortando y adaptando las dimensiones según se requiera. En general se estudia la zona del Atlántico Norte y el resto de la zona europea, siendo estas las áreas con mayor frecuencia de bloqueos; sin embargo, se ha creído más conveniente el estudio de la totalidad de longitudes, ya que formaciones y elementos en zonas más hacia el continente americano o hacia la zona asiática, pueden ser resultado u origen directo de las agrupaciones encontradas en la parte oeste europea.

Como se aprecia en la figura 2.2, mapa con fecha 26 de junio de 2019 a las 00:00:00 UTC. Además, presenta latitud y longitud completa y resaltando contornos de altura geopotencial a 0.25º y los grupos de puntos máximos y mínimos a resolución de 1.25º. Las zonas de más interés van desde una latitud inferior a 85º hasta latitudes de entorno a 25º; por lo que se fijará en ese rango de latitud el análisis y la búsqueda de los sistemas de bloqueo.

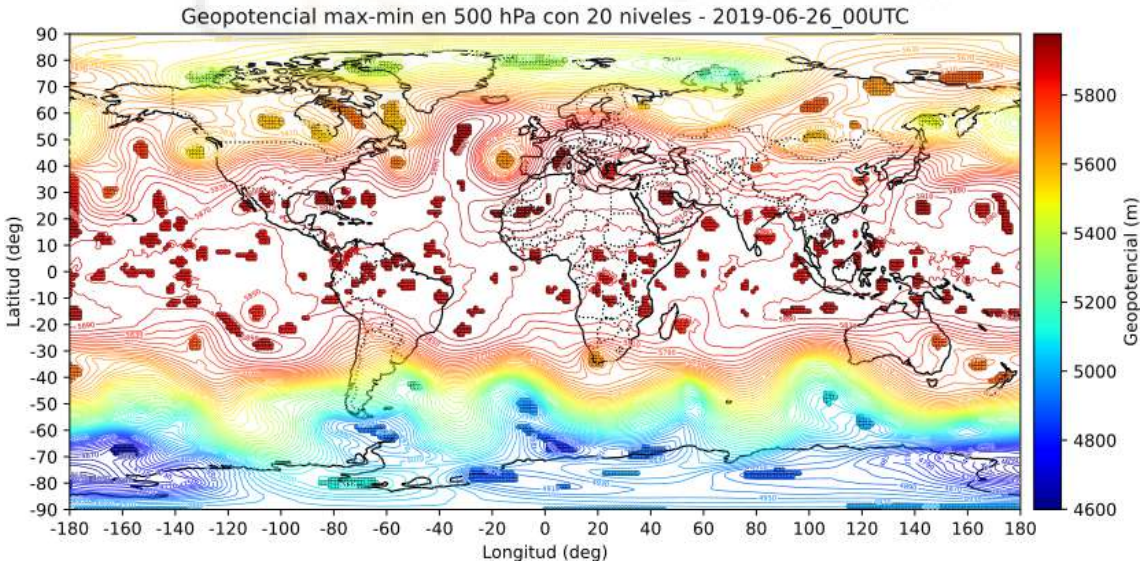

Figura 2.2: Mapa completo de latitud y longitud de grupos de puntos máximos y mínimos.

2.3.2. Resolución espacial de los campos meteorológicos

Para seleccionar la resolución, primero se debe tener en cuenta que los datos descargados de los archivos de NetCDF tienen por defecto una resolución horizontal de 0.25º. Así, a tamaño completo, se trabaja con una matriz de estudio de latitud, longitud con tamaño de 721×1440. Computacionalmente hablando, puede resultar costoso el índice resolución-eficiencia, por lo que para decidir si reducir la resolución original del archivo o no, podemos generar mapas de puntos a diferentes resoluciones y comprobar si la calidad de datos que perdemos es significativa o se puede sacrificar a cambio de ganar velocidad de cómputo. Para las pruebas, se generarán 4 mapas de muestra en la zona adyacente a la Península Ibérica (latitud (30, 55) y longitud (-15, 15)) y fecha 14 de marzo de 2022 a las 00:00:00 UTC con resoluciones 0.25º, 1.25º, 2.5º y 3.75º.

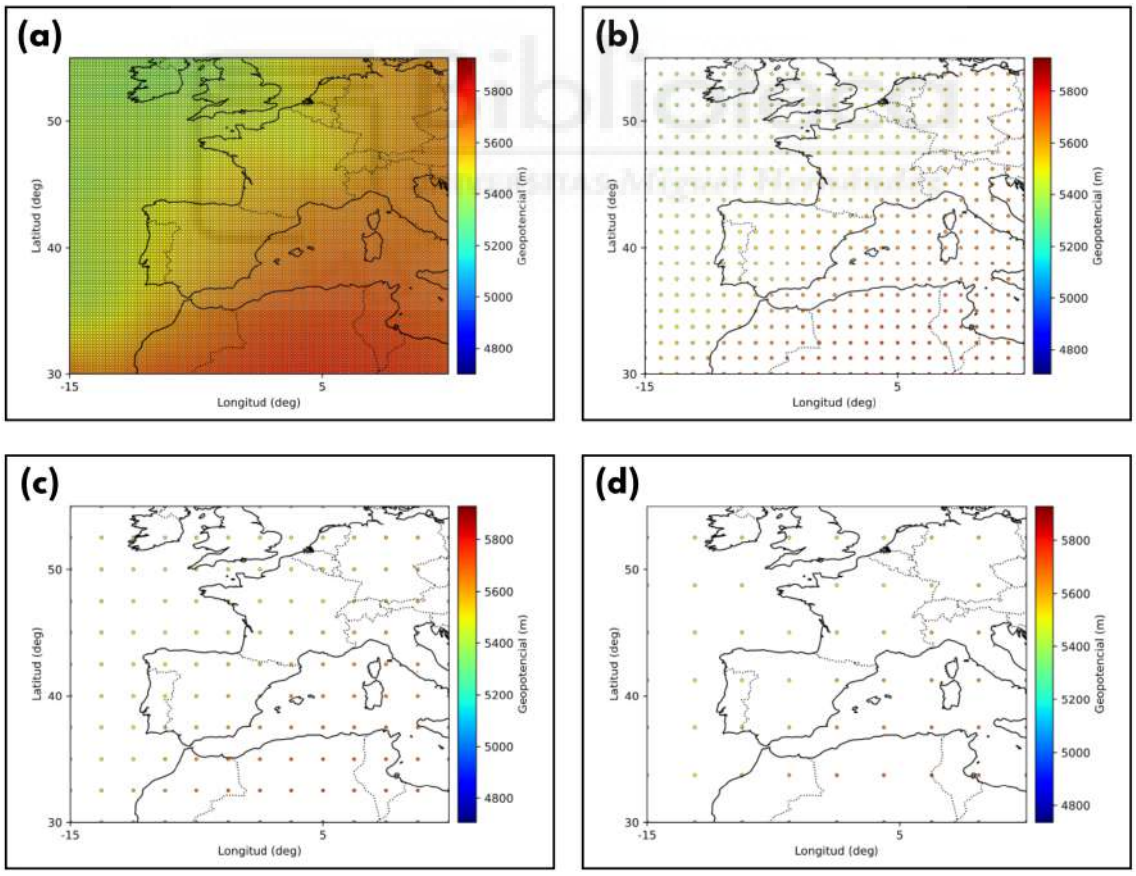

Figura 2.3: Mapas de altura geopotencial (Z) con resolución: **(a)** 0.25º (original del archivo), **(b)** 1.25º, **(c)** 2.5º y **(d)** 3.75º.

Con la resolución por defecto, en la que no hay ninguna pérdida de información, se obtiene una cantidad de puntos que puede llegar a resultar innecesaria, sacrificando tiempo de cómputo por un resultado visual excesivo. La ejecución y procesamiento de este instante se lleva a cabo en un rango de entre dos y tres minutos, por lo que se decide aumentar la resolución en favor de la mejora de tiempos y la eficiencia del programa.

Al estudiar el mapa con una resolución 5 veces menor (1.25^{o}), se aprecia una cantidad de puntos significativamente más reducida, ya que 25 puntos a la resolución original equivale a un punto en resolución 1.25^{o}. El tiempo de cómputo se ha reducido considerablemente, pasando a unos 8-10 segundos de ejecución. A pesar de la reducción de puntos, se sigue pudiendo observar el gradiente de intensidad que se nos muestra en el mapa original, por lo que esta opción se considera mejor alternativa.

Para la resolución siguiente, el doble de resolución que la anterior (2.5^{o}), no se puede apreciar una mejora importante de velocidad de ejecución, pasando esta a un rango de entre cuatro y seis segundos. Sin embargo, en cuanto a la visualización, se sigue observando el gradiente del primer mapa, pero de una forma mucho más diluida. Si se llegara a elegir esta resolución, podría provocar ciertos problemas en las formaciones con una figura más redondeada o alargada, generando picos puntiagudos o perdiendo información de zonas de interés.

Finalmente, si se vuelve a ampliar la resolución, ahora con 15 veces menor que en la del mapa original (3.75^{o}), el tiempo de ejecución vuelve a mejorar, pero de una forma mucho menos apreciable, pasando a un tiempo de aproximadamente tres segundos. En cambio, el principal problema de elección de este mapa es que apenas es perceptible el gradiente encontrado con resolución 0.25^{o}, lo cual indica una pérdida de datos importante, imposibilitando en algunas ocasiones el reconocimiento de estructuras. Con todos estos datos, se puede formar una tabla comparativa para clarificar y decidir la mejor opción.

Resolución	Tiempo de ejecución
0.25^{o}	2-3 min
1.25^{o}	8-10 s
2.5^{o}	4-6 s
3.75^{o}	3 s

Tabla 2.2: Tabla comparativa de elección de resoluciones de mapa.

Gracias a todo este análisis de la figura 2.3 y de la tabla 2.2, se decide finalmente que la resolución más óptima para este estudio es: 1.25^{o} latitud x 1.25^{o} longitud. El tamaño típico de los centros de acción (altas y bajas presiones) es un factor relevante y a lo largo del proyecto se ha comprobado que esta elección es la adecuada.

2.3.3. Cálculo de distancias

Otro factor a tener en cuenta además de la resolución y la zona de estudio es la forma de calcular las distancias entre puntos en la superficie terrestre. Si tuviéramos dos puntos (x_1, y_1) y (x_2, y_2) en un espacio euclídeo de 2 dimensiones, la forma de calcular la distancia (d) sería sencilla, puesto que se podría calcular de la siguiente forma:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Esto no puede aplicarse, puesto que la superficie terrestre no es plana y cuenta con 3 dimensiones. Para solventar este problema, se utilizan aproximaciones que facilitan los cálculos y permiten obtener estas distancias cómodamente. Se decide usar la fórmula de haversine [2] ya que permite calcular la distancia entre dos puntos de una esfera con un margen de error despreciable para el ámbito de este proyecto. Esta fórmula es:

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

donde:

- d es la distancia entre los dos puntos a lo largo de un *círculo grande* [11].
- r es el radio de la Tierra.
- ϕ_1 y ϕ_2 son las latitudes del primer y segundo punto, respectivamente (en radianes).
- λ_1 y λ_2 son las longitudes del primer y segundo punto, respectivamente (en radianes).

2.4. Algoritmos de procesamiento de imágenes

Durante la fase de investigación de este trabajo, se desarrolló un procedimiento con un enfoque diferente al filtrado y reconocimiento de formaciones; consiste en procesar cada mapa para cada instante de tiempo con métodos de filtrado y suavizado de imágenes. Este método desarrollado y analizado durante el trabajo es el detector de bordes de Sobel [16], que se basa en operaciones de cada píxel (en este caso, los correspondientes valores de Z), para obtener una aproximación al gradiente de intensidad de la imagen.

Generalmente, se parte de una imagen en escala de grises, a la que se le aplica un filtro de suavizado y, finalmente, se aplica el detector de bordes de Sobel. Se pueden utilizar varios métodos, pero suelen estar organizados en dos categorías: métodos de gradiente y métodos Laplacianos. Los métodos de gradiente usan la primera derivada de la imagen para encontrar máximos y mínimos; en cambio, los métodos Laplacianos se basan en la búsqueda de cruces de ceros en la segunda derivada.

El método probado fue la detección por máximos y mínimos con gradiente, debido a que el método Laplaciano podía generar errores o zonas menos claras, al hacer uso de la segunda derivada. En cualquier caso, las pruebas no fueron consideradas suficientemente fructíferas y se decide descartar esta aproximación de detección de la fluctuación de campo de Z.

2.5. Clasificación mediante medias

Otro método estudiado fue el de clasificar los puntos mediante la media del geopotencial de los vecinos. Gracias a este algoritmo, se consiguió un cambio de visión y de concepto, ya que aplicando esto a vecindarios más grandes y al comparar el resultado con el punto central, se llegaban a discretizar muy bien las distorsiones del campo. Aún con todo, no funcionaba para todos los casos de estudio. Tal como podemos observar en los mapas de la figura 2.4, se estudian los puntos de dispersión con resolución de 2º en **(a)**, **(b)**, el mapa con fecha 14 de marzo de 2022 a las 00:00:00 UTC y en **(c)**, **(d)**, el mapa con fecha 26 de junio de 2019 a las 00:00:00 UTC. En el mapa básico del episodio de 2022, se podía visualizar perfectamente la curva de puntos filtrados coincidente con la curva de los máximos. Sin embargo, en el episodio de 2019, no se lograba apreciar, ya que su geopotencial medio era mucho más elevado.

Se probó a restringir niveles de contornos, con lo que se afinaba más la forma para algunos casos, obteniendo resultados prometedores, pero finalmente se acaba descartando esta idea, puesto que dependía mucho de la media general del mapa estudiado y porque para analizar las curvas de puntos medios generadas, se necesitaba aplicar algún filtro extra de *antialiasing*, parecido a los usados en pasos de señales analógicas a digitales, pero todo esto era demasiado dependiente de la latitud donde se desarrollan los centros de acción, por lo que se terminó decidiendo no avanzar en esa dirección.

Finalmente, se realizaron unas últimas pruebas con el filtro de vecinos para identificar máximos y mínimos no locales basadas en la comparación con la media de los vecinos más próximos como un criterio más débil que permite pequeñas fluctuaciones alrededor de un posible candidato. Esto seguía resultando demasiado dependiente de la media global del mapa, además de que el criterio era necesario pero no suficiente para todos los puntos. Esto acabó determinando que lo mejor sería prescindir de su uso, ya que el ajuste necesario para conseguir los suficientes puntos de interés, al mismo tiempo que se mantenían los niveles de ruido y puntos innecesarios al mínimo, era demasiado arbitrario y sin un criterio claro que se adaptara al modelo deseado.

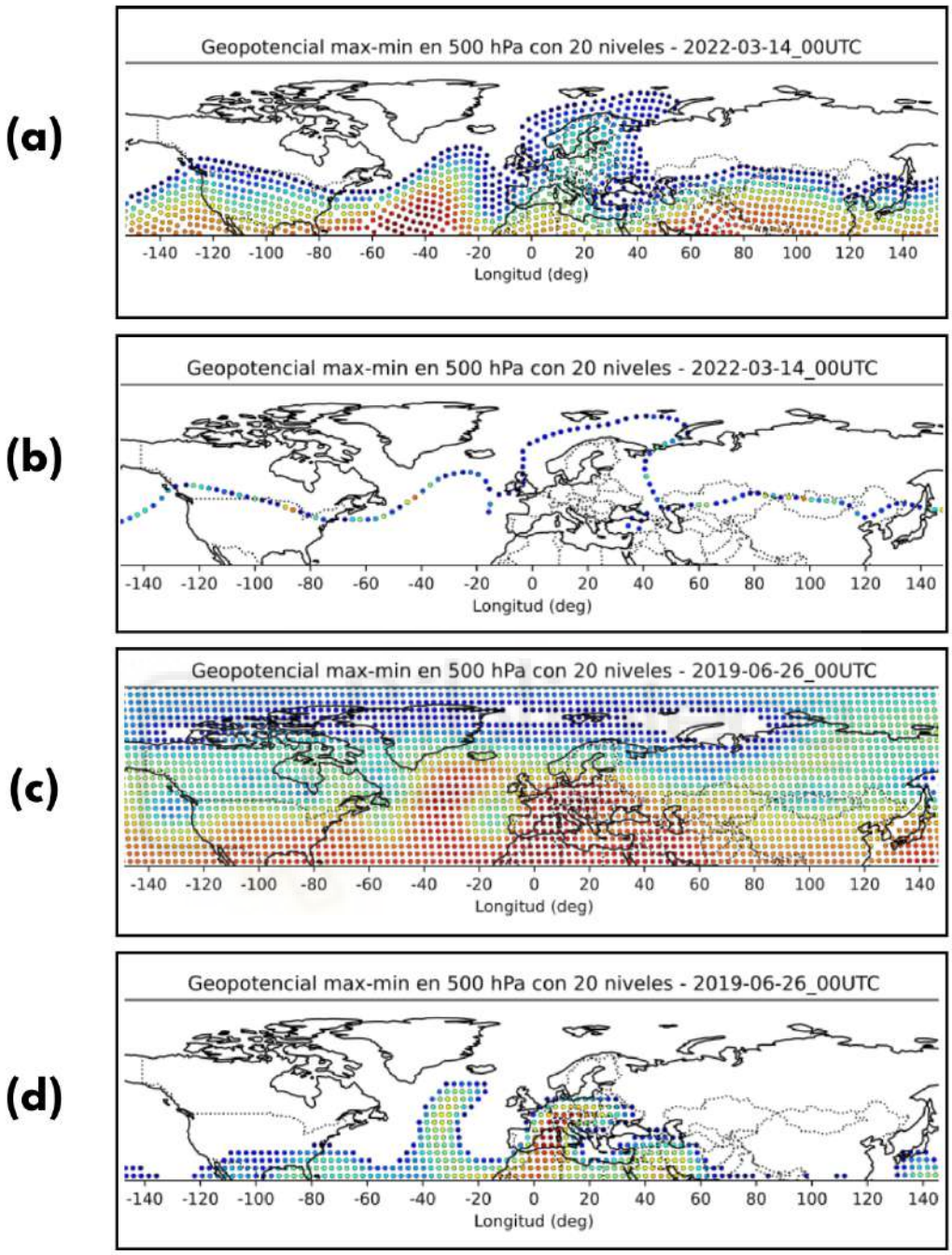

Figura 2.4: **(a)** Valores superiores a la media del vecindario de 21x21 vecinos. **(b)** Valores iguales a la media del vecindario de 21x21 vecinos, obteniendo así la frontera. **(c)** Valores superiores a la media del vecindario de 21x21 vecinos. **(d)** Mismos valores que el mapa anterior, pero limitando los contornos de valores de Z que pueden aparecer.

2.6. Filtrado de los datos

A partir de los datos ya preparados y estandarizados, la decisión de utilizar una serie de filtros para eliminar los niveles de ruido en zonas donde el gradiente de Z es pequeño, así como la información no relevante y la información no relevante del archivo de datos en crudo es crucial para la identificación de los puntos que son máximos o mínimos. Esta estrategia mejora significativamente la comodidad y la velocidad del manejo de datos, ya que se trabaja con agrupaciones de datos más pequeñas, lo que acelera el procesamiento.

2.6.1. Filtro de vecinos

En las publicaciones científicas, hay un consenso para la identificación de máximos y mínimos locales aplicando un filtro sobre la matriz de valores de Z de los 8 vecinos más próximos a cada punto del mapa a estudiar. Sin embargo, hemos podido comprobar que, usando tan solo 8 vecinos y aplicando un filtro estricto (todos son mayores o menores que el punto estudiado).

Se obtienen resultados muy buenos, pero solamente en las zonas que representan altas o bajas presiones muy marcadas o en zonas con mucho gradiente de altura geopotencial. Sin embargo, en los casos donde la zona presenta menor variación o no hay cambios significativos, muchos puntos que podían resultar importantes para el estudio y que deberían pasar el filtro, se quedaban atrás.

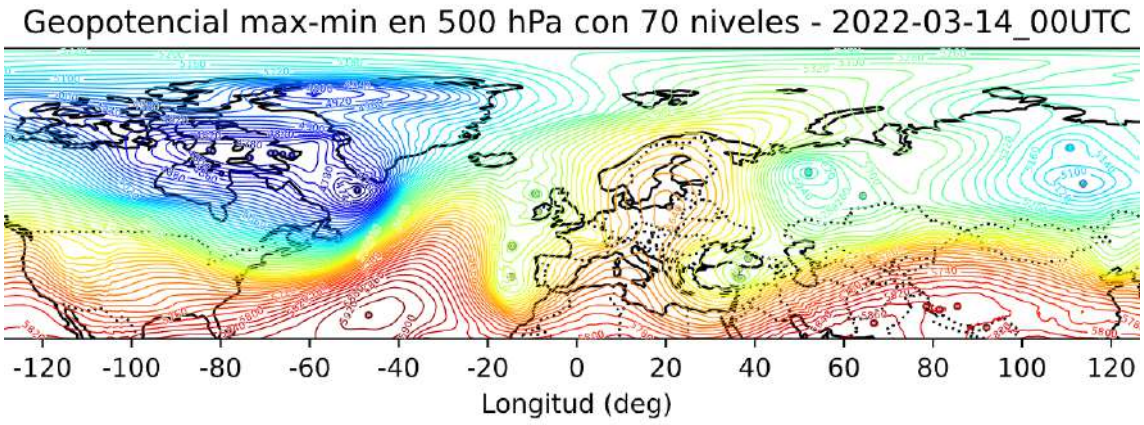

Figura 2.5: Mapa de los puntos filtrados por el proceso de 8 vecinos.

Tal como se observa en la figura 2.5, con un mapa de máximos y mínimos identificados y de líneas de contorno de altura geopotencial, con fecha 14 de marzo de 2022 a las 00:00:00 UTC, si se genera un mapa a resolución 0.25^{o} donde solamente se aplica este filtro, una gran cantidad de puntos no aparecen en las zonas de máximos y mínimos profundos y sólo son apreciables unos pocos en las zonas de mayor o menor geopotencial, en lugar de todos los puntos de esa zona. Esto se produce debido a que en los vecinos próximos hay fluctuaciones de Z que, al generar el mapa, son imperceptibles, pero a la hora de filtrar, se van produciendo descartes por estas variaciones tan insignificantes, limitando mucho nuestro programa de selección y conduciendo a posibles escenarios donde no aparezca ni tan solo un único punto en la zona que se requeriría para identificar una formación.

También se ha estudiado la manera de aplicar los filtros de forma menos restrictiva o de ampliar los vecinos (de una matriz 3x3 a una 5x5 o 7x7, o más). Se relajaron las restricciones para dejar pasar un porcentaje de vecinos que no cumpliera el filtro, o que se encontraran dentro de un rango o dentro del contorno del punto central, pero los resultados con ninguna eran los requeridos, ya que no había un término medio entre muchos puntos que generaban ruido y tan poca cantidad que era imposible una selección correcta.

2.6.2. Filtro por gradiente

El siguiente filtro a aplicar consiste en un filtrado por gradiente (o de bearing) para determinar si un punto es mínimo o máximo no local; se basa en el análisis de cada punto y la comparación con una zona a un determinado número de kilómetros de él en distintas direcciones. En un principio, este iría consecutivo al filtrado de los vecinos, pero al descartarse todas las opciones, se decidió probar únicamente este filtro con los datos en crudo.

Se observó que los resultados eran muy buenos, pero había que intentar reducir al máximo el ruido sin perder puntos de interés, por lo que se fue maximizando el número de puntos de interés encontrados. Se estudió la aplicación de varias distancias máximas con respecto al punto, siendo la inicial y también más usada, los 1000 km.

Tras varias comparativas, se obtenían mejores resultados usando 500 km, ganando muchos puntos de interés.

Además, se probó a aplicar el filtro comparando valores de contorno en lugar de valores de Z absolutos, pero también se acaba descartando esta opción por aceptar demasiados puntos no buscados. Esto provoca un criterio demasiado estricto en la comparativa, puesto que todos los puntos interpolados a 500 km de la zona de gradiente debían ser mayores (o menores, si es para máximo) que el punto central a comparar. Al analizar esto, se prueba a permitir un cierto grado de fallo, admitiendo un pase del punto si al menos un 90 % de los comparados cumplían el criterio. Este valor es el adecuado, puesto que 85 % admitía demasiado ruido y 95 % dejaba fuera puntos de interés en algunos mapas.

Por todo esto, el único filtro aplicado a los datos es el filtro por gradiente, que nos ayuda a obtener nubes de puntos de interés con el mínimo ruido posible.

2.6.2.1 Distancia del círculo grande

La fórmula de la distancia del círculo grande [11] (ya mencionada en la sección 2.3.3), es una fórmula usada para el cálculo de la distancia mínima entre dos puntos en una esfera. Esta se usa como una aproximación para cálculos de distancias y de rutas (como en la aviación por ejemplo) en la Tierra. Usando la aproximación esférica se logran buenos resultados, a pesar de presentar esta una forma de elipsoide.

En cuanto a la aplicación de esta fórmula en el proyecto, dado un punto inicial (ϕ_1, λ_1), una distancia d (en km), y un *bearing* θ (en radianes), las fórmulas para calcular el punto final (ϕ_2, λ_2) son:

$$\phi_2 = \arcsin\left(\sin(\phi_1)\cos\left(\frac{d}{R}\right) + \cos(\phi_1)\sin\left(\frac{d}{R}\right)\cos(\theta)\right),$$
$$\lambda_2 = \lambda_1 + \arctan\left(\frac{\sin(\theta)\sin\left(\frac{d}{R}\right)\cos(\phi_1)}{\cos\left(\frac{d}{R}\right) - \sin(\phi_1)\sin(\phi_2)}\right).$$

donde:

- ϕ_1 es la latitud del punto inicial en radianes,
- λ_1 es la longitud del punto inicial en radianes,
- d es la distancia entre los dos puntos en kilómetros,
- θ es el bearing en radianes,
- R es el radio de la Tierra, aproximadamente 6371 km.

Si se repite esta fórmula para N direcciones, donde en cada dirección modificamos el *bearing*, se pueden obtener todos los N puntos envolventes al central para explorar su gradiente. Para decidir la cantidad de direcciones, se realizaron distintas pruebas sobre diferentes eventos, logrando así encontrar un valor que cubre las máximas direcciones posibles sin provocar una excesiva carga computacional y sin evaluar dos veces la misma dirección. Con esto, se ha obtenido que N debe ser 64.

2.6.2.2 Interpolación bilineal

El método de la interpolación bilineal [13] se usa generalmente en procesamiento y filtrado de imágenes, pero también en algunos campos científicos como la meteorología. Consiste en encontrar el valor estimado de una función en un punto específico, sabiendo los valores concretos de los puntos de su entorno más cercano. Para este trabajo, es necesario aplicarlo a partir del punto generado con la función de círculo grande anterior, ya que este punto no pertenece a la matriz original con resolución de 0.25^{o} y, por tanto, su valor de Z no es conocido.

Para aplicar este método, tenemos como entrada un punto (x, y) y los cuatro puntos que lo envuelven con coordenadas (x_1, y_1), (x_1, y_2), (x_2, y_1), y (x_2, y_2) y con valores Z_{11}, Z_{12}, Z_{21}, y Z_{22} respectivamente; el valor interpolado $Z(x, y)$ se calcula como:

$$\begin{aligned} Z(x,y) = &\frac{(x_2-x)(y_2-y)}{(x_2-x_1)(y_2-y_1)}Z_{11} + \frac{(x-x_1)(y_2-y)}{(x_2-x_1)(y_2-y_1)}Z_{21} \\ &+ \frac{(x_2-x)(y-y_1)}{(x_2-x_1)(y_2-y_1)}Z_{12} + \frac{(x-x_1)(y-y_1)}{(x_2-x_1)(y_2-y_1)}Z_{22} \end{aligned}$$

donde:

- (x_1, y_1) y (x_2, y_2) son las coordenadas de los puntos de la cuadrícula,
- Z_{11} es el valor de altura geopotencial en (x_1, y_1),
- Z_{12} es el valor de altura geopotencial en (x_1, y_2),
- Z_{21} es el valor de altura geopotencial en (x_2, y_1),
- Z_{22} es el valor de altura geopotencial en (x_2, y_2).

Al aplicar la fórmula, se debe tener en consideración que puede llegar a darse el caso de que, en las zonas cercanas a los límites del mapa, a raíz de la aplicación del método del círculo grande, podrían generarse valores de coordenadas inexistentes, por lo que (x_1, y_1) y (x_2, y_2), así como el Z correspondiente, tomarían valor *Undefined*. Es por esto que, antes de aplicar la interpolación, se debe comprobar que los valores sean correctos y, de no ser así, no se procede con el cálculo de la fórmula y la función del programa retorna un valor con el que comprobar que esta situación ha sucedido.

Si se aplica esta fórmula para cada punto válido encontrado con el método de círculo grande, entonces se obtendrá cada valor de Z a 500 km de distancia del punto inicial en distintas direcciones. Gracias a esto, se puede estudiar la variación del gradiente de la altura geopotencial y decidir si el punto de estudio es un punto de interés o no, es decir, si pertenece a una estructura de máximos o de mínimos.

2.6.2.3 Variación de gradiente

Para la parte final de este filtro, se aplica una comprobación del gradiente de Z desde el punto central estudiado hasta cada uno de los otros puntos generados con círculo grande e interpolados. Si al menos un 90 % de los puntos son menores (o mayores en el caso del mínimo) que el punto central, entonces se decide que será un punto máximo (o mínimo) y se almacena como punto de interés. Cuando todos los puntos han sido estudiados de esta forma, se obtiene un mapa con los puntos clasificados como: no seleccionados, seleccionados-máximos y seleccionados-mínimos.

2.7. Uso de agrupaciones de puntos

A partir de los puntos seleccionados, se encuentra que en las zonas de mayor variación espacial, los puntos quedan cercanos entre sí, suficiente como para poder agruparlos y así facilitar la tarea de seleccionar las posibles formaciones.

En un primer momento, se plantea la posibilidad del uso de algoritmos de *clustering*, tales como *k-means* (k-medias) [9] o DBSCAN (*clustering* basado en densidad espacial con ruido) [5]. *K-means* se basa en la división de una cantidad de N puntos en k *clusters*, donde cada punto pertenecerá al grupo del cual el valor medio sea el más cercano. El algoritmo comienza con unos grupos con centroides aleatorios, con los cuales se hace una primera agrupación y, posteriormente, se continúa iterando, calculando los nuevos centroides y añadiendo y excluyendo puntos, hasta encontrar los grupos más adecuados. Este proceso, además, se puede repetir para distinto número de *clusters* k, de manera que, en un entorno de grupos indeterminados, es posible encontrar la cantidad "ideal" de grupos, así como la distribución de los puntos en esos grupos.

Esta idea fue finalmente descartada, puesto que para distintos mapas, las distribuciones de *clusters* serían muy variadas, tanto en número como en cantidad de puntos y posición, pudiendo identificar conjuntos de grupos incorrectos o uniones no adecuadas. Además, es considerado un algoritmo computacionalmente costoso para este proyecto.

Con *k-means* descartado, se estudia la implantación de DBSCAN, un algoritmo algo menos costoso y con el cual se mejora tanto en la forma de los *clusters*, como en la identificación de k grupos sin tener que prefijar el número previamente.

Para este algoritmo, donde dos puntos están en el mismo grupo si son "densamente" alcanzables, se deben definir dos variables; ε y $MinPts$:

- ε define la distancia del vecindario, es decir, la distancia máxima por la que dos puntos son considerados vecinos,
- $MinPts$ define el número de puntos vecinos que se necesitarán para que un punto sea considerado como punto central.

Una vez se definen estos dos parámetros, se comienza a iterar, buscando para cada punto que no haya sido visitado ya, todos los puntos vecinos. Si este número de puntos es mayor o igual que $MinPts$, se considera punto central; si no se cumple, pero entre sus vecinos tiene algún punto conectado que sea punto central, entonces es considerado punto de borde. Finalmente, si nada de lo anterior se cumple, el punto es considerado como ruido y no queda agrupado.

Esta método es muy eficaz para formaciones complejas o alargadas, pero también presenta un problema fundamental: la decisión de los valores de las variables iniciales. Para ε no habría demasiado problema, podría introducirse un criterio basado en cercanía, en valor de Z o incluso uno combinado. La decisión compleja viene con $MinPts$, ya que, para cada mapa y cada instante de tiempo, la cantidad de puntos filtrados es muy variada, pudiéndose encontrar grupos con menos de 4 o 5 puntos seleccionados mientras que otros contienen el triple o cuádruple. Al no haber una uniformidad mínima, ni tan solo dentro del mismo mapa de estudio, se decide descartar también este algoritmo.

Después de analizar el problema con distintos de eventos, se opta por una agrupación mucho más simple y, a su vez, rápida y efectiva. Al tener puntos estrictamente pegados (a menos de 1.25^{o} en cada uno de los lados) y, además, en general una gran separación entre grupos, se puede realizar el *clustering* de la siguiente forma: para cada punto que no pertenezca a un grupo, forma un nuevo grupo y expande el grupo por su vecindario. Al expandir el *cluster*, comprueba sus vecinos más próximos (8 vecinos) y, si cumplen el criterio de cercanía, son del mismo tipo (máximo o mínimo) y aún no tienen grupo, entonces se asignan al mismo grupo que el punto inicial y continúan expandiendo hasta que no quede ningún punto que cumpla los criterios sin agrupar. Con esto tenemos todos los puntos seleccionados, estrictamente pegados y del mismo tipo, agrupados.

2.8. Selección de formaciones de bloqueo

En general, se aplican métodos muy variados, pero uno de los más repetidos es el método de identificación del Trapezoide [10] [8], donde, esencialmente, se aplica al mapa filtrado una caja previamente configurada en forma de trapecio y se busca si hay formaciones de zonas de altas y bajas presiones dentro de la caja. Si las hay y la altura geopotencial (o la vorticidad) son mayores en la caja del centro del trapezoide, se determina una formación Rex, en cambio, si es más fuerte en los laterales de la base, se determina un bloqueo Omega.

Este método resulta arbitrario y poco natural, funcionando de una manera muy dependiente de la forma del bloqueo y su posición, así como de los valores que toman los mínimos. Por ello, se considera que se debe buscar una forma mucho menos arbitraria y más eficiente, ya que se podría considerar un método de "fuerza bruta"de identificación.

Habiendo descartado la selección por variación de latitud y longitud o usando filtros de *antialiasing*, se decidió estudiar la aplicación de una serie de criterios de restricción y descarte por generalidades comunes a todos los bloqueos. Con esto, se descubre que para todos los bloqueos en Omega, debe haber al menos una línea de contorno de Z, que no provoque un contorno cerrado en ninguna de las formaciones, que cruce por la parte superior del área máxima y por ambas zonas inferiores mínimas, formando el propio símbolo de Omega (Ω); además de esto, debe haber un mínimo a cada lado del máximo y ambos deben estar en una latitud inferior a él.

Por su parte, para los bloqueos de tipo Rex, debe haber al menos una línea de contorno no cerrada, que pase por la parte superior e inferior del máximo así como del mínimo y encontrando el contorno por la parte derecha del máximo y por la parte izquierda del mínimo, formando una S; otro criterio importante es que este mínimo debe encontrarse en latitud inferior y de longitud no desviarse más de 10^{o} de la longitud del máximo.

Con todos estos criterios claros, podemos asegurar un buen reconocimiento de las formaciones, quedando descartados todos los posibles falsos positivos.

Biblioteca
UNIVERSITAS Miguel Hernández

3. Diseño e Implementación

Los contenidos de este capítulo se centran en la descripción del programa, su estructura interna, el funcionamiento de los algoritmos implementados, la generación de los mapas y un recorrido completo desde el comando inicial usando el *Command Line Interface* (CLI) hasta la visualización de los resultados. En la figura 3.1, queda mostrado el funcionamiento del programa al más alto nivel y sin entrar en detalle en cada uno de los apartados.

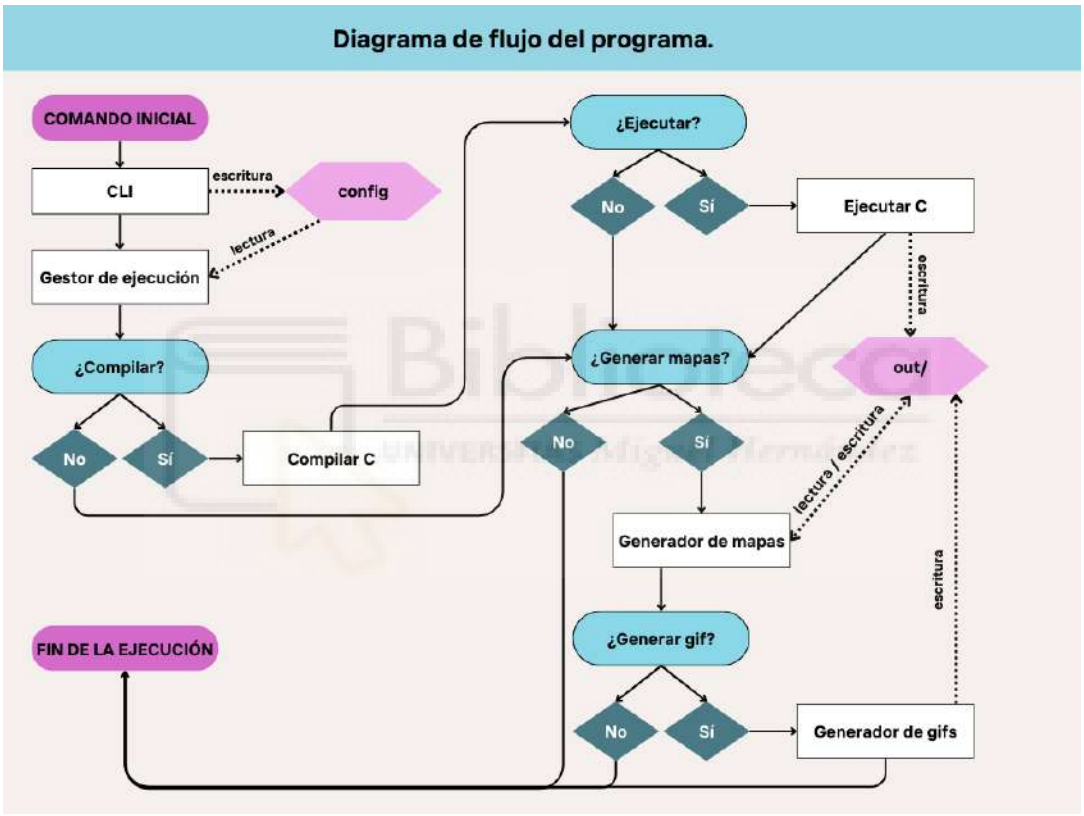

Figura 3.1: Diagrama de flujo de la ejecución del programa.

3.1. Gestor de entradas y de ejecución

A la hora de usar un programa con una gran cantidad de opciones y funcionalidades tan diversas, es crucial tener una buena organización del proyecto, facilitando así su comprensión y su uso, además de permitir actualizaciones y cambios de forma rápida, un uso correcto mediante funciones separadas y siguiendo la filosofía de "caja negra".

En la figura 3.2 podemos apreciar la estructura final del proyecto, donde cada parte está separada de las demás y los archivos *configurator_CLI.py* y *config_executor.py* son los encargados de comunicar todas las partes y mover, copiar y enrutar todos los elementos necesarios para el correcto funcionamiento del programa.

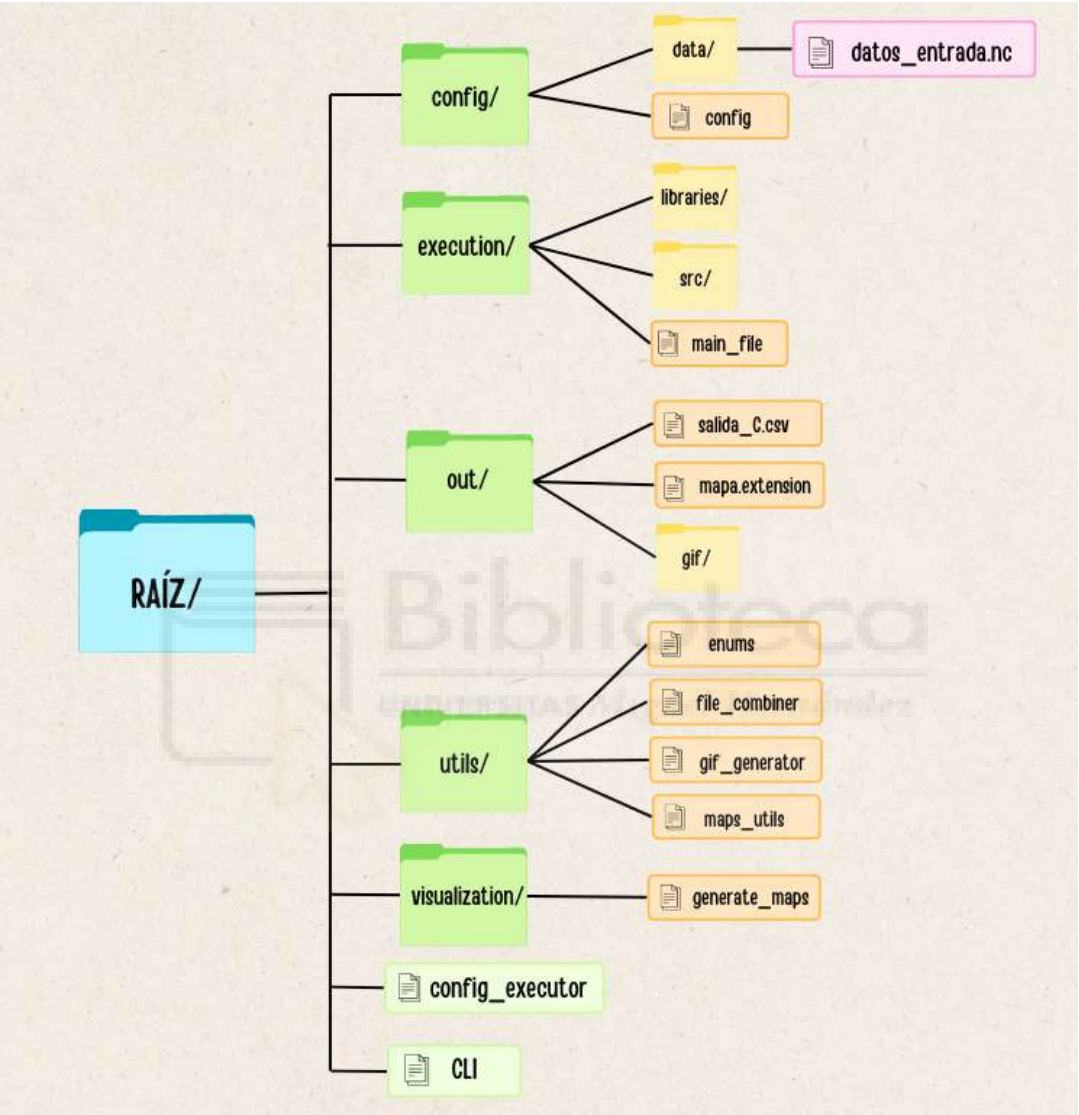

Figura 3.2: Estructura de la jerarquía de directorios y archivos.

Además de un buen orden de archivos y carpetas, es fundamental proporcionar una experiencia de usuario cómoda y agradable, de sencillo aprendizaje y adaptable según los requerimientos. Es por ello que se crea una interfaz por línea de comandos muy completa y con todas las ayudas posibles para una ejecución satisfactoria.

La creación y uso de una interfaz gráfica, pero se descartó principalmente debido a que el uso de este programa es, en general, sobre un servidor de cómputo y, en estos, a nivel de usuario conectado por conexión remota (*ssh*), no se podría visualizar esa interfaz. A pesar de esto, se deja abierta la posibilidad de su creación en futuras versiones del programa, para añadir una capa más de comodidad del usuario.

3.1.1. Command Line Interface

Para este programa, se requiere especificar una gran cantidad de opciones y poder seleccionar la configuración más adecuada para cada situación o estudio. Es por ello que, tal como se muestra en la tabla 3.1, el usuario tiene a su disposición una lista de comandos suficiente y altamente configurable, para que pueda conseguir el resultado deseado cómodamente. Tendremos los tipos: *types_enum*, que admite los tipos: disp, cont, comb, forms; *range_enum*, que admite: max, min, both, comb y *form_enum* que admite: png, jpg, jpeg, svg, pdf. Además, si se usa `--`*all* no podrá usarse `--`*instant*, y viceversa; así como también uno de los dos deberá estar presente obligatoriamente. Estas especificaciones y restricciones quedan recogidas en el comando de ayuda, que se puede obtener con `--`*help* y, gracias a este, el usuario podrá conocer en todo momento qué opciones tiene disponibles y cómo usarlas correctamente.

Comando	Tipo	Descripción	Requerido
--data	str [str ...]	ruta al archivo .nc	Sí
--type	types_enum	posibles mapas a generar	Sí
--range	range_enum	rangos del mapa a generar	No
--levels	int	rango de niveles para los contornos	No
--instant	int [int ...]	instantes de tiempo a ejecutar	No*
--all	No	genera todos los instantes indicados	No*
--latrange	int int	min. y max. de latitud	No
--lonrange	int int	min. y max. de longitud	No
--format	form_enum	formatos de mapas admitidos	No
--out	ruta	ruta a la carpeta de salida	No
--debug	No	ejecución en modo debug	No
--no-compile	No	no compilar la parte en C	No
--no-execute	No	no ejecutar la parte en C	No
--no-maps	No	no generar mapas en la salida	No
--animation	No	generar animación de la salida	No
--omp	No	ejecutar usando OpenMP	No
--mpi	No	ejecutar usando MPI	No
--n-threads	No	número de hilos a usar	No
--n-proces	No	número de procesos a usar	No

Tabla 3.1: Tabla con los comandos admitidos en CLI, sus tipos, descripción y si son requeridos o no.

Biblioteca
UNIVERSITAS Miguel Hernández

El ejecutable que hace funcionar el CLI usa Python, en concreto, las librerías *argparser* y *configparser*. Gracias a sus funciones, nos permite crear una interfaz con aspecto profesional, controlar las entradas y los posibles errores de tipos o de valores incorrectos, así como formatear y preparar de forma cómoda los datos de entrada obtenidos.

3.1.2. Archivos de configuración

Una vez todos los valores han sido validados y formateados, se prepara el archivo de configuración en formato YAML [17]. Este formato es ampliamente utilizado y aceptado para archivos de configuración, almacenaje de datos o, en general, listados de elementos que sea preciso almacenar y utilizar. Su funcionalidad es muy similar a otros formatos como XML o JSON, pero la elección de YAML sobre estos es debido a su mayor grado de legibilidad y comprensión; tanto a la hora de leer como de escribir. Este archivo se prepara, se genera y se escribe desde la interfaz de

comandos, para que, después, el gestor de ejecución lea todo y pueda llamar a las partes correspondientes.

```
        MAP:
          files:
          - config/data/geopot_500hPa_2022-03-14_00-06-12-18UTC.nc
          maps:
          -disp
          - comb
          es_max:
          - comb
          times:
          - 0
          - 1
          - 2
          - 3
          lat_range:
          - 25
          - 85
          lon_range:
          - -180
          - 180
          levels: 20
          file_format: svg
          output: null
          debug: true
          no_compile: false
          no_execute: false
          no_maps: false
          animation: true
          omp: false
          mpi: false
          n_threads: null
          n_proces: null

```

Figura 3.3: Ejemplo de archivo de configuración YAML para la generación de mapas.

Tal como se aprecia en la figura 3.3, tenemos representados todos los comandos anteriormente mencionados en el CLI. En este caso, se usará el mapa ubicado en /config/data/ con fecha 14 de marzo de 2022, se generará un mapa combinado (dispersión y contornos) y se representarán conjuntamente las formaciones máximas y mínimas. Además, se generarán todos los instantes (uso de *--all*) para el rango de latitudes (25, 85) y de longitudes (-180, 180). También, se requieren los contornos con un nivel de 20 unidades entre ellos y con el mapa en formato svg. Al no aportarse ruta de salida (*output*), se generarán los archivos de salida en la carpeta out/ por defecto. También, se ha indicado que se requiere el uso de debug (en este caso,

mediante gdb) y, también, se desea una animación de los 4 instantes indicados.

3.1.3. Gestor de ejecución

Una vez la configuración está lista, desde la interfaz de comandos se llama al gestor de ejecución (*config executor*), que será el encargado de, en un primer momento, leer el archivo de configuración y, posteriormente, llamar al resto de scripts pertinentes.

Su funcionamiento interno puede dividirse en 3 pasos: inicialización, llamada de cómputo y llamada de visualización[2]. En la primera parte, se leen del archivo YAML de configuración todos los datos, se traslada el archivo de datos NetCDF a /config/data/ (si no se encontraba en esa ubicación) y se compila el código en C.

Para el segundo paso, para cada archivo indicado en la configuración, se ejecutará el programa de cómputo con las configuraciones pertinentes y se recogerá el retorno del programa, capturando y enseñando por pantalla si ha surgido algún error.

Finalmente, si las ejecuciones de todos los mapas han sido correctas, se generará, para cada archivo, para cada tipo de mapa, para cada opción de rango seleccionada y para cada instante de tiempo, un archivo de imagen con el formato adecuado. Además, si se indica que se requiere una animación, se llamará a la función de generar una animación en formato gif, la cual nos añadirá en el directorio indicado como salida una animación de los instantes generados en formato gif.

3.2. Algoritmos

En esta sección se analiza la implementación y el diseño de todos los algoritmos usados en la parte escrita en lenguaje C para cómputo. Se decide usar C por la razón de ser uno de los lenguajes más óptimos y eficientes a la hora de realizar operaciones y cálculos, así como para realizar bucles con un gran número de iteraciones o una carga computacional grande. Estos códigos son acelerados mediante

[2]Se deberá tener en cuenta que los pasos descritos a continuación siguen este proceso siempre y cuando no se añadan los comandos de *–no-compile*, *–no-execute* ni *–no-maps*, ya que los pasos donde se deba hacer alguna de estas acciones pueden verse modificados.

el desarrollo de implementaciones paralelas para sistemas de memoria compartida, memoria distribuida e híbridos haciendo uso de OpenMP y MPI. Además, para que todos estos sistemas funcionen y estén correctamente integrados, todo se aplica y se diseña para funcionar en entornos con Sistemas Operativos basados en UNIX, ya que presenta una serie de cualidades muy deseadas a la hora de paralelizar y de implementar ciertas funcionalidades del programa que en sistemas Windows o MacOS no se encuentran.

Cabe destacar que a la hora de obtener los archivos de salida de la parte del programa encargada del cómputo, se decide usar el formato CSV en lugar de formato binario ya que, a pesar de ser más pesado que los archivos binarios, no tiene un tamaño excesivo y presenta ventajas como legibilidad y facilidad de estudio humano y la sencillez de uso como entrada para otros programas.

Al comenzar el programa, se tiene una primera parte de inicializaciones y reservas de memoria, donde todas las variables, vectores y matrices que serán usados quedan definidos y, en su caso, inicializados a sus valores correspondientes. Posteriormente, se procede a leer el archivo de datos NetCDF y obtener de este toda la información necesaria y, finalmente, estandarizarla para poder usarla correctamente.

Es importante destacar que todos los algoritmos descritos a continuación funcionan de forma independiente para cada instante de tiempo indicado en el archivo de configuración, es decir, cada iteración del bucle principal del programa (que recorre todos los instantes) es independiente de la anterior, ya que todas las iteraciones comienzan con filtrado y terminan con escritura de resultados en los archivos correspondientes.

3.2.1. Algoritmo de filtrado

En este primer algoritmo, se encuentra toda la parte de filtrado de los datos. Cada punto de la matriz[3] es, primeramente, ampliado a resolución 1.25^{o}. Esto se realiza eligiendo el punto central de cada uno de sus 25 puntos vecinos en resolución 0.25^{o}.

[3]Se llama punto a cada elemento (latitud, longitud) con valor Z de la matriz (latitud, longitud, Z) de datos y teniendo en cuenta que ses diferente para cada instante de tiempo T.

Posteriormente, ese punto es evaluado mediante el filtro de gradiente. Este filtro está compuesto por un bucle principal, que genera, para las N direcciones a 500 km del punto estudiado, otros puntos usando las funciones de círculo grande y de interpolación bilineal. Estos nuevos puntos son comparados con el punto inicial y, si este es mayor (o menor, en caso de ser un mínimo) que al menos el 90 %, entonces es un punto seleccionado. Se puede observar todo este proceso, dentro de un mismo instante, en el pseudocódigo del algoritmo 1.

Biblioteca
UNIVERSITAS Miguel Hernández

Algoritmo 1: Algoritmo de filtrado de puntos por gradiente.

```
Data: NLAT, NLON, STEP, DIST, N_BEARINGS, z
Result: selected_points
size_x = NLAT / STEP;
size_y = NLON / STEP;
for lat = 0 to size_x do
    for lon = 0 to size_y do
        b_count_max = 0;
        b_count_min = 0;
        for i = 0 to N_BEARINGS * 2 do
            point_aux = coord_from_great_circle(point, DIST, i);
            z_aux = bilinear_interpolation(point_aux);
            if z[lat][lon] ≥ z_aux then
                b_count_max++;
            else
                if z[lat][lon] ≤ z_aux then
                    b_count_min++;
        if b_count_max ≥ N_BEARINGS * 2 * 0.9 then
            point.type = MAX;
            selected_points.add(point);
        else
            if b_count_min ≥ N_BEARINGS * 2 * 0.9 then
                point.type = MIN;
                selected_points.add(point);
```

Las funciones de *coord_from_great_circle()* y *bilinear_interpolation()* han sido explicadas en detalle en la sección 2.6.

3.2.2. Algoritmo de agrupación

Una vez han quedado seleccionados todos los puntos que deben pasar el filtro de gradiente en el instante de tiempo, se realiza la agrupación de los mismos.

Este algoritmo cuenta con dos secciones diferentes: la primera parte es asignar a cada punto el ID del grupo al que pertenezca. Este proceso queda explicado en el algoritmo 2, donde, en primer lugar se recorren todos los puntos seleccionados y, si no tienen asignado un *cluster* (valor de -1), entonces este pertenece a un nuevo grupo (el siguiente por numeración). A continuación, llega la parte de expansión del *cluster*; esta función permite que, a partir de un punto, encontrar todos los otros puntos que cumplan el requisito para agruparse, es decir, deben estar a una distancia menor de 1.25º (tanto en latitud como en longitud) del punto que se está expandiendo y, además, no debe pertenecer a ningún *cluster*. Cada punto expande a sus 8 vecinos directos y, si alguno de estos cumple las condiciones, expandirá también a sus 8 vecinos de forma recursiva, hasta que no hayan más expansiones posibles.

Algoritmo 2: Algoritmo de agrupación de los puntos seleccionados.

```
Data: NEW_RES, filtered_points, size_x, size_y
Result: clusters
id = 0;
for i = 0 to size_x do
    for j = 0 to size_y do
        if filtered_points[i][j].cluster == -1 then
            filtered_points[i][j].cluster = id;
            expandCluster(filtered_points, size_x, size_y, i, j, id, NEW_RES);
            id++;
clusters = fill_clusters(filtered_points, size_x, size_y, id, offset, scale_factor);
```

La segunda parte es la creación de la estructura de *cluster*. No es necesaria como tal, pero facilita mucho la agilidad, comprensión y legibilidad del código. En esta estructura se almacena la siguiente información: ID del *cluster*, número de puntos

que contiene, valor del contorno que envuelve a todos los puntos, centroide, tipo y, finalmente, un vector que contiene todos los puntos que tengan su ID. La misión de la función de esta segunda parte es rellenar toda esta información de cada uno de los clusters, calculando el contorno y el centroide y rellenando los vectores.

Una vez los puntos están agrupados, los grupos creados y con toda la información necesaria, se puede pasar a realizar la selección de formaciones.

3.2.3. Algoritmo de selección

Teniendo todos los grupos listos, se itera con ellos realizando procesos de descarte y selección, consiguiendo así encontrar las formaciones que cumplan todos los requisitos y, si varias lo cumplen para una misma formación, seleccionar la más indicada. A continuación, se enumeran los pasos que sigue el algoritmo para la búsqueda de formaciones, partiendo de cada uno de los *clusters* de máximos:

1. Se busca la existencia de un contorno en la parte superior del máximo, que se encuentre en su zona de influencia (a menos de 3000 km de distancia) y que no haya sido visitado todavía.

2. Se comprueba que este contorno no esté cerrado en torno al máximo, ya que obligatoriamente debe ser un contorno abierto.

3. El contorno encontrado en la parte superior, se busca en las otras 3 direcciones.

4. Si se encuentra el contorno en los laterales y no se encuentra en la parte inferior, se ha localizado un candidato a Omega. Por el contrario, si se encuentra en la parte inferior y en el lado derecho, pero no en el izquierdo, se tendrá un candidato a Rex.

5. Se revisa cada *cluster* de tipo mínimo, buscando uno que se encuentre en la zona de influencia, en una latitud inferior y, en el caso de ser Rex, a no más de 5^{o} de longitud a cada lado respecto del máximo.

6. Se comprueba que el contorno no esté cerrado y, en el caso de Omega, que el contorno aparezca en la parte inferior y en la parte opuesta a la posición del

mínimo, es decir, si el mínimo se encuentra a la derecha, deberá localizar el contorno a su izquierda. En el caso del Rex, deberá localizar el contorno en su parte inferior, izquierda, superior y no deberá encontrarlo a su derecha.

7. En el caso de formación de tipo Rex, si esto se cumple y si mejora al candidato anterior (si hubiera alguno) en cuanto a cercanía con el máximo, esta formación queda seleccionada.

8. En el caso de formación de tipo Omega, se crea una ponderación entre la latitud que tiene (si tiene menor latitud, pondera más) y la distancia media de los mínimos con respecto al máximo y entre ellos. De esa ponderación, si el mínimo mejora al candidato anterior (si se tuviera), se queda seleccionado. Al final, si hay al menos un mínimo a la derecha y otro a la izquierda, la formación queda seleccionada.

9. Finalmente, se escribe en el archivo de formaciones la información correspondiente (instante, tipo y *clusters* que lo forman).

Biblioteca
UNIVERSITAS Miguel Hernández

3.3. Paralelización y optimización

Uno de los objetivos de este proyecto es conseguir una aplicación eficiente y que sea amigable con el usuario, tanto en interfaz y simplicidad de uso como en tiempos de espera. Para abordar esta parte, se deben realizar una serie de aceleraciones del código. Estas se pueden realizar a nivel de procesador (CPU).

Las aceleraciones mediante procesador se basan en optimizar el uso de los recursos computacionales disponibles para reducir los tiempos de ejecución y mejorar el rendimiento general de la aplicación. Existen dos enfoques principales para la aceleración de código a nivel de CPU: la memoria compartida y la memoria distribuida.

Memoria Compartida

En los sistemas de memoria compartida, todos los procesadores acceden a una memoria global común. Este modelo es adecuado para sistemas multiprocesador donde

se pueden utilizar múltiples hilos para ejecutar tareas en paralelo. OpenMP (Open Multi-Processing) es una API para la programación de memoria compartida en C, C++ y Fortran, que facilita la creación de aplicaciones paralelas mediante el uso de directivas de compilador, funciones de biblioteca y variables de entorno. Esta herramientas es una de las más utilizadas para implementar paralelismo en sistemas de memoria compartida, gracias a su simplicidad y efectividad.

Memoria Distribuida

En contraste, los sistemas de memoria distribuida consisten en múltiples nodos, cada uno con su propia memoria local. La comunicación entre nodos se realiza mediante el paso de mensajes. MPI (Message Passing Interface) es el estándar de facto para la programación en memoria distribuida y permite la creación de aplicaciones paralelas escalables. Permite a los programas de computación paralela intercambiar datos y sincronizar sus operaciones a través de una red de comunicación. Su principal desventaja es la complejidad en la programación, ya que se deben gestionar explícitamente la comunicación y la sincronización entre los diferentes nodos.

Memoria Híbrida

Es posible combinar ambos enfoques en aplicaciones híbridas, utilizando OpenMP para paralelizar tareas dentro de cada nodo y MPI para la comunicación entre nodos. Esto permite aprovechar al máximo los recursos disponibles en sistemas de computación heterogéneos.

La elección entre memoria compartida, distribuida o ambas depende de la arquitectura del sistema y de la naturaleza de la aplicación. En sistemas con un número limitado de núcleos, OpenMP puede ser suficiente para mejorar significativamente el rendimiento. En cambio, para aplicaciones que deben ejecutarse en entornos de alta computación con muchos nodos, MPI es más adecuado. Cabe destacar que, en el caso de OpenMP, nunca podrá ser usado en un sistema de nodos conectados por red (cluster), ya que no permite la comunicación entre nodos. Por su parte, MPI

puede ser usado en sistemas de memoria compartida, pero no es recomendable, ya que no aprovecha al máximo las ventajas de este tipo de arquitectura.

Para este trabajo, se decide comprobar qué combinación da mejor rendimiento; distribuida, compartida o híbrida. El uso de MPI se aplicará en el bucle principal del programa, es decir, el que recorre cada instante de tiempo. Se toma esta decisión porque MPI debe aplicarse a la paralelización de grano grueso, creando paquetes de trabajo. Además, posteriormente, se usará OpenMP en el nivel de paralelización interno, que presenta un grano más fino, en el filtrado de los datos mediante gradiente, puesto que es la parte con más carga computacional y con un tiempo de uso de CPU mayor dentro de cada instante.

Para MPI, se repartirá la carga entre el número de núcleos que sean indicados, logrando así un buen balanceo. En el caso de la paralelización de OpenMP, se aplicará al bucle de las latitudes, aprovechando la directiva *parallel for* para recorrerlo. Esta directiva requiere indicar cómo se repartirá dicha carga, pudiendo seleccionarse 3 opciones en el *scheduler*: static, dynamic o guided. Se probará y se estudiarán los resultados del programa con static y con dynamic, pues son los dos que más se ajustan a las necesidades del programa.

3.4. Generador de mapas

Tras todo el cómputo y procesamiento de los datos en C, se deben poder mostrar esos resultados de una forma visual y cómoda para el usuario, por ello, la decisión es usar mapas. Estos mapas tienen todos una serie de características comunes, además de sus particularidades únicas de cada tipo. En un primer lugar se enumerará su estructura general y todo aquello que comparten y, posteriormente, se analizará cada uno de ellos de manera individual.

Todos los mapas parten de los datos del archivo de configuración, dados por el gestor de ejecución. A partir de ellos, primero leen el archivo de NetCDF (si lo necesitan) y el o los archivos CSV que se han generado de la ejecución de la parte de cálculo. Una vez obtenidos, se extraen los datos del instante de tiempo pertinente, se

seleccionan máximos, mínimos o ambos, dependiendo de lo que sea requerido, y se vuelve a realizar una comprobación de estandarización de los datos. Una vez todos los datos son correctos, se procede a generar el mapa y todo lo relacionado con sus atributos; se usan las librerías de *cartopy*, *matplotlib* y *pandas* de Python para, entre otras cosas, establecer una proyección del mapa *Plate carrée*, donde los meridianos y paralelos están separados con la misma distancia y forman una matriz cartesiana. También se establecen los bordes de los países y las fronteras y los límites del mapa que usaremos, según lo indicado en la configuración.

Además, otro aspecto común entre mapas es la parte de la barra de color lateral (ver, por ejemplo, la figura 2.5), los nombres de los ejes y el título. Una vez que todo queda configurado y añadido al mapa, se encuentra la parte final de la generación, donde se guarda la figura con la extensión definida en el archivo de configuración y se indica el tipo de mapa, la variable estudiada (geopotencial), el tipo de mapa generado y la fecha del archivo de NetCDF utilizada.

A partir de toda la configuración general mencionada, en adición a las particularidades de cada mapa, se obtienen las figuras 3.4, 3.5, 3.6 y 3.7; siendo todas ellas mapas del evento ocurrido en fecha 26 de junio de 2019 a las 00:00:00 UTC y resolución de 1.25^{o}, con rango de latitud de 85 a 25 y longitud completa y con los mapas mostrando cada cluster de puntos y su ID correspondiente.

3.4.1. Mapa de Contornos

Estos mapas tienen la peculiaridad de no usar los datos de CSV generados, ya que los mapas de contornos se pueden generarar solamente con el archivo NetCDF. Para esto, se usa la función *contour* de matplotlib, la cual, a partir de dos dimensiones (latitud y longitud) y un valor de Z para cada latitud y longitud, genera una serie de contornos de valor, con intervalos entre los contornos de 20. Se elige 20 ya que, si fuera de un valor superior, serían contornos demasiado grandes y conllevaría pérdida de información; y, si fuera un valor inferior, saldrían demasiados contornos, pudiendo llevar a confusión y a saturación del mapa.

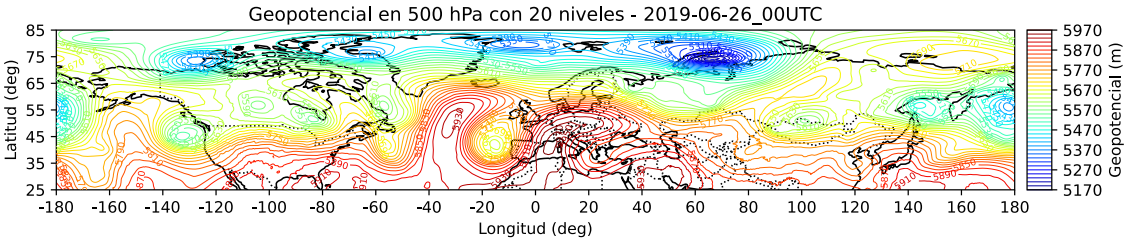

Figura 3.4: Mapa únicamente de contornos.

3.4.2. Mapa de Dispersión de máximos y mínimos

Con este mapa, ocurre al contrario que en el mapa de Contornos, ya que deben utilizarse los archivos CSV, pero no se utilizan los que provienen del NetCDF. Además, aquí se aplica la función de *scatter*, también de la librería matplotlib, donde, a partir de tres vectores (latitudes, longitudes, Z) y una escala de colores, en este caso se usa la escala *jet*, se generan los puntos con su color correspondiente según su geopotencial. Además, cabe destacar que, para indicar el ID de cada cluster, se usa la función *annotate*, la cual aplica a cada latitud y longitud un texto determinado.

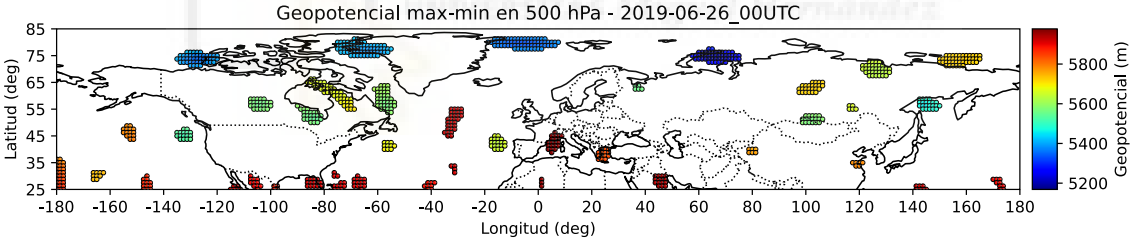

Figura 3.5: Mapa de dispersión de puntos ordenados en clusters.

3.4.3. Mapa Combinado

Con este mapa, se puede obtener la combinación de los dos mapas anteriores. Es el más idóneo para el estudio y análisis de los eventos, ya que permite visualizar las formas de los bloqueos claramente y, además, se aprecia la generación de los clusters y su ID. Presenta alguna particularidad adicional, ya que, al tener cada una de las dos representaciones una escala de colores diferente, se requiere de una modificación en los contornos; estos tomarán como referencia de color la escala generada por el mapa de dispersión, logrando así que los tonos y las intensidades observadas vayan

al unísono.

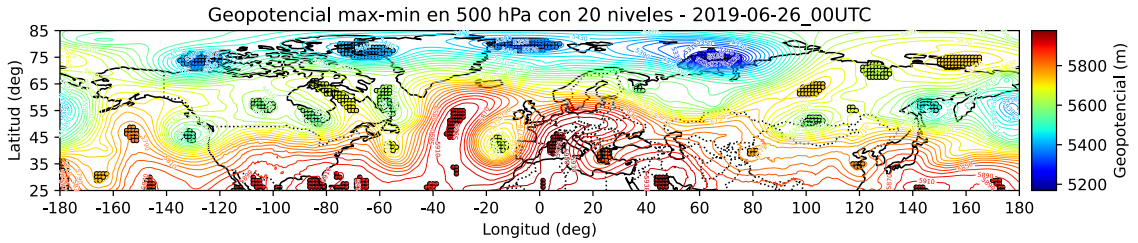

Figura 3.6: Mapa de contornos y puntos mostrando los clusters encontrados.

3.4.4. Mapa de resultados

Con el mapa que debe representar las formaciones, se trabaja algo distinto que con los demás. Primero, se almacena en una *namedTuple* toda la información que presenta cada una de las formaciones; seguidamente, se realiza una dispersión de los puntos y se representan en color rojo los clusters de tipo máximo y en azul los que son de tipo mínimo; se añade el ID del cluster y se muestran únicamente los contornos que están afectados por las formaciones. Finalmente, se añade el tipo de formación encontrada y se usa la función de *ConvexHull* para buscar el polígono más pequeño que pueda contener a la formación.

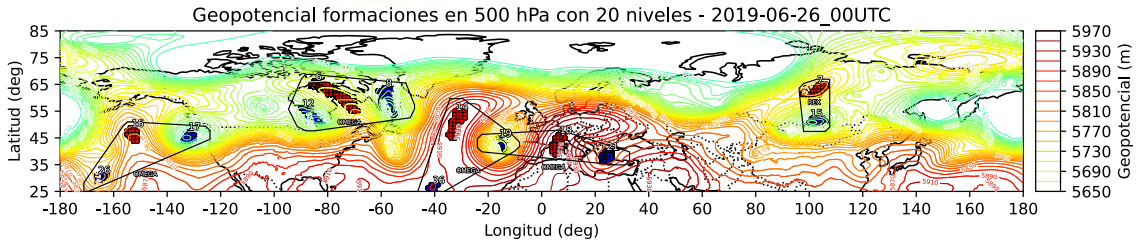

Figura 3.7: Mapa representando las formaciones que han sido localizadas.

3.4.5. Animaciones de instantes contiguos

Como parte final del visualizador y generador de mapas, se integra un generador de animaciones para varios instantes. Este script, también escrito en Python, ayuda a poder visualizar un evento como un conjunto, aportando un archivo GIF a la salida del programa.

Se usan las librerías de *CairoSVG* y de *imageio* y, gracias a estas, se puede implementar el algoritmo 3 para la conversión de archivos SVG (si los hubiera) y la posterior creación de la animación.

Algoritmo 3: Algoritmo para crear una animación a partir de archivos SVG y PNG.

Data: input_folder, output_folder

Result: gif_file

files = input_folder.content();

svg_files = [file for file in files if file.lower().endswith('.svg')];

png_files = [file for file in files if file.lower().endswith('.png')];

if *svg_files* **then**

    for *svg_file* ***in*** *svg_files* **do**

        png_file = svg_to_png(svg_file, temp_folder);

        png_files.append(png_file);

create_gif(png_files, output_gif);

4. Resultados

A lo largo de este capítulo, se detallan todos los resultados obtenidos, sus formatos y cómo interpretarlos. Se describirán los archivos CSV, se mostrarán mapas en formato de imagen, animaciones en formato gif y también tablas y gráficos para el análisis de los tiempos de ejecución del programa en diferentes situaciones.

4.1. Archivos de datos

Al finalizar la ejecución del programa, la primera salida que se obtiene, fruto de la parte de cálculo en C, son 3 archivos en formato CSV y un archivo de texto TXT. El archivo de texto contiene un *log* de la ejecución con toda la información importante, cómo ha ido el desarrollo de la misma y si ha aparecido algún error; en general, información que puede resultar de utilidad al usuario para conocer el estado del programa y si ha obtenido el resultado deseado.

Otro fichero de salida es *speed log*, o registro de tiempos, donde se exportan los tiempos que ha necesitado el programa en cada una de sus partes para ejecutarse, tiene el formato: parte, instante de tiempo, tiempo invertido en finalizarla (en segundos). Gracias a esta información, la realización de métricas, cálculos y comparativas de eficiencia resultan cómodas de realizar y el archivo es fácilmente exportable para ser usado, por ejemplo, en programas de hojas de cálculo.

Para terminar, se generan dos archivos de resultados finales: *formations* y *selected points*; cada uno de ellos aporta la información de los resultados de la ejecución. El archivo con las formaciones tiene un formato: instante de tiempo, ID del máximo, ID del mínimo izquierdo, ID del mínimo derecho, tipo. En el caso de ser de tipo Rex, el mínimo se indicará en el ID derecho y el ID izquierdo tendrá un valor de -1. En el archivo de los puntos seleccionados, se presentan: instante de tiempo, latitud, longitud, geopotencial, tipo, cluster. Ambos archivos deben ser trabajados juntos, ya que el archivo de formaciones depende del de puntos seleccionados para obtener todos los puntos de máximos o mínimos de los *clusters* afectados.

Todos estos datos podrían ser un buen punto de partida para un segundo programa que los use para realizar seguimientos y estudios de la evolución temporal de los fenómenos, para usarlos como apoyo para mapas de estudio de trayectorias del viento o de vorticidad o para representaciones visuales con mucho más detalle o a capas diferentes de la atmósfera. Con tan solo especificar el formato de los archivos CSV, se puede usar toda la información de forma cómoda y simple.

4.2. Análisis de mapas

Los resultados mostrados se centrarán en el análisis de, principalmente, tres eventos significativos de bloqueo atmosférico en tres periodos distintos: Evento 1, en marzo de 2022, del que se estudia el 14 de marzo; Evento 2, en junio de 2019, del que se estudia el 26 de junio; y Evento 3, de agosto de 2003, estudiado a lo largo de la primera quincena y de forma especial el día 3. En estos días se analizaron los campos de altura geopotencial cada 6 horas (4 diarios), y en esos periodos es posible observar evolución en las posiciones e intensidades relativas de los denominados centros de acción: áreas de elevadas y bajas alturas geopotenciales, que se corresponden respectivamente con elevadas y bajas presiones en superficie. La configuración de estos centros de acción provocó el bloqueo en Omega de los vientos del oeste típicos de la zona de estudio durante días, lo que condujo a olas de calor sobre Europa. En las figuras 4.1, 4.2 y 4.3, las isolíneas de altura geopotencial en 500 hPa, localizadas alrededor de los 5500 m de altura, son en cada figura una imagen fija de cómo está organizada la circulación del aire a esas alturas y en ellas se aprecia la obstrucción de los flujos del oeste.

El análisis automático de bloqueos identifica en el evento de 2022 (figura 4.1) un bloqueo en Omega sobre el continente europeo, mientras que en 2019 (figura 4.2) hay dos bloqueos en Omega afectando a Europa y al Atlántico noreste, así como otros dos en las costas este y oeste de Norteamérica y un bloqueo Rex (o de dipolo) en Siberia al norte de Mongolia. En el evento de 2003 (figura 4.3) hay otro bloqueo en Omega sobre el suroeste europeo otro en Rusia alrededor de la frontera euro-asiática, así como otros bloqueos en Omega y un bloqueo Rex transitorios.

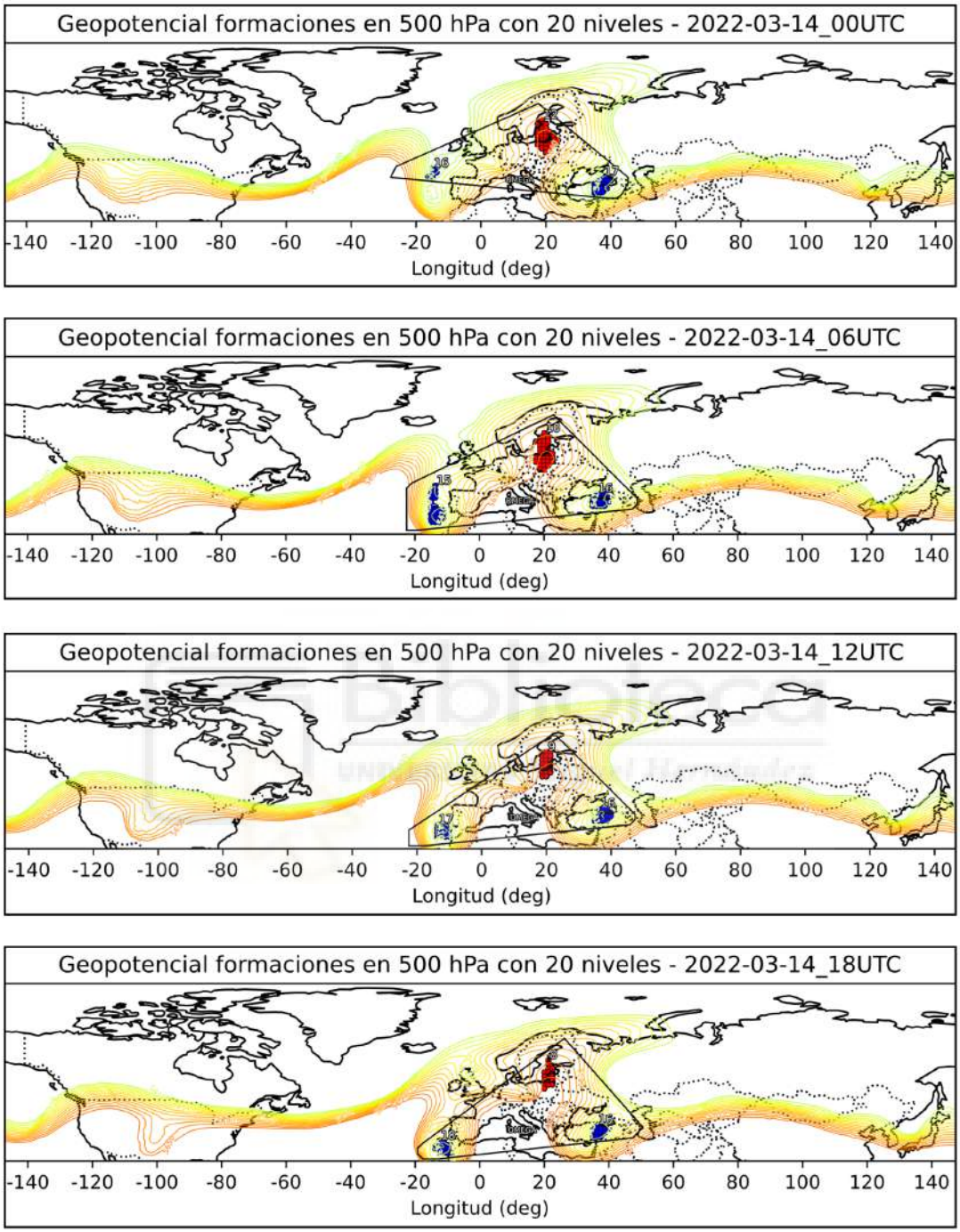

Figura 4.1: Mapa de formaciones encontradas en el Evento de 2022.

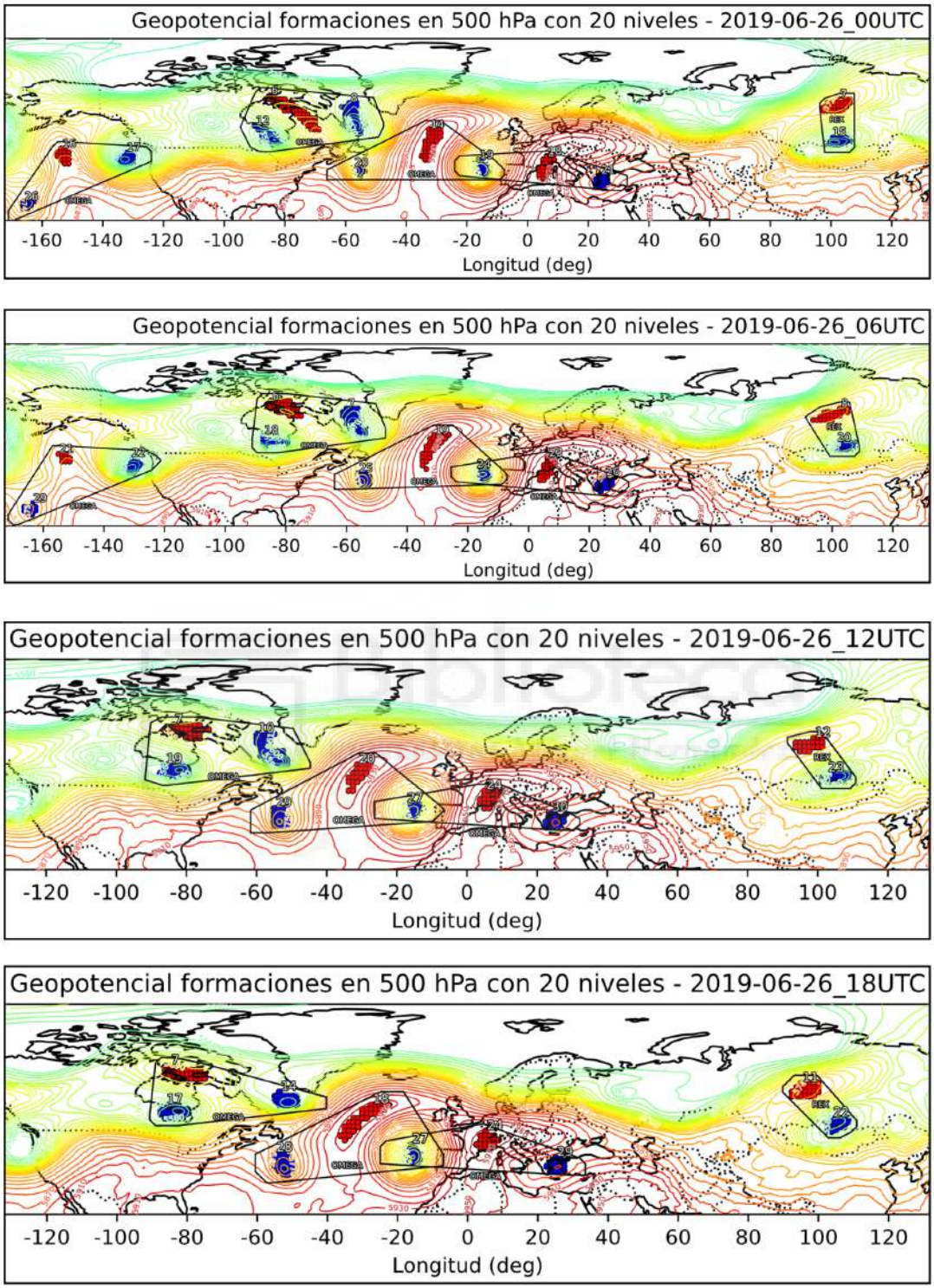

Figura 4.2: Mapa de formaciones encontradas en el Evento de 2019.

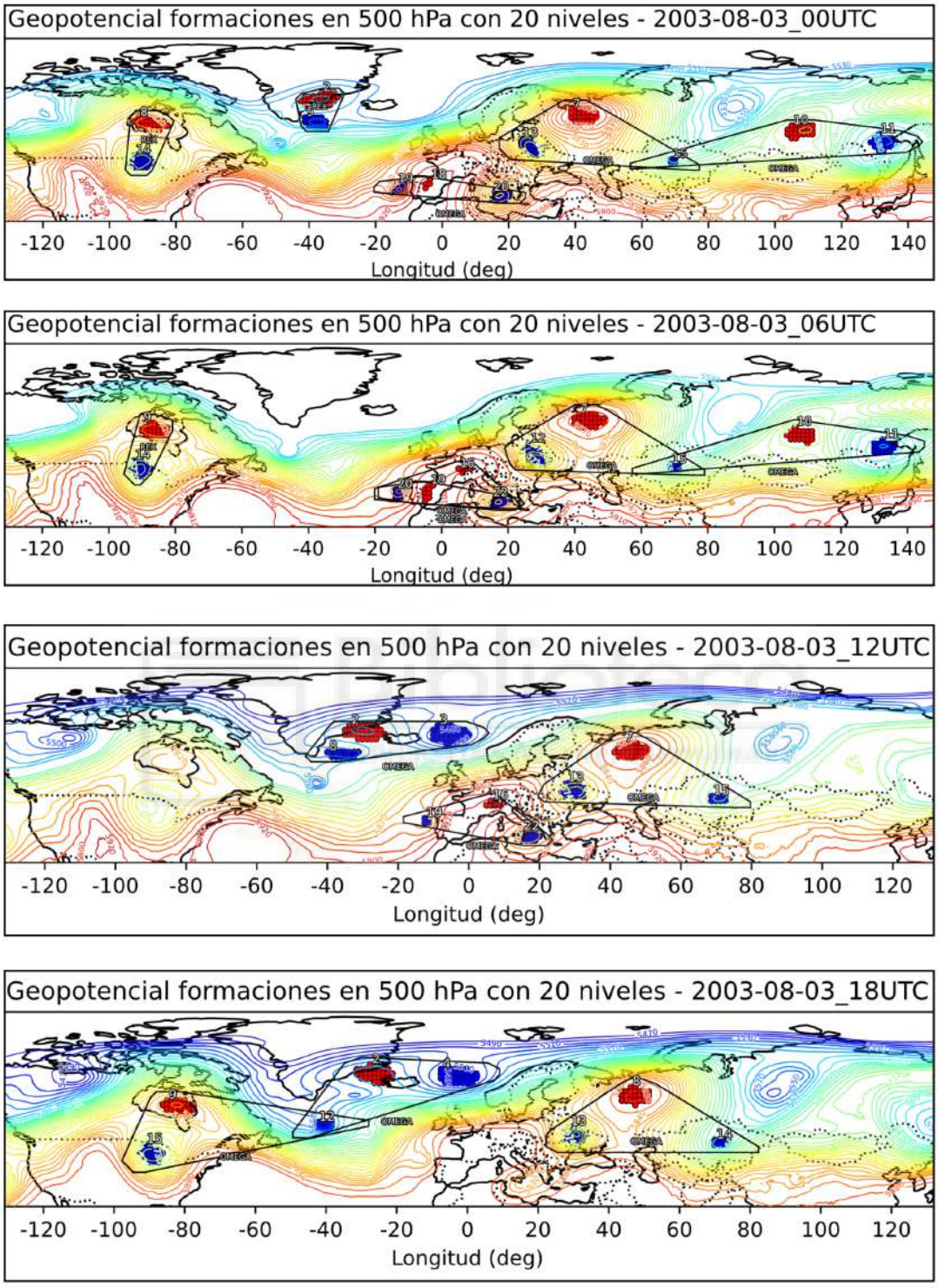

Figura 4.3: Mapa de formaciones encontradas en el Evento de 2003 en el día 3.

Como comparativa del correcto funcionamiento del algoritmo de identificación de bloqueos, se presenta la figura 4.4, donde se aprecia el mapa de resultados con fecha 26 de junio de 2019 a las 18:00:00 UTC y el mapa de identificación de bloqueos mediante método del trapecio y vorticidad, mencionado anteriormente y trabajado en el artículo de C. Detring *et al* [3].

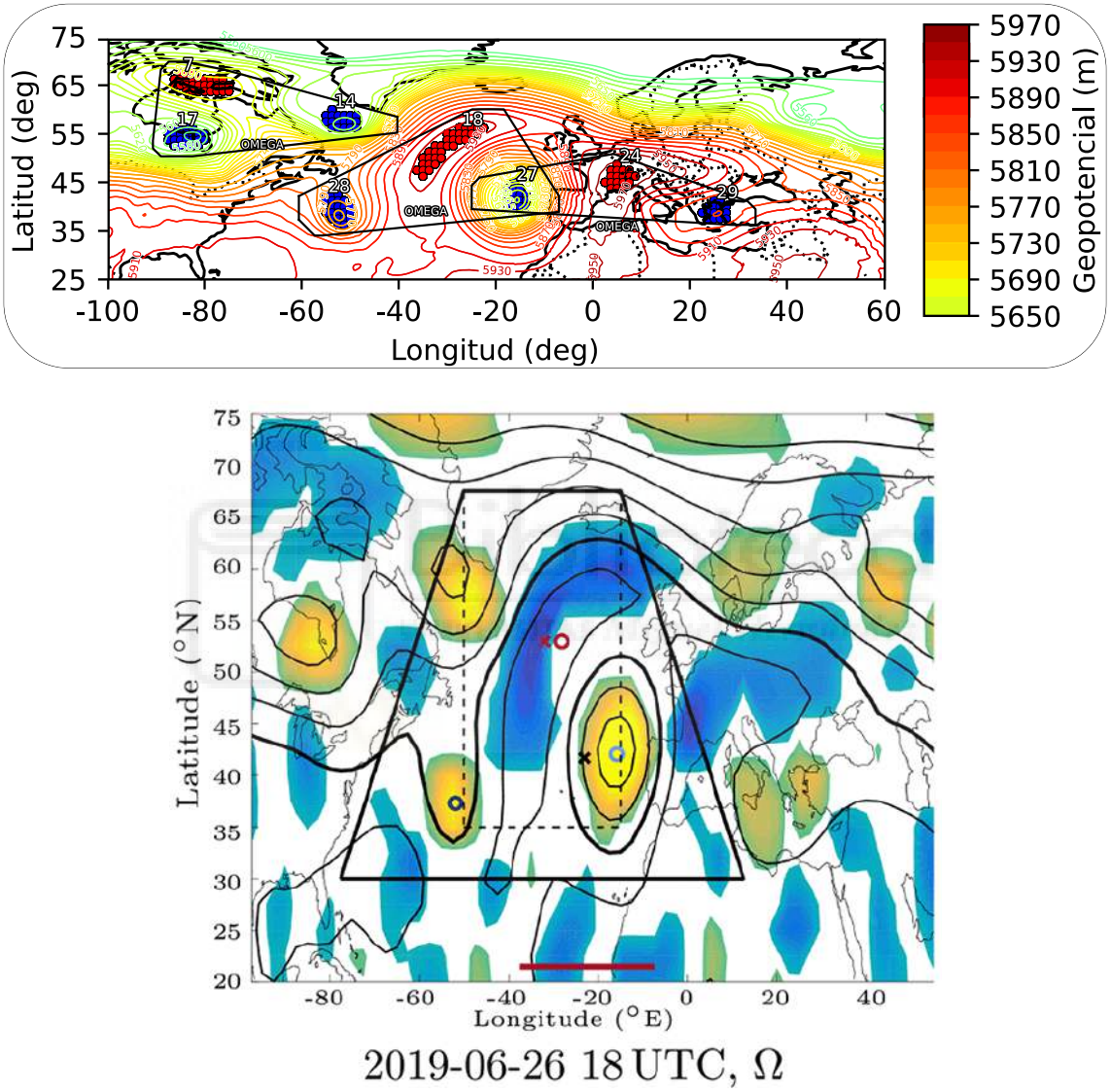

Figura 4.4: Mapa de formaciones y mapa de detección de bloqueos por trapecio. **Source:** C. Detring *et al* [3].

En el anterior mapa, se aprecia cómo en su trabajo son capaces de detectar el bloqueo central, siendo este el más intenso y significativo. No obstante, con nuestro algoritmo, se pueden llegar a identificar dos bloqueos en Omega más, uno a la izquierda que comparte uno de los mínimos y otro en la parte superior.

4.3. Estadísticas de rendimiento

Se ha realizado un estudio sobre el rendimiento y los tiempos que se obtienen al ejecutar el programa en diferentes situaciones y con configuraciones diversas. Se estudian los 3 eventos mencionados anteriormente y se aplican técnicas de paralelización, tanto de memoria distribuida como compartida e híbrida. Todas este estudio, se realiza en un entorno de servidor que presenta las siguientes características:

- 2 procesadores Xeon CascadeLake Gold 620R a 2,4GHz y 24 cores. 24MB de caché L2 + 35,75MB de caché L3,
- 192GB de memoria DDR4 a 2933 MHz,
- Sistema Operativo CentOS 7.0 y
- Compilador GCC 11 con optimizador -O3.

En primer lugar, se tienen los tiempos base de ejecución con el código en secuencial en la tabla 4.1, usando el *flag* de compilador *-O3*, la cual sirve para que el compilador realice todas las optimizaciones posibles, como la eliminación de código muerto, la reordenación de instrucciones, la eliminación de bucles innecesarios y la reducción de la complejidad de las operaciones, entre otras. Gracias a esto, se obtienen tiempos de ejecución muy reducidos, que permiten realizar pruebas de rendimiento con mayor facilidad.

Evento	**Instantes**	**Tiempo de ejecución**
Evento de 2022	4	12.297 s
Evento de 2019	4	13.805 s
Evento de 2003	60	201.381 s

Tabla 4.1: Tabla con los tiempos del código secuencial de cada evento.

A partir de esta tabla, se busca acelerar aplicando técnicas de paralelización; para ello, primero se debe decidir en OpenMP, la directiva *schedule* óptima: *static* o *dynamic*. *Static* tiene menor sobrecarga a costa de dividir los bloques de forma fija, es decir, la carga asignada a cada procesador no varía y no hay balanceo de carga

dinámico. Por su parte, *dynamic* presenta una mayor sobrecarga, pero ofrece un balanceo de carga dinámico, variando la carga asignada a cada procesador. Para este prueba, se toma el tamaño de bloque (*chunk-size*), redondeado hacia arriba al siguiente número entero:

$$\text{chunk\_size} = \left\lceil \frac{\text{size\_x}}{\text{N\_THREADS}} \right\rceil$$

donde:

- *size_x* es el número de latitudes que se procesarán para cada instante y
- $N\_THREADS$ es el número de hilos de esa ejecución.

Ahora que se tiene el *chunk-size*, se realiza una ejecución del evento de 2022 con cada uno de los tipos, para comprobar cual se debe escoger. Así, observando la figura 4.5 se aprecia que la mejor opción es usar *dynamic*, ya que pese a su sobrecarga, logra mejorar notablemente el resultado obtenido con respecto al uso de *static*.

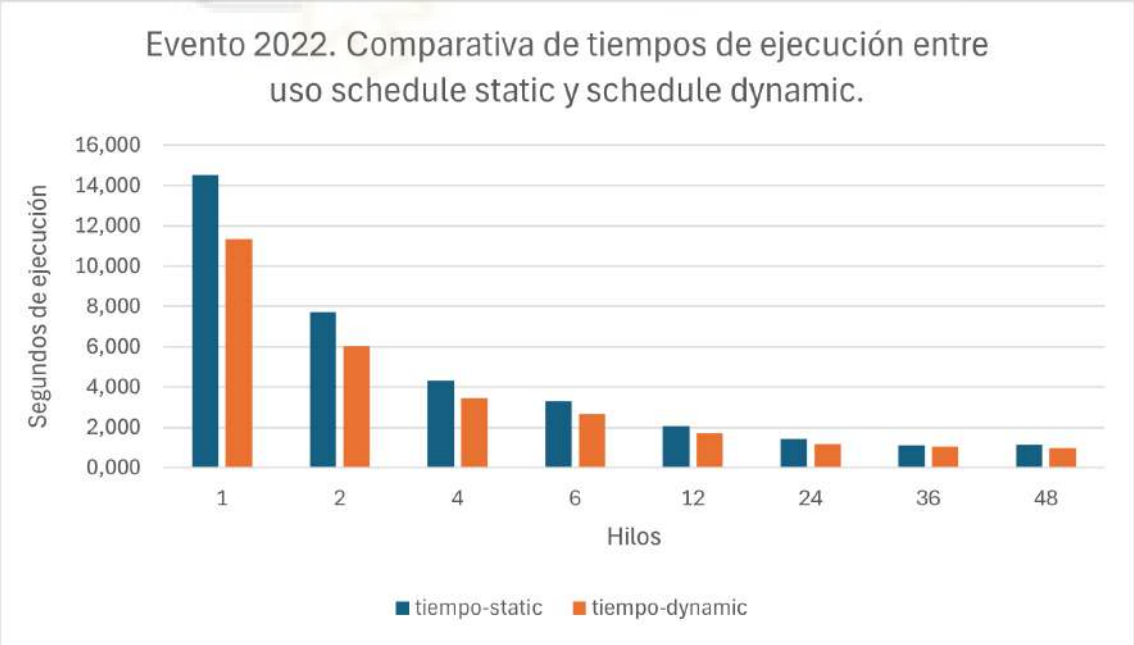

Figura 4.5: Comparativa de tiempos de ejecución en el evento de 2022 usando *static* y *dynamic* en la directiva *schedule*.

Cabe destacar que este *chunk-size* en un principio era el establecido para *static*, pero al realizar pruebas, se estima que su valor es igual a 2 y, siendo *dynamic* generalmente un valor inferior, se probó con valor de 1. Estas pruebas dieron como resultado que empeoraban los tiempos de ejecución; por tanto, finalmente se decide comparar ambos modos con el mismo *chunk-size*. Hay que tener en cuenta que si una ejecución particular tiene la carga balanceada, el uso de *dynamic* servirá para compensar situaciones en la que los procesadores realicen más de una tarea y por tanto tengan ya carga asignada. Además, cuando la carga no está balanceada, es decir, el número de iteraciones asignado a cada procesador no pueda ser el mismo, permitirá un balanceo automático.

Una vez este aspecto queda definido, se puede proceder a realizar ejecuciones usando la aceleración de OpenMP, con la cual se obtiene la tabla 4.2. En esta se aprecia cómo, a pesar de la diferencia de tiempos tan pronunciada entre los eventos 1 y 2 y el evento 3 por la cantidad de instantes, lleva a pensar que se encuentra una homogeneidad general de las ejecuciones del algoritmo, ya que, aún aumentando los instantes, el speed-up y la eficiencia son similares.

Evento 2022								
Hilos	**1**	**2**	**4**	**6**	**12**	**24**	**36**	**48**
Tiempo total (s)	11.34	6.03	3.43	2.67	1.71	1.18	1.05	0.96
Speed-up	-	1.88	3.31	4.25	6.65	9.61	10.78	11.86
Eficiencia (%)	-	94.08	82.65	70.84	55.41	40.03	29.93	24.70

Evento 2019								
Hilos	**1**	**2**	**4**	**6**	**12**	**24**	**36**	**48**
Tiempo total (s)	11.54	6.23	3.66	2.91	1.93	1.45	1.44	1.20
Speed-up	-	1.85	3.15	3.96	5.98	7.97	8.02	9.63
Eficiencia (%)	-	92.62	78.85	66.03	49.86	33.21	22.26	20.07

Evento 2003								
Hilos	**1**	**2**	**4**	**6**	**12**	**24**	**36**	**48**
Tiempo total (s)	173.96	95.62	55.61	44.76	29.73	22.61	19.68	19.60
Speed-up	-	1.82	3.13	3.89	5.85	7.70	8.84	8.88
Eficiencia (%)	-	90.96	78.21	64.77	48.76	32.07	24.55	18.49

Tabla 4.2: Resultados de ejecución con OMP para los eventos de 2022, 2019 y 2003.

Se aprecia cómo a partir de la franja de entre 12 y 24 hilos, se produce una pérdida de eficiencia, lo cual nos indica que el sistema escala correctamente hasta 12 hilos, por lo que se valora si el uso de una paralelización híbrida puede generar mejores resultados. Para estudiarlo, primero se debe observar la gráfica de mejora que presenta el uso de MPI de forma aislada, representado en la tabla 4.3. En primer lugar, para los eventos de 2022 y 2019 solamente se estudia el uso con hasta 6 procesos; esto se debe a que, como solamente tienen 4 instantes de tiempo, si se añaden más procesos que iteraciones del bucle principal (como en el caso de 6 procesos), produce ralentizaciones y tiempos muy poco óptimos. Para los casos con un número inferior a ese limite, vemos una eficiencia muy alta, apreciando así que el uso de MPI en este trabajo es ideal, teniendo en cuenta que MPI es utilizado para implementar una paralelización de grano más grueso que la realizada con OpenMP.

Por su parte, en el evento de 2003, se analizan hasta 36 procesos, siendo sustituidos los 48 por 30. La justificación de esto es similar a la de los eventos anteriores; como se presentan 60 instantes, a partir de 30 procesos baja mucho la eficiencia, al igual que entre 15 y 30, ya que si el número de procesos utilizados es múltiplo del número de iteraciones, se encuentra un punto óptimo de eficiencia, como es el caso de 30 procesos.

Evento 2022				
Procesos	**1**	**2**	**4**	**6**
Tiempo total (s)	11.38	5.73	2.93	12.05
Speed-up	-	1.99	3.88	0.94
Eficiencia (%)	-	99.40	97.06	15.74

Evento 2019				
Procesos	**1**	**2**	**4**	**6**
Tiempo total (s)	11.56	5.83	2.93	12.40
Speed-up	-	1.98	3.95	0.93
Eficiencia (%)	-	99.18	98.79	15.54

Evento 2003								
Procesos	**1**	**2**	**4**	**6**	**12**	**24**	**30**	**36**
Tiempo total (s)	174.33	88.04	45.21	30.52	20.83	45.96	6.69	89.33
Speed-up	-	1.98	3.86	5.71	8.37	3.79	26.07	1.95
Eficiencia (%)	-	99.00	96.41	95.20	69.73	15.80	86.90	5.42

Tabla 4.3: Resultados de ejecución con MPI para los eventos de 2022, 2019 y 2003.

Finalmente, se busca probar si el uso de ambas en un programa híbrido puede producir mejoras en los tiempos, ya que, debido a las limitaciones con el número de iteraciones, esta combinación resulta ideal, ya que se puede conseguir una eficiencia alta y tiempos menores, acelerando con el mayor número de procesos posible múltiplo del número de instantes y añadiendo los núcleos restantes a los hilos controlados por OpenMP. La tabla 4.4 muestra los tiempos de la paralelización híbrida en cada uno de los eventos.

Evento 2022			
Procesos (MPI)	**1**	**2**	**4**
Hilos (OpenMP)	**48**	**24**	**12**
Tiempo total (s)	1.05	0.67	0.55

Evento 2019			
Procesos (MPI)	**1**	**2**	**4**
Hilos (OpenMP	**48**	**24**	**12**
Tiempo total (s)	1.13	0.75	0.56

Evento 2003								
Procesos (MPI)	**1**	**2**	**4**	**6**	**8**	**12**	**15**	**20**
Hilos (OpenMP	**48**	**24**	**12**	**8**	**6**	**4**	**3**	**2**
Tiempo total (s)	19.87	11.75	8.81	7.43	8.90	6.47	6.40	6.36

Tabla 4.4: Resultados de ejecución con MPI + OMP para los eventos de 2022, 2019 y 2003.

Se aprecia claramente cómo la combinación híbrida es la mejor de las opciones, ya que reduce los tiempos ligeramente, pero más importante, optimiza y usa mejor los recursos asignados. Para finalizar, se muestra en la figura 4.6 una comparativa con los tiempos del evento de 2003 mediante el uso de memoria distribuida, compartida e híbrida.

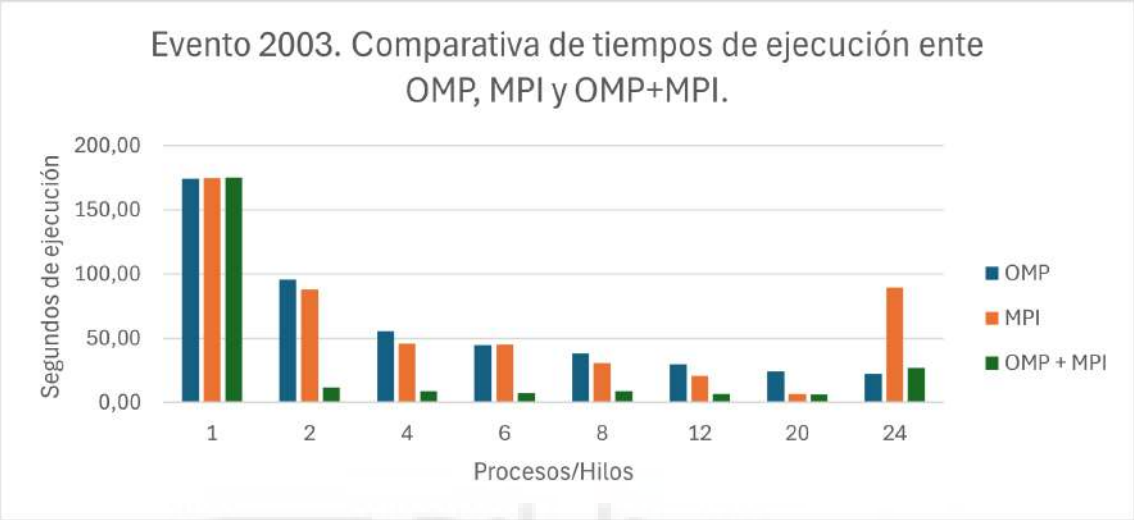

Figura 4.6: Comparativa de tiempos de ejecución en el evento de 2003 usando OMP, MPI y ambas juntas.

5. Conclusiones y Trabajo Futuro

En este capítulo final, se presentan las conclusiones obtenidas a partir de los resultados en el desarrollo de este trabajo, así como las posibles líneas de trabajo futuro que se pueden seguir para mejorar y ampliar el alcance de la solución propuesta.

5.1. Conclusiones

En general, se puede concluir que todos los objetivos planteados han sido cumplidos satisfactoriamente y se han obtenido resultados positivos en la implementación de la solución propuesta para la identificación automática de situaciones de bloqueo atmosférico a través del análisis de las configuraciones de distintos centros de acción. Como resultado del desarrollo de este proyecto, se ha decidido habilitar un repositorio de *GitHub* de acceso público donde se pueda descargar y usar el código implementado. El enlace es: `https://github.com/Victor-Hndz/FAST-IBAN_Project`. A continuación, se presentan las conclusiones obtenidas en cada uno de los objetivos específicos planteados:

Funcionalidad

Se ha logrado implementar un algoritmo en C que permite, a partir de la entrada de datos NetCDF, filtrar, agrupar y reconocer las formaciones de bloqueos atmosféricos correctamente. La solución propuesta es capaz de procesar los datos de entrada y generar una salida en archivos CSV con la información de los bloqueos detectados, de los puntos que los forman, una referencia del tiempo de ejecución empleado y un log que provee información relevante de la ejecución. Además, se ha usado Python y librerías como *matplotlib*, *cartopy* y *pandas* para el procesamiento de los datos y la generación de los diferentes tipos de interés. Finalmente, se aplican las librerías de *CairoSVG* y *imageio* para la generación de animaciones de los bloqueos detectados a fin de mejorar el estudio de los eventos.

Optimización

Se consigue optimizar el tiempo de ejecución de los algoritmos de detección de bloqueos atmosféricos, reduciendo el tiempo de procesamiento de los datos de entrada y generación de los resultados. En primer lugar, se comprueba el correcto funcionamiento de los algoritmos gracias a la compilación mediante CMake y las pruebas unitarias que permite. Después, para el código secuencial, se usan filosofías de programación funcional y se procura la eficiencia en el uso de los recursos de la máquina. Por último, se implementa la paralelización de los algoritmos, gracias a la cual, usando OpenMP y MPI, se llegan a logran mejoras de en torno a 20 veces el tiempo de ejecución con respecto a la ejecución en secuencia.

Escalabilidad

Para este objetivo, se ha logrado un sistema que permite procesar datos de entrada de diferentes tamaños y resoluciones y es capaz de adaptarse a las necesidades de los usuarios y procesar los datos de entrada de forma eficiente, independientemente de la zona de estudio o los instantes de tiempo requeridos. También se tiene una modularidad en el código que permite la fácil extensión del programa a otros tipos de datos de entrada o algoritmos de detección de bloqueos atmosféricos con modificaciones sencillas. Para conseguir esto, se encuentra presente, tanto dentro como fuera del código, una documentación detallada de las funciones y estructuras de datos implementadas, así como de las dependencias necesarias para su correcto funcionamiento.

Integración y Comodidad

Finalmente, se logra un programa que permite a los usuarios comprender el uso rápidamente y hacerlo de forma cómoda mediante el Command Line Interface con

una amplia variedad de opciones y parámetros. El sistema funciona de forma automática y no requiere de intervención del usuario una vez que se ha configurado correctamente.

En resumen, la investigación realizada en este proyecto ha sido muy satisfactoria, ya que se logra obtener una solución en forma de programa al problema de la detección de bloqueos atmosféricos de forma precisa, rápida y sencilla para el usuario.

5.2. Trabajo futuro

A lo largo del estudio y desarrollo de este proyecto, se han encontrado ciertos aspectos que mejorar o añadir, así como diversas ramas que podrían ser ampliadas en futuras investigaciones. A continuación, se presentan algunas de las posibles líneas de trabajo que se pueden seguir para mejorar y ampliar el alcance de la solución propuesta:

- **Interfaz gráfica**: Se podría desarrollar una interfaz gráfica para el programa que permita a los usuarios interactuar con el sistema de forma más visual y sencilla. Esto facilitaría su uso a usuarios menos experimentados y permitiría una mayor personalización. Se implementaría mediante el uso de Python y librerías como *PyQt* o *Tkinter* para la creación de un entorno en el que el usuario pueda introducir los datos de entrada y configurar los parámetros de ejecución de forma intuitiva. Otra posibilidad podría ser la implementación de un entorno de servidor web donde se realizan peticiones en forma de formularios HTML y se devuelven los resultados en formato de tabla o gráfica.

- **Estudio de aceleración mediante CUDA**: En un primer momento, se discutió sobre la decisión de probar todos los tipos de aceleración y realizar una comparativa entre ellos. Al avanzar el proyecto y analizar las características de los algoritmos, se opta por descartar el uso de aceleración por GPU, ya que, debido a las modificaciones necesarias a los algoritmos y las características de los datos, no se encuentran suficientes motivos para justificar una implementación de este tipo de aceleración. Sin embargo, en futuros desarrollos,

se podría estudiar la posibilidad de implementar la aceleración mediante CUDA para los algoritmos de detección de bloqueos atmosféricos y comparar los resultados obtenidos con los algoritmos secuenciales y paralelos.

- **Implementación de reconocimiento de imágenes**: Otra de las ramas descartadas por alejarse demasiado de los objetivos del proyecto, es la implementación de un sistema de reconocimiento de imágenes para la detección de bloqueos. Se podría estudiar los cambios y eficiencia de un programa con base similar al desarrollado, pero usando técnicas como filtrado de Sobel [16], detección de bordes, segmentación de imágenes y reconocimiento de patrones.

- **Seguimiento de eventos de bloqueo**: Se podría investigar la posibilidad de implementar un sistema de seguimiento de eventos de bloqueos atmosféricos a partir de los datos de entrada y los resultados obtenidos. Esto permitiría a los usuarios estudiar, de forma automática, la evolución de los bloqueos a lo largo del tiempo en todos los instantes procesados por el programa y analizar su comportamiento y características.

Biblioteca
UNIVERSITAS Miguel Hernández

BIBLIOGRAFÍA

[1] Danwild, *Install netCDF4 in Ubuntu*, 2016. dirección: https://gist.github.com/danwild/d7225afe4b7dbdeeb87982f0e71012f3.

[2] de Luca, G. and Aibin, M., «Haversine Formula,» *Baeldung on Computer Science*, 2024. dirección: https://www.baeldung.com/cs/haversine-formula.

[3] C. Detring, A. Müller, L. Schielicke, P. Névir y H. W. Rust, «Occurrence and transition probabilities of omega and high-over-low blocking in the Euro-Atlantic region,» *Weather Clim. Dyn.*, vol. 2, n.º 4, págs. 927-952, oct. de 2021. DOI: 10.5194/wcd-2-927-2021.

[4] *ERA5 hourly data on pressure levels from 1940 to present*, 2024. dirección: https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-pressure-levels?tab=form.

[5] M. Ester, H.-P. Kriegel, J. Sander y X. Xu, «A density-based algorithm for discovering clusters in large spatial databases with noise,» en *KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, ago. de 1996, págs. 226-231. DOI: 10.5555/3001460.3001507.

[6] A. Fouilloux, «Hands-on: Visualize Climate data with Panoply netCDF viewer,» *Galaxy Training Network*, nov. de 2023. dirección: https://training.galaxyproject.org/training-material/topics/climate/tutorials/panoply/tutorial.html.

[7] H. Hersbach, B. Bell, P. Berrisford, G. Biavati, A. Horányi, J. Muñoz Sabater, J. Nicolas, C. Peubey, R. Radu, I. Rozum, D. Schepers, A. Simmons, C. Soci, D. Dee y J.-N. Thépaut, *ERA5 hourly data on single levels from 1940*

to present, Accessed on 29-May-2024, 2023. DOI: 10.24381/cds.adbb2d47. dirección: https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-single-levels.

[8] M. Hirt, L. Schielicke, A. Müller y P. Névir, «Statistics and dynamics of blockings with a point vortex model,» *Tellus A: Dynamic Meteorology and Oceanography*, ene. de 2018. dirección: https://www.tandfonline.com/doi/full/10.1080/16000870.2018.1458565?scroll=top&needAccess=true.

[9] A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija y J. Heming, «K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data,» *Inform. Sci.*, vol. 622, págs. 178-210, abr. de 2023, ISSN: 0020-0255. DOI: 10.1016/j.ins.2022.11.139.

[10] A. Müller, P. Névir, L. Schielicke, M. Hirt, J. Pueltz e I. Sonntag, «Applications of point vortex equilibria: blocking events and the stability of the polar vortex,» *Stockholm University Press*, vol. 67, n.º 1, dic. de 2015, ISSN: 1600-0870. DOI: 10.3402/tellusa.v67.29184.

[11] H. C. Rajpoot, «Derivation of great-circle distance formula (Minimum distance between any two points on globe given latitude and longitude),» *ResearchGate*, ago. de 2016. dirección: https://www.researchgate.net/publication/333245201_Derivation_of_great-circle_distance_formula_Minimum_distance_between_any_two_points_on_globe_given_latitude_and_longitude.

[12] L. Schielicke, «Scale-dependent identification and statistical analysis of atmospheric vortex structures in theory, model and observation,» Dissertation, 2017. dirección: http://dx.doi.org/10.17169/refubium-6839.

[13] Simsangcheol, *Bilinear interpolation*, 2023. dirección: https://medium.com/@sim30217/bilinear-interpolation-e41fc8b63fb4.

[14] S. Tibaldi y F. Molteni, «On the operational predictability of blocking,» *Tellus A*, vol. 42, págs. 343-365, 1990. dirección: `https://www.researchgate.net/publication/227881904_On_the_operational_predictability_of_blocking`.

[15] *Unidata | NetCDF*, 2022. dirección: `https://www.unidata.ucar.edu/software/netcdf`.

[16] O. R. Vincent, O. Folorunso et al., «A descriptive algorithm for sobel image edge detection,» en *Proceedings of informing science & IT education conference (InSITE)*, vol. 40, 2009, págs. 97-107.

[17] *YAML Syntax — Ansible Community Documentation*. dirección: `https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html`.

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

ANEXOS

Biblioteca
UNIVERSITAS Miguel Hernández

A. Script de instalación de NetCDF

Este anexo explica el proceso de instalación de las bibliotecas `zlib`, `HDF5` y `NetCDF`. La instalación se puede realizar de forma manual o mediante un archivo `.sh` para mayor comodidad. La guía se basa en este tutorial.

Script en bash

A continuación, se presenta el script de instalación en `bash`. Se recomienda guardar este contenido en un archivo `install_netcdf.sh` y ejecutarlo para realizar la instalación de manera automática.

Biblioteca
UNIVERSITAS Miguel Hernández

```
#!/bin/bash

BASHRC="~/.bashrc"

# Instalar --> ZLIB
v=1.3.1
wget http://www.zlib.net/zlib-${v}.tar.gz
tar -xf zlib-${v}.tar.gz && cd zlib-${v}
./configure --prefix=/usr/local
sudo make install
cd ..

# Instalar --> HDF5
v=1.14.3
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.14/
hdf5-${v}/src/hdf5-${v}.tar.gz
tar -xf hdf5-${v}.tar.gz && cd hdf5-${v}
prefix="/usr/local/hdf5-$v"
if [ $HDF5_DIR != $prefix ]; then
    echo "Add HDF5_DIR=$prefix to .bashrc"
    echo "" >> $BASHRC
    echo "# HDF5 libraries for python" >> $BASHRC
    echo export HDF5_DIR=$prefix  >> $BASHRC
fi
./configure --enable-shared --enable-hl --prefix=$HDF5_DIR
make -j 2 # 2 for number of procs to be used
sudo make install
cd ..

# Instalar --> NetCDF
v=4.9.2
wget https://github.com/Unidata/netcdf-c/archive/refs/tags/
v${v}.tar.gz
tar -xf v${v}.tar.gz && cd netcdf-v${v}
prefix="/usr/local/"
if [ $NETCDF4_DIR != $prefix ]; then
    echo "Add NETCDF4_DIR=$prefix to .bashrc"
    echo "" >> $BASHRC
    echo "# NETCDF4 libraries for python" >> $BASHRC
    echo export NETCDF4_DIR=$prefix  >> $BASHRC
fi
CPPFLAGS=-I$HDF5_DIR/include LDFLAGS=-L$HDF5_DIR/lib ./configure
   ↪ --enable-netcdf-4 --enable-shared --enable-dap
   ↪ --prefix=$NETCDF4_DIR
# make check
make
sudo make install
cd ..
```

Código A.1: Script de Instalación en bash.