UNIVERSIDAD DE ZARAGOZA

TRABAJO FINAL DE MÁSTER

Generación de un Modelo de Orden Reducido para el Cálculo Detallado del Viento en entornos con Orografía Compleja

---

Author:
Jaime MILLA VAL

Director:
Carlos MONTAÑÉS BERNAL
Ponente:
Pedro MATEO COLLAZOS

Máster en Modelización e Investigación Matemática, Estadística y Computación

10 de septiembre de 2020

Resumen

This Master's Thesis aims to develop a methodology to build a reduced order model (ROM) from numerical weather prediction simulations (NWP) to obtain equivalent results to those given by computational fluid dynamics (CFD) technics. The ROM is built by fitting supervised learning regressor models to CFD products from NWP results. The proposed ROM is used over a complex terrain located in the Pyrenees mountains of the Aragon (Spain) region. The obtained results capture fairly well the general trend of the test data. The 85 % percentile of absolute errors in the horizontal components of the wind is ~ 2 [m/s] while for the vertical one is $\sim 0{,}8$ [m/s]. These results seem to be strongly affected by altitude. The numerical weather predictions simulations take around 2,5 min while CFD require ~ 40 min. Once the ROM is trained, using it is nearly instantaneous so the computational advantage of using this methodology is huge.

Agradecimientos

Quería agradecer a todas las personas que me han ayudado a lo largo de la realización de este trabajo. Destacar el apoyo y recursos prestados por la empresa NablaDot S.L y la oportunidad que me han brindado para poder trabajar junto a ellos. En general agradecer a todos los compañeros que aquí me he encontrado por sus consejos y ayuda.

Especialmente destacar el apoyo proporcionado por Carlos Montañés a lo largo de todo el proceso tanto en la concepción, desarrollo y redacción de este trabajo. Su inestimable contribución ha permitido sin ninguna duda llevar a buen término el proyecto.

También agradecer enormemente a Miguel Cámara por su experiencia y paciencia a la hora de enseñarme a usar OpenFOAM. Su apoyo técnico en la creación de la malla ha sido, sin duda, crucial para poder realizar este trabajo.

Finalmente agradecer a mis padres, mi hermano y mis amigos que siempre han estado a mi lado para darme aliento, especialmente cuando el desarrollo de este proyecto se hacía más duro. Muchas gracias a todos.

Índice general

Capítulo 1

Introducción

Uno de los principales temas de investigación dentro del campo de la dinámica de fluidos geofísica es el estudio de modelos numéricos y de laboratorio para los flujos atmosféricos. Estos incluyen aplicaciones que van desde la determinación de vientos cerca de la superficie para energía eólica [20, 31] a aplicaciones aeroespaciales para altitudes elevadas de las físicas de la atmósfera [56, 85]. Concretamente, el estudio de flujos de viento sobre terrenos complejos ha atraído especial interés no solo para la energía eólica [17, 42, 55, 90] sino también para aplicaciones relacionadas con la dispersión de partículas y contaminantes [16, 22, 23, 79]. Para las áreas de la geofísica y la meteorología, entender cómo se comportan estos flujos de aire, sobre todo los turbulentos, es de gran importancia [29]. Por ejemplo, conocer cómo estos flujos se mueven alrededor de edificios o ciudades es cada vez más importante para legisladores urbanísticos o ingenieros medioambientales a la hora de diseñar nuevos espacios urbanos que proporcionen un ambiente confortable y saludable [13]. Las simulaciones numéricas son una de las principales herramientas para comprender estos flujos y han demostrado su importancia en un amplio rango de investigaciones como física de la atmósfera [87], dispersión de contaminantes [5, 41, 74] y planificación urbanística [30, 51]. [92]

Entender los detalles del flujo del aire de manera local y específica es crítico para hacer predicciones a corto plazo de la variabilidad del viento; estos son necesarios para aplicaciones como energía eólica u otras industrias o defensa, en las que se requiere una predicción del viento detallada. Las técnicas de Fluidodinámica computacional (CFD) son capaces de modelar los detalles de las corrientes alrededor de geografías concretas y estructuras creadas por el hombre. Por otro lado, los modelos numéricos de predicción climática (NWP) proporcionan información sobre la variabilidad geofísica a una escala mayor [94]. Idealmente se buscaría un procedimiento que aunara las ventajas de ambos métodos. Unos de los principales inconvenientes de esto es el alto requerimiento computacional de la CFD. La solución a ello podría pasar por el uso de algún tipo de técnica que reduzca o evite estos requisitos computacionales tan elevados. Además, el acoplamiento entre estos dos métodos no es una cuestión trivial.

Una de las técnicas para reducir el coste computacional de la CFD sería el modelado de orden reducido, que está cobrando cierta relevancia para aplicaciones dentro de la ingeniería [26, 39]. Ofrecen una habilidad única para parametrizar ecuaciones diferenciales parciales dependientes del tiempo para problemas de flujo. La construcción de estos métodos está asociada con el cálculo de soluciones de alta fidelidad una única vez para un determinado conjunto de parámetros. Más adelante se aplican métodos "acaparadores" (*greedy*) como: teselacciones de centroide de Voronoi, descomposiciones de Taylor, Lagrange, Hermite, descomposición ortogonal apropiada (POD) o generalizada (PGD). De esta manera se obtiene una base/modos que representan el sistema. Particularmente, esta nueva base se puede proyectar sobre las ecuaciones que gobiernan el problema mediante aproximaciones de Galerkin. Así se obtienen resultados de alta fidelidad con una precisión aceptable en un tiempo relativamente reducido [70]. Uno de los primeros usos que se le dio a estos métodos en el campo "ingenieril" consisitió en identificar la estructura de un flujo con tamaños y distancias variables [7, 11]. Más adelante fueron usadas para analizar la capa límite con características turbulentas [6, 71]. También se aplican en [1, 2] para analizar la estructura turbulenta alrededor de perfiles cilíndricos. En [9] se emplean POD para estudiar las estelas

de turbinas eólicas. Esta técnica también es usada para generar un modelo de velocidad-presión acoplada parametrizando las ecuaciones de Navier-Stokes [78]. Varios autores [43, 76, 77, 89] han contribuido a este método para poder aplicarlo a flujos con número de Reynolds moderado-alto. Existen casos en los que mezclan estas técnicas con herramientas de *Machine Learning* como redes neuronales [82, 92]. Otro estudio explora la posibilidad de la predicción meteorológica a través análisis estadísticos y bosques aleatorios en una localización concreta [3].

Con respecto al acoplamiento entre simulaciones NWP y CFD también ha habido diferentes estudios. Aquí el principal problema radica en los diferentes modelos de turbulencia que se usa en los dos tipos de simulación. Por lo general transformar la energía cinética y su ratio de dispersión de un modelo a otro no es trivial y necesita del diseño y definición de modelos específicos [79]. En este sentido, en [94] se presenta un estudio preliminar para este acoplamiento, centrándose en el campo de la energía, mientras que [59] lo hace en entornos urbanos. Otros estudios [80, 93] exploran no solo el acoplamiento NWP-CFD sino también el CFD-NWP y modelos mixtos.

El objetivo de este proyecto es desarrollar una metodología para obtener un *modelo de orden reducido* (ROM), de forma que a partir de simulaciones NWP permita obtener resultados equivalentes a los de CFD. Mientras que con NWP se simularía un periodo temporal amplio (que puede abarcar días en el caso de predicción o incluso hasta años para reanálisis meteorológico), el modelo CFD se utilizaría sólo para calcular una selección concreta de instantes dentro de este intervalo. Para obtener resultados con la resolución del modelo CFD sin tener que realizar simulaciones adicionales en instantes complementarios, los datos NWP se ajustan mediante un modelo de regresión a las simulaciones CFD realizadas. Ejemplos de estos modelos pueden ser: bosques aleatorios , K-vecinos próximos, máquinas de vector soporte o gradientes estocásticos descendentes. El tiempo computacional requerido por las simulaciones NWP y la aplicación del ROM es muy inferior a las demandadas por la CFD. Por ello, lograr un modelo como el que se busca supondría obtener resultados tan fidedignos y detallados como los de CFD evitando su alto coste computacional. Esto además podría abrir las puertas a simulaciones en tiempo real con alto grado de detalle. Poder aplicar esta metodología a terrenos con orografía compleja o entornos urbanos abre la posibilidad a realizar estudios más detallados sobre estas regiones para el campo de la energía eólica. Además no sólo se restringe a las aplicaciones energéticas, también podría aplicarse al conocimiento más profundo, o potencialmente en tiempo real, de la dispersión de contaminantes o servir como herramienta de apoyo para proyectos urbanísticos. En la industria del ocio se podría aplicar en diferentes aspectos como: planificación de eventos culturales al aire libre o estaciones de esquí, por ejemplo. También en el ámbito de la seguridad a la hora de planificar rutas de vuelo para helicópteros en misiones de rescate de alta montaña. En este trabajo se aplica la metodología desarrollada en una región con orografía compleja situada en el Pirineo aragonés para evaluar su eficacia.

La estructura general de este trabajo es la siguiente: en el capítulo 2 se exponen los fundamentos teóricos básicos de los principales elementos empleados; en el capítulo 3 se describe la metodología desarrollada para la construcción de los ROM; finalmente, en el capítulo 4 se presenta una aplicación práctica de esta metodología a una zona concreta del Pirineo aragonés y se se realiza un pequeño análisis paramétrico respecto a la forma de construir el ROM.

Capítulo 2

Fundamentos

En este cápitulo se presentan los fundamentos teóricos de los tres elementos principales en los que se basa el trabajo desarrollado: modelo de predicción climática (sección 2.1), simulaciones CFD (sección 2.2) y el ROM, para el cual se exponen diferentes alternativas descritas en la sección 2.3.

2.1. Modelo de predicción climática

Históricamente los meteorólogos han estudiado la dinámica de la atmósfera desde dos puntos de vista claramente separados por sus escalas espaciales y temporales. Uno sería la microescala que trata de explicar el comportamiento de la superficie más cercana del entorno que habita el hombre, con escalas espaciales de unos cuantos metros y temporales del orden de minutos. Fenómenos como la difusión de plumas de chimeneas o la interacción viento-estructuras se encontrarían en este tipo de escala. Por otro lado, la macroescala se centraría en predecir la evolución atmosférica de patrones a nivel planetario, teniendo escalas espaciales mayores a un millar de kilómetros y temporales de semanas. Las olas baroclínicas[1] se enmarcarían en esta escala. El término mesoescala entonces recogería todos los estados intermedios entre la macro- y microescala. Se corresponde con fenómenos de escalas espaciales de ~ 1000 km a ~ 2 km y temporales de unos cuantos días. La mesoescala comprende pues los fenómenos de la parte de la cascada de energía[2] que involucra la transferencia de energía desde las escalas integrales de la turbulencia a nivel atmosférico (borrascas, huracanes) hasta las escalas locales (sistemas de cúmulos, tormentas).

Esta escala es la que se resuelve y simula con WRF. WRF por sus siglas en inglés *Weather Research and Forecasting* es un sistema predicción climática numérica de mesoescala diseñado tanto para predicción operativa como para análisis atmosférico [40]. ARW (*Advanced Research WRF*) [72] es una configuración de WRF y será la usada en este trabajo en su versión 4.1. A continuación se describirán brevemente las ecuaciones que se resuelven y cómo se lleva a cabo la simulación con este modelo.

El modelo de mesoescala requiere de datos de entrada (condiciones iniciales y de contorno) proporcionados por modelos atmosféricos globales[3] (como el GFS del NOAA o el IFS del ECMWF). Se genera una malla sobre la localización escogida. En la dirección horizontal esta malla suele ser una cuadrícula regular, y en la vertical se emplea una coordenada $\eta \in [0, 1]$. Esta variable se emplea porque permite adaptar los niveles verticales a la orografía del terreno, se define en términos de presiones como:

[1] La inestabilidad baroclínica es el mecanismo dominante en dar forma a las borrascas y anticiclones para latitudes medias [32].

[2] La cascada de energía es el proceso mediante el cual los torbellinos más grandes transfieren energía a escalas más pequeñas, dando lugar a vórtices cada vez menores hasta que los más pequeños disipan esta energía térmicamente (disipación viscosa) [19].

[3] Un modelo atmosférico global es un modelo compuesto por cuatro modelos separados (modelo atmosférico, oceánico, terrestre y de hielo marítimo), que funcionan conjuntamente para proporcionar una imagen precisa de las condiciones climatológicas. Fuente: GFS.

$$\eta = \frac{p_d - p_t}{p_s - p_t} \tag{2.1}$$

donde p_t y p_s los valores que toma p_d en el límite superior del dominio y en la superficie respectivamente. Por otro lado, p_d se podría asociar con la componente hidrostática de la presión del aire seco, sin embargo en ARW se modifica este valor para minimizar la influencia del terreno más rápidamente en alturas más elevadas (consultar [72] para más detalles). En la figura 2.1 podemos ver un esquema de cómo se comporta η.

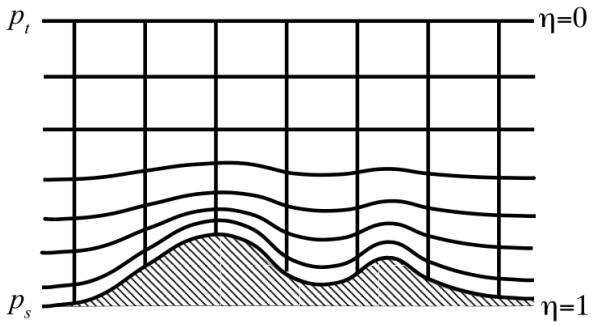


Figura 2.1: Coordenada vertical η de ARW. *Fuente: [72].*

El solver de las dinámicas de ARW integra las ecuaciones de Euler (flujo no viscoso) compresibles y no hidrostáticas derivadas usando variables que tienen propiedades de conservación. Se presentan a continuación de manera breve:

$$\begin{aligned}
\frac{\partial \mathscr{U}_u}{\partial t} + (\nabla \cdot \mathscr{U} u) + \mu_d \alpha \frac{\partial P}{\partial x} + (\alpha/\alpha_d)\frac{\partial P}{\partial \eta}\frac{\partial \phi}{\partial x} &= F_{\mathscr{U}_u} \\
\frac{\partial \mathscr{U}_v}{\partial t} + (\nabla \cdot \mathscr{U} v) + \mu_d \alpha \frac{\partial P}{\partial y} + (\alpha/\alpha_d)\frac{\partial P}{\partial \eta}\frac{\partial \phi}{\partial y} &= F_{\mathscr{U}_v} \\
\frac{\partial \mathscr{U}_w}{\partial t} + (\nabla \cdot \mathscr{U} w) - g[(\alpha/\alpha_d)\frac{\partial P}{\partial \eta} - \mu_d] &= F_{\mathscr{U}_w} \\
\frac{\partial \Theta_m}{\partial t} + (\nabla \cdot \mathscr{U} \theta_m) &= F_{\Theta_m} \\
\frac{\partial \mu_d}{\partial t} + (\nabla \cdot \mathscr{U}) &= 0 \\
\frac{\partial \phi}{\partial t} + \mu_d^{-1}[(\mathscr{U} \cdot \nabla \phi) - g\mathscr{U}_w] &= 0 \\
\frac{\partial Q_m}{\partial t} + (\nabla \cdot \mathscr{U} q_m) &= F_{Q_m}
\end{aligned} \tag{2.2}$$

Las ecuaciones se presentan a título informativo, para consultar en profundidad el trasfondo y significado de estas el lector puede consultar la documentación original [72].

2.2. Modelo CFD

Un modelo CFD calcula el flujo fluido, considerando los fenómenos físicos y químicos involucrados (tales como la turbulencia, la transferencia de calor o las reacciones químicas) resolviendo las ecuaciones de Navier-Stokes sobre una malla computacional del dominio discretizado.

Se emplea el método de volúmenes finitos [25, 37, 84, 88] para representar y evaluar las ecuaciones en derivadas parciales que representan los procesos físicos en ecuaciones algebraicas. Esta aproximación es especialmente útil cuando se trabaja con mallas no estructuradas como es el caso del presente trabajo [44]. La herramienta que se emplea para realizar las simulaciones CFD es OpenFOAM [88] que se trata de un software libre elaborado principalmente en C++. Se ha utilizado la versión 6 de la rama

gestionada por *The OpenFOAM Foundation*. La resolución de la malla computacional para este caso es del orden de metros (frente a los ~ 2 km que se emplean en mesoescala, ver sección 2.1). Se trata de una malla no estructurada y se compone de poliedros de diverso número de caras que ocupan todo el espacio, adaptándose al relieve o la estructura a simular (en el caso aplicado, a un terreno montañoso).

La ecuación general de conservación que se resuelve y sirve para modelizar diferentes procesos físicos, como por ejemplo el transporte de partículas debido a la velocidad del flujo, fuentes de calor, etc, es la ecuación de Navier-Stokes. Para una variable genérica ϕ tiene la forma:

$$\underbrace{\frac{\partial(\rho\phi)}{\partial t}}_{\text{Term. Temporal}} + \underbrace{\nabla\cdot(\rho\mathbf{U}\phi)}_{\textit{Term.Convectivo}} = \underbrace{\nabla\cdot(\Gamma_\phi\nabla\phi)}_{\text{Term. Difusivo}} + \underbrace{S_\phi}_{\text{Term. Fuente}} \tag{2.3}$$

donde ρ es la densidad del fluido , $\mathbf{U}$ es el vector velocidad, Γ_ϕ es el coeficiente difusivo para la variable ϕ (por ejemplo la viscosidad para la ecuación de conservación de momento) y S_ϕ el término fuente, que aglomera toda aportación (o sustracción) de la cantidad ϕ que no se pueden expresar en los términos especificados en la ecuación (2.3) (calor por radiación en una llama, por ejemplo) [44].

Las hipótesis bajo las que se resuelve el flujo son:

- Se considera un flujo **incompresible**, es decir, $\rho = \text{cte}$.
- Se asume **estado estacionario**.
- Se resuelven las ecuaciones promediadas (modelo RANS, *Reynolds-Averaged Navier-Stokes* [52]). Para cerrar algunos de los términos resultantes del promedidado en la ecuación de cantidad de movimiento, es necesasrio aplicar un modelo de turbulencia. En concreto se ha utilizado el modelo de **Spalart-Allmaras** [75].
- Se asume flujo **isotermo**. Puesto que el objetivo final del proyecto es desarrollar un modelo de orden reducido, se busca simplificar lo máximo posible la CFD y no se resuelve la transferencia de energía.

En el caso estudiado, el solver empleado para resolver el acoplamiento presión-velocidad es `simpleFoam` [53] que es la implementación en OpenFOAM del algoritmo numérico **SIMPLE** por sus siglas en inglés *Semi-Implicit Method for Pressure Linked Equations* [15].

Las ecuaciones que se han resuelto para los casos simulados, una vez aplicadas las hipótesis antes mencionadas son dos, continuidad:

$$\nabla\cdot\mathbf{U} = 0 \tag{2.4}$$

y conservación de momento:

$$\rho\nabla\cdot(\mathbf{U}\otimes\mathbf{U}) = -\nabla P + \nabla\tau' + \rho\mathbf{g} \tag{2.5}$$

donde $\mathbf{g}$ es la aceleración de la gravedad, P es la presión y τ' es el tensor de esfuerzos viscosos, que para flujo incompresible se expresa como:

$$\tau' = \mu(\nabla\mathbf{U} + (\nabla\mathbf{U})^T) - \frac{2}{3}\mu(\nabla\cdot\mathbf{U})\mathbf{I} \tag{2.6}$$

con μ es la viscosidad e $\mathbf{I}$ la matriz identidad.

2.3. Modelo de orden reducido

El objetivo buscado es ajustar los resultados obtenidos con WRF a los producidos por la CFD mediante la creación de un modelo de orden reducido (ROM). Ambas clases de datos se tratan de valores reales continuos; por ello se explora construir el ROM mediante diferentes opciones: técnicas de aprendizaje supervisado (apartado 2.3.1) e interpolación directa mediante triangulaciones de Delaunay (apartado 2.3.2). En la literatura científica encontramos [3] que emplea una técnica algo parecida a la

expuesta aquí. El autor de este artículo entrena un bosque aleatorio sobre unos datos de WRF para su uso en predicción meteorológica.

Se denominan a los datos de entrenamiento, los obtenidos con WRF, como $x_i \in \mathbb{R}^m$ con $i = 1, 2, \ldots, n$ y a las variables objetivo, las producidas por la CFD, $y_i \in \mathbb{R}^k$. El número de muestras que tenemos en los datos de entrenamiento es n, el número de características/propiedades descriptivas de los datos (*features*, variables independientes) lo denominamos m (número de variables de las simulaciones mesoescala empleados) y el de los valores objetivo (*labels*, variables dependientes) k (número de productos de la simulación CFD que se quieren predecir). De manera similar se denomina $x_l \in \mathbb{R}^m$ con $l = 1, 2, \ldots, p$ a las variables independientes e $y_l \in \mathbb{R}^k$ a las dependientes del conjunto de test siendo p el número de datos dentro de este conjunto. Se llama $\hat{y}$ a las variables objetivo generadas por el ROM.

Adicionalmente se emplea la métrica R^2 para medir el ajuste a los datos logrado por los modelos. Esta representa la proporción de varianza de y_l que se explica por las variables dependientes en los modelos. R^2 puede tomar valores en el intervalo $(-\infty, 1]$. El mejor valor que se puede obtener es $R^2 = 1$ e indicaría un ajuste perfecto. Viene definido por:

$$R^2 = 1 - \frac{\sum_{l=1}^{p} (y_l - \hat{y}_l)^2}{\sum_{l=1}^{p} (y_l - \bar{y})^2} \tag{2.7}$$

donde $\bar{y} = \frac{1}{p} \sum_{l=1}^{p} y_l$.

2.3.1. Aprendizaje supervisado

Para construir el ROM se han explorado diferentes técnicas de regresión de aprendizaje supervisado, que se detallan en las secciones 2.3.1.1 a 2.3.1.4. Para ello, se ha usado la librería de Python *scikit-learn* [57]. Para decidir qué estimadores emplear se han tenido en cuenta las pautas que presenta esta misma librería y que se pueden consultar en la figura 2.2.

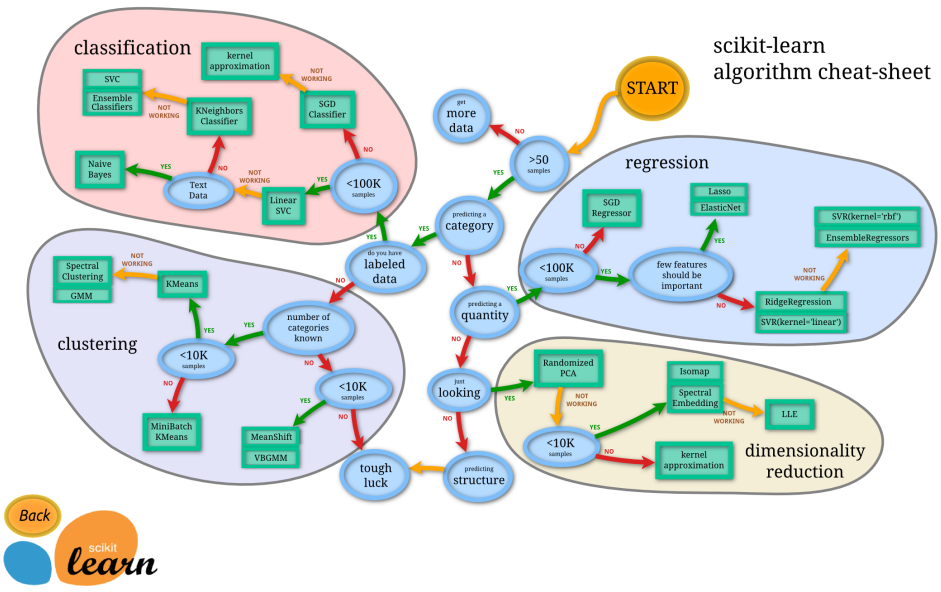


Figura 2.2: Diagrama de decisión de estimadores. Fuente [64].

Para ajustar este tipo de modelos suele necesitarse la definición de una función objetivo que represente cómo de bien se ajusta el modelo a los datos de entrenamiento. En general la función objetivo consta de dos partes:

$$\text{obj} = L + \Omega \tag{2.8}$$

donde L se denomina la pérdida o error de entrenamiento y Ω es el término de regularización que controla la complejidad del modelo y evita el sobreajuste. Una de las opciones más frecuentes es tomar

L como el error medio cuadrático definido como [91]:

$$L = \sum_i (y_i - \hat{y}_i)^2 \tag{2.9}$$

A la hora de entrenar estos modelos suele ser conveniente un procesamiento previo de las variables independientes. Muchos estimadores pueden comportarse de manera inadecuada si la distribución de las variables independientes no se asemeja a una distribución normal (por ejemplo las máquinas de vector soporte al emplear un kernel de base radial). Por ello de manera previa al ajuste de cualquiera de los métodos anteriormente mencionados, se realiza un escalado de los datos descriptivos, sustrayendo la media de la variable y dividiendo por su desviación estándar. Así, las variables independientes quedarían:

$$z = \frac{x - \bar{x}}{\sigma} \tag{2.10}$$

donde z son las variables independientes tras su escalado, $\bar{x}$ la media de estas y σ su desviación estándar (para cada una de sus componentes) [67].

2.3.1.1. Bosques aleatorios - RF

Los bosques aleatorios (RF) son un tipo de estimador de "conjunto". Este conjunto está formado por árboles de decisión, que son estimadores individuales construidos a partir de muestras con reemplazamiento del conjunto de entrenamiento. Además, al dividir cada nodo durante la construcción del árbol, la mejor partición se encuentra de entre todas las características de los datos o de una subsección aleatoria de ellas. El propósito de estas dos fuentes de aleatoriedad es disminuir la varianza del estimador global. De hecho, árboles de decisión individuales exhiben típicamente una gran varianza y tienden a sobreajustarse. Esta aleatoriedad en el bosque induce de alguna forma errores en la predicción desacoplados, y tomando su media pueden llegar a cancelarse. Los bosques aleatorios consiguen una reducción en la varianza aumentando ligeramente su sesgo en algunos casos. En la práctica, esta reducción en la varianza es significativa, teniendo como resultado final un mejor modelo. [65]

Un árbol de decisión consiste en ir dividiendo los datos de entrada en función de alguna de sus características. De esta manera se van creando subgrupos (llamados subnodos) a los que se les asigna una categoría o un valor $\hat{y}_i$. Así, la homogeneidad o pureza de los subnodos va aumentando, lo que conlleva una reducción de L. Existen muchos criterios para hacer estas divisiones, muchas de ellas relacionadas con la pureza de los subnodos, como lo son: índice Gini, χ^2, ganancia de la información y reducción en la varianza. En la implementación de *scikit-learn* el criterio que se sigue es el de reducir L en cada división, que es equivalente al de reducción de la varianza [65]. Por otro lado, si se itera la creación de subnodos hasta el final se obtiene un modelo sobreajustado en el que cada elemento de los datos de entrenamiento constituiría un subnodo. Para evitar esto estaría Ω en la función objetivo. Sin embargo, en la práctica muchas veces se desprecia este término y para solventar este problema se emplean técnicas heurísticas. Alguna de estas técnicas consisten en establecer un límite al cambio de la pureza al realizar una división, imponer que cada subnodo final tenga un número mínimo de elementos de los datos de entrenamiento o limitar la profundidad del árbol. [4, 91]

2.3.1.2. K vecinos próximos - KNn

El principio detrás del método de K vecinos próximos (KNn) es encontrar un número de muestras dentro de los datos de entrenamiento más cercanos en distancia al punto que se quiere predecir y derivarlo de estos. La distancia se calcula en función de las características de estos datos y puede ser, en general, cualquier tipo de métrica: la distancia Euclídea estándar es la opción más común y la que se usa en este trabajo. Las regresiones basadas en vecinos se pueden usar en casos en los que los valores objetivos de los datos sean continuos. Estos valores son asignados al punto predicho a partir de la media de los valores de sus vecinos más próximos. El método más básico usa pesos uniformes: esto es, cada punto de los considerados como más próximos contribuyen uniformemente a la predicción final. Sin

embargo resulta ventajoso pesar cada punto de manera que los más cercanos contribuyan más que los alejados. En la implementación que se emplea en el trabajo se pesarán estos puntos proporcionalmente al inverso de su distancia con el punto requerido. [66]

2.3.1.3. Máquinas de vector soporte - SVM

Una máquina de vector soporte (SVM) construye un hiperplano o conjunto de ellos en un espacio con alta dimensionalidad, que puede ser usado para clasificación o regresión. La idea que subyace en los SVM es buscar uno o varios hiperplanos que separen de forma óptima las características (clases en los problemas de clasificación) de los puntos entrenados, que se han podido proyectar a un espacio de dimensión superior. En este contexto buscar la separación óptima se refiere a buscar el hiperplano que tenga la máxima distancia o margen con los puntos (de ambas clases, en clasificación) que estén más cerca del mismo. A estos puntos se los denominan vectores soporte.

Concretamente, en las aplicaciones de regresión, el objetivo se traduce en encontrar $w \in \mathbb{R}^m$ y $b \in \mathbb{R}$ teniendo un error mínimo para las predicciones dadas por: $w^T \phi(x) + b$, minimizando:

$$\begin{aligned} \min_{w,b,\xi,\xi^*} & \underbrace{\frac{1}{2} w^T w}_{L} + \underbrace{C \sum_{i=1}^{n} \xi_i + \xi_i^*}_{\Omega} \\ \text{s.a.} : & \; y_i - w^T \phi(x_i) - b \leq \varepsilon + \xi_i, \\ & \; w^T \phi(x_i) + b - y_i \leq \varepsilon + \xi_i^*, \\ & \; \xi_i, \xi_i^* \geq 0, i = 1, \ldots, n \end{aligned} \tag{2.11}$$

Intuitivamente, se maximiza el margen (de anchura 2ε) del hiperplano (minimizando $\|w\|^2 = w^T w$) a la vez que se penalizan las muestras cuya predicción se vaya más de ε de su valor real. Estas muestras penalizan la función objetivo con ξ_i o ξ_i^* dependiendo si la predicción se encuentra por encima o por debajo de su valor real. Se resuelve su problema dual que viene dado por:

$$\begin{aligned} \min_{\alpha,\alpha^*} & \frac{1}{2} (\alpha - \alpha^*)^T Q (\alpha - \alpha^*) + \varepsilon l^T (\alpha + \alpha^*) - y^T (\alpha - \alpha^*) \\ \text{s.a.} : & \; l^T (\alpha - \alpha^*) = 0 \\ & \; 0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \ldots, n \end{aligned} \tag{2.12}$$

Donde l es un vector con todas sus componentes iguales a la unidad, Q es una matriz $n \times n$ semidefinida positiva, $Q_{i,j} \equiv K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ denominada kernel. Aquí los vectores de entrenamiento se transforman implícitamente a un espacio de dimensión superior por la función ϕ. La predicción viene dada por:

$$\sum_{i \in SV} (\alpha - \alpha^*) K(x_i, x) + b \tag{2.13}$$

Siendo SV los vectores soporte, es decir, aquellos puntos que caen justo en el margen del hiperplano. Algunas de las opciones del kernel, $K(x_i, x_j)$, son:

- Lineal: $\langle x_i, x_j \rangle$.
- Polinomial: $\left(\gamma \langle x_i, x_j \rangle + r\right)^d$.
- Función de base radial: $\exp\left(-\gamma \left\| x_i - x_j \right\|^2\right)$.
- Sigmoidea: $\tanh\left(\gamma \langle x_i, x_j \rangle + r\right)$.

Donde $\langle \cdot, \cdot \rangle$ representa el producto escalar convencional. Tanto ε, C, γ, r y d son hiperparámetros del modelo que hay que ajustar, así como el kernel que se emplea. [12, 69, 73]

2.3.1.4. Gradiente estocástico descendente - SGD

El gradiente estocástico descendente es un simple pero a la vez eficiente método para ajustar clasificadores o regresores lineales bajo una función de pérdida convexa. El objetivo es encontrar una función lineal $f(x) = w^T x + b$ que nos dé los valores $\hat{y}$ con parámetros del modelo $w \in \mathbb{R}^m$ y $b \in \mathbb{R}$. Para encontrar estos parámetros, se minimiza el error de entrenamiento regularizado, E, definido por la siguiente expresión:

$$E(w,b) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f(x_i))}_{L} + \underbrace{\alpha R(w)}_{\Omega} \tag{2.14}$$

donde $\mathscr{L}$ es una función de pérdida que mide el error del ajuste del modelo y R es el término de regularización que penaliza la complejidad del modelo; $\alpha > 0$ es un hiperparámetro no negativo que controla la intensidad de la regularización. Diferentes elecciones de $\mathscr{L}$ dan como resultado diferentes regresores:

- Hinge: equivalente a la clasificación por SVM. $\mathscr{L}(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$.
- Perceptron: $\mathscr{L}(y_i, f(x_i)) = \max(0, -y_i f(x_i))$.
- Huber modificado: $\mathscr{L}(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))^2$ si $y_i f(x_i) > 1$, y $\mathscr{L}(y_i, f(x_i)) = -4 y_i f(x_i)$ en otro caso.
- Log: equivalente a una regresión logística. $\mathscr{L}(y_i, f(x_i)) = \log(1 + \exp - y_i f(x_i))$.
- Mínimos cuadrados: regresión lineal (Ridge o Lasso dependiendo de R). $\mathscr{L}(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$.
- Huber: menos sensible a *outliers* que mínimos cuadrados. Es equivalente a mínimos cuadrados cuando $|y_i - f(x_i)| \leq \varepsilon$, y $\mathscr{L}(y_i, f(x_i)) = \varepsilon |y_i - f(x_i)| - \frac{1}{2}\varepsilon^2$ en otro caso.
- Insensible a epsilon: equivalente a una regresión por SVM. $\mathscr{L}(y_i, f(x_i)) = \max(0, |y_i - f(x_i)| - \varepsilon)$.

Algunas elecciones comunes para el término de regularización R son:

- Norma $L2$: $R(w) := \frac{1}{2}\sum_{j=1}^{m} w_j^2 = \|w\|_2^2$.
- Norma $L1$: $R(w) := \sum_{j=1}^{m} |w_j|$.
- Red elástica: $R(w) = \frac{\rho}{2}\sum_{j=1}^{m} w_j^2 + (1-\rho)\sum_{j=1}^{m} |w_j|$, una combinación convexa de $L2$ y $L1$ regulada por ρ; se hace igual a $L2$ para $\rho = 1$ y a $L1$ para $\rho = 0$.

El gradiente estocástico descendente es un método de optimización para problemas sin restricciones. Al contrario que el gradiente descendente, SGD aproxima el verdadero gradiente de $E(w,b)$ considerando una muestra concreta de todos los datos de entrenamiento a la vez. En la implementación de *scikit-learn* se emplea una rutina SGD de primer orden. El algoritmo itera sobre las muestras de los datos de entrenamiento y para cada una actualiza los parámetros de acuerdo a la regla:

$$w \leftarrow w - \eta \left[\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial \mathscr{L}(w^T x_i + b, y_i)}{\partial w} \right] \tag{2.15}$$

donde η es el ritmo de aprendizaje que controla el tamaño de los pasos en el espacio de parámetros. b se actualiza de manera similar pero sin el término de regularización. El ritmo de aprendizaje η puede ser constante o decaer gradualmente. Para regresión, el esquema seguido para η es el escalado inverso dado por:

$$\eta^{(t)} = \frac{\eta_0}{t^\theta} \tag{2.16}$$

donde t es el paso temporal del proceso (hay un total de $n \times$ `n_iter` pasos temporales). Tanto η_0, θ y `n_iter` son hiperparámetros del modelo. [68][95]

2.3.2. Interpolación mediante triangulación de Delaunay

Este método consiste en construir una triangulación de Delaunay con los elementos x_i de los datos de entrenamiento, para después realizar una interpolación lineal con los puntos correspondientes al punto requerido de esta triangulación.

Una triangulación de Delaunay es una subdivisión de un objeto m-dimensional en m-símplex o símplices[4], de manera que ningún punto se encuentre dentro de la hiper-esfera circunscrita de ningún símplex [18]. Las propiedades de este tipo de triangulación maximizan el ángulo mínimo de todos los ángulos de los símplex generados. Esto minimiza los símplex estrechos o retorcidos, que a su vez mejora la calidad de la interpolación [10]. Se usa el algoritmo QHull [8] para construir la triangulación. Este se basa en una propiedad matemática que estipula que cada símplex en la triangulación de Delaunay de un conjunto de puntos m-dimensional corresponde a una faceta de la envoltura convexa de la proyección de los puntos a un paraboloide $(m+1)$-dimensional, y viceversa [14].

La localización del punto requerido para la interpolación dentro de la triangulación es esencial. Para ello, el punto se proyecta en el paraboloide $(m+1)$-dimensional y usa el algoritmo QHull [8] para realizar una búsqueda directa de la faceta de la envoltura convexa en $m+1$ dimensiones que lo contiene. En el espacio original m-dimensional, esta faceta corresponde a un símplex de la triangulación de Delaunay que contiene el punto buscado.

Se realiza una interpolación lineal del punto con los vértices del símplex que lo contiene, basándose en coordenadas baricéntricas. Un sistema de coordenadas baricéntricas permite expresar la localización de un punto relativo a los vértices de un m-símplex [83]. Sea $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_{m+1})$ las coordenadas baricéntricas del punto objetivo y $\psi = (\psi_1, \psi_2, \ldots, \psi_{m+1})$ los valores de una característica/propiedad de los datos en los vértices del m-símplex. Entonces el valor de la característica del punto buscado se interpola según:

$$\hat{\psi} = \lambda \cdot \psi^T \tag{2.17}$$

Con todo esto, la predicción final sería $\hat{y}_i = (\hat{\psi}_1, \hat{\psi}_2, \ldots, \hat{\psi}_k)$, con cada $\hat{\psi}$ dada por la expresión (2.17). Este método puede fallar si el punto requerido se encuentra fuera de la triangulación (no contenido por ningún símplex). Para intentar evitar esto, es conveniente introducir puntos a la triangulación que expandan lo máximo su alcance y eviten este problema. En el desarrollo de la metodología general se añaden estos puntos mediante los "casos frontera".

Nótese que pueden encontrarse ciertas similitudes con el método KNn (sección 2.3.1.2). Mientras que en KNn existen ciertos hiperparámetros para ajustar el modelo, en la interpolación mediante triangulaciones de Delaunay no se tiene ninguno; esta es la razón principal por la que no se incluye dentro de las técnicas de aprendizaje supervisado. Por otro lado, en KNn se obtiene el valor final con cierto número de vecinos más próximos, mientras que con Delaunay se hace con los vértices del m-símplex. No se asegura que estos vértices a su vez sean los puntos más próximos al punto requerido, aunque nada impediría que lo fueran.

La metodología para emplear este método para el ROM se describe en [36] y [58] y un ejemplo de aplicación se puede encontrar en [46]. Para el proceso de interpolación y la generación se emplea la implementación de la librería de Python *SciPy* [86].

[4] Un símplex o símplice es un objeto m-dimensional definido por la envoltura convexa de $m+1$ vértices. Por ejemplo, en dos dimensiones, un símplex es un triángulo; en tres es un tetraedro.

Capítulo 3

Metodología

En esta sección se presentan los bloques principales que constituyen la metodología desarrollada en el proyecto. Una de las ventajas que presenta este método es su modularidad. Cada bloque que conforma el proceso tiene un amplio rango de modificación sin tener que alterar directamente el resto. Por ejemplo, se pueden cambiar las físicas que se resuelven en las simulaciones de mesoescala en función de la región que se trate; también se pueden cambiar los modelos de turbulencia en las simulaciones CFD adaptándose a entornos urbanos o montañosos. Esta segmentación, además de una mayor flexibilidad para adaptarse al problema concreto que se trate, proporciona una trazabilidad tanto de errores como mejoras en los resultados finales. Los elementos más importantes son los siguientes y se esquematizan en la figura 3.1:

1. Simulación de mesoescala y selección de eventos[1] representativos (sección 3.1).
2. Simulación de estos eventos con CFD (sección 3.2).
3. Construcción del ROM (sección 3.3).
4. Una vez construido el ROM, la forma de operar consiste en introducir el resultado de una simulación de mesoescala a este para obtener los resultados finales.

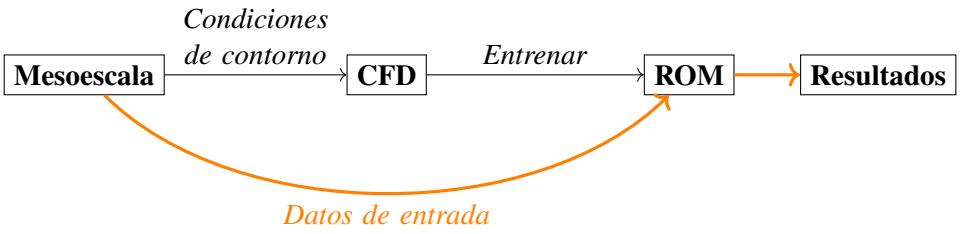


Figura 3.1: Esquema general de la metodología.

3.1. Simulación de mesoescala

La función principal de las simulaciones mesoescala dentro del proceso de la construcción del ROM es la de proporcionar condiciones iniciales y de contorno a las simulaciones CFD. Las simulaciones con WRF se pueden enfocar a dos objetivos principalmente: reanálisis de eventos temporales pasados o predicción. En ambos casos, los software disponibles para la simulación de mesoescala (como WRF)

[1] Se entiende como "evento" al estado meteorológico concreto de una fecha o un instante determinado, por ejemplo, las condiciones meteorológicas del día 1 de mayo del 2018 a las 16:00.

requieren datos de entrada (condiciones iniciales y de contorno) proporcionados por modelos atmosféricos globales, tales como el GFS o IFS. Estos datos, por lo general, se encuentran disponibles online y son de acceso gratuito.

La simulación de mesoescala se centra sobre la región de interés. La idea general es realizarla sobre un periodo de tiempo suficientemente grande como para capturar los eventos más representativos y que con mayor frecuencia se dan históricamente en la región. Una selección de estos eventos más característicos (este proceso se detalla en la sección 3.1.1) se simularían entonces con CFD. De esta manera se tienen simulaciones detalladas de las condiciones meteorológicas más habituales que se puedan dar en la región de estudio. Con ello, el ROM es entrenado con los eventos que mejor definen la climatología que se da en la localización de interés. De esta manera, el ROM podría aplicarse a instantes temporales fuera del periodo simulado por WRF durante su construcción. En el caso aplicado que se desarrolla en este proyecto, se realizan simulaciones WRF de reanálisis sobre un año natural (en concreto el 2018). Podría considerarse este periodo algo reducido para obtener una representación fidedigna de las condiciones meteorológicas históricas que mejor describen esta localización. Sin embargo, al igual que ocurre con las simulaciones CFD, se ha tomado este periodo temporal en aras de simplificar el proceso y hacer la aplicación de la metodología global más rápida (además de menos exigente en recursos computacionales, sólo las simulaciones WRF ya ocupan 175 Gb).

Otro aspecto importante a considerar es la frecuencia, dentro del periodo temporal simulado con WRF, con la que se almacenan resultados de un evento específico. Esta frecuencia puede ser variable pero no muy elevada ni muy baja por un motivo principalmente. Tiene que ser correcta para muestrear el periodo de tiempo total adecuadamente; si se almacenan resultados cada minuto es lógico pensar que las condiciones meteorológicas han cambiado poco en este intervalo de tiempo, por lo que se tendrán resultados de estas simulaciones prácticamente idénticas; por el contrario, si las simulaciones almacenadas son semanales, es altamente probable que se esté perdiendo información en este periodo y cambios significativos en sus condiciones meteorológicas. A todo ello se le suman aspectos algo más técnicos como el tiempo de procesamiento posterior de los resultados (aumenta cuantos más eventos se almacenen) o la memoria que ocupa en disco (cada evento de las simulaciones WRF realizadas para el caso aplicado de este proyecto ocupan aproximadamente ~ 20 Mb). Al final esta frecuencia de almacenamiento se determina de manera heurística teniendo en cuenta estos factores. En la aplicación de la metodología que se lleva a cabo en este trabajo, se almacenan eventos de WRF cada hora.

Para la construcción del ROM se emplean simulaciones de WRF de reanálisis. Sin embargo, a la hora de emplearlo, los datos de entrada pueden ser tanto de reanálisis para estudiar instantes pretéritos como productos WRF de predicción que darían resultados de tiempos futuros. De manera adicional, en el apéndice A se presenta un breve estudio llevado a cabo en el contexto de este TFM de los efectos que tienen algunos de los parámetros de operación de WRF, como las diferentes físicas que se resuelven o la resolución espacial empleada en su dominio.

3.1.1. Selección de eventos

En este apartado se expone el proceso de selección de los eventos de las simulaciones WRF que se simularán en CFD para construir finalmente el ROM. Se busca que el ROM opere de forma adecuada bajo cualquier condición meteorológica que se pueda dar en la zona de estudio. En definitiva, para generar el ROM, lo que se hace es ajustar algún tipo de modelo de regresión. Para que estos modelos devuelvan resultados aceptables lo óptimo es entrenarlos con datos lo más semejantes a estas condiciones. Por ello, escoger eventos que representen y describan lo mejor posible las condiciones meteorológicas que se dan en la región de estudio es primordial. Por un lado, se puede pensar, para ahorrarse el proceso de selección de eventos, en seleccionarlos todos. De esta manera seguro que se entrenaría adecuadamente el ROM. Sin embargo, el inconveniente principal de esto es que habría que realizar una gran cantidad de simulaciones CFD (una por cada evento) que requieren un elevado tiempo de cómputo. Esto es precisamente lo que se trata de evitar con el desarrollo de la metodología presentada en el proyecto.

Debido a lo anterior, los eventos que resultarían óptimos para entrenar estos modelos de regresión son aquellos que cubran lo mejor posible el espacio de condiciones meteorológicas posibles (y

especialmente las más frecuentes) de la zona de estudio. Estos modelos, en general, no se comportan especialmente bien en los límites del espacio de fases definido por sus datos de entrenamiento, y pueden llegar incluso a fallar si se les requiere algún punto que se encuentre fuera del espacio de fases recorrido. Por ello los eventos extremos (por ejemplo, en los que se den velocidades del viento máximas) también cobran relevancia a la hora de caracterizar el periodo. Además, estos eventos cobran especial interés en aplicaciones como cálculos de esfuerzos sobre edificios o condiciones seguras de operación de un teleférico en una pista de esquí, por ejemplo.

En la metodología propuesta en este trabajo se determinan los eventos que se escogen para simular en CFD (y posteriormente entrenar el ROM) siguiendo el siguiente procedimiento que explora de manera sistemática los eventos simulados con WRF y tiene en cuenta los requerimientos expuestos con anterioridad:

1. Se calcula el módulo y dirección del viento a 50 m sobre la superficie en el centro del dominio de estudio y se descartan aquellos eventos con una velocidad inferior a 2 [m/s]. La razón para ello es que las situaciones de calma tienen (a priori) poco interés desde el punto de vista de aplicaciones del modelo, por lo que recorrer esta parte del espacio de fases es en principio poco útil.

2. Se calcula la rosa de los vientos en este punto; esto es, se agrupan los resultados en intervalos discretos de velocidad y dirección de viento (ver figura 4.2). El número de direcciones escogido es 64 y el de velocidades 10. Estas cantidades se eligen de manera que los intervalos discreticen lo máximo posible el rango de valores de velocidad y dirección pero manteniendo un valor estadístico relevante (con esta elección las direcciones principales superan los 150 eventos).

3. Se recorren estos intervalos, para cada dirección, seleccionando los eventos extremos (aquellos con velocidad máxima) hasta un máximo de 64 (un evento extremo por cada posible intervalo de dirección).

4. Se exploran los intervalos de velocidad en función de su frecuencia mayor (independientemente de su dirección), escogiendo un evento perteneciente a cada uno de estos. En este proceso se prohíben los intervalos adyacentes a los ya seleccionados para forzar una mayor exploración de todo el espacio. El comienzo de esta desestimación de intervalos lo marca el intervalo de velocidad con mayor frecuencia (el primero que se elige). Por ejemplo, si se selecciona el intervalo de velocidad 5 para el intervalo de dirección 24, el intervalo de velocidad 5 queda prohibido para las direcciones 23 y 25, así como los intervalos de velocidad 4 y 6 para la dirección 24.

5. Por último, los "casos frontera" son unos eventos artificiales que se seleccionan especialmente para dar mayor soporte a la interpolación mediante triangulaciones de Delaunay (sección 2.3.2). Para generarlos se recorren 16 intervalos de dirección equiseparados comenzando por el primero. De cada una de estas direcciones se escoge un evento perteneciente al intervalo de velocidad más frecuente independientemente de los eventos seleccionados en los puntos anteriores. La idea es identificar las condiciones meteorológicas más frecuentes para cada una de estas direcciones y modificar la intensidad de viento. Seguidamente se dividen los valores de velocidad del viento por la velocidad máxima de cada uno de los eventos. Más adelante, se obtiene la velocidad máxima de todos los eventos seleccionados previamente, se eleva un 10 % y se multiplica este valor a los campos de velocidad de los eventos de estas 16 direcciones. Al ser físicamente posibles, se dejan para el resto de modelos de regresión además de la interpolación mediante triangulaciones de Delaunay.

Esta metodología ha sido desarrollada e implementada en Python por el autor de este trabajo. El código se puede consultar en el apéndice B.3.

3.2. Simulación de CFD

Las simulaciones CFD que se emplean resuelven el campo de velocidades de la región de interés con una resolución espacial alta, teniendo en cuenta detalles de la orografía que se escapan del alcance de las simulaciones de mesoescala. El dominio WRF se ha definido de tal forma que el dominio CFD se encuentra centrado sobre este; esto se debe a que los resultados de la simulación WRF tienen mejor calidad (no afectan tanto las condiciones de contorno del dominio) en los puntos centrales de la región a simular. Se simulan los eventos determinados en el apartado anterior (sección 3.1.1) y los resultados obtenidos son usados como valores objetivo para entrenar el ROM. Se plantean dos retos principales para llevar a cabo estas simulaciones CFD. El primero es referente a la malla computacional empleada y cómo se construye. El segundo radica en especificar de manera adecuada las condiciones de contorno de las simulaciones CFD para simular unas u otras condiciones meteorológicas determinadas por las simulaciones de WRF.

Los dominios (mallas computacionales) para las simulaciones CFD de este tipo (simulaciones de viento sobre un terreno) suelen ser en general rectangulares y se extienden aguas abajo para resolver las estelas y que no interfieran con las condiciones de contorno en la salida [59, 94]. Si además se quiere estudiar el viento con diferentes direcciones, es normal emplear una malla computacional orientada para cada una de estas. Esto obliga a realizar diferentes mallas para cada dirección y luego desechar todas las zonas de estos dominios que no sean comunes. La creación de este tipo de mallas computacionales no es sencilla, requiere tiempo y habilidad, además añade complejidad y tiempo de cálculo al proceso total. Lo que se propone en esta metodología es crear un dominio cuyos límites en el plano horizontal sean circulares o cuasi-circulares haciéndolo válido para cualquier dirección. Esta forma de operar presenta una ventaja principal frente a los métodos más extendidos explicados anteriormente; se evita un tiempo y esfuerzo sustancial al no tener que generar un gran número de mallas para las diferentes direcciones del viento. Por otro lado, señalar que la parte exterior del dominio simulado es descartado para eliminar posibles resultados espurios causados por la cercanía a las condiciones de contorno y dejar que el flujo se desarrolle correctamente de acuerdo a la orografía local.

En relación al segundo reto que se plantea en esta sección, determinar condiciones de contorno a una simulación CFD no es una tarea trivial. Una elección estándar de estas condiciones de contorno pasa por fijar un valor determinado para la velocidad de entrada, un perfil de velocidades en altura o estipular unos valores de la presión [54]. Lo que se busca es capturar lo mejor posible las condiciones meteorológicas de un evento determinado. Para transmitir esto a las simulaciones CFD se opta por una condición de contorno que tenga en cuenta el flujo de viento simulado en el interior del dominio y se adapte a él. Esta condición exige unos valores iniciales para la velocidad en las celdas que conforman el dominio de simulación. Por ello es necesario obtener los valores de las simulaciones WRF e introducirlos a OpenFOAM de manera adecuada.

Para ello, primero se extraen los datos WRF así como las coordenadas de su malla computacional. Por otra parte, se obtienen las coordenadas de las celdas del dominio CFD y se transforman a las mismas que tiene WRF (por lo general los sistemas de coordenadas que emplean ambos software no coinciden). Por último, se interpolan los datos WRF a las celdas que conforman los contornos en el plano horizontal del dominio CFD. Todo este proceso se ha implementado por el autor de este trabajo en Python. El código resultante se puede consultar en el apéndice B.2. En cuanto a la condición de contorno para la turbulencia, se ha impuesto una intensidad turbulenta genérica para todos los eventos en aras de simplificar el proceso.

3.3. ROM

El ROM se compone de modelos de regresión que toman como valores objetivo los resultados del modelo CFD y como valores descriptores los de WRF. Una aproximación para entrenar los modelos de regresión pasa por construir uno único al que se introduzca directamente variables descriptivas para definir la localización del punto (latitud, longitud y altura, por ejemplo). Durante el desarrollo de la

metodología se exploró esta posibilidad, pero se descartó debido a la baja calidad de los resultados obtenidos. Se hace fehaciente el hecho de que este modelo único no es capaz de distinguir correctamente la localización espacial de los puntos requeridos a través de variables descriptivas. Por ello se opta por construir modelos de regresión para cada localización sobre una "malla puente" que no requieran variables descriptivas para incluir la localización espacial. De esta manera, se construyen de manera independiente tantos regresores como puntos tenga esta malla. Sin embargo, haciéndola suficientemente densa, los resultados finales se pueden obtener en cualquier localización del dominio mediante interpolaciones. Notar que la elección del modelo de regresión no se restringe a los expuestos en las secciones 2.3.1 y 2.3.2 sino que el procedimiento es lo suficientemente flexible como para admitir un tipo diferente como podrían ser las redes neuronales. No obstante, en la aplicación de esta metodología (capítulo 4) se comparan los resultados obtenidos con un ROM construido a partir de los diferentes métodos de regresión expuestos en la sección 2.3. A continuación se expone el proceso para elaborar la malla puente, las ventajas que proporciona y cómo se tratan los datos de las simulaciones de WRF y CFD.

La malla puente se construye sobre la región de estudio; horizontalmente se constituye por puntos equidistantes y verticalmente se compone de niveles η' definidos como:

$$h(\eta') = \eta' \cdot (h_t - h_s) + h_s \tag{3.1}$$

donde $h(\eta')$ nos da la altura para un nivel η' dado, h_t es la altura máxima de la malla computacional CFD y h_s es la elevación del terreno para del que se calcula su altura; $\eta' \in [0,1]$, de forma que $\eta' = 0$ se corresponde con la altura del terreno y con $\eta' = 1$ a la máxima. En la figura 4.5 se presenta un ejemplo de una malla de este tipo. La similitud con la malla computacional empleada por WRF es alta. De hecho, horizontalmente las dos se construyen de la misma manera, la diferencia esencial está en los niveles verticales. En WRF estos se definen en función de las presiones, por lo que cada evento tiene unos niveles verticales diferentes. Sin embargo los niveles η' son constantes.

Tanto las simulaciones mesoescala como las de CFD se calculan sobre mallas computacionales diferentes y por lo tanto los resultados de cada una no coinciden espacialmente. La malla puente definida anteriormente sirve de soporte para establecer concordancia espacial entre WRF y CFD y poder entrenar los modelos de regresión en los puntos que la conforman. Estos datos se interpolan de manera similar al proceso que se sigue en la sección 3.2 para establecer condiciones de contorno a las simulaciones CFD. Mencionar que en el caso aplicado que se desarrolla en el trabajo (capítulo 4) las variables empleadas de los resultados de las simulaciones (tanto WRF como CFD) son las componentes de la velocidad en las tres direcciones espaciales. Sin embargo, esto no restringe a 3 el número de variables objetivo o descriptoras ni que tengan que ser componentes de la velocidad; otras variables como la presión o temperatura podrían añadirse a estos conjuntos indistintamente.

Una pregunta que puede surgir en todo el proceso es la de por qué no usar la malla computacional de la CFD como malla puente directamente. Uno de los motivos es el de establecer cierta independencia entre los diferentes elementos de la metodología; es decir, que la resolución de estas simulaciones no defina la cantidad de regresores que se entrena ni su localización. Además, la creación de una malla puente según la manera descrita asegura que esta sea una malla estructurada, lo que favorece las interpolaciones sobre ella.

En el apéndice B.1 se presenta el código principal implementado en Python por el autor de este proyecto que se encarga de construir el ROM.

Capítulo 4

Caso aplicado

En este capítulo se presentan los resultados de aplicar la metodología expuesta en el capítulo 3 a un caso práctico.

La región geográfica sobre la que se centra el caso aplicado es el Pirineo aragonés, más concretamente entre los valles del Aragón y Tena, en el paraje conocido como sierra de la Partacua. Se ha escogido esta localización porque la orografía del terreno es compleja (la zona cuenta con numerosas paredes, canales y corredores con desniveles del orden de 600 m en distancias horizontales menores de 200 m) y resulta adecuada para poner a prueba la metodología de manera exigente.

El intervalo temporal de análisis es el año 2018 completo, con una resolución de 1 hora entre eventos. Los datos de entrada para WRF se han obtenido del conjunto ds090.0 [50] generados mediante reanálisis con el modelo global GFS. Se construyen 4 dominios rectangulares anidados para realizar las simulaciones WRF (ver figura 4.1). La relación de tamaño entre ellos es de 1 : 3, y el más pequeño (del que se obtienen los productos que son empleados por la CFD) tiene una extensión de $\sim$ 81 km en la dirección este-oeste y $\sim$ 54 km en la norte-sur con una resolución espacial de $\sim$ 900 m.

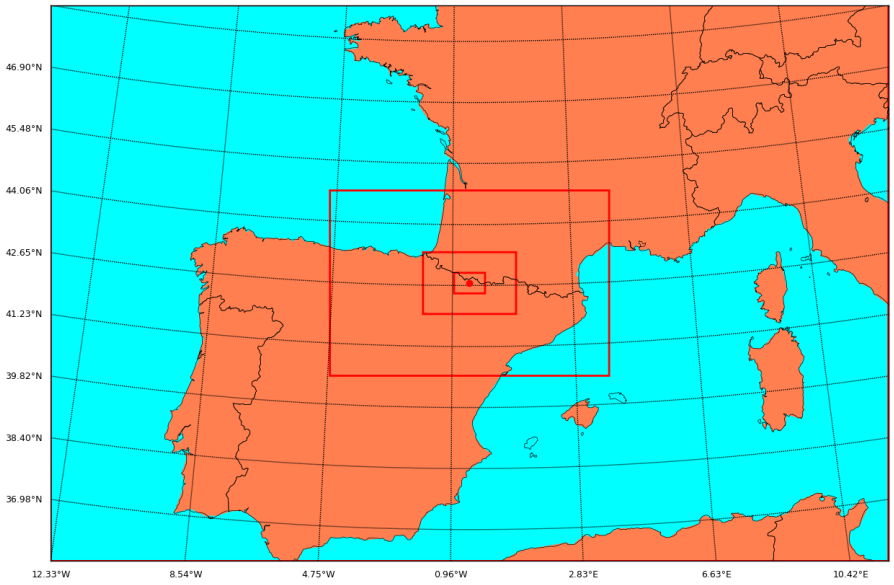
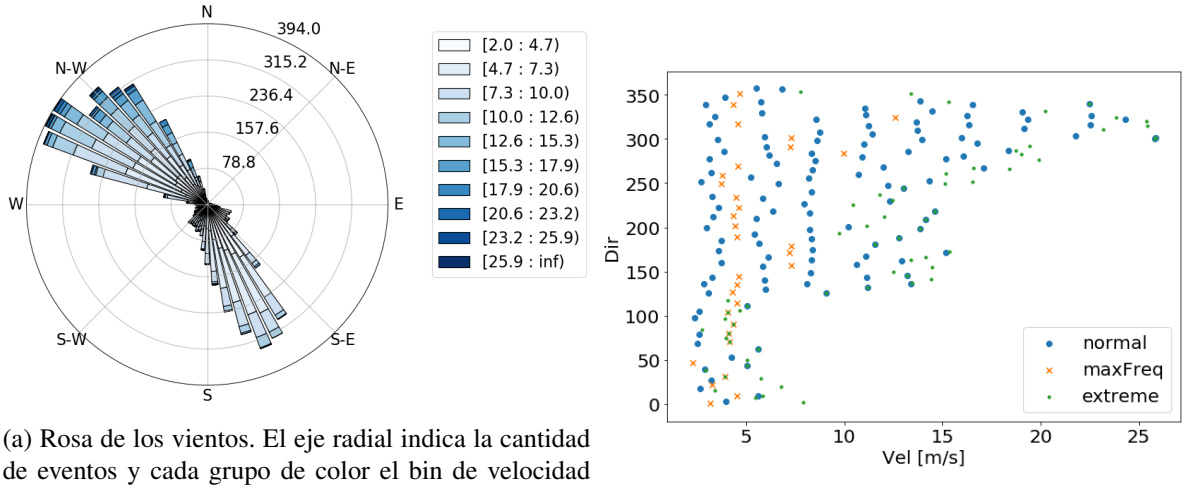


Figura 4.1: Dominio de las simulaciones de WRF.

Para escoger los eventos que se simulan en CFD se sigue el procedimiento descrito en la sección 3.1.1. La rosa de los vientos se realiza con 64 intervalos para la dirección del viento y 10 para su módulo. En total se han seleccionado 193 eventos significativos (más 16 casos *frontera*), que son los que se simulan con CFD. En la figura 4.2 se presentan la rosa de los vientos y la velocidad y dirección de estos eventos.

La figura 4.3 muestra el dominio del último anidamiento de la simulación de mesoescala, donde los



(a) Rosa de los vientos. El eje radial indica la cantidad de eventos y cada grupo de color el bin de velocidad $[m/s]$.

(b) Eventos seleccionados.

Figura 4.2: Selección de eventos de WRF para el caso aplicado.

puntos verdes indican los nodos de la malla. En esta imagen, el círculo azul delimita el dominio CFD. Concretamente, la malla computacional CFD se encuentra centrada en las coordenadas $42{,}7°N - 0{,}36°O$

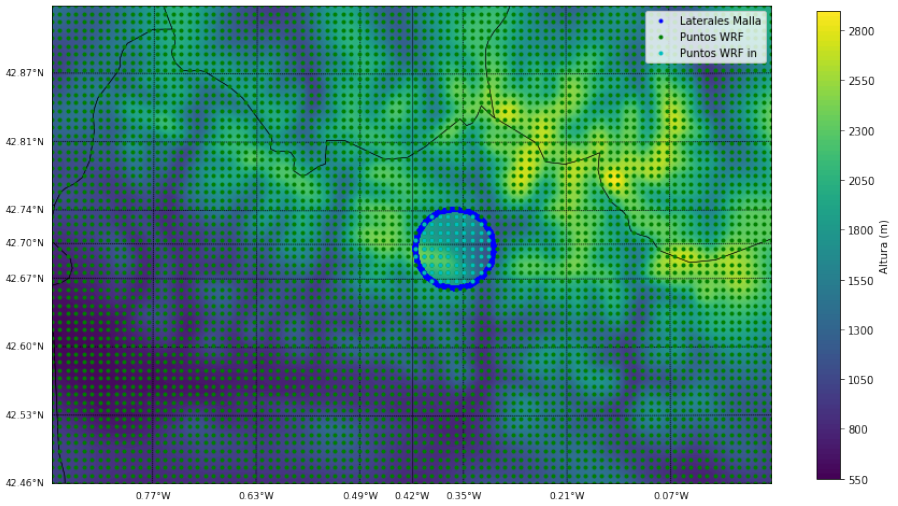
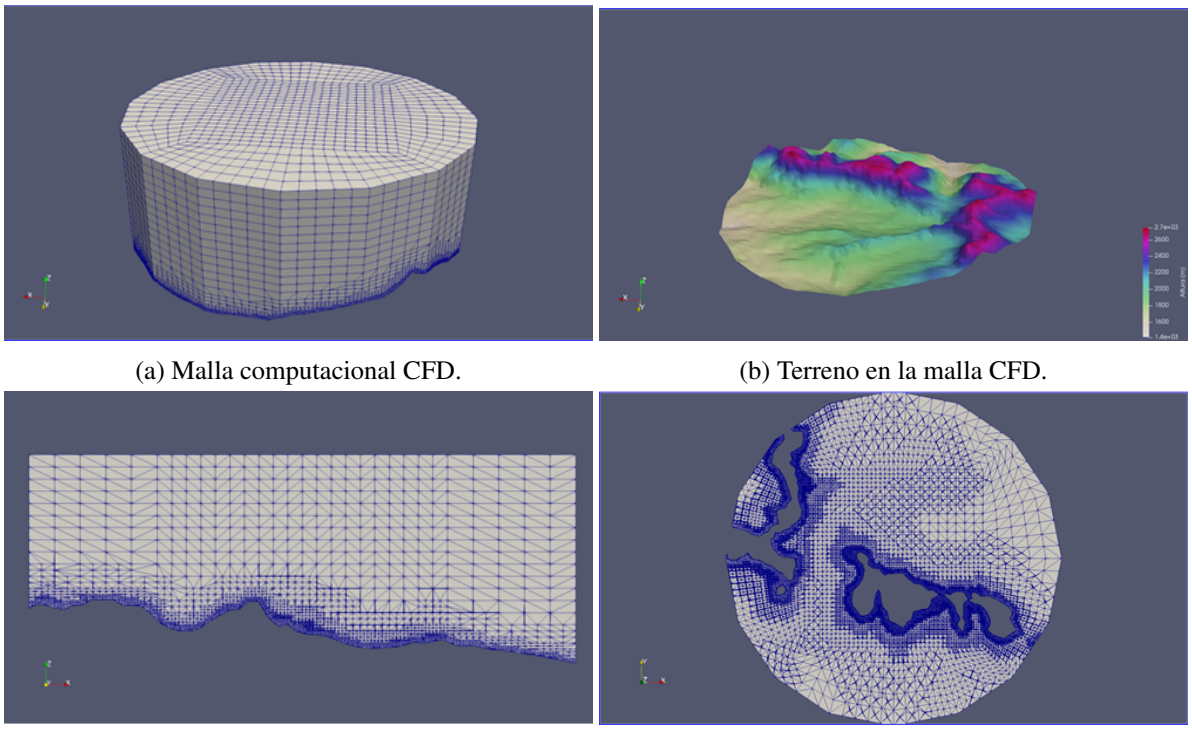


Figura 4.3: Localización del dominio CFD dentro del de WRF.

y consta de unas 250000 celdas[1]. La malla está refinada cerca de la superficie, donde la resolución llega a los 12 m, haciéndose más grosera según la distancia al suelo aumenta. El dominio se define como una figura cilíndrica de radio 9,8 km cuyos laterales se componen de 16 caras planas, con una altura de 5 km sobre el nivel del mar (la altura mínima de la superficie en el dominio CFD es de $\sim$ 1400 m y la máxima de $\sim$ 2700 m). La cara inferior se adapta a la orografía del terreno. En la figura 4.4 se presentan imágenes con el dominio CFD empleado. Con el fin de minimizar la influencia de las condiciones de contorno en los resultados finales, se descarta la solución obtenida en la corona externa ($\sim$ 1 km desde los laterales) y en los $\sim$ 500 m superiores del dominio. Adicionalmente a los $193 + 16$ eventos significativos seleccionados con anterioridad se simulan en CFD 80 más; se tratan de eventos totalmente aleatorios dentro del año escogido para probar el ROM final sobre estos. En el apéndice C se pueden consultar los resultados detallados obtenidos de estas simulaciones.

[1]Esta cantidad de celdas para resolver un dominio CFD de estas dimensiones es claramente escaso para obtener unos resultados que cumplan con los estándares generales de la CFD. Sin embargo, el tamaño de la malla está estrechamente relacionada con el tiempo y memoria requerida por una simulación, por lo que se ha reducido este número para simplificar la aplicación de la metodología.



(a) Malla computacional CFD.

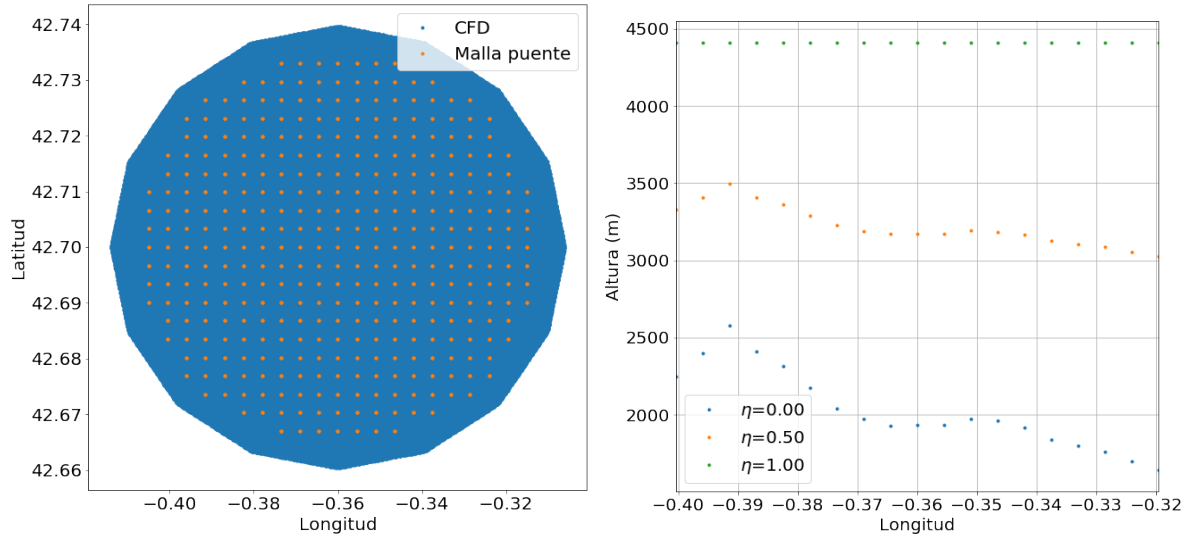
(b) Terreno en la malla CFD.

(c) Malla computacional CFD. Plano lateral.

(d) Malla computacional CFD. Plano cenital

Figura 4.4: Malla computacional para las simulaciones CFD.

La malla puente sobre la que se define el ROM consta de 21 puntos de diámetro y tres niveles η' ($\eta' = \{0,0{,}5,1\}$). En la imagen 4.5 se presenta la malla puente empleada.



(a) Vista cenital de la malla puente.

(b) Sección lateral de la malla puente.

Figura 4.5: Malla puente.

4.1. Comparación de regresores

En esta sección se analiza cuáles son los resultados obtenidos por cada regresor explicado en las secciones 2.3.1 y 2.3.2. De todos ellos se ha explorado de manera sucinta su espacio de hiperparámetros

y se escogen los siguientes:

- **RF**: *número de árboles*= 500, *profundidad del bosque*= 50 y el *criterio calidad de separación de los nodos*: error cuadrático medio (equivalente a reducción de varianza).
- **KNn**: *Número de vecinos próximos*= 5, *Norma para la distancia*: *L2* y *Pesos*: inverso de la distancia.
- **SVM**: *Kernel*: función de base radial, $\varepsilon = 0{,}5$ y $C = 1$.
- **SGD**: $\mathscr{L}$: Huber, $\varepsilon = 0{,}5$, *R*: norma *L2*, $\alpha = 1e-3$, $\eta_0 = 0{,}1$, $\theta = 0{,}1$ y $n\_iter = 1000$

Todos ellos se entrenan con los $193 + 16$ eventos simulados en CFD. En la figura 4.6 se presentan los valores de R^2 para cada una de las componentes del viento (U_x, U_y, U_z) obtenidos por los diferentes regresores.

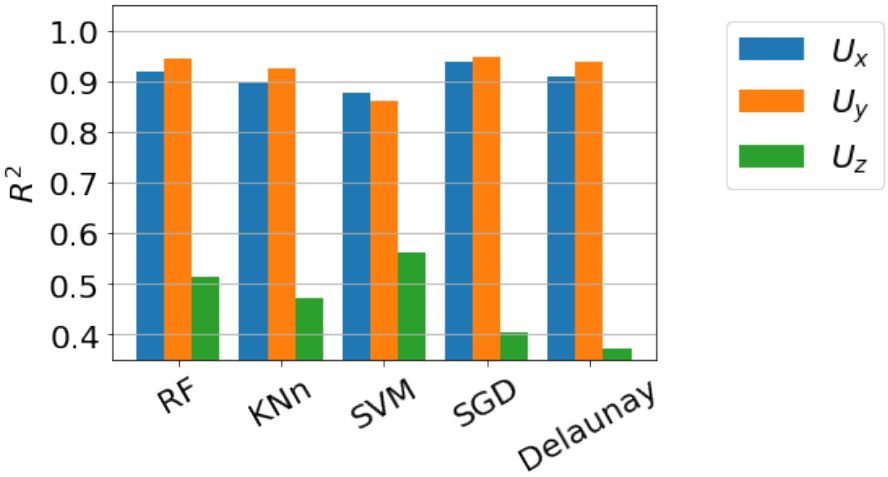


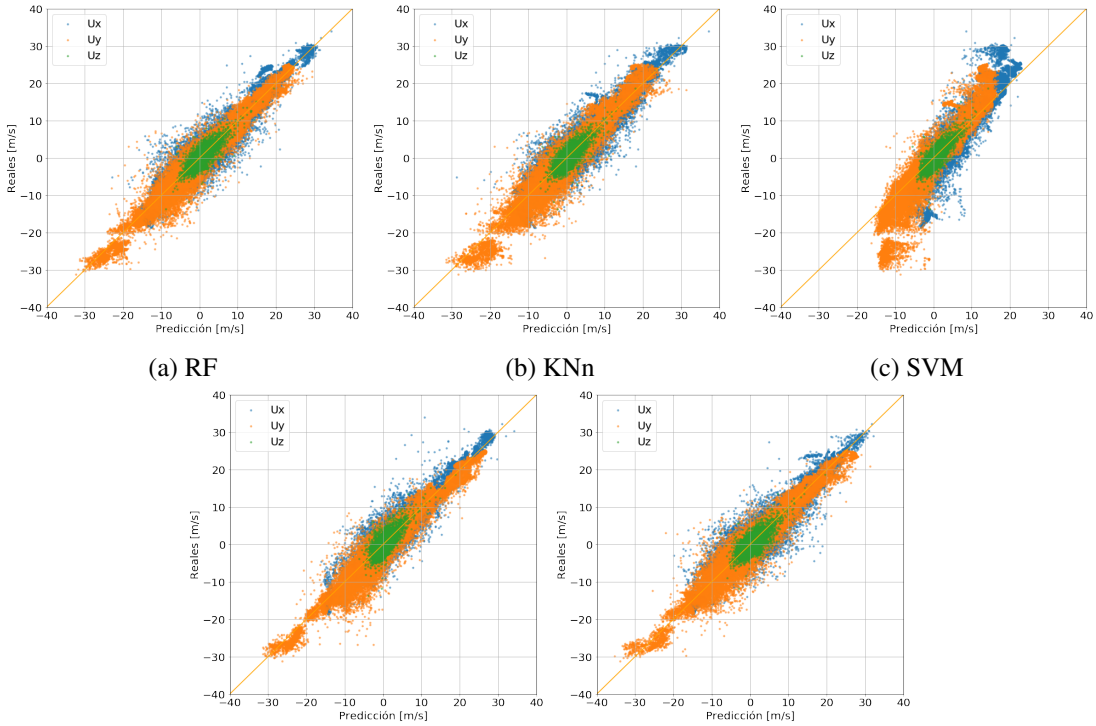
Figura 4.6: Valores de R^2 para cada una de las componentes del viento para diferentes regresores.

Los valores de R^2 para las componentes de viento U_x y U_y son en general bastante similares entre los diferentes modelos y satisfactorios (mayores a 0,85) en todos los casos ,si bien SVM se encontraría algo por debajo. Para la componente U_z sin embargo, R^2 obtiene valores mucho menores (por debajo de 0,57) lo que indica que esta componente no ha sido capturada tan bien por los modelos; en este caso es SVM el que mejor se comporta.

Para estudiar en más detalle los resultados obtenidos, en la figura 4.7 se muestran los productos finales para cada uno de los regresores frente a los valores reales de los casos test. Se aprecia que se captura la tendencia general de los datos aunque se observa una dispersión notable en los resultados obtenidos. De la exploración directa de esta imagen se ve que SVM está afectado por algún tipo de sesgo (inclinación de la nube de puntos con respecto a la diagonal). La dispersión y la presencia de *outliers* parece independiente del modelo de regresión empleado y es similar en todos ellos.

La figura 4.8 presenta diferentes percentiles de los errores absolutos obtenidos en cada componente para los distintos regresores. De nuevo se observa un comportamiento similar para U_x y U_y con todos los regresores; en esta ocasión KNn arroja los peores resultados en comparación al resto. Los errores absolutos para el percentil 85 % son de $\sim 2{,}5$ [m/s], teniendo en cuenta el rango de valores para estas componentes (valores absolutos de hasta 30 m/s), se puede considerar que los resultados obtenidos en estas variables son aceptables. Destacar en este caso el desempeño del modelo SGD con el que se obtienen errores menores de ~ 2 [m/s]. En términos absolutos, los errores para U_z son menores ($\sim 0{,}8$ [m/s]) que para el resto de componentes. Sin embargo su rango de valores hace que este error sea mucho más significativo, obteniendo unos resultados, pese a ser tolerables, algo peores a los de U_x y U_y.

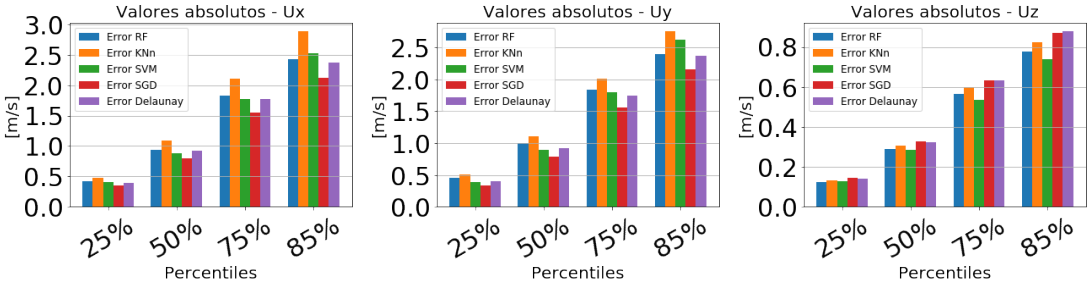
En general se observa que los comportamientos de los modelos están claramente diferenciados para las componentes U_x y U_y por un lado y U_z por otro. Esto podría indicar que una elección diferente de



(a) RF (b) KNn (c) SVM

(d) SGD (e) Triangulación Delaunay

Figura 4.7: Resultados obtenidos por el ROM (eje Y) frente a los valores reales (eje X) para diferentes regresores.



(a) U_x (b) U_y (c) U_z

Figura 4.8: Valores de los percentiles del error absoluto para cada una de las componentes del viento determinadas con los diferentes regresores.

modelos de regresión (o los mismos con otros hiperparámetros) para cada uno de estos grupos podría ser beneficioso para el producto final del ROM. Por otro lado, una exploración más exhaustiva de hiperparámetros de los modelos potencialmente permitiría un rendimiento superior del ROM. Otra posible mejora del ROM pasa por la adición de más variables independientes (como la helicidad o vorticidad) además del campo de velocidades de WRF que se usa actualmente.

4.2. Influencia de la altura

Con el objetivo de explorar en más profundidad estos resultados, se estudia la influencia que pudiera tener η' sobre ellos. Para ello se estudia el caso particular del modelo SGD y se analizan los resultados de manera independiente para los tres niveles η'. En la figura 4.9 se encuentran los valores R^2 obtenidos para los tres valores de η' empleando el modelo SGD. En esta imagen se aprecia que las altitudes bajas ($\eta' = 0$) favorecen el ajuste de la componente U_z mientras que para U_x y U_y no resulta una condición tan

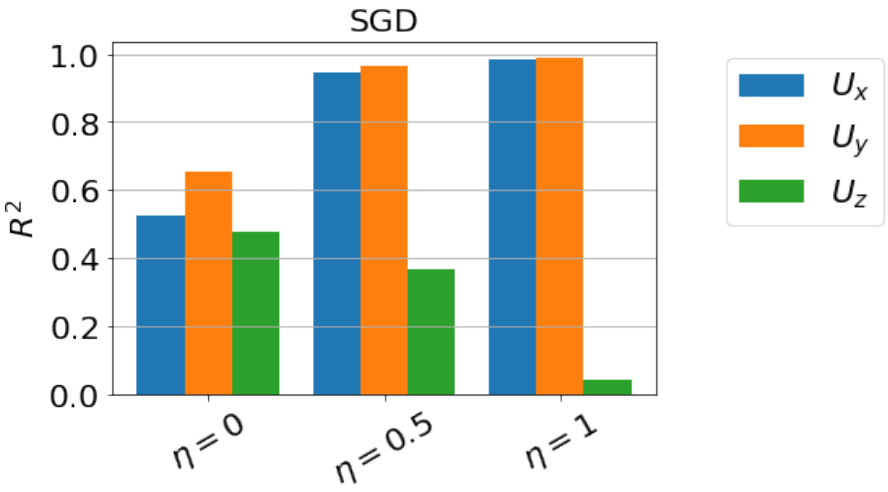
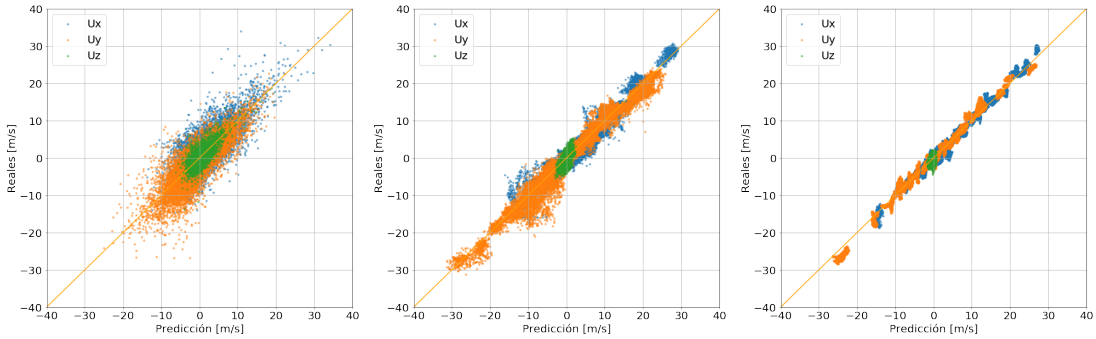


Figura 4.9: Valores de R^2 para cada una de las componentes del viento para diferentes valores de η' con el regresor SGD.

conveniente. Este comportamiento se invierte a medida que se sube en altura, haciendo que el valor de R^2 para la componente vertical de la velocidad sea casi 0. En la imagen 4.10 se presentan los resultados obtenidos en este caso y puede encontrarse una razón a este valor tan bajo para R^2, dado que los valores predichos por el modelo para $\eta' = 1$ en la componente U_z son prácticamente constantes sin atender a los valores de las variables independientes.

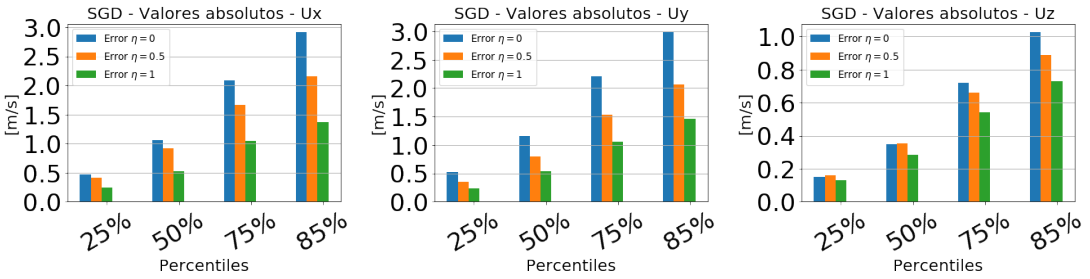


(a) $\eta' = 0$ (b) $\eta' = 0{,}5$ (c) $\eta' = 1$

Figura 4.10: Resultados obtenidos por el ROM (eje Y) frente a los valores reales (eje X) para diferentes valores de η' con el regresor SGD.

También se observan ciertas concentraciones de las predicciones (alineamiento parcial de manera vertical de algunos puntos de la gráfica) para otras componentes del viento. Esto podría indicar que los eventos seleccionados para el entrenamiento en estas alturas tienen las velocidades algo más concentradas y no forman un espectro tan continuo como en altitudes menores. De esta manera el modelo se encuentra influenciado altamente por estos valores más discretizados y no captura de manera tan eficiente valores de la velocidad diferentes. Otra posible selección de eventos de entrenamiento podría ayudar a solventar este inconveniente. Por otro lado, se observa claramente en la figura que el origen principal de la dispersión de los resultados y los *outliers* se concentra en las predicciones para $\eta' = 0$. A medida que se sube en altura esta dispersión va desapareciendo y los puntos se acercan más a la diagonal, obteniendo unas predicciones más acertadas a la realidad.

Para estudiar los errores de cada componente se exponen los percentiles del error absoluto en la figura 4.11. Aquí se ve de manera fehaciente que el error del modelo se concentra primordialmente en los niveles de altura más bajos.



(a) U_x (b) U_y (c) U_z

Figura 4.11: Valores de los percentiles del error absoluto para cada una de las componentes del viento determinadas para diferentes valores de η' con el regresor SGD.

Un estudio pormenorizado del nivel $\eta' = 0$ ayudaría enormemente a la subsiguiente mejora del rendimiento global del ROM. Entre las razones por las que se concentra el error en las alturas más bajas, una sería una posible dependencia espacial con algún punto de la malla puente (una localización especialmente compleja del punto en cuestión, como el fondo de un valle por ejemplo). Otra posibilidad es que uno o varios eventos test correspondan a unas condiciones excepcionalmente anómalas que no estén representadas por los datos de entrenamiento. También es posible que en este nivel inferior, los detalles de la orografía hagan que la diferencia entre los campos de velocidad obtenidos por WRF y CFD sean máximas, y que el modelo de regresión propuesto sea insuficiente para capturarlas.

4.3. Influencia del número de eventos de entrenamiento

Se expone en esta sección la influencia en los resultados finales del número de eventos utilizados para el entrenamiento del ROM, empleando de manera particular el regresor SGD. Se construye el ROM empleando el regresor SGD entrenándolo con diferente número de eventos: 16 (casos frontera), 50, 100, 150 y 209 (todos los disponibles). En la figura 4.12 se encuentran los valores R^2 obtenidos en este caso. Exceptuando la situación en la que se entrena el regresor sólo con los casos frontera, no

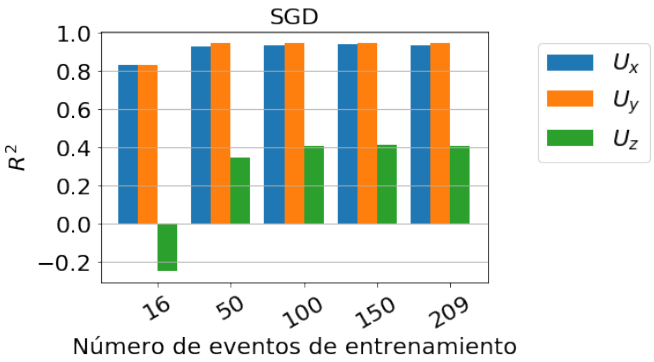
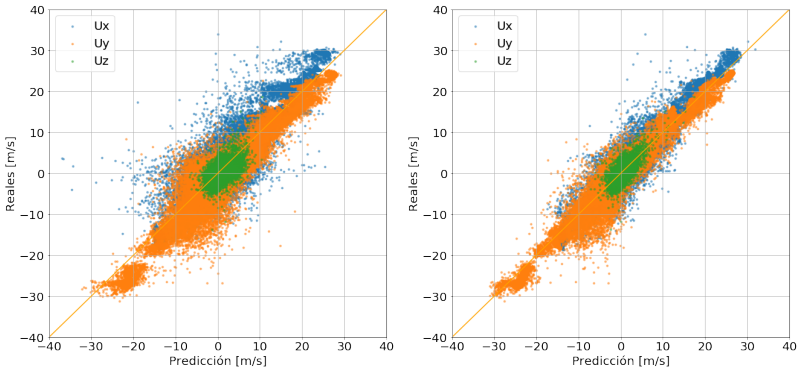
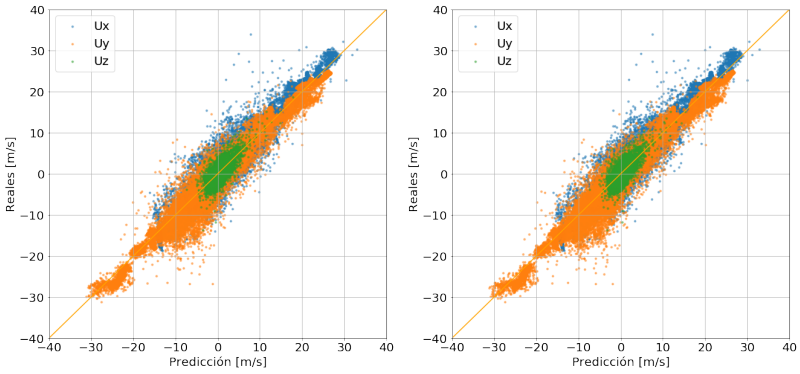


Figura 4.12: Valores de R^2 para cada una de las componentes del viento para diferentes valores del número de eventos de entrenamiento con el regresor SGD.

se aprecia una diferencia significativa en el resto. Con un valor negativo para R^2 se hace evidente que entrenar el modelo únicamente con los casos frontera no es efectivo. Este hecho se corrobora también en la imagen 4.13, que presenta los resultados obtenidos frente a los test para estas condiciones (en la figura 4.7d se encuentran los resultados para 209 eventos de entrenamiento). De nuevo, excluyendo el caso con 16 eventos de entrenamiento, las gráficas son prácticamente idénticas (incluyendo la figura 4.7d). En la situación con 16 eventos de entrenamiento se observa que el error y la dispersión asociadas a esa predicción son mayores que en el resto. La figura 4.14 muestra los percentiles del error absoluto



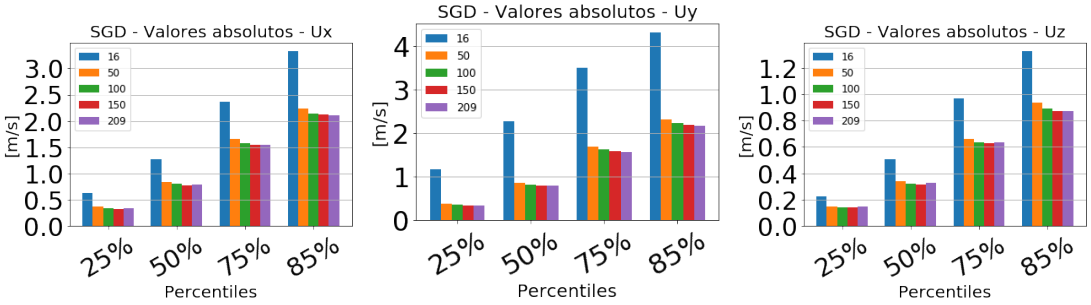
(a) 16 eventos de entrenamiento (b) 50 eventos de entrenamiento



(c) 100 eventos de entrenamiento (d) 150 eventos de entrenamiento

Figura 4.13: Resultados obtenidos por el ROM (eje Y) frente a los valores reales (eje X) para diferentes valores del número de eventos de entrenamiento con el regresor SGD.

obtenido.



(a) U_x (b) U_y (c) U_z

Figura 4.14: Valores de los percentiles del error absoluto para cada una de las componentes del viento determinadas para diferentes valores del número de eventos de entrenamiento con el regresor SGD.

Esta pone en evidencia que el número de eventos con los que se entrena el ROM no es significativo en los resultados finales a partir de 50-100 eventos de entrenamiento para tres variables independientes.

4.4. Recursos computacionales

Los procedimientos propuestos en la metodología expuesta en este trabajo son por lo general exigentes computacionalmente. Se detalla a continuación el tiempo de cálculo requerido para cada paso de la aplicación de la metodología.

- Se ha realizado una simulación (transitoria) de reanálisis de un año en WRF empleando 6 procesadores que lleva en total 15 días y medio cuyos resultados ocupan en disco $\sim$ 175 Gb. Cada hora volcada de esta simulación toma aproximadamente $\sim$ 2,5 min y ocupa $\sim$ 20 Mb. [2]

- Respecto a la CFD, se han ejecutado un total de 289 (209 eventos de entrenamiento y 80 para test). Para la malla de 250000 celdas que se ha empleado, realizándola en 6 núcleos, una única simulación toma $\sim$ 40 min por lo que el proceso total ha requerido unos 8 días. Almacenar el campo de velocidades de cada simulación ocupa en disco 10 Mb, casi 3 Gb en total. [2]

- La selección de eventos significativos se ha completado en $\sim$ 40 min con picos en el uso de RAM de $\sim$ 40 Gb (notar que analizar los 8785 resultados producidos por la simulación WRF supone manejar una gran cantidad de datos). [2]

- El procesamiento de todos los resultados procedentes de las simulaciones WRF y CFD, la construcción de malla puente y el acondicionamiento de los datos para su posterior uso por los modelos de regresión ha costado $\sim$ 5 h. [3]

- Finalmente, ajustar (con los 209 eventos de entrenamiento) y obtener los resultados con cada uno de los regresores requiere de tiempos de cómputo diferentes. Para RF se necesitan 22 min, para KNn unos 4 min, en el caso de SVM y SGD alrededor de 2 min y para el más rápido de todos, las interpolaciones mediante triangulaciones de Delaunay, 1 min 40 s. Es lógico que RF sea el que más tiempo requiere, ya que según la configuración escogida se entrenan 500 árboles de decisión para generar el bosque. [3]

Las ventajas computacionales de aplicar el método propuesto para la construcción del ROM se hacen patentes con estos resultados. A partir de una simulación de mesoescala, bien de predicción o reanálisis (con un coste computacional de unos 2,5 min por hora simulada), se pueden obtener de forma prácticamente instantánea mediante el ROM, resultados equivalentes a los del modelo CFD (cuyo coste computacional está entorno a los 40 minutos por evento). Si además se tiene en consideración que la malla CFD típica para la explotación de este modelo (y consecuentemente el coste computacional de resolver el flujo en ella) es del orden de 100 veces superior, la conveniencia de esta herramienta es patente.

[2] Se ha empleado una máquina con 64 Gb de RAM y un procesador Intel(R) Core(TM) i7-5820K a 3,30 GHz con 6 núcleos y 15360 Kb de memoria caché.

[3] Se ha empleado una máquina con 32 Gb de RAM y un procesador Intel(R) Core(TM) i7-2600K a 3,40 GHz con 4 núcleos y 8192 Kb de memoria caché.

Capítulo 5

Conclusiones

En este proyecto se ha desarrollado e implementado una metodología para construir un modelo de orden reducido a partir de simulaciones de mesoescala y CFD. Este modelo devuelve resultados detallados del campo de velocidades a partir de simulaciones de mesoescala, empleando diferentes tipos de modelos de regresión. Esta metodología consiste en realizar simulaciones WRF de un periodo temporal amplio identificando los eventos más significativos que se dan en la localización de estudio. Estos eventos se emplean como condiciones de contorno para realizar simulaciones CFD con las que entrenar el ROM.

Se ha aplicado la metodología propuesta sobre una región con orografía compleja del Pirineo aragonés. Los resultados obtenidos son aceptables y se logra capturar la tendencia general de los datos de entrenamiento. Los valores de R^2 obtenidos para los diferentes regresores son muy cercanos a la unidad para las componentes U_x y U_y, y algo inferiores para U_z. El percentil 85 % de los errores absolutos de las predicciones obtenidas es de ~ 2 [m/s] para las componentes horizontales del viento y de $\sim 0{,}8$ [m/s] para la componente vertical. Un análisis de los resultados por alturas ha permitido identificar que la principal fuente de error del modelo radica en los puntos con altitudes más bajas. Se ha realizado un análisis de sensibilidad para estudiar la influencia del número de eventos del entrenamiento en los resultados y las conclusiones son que el número de eventos con los que se entrena el ROM no es significativo a partir de $50 - 100$ eventos.

En referencia a los recursos computacionales, se requieren unos 2,5 min para obtener resultados WRF correspondientes a un evento, mientras que son necesarios ~ 40 min (la malla computacional empleada es realmente grosera y en una aplicación comercial este tiempo se incrementaría notablemente) para realizar una simulación directa con CFD. El uso del ROM una vez entrenado es prácticamente instantáneo, por lo que las ventajas computacionales de aplicar el método propuesto para la construcción del ROM son fehacientes.

No obstante, el ROM obtenido de la aplicación de la metodología propuesta tiene un gran margen de perfeccionamiento. Una de las posibles mejoras de este método pasan por ampliar el número de variables independientes para los regresores que conforman el ROM y no restringirse al campo de velocidades generado por WRF. Además, explorar el espacio de hiperparámetros de los diferentes modelos de regresión empleados de manera más exhaustiva y probar regresores diferentes (como redes neuronales) puede resultar ventajoso en los resultados finales obtenidos. Asimismo, identificar de manera más concreta las fuentes de error y *outliers* (detección de eventos test anómalos o localizaciones espaciales complejas) ayudaría a la hora de adoptar estrategias concretas de mejora. Como trabajo futuro resultaría interesante evaluar esta metodología en un entorno urbano. Por otro lado, las *Physics Informed Neural Networks* (PINNs) [60, 61, 62] son un tipo de redes neuronales que son entrenadas para resolver tareas de entrenamiento supervisado respetando las leyes físicas subyacentes a ecuaciones diferenciales parciales no lineales como es el caso de las ecuaciones de Navier-Stokes (2.3). La aplicación de las PINNs para el problema que se plantea resolver en este proyecto resulta realmente atractiva para ser explorada en el futuro.

Apéndice A

Estudio paramétrico de WRF

Dentro de los los elementos principales de la metodología desarrollada en este proyecto, el software que se emplea para llevar a cabo las simulaciones de mesoescala es WRF (sección 2.1). En el presente capitulo se explora la influencia de diferentes esquemas de resolución de las físicas de este software así como la influencia de la resolución espacial de su malla computacional y los datos de entrada de modelos globales para una simulación de reanálisis.

Las métricas que se emplean para evaluar los resultados obtenidos son las siguientes:

$$\begin{aligned} STD &= \left[\frac{\sum_i (m_i - o_i)^2}{N}\right]^{1/2} \\ STD\% &= \frac{N \cdot \left[\sum_i (m_i - o_i)^2\right]^{1/2}}{\sum_i o_i} \cdot 100 \\ BIAS &= \frac{\sum_i (m_i - o_i)}{N} \\ BIAS\% &= \frac{\sum_i (m_i - o_i)}{\sum_i o_i} \cdot 100 \end{aligned} \tag{A.1}$$

Donde STD es la desviación estándar, $BIAS$ el sesgo, $STD\%$ la desviación estándar y $BIAS\%$ el sesgo relativo. N el número de datos a comparar, o_i y m_i los valores observados (reales) y los modelados respectivamente.

Las simulaciones corresponden a un mes, más concretamente del 2 al 31 de mayo del 2016. La región sobre la que se realizan se encuentra centrada en Zaragoza debido a que los valores reales con los que se compara se obtienen de su aeropuerto[1]. Se emplean cuatro dominios cuadrados anidados concéntricamente, con resoluciones espaciales de 27, 9, 3 y 1 km respectivamente. La relación de tamaño entre ellos es de 3 : 1 y el lado el dominio más pequeño es de 51 km. Se presenta en la figura A.1 una imagen de estos. Los productos de las simulaciones WRF que se evalúan son: la temperatura y humedad relativa a 2 m sobre el terreno, la velocidad del viento y su dirección a 10 m sobre el terreno y la presión sobre el nivel del mar. Estos valores son interpolados a la localización concreta del aeropuerto (41,666° N - 1,042° W).

WRF implementa gran variedad de físicas para resolver diferentes fenómenos atmosféricos. En esta ocasión analizaremos la influencia que tienen cinco de ellas y las diferentes aproximaciones que se usan para resolverlas. Las diferentes selecciones de esquemas para resolver las físicas son similares a los que se encuentran en [63] y se pueden consultar a continuación:

- **Capa límite planetaria** (PBL): La capa límite planetaria es la parte más baja de la atmósfera y su comportamiento se encuentra influenciado de manera directa por su contacto con la superficie. Las diferentes implementaciones exploradas para su resolución son: 1 - Yonsei University Scheme [34]; 5 - Mellor–Yamada Nakanishi Niino Level 2.5 [48, 49].

[1] Identificación del aeropuerto: LEZG, ZARAGOZA AB, USAF:081600

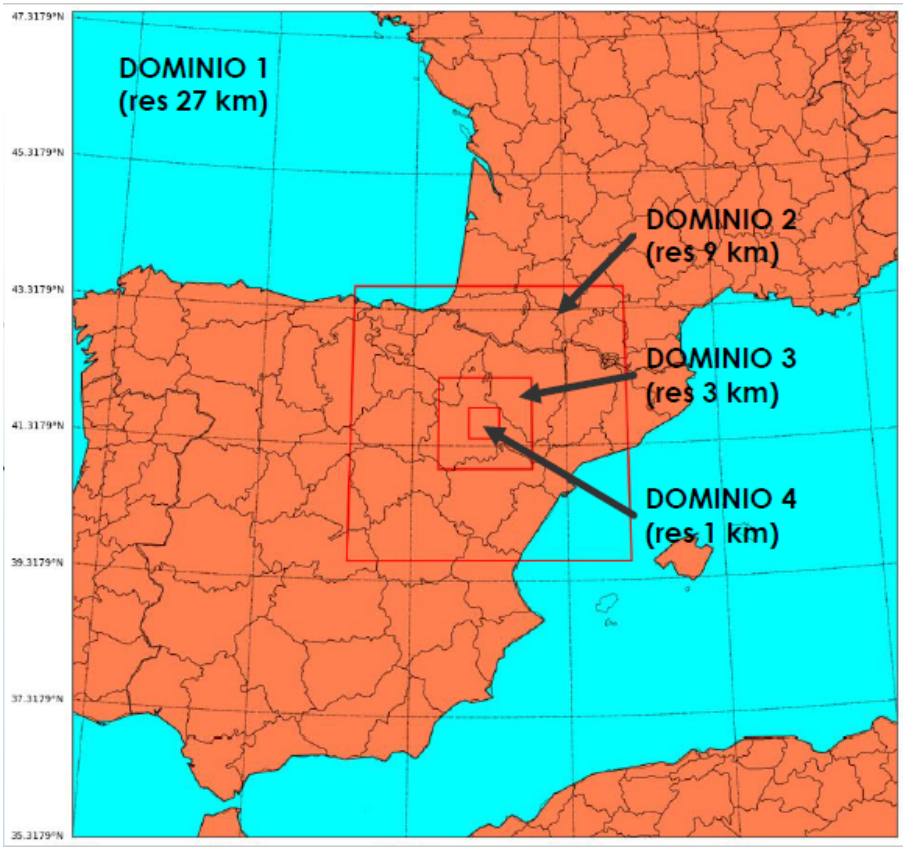


Figura A.1: Dominios de las simulaciones WRF realizadas para análisis paramétrico.

- **Micro físicas** (MPH): La microfísica de las nubes es la rama de la ciencia atmosférica que se ocupa de estudiar las diferentes partículas que conforman una nube. Las diferentes implementaciones exploradas para su resolución son: 3 - WRF Single–moment 3–class [33]; 8 - Thompson [81]; 10 - Morrison 2–moment [47].
- **Parametrización de cúmulos** (CMS): resuelve las dinámicas relacionadas con las nubes cúmulus. Las diferentes implementaciones exploradas para su resolución son: 1 - Kain–Fritsch [38]; 5 - Grell 3D Ensemble [27, 28].
- **Radiación de onda corta** (SWR): modela los efectos causados por la energía de la radiación solar en la atmósfera. Las diferentes implementaciones exploradas para su resolución son: 1 - Dudhia [21]; 4 - RRTMG [35]; 24 - Faster RRTMG.
- **radiación de onda larga** (LWR): modela los efectos causados por la energía de la reemisión de radiación terrestre en la atmósfera. Las diferentes implementaciones exploradas para su resolución son: 1 - RRTM [45]; 24 - Faster RRTMG.

Para realizar las simulaciones, WRF requiere datos de entrada proporcionados por un modelo global. En este análisis paramétrico se estudian los resultados obtenidos con dos conjuntos de datos de entrada diferentes: el ds090.0 [50] y el ds627.0 [24]. El tamaño en disco de ambos es significativamente diferente; los datos necesarios para ejecutar la simulación de un mes del conjunto ds090.0 ocupan $\sim$ 400 Mb mientras que los de ds627.0 tienen un tamaño de $\sim$ 16 Gb. En la tabla A.1 se recogen las diferentes selecciones de físicas que se han hecho, los datos de entrada empleados, el tiempo que a tomado realizar la simulación para cada una de ellas y el tamaño que ocupan en disco.

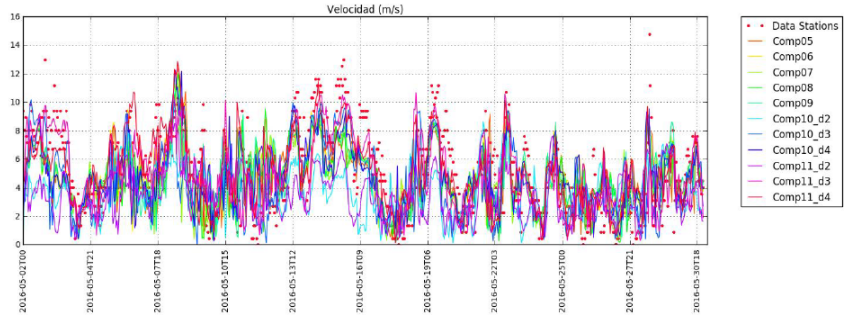
[2]Se ha empleado una máquina con 64 Gb de RAM y un procesador Intel(R) Core(TM) i7-6800K a 3,40 GHz con 6 núcleos y 15360 Kb de memoria caché.

Nombre	**PBL**	**MPH**	**CMS**	**SWR**	**LWR**	**Tiempo [h][2]**	**Tamaño [Gb]**	**Datos entrada**
*Comp*05	1	3	1	1	1	~ 11	$\sim 4{,}1$	ds090.0
*Comp*06	1	8	5	4	1	~ 20	$\sim 5{,}5$	ds090.0
*Comp*07	5	10	5	24	24	~ 22	~ 7	ds090.0
*Comp*08	5	3	1	1	1	~ 13	$\sim 4{,}9$	ds090.0
*Comp*09	5	3	5	1	1	~ 12	$\sim 4{,}9$	ds090.0
*Comp*10	1	8	5	24	1	~ 20	$\sim 5{,}5$	ds090.0
*Comp*11	1	8	5	24	1	~ 18	$\sim 5{,}5$	ds627.0

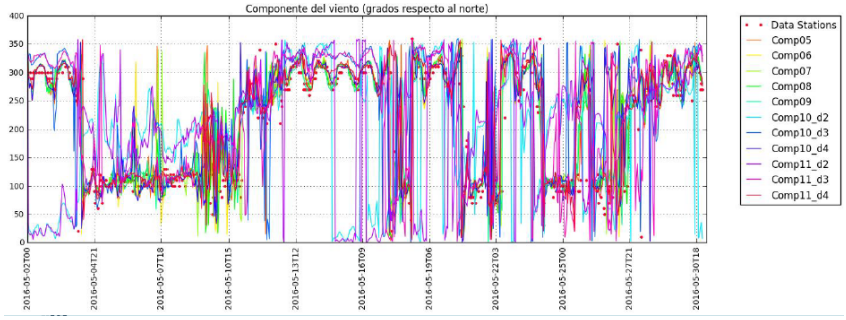
Cuadro A.1: Conjuntos de físicas sobre las que se realiza el análisis, tiempos de cómputo, tamaño de los resultados y datos de entrada empleados para las simulaciones.

Por lo general, los resultados que se comparan se toman del dominio más pequeño con un 1 km de resolución (se denota con el sufijo "_d4"). Sin embargo, la configuración de físicas de *Comp*10 es la que resulta más óptima en el trabajo [63] y por ello se evalúa con esta la influencia de la resolución de la malla computacional usada. Para ello se comparan también los resultados de los dominios con resoluciones de 3 y 9 km (marcadas por los sufijos "_d3" y "_d2" respectivamente). Adicionalmente se repite el mismo análisis para la resolución del dominio usado pero empleando los datos de entrada del conjunto ds627.0. En la figura A.2 se presentan los resultados obtenidos por las simulaciones de los diferentes productos seleccionados junto con los valores reales (puntos rojos, "Data Stations" en las gráficas). Por otro lado, en la figura A.3 se muestran los valores de las métricas definidas en (A.1).

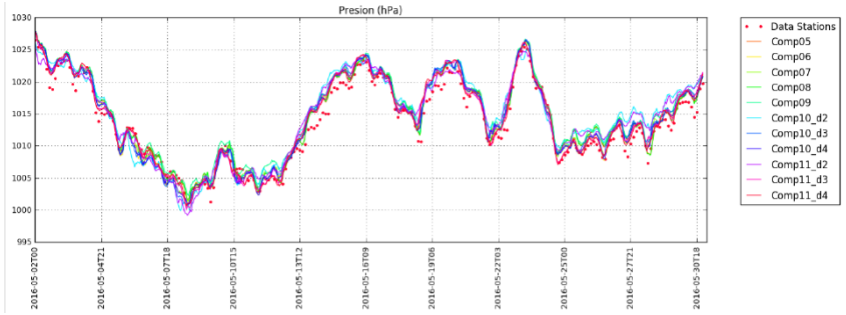
Se aprecia que, por lo general, las simulaciones $comp10_d4$ y $comp11_d4$ obtienen los mejores resultados para los diferentes productos comparados. Por otro lado, resoluciones menores de la malla computacional tienen un impacto positivo en los resultados finales. Sin embargo los diferentes datos de entrada no tienen influencia significativa. A partir de los resultados del análisis realizado en este capítulo, las físicas correspondientes a *Comp*10, los datos de entrada correspondientes a ds090.0 y la resolución espacial de 0,9 km son usados para las simulaciones WRF llevadas a cabo en la aplicación de la metodología propuesta en este proyecto.



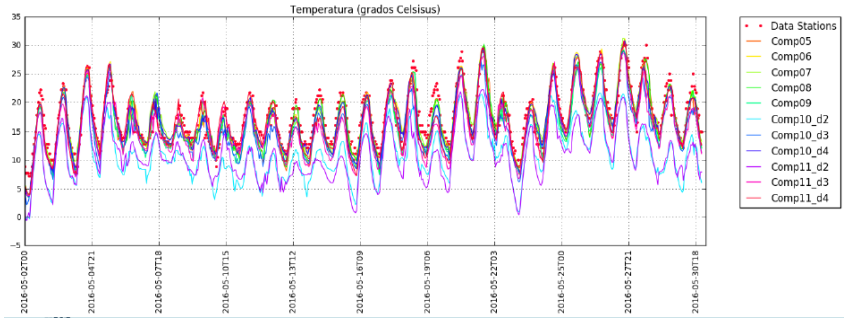
(a) Velocidad del viento



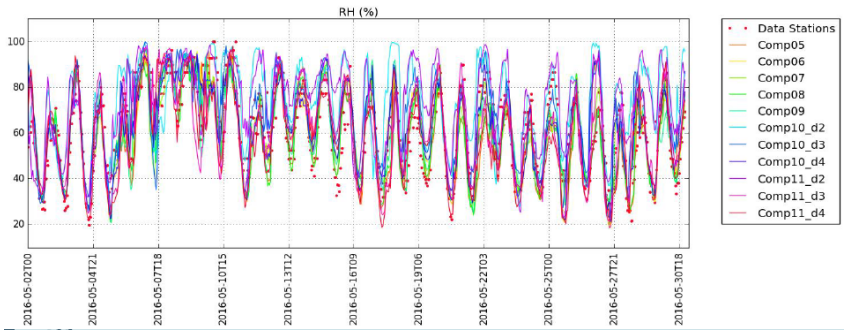
(b) Dirección del viento



(c) Presión sobre el nivel del mar

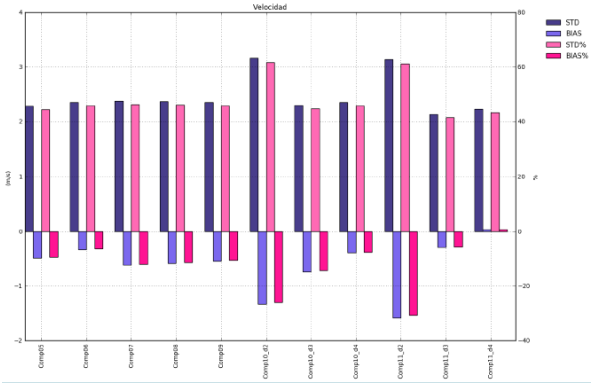


(d) Temperatura

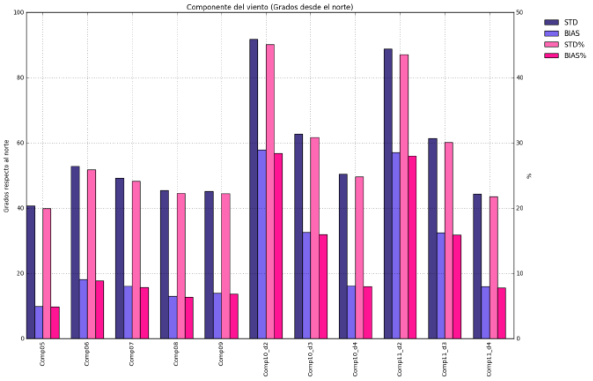


(e) Humedad relativa

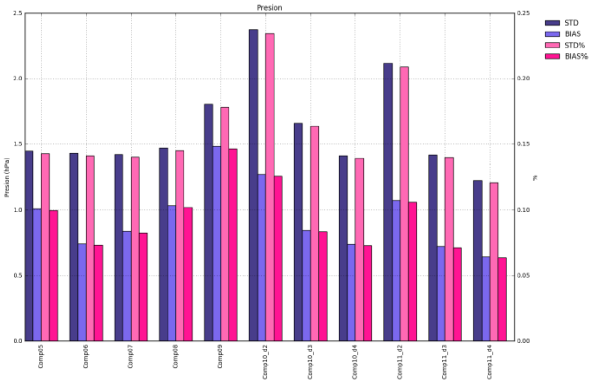
Figura A.2: Resultados obtenidos por las simulaciones junto con los valores reales.



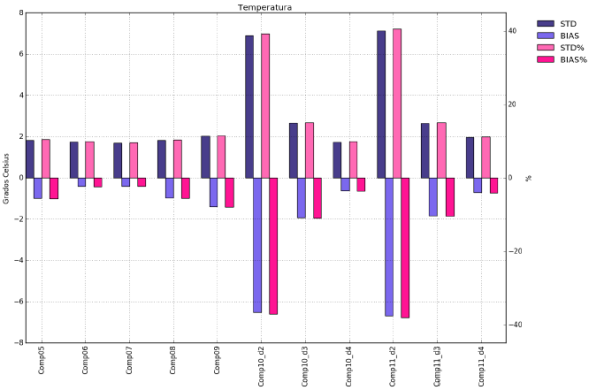
(a) Velocidad del viento



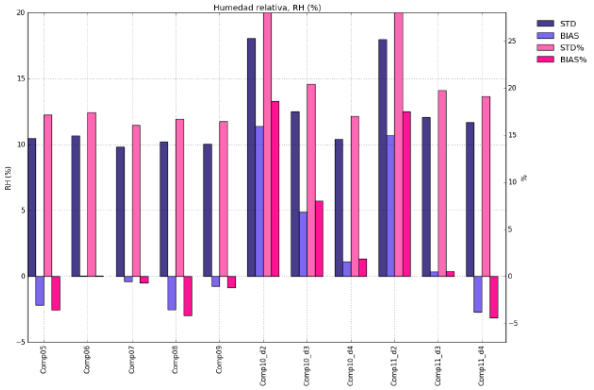
(b) Dirección del viento



(c) Presión sobre el nivel del mar



(d) Temperatura



(e) Humedad relativa

Figura A.3: Valores de STD , $BIAS$, $STD\%$ y $BIAS\%$ para las simulaciones realizadas con las diferentes configuraciones.

Apéndice B

Códigos

En este apéndice se adjuntan los diferentes códigos implementados en Python que se emplean para el desarrollo de este proyecto.

B.1. Código principal

Se presenta a continuación el código principal desarrollado en Python para la construicción del ROM y las gráficas de resultados.

```
#!/usr/bin/env python
# coding: utf-8

from scipy.interpolate import LinearNDInterpolator
# Funciones y definiciones auxiliares
from ROM_class import *

import resource
soft, hard = resource.getrlimit(resource.RLIMIT_NOFILE)
resource.setrlimit(resource.RLIMIT_NOFILE, (hard, hard))

tiki = ROM(pathOpen='/home/jmilla/000home_nodos/openfoam6/Casos/casoFinal/',
           pathWRF='/home/jmilla/000home_nodos/wrf/Domains/000Agno2018/')
path_rsm = ('/home/jmilla/000home_nodos/Script_AnalizaEstadistica/'
            +'Resultados/v2_20171231_20190101/')

tiki.fechas_datos_sim = tiki.lee_Resumen(path_rsm)

# ============================================================================
tiki.fechas_datos_test_indx = [3573, 3837, 2802,
                               1395,  663, 5944, 5205,
                               7259, 1033, 2012, 3685,
       8700, 3782, 2499, 6634, 7815,  193, 6231, 4849, 4595, 7508, 7214,
       6900,  381, 1174, 1248, 8262, 6359, 3958, 5868, 6296, 5437, 4219,
       4195, 5100, 7262, 5284, 6166, 3783, 6518, 8459, 3835, 2682, 3369,
       3790, 2232,  305, 5579, 3084,  455, 3779, 2924, 7426, 7485, 6547,
       7338, 3713, 5922, 7656, 3583,   41, 8400, 3753, 6873, 6247,   27,
       3236, 2144, 3478, 5607, 2637, 8181, 3351, 4428, 5937, 3905, 3049,
       3114, 8396, 2857
                            ]

tiki.fechas_datos_test = (np.array(tiki.dates).astype(str)
                          [tiki.fechas_datos_test_indx].tolist())

tiki.fechas_datos_sim = np.append(tiki.fechas_datos_sim,
                                  tiki.fechas_datos_test).tolist()
tiki.fechas_datos_sim_indx = (np.append(tiki.fechas_datos_sim_indx,
                                        tiki.fechas_datos_test_indx)
                                            .astype(int).tolist())
# ============================================================================

# ============================================================================
# los casos boundary!!!!
tiki.fechas_datos_boun_indx = (np.array([
    5735, 1571,
                                         3803, 8726, 3825, 4500, 3871, 3536,
                                         2849, 6135, 3757, 7464, 5821,  363,
                                         7923, 5654
                                        ])+9999*1e4).astype(int).tolist()
tiki.fechas_datos_boun = []
for kk in tiki.fechas_datos_boun_indx:
    tiki.fechas_datos_boun.append('boundary_'+np.array(tiki.dates).astype(str)
                                  [int(kk-9999*1e4)])

tiki.fechas_datos_sim = np.append(tiki.fechas_datos_sim,
                                  tiki.fechas_datos_boun).tolist()
tiki.fechas_datos_sim_indx = (np.append(tiki.fechas_datos_sim_indx,
                                        tiki.fechas_datos_boun_indx)
                                            .astype(int).tolist())
```

```
# ======================================================================

tiki.rutas_datos_sim = tiki.genera_rutaSim()

tiki.elimina_fechas()

df_var = tiki.genera_df_var(lat=tiki.CentroLatSTL, lon=tiki.CentroLonSTL)
full_data_backup = tiki.dame_full_data(df_var=df_var, radio=1)

malla_puente_tot = tiki.crea_malla_puente(n_pts=25, n_niveles_eta=3)
maskCFDin = tiki.dame_maskCFDin(malla_puente_tot)
malla_puente = tiki.actualiza_malla_puente(malla_puente_tot[maskCFDin],
                                           offset=10)

def pinta_mp_vert(tiki):
    latitudicas = tiki.malla_puente.Lat.unique()
    longitudicas = tiki.malla_puente.Lon.unique()
    n_pts = len(latitudicas)
    plt.figure(figsize=(10,10))
    auxiliarico = tiki.malla_puente.loc[tiki.malla_puente.Lat
                                        == latitudicas[n_pts//2]]
    for etiki in auxiliarico.Eta.unique():
        plt.plot(auxiliarico.loc[auxiliarico.Eta==etiki].Lon,
                 auxiliarico.loc[auxiliarico.Eta==etiki].Altura,
                 '.', label='$\eta$=%3.2f'%etiki);
    plt.legend()
    plt.xlim(auxiliarico.Lon.min(), auxiliarico.Lon.max())
#     plt.ylim(auxiliarico.Altura.min(), auxiliarico.Altura.max()*1.05)
    plt.grid()
    plt.xlabel('Longitud', fontsize=20)
    plt.ylabel('Altura (m)', fontsize=20)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.legend(fontsize=20)
pinta_mp_vert(tiki)

tiki.dame_full_data_mp(df_var,
                       ['Ux', 'Uy', 'Uz', 'u', 'v', 'wa'],
                       parallel=False)

(tiki.full_data_mp.loc[:, 'uvmet_interp_29:50_u'] =
 tiki.full_data_mp.loc[:, 'uvmet_interp_29:50_u'].astype(float))
(tiki.full_data_mp.loc[:, 'uvmet_interp_29:50_v'] =
 tiki.full_data_mp.loc[:, 'uvmet_interp_29:50_v'].astype(float))

def separa_train_test_mp(aux_train, fechas_test, fechas_train_drop):

    aux_fecha = np.array([],dtype=int)
    for kk in fechas_test:
        aux_fecha = np.append(aux_fecha, aux_train.loc[aux_train.Fecha==kk]
                              .index.to_numpy())
    assert len(aux_fecha)>1
    aux_test = aux_train.loc[aux_fecha]

    aux_fecha = np.array([],dtype=int)
    for kk in fechas_train_drop:
        aux_fecha = np.append(aux_fecha, aux_train.loc[aux_train.Fecha==kk]
                              .index.to_numpy())
    aux_train = aux_train.drop(index=aux_fecha)

    return aux_train, aux_test

from scipy.spatial import Delaunay

def pinta_tri(aux1, aux2, tri, aux_train, aux_test, features):
    fig = plt.figure(figsize = (7,5))
    plt.triplot(tri.points[:,aux1], tri.points[:,aux2], triangles=tri.simplices[:,[aux1,aux2]])

    plt.plot(aux_train.loc[:,features].to_numpy()[:,aux1],
             aux_train.loc[:,features].to_numpy()[:,aux2], '.')
    plt.plot(aux_test.loc[:,features].to_numpy()[:,aux1],
             aux_test.loc[:,features].to_numpy()[:,aux2], '.')

    plt.xlabel(features[aux1])
    plt.ylabel(features[aux2])

def saca_prediccion_mp(tri, aux_train, aux_test, features, labels, tol=1e-10):
    fn = LinearNDInterpolator(tri, aux_train.loc[:, labels], fill_value=np.nan, tol=tol)
    predictions = fn(aux_test.loc[:,features])

    mask_nonan = ~np.isnan(predictions[:,0])

    return predictions, mask_nonan

def evalua_prediccion_mp(aux_test, predictions, labels, mask_nonan, modvel=True):
    from sklearn.metrics import r2_score
    copy = aux_test.loc[:,labels].copy()



    error = np.abs(aux_test.loc[mask_nonan,labels]
                   -predictions[mask_nonan])

    if modvel:
        vel = np.sqrt(np.power(aux_test.loc[:,labels[0]].to_numpy(),2)+
              np.power(aux_test.loc[:,labels[1]].to_numpy(),2))

        direc = np.rad2deg(np.arctan2(aux_test.loc[:,labels[0]].to_numpy(),
                                      aux_test.loc[:,labels[1]].to_numpy()))
```

```
        direc[direc<0] = direc[direc<0] + 360
        veldir = pd.DataFrame(np.vstack((vel, direc, aux_test
                                         .loc[:,labels[2:]].to_numpy()
                                         .transpose())).transpose(),
                              columns=list(np.append(['Vel', 'Dir'],
                                                     labels[2:])))


        vel = np.sqrt(np.power(predictions[:,0],2)+
                      np.power(predictions[:,1],2))

        direc = np.rad2deg(np.arctan2(predictions[:,0],
                                      predictions[:,1]))
        direc[direc<0] = direc[direc<0] + 360
        veldir_p = pd.DataFrame(np.vstack((vel, direc, predictions[:,2:]
                                           .transpose()))).transpose(),
                              columns=list(np.append(['Vel', 'Dir'],
                                                     labels[2:])))

        error = np.abs(veldir.loc[mask_nonan]
                       -veldir_p.loc[mask_nonan])

        error.loc[error.Dir>180, 'Dir'] = 360-error.loc[error.Dir>180,
                                                        'Dir']
    if modvel:
        aaa = np.array([np.mean(error), np.var(error), np.max(error),
                        r2_score(veldir.loc[mask_nonan,:],
                                 veldir_p.loc[mask_nonan,:],
                                 multioutput='raw_values')])
        print('Mean error:', np.mean(error),
          '\nVar error:', np.var(error),
          '\nMax error:', np.max(error),
          '\nR2 score:', r2_score(veldir.loc[mask_nonan,:],
                                   veldir_p.loc[mask_nonan,:],
                                   multioutput='raw_values'),
          sep='\n')
    else:
        aaa = np.array([np.mean(error), np.var(error), np.max(error),
                        r2_score(aux_test.loc[mask_nonan,labels],
                                       predictions[mask_nonan],
                                       multioutput='raw_values')])
        print('Mean error:', np.mean(error),
              '\nVar error:', np.var(error),
              '\nMax error:', np.max(error),
              '\nR2 score:', r2_score(aux_test.loc[mask_nonan,labels],
                                       predictions[mask_nonan],
                                       multioutput='raw_values'),
              sep='\n')

    fig = plt.figure(figsize = (10,10))
    if modvel:
        for kk in veldir.columns:
            plt.scatter(veldir_p.loc[:,kk],
                        veldir.loc[:,kk],
                        marker='.', label=kk, alpha=0.5)

        plt.xlim([-50,360])
        plt.ylim([-50,360])

    else:
        kkk=0
        for kk in labels:
            plt.scatter(predictions[:,kkk],
                        aux_test.loc[:,[kk]],
                        marker='.', label=kk, alpha=0.5)
            kkk+=1

        plt.xlim([-40,40])
        plt.ylim([-40,40])

    plt.grid()
    plt.plot(np.arange(-100,360),np.arange(-100,360), c='orange')
    plt.legend(fontsize=20)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.legend(fontsize=20)
    plt.xlabel('Predicción [m/s]', fontsize=20)
    plt.ylabel('Reales [m/s]', fontsize=20)


    error['Lat'] = aux_test.loc[mask_nonan,'Lat']
    error['Lon'] = aux_test.loc[mask_nonan,'Lon']
    error['Eta'] = aux_test.loc[mask_nonan,'Eta']
    error['Fecha'] = aux_test.loc[mask_nonan,'Fecha']

    return aaa,error

n_events_tri = 193
features = ['u', 'v','wa']
labels = ['Ux', 'Uy', 'Uz']
# features = ['uvmet_interp_29:50_u',
              'uvmet_interp_29:50_v',
             'wa_interp_29:50']

np.random.seed(1234)
aux_indx = np.random.choice(np.arange(len(tiki.fechas_datos_test)),
                            int(len(tiki.fechas_datos_test)*1),
                            replace=False)
fechas_test = np.array(tiki.fechas_datos_test)[aux_indx]
```

```
258
259 aaa = np.setdiff1d(np.setdiff1d(tiki.fechas_datos_sim,
260                                  tiki.fechas_datos_test),
261                     tiki.fechas_datos_boun)
262 nnn = len(aaa)-n_events_tri
263 aux_indx = np.random.choice(np.arange(len(aaa)),
264                             nnn,
265                             replace=False)
266 fechas_train_drop = np.append(tiki.fechas_datos_test,
267                               aaa[aux_indx])
268
269 lat = tiki.full_data_mp.Lat.unique()[0]
270 lon = tiki.full_data_mp.Lon.unique()[0]
271 eta = tiki.full_data_mp.Eta.unique()[2]
272 dt_in = tiki.full_data_mp.loc[(tiki.full_data_mp.Lat==lat) &
273                               (tiki.full_data_mp.Lon==lon) &
274                               (tiki.full_data_mp.Eta==eta)]
275 dt_in = tiki.full_data_mp
276
277 print('Test:  ', len(fechas_test))
278 print('Train: ', len(np.setdiff1d(tiki.fechas_datos_sim, fechas_train_drop)))
279
280 from sklearn.experimental import enable_hist_gradient_boosting  # noqa
281 from sklearn.ensemble import HistGradientBoostingRegressor
282
283 from sklearn.linear_model import SGDRegressor
284 from sklearn.svm import SVR
285 from sklearn.ensemble import RandomForestRegressor
286 from sklearn.ensemble import AdaBoostRegressor
287 from sklearn.ensemble import GradientBoostingRegressor
288 from sklearn.neighbors import KNeighborsRegressor
289 from sklearn.neural_network import MLPRegressor
290
291 from sklearn.model_selection import train_test_split
292 from sklearn.pipeline import make_pipeline
293 from sklearn.decomposition import PCA
294 from sklearn.preprocessing import StandardScaler
295 from sklearn.metrics import r2_score
296
297 from sklearn.dummy import DummyRegressor
298 from sklearn.metrics import mean_absolute_error
299
300 dmy = DummyRegressor('median')
301 scaler = StandardScaler()
302
303 rfr = RandomForestRegressor(n_estimators=500,
304                             criterion="mse",
305                             max_depth=50,
306                              max_features="auto",
307                              bootstrap=True,
308                              oob_score=False,
309                              n_jobs=-1,
310                              random_state=0,
311                              verbose=0)
312
313 sgd = SGDRegressor(loss='huber',
314                    penalty='L2',
315                    alpha=1e-3,
316                    fit_intercept=True,
317                    max_iter=1000,
318                    tol=1e-3,
319                    shuffle=True,
320                    epsilon=0.5,
321                    random_state=0,
322                    learning_rate='invscaling',
323                    eta0=0.1,
324                    power_t=0.1,
325                    early_stopping=False,
326                    validation_fraction=0.1,
327                    n_iter_no_change=5,
328                    warm_start=False,
329                    average=False)
330
331 svm = SVR(kernel='rbf',
332           gamma='auto',
333           tol=0.1,
334           C=1,
335           epsilon=0.5,
336           shrinking=True,
337           max_iter=-1,
338           cache_size=2000)
339
340 KNn = KNeighborsRegressor(n_neighbors=5,
341                           weights='distance',
342                           algorithm='auto',
343                           p=2,
344                           leaf_size=30,
345                           n_jobs=-1)
346
347 from sklearn.multioutput import MultiOutputRegressor
348
349 regr = make_pipeline(scaler, MultiOutputRegressor(sgd, n_jobs=-1))
350 # regr = make_pipeline(scaler, rfr)
351
352 np.random.seed(1234)
353 aaa = 0
354 stats_cum = 0
355 error_cum = 0
356 del aaa
```

```
del stats_cum
del error_cum
for kk in np.arange(1):
    aux_indx = np.random.choice(np.arange(len(tiki.fechas_datos_test)),
                                int(len(tiki.fechas_datos_test)*1),
                                replace=False)
    fechas_test = np.array(tiki.fechas_datos_test)[aux_indx]

    acm_pred = []
    acm_maskNoNan = np.array([], dtype=np.bool)
    acm_test = pd.DataFrame([])

    for lat in tiki.full_data_mp.Lat.unique():
        for lon in tiki.full_data_mp.Lon.unique():
            for eta in tiki.full_data_mp.Eta.unique():
                dt_in = tiki.full_data_mp.loc[(tiki.full_data_mp.Lat==lat) &
                                              (tiki.full_data_mp.Lon==lon) &
                                              (tiki.full_data_mp.Eta==eta)]
                if not len(dt_in):
                    break
                aux_train, aux_test = separa_train_test_mp(dt_in, fechas_test,
                                                           fechas_train_drop)

#                 tri = Delaunay(aux_train.loc[:,features] ,
#                                qhull_options="Qbb Qc Qz Qx Q12 Qs")
#                 predictions, mask_nonan = saca_prediccion_mp(tri,
#                                                              aux_train,
#                                                              aux_test,
#                                                              features,
#                                                              labels,
#                                                             )
                regr.fit(aux_train.loc[:,features], aux_train.loc[:,labels])
                predictions = regr.predict(aux_test.loc[:,features])
                mask_nonan = ~np.isnan(predictions[:,0])

                try:
                    acm_pred = np.vstack((acm_pred, predictions))
                except ValueError:
                    acm_pred = predictions

                acm_maskNoNan = np.append(acm_maskNoNan, mask_nonan)
                acm_test = acm_test.append(aux_test)

    stats, error = evalua_prediccion_mp(acm_test, acm_pred,
                                        labels, acm_maskNoNan, modvel=False)

    error_aux = np.full((len(acm_maskNoNan),3),
                        np.nan)
    error_aux[acm_maskNoNan] = error.iloc[:,0:3]
    try:
        error_cum = np.concatenate((error_cum,
                                    [error_aux]), axis=0)
        stats_cum = np.concatenate((stats_cum,
                                    [stats]), axis=0)
    except NameError:
        error_cum = np.array([error_aux])
        stats_cum = np.array([stats])')

N = 5
ind = np.arange(N)  # the x locations for the groups
width = 0.27       # the width of the bars

fig = plt.figure()
ax = fig.add_subplot(111)

yvals = [0.8306, 0.9285, 0.9355, 0.9373, 0.9368]
rects1 = ax.bar(ind, yvals, width)
zvals = [0.8326, 0.9439, 0.9483, 0.9491, 0.9492]
rects2 = ax.bar(ind+width, zvals, width)
kvals = [-0.2514, 0.3452, 0.4054, 0.4131, 0.4034]
rects3 = ax.bar(ind+width*2, kvals, width)

ax.set_ylabel('$R^2$', fontsize=20)
ax.set_xticks(ind+width*1.5)
ax.set_xticklabels(('16', '50', '100', '150', '209'),
                   fontsize=20, rotation=30)
ax.legend((rects1[0], rects2[0], rects3[0]), ('$U_x$', '$U_y$', '$U_z$'),
          fontsize=20, loc=2, frameon=1, bbox_to_anchor=(1.1, 1))

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
ax.grid(axis='y')
plt.xlabel('Número de eventos de entrenamiento', fontsize=20)
plt.title('SGD', fontsize=20);

N = 3
ind = np.arange(N)  # the x locations for the groups
width = 0.27       # the width of the bars

fig = plt.figure()
ax = fig.add_subplot(111)

yvals = [0.5260, 0.9462, 0.9859]
rects1 = ax.bar(ind, yvals, width)
zvals = [0.6545, 0.9663, 0.9884]
rects2 = ax.bar(ind+width, zvals, width)
kvals = [0.4771, 0.3681, 0.0424]
rects3 = ax.bar(ind+width*2, kvals, width)

```

```
ax.set_ylabel('$R^2$', fontsize=20)
ax.set_xticks(ind+width)
ax.set_xticklabels(('$\eta=0$', '$\eta=0.5$', '$\eta=1$'),
                   fontsize=20, rotation=30)
ax.legend((rects1[0], rects2[0], rects3[0]), ('$U_x$', '$U_y$', '$U_z$'),
          fontsize=20, loc=2, frameon=1, bbox_to_anchor=(1.1, 1))

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
ax.grid(axis='y')
plt.title('SGD', fontsize=20);

N = 5
ind = np.arange(N)  # the x locations for the groups
width = 0.27       # the width of the bars

fig = plt.figure()
ax = fig.add_subplot(111)

yvals = [0.9189, 0.8975, 0.8755, 0.9368, 0.9089]
rects1 = ax.bar(ind, yvals, width)
zvals = [0.9445, 0.9252, 0.8603, 0.9492, 0.9395]
rects2 = ax.bar(ind+width, zvals, width)
kvals = [0.5131, 0.4710, 0.5608, 0.4035, 0.3730]
rects3 = ax.bar(ind+width*2, kvals, width)

ax.set_ylabel('$R^2$', fontsize=20)
ax.set_xticks(ind+width)
ax.set_xticklabels(('RF', 'KNn', 'SVM', 'SGD', 'Delaunay'),
                   fontsize=20, rotation=30)
ax.legend((rects1[0], rects2[0], rects3[0]), ('$U_x$', '$U_y$', '$U_z$'),
          fontsize=20, loc=2, frameon=1, bbox_to_anchor=(1.1, 1))

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.ylim(0.35,1.05)
ax.grid(axis='y')

perc = [0.25, 0.5, 0.75, 0.85]
perc_str = ['25%', '50%', '75%', '85%']
perc_str.append('Test')
col = 'Uz'

N = 4
ind = np.arange(N)  # the x locations for the groups
width = 0.15       # the width of the bars

fig = plt.figure()
ax = fig.add_subplot(111)

yvals = [error_RF.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_RF.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_RF.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_RF.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects1 = ax.bar(ind, yvals, width)
zvals = [error_KNn.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_KNn.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_KNn.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_KNn.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects2 = ax.bar(ind+width, zvals, width)
kvals = [error_SVM.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SVM.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SVM.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SVM.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects3 = ax.bar(ind+width*2, kvals, width)
kvals = [error_SGD.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects4 = ax.bar(ind+width*3, kvals, width)
kvals = [error_DEL.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_DEL.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_DEL.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_DEL.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects5 = ax.bar(ind+width*4, kvals, width)

ax.set_ylabel('[m/s]', fontsize=20)
ax.set_xticks(ind+width*2)
ax.set_xticklabels( perc_str, fontsize=20, rotation=30)
ax.legend( (rects1[0], rects2[0], rects3[0], rects4[0], rects5[0]),
          ('Error RF', 'Error KNn', 'Error SVM', 'Error SGD', 'Error Delaunay'),
          fontsize=12)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('Percentiles', fontsize=20)
plt.title('Valores absolutos - '+col, fontsize=20)
ax.grid(axis='y')

perc = [0.25, 0.5, 0.75, 0.85]
perc_str = ['25%', '50%', '75%', '85%']
perc_str.append('Test')
col = 'Uz'

N = 4
ind = np.arange(N)  # the x locations for the groups
width = 0.15       # the width of the bars

fig = plt.figure()
```

```
ax = fig.add_subplot(111)

yvals = [error_SGD_16.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD_16.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD_16.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD_16.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects1 = ax.bar(ind, yvals, width)
yvals = [error_SGD_50.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD_50.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD_50.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD_50.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects2 = ax.bar(ind+width, yvals, width)
zvals = [error_SGD_100.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD_100.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD_100.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD_100.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects3 = ax.bar(ind+width*2, zvals, width)
kvals = [error_SGD_150.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD_150.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD_150.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD_150.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects4 = ax.bar(ind+width*3, kvals, width)
kvals = [error_SGD.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects5 = ax.bar(ind+width*4, kvals, width)

ax.set_ylabel('[m/s]', fontsize=20)
ax.set_xticks(ind+width*2)
ax.set_xticklabels( perc_str, fontsize=20, rotation=30)
ax.legend( (rects1[0], rects2[0], rects3[0], rects4[0], rects5[0]),
          ('16', '50', '100', '150', '209'),
          fontsize=12)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('Percentiles', fontsize=20)
plt.title('SGD - Valores absolutos - '+col, fontsize=20)
ax.grid(axis='y')

perc = [0.25, 0.5, 0.75, 0.85]
perc_str = ['25%', '50%', '75%', '85%']
perc_str.append('Test')
col = 'Uz'

N = 4
ind = np.arange(N)  # the x locations for the groups
width = 0.15       # the width of the bars

fig = plt.figure()
ax = fig.add_subplot(111)

yvals = [error_SGD_0.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD_0.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD_0.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD_0.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects1 = ax.bar(ind, yvals, width)
zvals = [error_SGD_05.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD_05.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD_05.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD_05.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects2 = ax.bar(ind+width, zvals, width)
kvals = [error_SGD_1.loc[:, col].abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         error_SGD_1.loc[:, col].abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         error_SGD_1.loc[:, col].abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         error_SGD_1.loc[:, col].abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects3 = ax.bar(ind+width*2, kvals, width)

ax.set_ylabel('[m/s]', fontsize=20)
ax.set_xticks(ind+width*2)
ax.set_xticklabels( perc_str, fontsize=20, rotation=30)
ax.legend( (rects1[0], rects2[0], rects3[0]),
          ('Error $\eta=0$', 'Error $\eta=0.5$', 'Error $\eta=1$'),
          fontsize=12)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('Percentiles', fontsize=20)
plt.title('SGD - Valores absolutos - '+col, fontsize=20)
ax.grid(axis='y')

perc = [0.25, 0.5, 0.75, 0.85]
perc_str = ['25%', '50%', '75%', '85%']
perc_str.append('Test')
col = 'Uz'

N = 4
ind = np.arange(N)  # the x locations for the groups
width = 0.17       # the width of the bars

fig = plt.figure()
ax = fig.add_subplot(111)

yvals = [(error_RF.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         (error_RF.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         (error_RF.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         (error_RF.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects1 = ax.bar(ind, yvals, width)
```

```
zvals = [(error_KNn.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         (error_KNn.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         (error_KNn.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         (error_KNn.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects2 = ax.bar(ind+width, zvals, width)
kvals = [(error_SVM.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         (error_SVM.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         (error_SVM.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         (error_SVM.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects3 = ax.bar(ind+width*2, kvals, width)
kvals = [(error_SGD.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         (error_SGD.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         (error_SGD.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         (error_SGD.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects4 = ax.bar(ind+width*3, kvals, width)
kvals = [(error_DEL.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[0]]).loc[perc_str[0]],
         (error_DEL.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[1]]).loc[perc_str[1]],
         (error_DEL.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[2]]).loc[perc_str[2]],
         (error_DEL.loc[:, col]/acm_test.loc[:, col]).abs().describe(percentiles=[perc[3]]).loc[perc_str[3]]]
rects5 = ax.bar(ind+width*4, kvals, width)

ax.set_ylabel('Error relativo', fontsize=20)
ax.set_xticks(ind+width*2)
ax.set_xticklabels( perc_str, fontsize=20, rotation=30)
ax.legend( (rects1[0], rects2[0], rects3[0], rects4[0], rects5[0]),
           ('RF', 'KNn', 'SVM', 'SGD', 'Delaunay'),
           fontsize=12)

plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('Percentiles', fontsize=20)
plt.title('Error relativo - '+col, fontsize=20)
plt.ylim(0,4.6)
ax.grid(axis='y')
```

B.2. Clase auxiliar

A continuación se incluye la clase implementada en Python que gestiona la comunicación de información entre los software usados (WRF y OpenFOAM).

```
#!/usr/bin/env python
# coding: utf-8

from __future__ import print_function

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import subprocess
import shapefile
import time
import os
import fnmatch
import calendar
import sys

if sys.version_info[0] == 2:
    os.environ['PROJ_LIB'] = ('/home/jmilla/anaconda2/pkgs/'
                              +'proj4-5.2.0-he1b5a44_1004/share/proj')
elif sys.version_info[0] == 3:
    os.environ['PROJ_LIB'] = '/home/jmilla/anaconda3/share/proj'
else:
    raise Exception("Version de Python: "+str(sys.version_info[0]))

from netCDF4 import Dataset
from matplotlib.path import Path
from matplotlib.patches import PathPatch
from wrf import (to_np, getvar, get_basemap, ALL_TIMES, interplevel)
from scipy.interpolate import LinearNDInterpolator
from IPython.display import display,HTML

class simulacionCFD:
    '''
    Clase para manipular:
    - Interpolacion y acople de datos WRF para la simulacion OpenFOAM
    - Recoleccion de datos de la simulacion resultante
    '''

    def __init__ (self,
                  pathWRF = '/home/jmilla/wrf/Domains/',
                  CentroLatSTL = 42.7,
                  CentroLonSTL = -0.36,
                  pathOpen = '/home/jmilla/openfoam5/Casos/casoTest/'):
        # Para tiempos de ejecucion
        TIME_INI = time.time()

        self.bound_coef = {}

        # parent path de TODOS los datos de wrf
        self.pathWRF = pathWRF
        # Centro del .stl
        self.CentroLatSTL = CentroLatSTL
```

```
        self.CentroLonSTL = CentroLonSTL
        # path del caso de openFoam en los que escribimos los datos de wrf
        self.pathOpen = pathOpen

        self.DatosWRF(self.pathWRF)

        self.rutas_datos_sim = [self.pathOpen+'1000_date40/',
                                self.pathOpen+'1000_date80/',
                                self.pathOpen+'1000_date120/',
                                self.pathOpen+'1000_date160/']
        self.fechas_datos_sim = (np.array(self.dates).astype(str)[[40, 80, 120, 160]]
                                 .tolist())

        # datos malla y puntos de interpolacion
        self.MallaPtsInterpolacion(self.pathOpen+'/system/',
                                   self.pathOpen+'/0/')

        print("\ninicializacion en %.1f s\n"%(time.time()-TIME_INI))


    def MallaPtsInterpolacion (self, ruta_blockMeshDict, ruta_Cfile):
        # datos malla y puntos de interpolacion
        self.__lee_blockMeshDict(self.pathOpen+'/system/')
        self.__lee_CoordCeldas(self.pathOpen+'/0/')
        self.__sacaBasemapRadio()
        self.__coordMalla()
        self.__maskWRFin()

    def __sacaBasemapRadio (self):
        # Cogemos una variable cualquiera, lo que queremos es el basemap para
        # sacar relacion entre coordenadas (lat, lon) y puntos en el mapa
        # (plano)

        d1 = self.d[0]
        self.data = getvar(d1, "ter")

        self.m = get_basemap(self.data)
        self.lons = to_np(self.data.XLONG)
        self.lats = to_np(self.data.XLAT)

        self.xx, self.yy = self.m(self.lons, self.lats)

        # convertToMeters seria el radio de nuestro circulo
        self.R = (self.convertToMeters * (np.max(self.yy)- np.min(self.yy))
                  /(self.data.shape[0]*d1.DY))
        self.CentroXMap = self.m(self.CentroLonSTL, self.CentroLatSTL)[0]
        self.CentroYMap = self.m(self.CentroLonSTL, self.CentroLatSTL)[1]

        dTheta = np.arange(16)+3
        x = self.R*np.cos(np.deg2rad(22.5*dTheta)) + self.CentroXMap
        y = self.R*np.sin(np.deg2rad(22.5*dTheta)) + self.CentroYMap

        self.circcoord = np.array(self.m(x, y, inverse=True))

        # En coordenadas del mapa (planas, no esfericas como lat-lon)
        self.CentroLonSTL_map, self.CentroLatSTL_map = self.m(
            self.CentroLonSTL, self.CentroLatSTL)

    def __coordMalla (self):
        # Coordenadas de los puntos interpolados, es decir, los de la malla
        # de la simulacion CFD

        # Coordenadas del internal field
        interpCoordLonLat_int = np.array(self.m(
            (self.R*(self.centroCellsXint-self.centroX)
             /self.convertToMeters+self.CentroXMap),
            (self.R*(self.centroCellsYint-self.centroY)
             /self.convertToMeters+self.CentroYMap),
            inverse=True))
        self.interpCoord_int = np.vstack((interpCoordLonLat_int[::-1,:],
                                  self.centroCellsZint)).transpose()

        self.Radio_CFDregion_map = max(np.sqrt((
            self.m(self.interpCoord_int[:,1], self.interpCoord_int[:,0])[0]
            -self.CentroLonSTL_map)**2
            +(self.m(self.interpCoord_int[:,1], self.interpCoord_int[:,0])[1]
              -self.CentroLatSTL_map)**2))

        # Coordenadas de los laterales (interpCoord, con z,
        # se calcula mas adelante)
        self.interpCoordLonLat = []
        for indice in np.arange(len(self.centroCellsX)):
            self.interpCoordLonLat.append(np.array(self.m(
                (self.R*(self.centroCellsX[indice]-self.centroX)
                 /self.convertToMeters+self.CentroXMap),
                (self.R*(self.centroCellsY[indice]-self.centroY)
                 /self.convertToMeters+self.CentroYMap),
                inverse=True)))

        # Coordenadas del terreno
        interpCoordLonLat_ter = np.array(self.m(
            (self.R*(self.centroCellsX_ter-self.centroX)
             /self.convertToMeters+self.CentroXMap),
            (self.R*(self.centroCellsY_ter-self.centroY)
             /self.convertToMeters+self.CentroYMap),
            inverse=True))
        self.interpCoord_ter = np.vstack((interpCoordLonLat_ter[::-1,:],
                                  self.centroCellsZ_ter)).transpose()

```

```
153     def __maskWRFin (self):
154         '''
155         Calcula la mask (shape: south_north, west_east) de los datos de wrf
156         para quedarnos con los que estan dentro del dominio de la CFD
157         '''
158         self.dist_centroWRF_map = np.sqrt(
159             (self.m(to_np(self.data.XLONG), to_np(self.data.XLAT))[0]
160              -self.CentroLonSTL_map)**2
161             +(self.m(to_np(self.data.XLONG), to_np(self.data.XLAT))[1]
162               -self.CentroLatSTL_map)**2)
163 
164         self.maskWRFin = self.dist_centroWRF_map<=self.Radio_CFDregion_map
165 
166     def __lee_blockMeshDict (self, ruta_blockMeshDict):
167         with open(ruta_blockMeshDict+'blockMeshDict', 'r') as fichero:
168             for linea in fichero.readlines():
169                 if 'convertToMeters ' in linea:
170                     self.convertToMeters = float(linea.split()[1]
171                                                  .replace(';', ''))
172                 if 'alturaReal ' in linea:
173                     self.alturaMalla = float(linea.split(' ')[1]
174                                              .replace(';',''))
175                 if 'centroX #' in linea:
176                     self.centroX = float(linea.split('"')[1].split('/')[0])
177                 if 'centroY #' in linea:
178                     self.centroY = float(linea.split('"')[1].split('/')[0])
179 
180     def __lee_CoordCeldas (self, ruta_Cfile):
181     # Se requiere llamar al comando 'postProcess -func writeCellCentres'
182     # para tener los ficheros C, Cx, Cy, Cz con las coordenadas de las celdas
183 
184         with open(ruta_Cfile+'Cx', 'r') as fichero:
185             texto = np.array(fichero.readlines())
186             self.centroCellsX = []
187             self.centroCellsX_ter = []
188             for taca in np.arange(len(texto)):
189                 if 'internalField' in texto[taca]:
190                     taca += 1
191                     numCells = int(texto[taca])
192                     taca += 2
193                     self.centroCellsXint = np.array(
194                         texto[taca+np.arange(numCells)]).astype(float)
195                     taca += numCells
196                 if 'lateral' in texto[taca]:
197                     taca += 4
198                     numCells = int(texto[taca])
199                     taca += 2
200                     aux = np.array([texto[taca+np.arange(numCells)]]
201                                    ).astype(float)
202                     self.centroCellsX.append(aux)
203                     taca += numCells
204                 if '_wall' in texto[taca]:
205                     taca += 4
206                     numCells = int(texto[taca])
207                     taca += 2
208                     aux = np.array([texto[taca+np.arange(numCells)]]
209                                    ).astype(float)
210                     self.centroCellsX_ter = np.append(self.centroCellsX_ter, aux)
211                     taca += numCells
212 
213         with open(ruta_Cfile+'Cy', 'r') as fichero:
214             texto = np.array(fichero.readlines())
215             self.centroCellsY = []
216             self.centroCellsY_ter = []
217             for taca in np.arange(len(texto)):
218                 if 'internalField' in texto[taca]:
219                     taca += 1
220                     numCells = int(texto[taca])
221                     taca += 2
222                     self.centroCellsYint = np.array(
223                         texto[taca+np.arange(numCells)]).astype(float)
224                     taca += numCells
225                 if 'lateral' in texto[taca]:
226                     taca += 4
227                     numCells = int(texto[taca])
228                     taca += 2
229                     aux = np.array([texto[taca+np.arange(numCells)]]
230                                    ).astype(float)
231                     self.centroCellsY.append(aux)
232                     taca += numCells
233                 if '_wall' in texto[taca]:
234                     taca += 4
235                     numCells = int(texto[taca])
236                     taca += 2
237                     aux = np.array([texto[taca+np.arange(numCells)]]
238                                    ).astype(float)
239                     self.centroCellsY_ter = np.append(self.centroCellsY_ter, aux)
240                     taca += numCells
241 
242         with open(ruta_Cfile+'Cz', 'r') as fichero:
243             texto = np.array(fichero.readlines())
244             self.centroCellsZ = []
245             self.centroCellsZ_ter = []
246             for taca in np.arange(len(texto)):
247                 if 'internalField' in texto[taca]:
248                     taca += 1
249                     numCells = int(texto[taca])
250                     taca += 2
251                     self.centroCellsZint = np.array(
```

```
                        texto[taca+np.arange(numCells)]).astype(float)
                    taca += numCells
                if 'lateral' in texto[taca]:
                    taca += 4
                    numCells = int(texto[taca])
                    taca += 2
                    aux = np.array([texto[taca+np.arange(numCells)]]
                                   ).astype(float)
                    self.centroCellsZ.append(aux)
                    taca += numCells
                if '_wall' in texto[taca]:
                    taca += 4
                    numCells = int(texto[taca])
                    taca += 2
                    aux = np.array([texto[taca+np.arange(numCells)]]
                                   ).astype(float)
                    self.centroCellsZ_ter = np.append(self.centroCellsZ_ter, aux)
                    taca += numCells

        # Tanto en X, Y, Z debemos tener el mismo numero de celdas...
        assert (len(self.centroCellsX) ==
                len(self.centroCellsY) ==
                len(self.centroCellsZ))
        assert (len(self.centroCellsXint) ==
                len(self.centroCellsYint) ==
                len(self.centroCellsZint))
        assert (len(self.centroCellsX_ter) ==
                len(self.centroCellsY_ter) ==
                len(self.centroCellsZ_ter))

    def printParametros(self):
        print("%-35s"%"Path datos WRF:"+self.pathWRF)
        print("%-35s"%"Fecha inicio/final datos WRF:"
              +str(self.dates[0])+" >> "+str(self.dates[-1]))
        print("%-35s"%"# datos temporales WRF:"+str(len(self.d)))
        print("%-35s"%"Centro Lat/Lon del .stl:"+"%6.3f %6.3f"%(self.CentroLatSTL,
                                                                self.CentroLonSTL))
        print("%-35s"%"Path caso OpenFOAM:"+self.pathOpen)
        print("%-35s"%"Path resultados simulacion:", *self.rutas_datos_sim,
              sep='\n - ')
        print("%-35s"%"Fechas resultados simulacion:", *self.fechas_datos_sim,
              sep='\n - ')

    def printFechas(self):
        '''
        Printea un calendrio en html con las fechas de WRF resaltadas
        Las fechas de la simulacion las subraya
        '''
        for fch in self.dates:
            t = fch.item()

            try:
                if oldM != t.month:
                    display(HTML(ss))

                    # create HTML Calendar month
                    cal = calendar.HTMLCalendar()
                    ss= cal.formatmonth (t.year,t.month)
            except NameError:
                # create HTML Calendar month
                cal = calendar.HTMLCalendar()
                ss= cal.formatmonth (t.year,t.month)

            if str(fch) in self.fechas_datos_sim:
                # Subrayamos las fechas de la simulacion
                ss = ss.replace('>%i<'%t.day,
                                ' bgcolor="#66ff66"><b><u>%i</u></b><'%t.day)
            else:
                # add cell's backgroundcolor and bold html tags around date
                ss = ss.replace('>%i<'%t.day,
                                ' bgcolor="#66ff66"><b>%i</b><'%t.day)

            oldM = t.month

        display(HTML(ss))
        del oldM

    def DatosWRF (self, rutaWRF='/home/jmilla/wrf/Domains/'):
        ruta_filesWRF = self.__filesWRF(rutaWRF)
        # d contiene los datos de wrf
        self.d = self.__ruta_datos(ruta_filesWRF)
        self.dates = np.array(getvar(self.d, "Times", timeidx=ALL_TIMES),
                              dtype="datetime64[h]")

    def __filesWRF (self, rutaWRF, drop=6):
        '''
        Recorre todos los subdirectorios de rutaWRF y busca, lista y ordena
        todos los ficheros que empiecen por wrfout_d04_*
        Por defecto quita los "drop" primeros ficheros que encuentra, ya que
        suponemos que son los de inicializacion del primer periodo de las
        simulaciones de WRF
        '''
        files_name = []
        full_files_name = []
        for (dirpath, dirnames, filenames) in os.walk(rutaWRF, followlinks=True):
            # Recorremos carpetas por orden alfabetico (para poder descartar
            # las primeras 6-7 h de simulacion de cada semana)
            dirnames.sort()
```

```
            for filee in filenames:
                if fnmatch.fnmatchcase(filee, 'wrfout_d04_*'):
                    if filee in files_name:
                        continue
                    files_name.append(filee)
                    full_files_name.append(dirpath+'/'+filee)

        full_files_name = np.array(full_files_name)[np.argsort(files_name)]

        return np.delete(full_files_name, np.s_[0:drop])

    def __ruta_datos (self, files):
        # Devuelve el dataset de los datos "files"
        d1=[]
        for filee in files:
            d1.append(Dataset(filee))
        return d1

    def lee_Resumen(self, path_rsm):
        with open(path_rsm+'Resumen.txt', 'r') as fichero:
            record = False
            id_events = np.array([], dtype=int)
            for linea in fichero.readlines():
                if record:
                    ID = linea.split()[0]
                    id_events = np.append(id_events, int(ID))
                if 'ID_evento               Fecha' in linea:
                    record = True
        self.fechas_datos_sim_indx = id_events.tolist()
        return (np.array(self.dates).astype(str)[self.fechas_datos_sim_indx]
                                                .tolist())

    def genera_rutaSim(self):
        rut = []
        for kk in self.fechas_datos_sim_indx:
            rut.append(self.pathOpen+str(kk)+'/')
        return rut

    def elimina_fechas(self):
        '''
        Elimina del dataset de wrf self.d las fechas que no coincidan con las de las
        simulaciones, para hacer el proceso general mas eficiente
        '''

        self.dates = np.array(getvar(self.d, "Times", timeidx=ALL_TIMES),
                              dtype="datetime64[h]")

        count = 0
        for kk in self.dates.astype(str):
            if 'boundary_' in kk:
                kk = kk.replace('boundary_', '')
            if kk in self.fechas_datos_sim:
                count+=1
            else:
                del self.d[count]

        self.dates = np.array(getvar(self.d, "Times", timeidx=ALL_TIMES),
                              dtype="datetime64[h]")

        assert len(self.d)==len(self.dates)==(len(self.fechas_datos_sim)-
                                              len(self.fechas_datos_boun))

    def Dime_fisicas (self, d):
        print ("\tPBL=%d\tMPH=%d\tCMS=%d\tSWR=%d\tLWR=%d"%(d.BL_PBL_PHYSICS,
                                                          d.MP_PHYSICS,
                                                          d.CU_PHYSICS,
                                                          d.RA_SW_PHYSICS,
                                                          d.RA_LW_PHYSICS))

    def interpolaYescribe (self, d1, ruta_escritura, mltp=1.0, boundary=False, fech_bound='None'):
        '''
        Este metodo interpola los datos WRF referentes a "d1" (una fecha dada)
        en la malla de la simulacion de OpenFOAM y escribe los resultados
        en "ruta_escritura"
        -'mlpt' es un multiplicador, para hacer test
        '''

        # Para tiempos de ejecucion de interpolacion y escritura
        TIME_INI = time.time()

        # Creamos la carpeta, si no esta creada, donde se guarda los datos
        try:
            os.makedirs(ruta_escritura)
        except OSError:
            if not os.path.isdir(ruta_escritura):
                raise

        # Sacamos la altura de los datos de wrf y las coordenadas de estos

        z = getvar(d1, "z")

        datosCoord = []
        for kkk in np.arange(len(z)):
            try:
                datosCoord = np.vstack((datosCoord,
                                        np.vstack((np.array(self.lats).flatten(),
                                            np.array(self.lons).flatten(),
                                            np.array(z[kkk]).flatten()))).transpose()))
            except ValueError:
```

```
                datosCoord = np.vstack((np.array(self.lats).flatten(),
                                        np.array(self.lons).flatten(),
                                        np.array(z[kkk]).flatten())).transpose()


        # Ahora cogemos todos los datos, generamos las funciones de
        # interpolacion y escribimos resultados

        wspdZZZ = [getvar(d1, 'uvmet', timeidx=0)[0],
                   getvar(d1, 'uvmet', timeidx=0)[1],
                   getvar(d1, 'wa', timeidx=0)]

        header = '''/*--------------------------------*- C++ -*----------------------------------*\\
| =========                 |                                                 |
| \\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\\    /   O peration     | Version:  6                                     |
|   \\\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
        '''

        title = ['U [m s-1]', 'V [m s-1]', 'W [m s-1]']
        assert len(title)==len(wspdZZZ)

        # Datos internal field

        fnU = LinearNDInterpolator(datosCoord,
                                   np.array(wspdZZZ[0]).flatten(),
                                   fill_value=0, rescale=True)
        fnV = LinearNDInterpolator(datosCoord,
                                   np.array(wspdZZZ[1]).flatten(),
                                   fill_value=0, rescale=True)
        fnW = LinearNDInterpolator(datosCoord,
                                   np.array(wspdZZZ[2]).flatten(),
                                   fill_value=0, rescale=True)

        interpDatosUint = mltp*fnU(self.interpCoord_int)
        interpDatosVint = mltp*fnV(self.interpCoord_int)
        interpDatosWint = mltp*fnW(self.interpCoord_int)

        with open(ruta_escritura+'internalField', 'w') as fichero:
            fichero.writelines(header)
            fichero.writelines('\ninternalField  nonuniform List<vector>\n')
            fichero.writelines('%d'%len(interpDatosUint)+'\n')
            fichero.writelines('(\n')
            for kk in np.arange(len(interpDatosUint)):
                fichero.writelines("%4s(%8.4f %8.4f %8.4f)\n"
                                   %('',
                                     interpDatosUint[kk],
                                     interpDatosVint[kk],
                                     interpDatosWint[kk]))

            fichero.writelines(');\n')


        # Datos del lateral

        interpDatos = np.array([])
        datosLateral = {}
        for kk in np.arange(len(self.interpCoordLonLat)):
            interpCoord = np.vstack((self.interpCoordLonLat[kk][::-1,:],
                                     np.array([(self.centroCellsZ)[kk]]))).transpose()

            interpDatosU = mltp*fnU(interpCoord[:,0,:])
            interpDatosV = mltp*fnV(interpCoord[:,0,:])
            interpDatosW = mltp*fnW(interpCoord[:,0,:])

            interpListica = np.array([interpDatosU, interpDatosV, interpDatosW])
            datosLateral[kk] = interpListica
            try:
                interpDatos = np.vstack((interpDatos, [interpListica]))
            except ValueError:
                interpDatos = np.array([interpListica])

            if not boundary:
                with open(ruta_escritura+'lateral'+'%02d'%(kk+1)+'U', 'w') as fichero:
                    fichero.writelines(header)
                    fichero.writelines('    lateral'+'%02d\n'%(kk+1))
                    fichero.writelines('    {\n')
                    fichero.writelines('        type            freestream;\n')
                    fichero.writelines('        freestreamValue nonuniform List<vector>\n')
                    fichero.writelines('        %d'%len(interpDatosU)+'\n')
                    fichero.writelines('        (\n')
                    for kkk in np.arange(len(interpDatosU)):
                        fichero.writelines("%12s(%8.4f %8.4f %8.4f)\n"
                                           %('',
                                             interpDatosU[kkk],
                                             interpDatosV[kkk],
                                             interpDatosW[kkk]))

                    fichero.writelines('        );\n')
                    fichero.writelines('    }\n')

        self.datosLateral = datosLateral

        norm_max = np.array([])
        norm_dic = {}
        for kk in np.arange(len(self.interpCoordLonLat)):
            norm_dic[kk] = np.linalg.norm(datosLateral[kk], axis=0)
```

```
            norm_max = np.append(norm_max, np.max(norm_dic[kk]))

        if not boundary:
            try:
                self.norm_max = np.max(np.append(self.norm_max, norm_max))
            except AttributeError:
                self.norm_max = np.max(norm_max)

        if boundary:
            for kk in np.arange(len(self.interpCoordLonLat)):
                self.bound_coef[fech_bound] = (1.1*self.norm_max)/np.max(norm_max)
                with open(ruta_escritura+'lateral'+'%02d'%(kk+1)+'U', 'w') as fichero:
                    fichero.writelines(header)
                    fichero.writelines('    lateral'+'%02d\n'%(kk+1))
                    fichero.writelines('    {\n')
                    fichero.writelines('        type            freestream;\n')
                    fichero.writelines('        freestreamValue nonuniform List<vector>\n')
                    fichero.writelines('        %d'%len(datosLateral[kk][0])+'\n')
                    fichero.writelines('        (\n')
                    for kkk in np.arange(len(datosLateral[kk][0,:])):
                        fichero.writelines("%12s(%8.4f %8.4f %8.4f)\n"
                                           %('',
                                             datosLateral[kk][0,kkk]*self.bound_coef[fech_bound],
                                             datosLateral[kk][1,kkk]*self.bound_coef[fech_bound],
                                             datosLateral[kk][2,kkk]*self.bound_coef[fech_bound]))

                    fichero.writelines('        );\n')
                    fichero.writelines('    }\n')

        print("\ninterpolacion y escritura ejecutado en %.1f s\n"
              %(time.time()-TIME_INI))

    def uniformeYescribe (self, vel, ruta_escritura):
        '''
        Este metodo establece velocidades uniformes de entrada
        - vel contiene las tres componentes del viento que queremos meter
        '''

        # Para tiempos de ejecucion de interpolacion y escritura
        TIME_INI = time.time()

        # Creamos la carpeta, si no esta creada, donde se guarda los datos
        try:
            os.makedirs(ruta_escritura)
        except OSError:
            if not os.path.isdir(ruta_escritura):
                raise

        header = '''/*--------------------------------*- C++ -*----------------------------------*\\
| =========                 |                                                 |
| \\\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\\    /   O peration     | Version:  6                                     |
|   \\\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
        '''

        with open(ruta_escritura+'internalField', 'w') as fichero:
            fichero.writelines(header)
            fichero.writelines('\ninternalField  uniform (%8.4f %8.4f %8.4f);\n'
                               %(vel[0],vel[1], vel[2]))


        # Datos del lateral

        for kk in np.arange(len(self.interpCoordLonLat)):
            with open(ruta_escritura+'lateral'+'%02d'%(kk+1)+'U', 'w') as fichero:
                fichero.writelines(header)
                fichero.writelines('    lateral'+'%02d\n'%(kk+1))
                fichero.writelines('    {\n')
                fichero.writelines('        type            freestream;\n')
                fichero.writelines('        freestreamValue uniform (%8.4f %8.4f %8.4f);\n'
                                   %(vel[0],vel[1], vel[2]))
                fichero.writelines('    }\n')

        print("\nescritura valores uniformes ejecutado en %.1f s\n"
              %(time.time()-TIME_INI))

    def LanzaOF(self, subf=''):
        '''
        Llama al script 'LanzaOF.sh' situado en la raiz del directorio del
        caso de OpenFOAM para lanzar la simulacion de OpenFOAM
        '''

        # Leemos 'decomposeParDict' para saber cuantos proc usar
        with open(self.pathOpen+'system/decomposeParDict', 'r') as fichero:
            for linea in fichero.readlines():
                if 'numberOfSubdomains' in linea:
                    numproc = int(linea.split()[1].replace(';', ''))
                    break

        # Leemos 'controlDict' para poner label al final
        with open(self.pathOpen+'system/controlDict', 'r') as fichero:
            for linea in fichero.readlines():
                if 'endTime' in linea:
                    try:
                        lastTime = int(linea.split()[1].replace(';', ''))
                    except ValueError:
                        pass
```

```
        subprocess.call(['sh',
                         self.pathOpen+'LanzaOF.sh',
                         str(numproc),
                         str(lastTime),
                         str(subf)])

    def ploteaMapa (self,
                    clipToSpain_flag=False,
                    LateralesMalla_flag=True,
                    CentroSTL_flag=True,
                    PuntosWRF_flag=False,
                    PuntosWRFin_flag=True,
                    Estaciones_flag=False,
                    # EstacionesName_flag=False,
                    DatosSim_flag=False,
                    ruta_shpFile=("/home/jmilla/wrf/Domains/20180520/"
                                  +"shp_files/ne_10m_admin_1_states_provinces/"
                                  +"ne_10m_admin_1_states_provinces"),
                                  shpFile="ne_10m_admin_1_states_provinces.shp"):
        # Aqui ploteamos diferentes puntos interesantes

        # Para tiempos de ejecucion de ploteo
        TIME_INI = time.time()

        fig = plt.figure(figsize = (15,10))
        ax = fig.add_subplot(111)

        cs = self.m.contourf(self.xx, self.yy, self.data, 100)
        plt.colorbar(shrink=0.8, label='Altura (m)')

        if clipToSpain_flag:
            sf = shapefile.Reader(ruta_shpFile)
            vertices = []
            codes = []
            for shape_rec in sf.shapeRecords():
                if (("ESP-5825" in shape_rec.record[3])
                    |("ESP-5847" in shape_rec.record[3])
                    |("ESP-5853" in shape_rec.record[3])
                    |("ESP" in shape_rec.record[3])):

                    pts = shape_rec.shape.points
                    prt = list(shape_rec.shape.parts) + [len(pts)]
                    for i in range(len(prt) - 1):
                        for j in range(prt[i], prt[i+1]):
                            vertices.append(self.m(pts[j][0], pts[j][1]))
                        codes += [Path.MOVETO]
                        codes += [Path.LINETO] * (prt[i+1] - prt[i] -2)
                        codes += [Path.CLOSEPOLY]
                    clip = Path(vertices, codes)
                    clip = PathPatch(clip, transform=ax.transData)

            for contour in cs.collections:
                    contour.set_clip_path(clip)

        self.m.readshapefile(ruta_shpFile, shpFile)
        self.m.drawparallels([self.m.projparams['lat_0']],
                         labels=[1,0,0,0], fontsize=9, fmt="%.2f")
        self.m.drawparallels(np.linspace(min(self.m.boundarylats),
                                         max(self.m.boundarylats), 8),
                             labels=[1,0,0,0], fontsize=9, fmt="%.2f")
        self.m.drawmeridians([self.m.projparams['lon_0']],
                         labels=[0,0,0,1], fontsize=9, fmt="%.2f")
        self.m.drawmeridians(np.linspace(self.m.boundarylonmin,
                                         self.m.boundarylonmax, 8),
                         labels=[0,0,0,1], fontsize=9, fmt="%.2f")

        if LateralesMalla_flag:
            for taca in np.arange(len(self.interpCoordLonLat)-1):
                plt.plot(self.m(self.interpCoordLonLat[taca][0,0],
                                self.interpCoordLonLat[taca][1,0])[0],
                         self.m(self.interpCoordLonLat[taca][0,0],
                                self.interpCoordLonLat[taca][1,0])[1],
                         'b.')
            # Saco la ultima para poner la label
            plt.plot(self.m(self.interpCoordLonLat[taca+1][0,0],
                            self.interpCoordLonLat[taca+1][1,0])[0],
                     self.m(self.interpCoordLonLat[taca+1][0,0],
                            self.interpCoordLonLat[taca+1][1,0])[1],
                     'b.', label='Laterales Malla')

        if DatosSim_flag:
            try:
                plt.plot(self.m(self.df_datos_sim.Lon.to_numpy(),
                                self.df_datos_sim.Lat.to_numpy())[0],
                         self.m(self.df_datos_sim.Lon.to_numpy(),
                                self.df_datos_sim.Lat.to_numpy())[1],
                         'y.', label='Datos sim')
            except AttributeError:
                self.datos_sim()
                plt.plot(self.m(self.df_datos_sim.Lon.to_numpy(),
                                self.df_datos_sim.Lat.to_numpy())[0],
                         self.m(self.df_datos_sim.Lon.to_numpy(),
                                self.df_datos_sim.Lat.to_numpy())[1],
                         'y.', label='Datos sim')

        if CentroSTL_flag:
            plt.plot(self.m(self.CentroLonSTL, self.CentroLatSTL)[0],
                     self.m(self.CentroLonSTL, self.CentroLatSTL)[1],
```

```
                    'ro', label='Centro .stl')

        if PuntosWRF_flag:
            plt.plot(self.m(to_np(self.data.XLONG).flatten(),
                      to_np(self.data.XLAT).flatten())[0],
                     self.m(to_np(self.data.XLONG).flatten(),
                      to_np(self.data.XLAT).flatten())[1],
                     'g.', label='Puntos WRF')

        if PuntosWRFin_flag:
            plt.plot(self.m(to_np(self.data.XLONG)[self.maskWRFin].flatten(),
                      to_np(self.data.XLAT)[self.maskWRFin].flatten())[0],
                     self.m(to_np(self.data.XLONG)[self.maskWRFin].flatten(),
                      to_np(self.data.XLAT)[self.maskWRFin].flatten())[1],
                     'c.', label='Puntos WRF in')

        if Estaciones_flag:
            try:
                plt.plot(self.m(self.dfStations.Longitude.to_numpy(),
                                self.dfStations.Latitude.to_numpy())[0],
                         self.m(self.dfStations.Longitude.to_numpy(),
                                self.dfStations.Latitude.to_numpy())[1],
                         'm*', label='Estaciones')
            except AttributeError:
                self.__sacaDatosEstaciones()
                plt.plot(self.m(self.dfStations.Longitude.to_numpy(),
                                self.dfStations.Latitude.to_numpy())[0],
                         self.m(self.dfStations.Longitude.to_numpy(),
                                self.dfStations.Latitude.to_numpy())[1],
                         'm*', label='Estaciones')

        plt.legend()
        plt.show()

        print("\nploteo en %.1f s\n"
              %(time.time()-TIME_INI))

    def __sacaDatosEstaciones(self):
        from selenium import webdriver
        from selenium.webdriver.support import expected_conditions as EC
        from selenium.webdriver.support.ui import WebDriverWait
        from selenium.webdriver.support.ui import Select
        from selenium.webdriver.common.by import By
        from selenium.webdriver.common.keys import Keys
        from selenium.webdriver.chrome.options import Options
        from selenium.common.exceptions import NoSuchElementException

        def decGrad (array):
            return (array[:,0]+array[:,1]/60.0+array[:,2]/3600.0)*array[:,3]

        chrome_options = Options()

        driver = webdriver.Chrome(executable_path=('/home/jmilla/Pruebas/'+
                                                   'Selenium/chromedriver'),
                                  options=chrome_options)

        wait = WebDriverWait(driver, 20)
        tries = 0

        driver.get('https://www.ogimet.com/display_stations.php?lang=en&tipo'+
                   '=OR&isyn=&oaci=&nombre=&estado=Spain&Send=Send')

        table = wait.until(EC.presence_of_all_elements_located(
            (By.XPATH, '/html/body/table/tbody/tr[2]/td[2]/table/tbody')))

        daticos = []
        for tbl in table:
            elements = tbl.find_elements_by_tag_name('tr')
            for elmt in elements[1:]:
                casillas = elmt.find_elements_by_tag_name('td')
                daticos.append([casillas[0].text, casillas[1].text,
                                casillas[2].text, casillas[3].text,
                                casillas[4].text, casillas[5].text,
                                casillas[6].text])
        driver.quit()

        dfStations = pd.DataFrame(daticos, columns=['WMO', 'ICAO', 'Name',
                                                    'Country', 'Latitude',
                                                    'Longitude', 'Altitude'])
        prueba = (dfStations.Latitude
                  .str.replace('-',' ').str.replace('N',' 1')
                  .str.replace('S',' -1').str.split(' ', expand=True)
                  .astype(float))
        areNa = prueba.isna().loc[:,prueba.columns[-1]]
        prueba.loc[areNa, prueba.columns[3]] = prueba.loc[areNa,
                                                          prueba.columns[2]]
        prueba.loc[areNa, prueba.columns[2]] = 0
        dfStations.Latitude = decGrad(prueba.to_numpy())

        prueba = (dfStations.Longitude
                  .str.replace('-',' ').str.replace('E',' 1').str.replace('W',' -1')
                  .str.split(' ', expand=True).astype(float))
        areNa = prueba.isna().loc[:,prueba.columns[-1]]
        prueba.loc[areNa, prueba.columns[3]] = prueba.loc[areNa, prueba.columns[2]]
        prueba.loc[areNa, prueba.columns[2]] = 0

        dfStations.Longitude = decGrad(prueba.to_numpy())

        dfStations.Altitude = dfStations.Altitude.astype(float)
```

```
        self.dfStations = dfStations


    def datos_sim (self, **kwargs):
        # Recogemos los datos de las simulaciones
        '''
        Si se le pasa un argumento llamado "radio" (float entre 0 y 1) criba
        los datos que devuelve, y se queda con los datos dentro del "radio"
        siendo 1 -> se queda con todos, 0 -> no se queda con ninguno
        '''
        # Para tiempos de ejecucion
        TIME_INI = time.time()

        rutas_datos_sim = self.rutas_datos_sim
        fechas_datos_sim = self.fechas_datos_sim

        assert isinstance(rutas_datos_sim, list)
        assert isinstance(fechas_datos_sim, list)
        assert len(rutas_datos_sim) == len(fechas_datos_sim)

        df_datos_sim = []
        for kkk in range(len(rutas_datos_sim)):
            aux = self.__datosU_sim(rutas_datos_sim[kkk],
                                    fechas_datos_sim[kkk])
            if kkk:
                df_datos_sim = df_datos_sim.append(aux, ignore_index=True)
            else:
                df_datos_sim = aux.copy()

        assert len(df_datos_sim)%len(fechas_datos_sim)==0

        if "radio" in kwargs:
            assert kwargs["radio"] >= 0
            assert kwargs["radio"] <= 1
            self.dist_centro_map = np.sqrt((self.m(
                df_datos_sim.Lon.to_numpy(), df_datos_sim.Lat.to_numpy())[0]
                                        -self.CentroLonSTL_map)**2
                                       +(self.m(
                df_datos_sim.Lon.to_numpy(), df_datos_sim.Lat.to_numpy())[1]
                                         -self.CentroLatSTL_map)**2)
            df_datos_reducidos = df_datos_sim.loc[
                self.dist_centro_map<=kwargs["radio"]
                *self.Radio_CFDregion_map].copy()
            # La cantidad de datos para cada fecha de simulacion deben ser
            # los mismos
            assert len(df_datos_reducidos) % len(fechas_datos_sim) == 0

            self.df_datos_sim = df_datos_reducidos

            print("\ndatos leidos en %.1f s\n"
                  %(time.time()-TIME_INI))
            return df_datos_reducidos

        self.df_datos_sim = df_datos_sim
        print("\ndatos leidos en %.1f s\n"
              %(time.time()-TIME_INI))
        return df_datos_sim

    def __datosU_sim (self, ruta_datosU, fecha_sim):
        # Para obtener los resultados de la simulacion
        with open(ruta_datosU+'U', 'r') as fichero:
            texto = np.array(fichero.readlines())
            for taca in np.arange(len(texto)):
                if 'internalField' in texto[taca]:
                    taca += 1
                    numCells = int(texto[taca])
                    taca += 2
                    Ucfd_intDF = (pd.Series(texto[taca
                                                  +np.arange(numCells)])
                                  .str.replace('(','').str.replace(')','')
                                  .str.replace('\n','')
                                 ).str.split(' ',
                                             expand=True).astype(float)
                    Ucfd_intDF.columns = ['Ux', 'Uy', 'Uz']
                    break

        df = Ucfd_intDF.join(pd.DataFrame(self.interpCoord_int,
                                          columns=['Lat', 'Lon',
                                                   'Altura']))
        df.insert(6, "Fecha", fecha_sim)
        return df
```

El código que se adjunta a continuación es una extensión de la clase de Python anterior y añade métodos para el manejo de los datos y su procesamiento para alimentar los modelos de regresión.

```
'''
Definimos funciones para obtener el dataframe buscada, juntando los resultados
de la CFD con los de WRF
'''

from common import *

import multiprocessing as mp

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
12
13  from scipy.spatial import Delaunay
14
15  def interpola_aux(dic):
16      '''
17      Funcion global para la paralelizacion de la interpolacion
18      '''
19      result_dic = {}
20      fn = LinearNDInterpolator(dic['tri'],
21                                dic['values'])
22      result = fn(dic['points'])
23      assert not np.isnan(result).any()
24      result_dic[dic['fecha']] = result
25      return result_dic
26
27  class ROM(simulacionCFD):
28      '''
29      Extendemos la clase de simulacionCFD para recolectar los datos de manera adecuada
30      para poder alimentar modelos estadisiticos, etc
31      '''
32
33      def __variablesWRF (self):
34          '''
35          Devuelve una lista con las variables de WRF que queremos que sean features
36          Tambien devuelve valores interpolados a cierta altura/nivel
37
38          d es el dataset con todos los tiempos de los datos WRF
39          '''
40
41          # Para tiempos de ejecucion
42          TIME_INI = time.time()
43
44          d = self.d
45
46          def sacaVariableWRF (d, var):
47              if isinstance(var, list):
48                  assert len(var)==2
49                  return getvar(d, var[0], ALL_TIMES, units=var[1])
50              else:
51                  return getvar(d, var, ALL_TIMES)
52
53          variables = [["uvmet", 'm s-1'], ["wa", 'm s-1']]
54
55          vvv = []
56          for var in variables:
57              vvv.append(sacaVariableWRF(d, var))
58
59          print("\nvariablesWRF en %.1f s\n"
60                %(time.time()-TIME_INI))
61
62          return vvv
63
64      def __interpolaWRFtoMesh (self, df, datos, fecha, z):
65          '''
66          Interpolamos la vairbale de WRF "datos" en una "fecha" dada
67          a los puntos de la malla para que sean nuestras features
68          "df" es el panda con los datos de las simulaciones que devuelve
69          simulacionCFD.datos_sim()
70          '''
71
72          coord = np.array([to_np(datos.XLAT).flatten(),
73                            to_np(datos.XLONG).flatten()]).transpose()
74          aaa = np.tile(coord, (z.shape[0],1))
75          kk = np.stack((aaa[:,0], aaa[:,1], to_np(z).flatten()), axis=-1)
76
77          fn = LinearNDInterpolator(kk,
78                                    to_np(datos).flatten())
79
80          result = fn(df.loc[fecha == df.Fecha, ('Lat', 'Lon', 'Altura')])
81          assert not np.isnan(result).any()
82          return result
83
84      def indxTimeWRF(self, date):
85          '''
86          Devuelve el indice de wrf que coresponde a la fecha 'date' (YYYY-mm-ddThh)
87          '''
88          try:
89              indxwrf = int(np.nonzero(np.char.find(to_np(self.TimeWRF.astype(str)), date) == 0)[0])
90          except AttributeError:
91              aux = getvar(self.d, 'ter', timeidx=ALL_TIMES)
92              self.TimeWRF = aux.Time
93              indxwrf = int(np.nonzero(np.char.find(to_np(self.TimeWRF.astype(str)), date) == 0)[0])
94
95          return indxwrf
96
97      def indxLatLonWRF(self, lat, lon, verbose=False):
98          '''
99          Nos da los indices de un punto de WRF correspondiente a la lat/lon
100         proporcionadas mas cercanas
101         '''
102         # # x,y son los indices en los arrays de las variables que nos indica el punto escogido
103         try:
104             bm_zz = get_basemap(self.lats_zz)
105             dummy1 = np.array(bm_zz(to_np(self.lats_zz), to_np(self.lons_zz)))
106             dummy2 = np.array(bm_zz(lat, lon))
107         except AttributeError:
108             self.lats_zz = getvar(self.d, "lat", timeidx=0)
109             self.lons_zz = getvar(self.d, "lon", timeidx=0)
110             bm_zz = get_basemap(self.lats_zz)
```

```
            dummy1 = np.array(bm_zz(to_np(self.lats_zz), to_np(self.lons_zz)))
            dummy2 = np.array(bm_zz(lat, lon))

        x, y = np.unravel_index(np.argmin((dummy1[0]-dummy2[0])**2
                                          +(dummy1[1]-dummy2[1])**2),
                                dummy1[0].shape)

        if verbose:
            print( "SOLICITADOS\tLatitud=%.4f Longitud=%.4f"%(lat,lon))
            (lat_u,lon_u)=(self.lats_zz[x,y], self.lons_zz[x,y])
            print( "EMPLEADOS\tLatitud=%.4f Longitud=%.4f"%(lat_u,lon_u))

        return x,y

    def dame_nombre_var(self, var, x, y):
        nombre_var = var.name+'_'+str(x)+':'+str(y)

        try:
            nombre_var += ('_'+str(var.wspd_wdir.values))
        except AttributeError:
            try:
                nombre_var += ('_'+str(var.u_v.values))
            except AttributeError:
                pass
        return nombre_var

    def genera_df_var(self, lat=42.7, lon=-0.36):
        '''
        Genera un dataset para mandarlo posteriormente a dame_full_data
        y cargar variables iguales para toda los puntos de la malla pero
        diferentes para cada fecha
        '''

        # [m]
        z_agl = getvar(self.d, "height_agl", timeidx=ALL_TIMES)
        nivelVel_m = 50

        wa = getvar(self.d, "wa", ALL_TIMES, units='m s-1')
        wa_interp = interplevel(wa, z_agl, nivelVel_m)
        assert not np.isnan(wa_interp).any()

        uvmet = getvar(self.d, "uvmet", ALL_TIMES, units='m s-1')
        u_interp = interplevel(uvmet[0], z_agl, nivelVel_m)
        assert not np.isnan(u_interp).any()
        v_interp = interplevel(uvmet[1], z_agl, nivelVel_m)
        assert not np.isnan(v_interp).any()

        df_var = pd.DataFrame(columns=['Fecha'], data=self.fechas_datos_sim)
#         Aqui indicamos las variables que queremos
        var = [u_interp, v_interp, wa_interp]

        x, y = self.indxLatLonWRF(lat=lat, lon=lon, verbose=False)
        count = 0
        for aux in var:
            dummy1 = []
            for dum in self.fechas_datos_sim:
                if 'boundary' in dum:
                    dum = dum.replace('boundary_', '')

                indxwrf = self.indxTimeWRF(dum)
                dummy1.append(to_np(aux[indxwrf,x,y]))
            if 'boundary' in dum:
                df_var[self.dame_nombre_var(aux[0], x, y)] = self.bound_coef['boundary_'+dum]*dummy1
            else:
                df_var[self.dame_nombre_var(aux[0], x, y)] = dummy1
            count += 1

        return df_var

    def dame_full_data(self, df_var, radio=1):
        '''
        Generamos un panda con los datos de la simulacion CFD y las features de
        WRF
        -"df" es el panda con los datos de las simulaciones que devuelve
        simulacionCFD.datos_sim()
        -"vvv" una lista con las variables de WRF deseadas, como lo devuelve
        la funcion "variablesWRFl"
        -"dates" un numpy array con las fechas de TODAS las simulaciones WRF, del
        mismo tipo que el atributo simulacionCFD.dates
        -"df_var" es un panda del tipo que devuelve genera_df_var
        '''

        # Para tiempos de ejecucion
        TIME_INI = time.time()

        vvv = self.__variablesWRF()
        df = self.datos_sim(radio=radio)
        dates = self.dates
        z = getvar(self.d, "z", timeidx=ALL_TIMES)

        full_data = df.iloc[:, 0:-1].copy()

        dicc = {'u': 'Ux', 'v': 'Uy', 'wa': 'Uz'}

#         --- Para agnadir variables interpoladas a las celdas de CFD ---
        for kkk in vvv:
            print(kkk.name+' ['+kkk.units+']')

            try:
```

```
                lvl = '_'+str(kkk.level.values)
            except AttributeError:
                lvl = ''

            if len(kkk.shape)>4:
                try:
                    cols = kkk.u_v.values
                except AttributeError:
                    cols = kkk.wspd_wdir.values

                aux1 = None
                aux2 = None
                for dum in df.Fecha.unique():
                    if 'boundary' in dum:
                        dum = dum.replace('boundary_', '')
                        indx = int(np.argwhere(dates.astype(str) == dum))
                        indxwrf = int(np.nonzero(np.char.find(to_np(z.Time.astype(str)), dum) == 0)[0])

                        if (aux1 is None) | (aux2 is None):
                            aux1 = (self.bound_coef['boundary_'+dum]
                                    *self.__interpolaWRFtoMesh(df, kkk[0, indx], dum, z[indxwrf]))
                            aux2 = (self.bound_coef['boundary_'+dum]
                                    *self.__interpolaWRFtoMesh(df, kkk[1, indx], dum, z[indxwrf]))
                        else:
                            aux1 = np.append(aux1,
                                             (self.bound_coef['boundary_'+dum]
                                             *self.__interpolaWRFtoMesh(df, kkk[0, indx], dum, z[indxwrf])))
                            aux2 = np.append(aux2,
                                             (self.bound_coef['boundary_'+dum]
                                             *self.__interpolaWRFtoMesh(df, kkk[1, indx], dum, z[indxwrf])))
                    else:
                        indx = int(np.argwhere(dates.astype(str) == dum))
                        indxwrf = int(np.nonzero(np.char.find(to_np(z.Time.astype(str)), dum) == 0)[0])

                        if (aux1 is None) | (aux2 is None):
                            aux1 = self.__interpolaWRFtoMesh(df, kkk[0, indx], dum, z[indxwrf])
                            aux2 = self.__interpolaWRFtoMesh(df, kkk[1, indx], dum, z[indxwrf])
                        else:
                            aux1 = np.append(aux1,
                                             self.__interpolaWRFtoMesh(df, kkk[0, indx], dum, z[indxwrf]))
                            aux2 = np.append(aux2,
                                             self.__interpolaWRFtoMesh(df, kkk[1, indx], dum, z[indxwrf]))

                full_data[cols[0]+lvl] = aux1
                full_data[cols[1]+lvl] = aux2
            else:
                aux1 = None
                z = getvar(self.d, "z", timeidx=ALL_TIMES)
                for dum in df.Fecha.unique():
                    if 'boundary' in dum:
                        dum = dum.replace('boundary_', '')
                        indx = int(np.argwhere(dates.astype(str) == dum))
                        indxwrf = int(np.nonzero(np.char.find(to_np(z.Time.astype(str)), dum) == 0)[0])

                        if (aux1 is None):
                            aux1 = (self.bound_coef['boundary_'+dum]
                                    *self.__interpolaWRFtoMesh(df, kkk[indx], dum, z[indxwrf]))
                        else:
                            aux1 = np.append(aux1,
                                             (self.bound_coef['boundary_'+dum]
                                             *self.__interpolaWRFtoMesh(df, kkk[indx], dum, z[indxwrf])))

                    else:
                        indx = int(np.argwhere(dates.astype(str) == dum))
                        indxwrf = int(np.nonzero(np.char.find(to_np(z.Time.astype(str)), dum) == 0)[0])

                        if (aux1 is None):
                            aux1 = self.__interpolaWRFtoMesh(df, kkk[indx], dum, z[indxwrf])
                        else:
                            aux1 = np.append(aux1,
                                             self.__interpolaWRFtoMesh(df, kkk[indx], dum, z[indxwrf]))

                full_data[kkk.name+lvl] = aux1

        try:
            del aux1, aux2
        except UnboundLocalError:
            pass

#         --- FIN de interpolacion variables celdas CFD ---

#         --- Agnadimos variables fijas para todas las celdas CFD y para cada fecha ---

        rep = len(df.loc[self.fechas_datos_sim[0] == df.Fecha])
        assert rep == (len(df)/len(df.Fecha.unique()))

        for col in df_var.columns[1:]:
            aux1 = None
            for dum in df.Fecha.unique():
                if (aux1 is None):
                    aux1 = np.repeat(df_var.loc[df_var.Fecha==dum, col], rep)
                else:
                    aux1 = np.append(aux1,
                                     np.repeat(df_var.loc[df_var.Fecha==dum, col], rep))

            full_data[col] = aux1

#         --- FIN variables fijas para todas las celdas CFD y para cada fecha ---

```

```
309         # Copiamos la columma de fechas al final
310         full_data[df.columns[-1]] = df[df.columns[-1]]
311 
312         assert len(full_data) == len(df)
313 
314 
315         print("\ndame_full_data en %.1f s\n"
316               %(time.time()-TIME_INI))
317 
318         self.full_data = full_data
319         return self.full_data
320 
321 
322     def dame_keep_data(self, toleranciaMetros=10):
323         '''
324         Seleccionamos los puntos de la malla CFD de manera ordenada
325         alrededor de las lats, lons proporcionadas
326         -full_data_backup: lo que devuelve dame_full_data
327         '''
328 
329         lats = self.lats[self.maskWRFin]
330         lons = self.lons[self.maskWRFin]
331         full_data_backup = self.full_data
332 
333         assert len(lats)==len(lons)
334         tolDeg = toleranciaMetros/111177.5
335 
336         keep_data = []
337         for dla, dlo in zip(lats,lons):
338             keep_data = np.append(keep_data, (full_data_backup.loc[(
339                 (full_data_backup.Lat>=dla-tolDeg)&
340                 (full_data_backup.Lat<=dla+tolDeg)&
341                 (full_data_backup.Lon>=dlo-tolDeg)&
342                 (full_data_backup.Lon<=dlo+tolDeg))].index.to_numpy()))
343 
344         self.keep_data = pd.Index(keep_data).drop_duplicates()
345         self.full_data_keep = self.full_data.loc[self.keep_data]
346 
347         return self.keep_data, self.full_data_keep
348 
349     def actualiza_keep_data(self, keep_data):
350         '''
351         Aqui podemos darle cualquier keep_data y quedarnos con esos datos
352         '''
353         # p.ej keep data podrian ser puntos aleatorios dentro del dominio
354         # np.random.seed(0)
355         # keep_data = np.random.choice(self.full_data.shape[0],
356         #                              int(self.full_data.shape[0]*0.025),
357         #                              replace=False)
358 
359         # O todo los puntos haciendo que
360 #         keep_data=self.full_data.index
361 
362         self.keep_data = pd.Index(keep_data).drop_duplicates()
363         self.full_data_keep = self.full_data.loc[self.keep_data]
364 
365         return self.keep_data, self.full_data_keep
366 
367     def print_keep_data(self):
368         '''
369         Simplemente informativo, sacamos por pantalla cuantos datos habia
370         originalmente y cuantos nos hemos quedado en la criba inicial
371         '''
372 
373         print("Shape original")
374         print(self.full_data.shape)
375         print("Shape cribado")
376         print(self.full_data_keep.shape)
377         print("")
378         print("Nos hemos quedado con un %5.2f%% de los datos originales"%
379               (100*self.full_data_keep.shape[0]/self.full_data.shape[0]))
380 
381     def pinta_puntos_data(self):
382         '''
383         Pinta los puntos de full_data_keep
384         '''
385         fig = plt.figure(figsize = (15,10))
386         ax = fig.add_subplot(111)
387 
388         plt.scatter(x=self.full_data_keep.loc[self.fechas_datos_sim[1] == self.full_data_keep.Fecha, 'Lon'],
389                     y=self.full_data_keep.loc[self.fechas_datos_sim[1] == self.full_data_keep.Fecha, 'Lat'])
390         plt.show()
391 
392     def dame_fechas_test(self, test_pr=0.3, **kwargs):
393         '''
394         Nos dice las fechas que tenemos que dejar para el test
395 
396         Si se le pasa un argumento llamado "fechas_drop" debe ser una lista
397         con las fechas que queremos que escoja para test.
398         Deben ser de la forma 'aaaa-mm-ddThh', ej: '2018-06-02T08'
399         Si es de alguna otra forma simplemente no encontrara esa fecha y
400         es como si ignorase esa entrada
401         '''
402 
403         fechas_drop = np.random.choice(self.fechas_datos_sim,
404                                        int(len(self.fechas_datos_sim)*test_pr),
405                                        replace=False).tolist()
406 
407         if "fechas_drop" in kwargs:
```

```
408             assert isinstance(kwargs["fechas_drop"], list)
409             fechas_drop = kwargs["fechas_drop"]
410
411         print()
412         print("Las fechas test son:")
413         print(fechas_drop)
414         print()
415
416         self.fechas_drop = fechas_drop
417         return self.fechas_drop
418
419     def split_test_train(self, features, labels):
420         '''
421         Hace la separacion entre el train y el test set a partir de
422         las fechas de self.fechas_drop del metodo "dame_fechas_test"
423
424         features y labels son los indicies (empezando por 0)
425         de las columanas de self.full_data_keep que queremos que sean
426         o labels o features
427         '''
428
429         assert isinstance(features, list)
430         assert isinstance(labels, list)
431
432         self.features = features
433         self.labels = labels
434
435         auxIndx = pd.Index([])
436         for fchdrp in self.fechas_drop:
437             auxIndx = auxIndx.join(self.full_data_keep.index[self.full_data_keep.Fecha==fchdrp],
438                                    how='outer')
439
440         self.train_features = (self.full_data_keep
441                                .drop('Fecha',axis=1)
442                                .drop(auxIndx, axis=0)
443                                .iloc[:,features])
444
445         self.test_features = (self.full_data_keep
446                                .drop('Fecha',axis=1)
447                                .loc[auxIndx]
448                                .iloc[:,features])
449
450         self.test_features_full = (self.full_data
451                                .drop('Fecha',axis=1)
452                                .loc[auxIndx]
453                                .iloc[:,features])
454
455         self.train_labels = (self.full_data_keep
456                                .drop('Fecha',axis=1)
457                                .drop(auxIndx, axis=0)
458                                .iloc[:,labels])
459
460         self.test_labels = (self.full_data_keep
461                                .drop('Fecha',axis=1)
462                                .loc[auxIndx]
463                                .iloc[:,labels])
464
465         self.test_labels_full = (self.full_data
466                                .drop('Fecha',axis=1)
467                                .loc[auxIndx]
468                                .iloc[:,labels])
469
470         self.train_fechas = (self.full_data_keep.Fecha
471                                .drop(auxIndx, axis=0))
472
473         self.test_fechas = (self.full_data_keep.Fecha
474                                .loc[auxIndx])
475
476         self.test_fechas_full = (self.full_data.Fecha
477                                .loc[auxIndx])
478
479         print("           %-20s %-20s"%("Train","Test"))
480         print("features:  %-20s %-20s"%(str(self.train_features.shape),
481                                          str(self.test_features.shape)))
482         print("labels:    %-20s %-20s"%(str(self.train_labels.shape),
483                                          str(self.test_labels.shape)))
484         print("fechas:    %-20s %-20s"%(str(self.train_fechas.shape),
485                                          str(self.test_fechas.shape)))
486
487     def dame_maskCFDin (self, lat_lon_puente):
488         '''
489         Devuelve una mask indicando los puntos que
490         estan dentro de la circunferencia inscrita en el poligono regular
491         de 16 lados que forman los laterales
492
493         lat_lon_puente es un array con shape (n_pts, 2)
494         la primera dimension para lat y la segunda para lon
495         '''
496
497         aux1, aux2 = self.m(lat_lon_puente[:,1],lat_lon_puente[:,0])
498
499         self.maskCFDin= (np.sqrt((aux1-self.CentroLonSTL_map)**2
500                         + (aux2-self.CentroLatSTL_map)**2)
501                 <(self.R*np.cos(np.deg2rad(90/8.0)))*0.9)
502         return self.maskCFDin
503
504     def crea_malla_puente(self, n_pts=25,n_niveles_eta=10):
505         '''
506         Crea una malla regular cuadrada con n_ptsxn_pts
```

```
        la primera dimension para lat y la segunda para lon
        y la tercera un nivel vertical eta
        '''

        lats_puente = np.linspace(self.full_data.Lat.min(), self.full_data.Lat.max(), n_pts)
        lons_puente = np.linspace(self.full_data.Lon.min(), self.full_data.Lon.max(), n_pts)
        eta_puente = np.linspace(0,1,n_niveles_eta)
        xv, yv, zv = np.meshgrid(lats_puente, lons_puente, eta_puente)
        lat_lon_puente = np.stack((xv.flatten(), yv.flatten(), zv.flatten()),axis=1)

        return lat_lon_puente

    def altura_ter(self, lat, lon):
        try:
            inter = self.ter_interpolator(lat, lon)
        except AttributeError:
            self.ter_interpolator = LinearNDInterpolator(self.interpCoord_ter[:,0:2],
                                                         values=self.interpCoord_ter[:,2])
            inter = self.ter_interpolator(lat, lon)

        assert not np.isnan(inter).any()
        return inter

    def interpolaVelInt(self, lat, lon, altura, fecha):
        '''
        Interpola a partir de los puntos de la malla CFD a cualquier punto
        (lat, lon, altura) dentro del dominio.
        Da Ux, Uy, Uz
        '''
        try:
            df = self.df_datos_sim
        except AttributeError:
            df = self.datos_sim()

        nd = LinearNDInterpolator(df.loc[df.Fecha==fecha,['Lat', 'Lon', 'Altura']],
                                  values=df.loc[df.Fecha==fecha,['Ux', 'Uy', 'Uz']])
        inter = nd(lat, lon, altura)

        assert not np.isnan(inter).any()
        return inter

    def altura2eta(self, lat, lon, eta, offset):
        return eta*(np.max(self.interpCoord_int[:,2])*0.9-self.altura_ter(lat,lon)-offset)+self.altura_ter(lat,lon)+offset

    def actualiza_malla_puente(self, malla_puente, offset=0):
        aux=pd.DataFrame(malla_puente, columns=['Lat', 'Lon', 'Eta'])
        aux['Altura'] = self.altura2eta(aux.Lat, aux.Lon, aux.Eta, offset)
        self.malla_puente = aux
        return self.malla_puente

    def pinta_mp (self):
        dum = self.full_data.Fecha.unique()[0]
        plt.figure(figsize=(10,10))
        plt.plot(self.full_data.loc[self.full_data.Fecha==dum,['Lon']].to_numpy(),
                 self.full_data.loc[self.full_data.Fecha==dum,['Lat']].to_numpy(), '.')

        plt.plot(self.malla_puente.iloc[:,1], self.malla_puente.iloc[:,0], '.');
        plt.legend(['CFD', 'Malla puente'], loc='lower right')
        plt.xlabel('Longitud')
        plt.ylabel('Latitud')

    def interpola_a_mp(self, col_int, parallel=True):
        '''
        Interpolamos las columnas de self.full_data a la
        malla puente.
        Opcion para lanzar esta interpolacion en paralelo
        '''

        dum = self.full_data.Fecha.unique()[0]
        tri_aux = Delaunay(self.full_data.loc[self.full_data.Fecha==dum,
                                              ['Lat',
                                               'Lon',
                                               'Altura']].to_numpy())

        dic_list = []
        resultadicos = []
        for dum in self.full_data.Fecha.unique():
            dic = {}
            dic['tri'] = tri_aux
            dic['points'] = self.malla_puente.loc[:,['Lat',
                                                     'Lon',
                                                     'Altura']].to_numpy()
            dic['values'] = self.full_data.loc[self.full_data.Fecha==dum,
                                               col_int].to_numpy()
            dic['fecha'] = dum

            if parallel:
                dic_list.append(dic)
            else:
                resultadicos.append(interpola_aux(dic))

        if parallel:
            pool = mp.Pool(processes=mp.cpu_count()-1)
            resultadicos = pool.map(interpola_aux, dic_list)

        self.interpAmp = dict((key,d[key]) for d in resultadicos for key in d)
        return self.interpAmp

    def dame_full_data_mp(self, df_var, columns, parallel):
```

```
        full_data_mp = pd.DataFrame([])

        # --- Agnadimos valores interpolados ---

        self.interpola_a_mp(columns, parallel)

        for kk in np.arange(len(columns)):
            aux1 = None
            for dum in self.full_data.Fecha.unique():
                assert self.interpAmp[dum].shape[1] == len(columns)

                if (aux1 is None):
                    aux1 =  self.interpAmp[dum][:,kk]

                else:
                    aux1 = np.append(aux1,
                                     self.interpAmp[dum][:,kk])

            full_data_mp[columns[kk]] = aux1

        # --- FIN valores interpolados ---

        # --- Agnadimos las coordenadas de la mp

        for kk in np.arange(self.malla_puente.shape[1]):
            aux1 = np.tile(self.malla_puente.to_numpy()[:,kk],
                    len(self.full_data.Fecha.unique()))

            full_data_mp[self.malla_puente.columns[kk]] = aux1


        # --- FIN Agnadimos las coordenadas de la mp

        #         --- Agnadimos variables fijas para la mp y para cada fecha ---

        rep = self.malla_puente.shape[0]
        aa = df_var.columns[1:].to_list()
        aa.append(df_var.columns[0])

        for col in aa:
            aux1 = None
            for dum in self.full_data.Fecha.unique():
                if (aux1 is None):
                    aux1 = np.repeat(df_var.loc[df_var.Fecha==dum, col], rep)
                else:
                    aux1 = np.append(aux1,
                                     np.repeat(df_var.loc[df_var.Fecha==dum, col], rep))

            full_data_mp[col] = aux1

        #         --- FIN variables fijas para la mp y para cada fecha ---


        self.full_data_mp = full_data_mp
        return full_data_mp

```

B.3. Selección de eventos significativos

La implementación en Python que se ha realizado para la selección de eventos significativos se adjunta a continuación.

```
#!/usr/bin/env python
# coding: utf-8

# Este programa intenta determinar los instantes representativos de un punto en una simulacion de wrf,
# a partir, basicamente, de velocidades y direcciones mas frecuentes del viento

import subprocess
import os
import inspect
import sys

if sys.version_info[0] == 2:
    os.environ['PROJ_LIB'] = ('/home/jmilla/anaconda2/pkgs/'
                              +'proj4-5.2.0-he1b5a44_1004/share/proj')
elif sys.version_info[0] == 3:
    os.environ['PROJ_LIB'] = '/home/jmilla/anaconda3/share/proj'
else:
    raise Exception("Version de Python: "+str(sys.version_info[0]))

# Para poder abrir muchos ficheros
import resource
soft, hard = resource.getrlimit(resource.RLIMIT_NOFILE)
resource.setrlimit(resource.RLIMIT_NOFILE, (hard, hard))

import numpy as np
import pandas as pd
import time

import matplotlib.pyplot as plt
import matplotlib.cm as cm

```

```
from netCDF4 import Dataset

import mpl_toolkits.basemap
from mpl_toolkits.basemap import Basemap

from wrf import getvar, get_basemap ,geo_bounds, ALL_TIMES, to_np, rh, disable_xarray
from wrf import xy_to_ll, ll_to_xy, vertcross, CoordPair, interplevel

from windrose import WindroseAxes

# Control de tiempos
inicio_analisis=time.time()

# Las cinco pimeras variables se pueden leer de un archivo llamado Variables.txt si lee_archivo=1
lee_archivo=1

# Los ****** marcan los parametros de ajuste del programa, se puede editar lo que este dentro de ellos,
# no el nombre de las variables, pero si su contenido

# ************************************EDITABLE************************************************

# Ruta de los datos de los que se saca la estadistica
files = ["/home/jmilla/wrf/Domains/AllDates/wrfout_d04_"]

# Ruta del directorio en el que se guardaran los resultados del analisis
ruta_save_fig="/home/jmilla/Windows/Curso_2019-2020/TFM/Script_AnalizaEstadistica/Resultados"
# Ruta del directorio en el que se guardaran los .csv de cada evento escogido
# (Ojo, en cada ejecuccion del programa borrara los .csv que se encuentre en el directorio especificado)
ruta_save_csv="/home/jmilla/Windows/Curso_2019-2020/TFM/Script_AnalizaEstadistica/CSV"

# Se especifica la fecha inicial y final de los datos del analisis (o un rango de fechas contenido en los datos WRF)
ini="2018-05-20T00"
fin="2018-06-03T00"

# Se puede pedir una lat,long especificas, pero para evitar que se salga del dominio, default es el punto central
# Para especificar lat y long, establecer un valor diferente a 999, aunque se recomienda no tocarlo, ya que no se
# interpolan, se coge un punto de la malla.
lat_s=42.7
lon_s=-0.36

# Altura de interpolacion (en m) de los datos, en esta altura se evaluara
# la frecuencia del modulo y direccion del viento
altura_interpol=50.0

# string que se pone delante del nombre de los productos del analisis, para diferenciarlos
NUM_test=str(altura_interpol)+"m_"

# Numero de bins de velocidad, que tendra la rosa de los vientos
n_bins=10
# vel minima permitida para los eventos
vel_min = 2

# Numero de direcciones que tendra la rosa de los vientos
n_direcc=64

# Numero maximo total de eventos significativos que se quieren determinar
# (finalmente se puede llegar a un numero inferior de eventos)
n_events=150
# n_events que seran eventos significativos extremos
n_events_extr=64

# ************************************FIN_EDITABLE************************************************
dir_section = 360.0/n_direcc

if (lee_archivo):
    f=open(os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
           +"/Variables.txt","r")
    leido=f.readlines()
    f.close()
    ini=str(leido[4][5:-1]).replace('"','')
    fin=str(leido[5][5:-1]).replace('"','')
    files=[str(leido[7][8:-3])]
    ruta_save_fig=str(leido[9][15:-2])
    ruta_save_csv=str(leido[12][15:-2])
    print(ini,fin)
    print(files)
    print(ruta_save_fig)

# Creamos la carpeta, si no esta creada, donde se guarda el resultado del analisis
try:
    os.makedirs(ruta_save_fig)
except OSError:
    if not os.path.isdir(ruta_save_fig):
        raise
ruta_save_fig=ruta_save_fig+"/"


# Para datos de wrf de varias horas (en diferentes archivos), esta funcion los va cogiendo todos
# y los agrupa en un unico dataset, que agrupa todos los datos para todas las horas de la simulacion
def ruta_datos (files):
    date_ini=np.datetime64(ini,"h")
    date_fin=np.datetime64(fin,"h")
    dates_aux=np.arange(date_ini,date_fin+1,dtype="datetime64[h]")

    d1=[]
    data_name=[]
    dates=[]
    for i in np.arange(0,len(dates_aux),1):
        dates.append(str(dates_aux[i]).replace("T","_").replace("'",""))
```

```
                    .replace("[","").replace("]","")+":00:00")
        data_name.append(files+dates[i])
        d1=np.append(d1,Dataset(files+dates[i]))
    return d1

print(files[0])
d1=ruta_datos(files[0])
print(files[0].split("/")[len(files[0].split("/"))-2])

Times=getvar(d1,"Times",timeidx=ALL_TIMES)
Times=np.arange(to_np(Times[0]),to_np(Times[len(d1)-1]),dtype='datetime64[h]')
Times=np.append(Times,Times[len(Times)-1]+1)

# Si no se ha especificado lat y lon, coge el punto central
if (lat_s==999):
    lat=d1[0].CEN_LAT
else:
    lat=lat_s

if (lon_s==999):
    lon=d1[0].CEN_LON
else:
    lon=lon_s

lats_zz = getvar(d1, "lat", timeidx=0)
lons_zz = getvar(d1, "lon", timeidx=0)
bm_zz = get_basemap(lats_zz)
dummy1 = np.array(bm_zz(to_np(lats_zz), to_np(lons_zz)))
dummy2 = np.array(bm_zz(lat, lon))

x, y = np.unravel_index(np.argmin((dummy1[0]-dummy2[0])**2
                                  +(dummy1[1]-dummy2[1])**2),
                        dummy1[0].shape)

print("SOLICITADOS\tLatitud=%.4f Longitud=%.4f"%(lat,lon))
(lat_u,lon_u)=(lats_zz[x,y], lons_zz[x,y])
print("EMPLEADOS\tLatitud=%.4f Longitud=%.4f"%(lat_u,lon_u))

print(x,y)


# Obtenemos diferentes variables del wrf

# # Altura de cada punto de la grid
print(\'z_agl\')
z_agl = to_np(getvar(d1, "height_agl", timeidx=ALL_TIMES)[:,:,x,y])
print(z_agl.shape)

# Direccion y modulo del viento
print("vdir")
vdir=to_np(getvar(d1,"uvmet_wspd_wdir",timeidx=ALL_TIMES)[:,:,:,x,y])

velocidades=vdir[0]
print(velocidades.shape)
direcc=vdir[1]
print(direcc.shape)')

# vel_interp = interplevel(velocidades, z_agl, altura_interpol)
vel_interp = []
dir_interp = []
for alt, vel, a_dir in zip(z_agl, velocidades, direcc):
    vel_interp = np.append(vel_interp,
                           np.interp(altura_interpol,alt,vel))
    dir_interp = np.append(dir_interp,
                           np.interp(altura_interpol,alt,a_dir))
assert not np.isnan(vel_interp).all()
assert not np.isnan(dir_interp).all()
del velocidades
del direcc
del z_agl
del vdir

df_interp = pd.DataFrame(np.array([vel_interp, dir_interp]).transpose(),
                         columns=['Vel', 'Dir'])
df_interp.drop(df_interp.loc[df_interp.Vel<vel_min].index, inplace=True)

# Saca la rosa de los vientos y el histograma de direcciones, en el punto especificado y para
# la altura de interpolacion dada
# Ademas recoge en variables esta informacion para su posterior uso

# Rosa de los vientos
ax = WindroseAxes.from_ax()
ax.bar(df_interp.Dir, df_interp.Vel, normed=0, opening=0.8,
       bins=n_bins,edgecolor='black', cmap=cm.Blues, nsector=n_direcc)
ax.tick_params(axis='both', which='major', labelsize=20)
ax.set_legend()
ax.legend(loc=2, frameon=1, bbox_to_anchor=(1.1, 1), fontsize=20)
plt.savefig((ruta_save_fig+NUM_test+"Wind_rose.png"), bbox_inches='tight')

table = ax._info['table']
wd_freq = np.sum(table, axis=0)
bins=ax._info["bins"]

fig, ax = plt.subplots(figsize=(10,5))

# Histograma
width=5
idn=np.linspace(0.0,width*n_direcc,n_direcc)
ax.bar(idn, wd_freq, width=width)
```

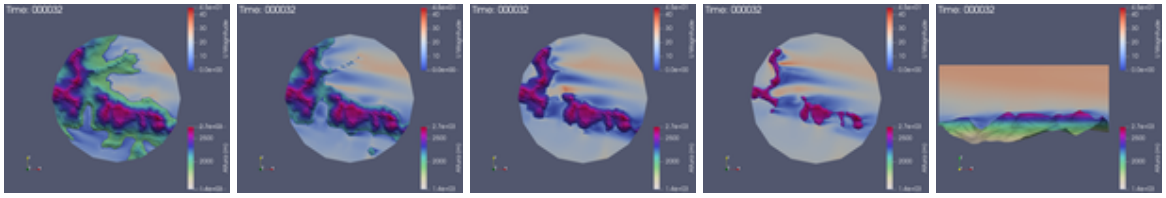
```
xticks=idn+width*0.5
ax.set_xticks(xticks)
ax.grid()
plt.savefig((ruta_save_fig+NUM_test+"Wind_rose_hist.png"), bbox_inches='tight')

def print_hist_dir(aux_dir):
    # Histograma
    fig, ax = plt.subplots(figsize=(10,5))
    width=5
    idn=np.linspace(0.0,width*n_bins,n_bins)
    ax.bar(idn, table[:,aux_dir], width=width)
    xticks=idn+width*0.5-width
    ax.set_xticks(xticks)
    ax.set_xticklabels(np.round(bins,2)[0:-1].astype(str))

    plt.xlabel('Velocidad [m/s]')
    plt.ylabel('Frecuencia')
    plt.title('Direccion %6.2f'%(aux_dir*dir_section))

    ax.grid()

def dame_TimeID(v_min, v_max, d_min, d_max):
    '''
    Devuelve los indices temporales de aquellos instantes
    que tengan velocidad entre v_max y v_min y direccion entre d_max y d_min
    '''

    assert d_max > 0

    if d_min>360:
        d_min = d_min - (d_min//360)*360
    if d_max>360:
        d_max = d_max - (d_max//360)*360

    if d_min<0:
        d_min = 360 + d_min
        indx1 = dame_TimeID(v_min, v_max, d_min, 360)
        indx2 = dame_TimeID(v_min, v_max,     0, d_max)

        return np.append(indx1, indx2)

    indx = df_interp.loc[(df_interp.Vel>=v_min)&
                         (df_interp.Vel<v_max)&
                         (df_interp.Dir>=d_min)&
                         (df_interp.Dir<d_max)].index.to_numpy()
    return indx

def decide_Evento(indx, maxi=False):
    '''
    De las velocidades que se le pasa se queda con la mayor si maxi=True
    si maxi=False se queda con la intermedia

    Devuelve un array, si indx es empty devuelve un array vacio
    '''

    if len(indx) == 0:
        return np.array([])
    aux = df_interp.loc[indx].Vel.sort_values()
    if maxi:
        return aux.index[-1]
    else:
        return aux.index[int(len(aux)/2)]

def escoge_EventExt():
    sele = []
    for dd in np.argsort(-wd_freq):
        try:
            a = np.nonzero(table[:,dd])[-1][-1]
        except IndexError:
            continue
        indx = dame_TimeID(bins[a], bins[a+1],
                           (dd-0.5)*dir_section, (dd+0.5)*dir_section)
        sele = np.append(sele, decide_Evento(indx, maxi=True))

        if len(sele)>=int(n_events_extr):
            break
    return sele.astype(int)

selecc_ext = escoge_EventExt()

def isBin_adyc(list_dir, direc):
    result = True
    if len(list_dir) == 0:
        pass
    else:
        list_dir = np.array(list_dir)
        result = (np.abs(list_dir-direc)>1).all()
    return not result

def escoge_EventFreq(n_each=1):
    sele = []
    for dd in np.arange(0,n_direcc,n_each):
        try:
            a = np.argsort(-table[:,dd])[0]
        except IndexError:
            continue

        indx = dame_TimeID(bins[a], bins[a+1],
                           (dd-0.5)*dir_section, (dd+0.5)*dir_section)
```

```
        sele = np.append(sele, decide_Evento(indx, maxi=True))

        if len(sele)>=int(n_events):
            break
    return sele.astype(int)

selecc_freq = escoge_EventFreq()

def escoge_EventNor():
   sele = np.arange(0)
   dir_sele = np.arange(0)
   vel_sele = np.arange(0)
   dic = {}
   for kk in np.arange(-1, n_direcc+1):
       dic[kk] = np.arange(0)
   for a in np.argsort(-table, axis=None):
       kk = a//n_direcc
       dd = a%n_direcc
       if (isBin_adyc(dic[dd], kk) |
           (kk in dic[dd-1]) |
           (kk in dic[dd+1]) | (table[kk,dd]==0)):
           continue
       dic[dd] = np.append(dic[dd], kk)
       dir_sele = np.append(dir_sele, dd)
       vel_sele = np.append(vel_sele, kk)
       indx = dame_TimeID(bins[kk], bins[kk+1],
                          (dd-0.5)*dir_section, (dd+0.5)*dir_section)
       sele = np.append(sele, decide_Evento(indx, maxi=False))


       if len(sele)>=int(n_events)-len(selecc_freq):
           break

   assert len(sele) == len(dir_sele) == len(vel_sele)
   return sele.astype(int), dir_sele.astype(int), vel_sele.astype(int)

selecc_nor, a, aa = escoge_EventNor()

plt.figure(figsize=(10,13))
for kk in np.arange(n_bins):
    plt.scatter(np.repeat(kk, len(np.nonzero(table[kk,:])[0])),
                np.nonzero(table[kk,:]),
                c='orange', marker='s', s=80)
plt.scatter(aa, a, marker='o')
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.xlabel('Vel [bin]', fontsize=20)
plt.ylabel('Dir [bin]', fontsize=20)
plt.savefig((ruta_save_fig+NUM_test+"VelDirBins_Distr.png"), bbox_inches='tight')

fig = plt.figure(figsize=(10,7))
plt.scatter(df_interp.loc[selecc_nor].Vel, df_interp.loc[selecc_nor].Dir,
            label='normal')
plt.scatter(df_interp.loc[selecc_freq].Vel, df_interp.loc[selecc_freq].Dir,
            marker='x', label='maxFreq')
plt.scatter(df_interp.loc[selecc_ext].Vel, df_interp.loc[selecc_ext].Dir,
            marker='.', label='extreme', )

plt.xlabel('Vel [m/s]', fontsize=20)
plt.ylabel('Dir', fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.legend(fontsize=20)
plt.savefig((ruta_save_fig+NUM_test+"VelDir_Distr.png"), bbox_inches='tight')

final_selec = pd.DataFrame(np.append(selecc_nor,
                                     np.append(selecc_ext,
                                               selecc_freq))).drop_duplicates().to_numpy()[:,0]

with open(ruta_save_fig+"Resumen.txt","w") as outfile:
    fmt=" - %-50s: %s\n"
    outfile.write('{:!^34s}'.format(''))
    outfile.write('{:^35}'.format('\n! RESUMEN ANALISIS ESTADISTICO v2!'))
    outfile.write('{:!<35s}'.format('\n'))
    outfile.write("\n\n")

    outfile.write(fmt%("Fecha ini",ini[:-3]+"_"+ini[-2:]+":00:00"))
    outfile.write(fmt%("Fecha fin",fin[:-3]+"_"+fin[-2:]+":00:00"))
    outfile.write(fmt%("#Eventos temporales de entrada",len(Times)))
    outfile.write(fmt%("#Eventos que superan el umbral de %.2f m/s"%vel_min,len(df_interp)))
    outfile.write(fmt%("#Eventos totales",len(final_selec)))
    outfile.write(fmt%("Tiempo total proceso","%.2f min"%((time.time()-inicio_analisis)/60.0)))
    outfile.write("ID_evento es el indice temporal de los eventos temporales de entrada\n")
    outfile.write("\n\n")
    fmt="%15s    %15s    %15s    %15s\n"
    outfile.write(fmt%("ID_evento",
                       "Fecha",
                       "Velocidad(m/s)",
                       "Direccion(deg)"))

    for time_id in np.sort(final_selec):
        outfile.write(fmt%(time_id,Times[time_id],
                           "%.2f"%df_interp.loc[time_id].Vel,
                           "%.2f"%df_interp.loc[time_id].Dir))

np.random.seed(1234)
np.random.choice(np.setdiff1d(df_interp.index.to_numpy(), final_selec), 80, replace=False)
escoge_EventFreq(n_each=n_direcc//16)
```

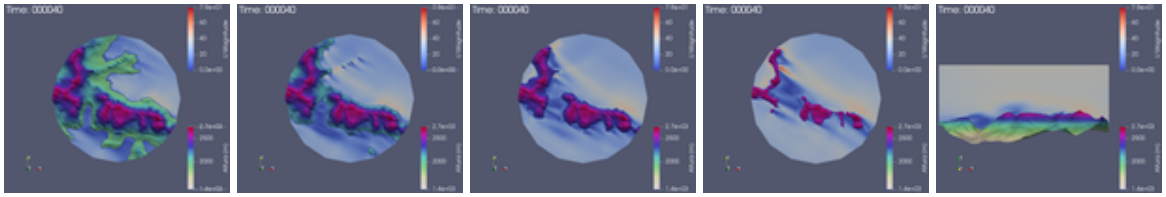
Apéndice C

Resultados CFD

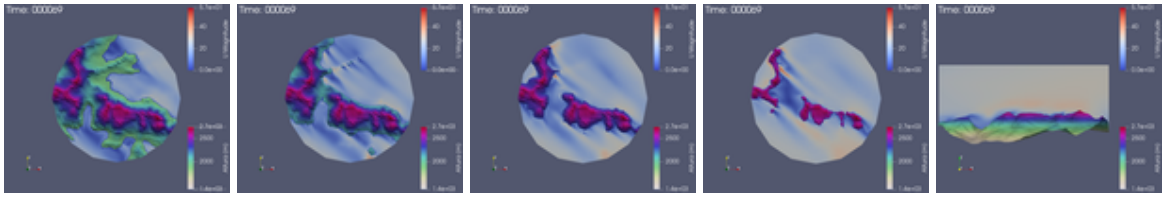
En las siguientes páginas se presentan algunas secciones (horizontales a diferentes alturas y una vertical) de los resultados obtenidos en las simulaciones CFD que se llevan a cabo en este proyecto empleando OpenFOAM (sección 2.2).



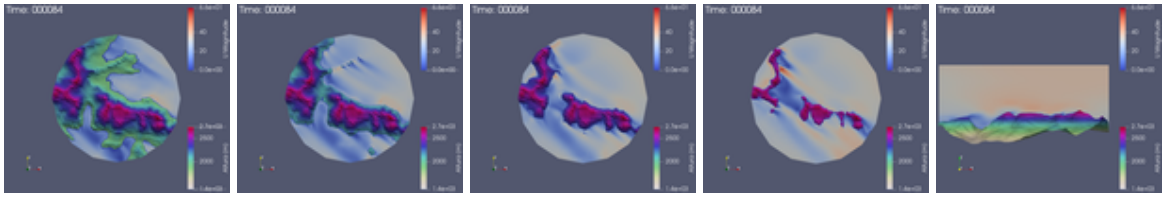
1900 m 2100 m 2300 m 2500 m Lateral Y01



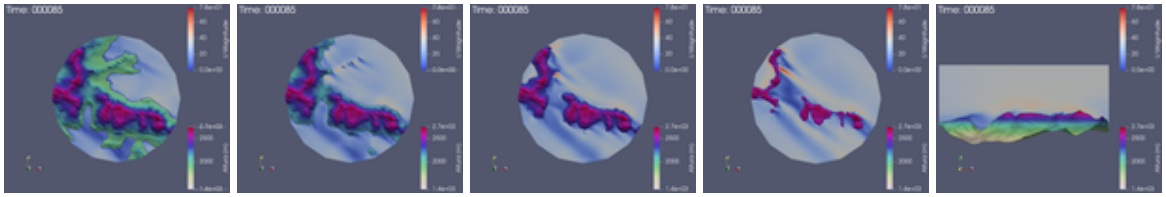
1900 m 2100 m 2300 m 2500 m Lateral Y01



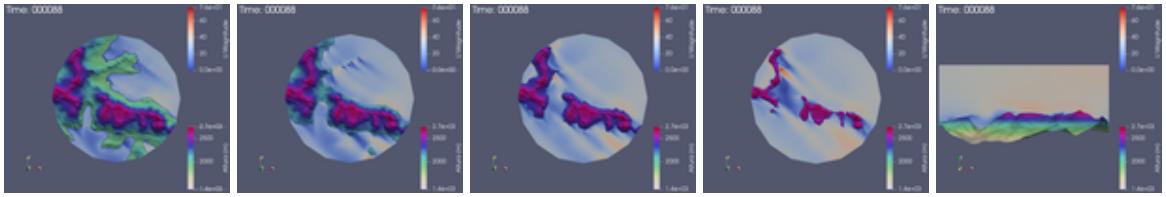
1900 m 2100 m 2300 m 2500 m Lateral Y01



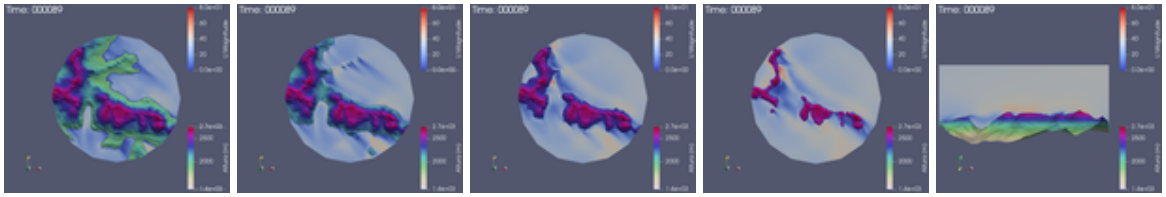
1900 m 2100 m 2300 m 2500 m Lateral Y01



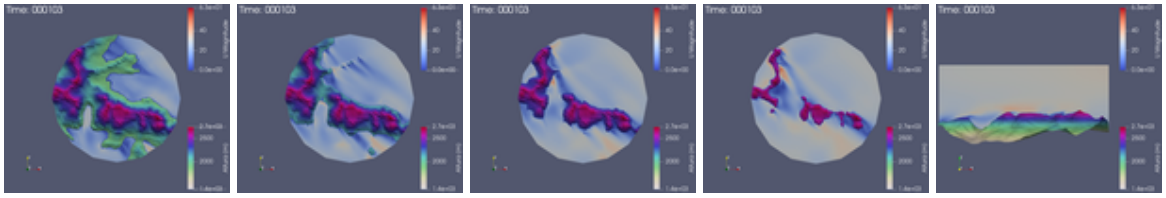
1900 m 2100 m 2300 m 2500 m Lateral Y01



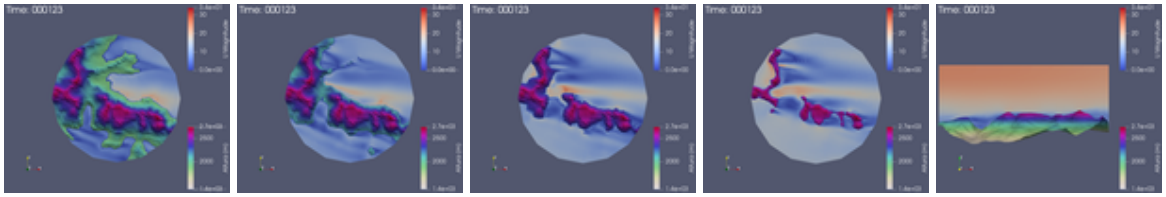
1900 m 2100 m 2300 m 2500 m Lateral Y01



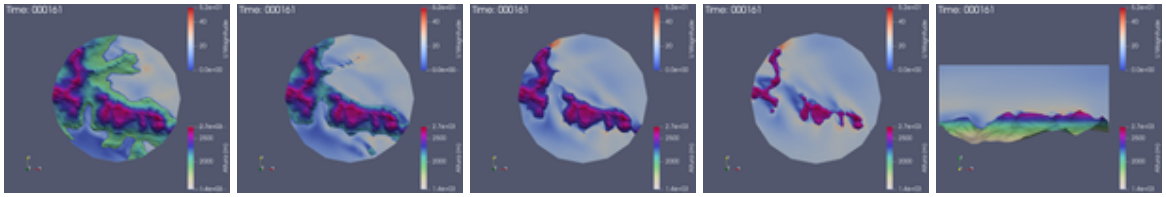
1900 m 2100 m 2300 m 2500 m Lateral Y01



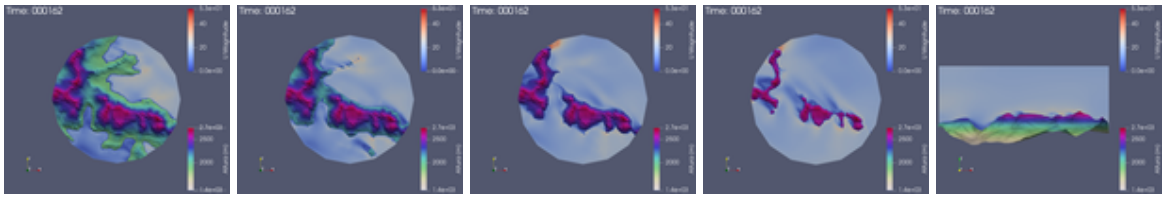
1900 m 2100 m 2300 m 2500 m Lateral Y01



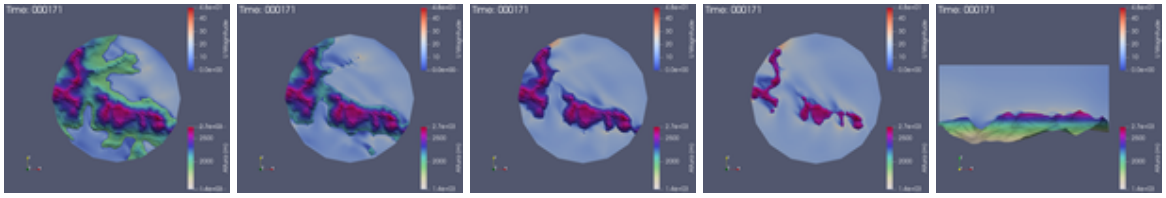
1900 m 2100 m 2300 m 2500 m Lateral Y01



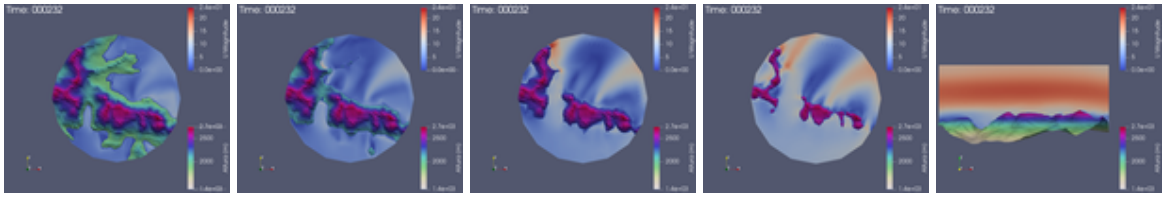
1900 m 2100 m 2300 m 2500 m Lateral Y01



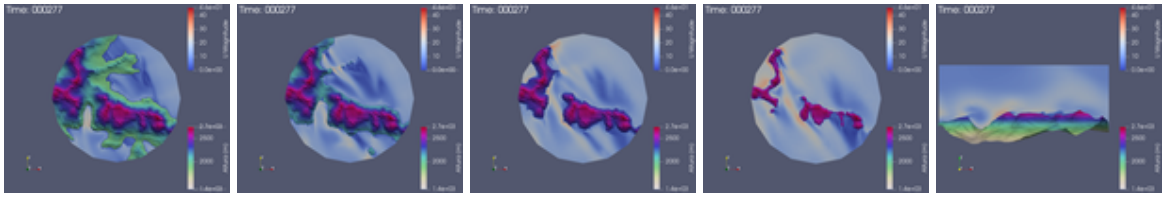
1900 m 2100 m 2300 m 2500 m Lateral Y01



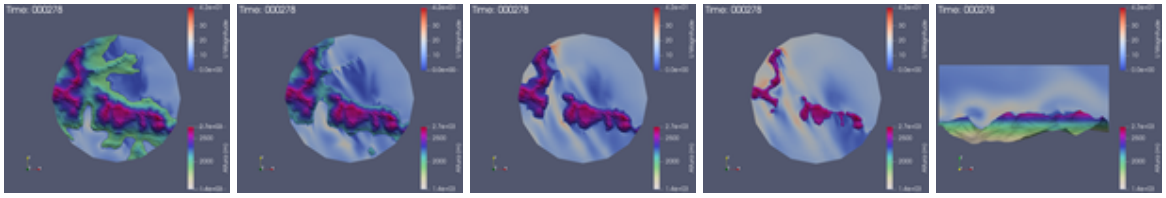
1900 m 2100 m 2300 m 2500 m Lateral Y01



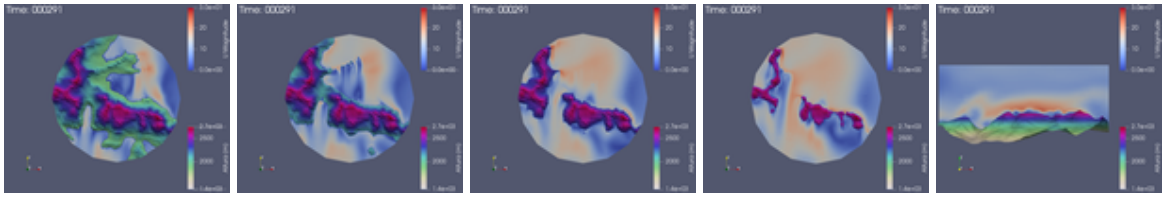
1900 m 2100 m 2300 m 2500 m Lateral Y01



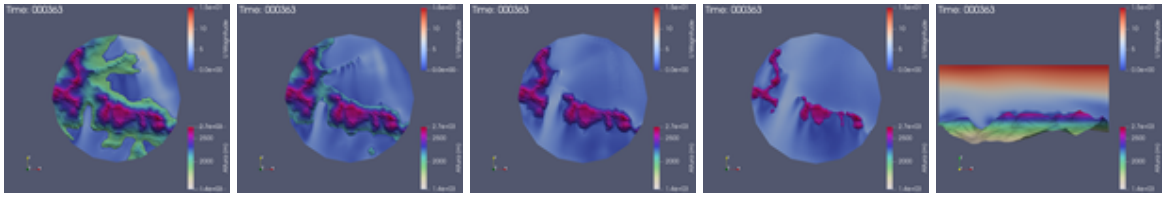
1900 m 2100 m 2300 m 2500 m Lateral Y01



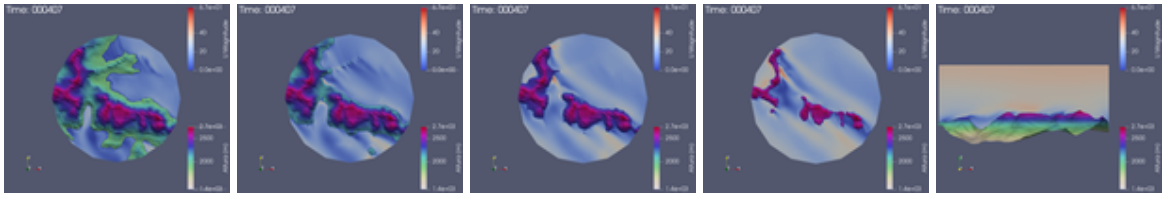
1900 m 2100 m 2300 m 2500 m Lateral Y01



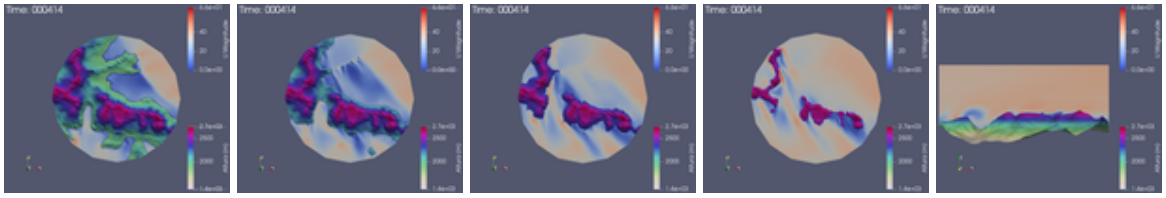
1900 m 2100 m 2300 m 2500 m Lateral Y01



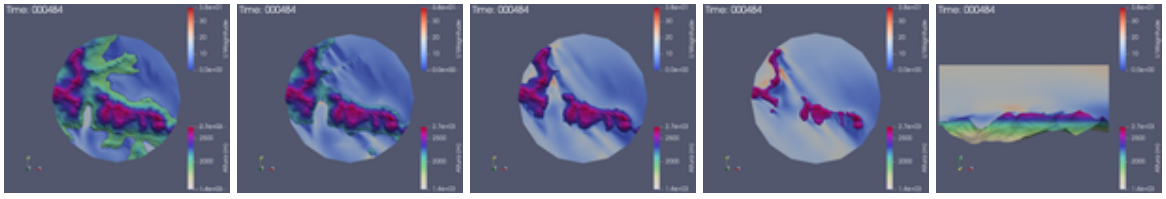
1900 m 2100 m 2300 m 2500 m Lateral Y01



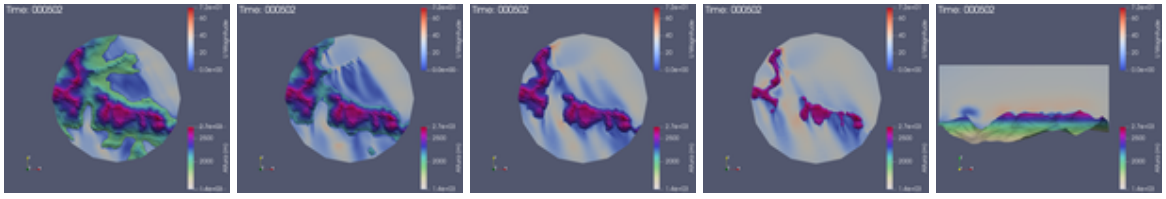
1900 m 2100 m 2300 m 2500 m Lateral Y01



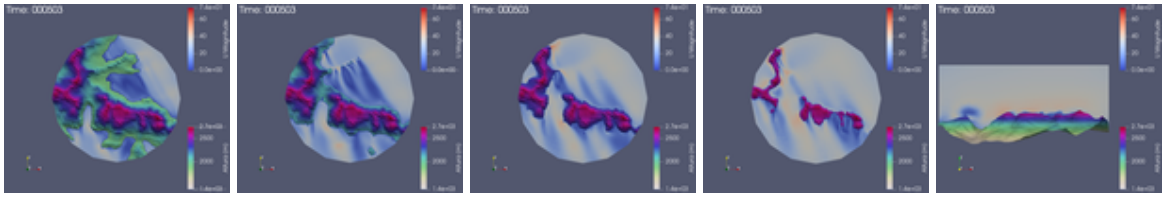
1900 m 2100 m 2300 m 2500 m Lateral Y01



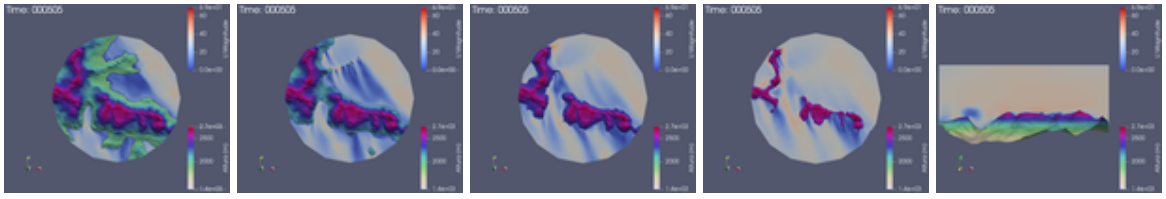
1900 m 2100 m 2300 m 2500 m Lateral Y01



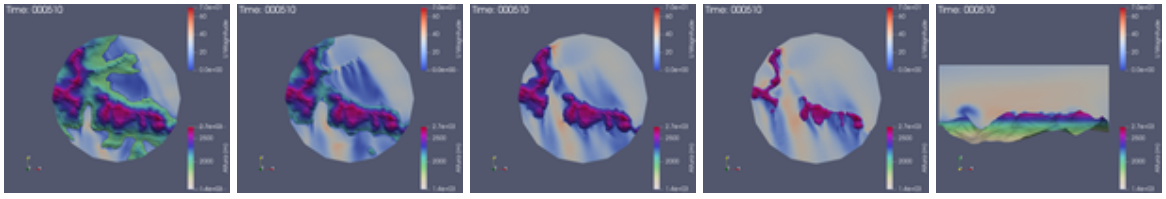
1900 m 2100 m 2300 m 2500 m Lateral Y01



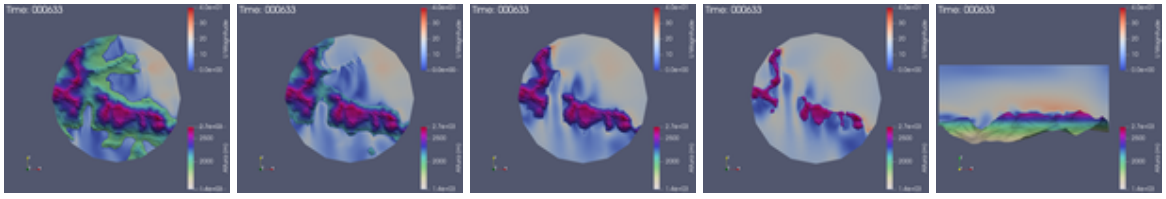
1900 m 2100 m 2300 m 2500 m Lateral Y01



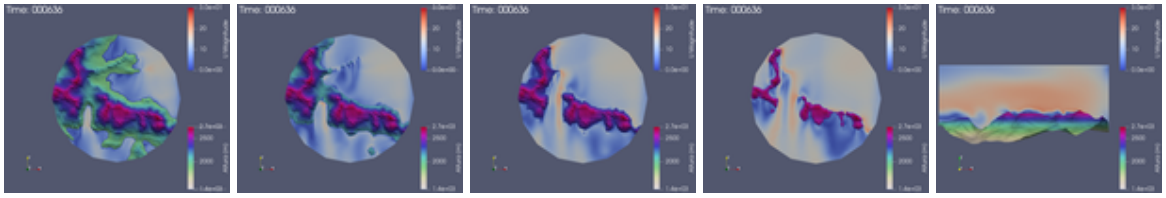
1900 m 2100 m 2300 m 2500 m Lateral Y01



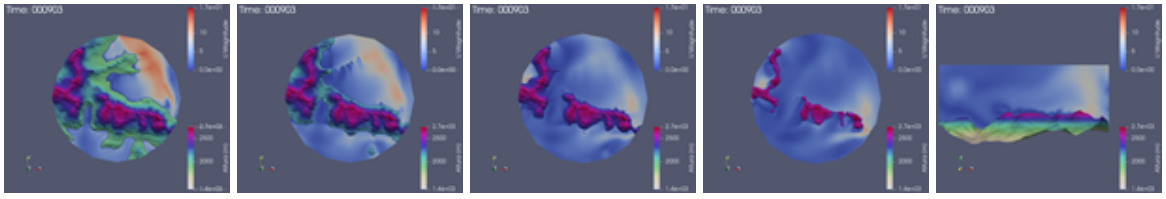
1900 m 2100 m 2300 m 2500 m Lateral Y01



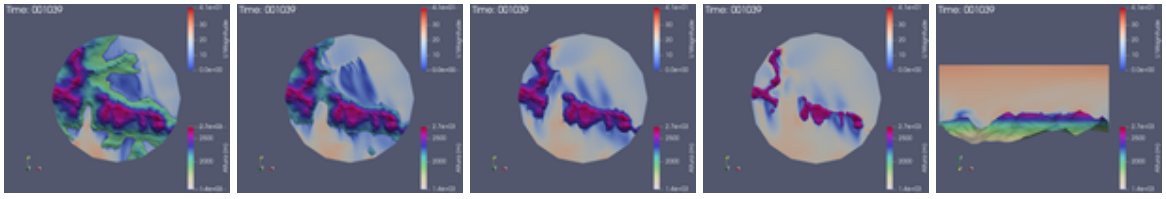
1900 m 2100 m 2300 m 2500 m Lateral Y01



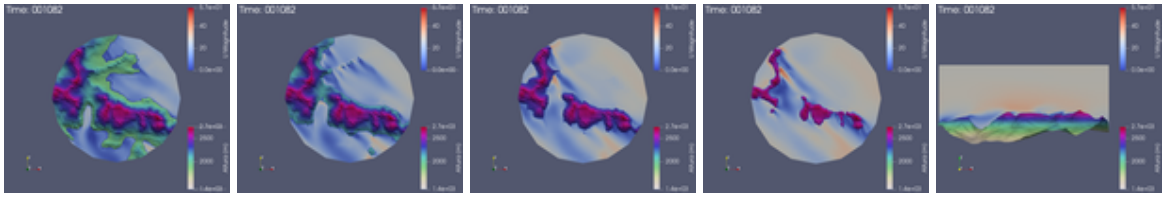
1900 m 2100 m 2300 m 2500 m Lateral Y01



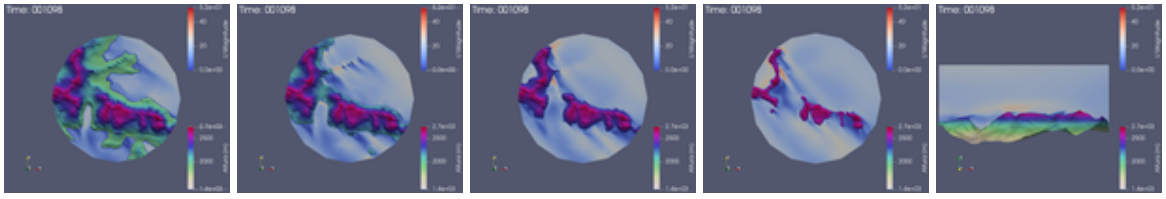
1900 m 2100 m 2300 m 2500 m Lateral Y01



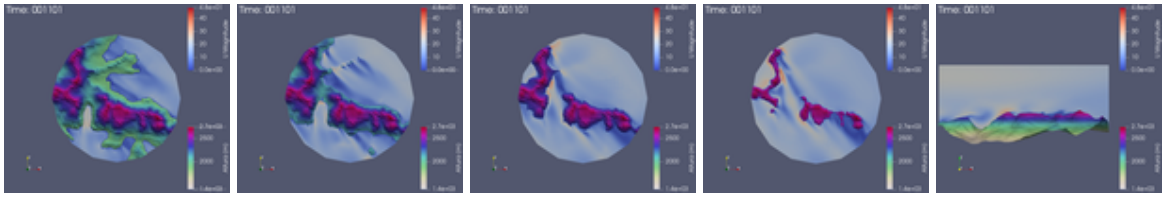
1900 m 2100 m 2300 m 2500 m Lateral Y01



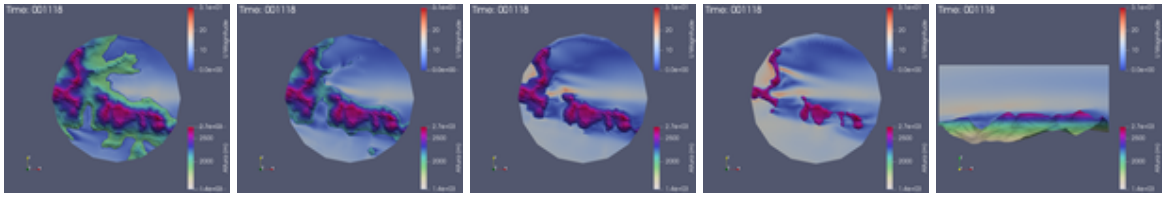
1900 m 2100 m 2300 m 2500 m Lateral Y01



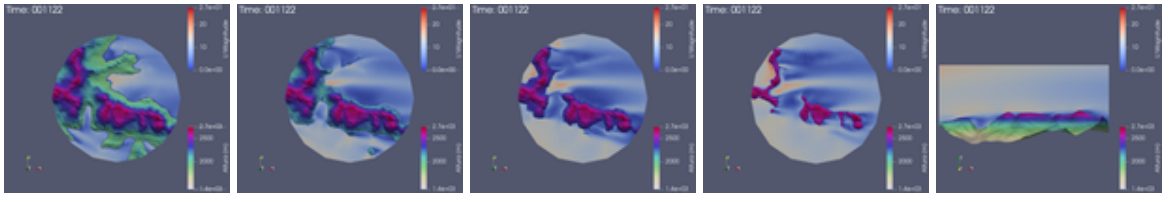
1900 m 2100 m 2300 m 2500 m Lateral Y01



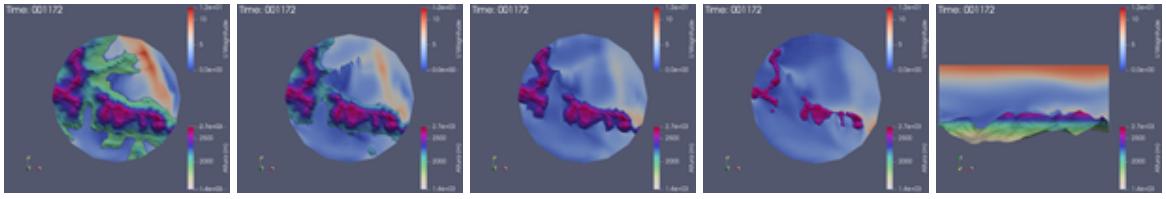
1900 m 2100 m 2300 m 2500 m Lateral Y01



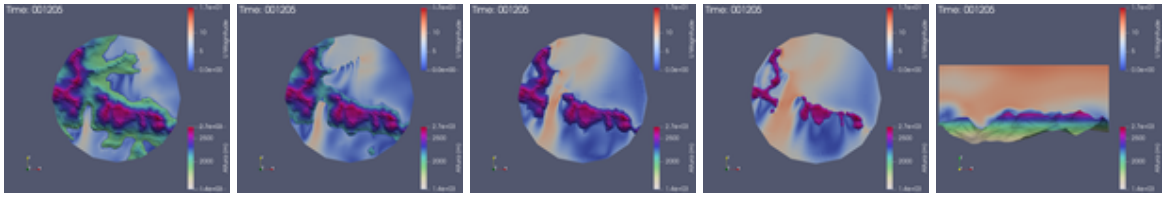
1900 m 2100 m 2300 m 2500 m Lateral Y01



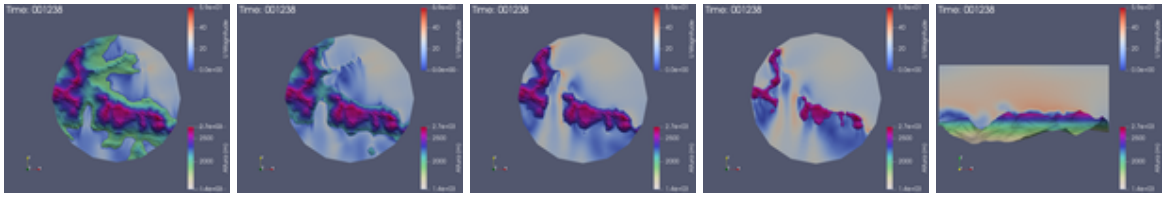
1900 m 2100 m 2300 m 2500 m Lateral Y01



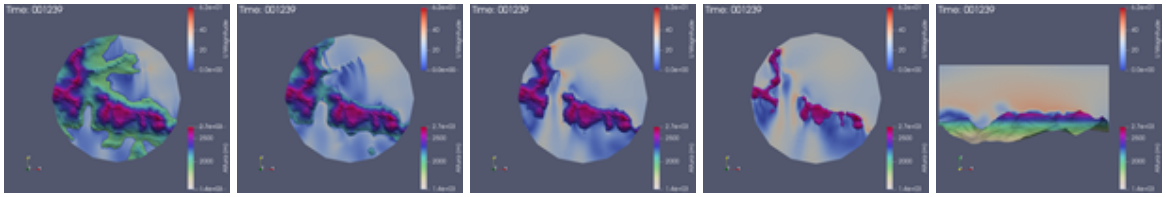
1900 m 2100 m 2300 m 2500 m Lateral Y01



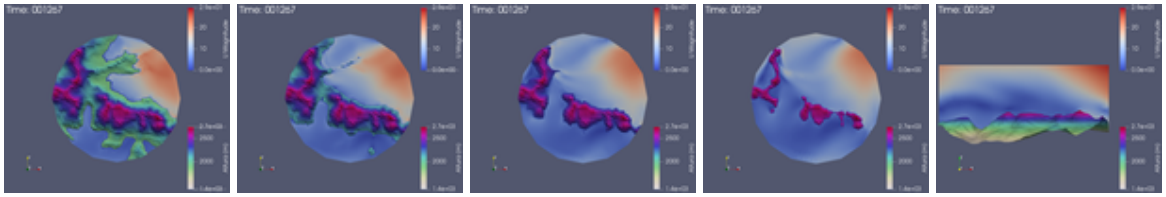
1900 m 2100 m 2300 m 2500 m Lateral Y01



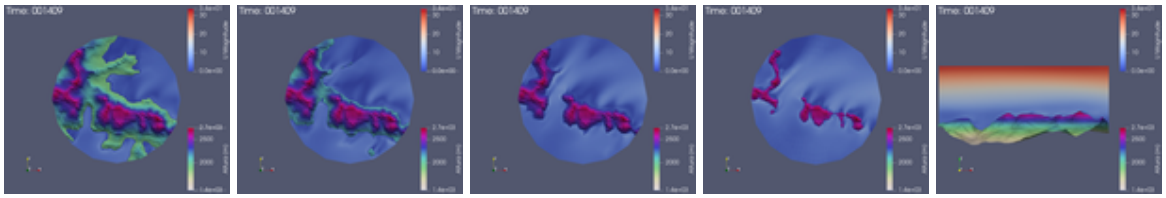
1900 m 2100 m 2300 m 2500 m Lateral Y01



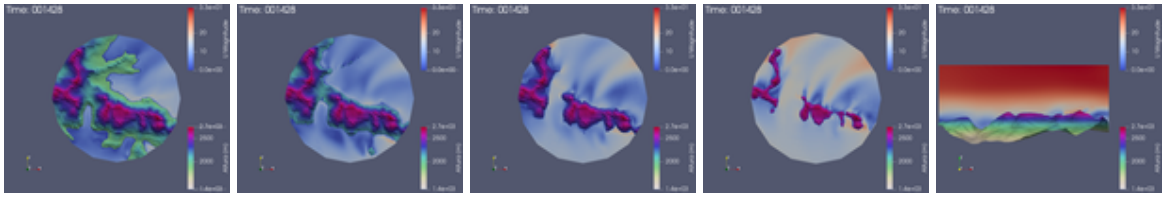
1900 m 2100 m 2300 m 2500 m Lateral Y01



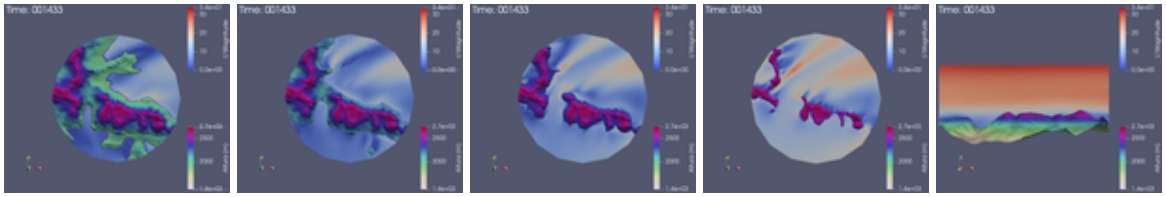
1900 m 2100 m 2300 m 2500 m Lateral Y01



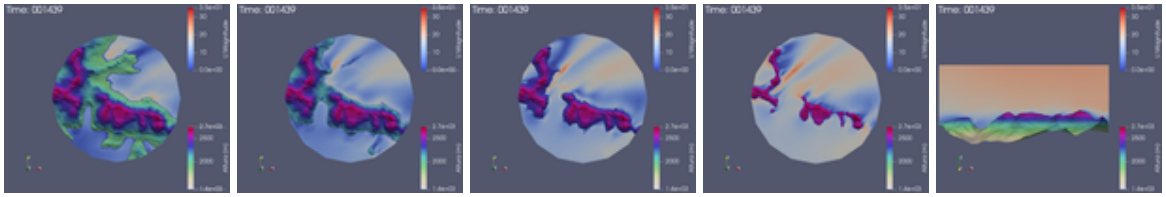
1900 m 2100 m 2300 m 2500 m Lateral Y01



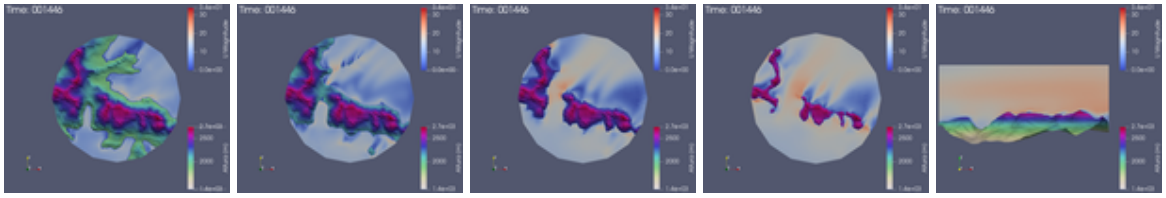
1900 m 2100 m 2300 m 2500 m Lateral Y01



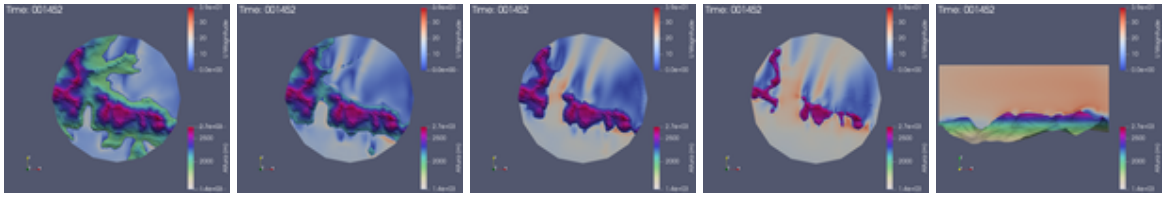
1900 m 2100 m 2300 m 2500 m Lateral Y01



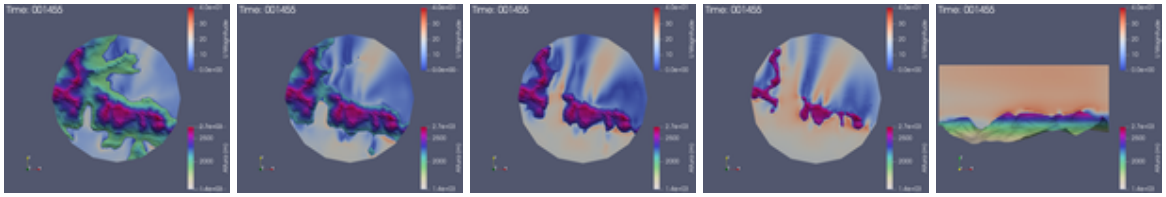
1900 m 2100 m 2300 m 2500 m Lateral Y01



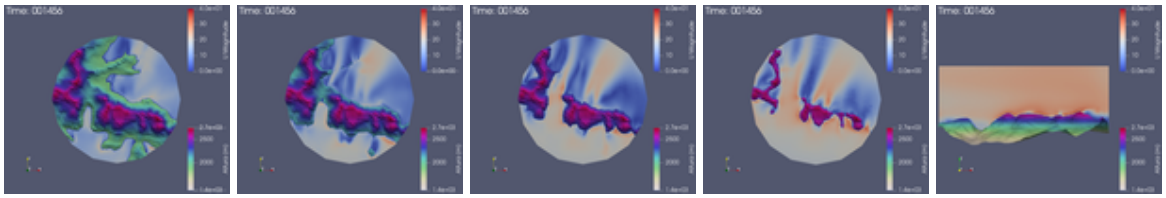
1900 m 2100 m 2300 m 2500 m Lateral Y01



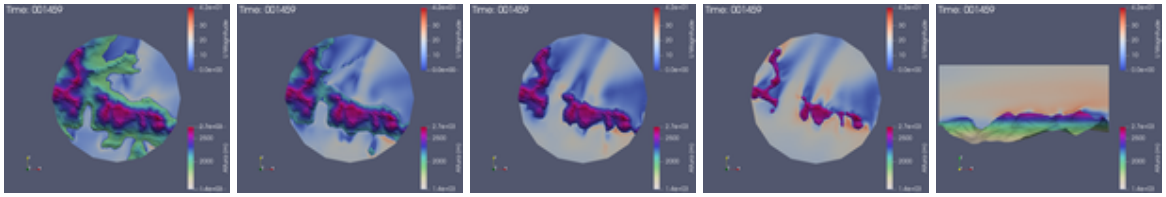
1900 m 2100 m 2300 m 2500 m Lateral Y01



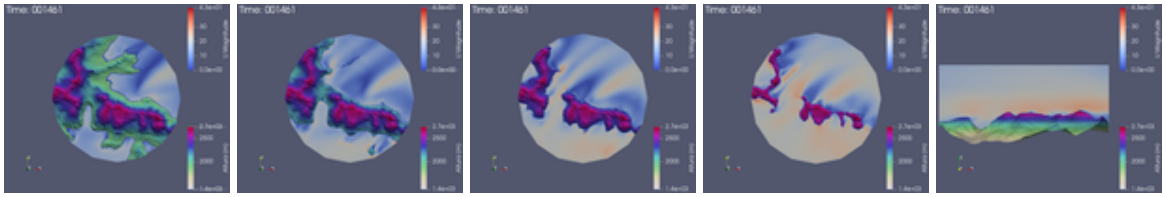
1900 m 2100 m 2300 m 2500 m Lateral Y01



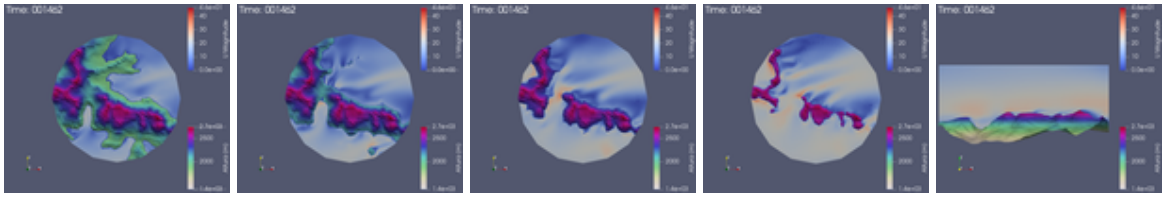
1900 m 2100 m 2300 m 2500 m Lateral Y01



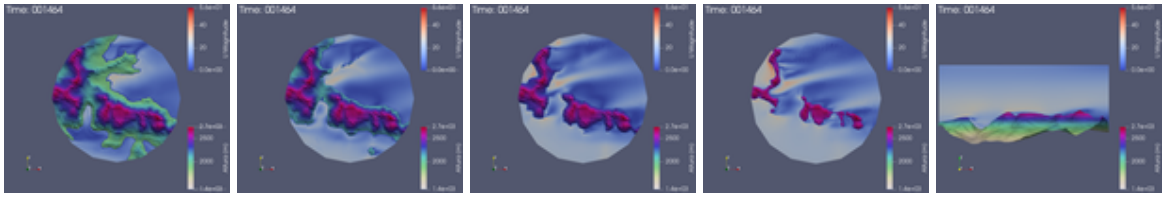
1900 m 2100 m 2300 m 2500 m Lateral Y01



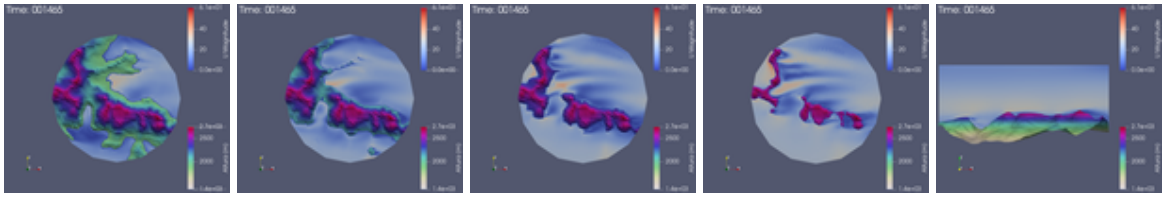
1900 m 2100 m 2300 m 2500 m Lateral Y01



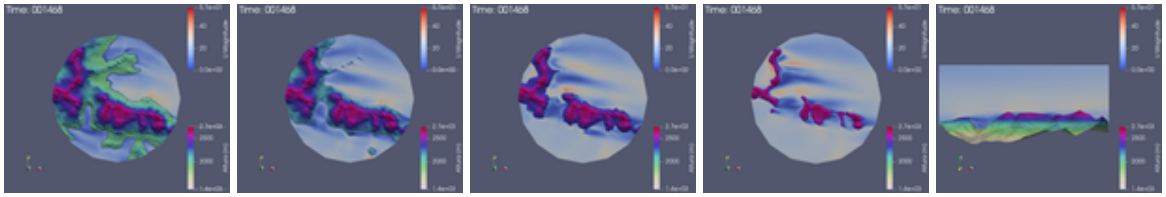
1900 m 2100 m 2300 m 2500 m Lateral Y01



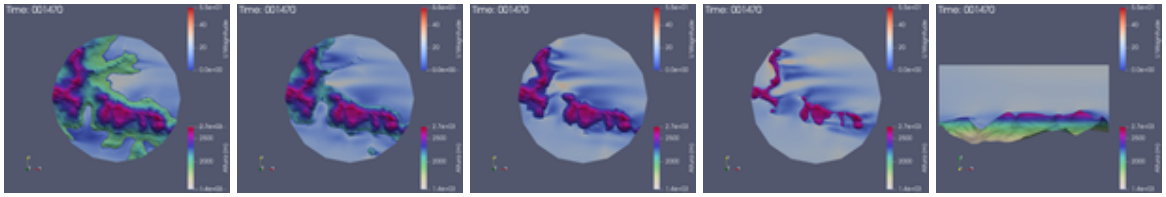
1900 m 2100 m 2300 m 2500 m Lateral Y01



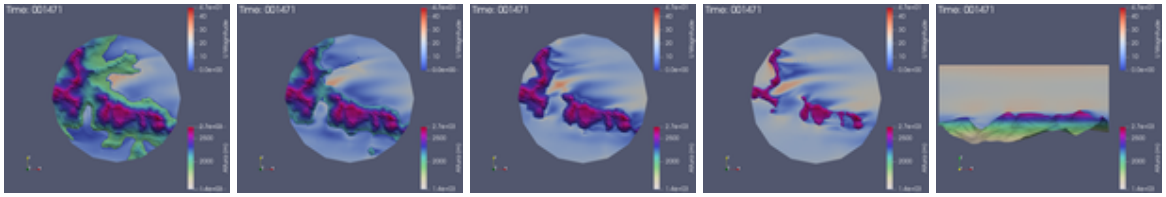
1900 m 2100 m 2300 m 2500 m Lateral Y01



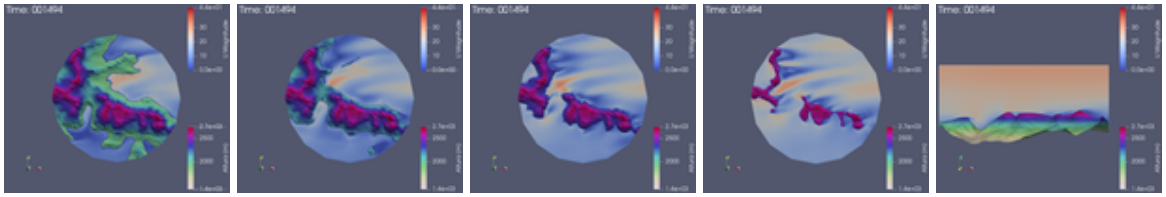
1900 m 2100 m 2300 m 2500 m Lateral Y01



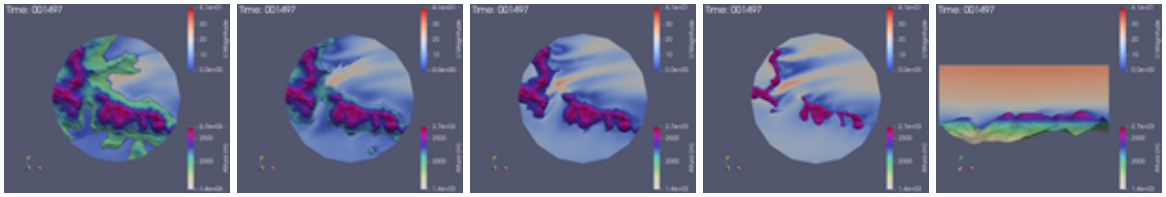
1900 m 2100 m 2300 m 2500 m Lateral Y01



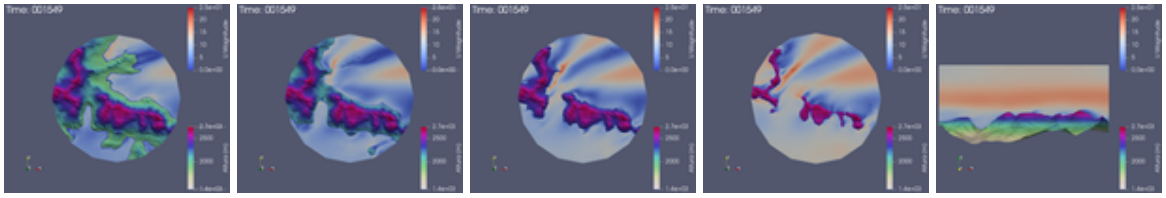
1900 m 2100 m 2300 m 2500 m Lateral Y01



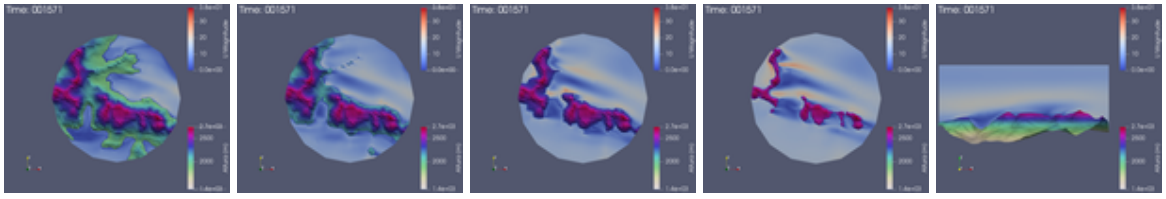
1900 m 2100 m 2300 m 2500 m Lateral Y01



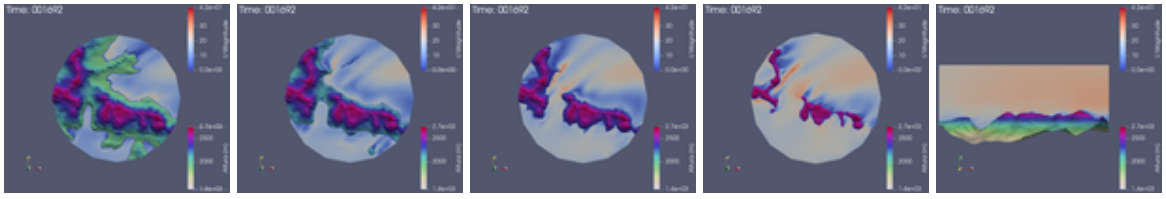
1900 m 2100 m 2300 m 2500 m Lateral Y01



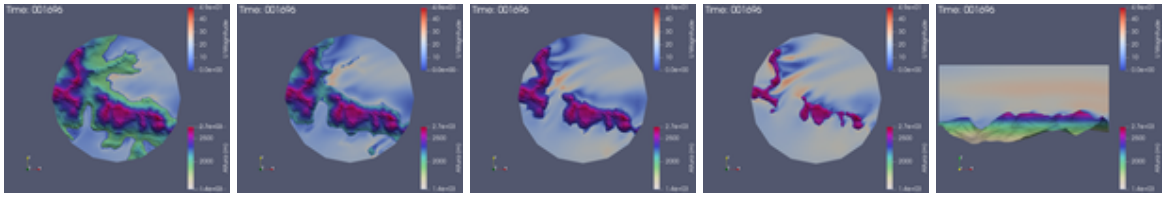
1900 m 2100 m 2300 m 2500 m Lateral Y01



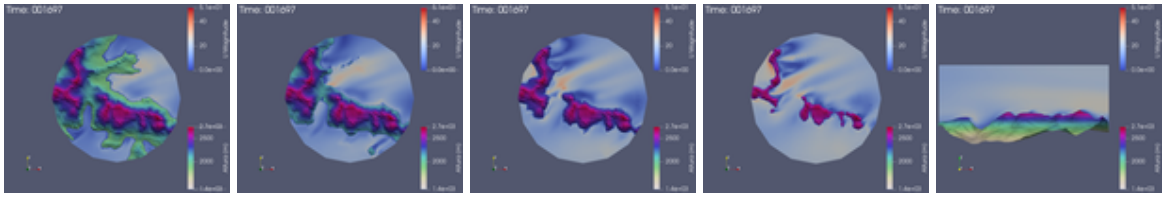
1900 m 2100 m 2300 m 2500 m Lateral Y01



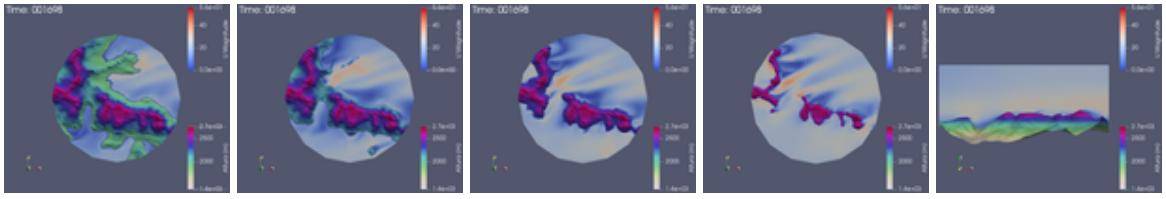
1900 m 2100 m 2300 m 2500 m Lateral Y01



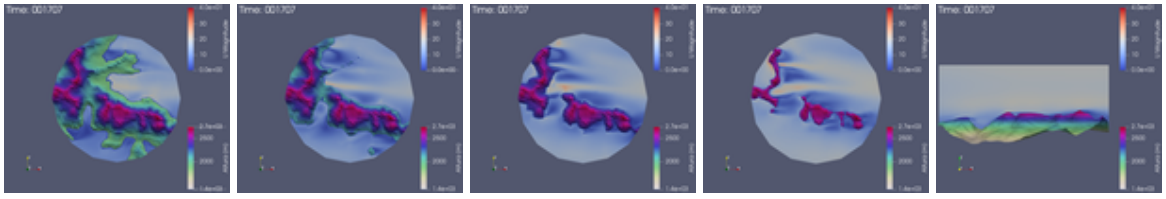
1900 m 2100 m 2300 m 2500 m Lateral Y01



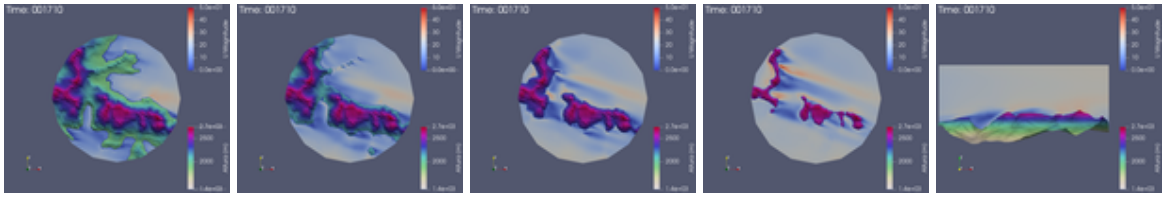
1900 m 2100 m 2300 m 2500 m Lateral Y01



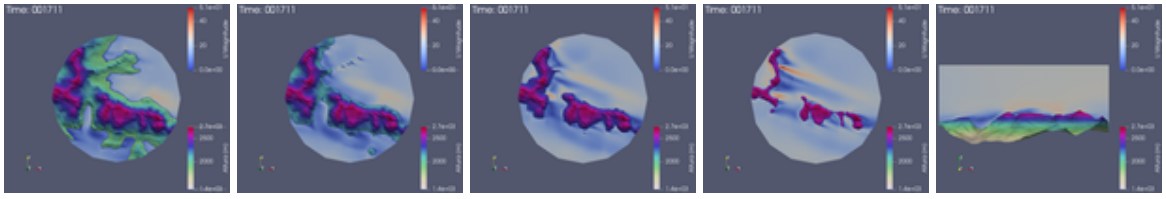
1900 m 2100 m 2300 m 2500 m Lateral Y01



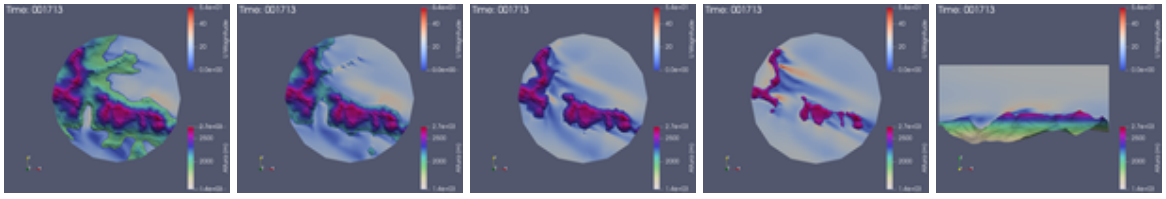
1900 m 2100 m 2300 m 2500 m Lateral Y01



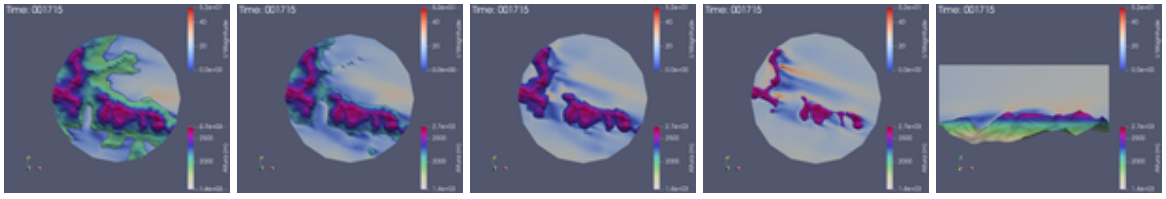
1900 m 2100 m 2300 m 2500 m Lateral Y01



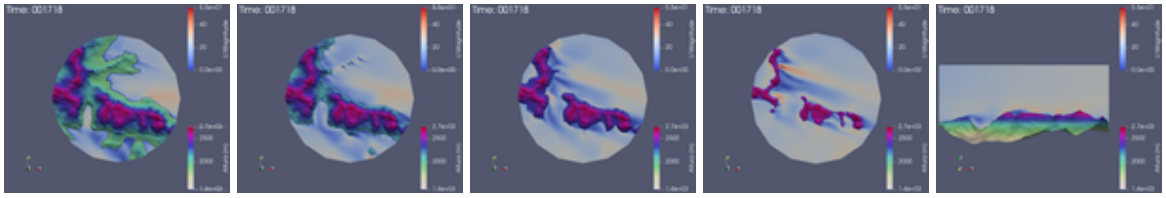
1900 m 2100 m 2300 m 2500 m Lateral Y01



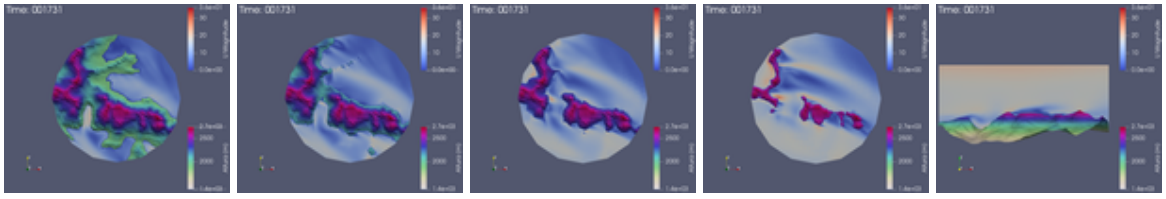
1900 m 2100 m 2300 m 2500 m Lateral Y01



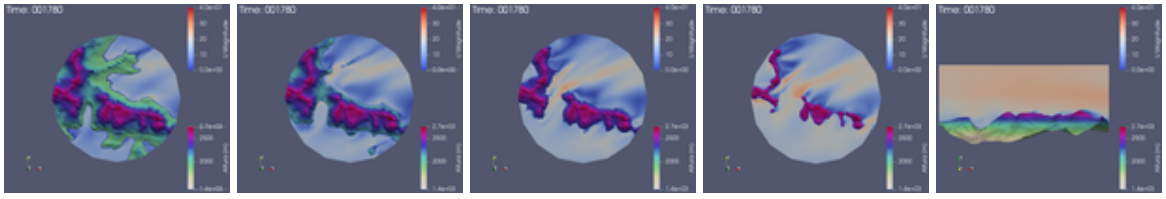
1900 m 2100 m 2300 m 2500 m Lateral Y01



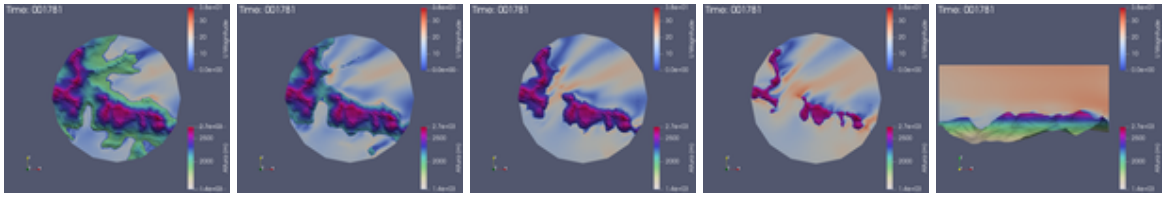
1900 m 2100 m 2300 m 2500 m Lateral Y01



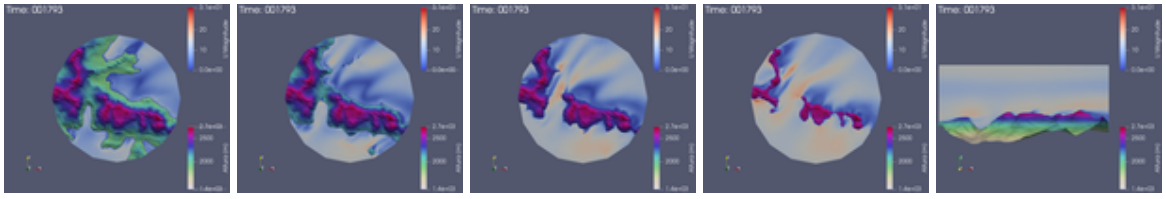
1900 m 2100 m 2300 m 2500 m Lateral Y01



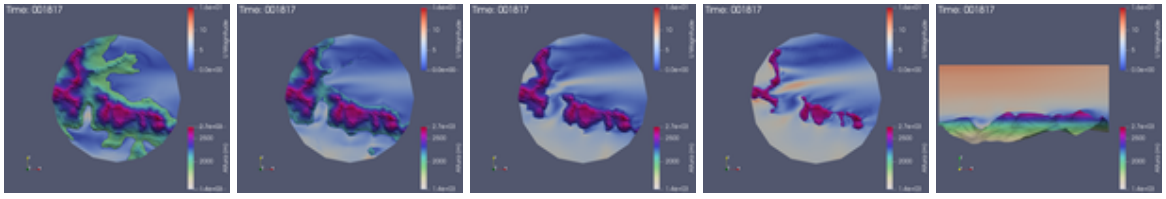
1900 m 2100 m 2300 m 2500 m Lateral Y01



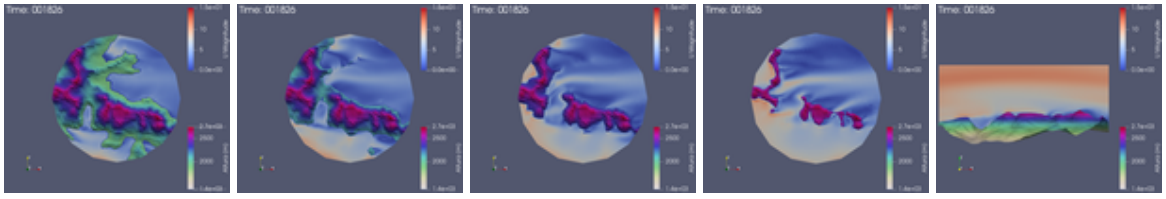
1900 m 2100 m 2300 m 2500 m Lateral Y01



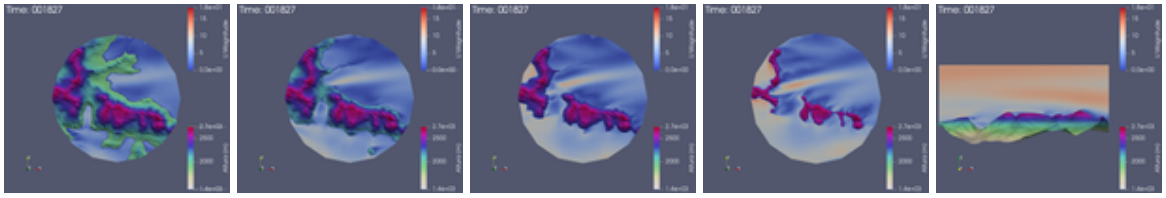
1900 m 2100 m 2300 m 2500 m Lateral Y01



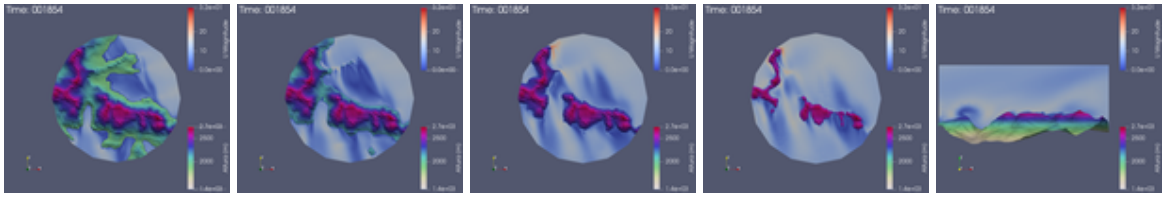
1900 m 2100 m 2300 m 2500 m Lateral Y01



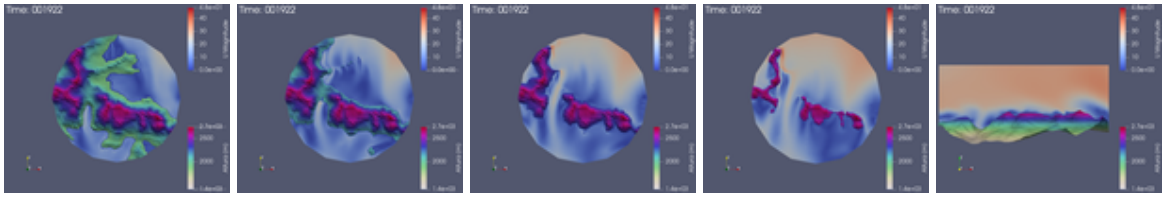
1900 m 2100 m 2300 m 2500 m Lateral Y01



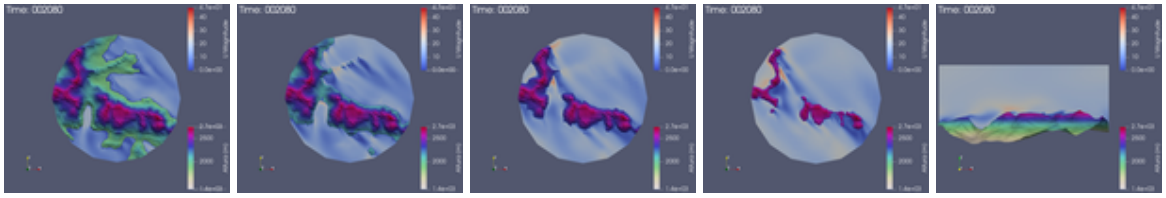
1900 m 2100 m 2300 m 2500 m Lateral Y01



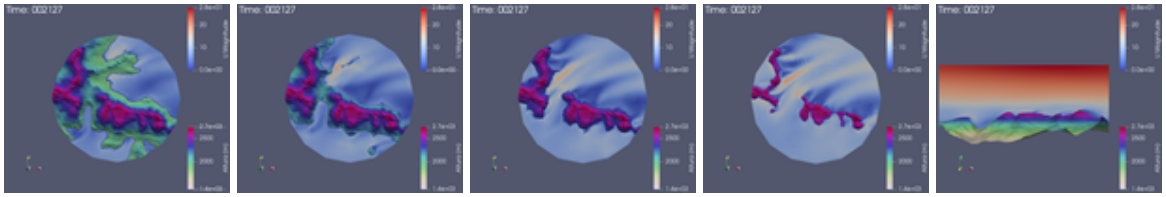
1900 m 2100 m 2300 m 2500 m Lateral Y01



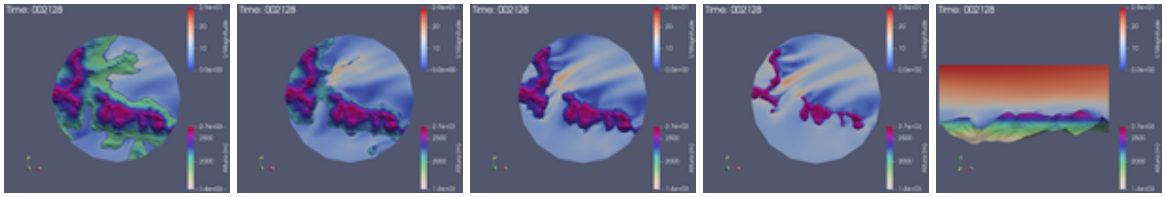
1900 m 2100 m 2300 m 2500 m Lateral Y01



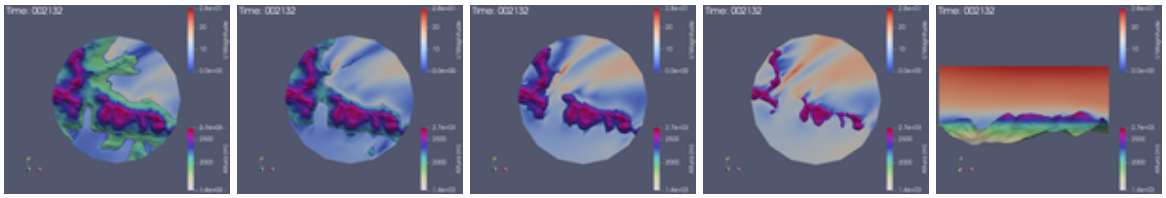
1900 m 2100 m 2300 m 2500 m Lateral Y01



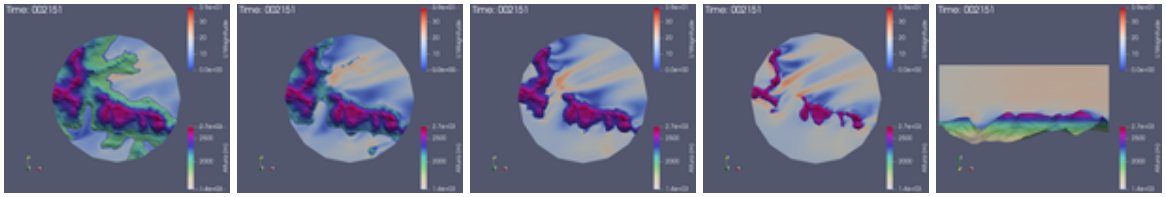
1900 m 2100 m 2300 m 2500 m Lateral Y01



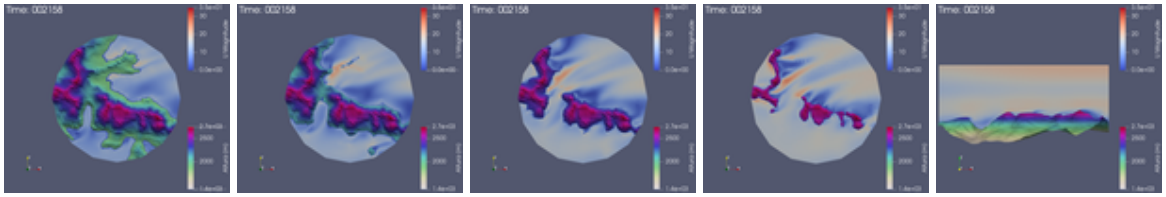
1900 m 2100 m 2300 m 2500 m Lateral Y01



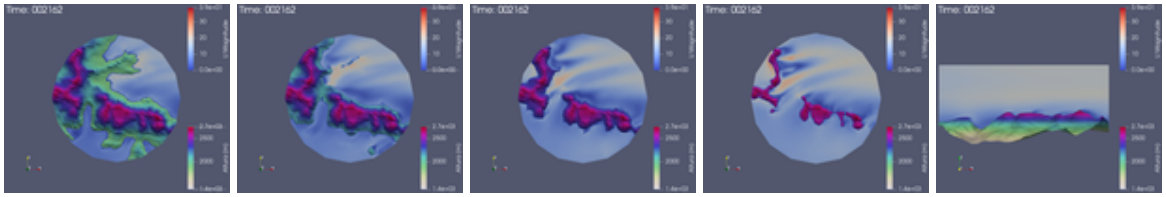
1900 m 2100 m 2300 m 2500 m Lateral Y01



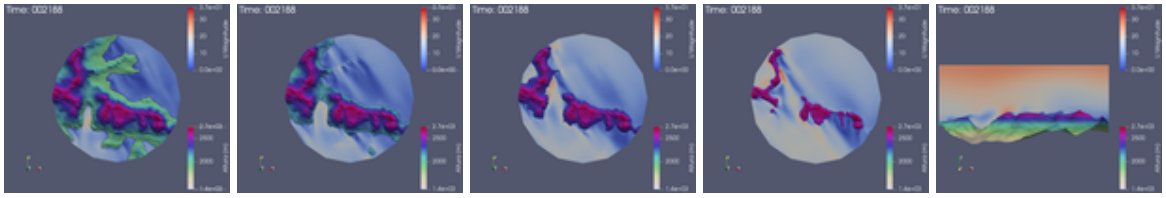
1900 m 2100 m 2300 m 2500 m Lateral Y01



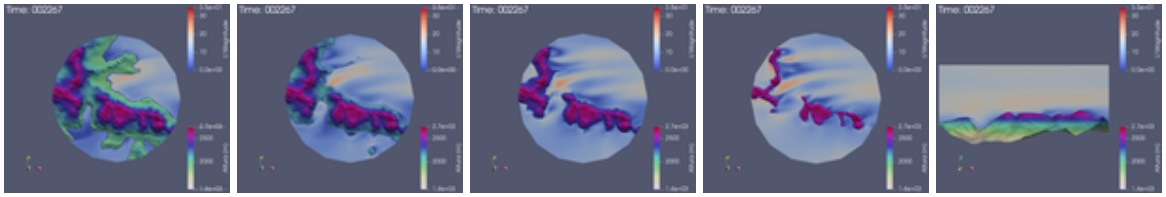
1900 m 2100 m 2300 m 2500 m Lateral Y01



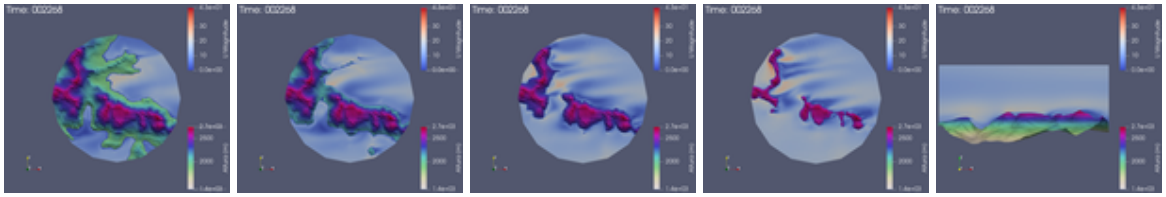
1900 m 2100 m 2300 m 2500 m Lateral Y01



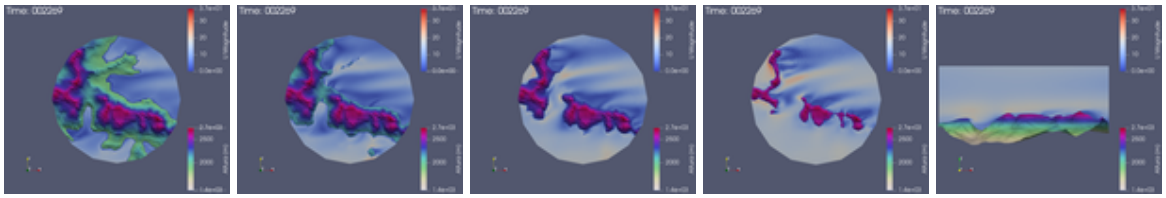
1900 m 2100 m 2300 m 2500 m Lateral Y01



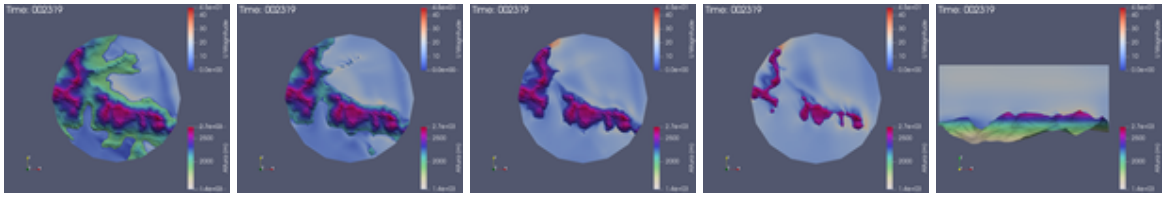
1900 m 2100 m 2300 m 2500 m Lateral Y01



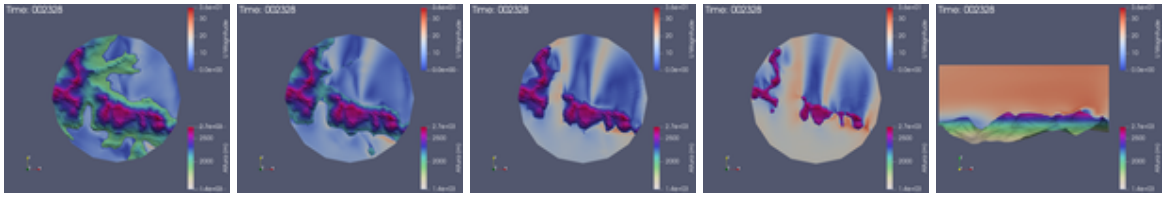
1900 m 2100 m 2300 m 2500 m Lateral Y01



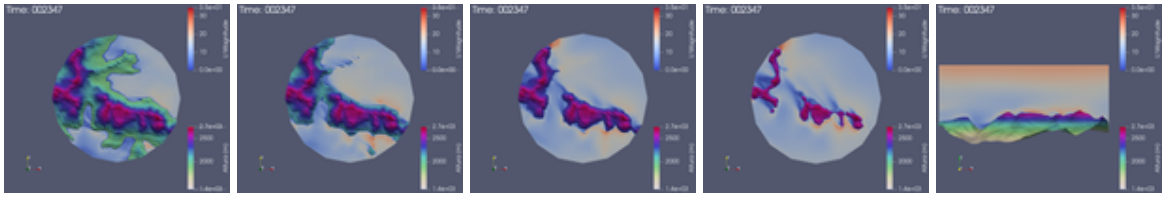
1900 m 2100 m 2300 m 2500 m Lateral Y01



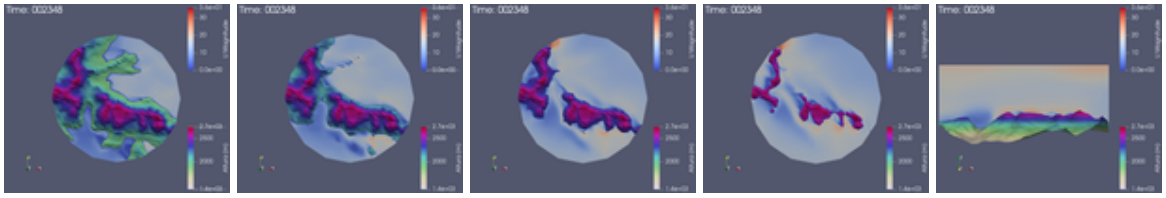
1900 m 2100 m 2300 m 2500 m Lateral Y01



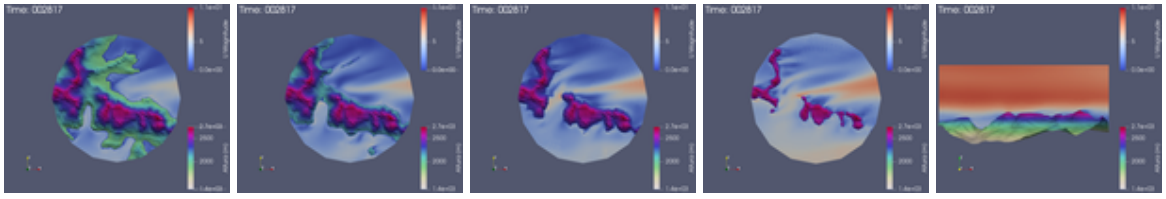
1900 m 2100 m 2300 m 2500 m Lateral Y01



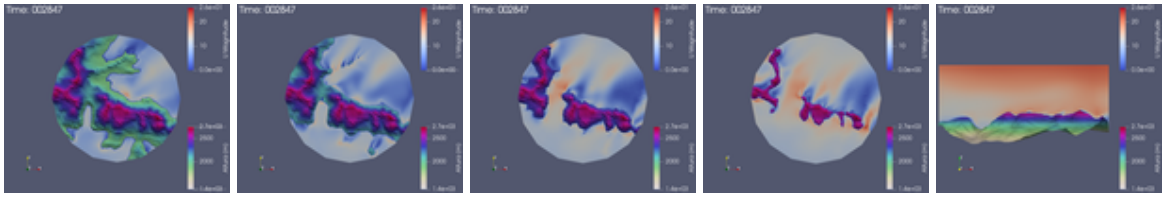
1900 m 2100 m 2300 m 2500 m Lateral Y01



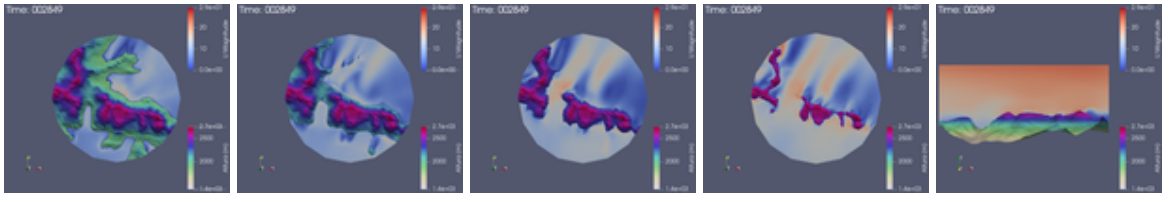
1900 m 2100 m 2300 m 2500 m Lateral Y01



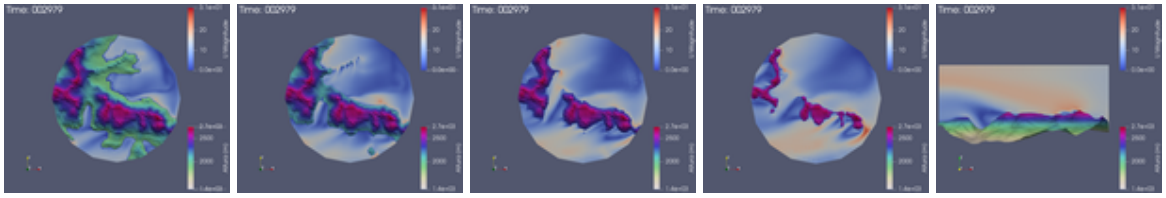
1900 m 2100 m 2300 m 2500 m Lateral Y01



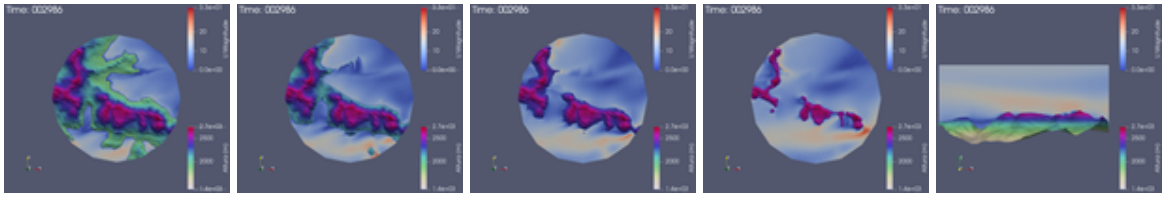
1900 m 2100 m 2300 m 2500 m Lateral Y01



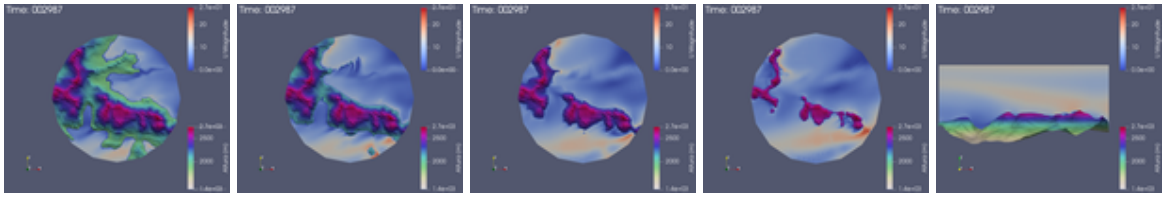
1900 m 2100 m 2300 m 2500 m Lateral Y01



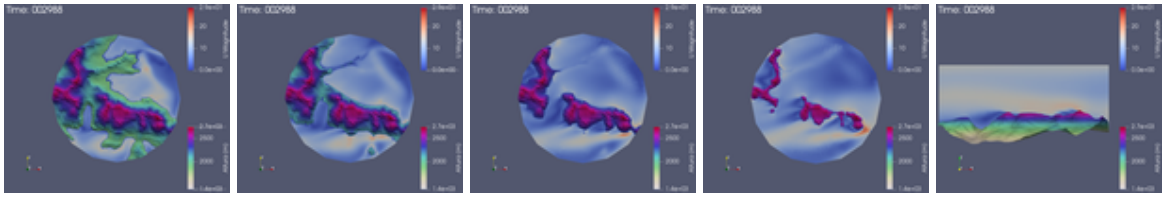
1900 m 2100 m 2300 m 2500 m Lateral Y01



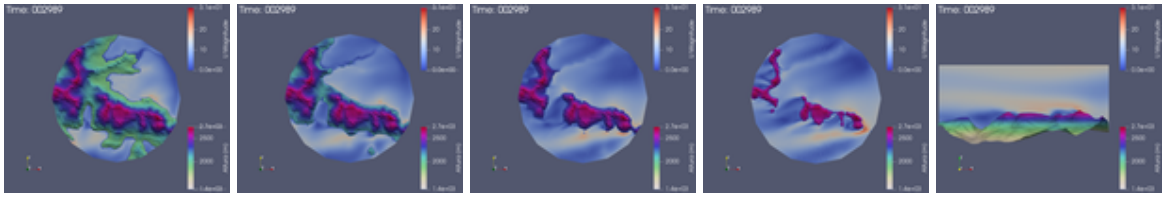
1900 m 2100 m 2300 m 2500 m Lateral Y01



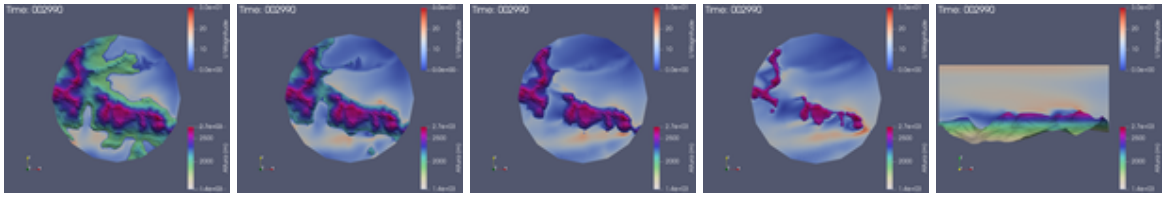
1900 m 2100 m 2300 m 2500 m Lateral Y01



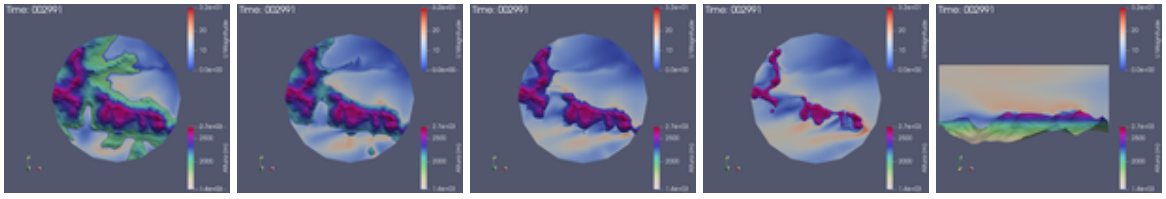
1900 m 2100 m 2300 m 2500 m Lateral Y01



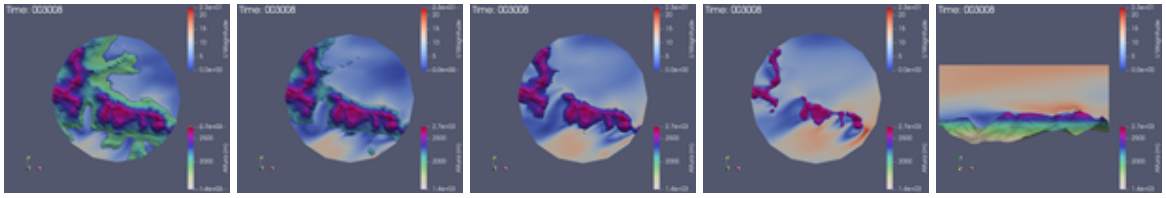
1900 m 2100 m 2300 m 2500 m Lateral Y01



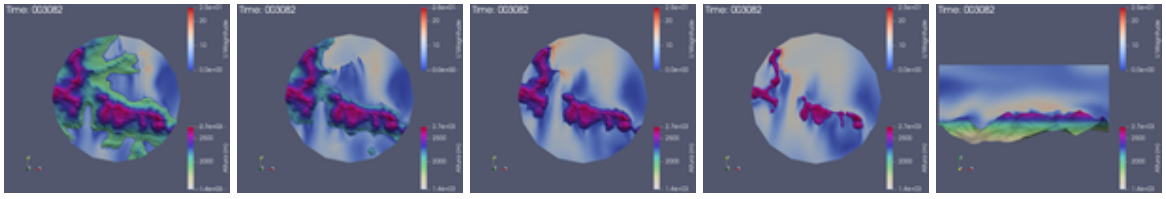
1900 m 2100 m 2300 m 2500 m Lateral Y01



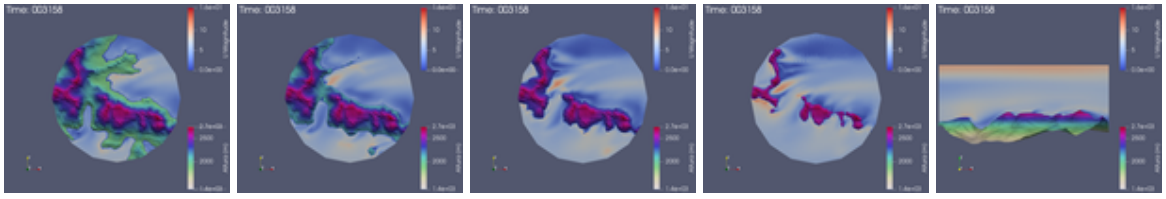
1900 m 2100 m 2300 m 2500 m Lateral Y01



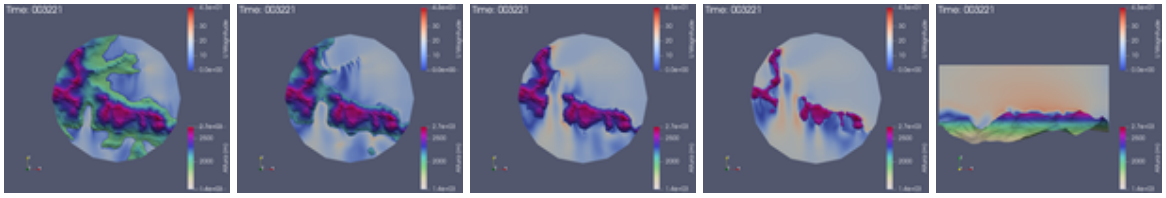
1900 m 2100 m 2300 m 2500 m Lateral Y01



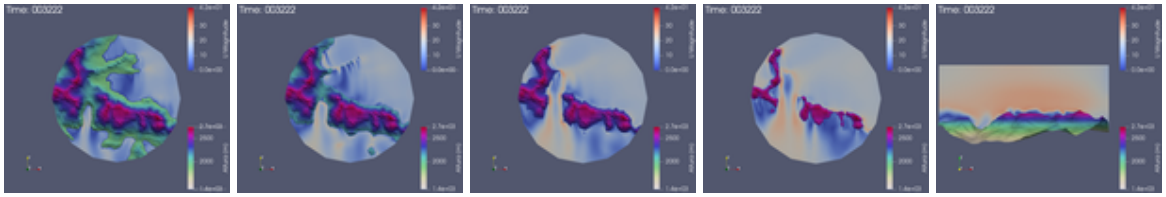
1900 m 2100 m 2300 m 2500 m Lateral Y01



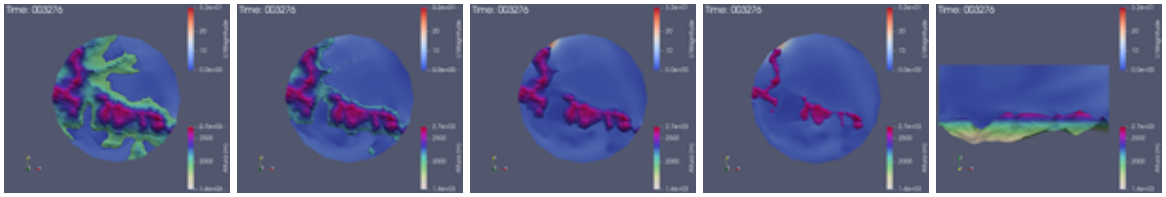
1900 m 2100 m 2300 m 2500 m Lateral Y01



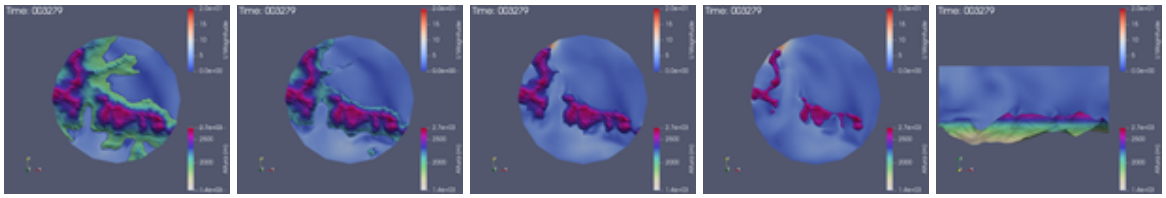
1900 m 2100 m 2300 m 2500 m Lateral Y01



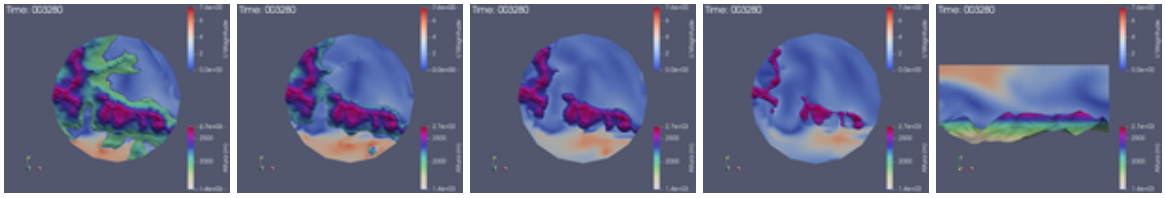
1900 m 2100 m 2300 m 2500 m Lateral Y01



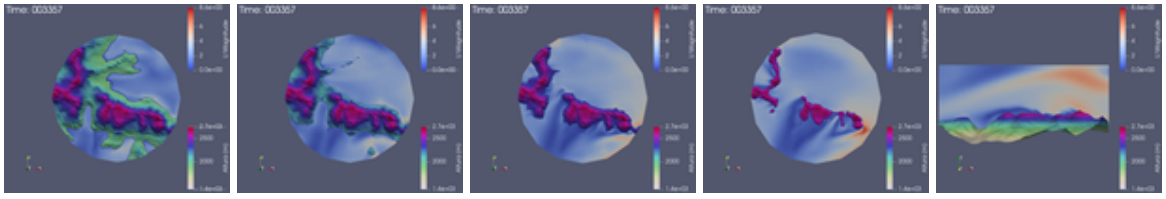
1900 m 2100 m 2300 m 2500 m Lateral Y01



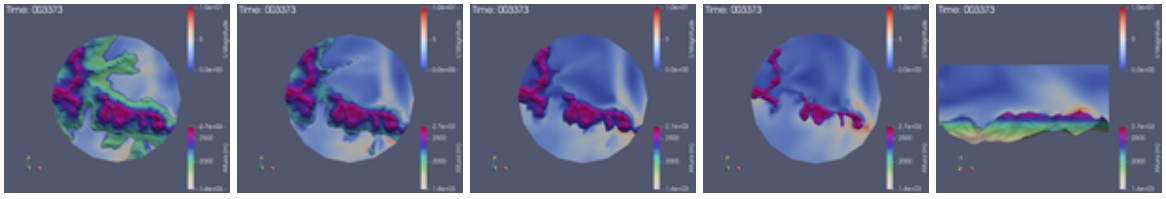
1900 m 2100 m 2300 m 2500 m Lateral Y01



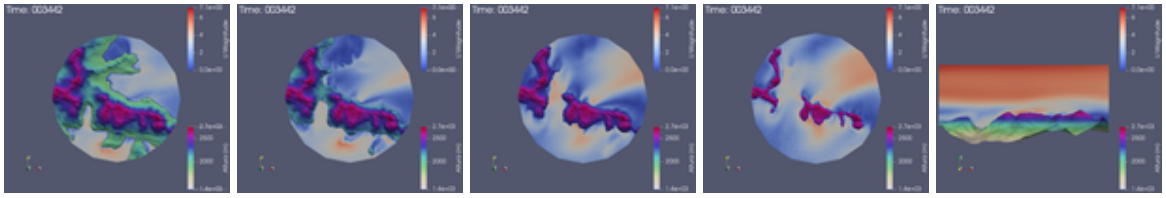
1900 m 2100 m 2300 m 2500 m Lateral Y01



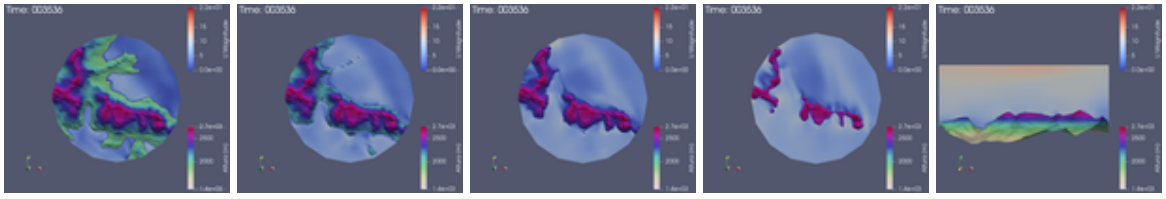
1900 m 2100 m 2300 m 2500 m Lateral Y01



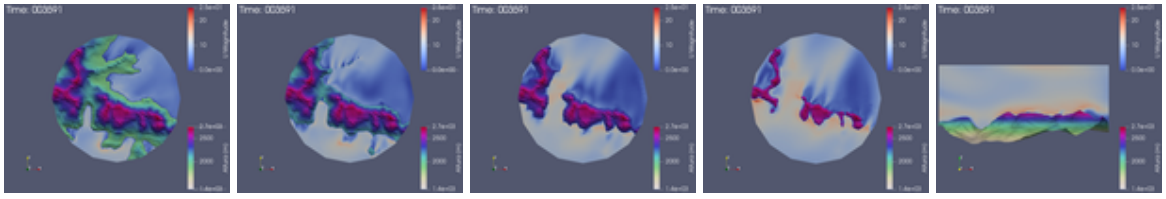
1900 m 2100 m 2300 m 2500 m Lateral Y01



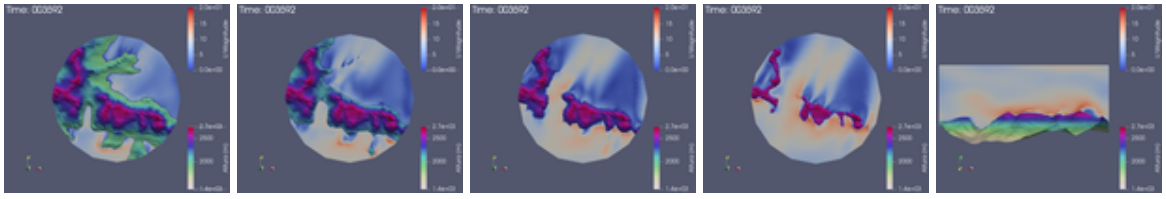
1900 m 2100 m 2300 m 2500 m Lateral Y01



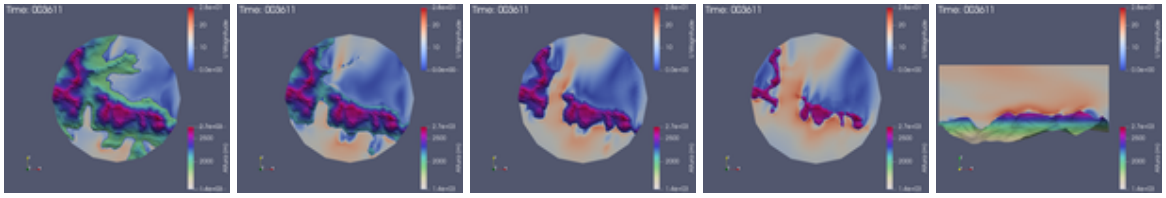
1900 m 2100 m 2300 m 2500 m Lateral Y01



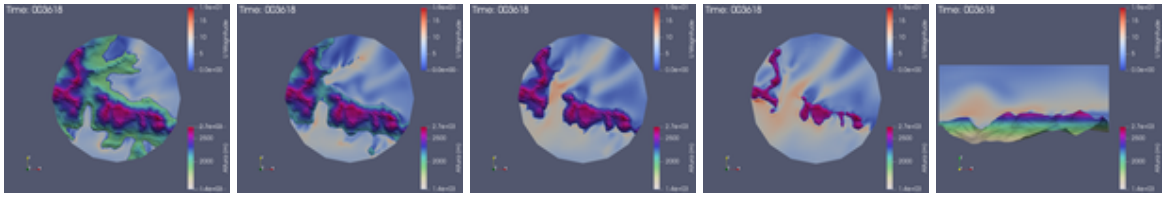
1900 m 2100 m 2300 m 2500 m Lateral Y01



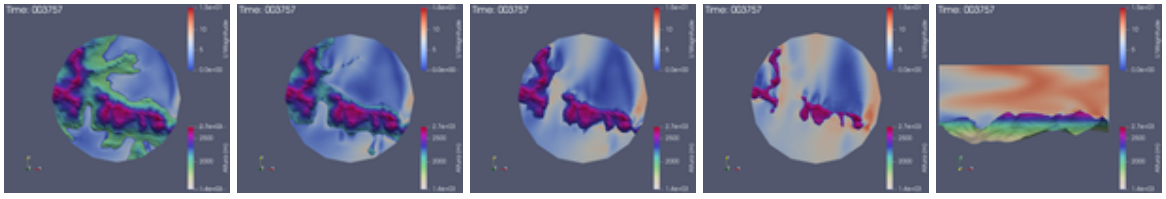
1900 m 2100 m 2300 m 2500 m Lateral Y01



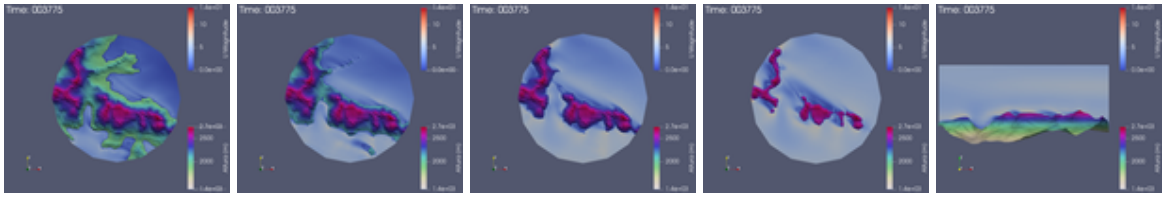
1900 m 2100 m 2300 m 2500 m Lateral Y01



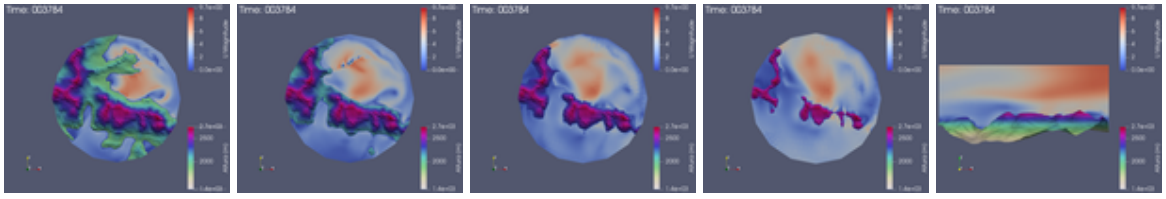
1900 m 2100 m 2300 m 2500 m Lateral Y01



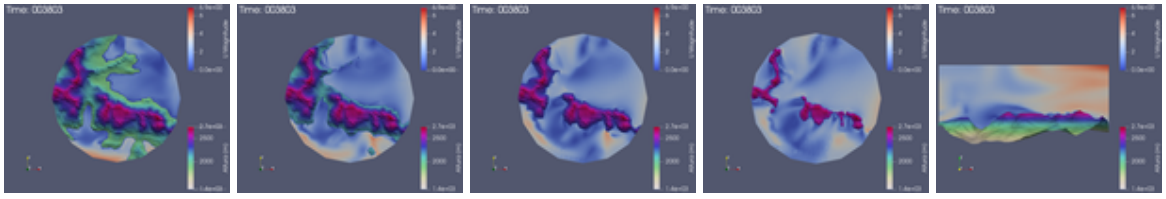
1900 m 2100 m 2300 m 2500 m Lateral Y01



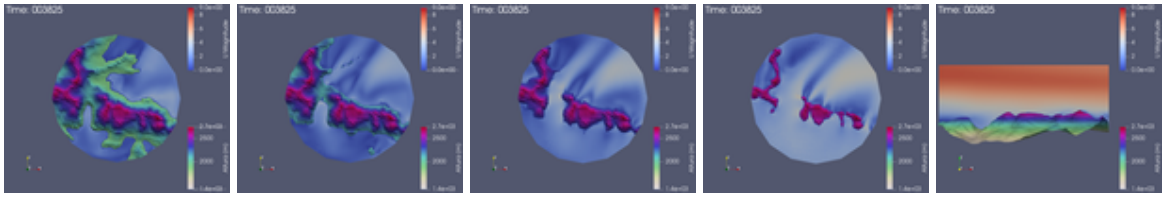
1900 m 2100 m 2300 m 2500 m Lateral Y01



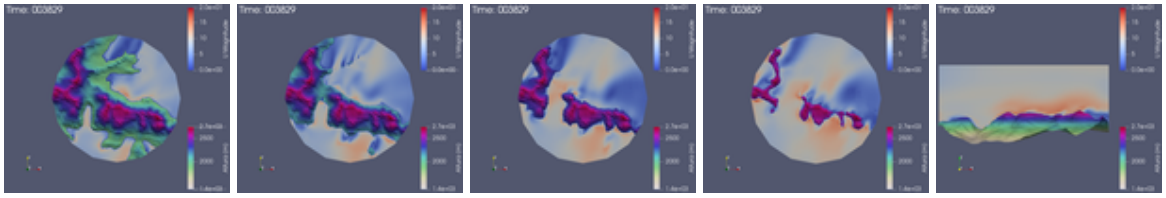
1900 m 2100 m 2300 m 2500 m Lateral Y01



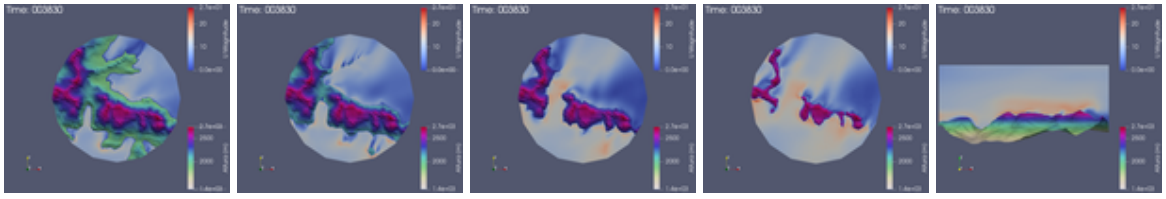
1900 m 2100 m 2300 m 2500 m Lateral Y01



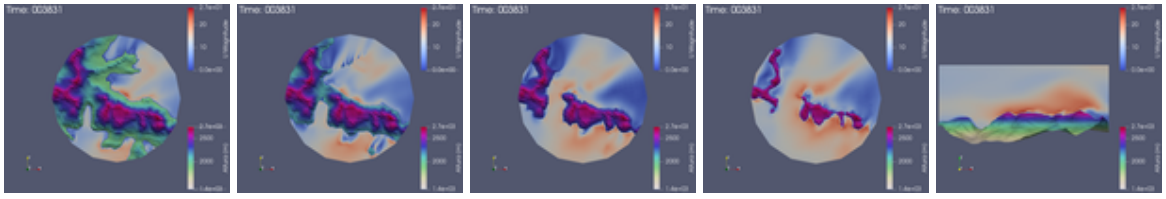
1900 m 2100 m 2300 m 2500 m Lateral Y01



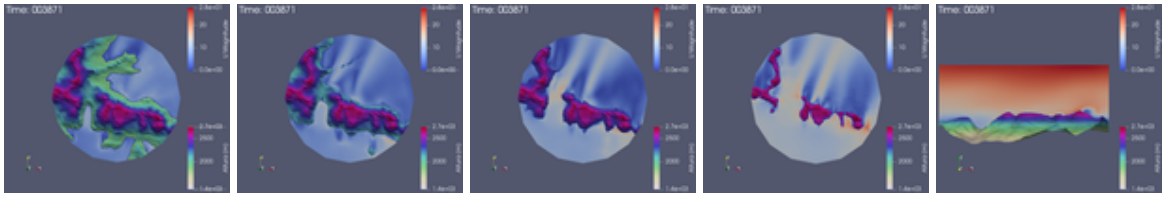
1900 m 2100 m 2300 m 2500 m Lateral Y01



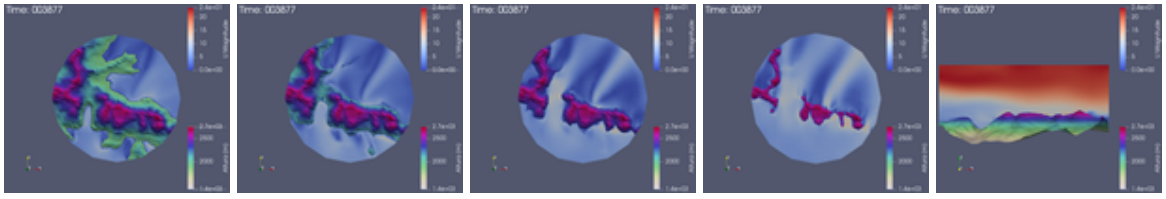
1900 m 2100 m 2300 m 2500 m Lateral Y01



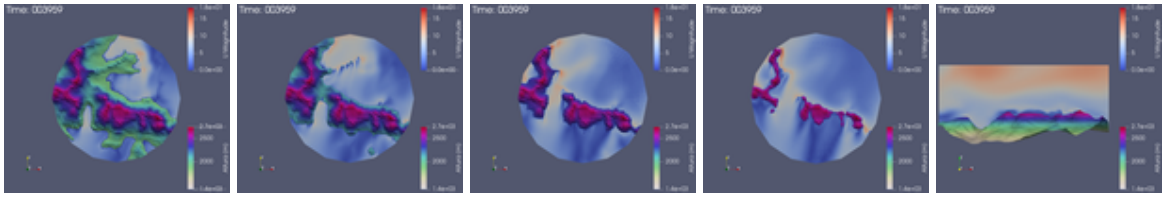
1900 m 2100 m 2300 m 2500 m Lateral Y01



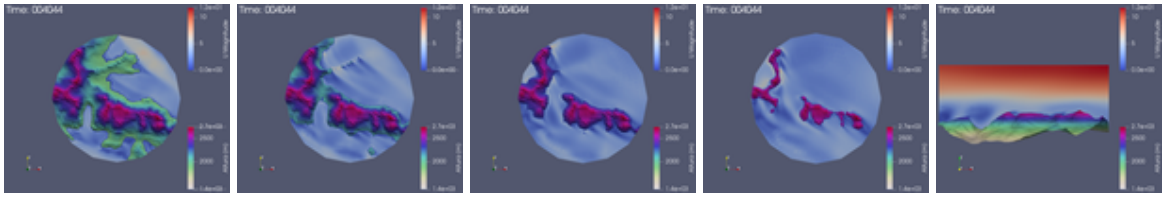
1900 m 2100 m 2300 m 2500 m Lateral Y01



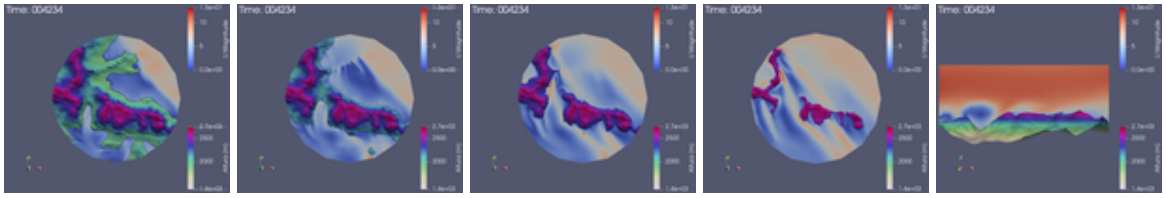
1900 m 2100 m 2300 m 2500 m Lateral Y01



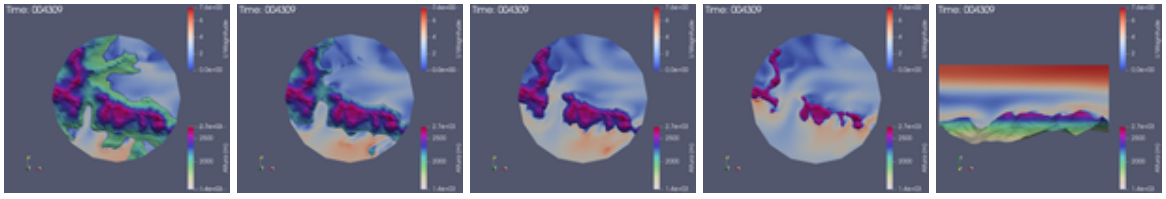
1900 m 2100 m 2300 m 2500 m Lateral Y01



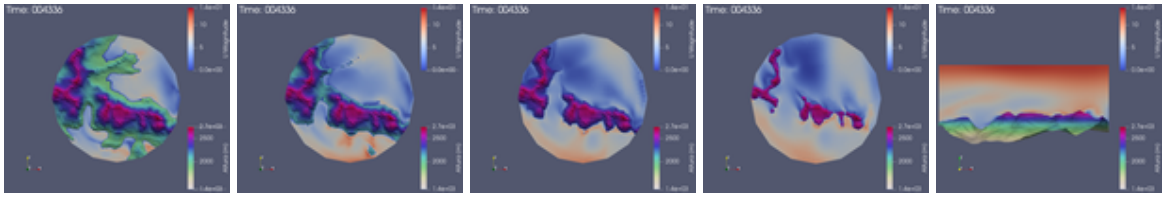
1900 m 2100 m 2300 m 2500 m Lateral Y01



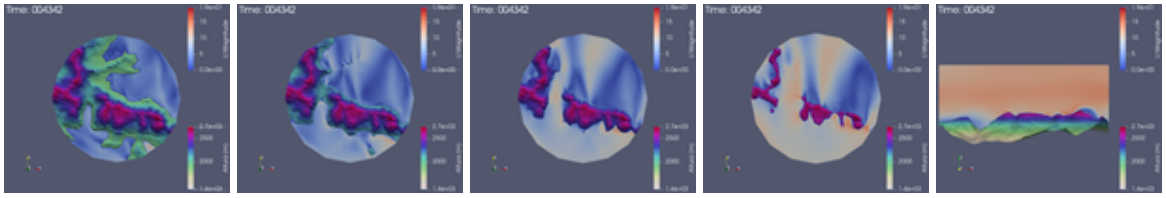
1900 m 2100 m 2300 m 2500 m Lateral Y01



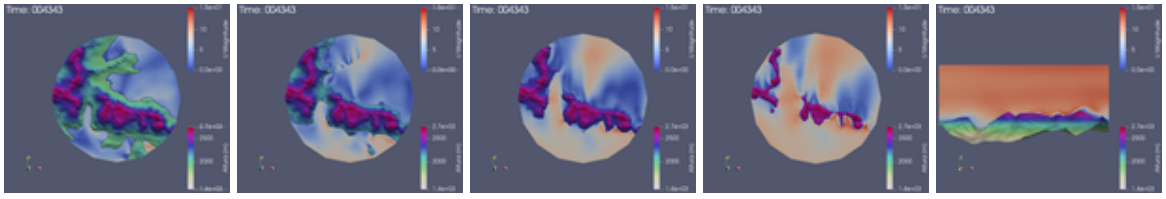
1900 m 2100 m 2300 m 2500 m Lateral Y01



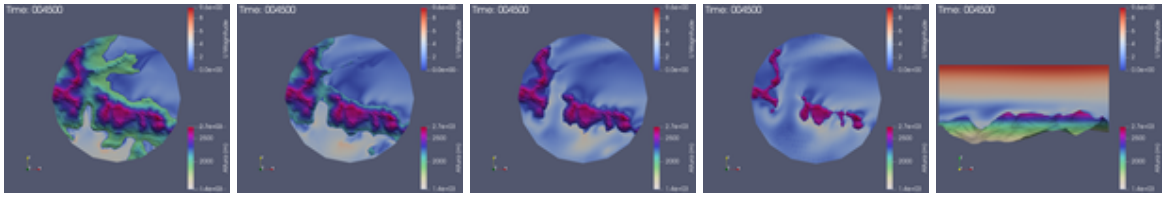
1900 m 2100 m 2300 m 2500 m Lateral Y01



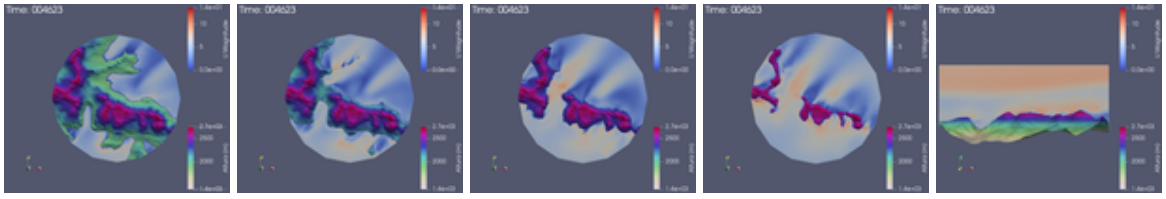
1900 m 2100 m 2300 m 2500 m Lateral Y01



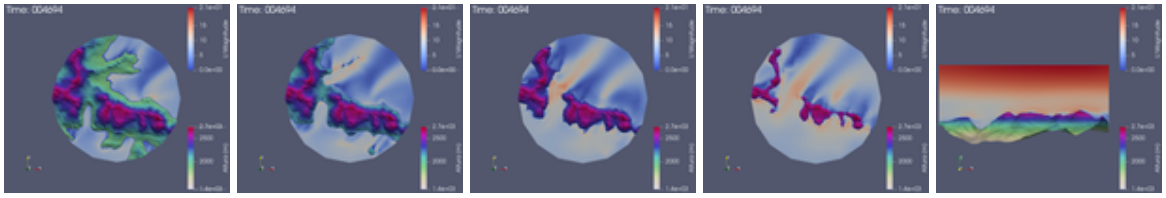
1900 m 2100 m 2300 m 2500 m Lateral Y01



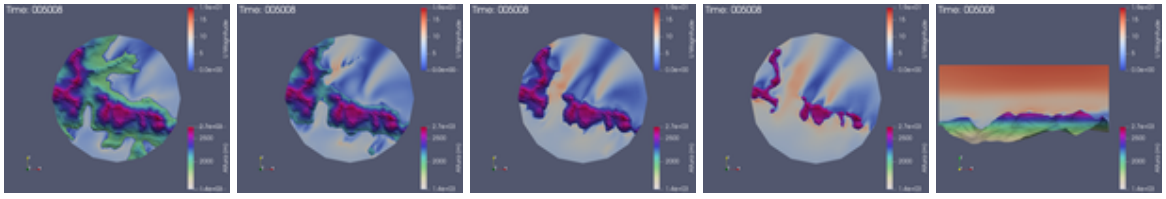
1900 m 2100 m 2300 m 2500 m Lateral Y01



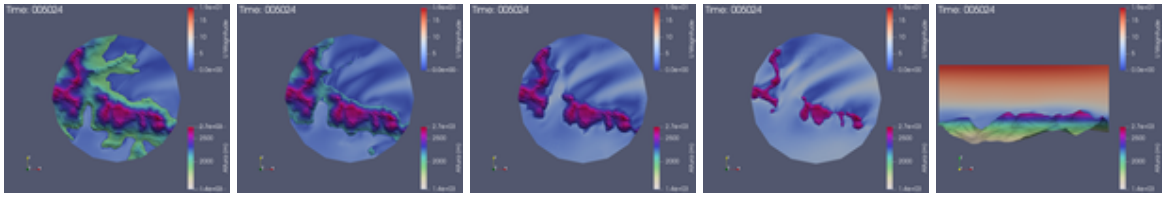
1900 m 2100 m 2300 m 2500 m Lateral Y01



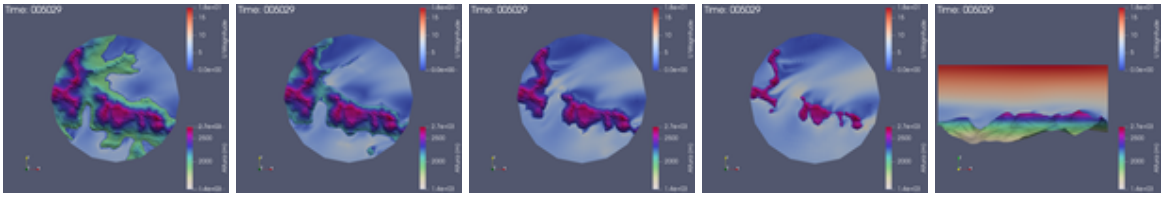
1900 m 2100 m 2300 m 2500 m Lateral Y01



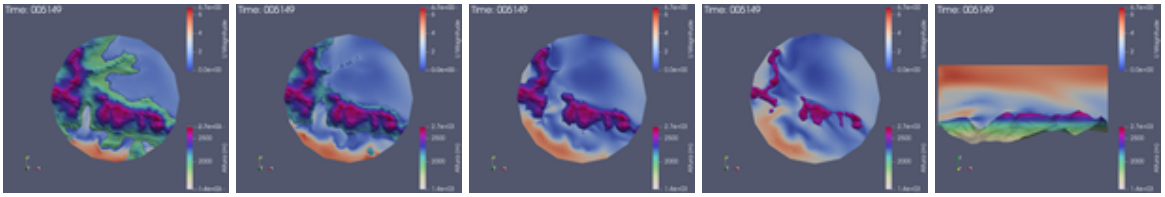
1900 m 2100 m 2300 m 2500 m Lateral Y01



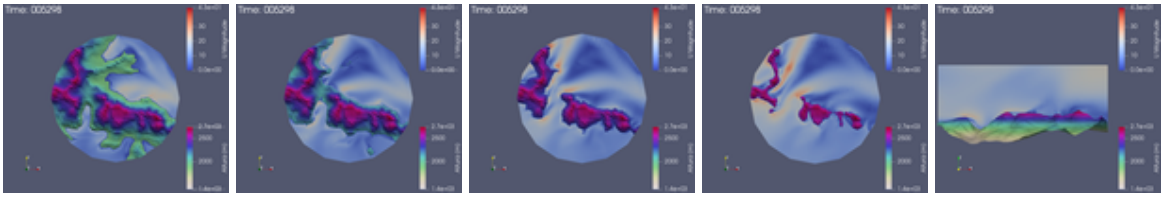
1900 m 2100 m 2300 m 2500 m Lateral Y01



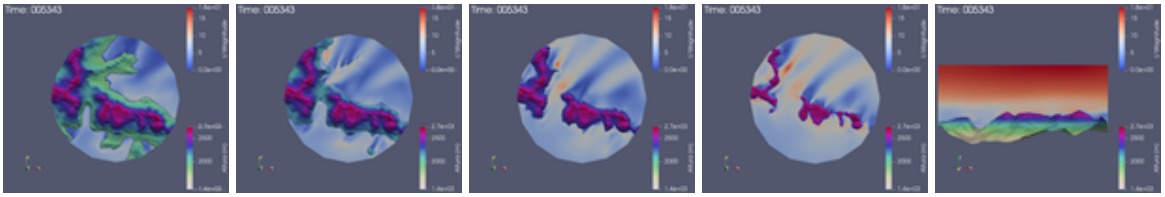
1900 m 2100 m 2300 m 2500 m Lateral Y01



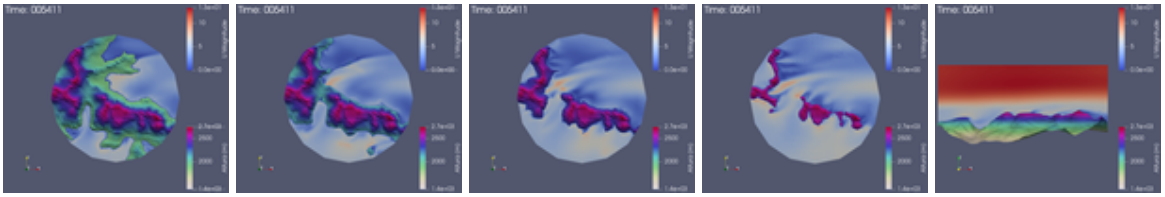
1900 m 2100 m 2300 m 2500 m Lateral Y01



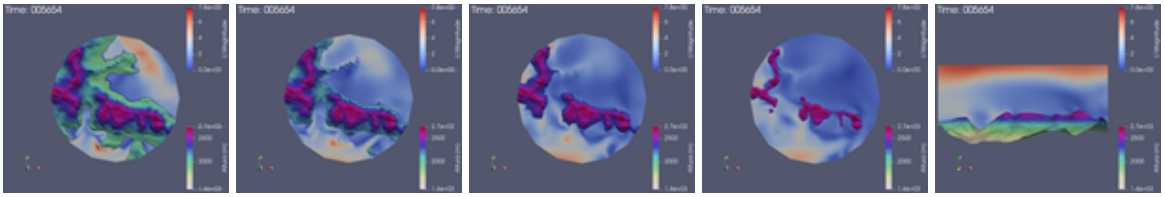
1900 m 2100 m 2300 m 2500 m Lateral Y01



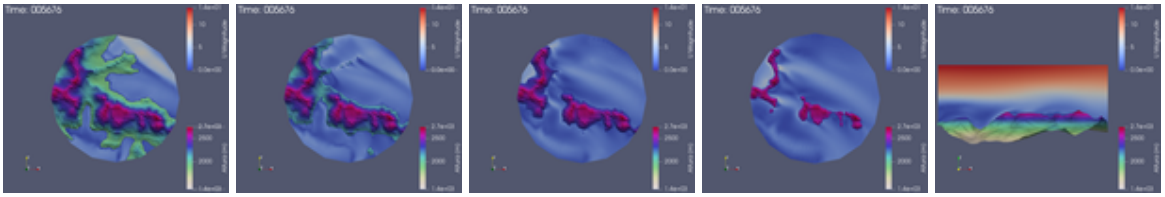
1900 m 2100 m 2300 m 2500 m Lateral Y01



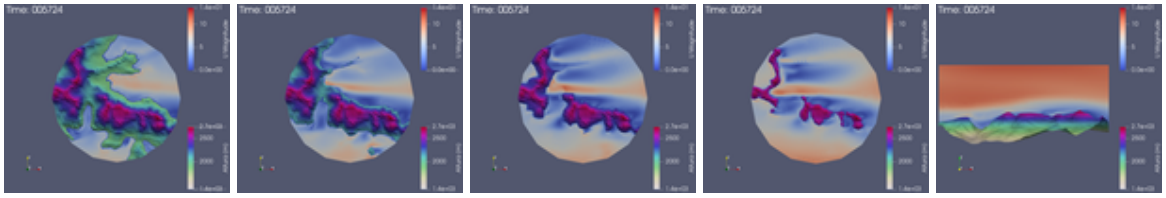
1900 m 2100 m 2300 m 2500 m Lateral Y01



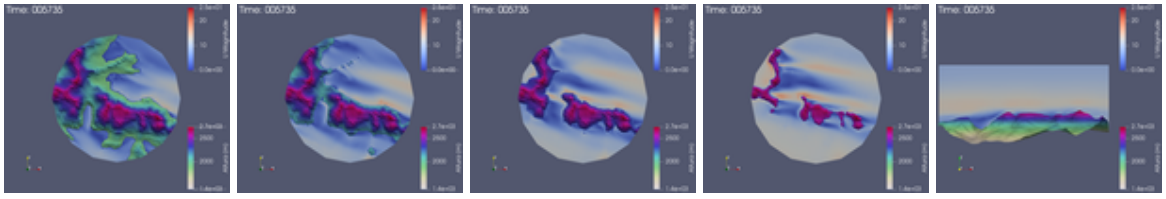
1900 m 2100 m 2300 m 2500 m Lateral Y01



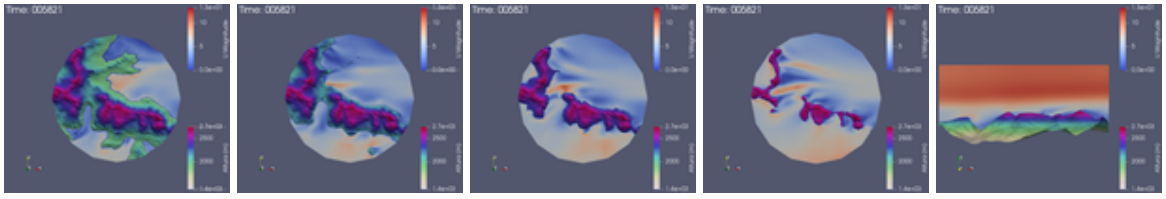
1900 m 2100 m 2300 m 2500 m Lateral Y01



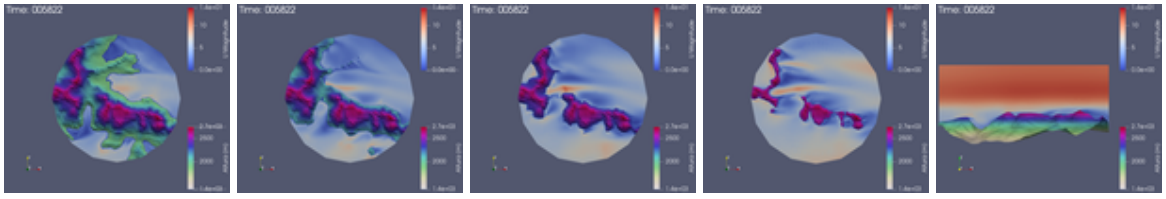
1900 m 2100 m 2300 m 2500 m Lateral Y01



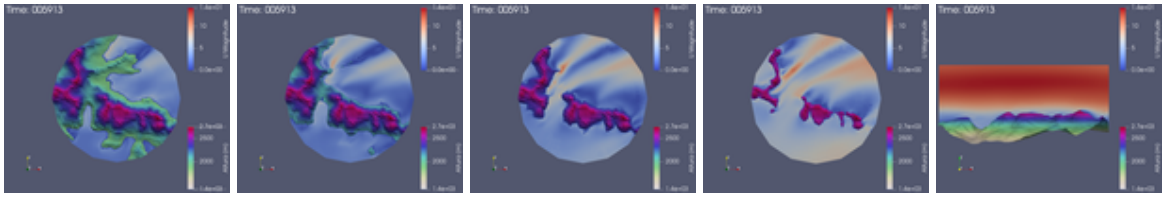
1900 m 2100 m 2300 m 2500 m Lateral Y01



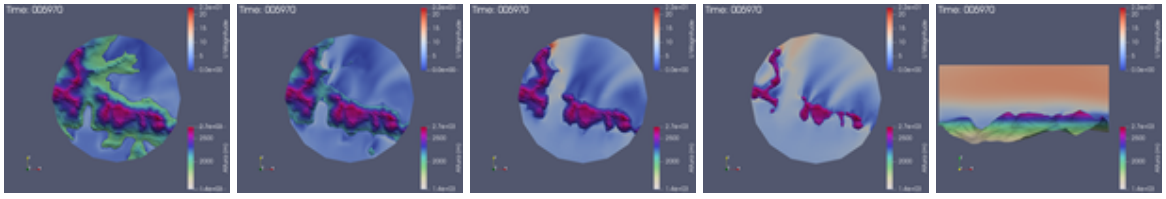
1900 m 2100 m 2300 m 2500 m Lateral Y01



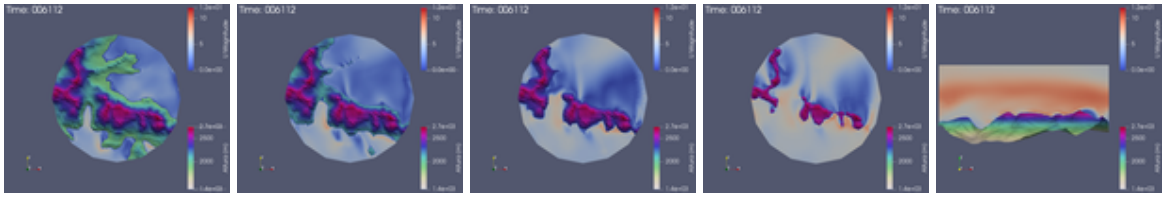
1900 m 2100 m 2300 m 2500 m Lateral Y01



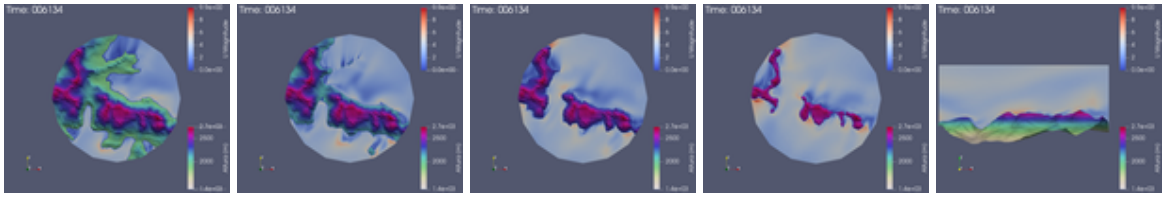
1900 m 2100 m 2300 m 2500 m Lateral Y01



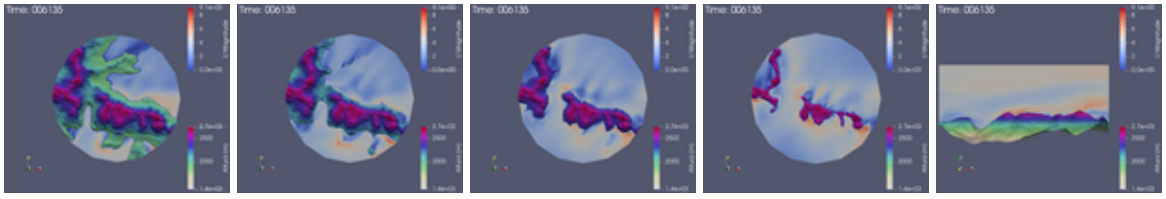
1900 m 2100 m 2300 m 2500 m Lateral Y01



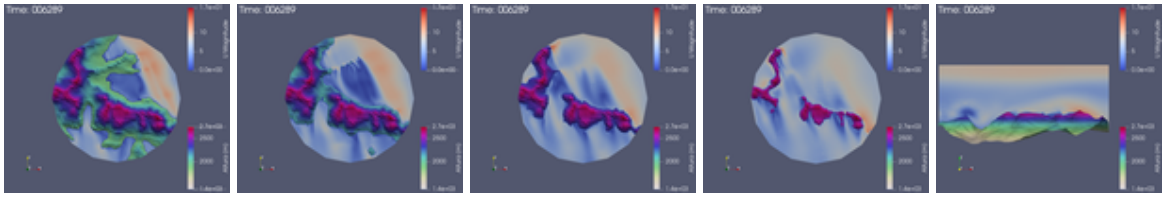
1900 m 2100 m 2300 m 2500 m Lateral Y01



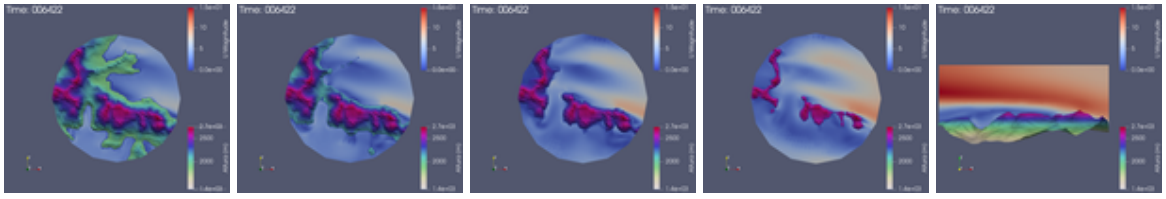
1900 m 2100 m 2300 m 2500 m Lateral Y01



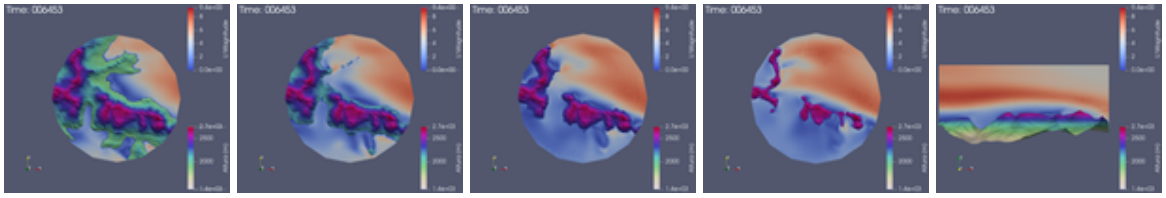
1900 m 2100 m 2300 m 2500 m Lateral Y01



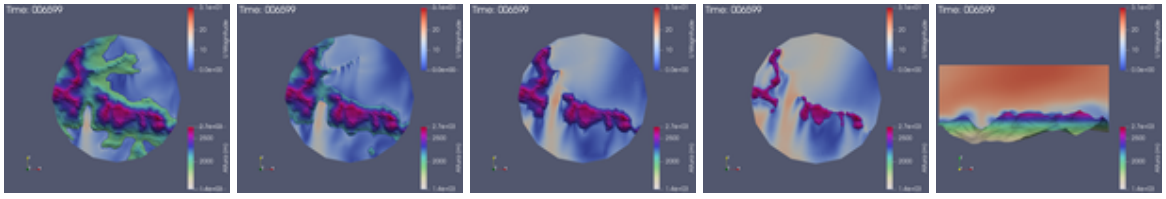
1900 m 2100 m 2300 m 2500 m Lateral Y01



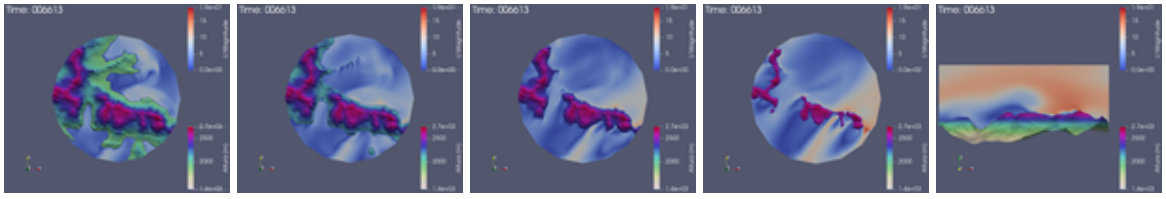
1900 m 2100 m 2300 m 2500 m Lateral Y01



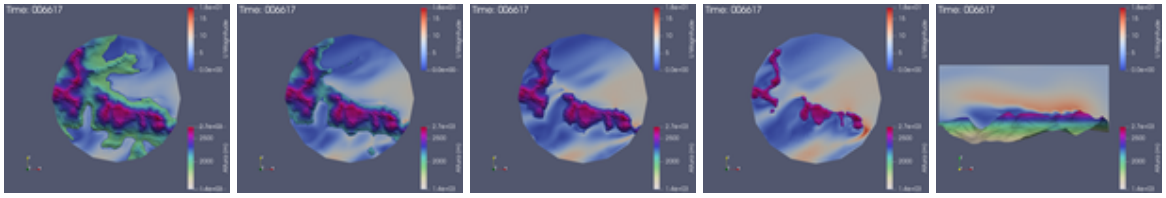
1900 m 2100 m 2300 m 2500 m Lateral Y01



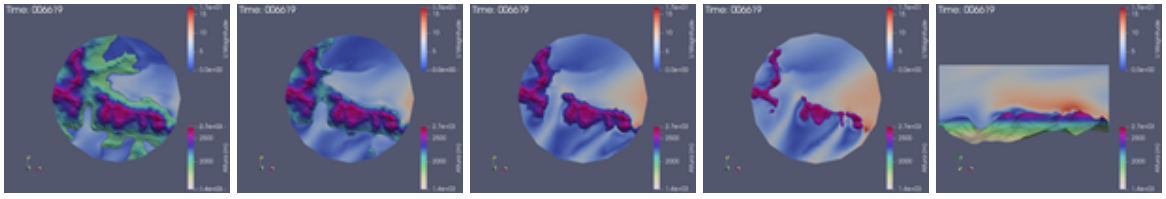
1900 m 2100 m 2300 m 2500 m Lateral Y01



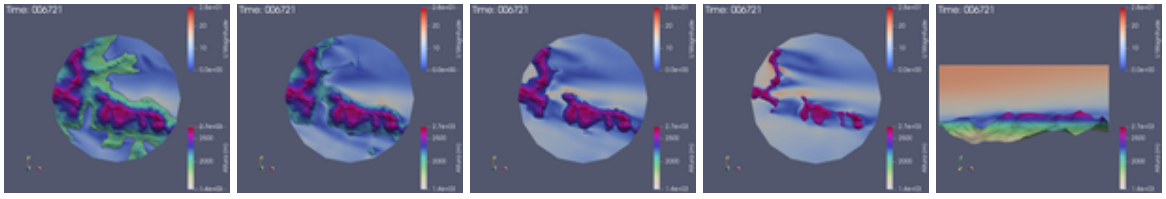
1900 m 2100 m 2300 m 2500 m Lateral Y01



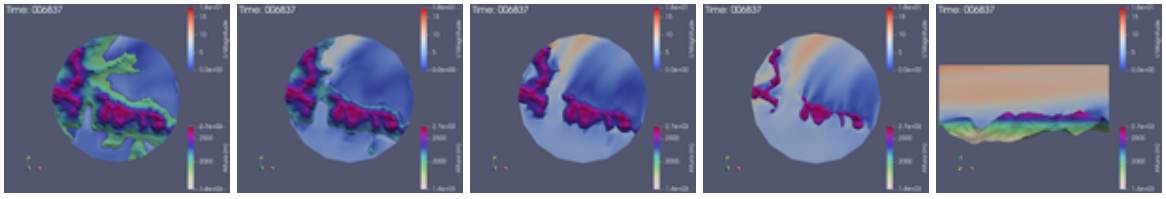
1900 m 2100 m 2300 m 2500 m Lateral Y01



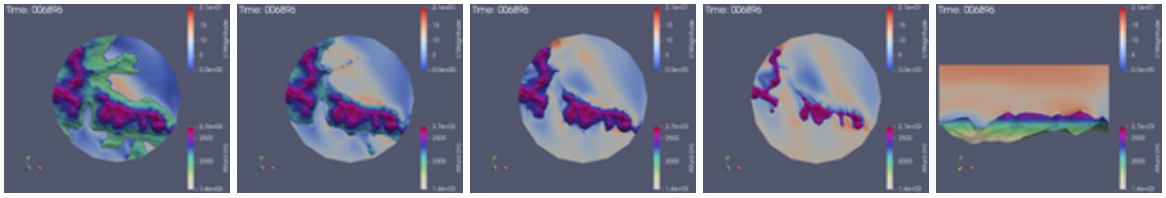
1900 m 2100 m 2300 m 2500 m Lateral Y01



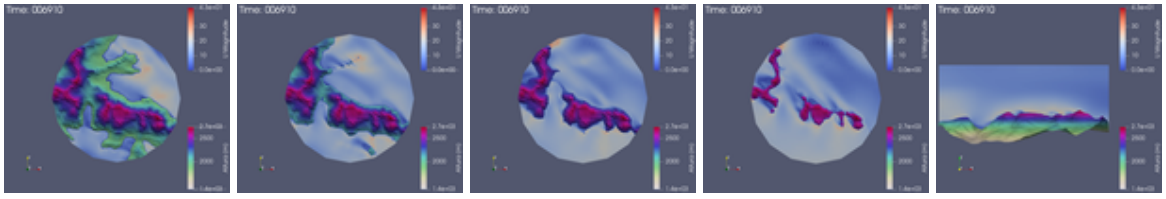
1900 m 2100 m 2300 m 2500 m Lateral Y01



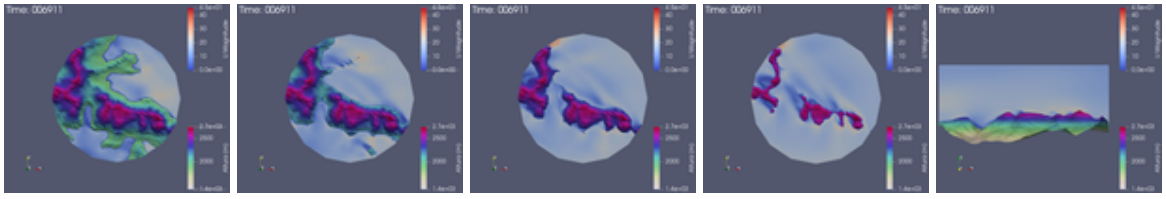
1900 m 2100 m 2300 m 2500 m Lateral Y01



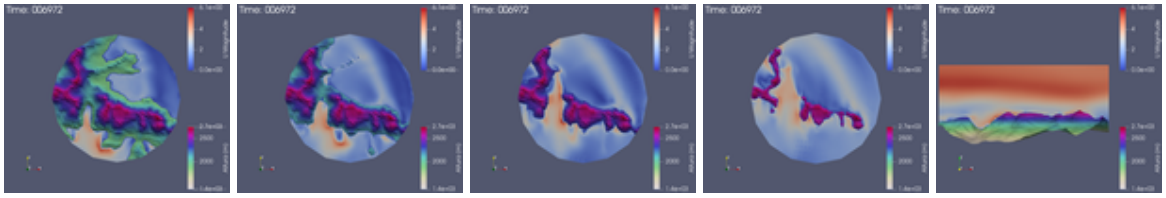
1900 m 2100 m 2300 m 2500 m Lateral Y01



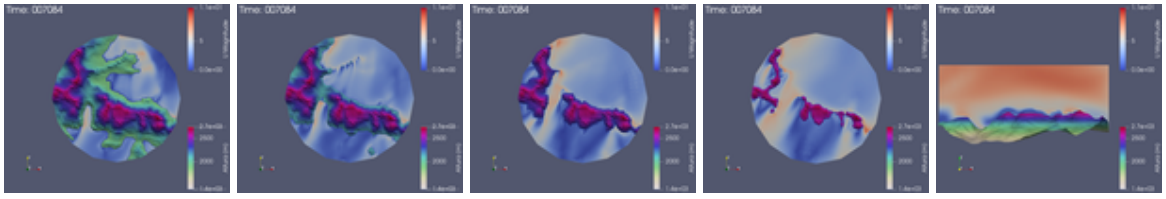
1900 m 2100 m 2300 m 2500 m Lateral Y01



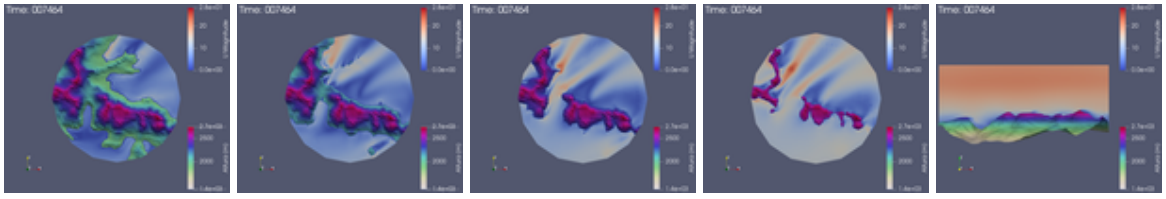
1900 m 2100 m 2300 m 2500 m Lateral Y01



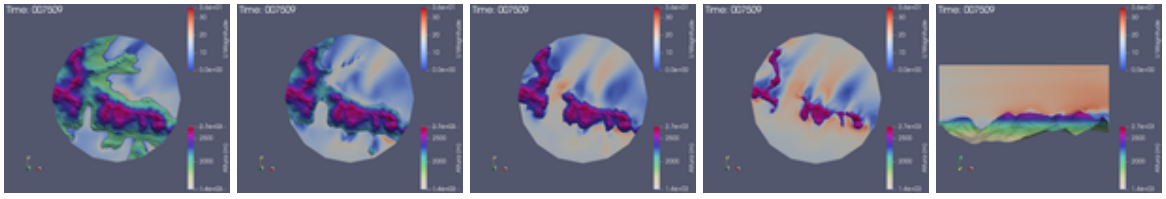
1900 m 2100 m 2300 m 2500 m Lateral Y01



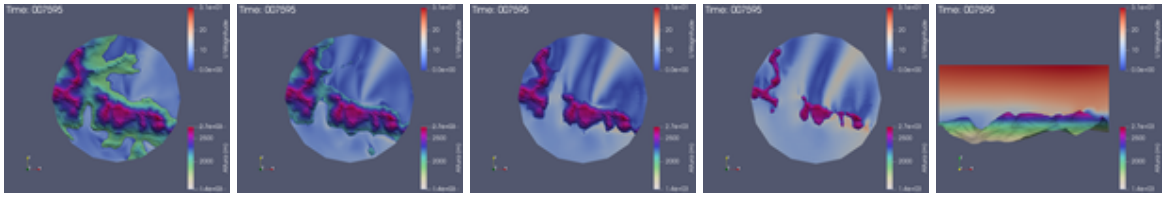
1900 m 2100 m 2300 m 2500 m Lateral Y01



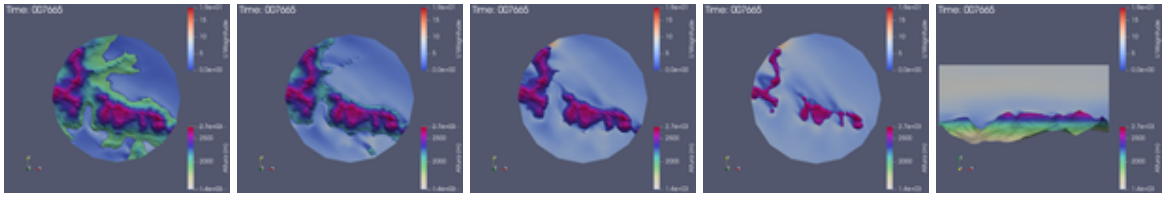
1900 m 2100 m 2300 m 2500 m Lateral Y01



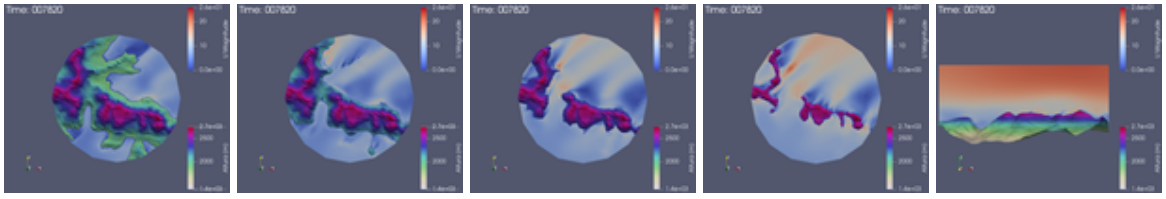
1900 m 2100 m 2300 m 2500 m Lateral Y01



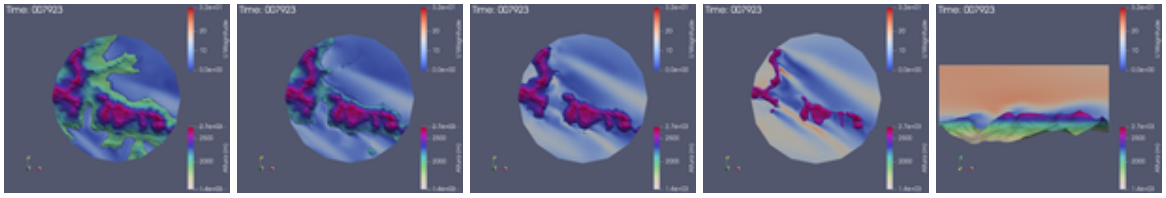
1900 m 2100 m 2300 m 2500 m Lateral Y01



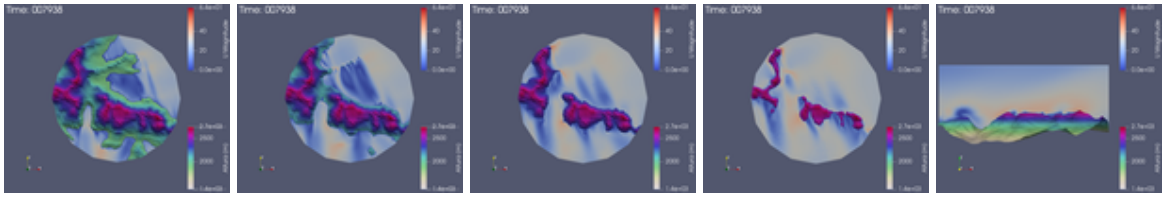
1900 m 2100 m 2300 m 2500 m Lateral Y01



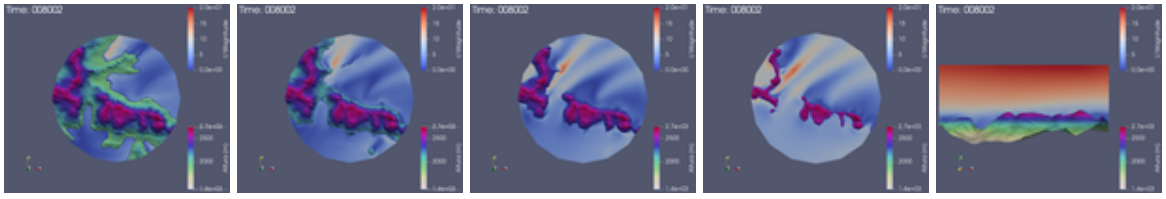
1900 m 2100 m 2300 m 2500 m Lateral Y01



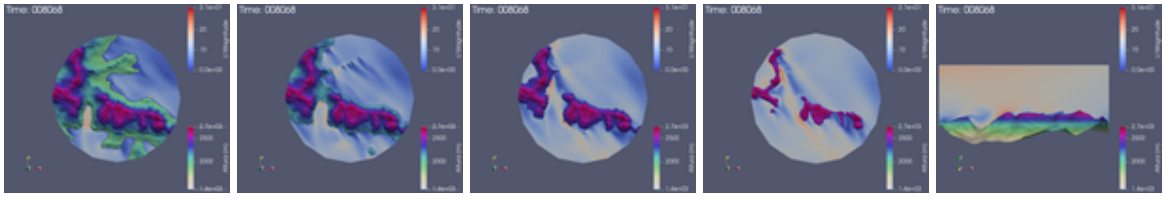
1900 m 2100 m 2300 m 2500 m Lateral Y01



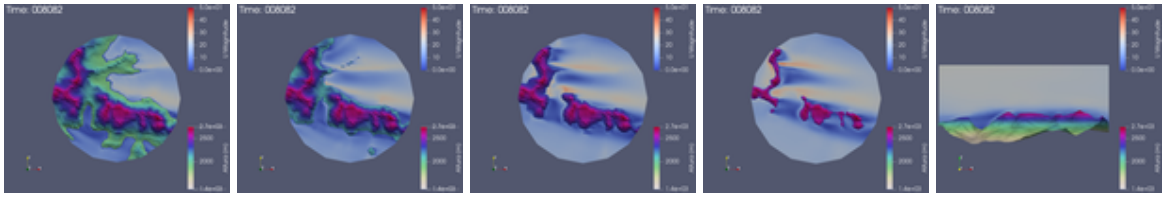
1900 m 2100 m 2300 m 2500 m Lateral Y01



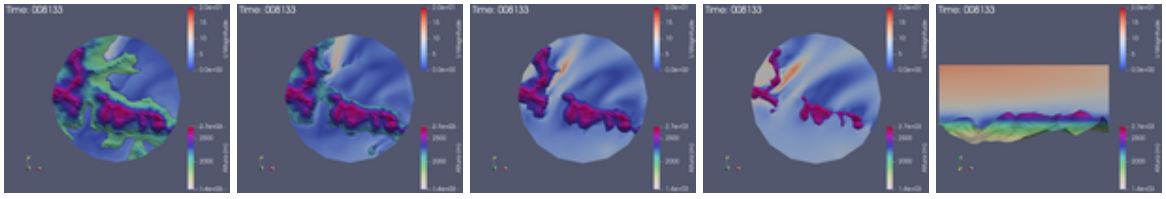
1900 m 2100 m 2300 m 2500 m Lateral Y01



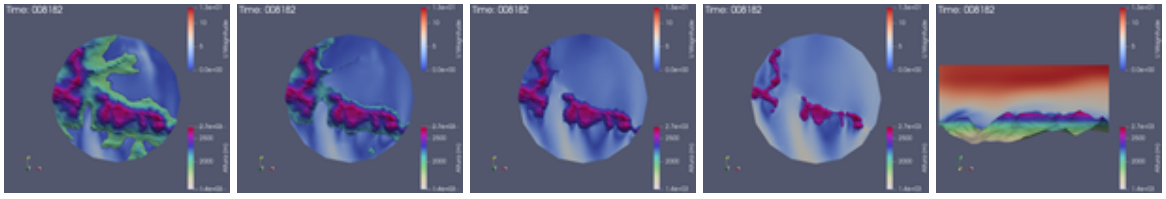
1900 m 2100 m 2300 m 2500 m Lateral Y01



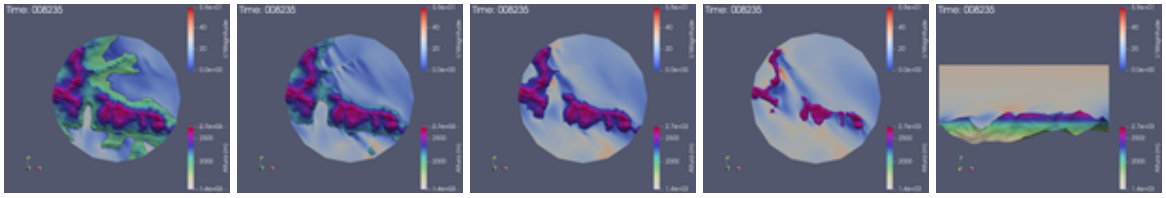
1900 m 2100 m 2300 m 2500 m Lateral Y01



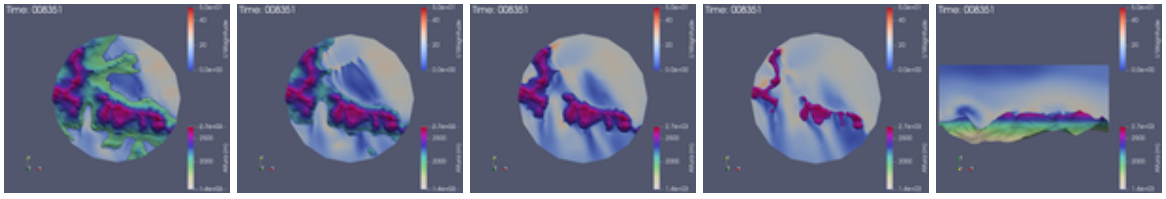
1900 m 2100 m 2300 m 2500 m Lateral Y01



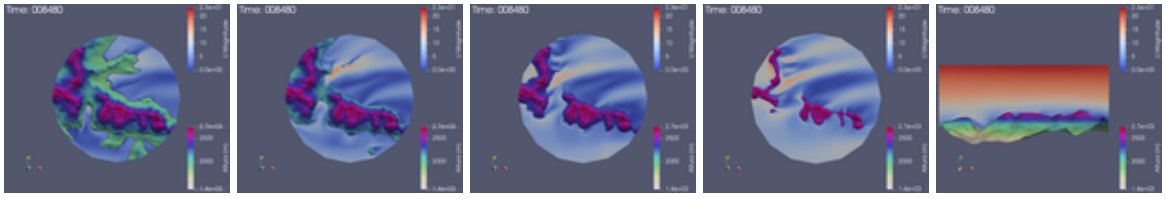
1900 m 2100 m 2300 m 2500 m Lateral Y01



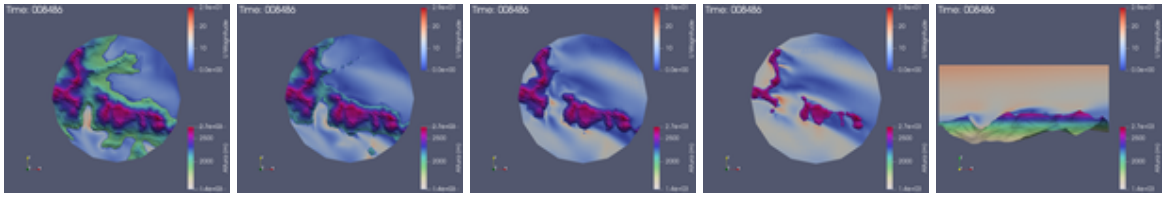
1900 m 2100 m 2300 m 2500 m Lateral Y01



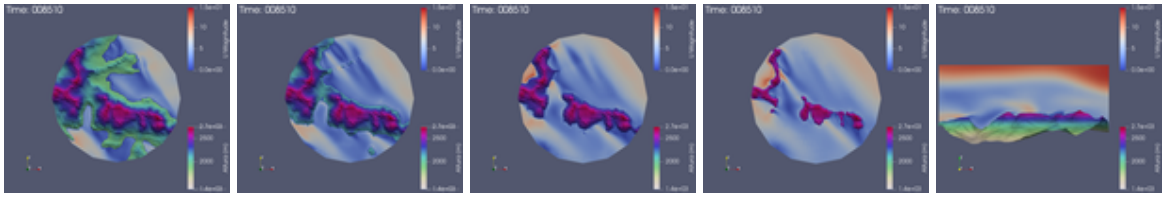
1900 m 2100 m 2300 m 2500 m Lateral Y01



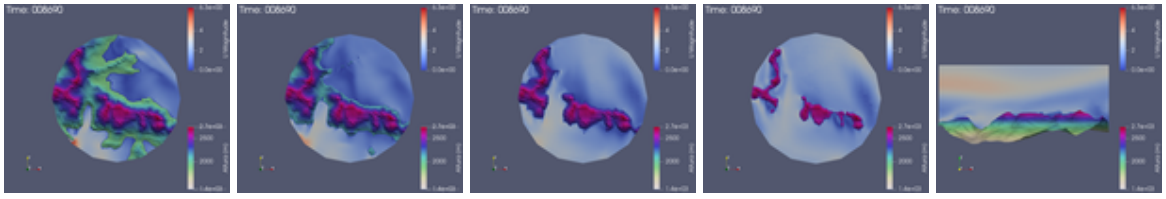
1900 m 2100 m 2300 m 2500 m Lateral Y01



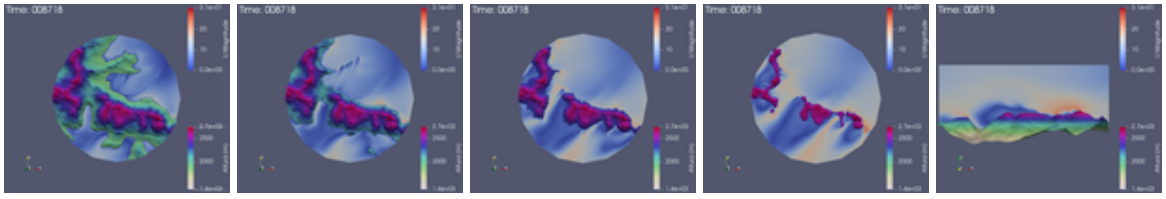
1900 m 2100 m 2300 m 2500 m Lateral Y01



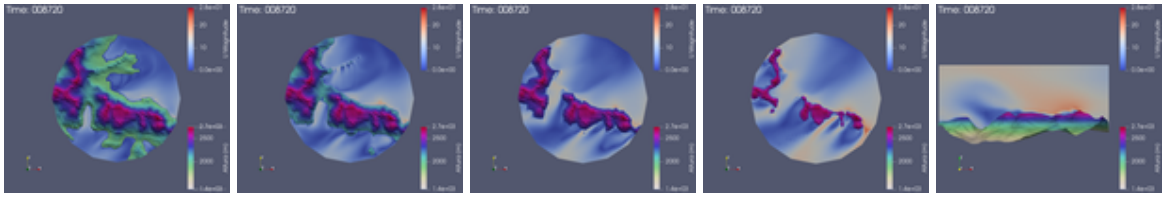
1900 m 2100 m 2300 m 2500 m Lateral Y01



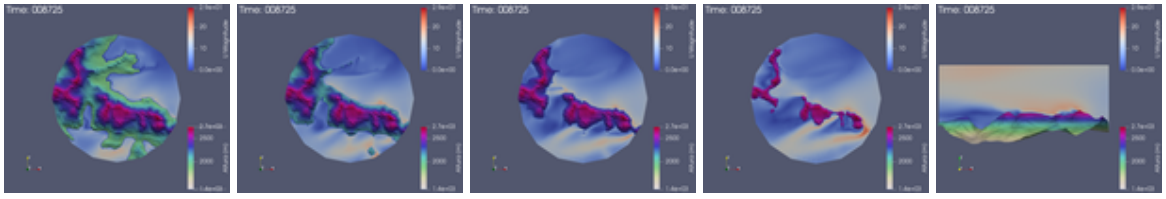
1900 m 2100 m 2300 m 2500 m Lateral Y01



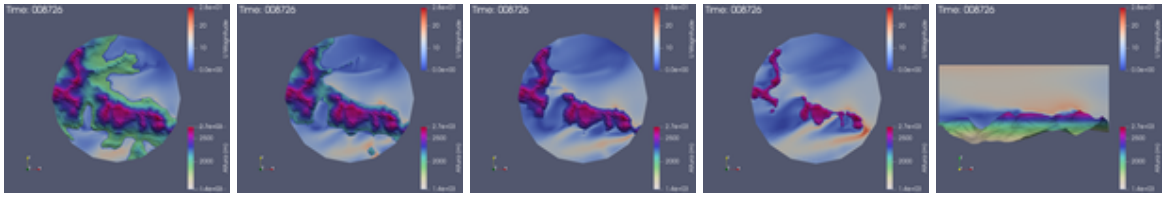
1900 m 2100 m 2300 m 2500 m Lateral Y01



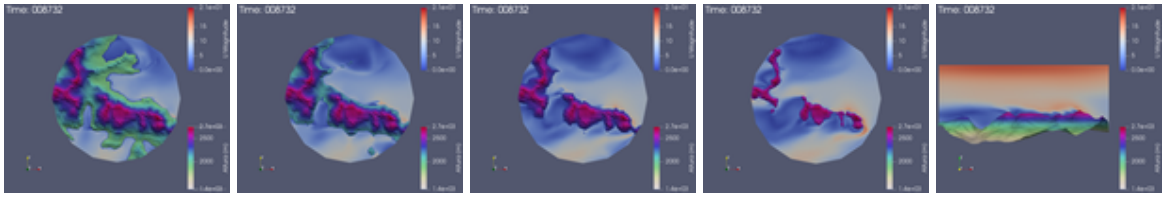
1900 m 2100 m 2300 m 2500 m Lateral Y01



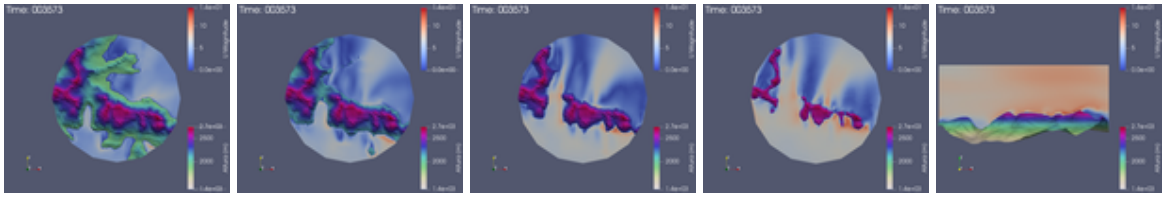
1900 m 2100 m 2300 m 2500 m Lateral Y01



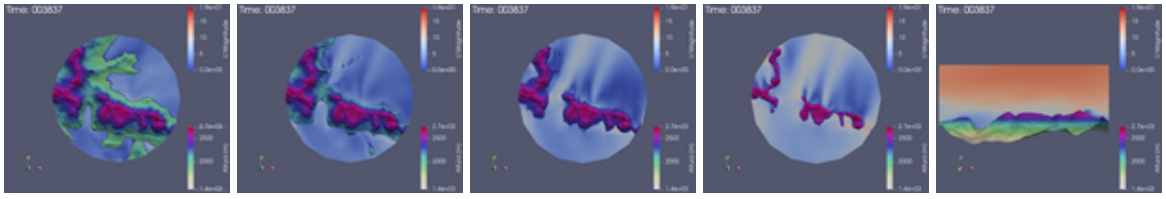
1900 m 2100 m 2300 m 2500 m Lateral Y01



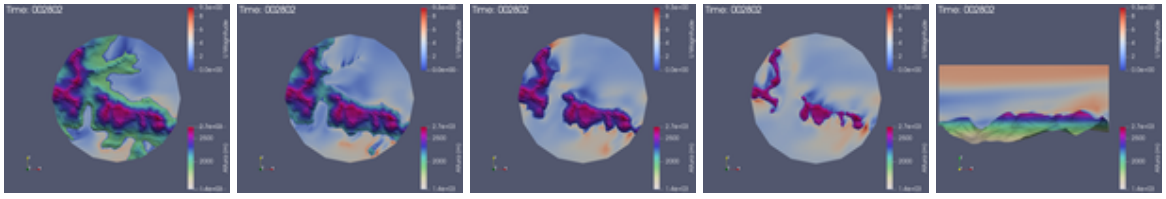
1900 m 2100 m 2300 m 2500 m Lateral Y01



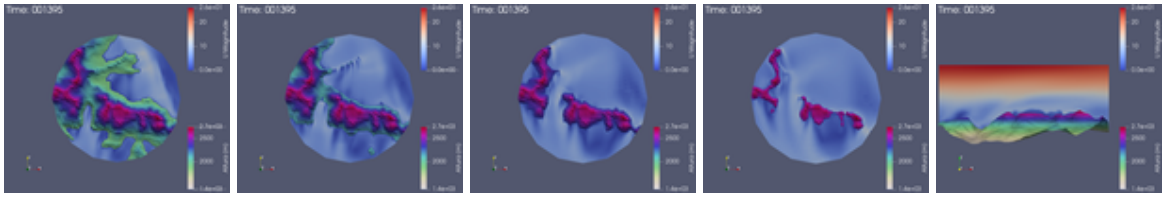
1900 m 2100 m 2300 m 2500 m Lateral Y01



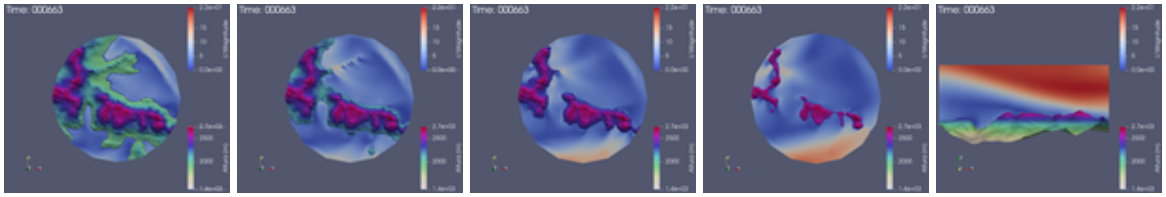
1900 m 2100 m 2300 m 2500 m Lateral Y01



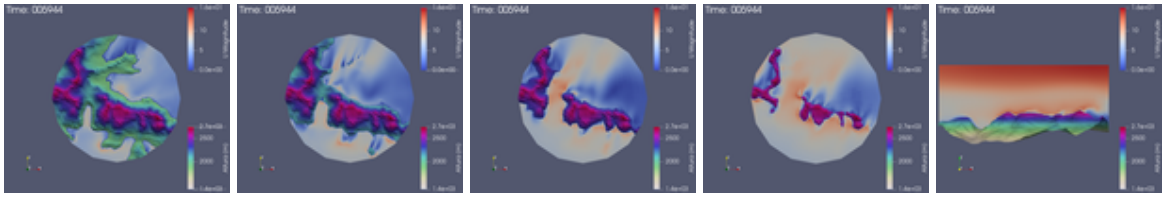
1900 m 2100 m 2300 m 2500 m Lateral Y01



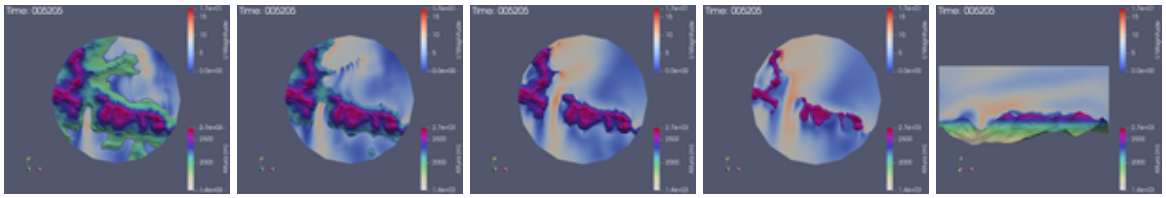
1900 m 2100 m 2300 m 2500 m Lateral Y01



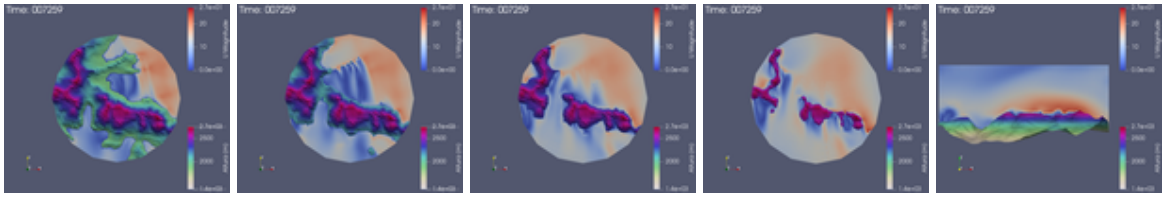
1900 m 2100 m 2300 m 2500 m Lateral Y01



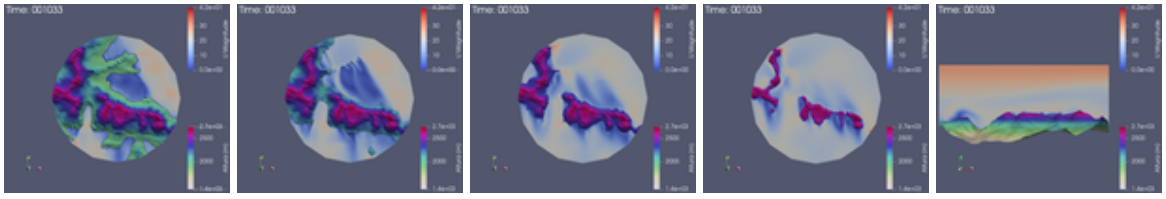
1900 m 2100 m 2300 m 2500 m Lateral Y01



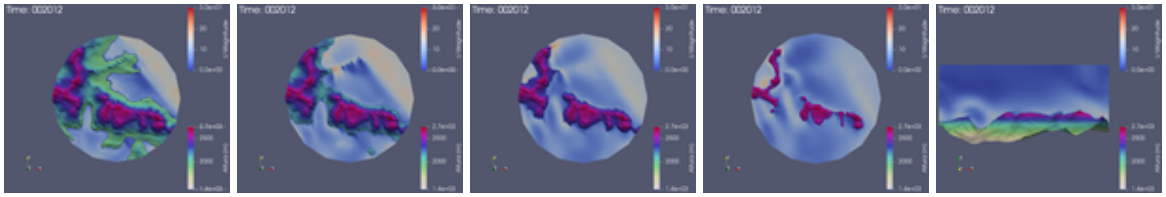
1900 m 2100 m 2300 m 2500 m Lateral Y01



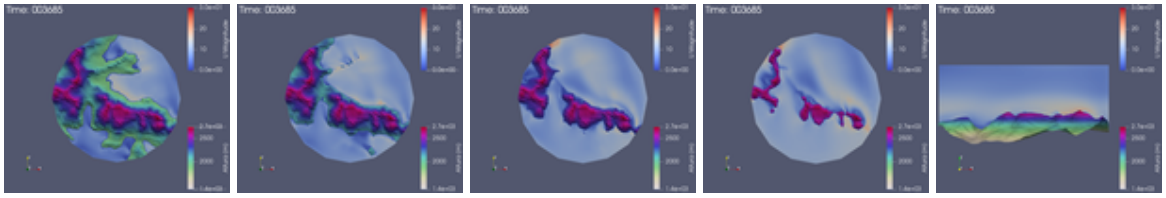
1900 m 2100 m 2300 m 2500 m Lateral Y01



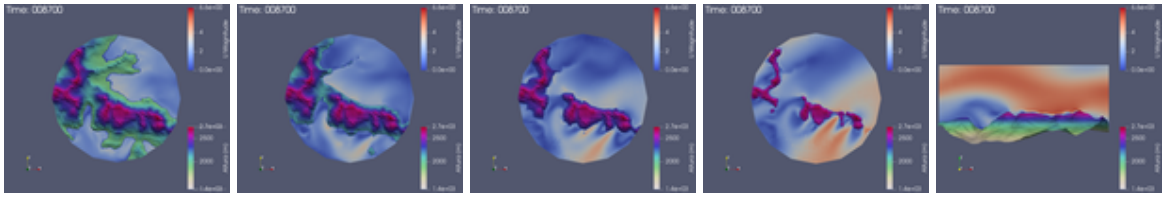
1900 m 2100 m 2300 m 2500 m Lateral Y01



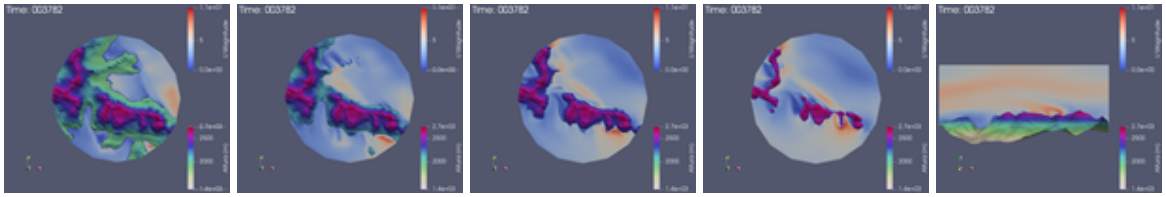
1900 m 2100 m 2300 m 2500 m Lateral Y01



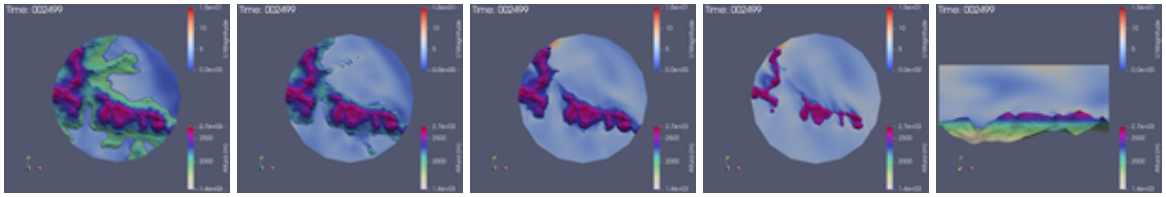
1900 m 2100 m 2300 m 2500 m Lateral Y01



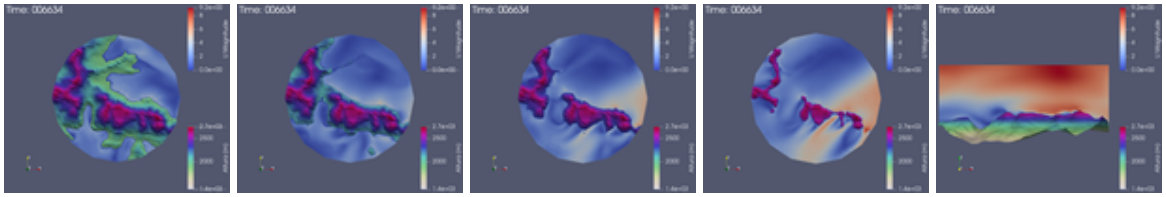
1900 m 2100 m 2300 m 2500 m Lateral Y01



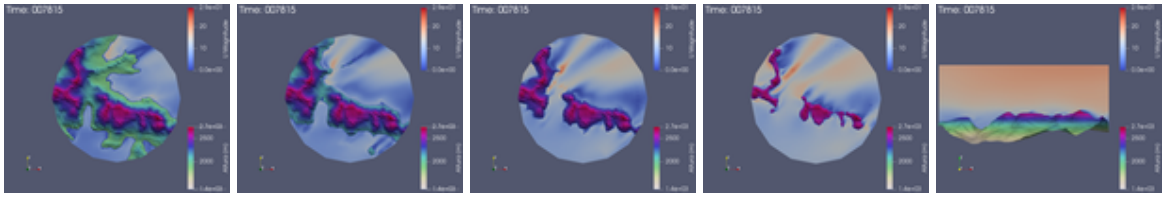
1900 m 2100 m 2300 m 2500 m Lateral Y01



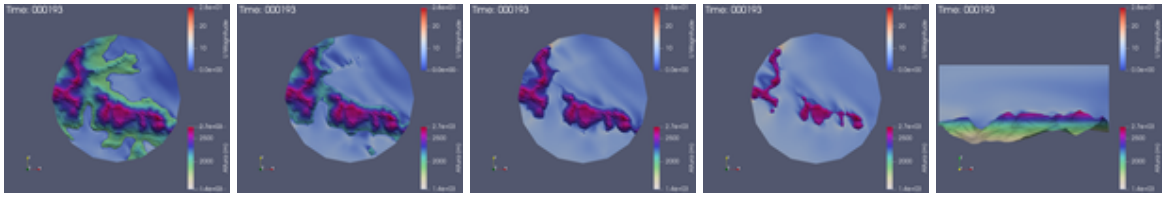
1900 m 2100 m 2300 m 2500 m Lateral Y01



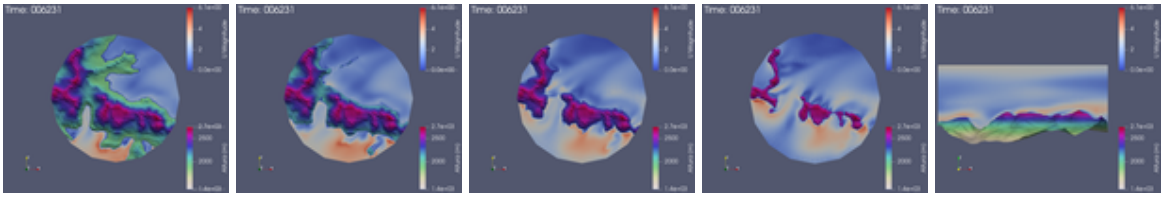
1900 m 2100 m 2300 m 2500 m Lateral Y01



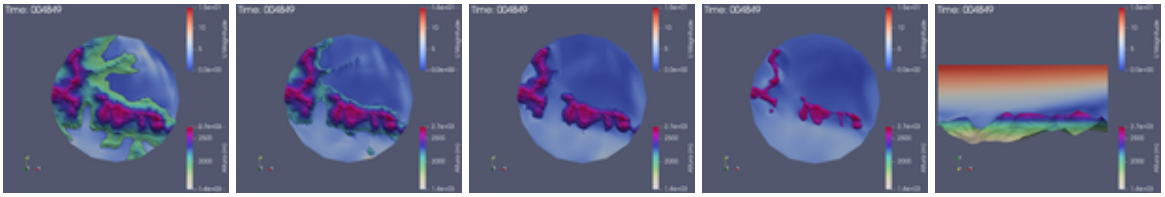
1900 m 2100 m 2300 m 2500 m Lateral Y01



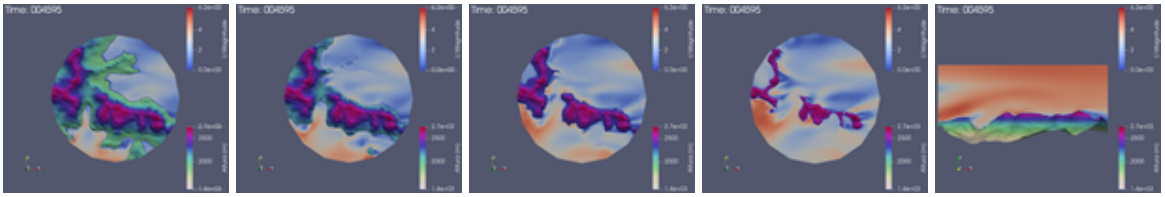
1900 m 2100 m 2300 m 2500 m Lateral Y01



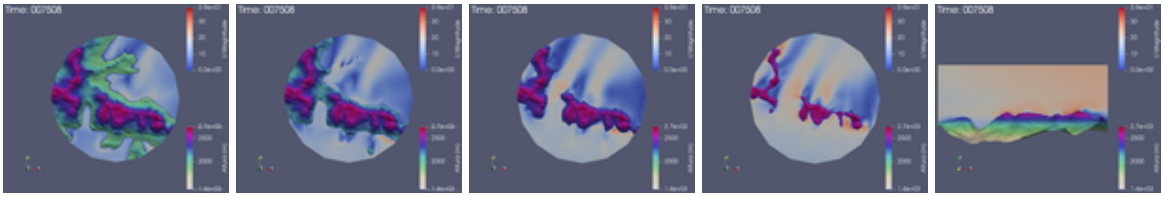
1900 m 2100 m 2300 m 2500 m Lateral Y01



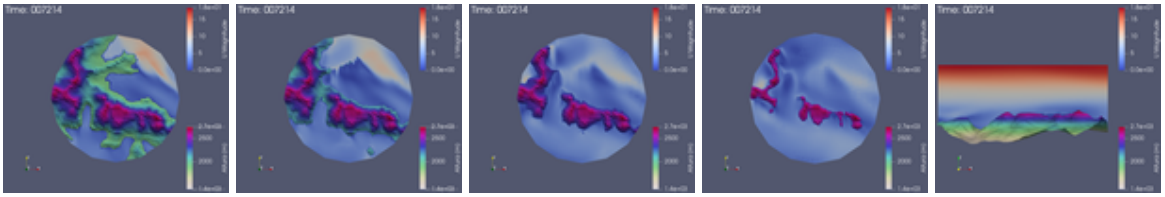
1900 m 2100 m 2300 m 2500 m Lateral Y01



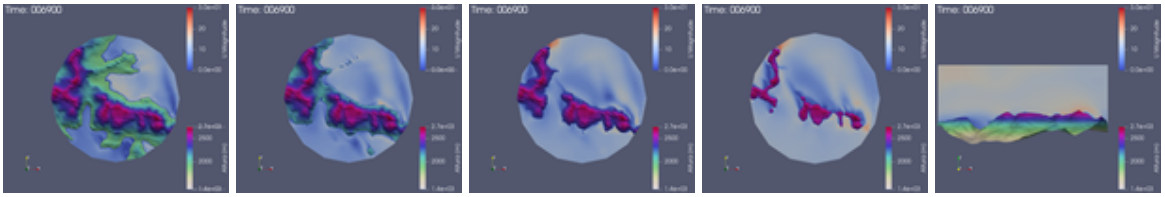
1900 m 2100 m 2300 m 2500 m Lateral Y01



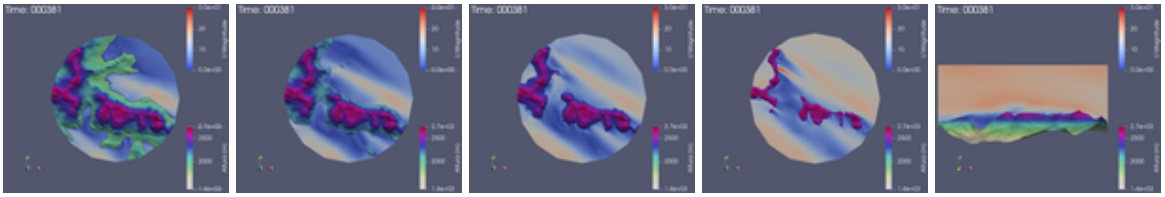
1900 m 2100 m 2300 m 2500 m Lateral Y01



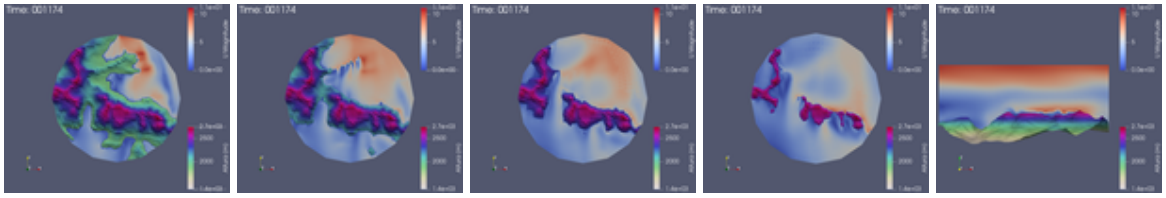
1900 m 2100 m 2300 m 2500 m Lateral Y01



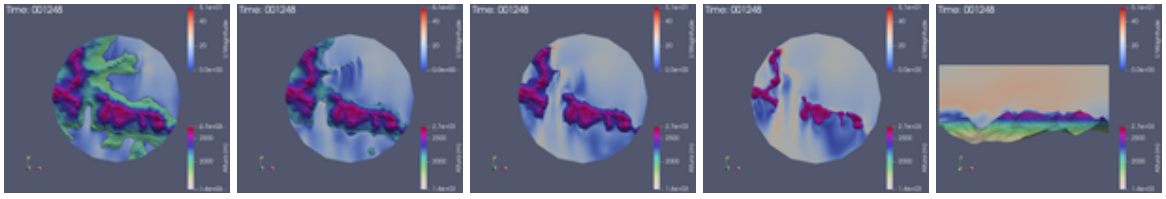
1900 m 2100 m 2300 m 2500 m Lateral Y01



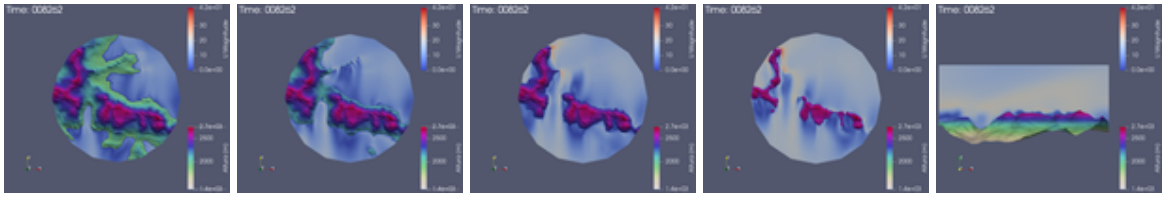
1900 m 2100 m 2300 m 2500 m Lateral Y01



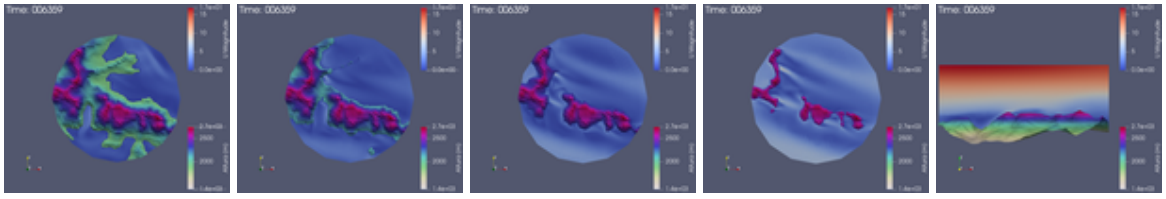
1900 m 2100 m 2300 m 2500 m Lateral Y01



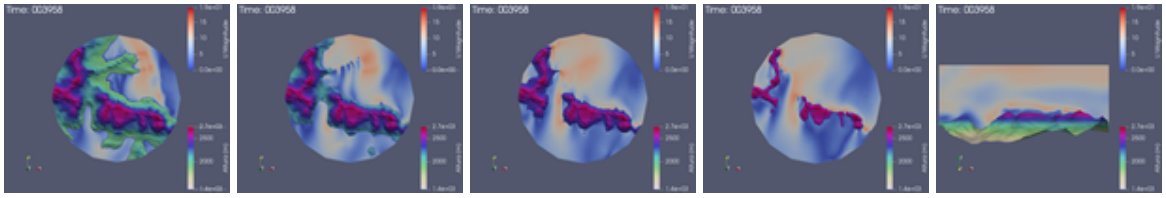
1900 m 2100 m 2300 m 2500 m Lateral Y01



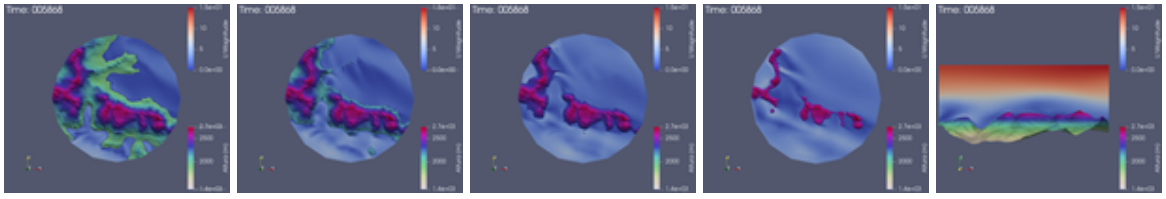
1900 m 2100 m 2300 m 2500 m Lateral Y01



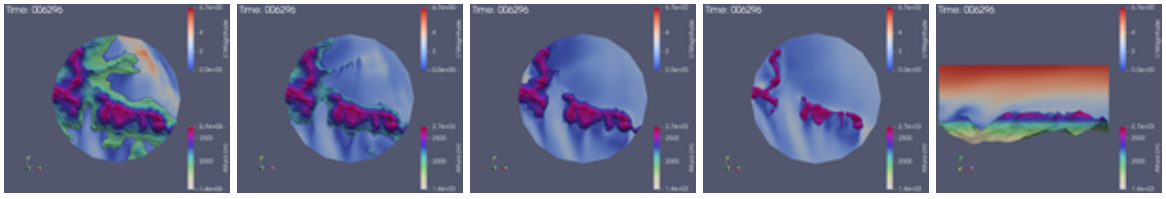
1900 m 2100 m 2300 m 2500 m Lateral Y01



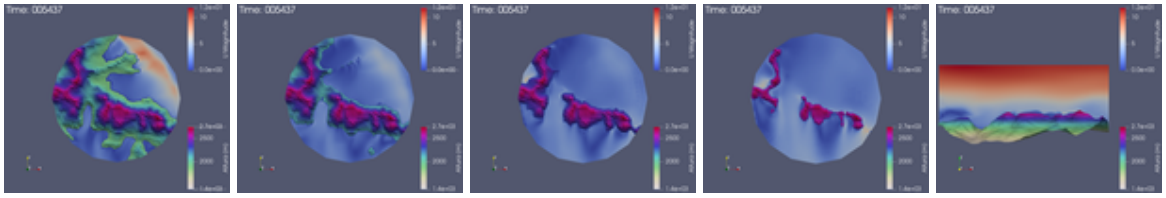
1900 m 2100 m 2300 m 2500 m Lateral Y01



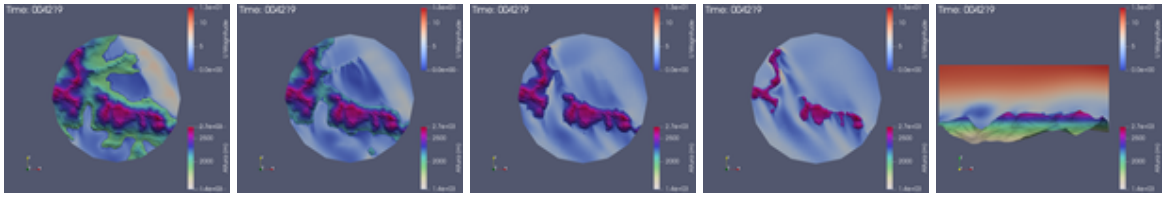
1900 m 2100 m 2300 m 2500 m Lateral Y01



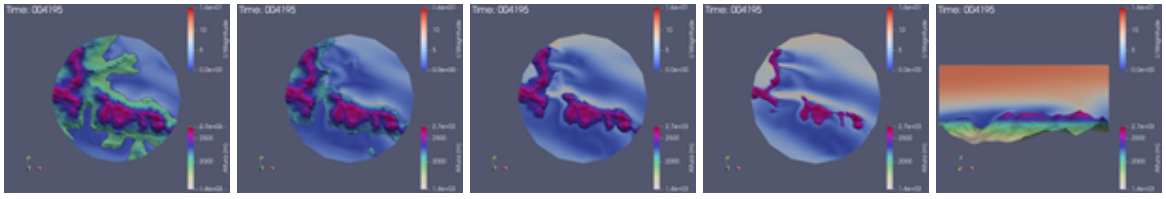
1900 m 2100 m 2300 m 2500 m Lateral Y01



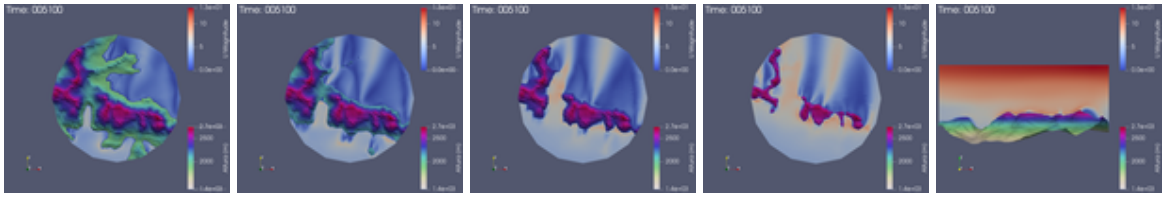
1900 m 2100 m 2300 m 2500 m Lateral Y01



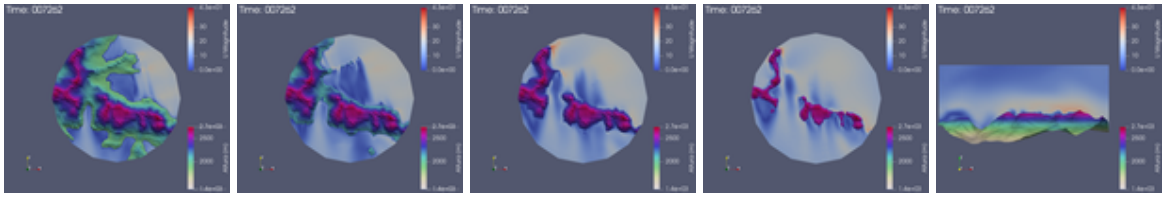
1900 m 2100 m 2300 m 2500 m Lateral Y01



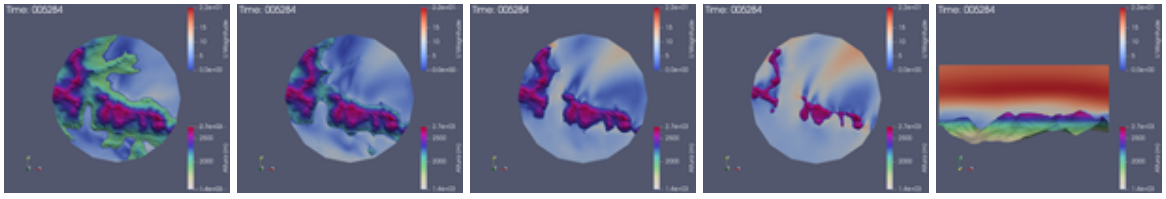
1900 m 2100 m 2300 m 2500 m Lateral Y01



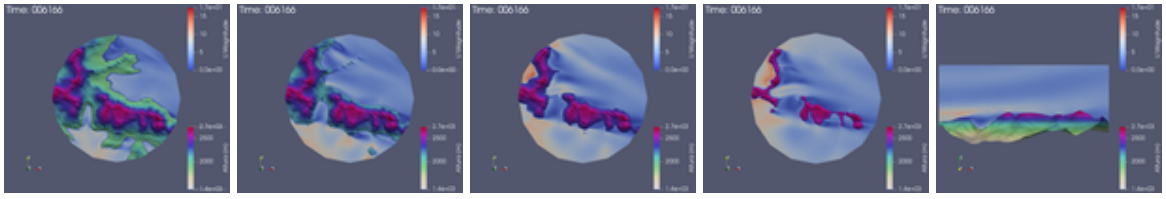
1900 m 2100 m 2300 m 2500 m Lateral Y01



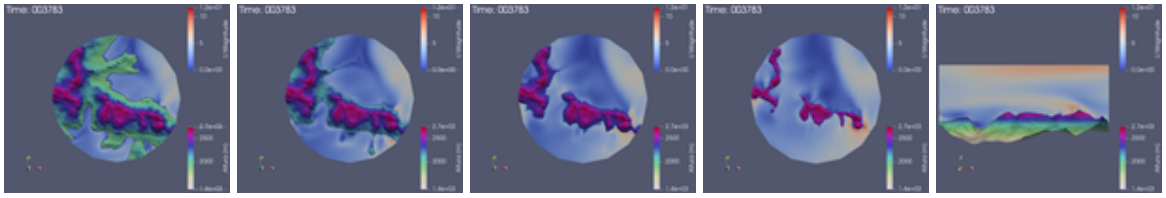
1900 m 2100 m 2300 m 2500 m Lateral Y01



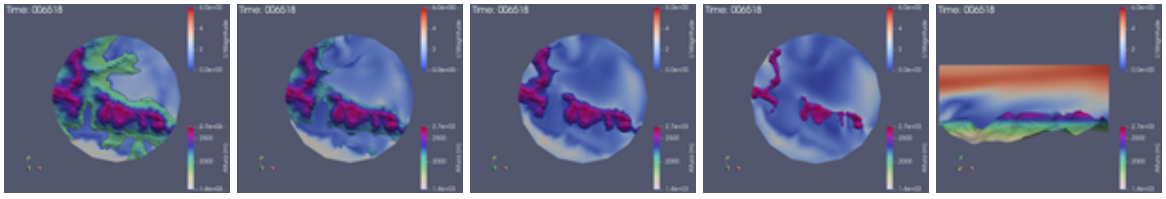
1900 m 2100 m 2300 m 2500 m Lateral Y01



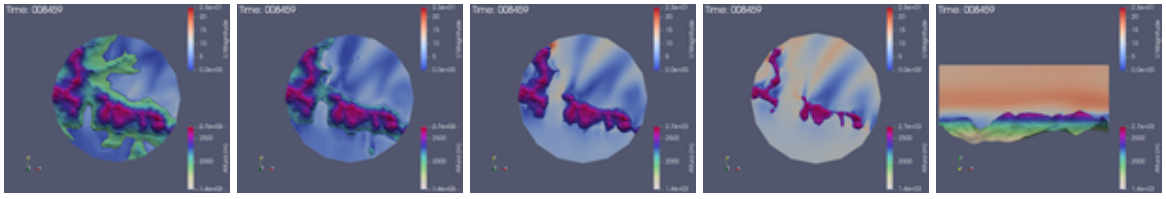
1900 m 2100 m 2300 m 2500 m Lateral Y01



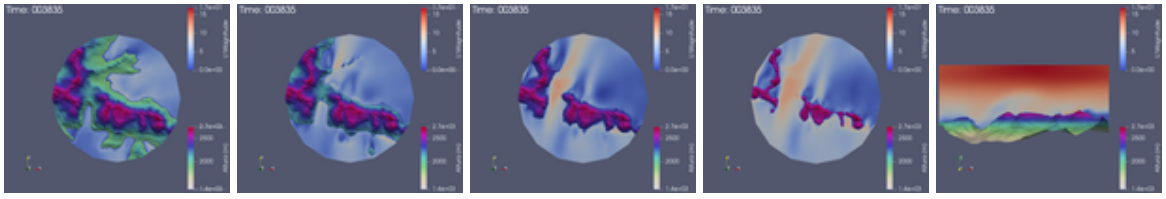
1900 m 2100 m 2300 m 2500 m Lateral Y01



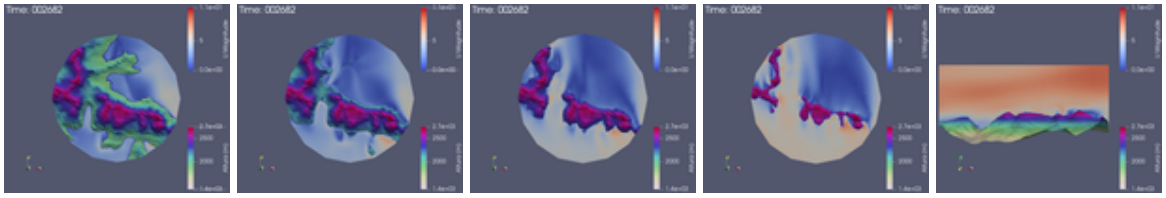
1900 m 2100 m 2300 m 2500 m Lateral Y01



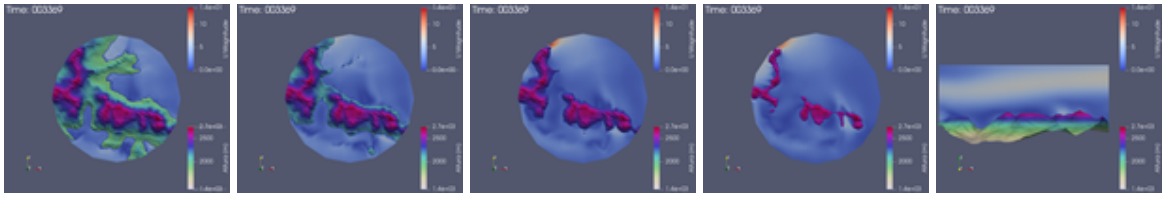
1900 m 2100 m 2300 m 2500 m Lateral Y01



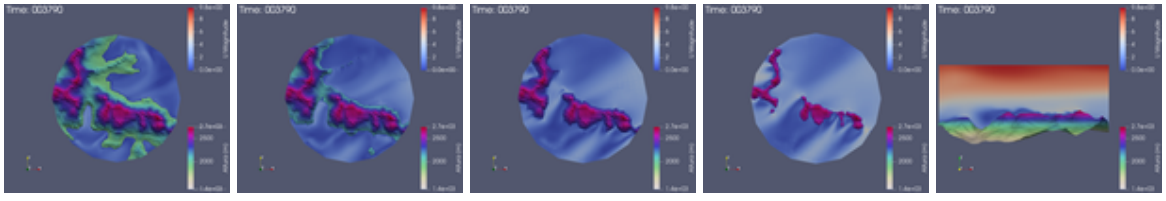
1900 m 2100 m 2300 m 2500 m Lateral Y01



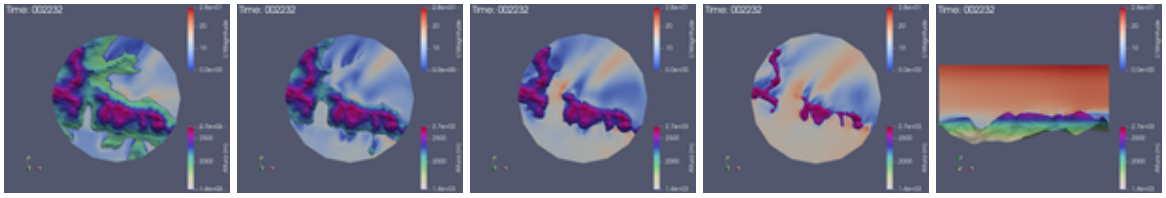
1900 m 2100 m 2300 m 2500 m Lateral Y01



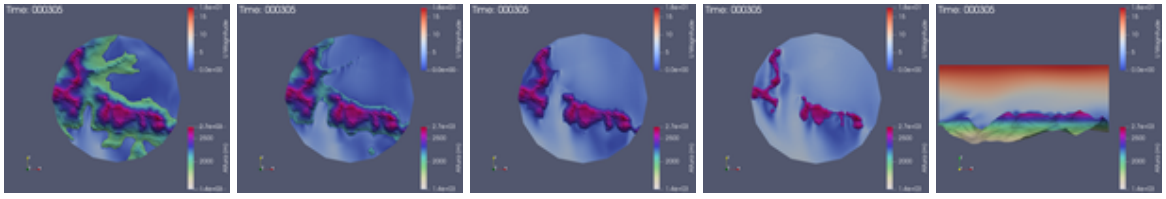
1900 m 2100 m 2300 m 2500 m Lateral Y01



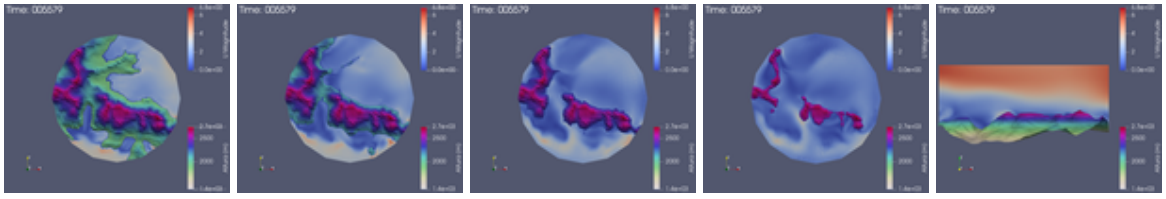
1900 m 2100 m 2300 m 2500 m Lateral Y01



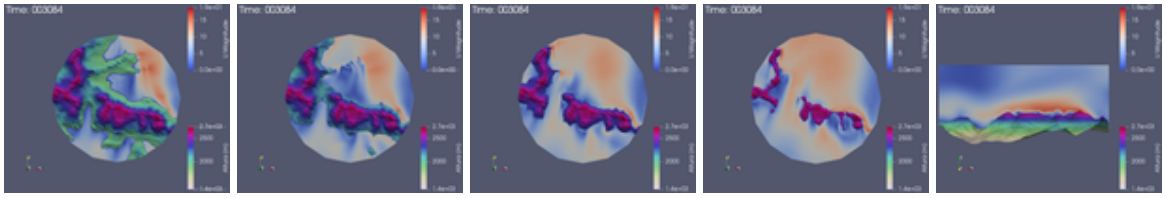
1900 m 2100 m 2300 m 2500 m Lateral Y01



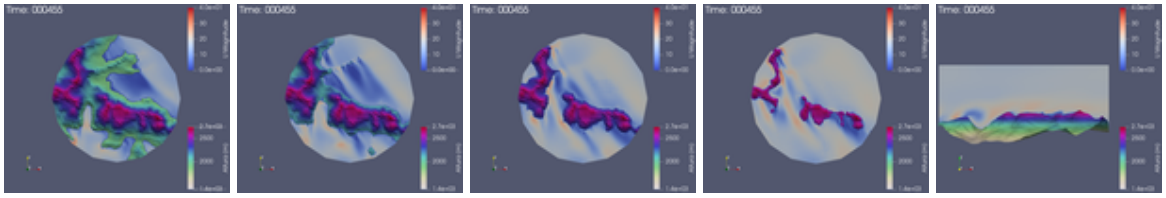
1900 m 2100 m 2300 m 2500 m Lateral Y01



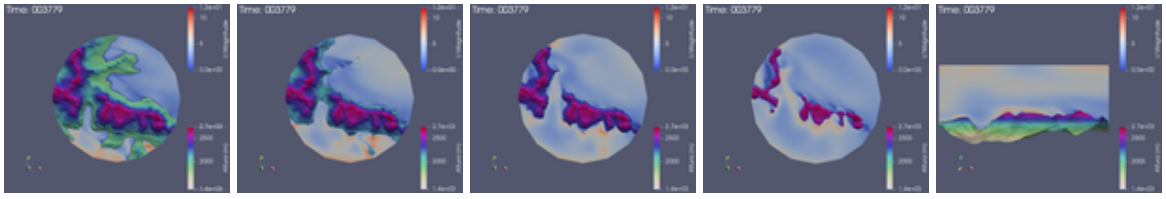
1900 m 2100 m 2300 m 2500 m Lateral Y01



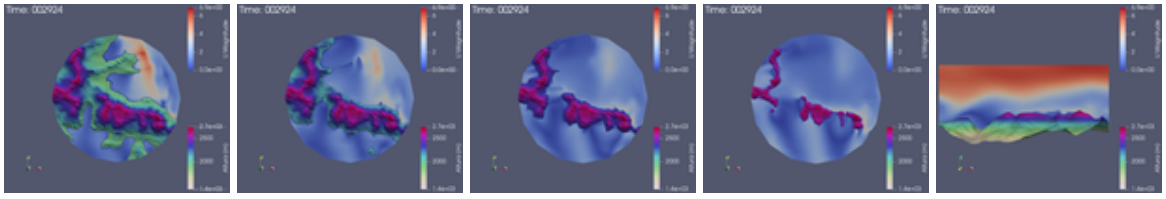
1900 m 2100 m 2300 m 2500 m Lateral Y01



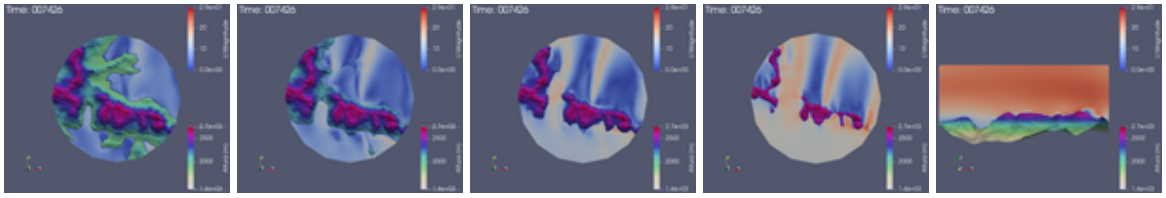
1900 m 2100 m 2300 m 2500 m Lateral Y01



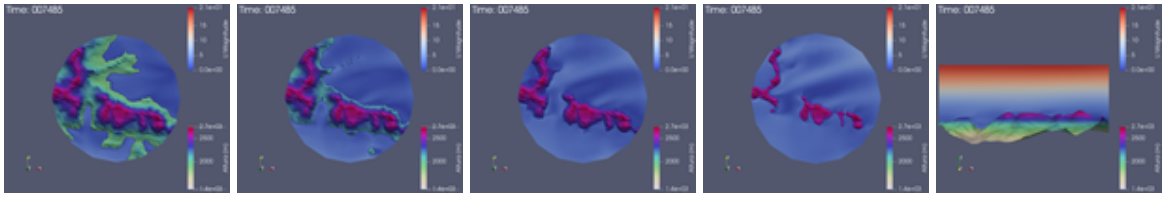
1900 m 2100 m 2300 m 2500 m Lateral Y01



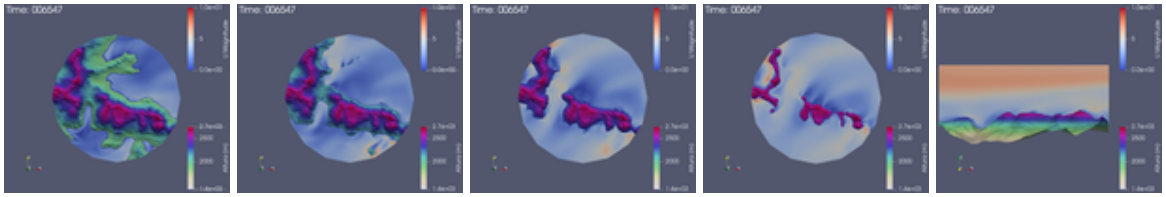
1900 m 2100 m 2300 m 2500 m Lateral Y01



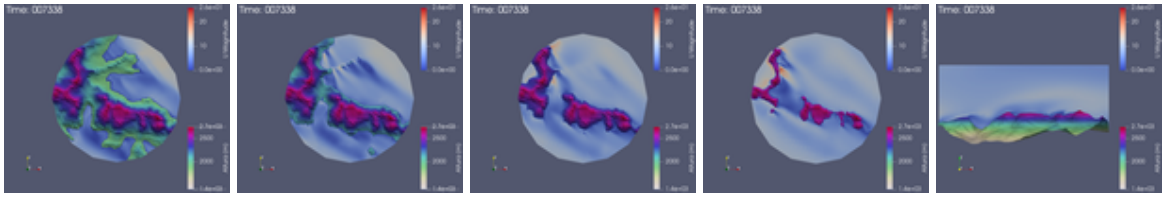
1900 m 2100 m 2300 m 2500 m Lateral Y01



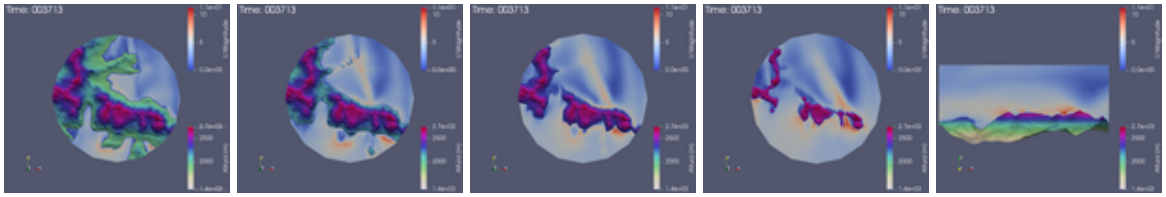
1900 m 2100 m 2300 m 2500 m Lateral Y01



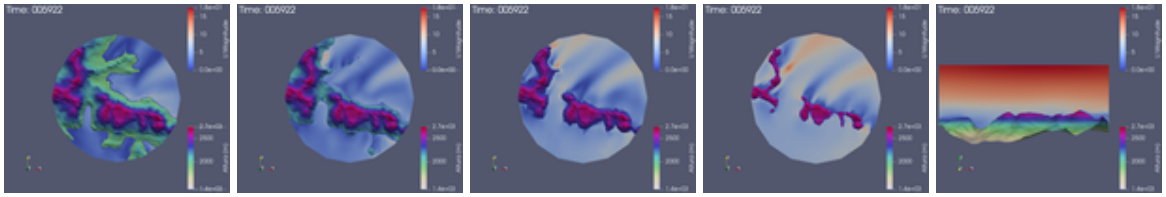
1900 m 2100 m 2300 m 2500 m Lateral Y01



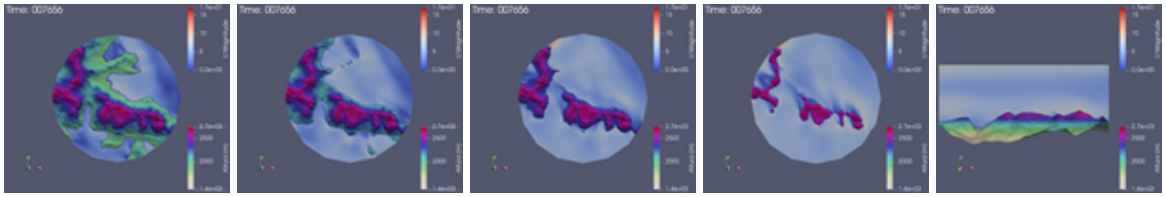
1900 m 2100 m 2300 m 2500 m Lateral Y01



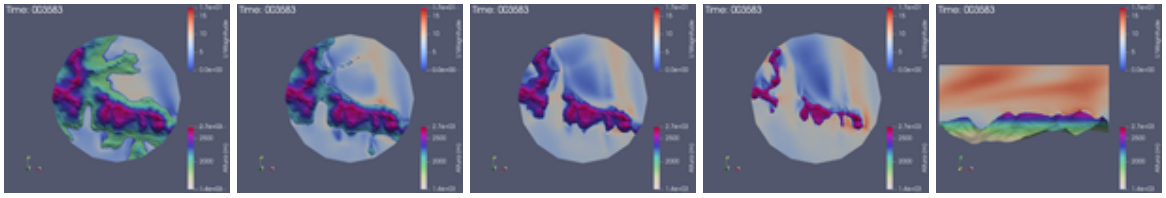
1900 m 2100 m 2300 m 2500 m Lateral Y01



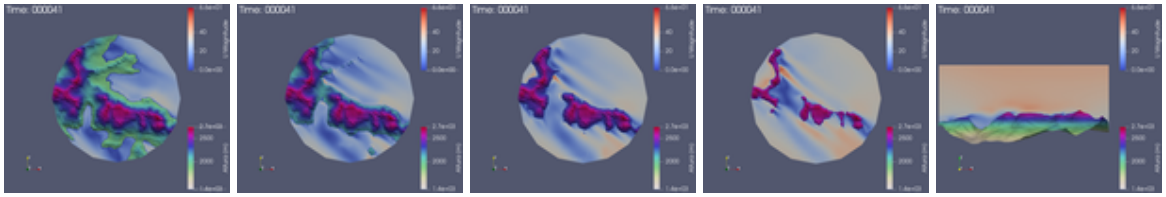
1900 m 2100 m 2300 m 2500 m Lateral Y01



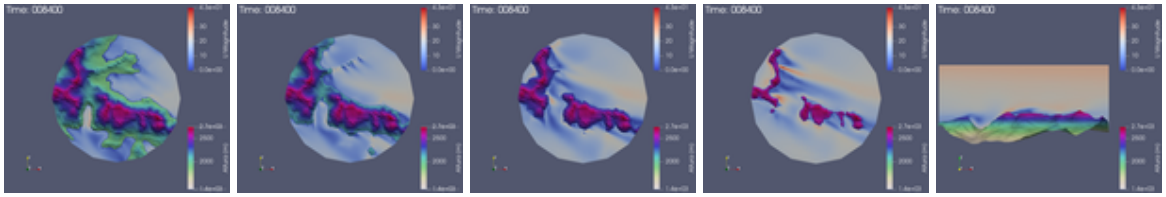
1900 m 2100 m 2300 m 2500 m Lateral Y01



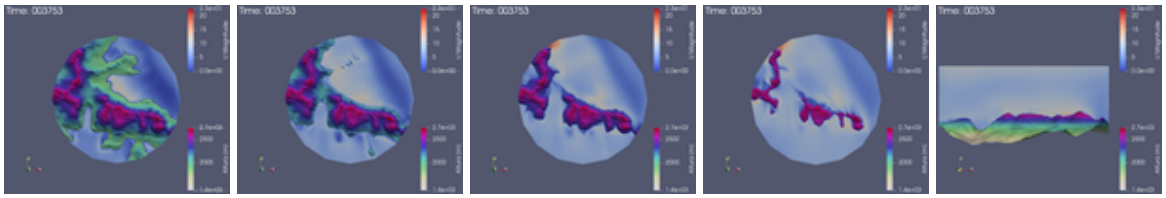
1900 m 2100 m 2300 m 2500 m Lateral Y01



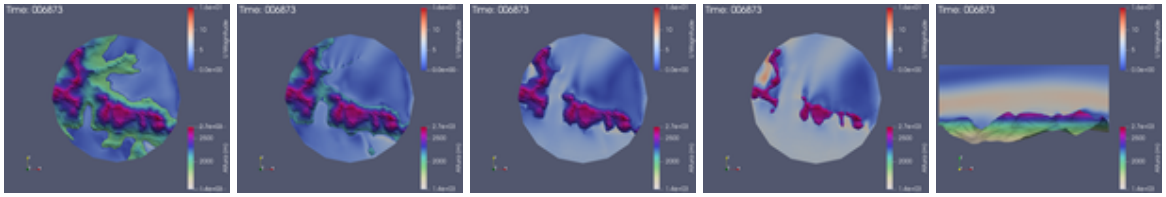
1900 m 2100 m 2300 m 2500 m Lateral Y01



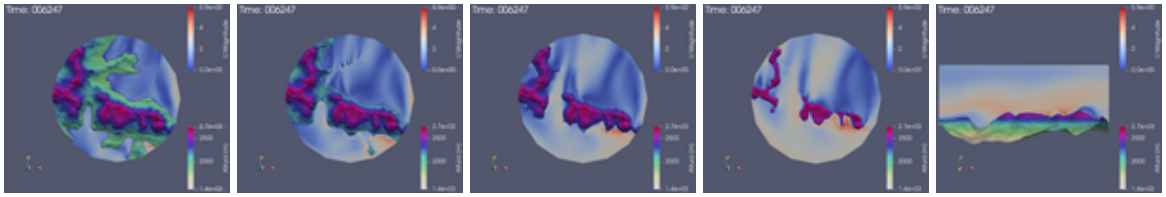
1900 m 2100 m 2300 m 2500 m Lateral Y01



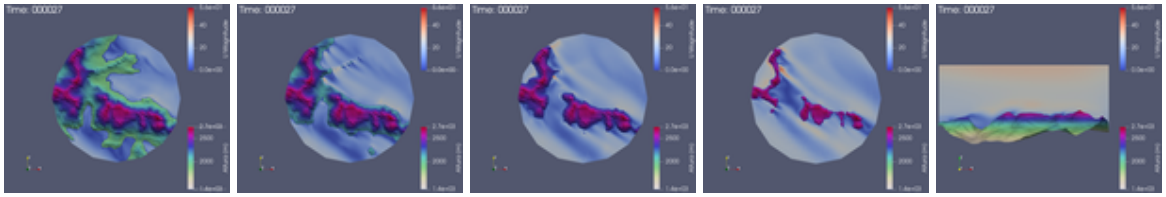
1900 m 2100 m 2300 m 2500 m Lateral Y01



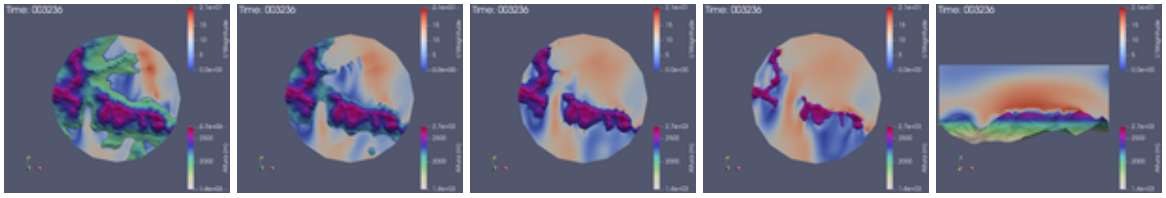
1900 m 2100 m 2300 m 2500 m Lateral Y01



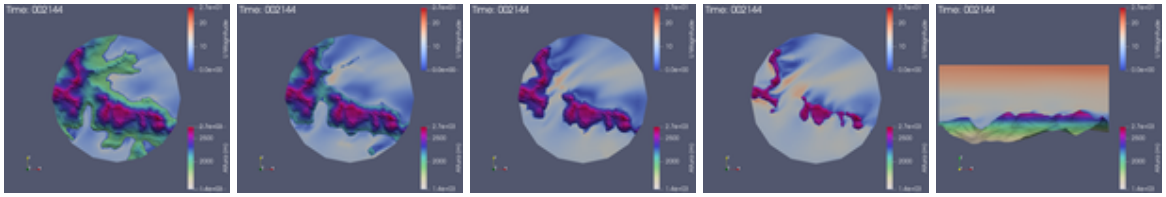
1900 m 2100 m 2300 m 2500 m Lateral Y01



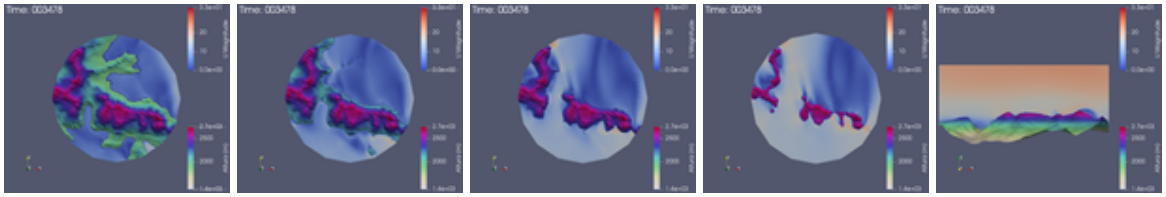
1900 m 2100 m 2300 m 2500 m Lateral Y01



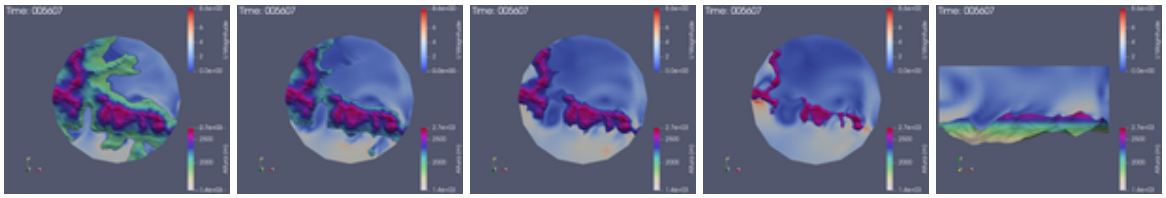
1900 m 2100 m 2300 m 2500 m Lateral Y01



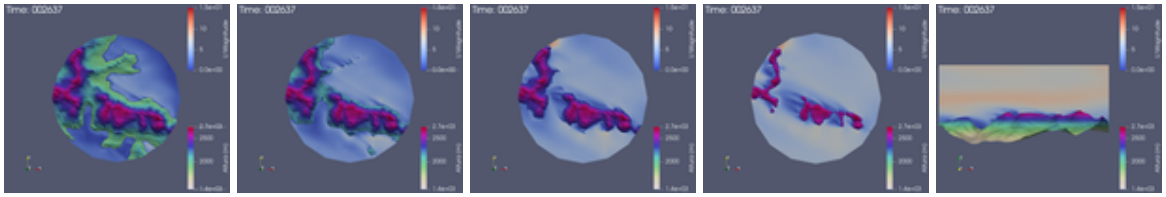
1900 m 2100 m 2300 m 2500 m Lateral Y01



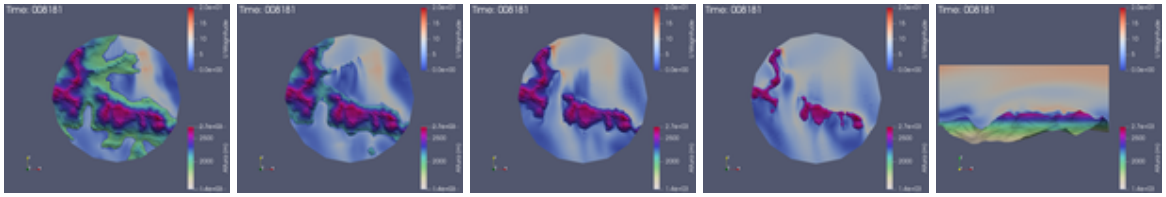
1900 m 2100 m 2300 m 2500 m Lateral Y01



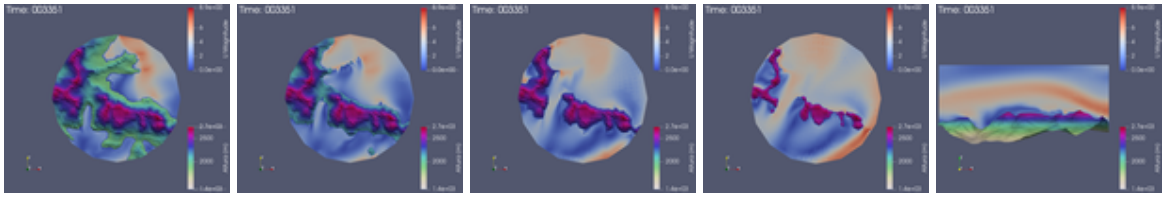
1900 m 2100 m 2300 m 2500 m Lateral Y01



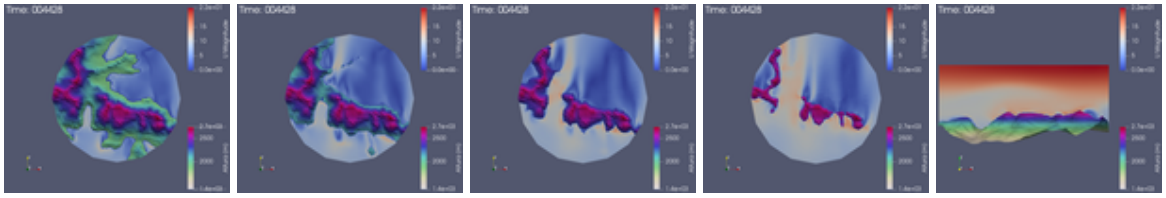
1900 m 2100 m 2300 m 2500 m Lateral Y01



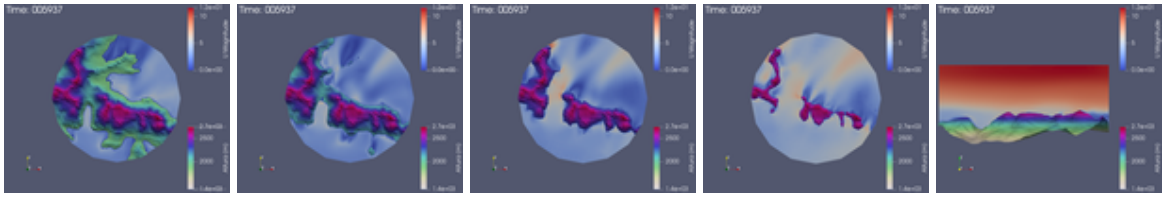
1900 m 2100 m 2300 m 2500 m Lateral Y01



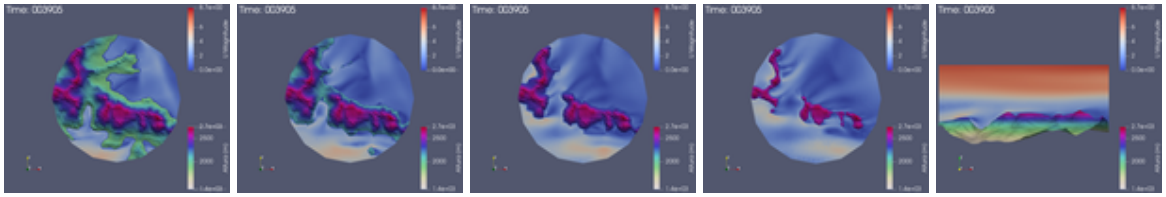
1900 m 2100 m 2300 m 2500 m Lateral Y01



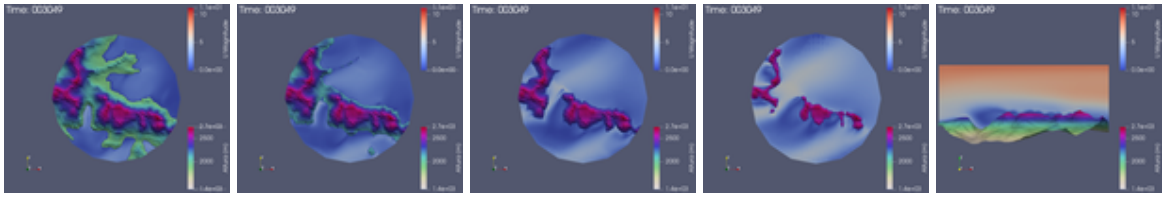
1900 m 2100 m 2300 m 2500 m Lateral Y01



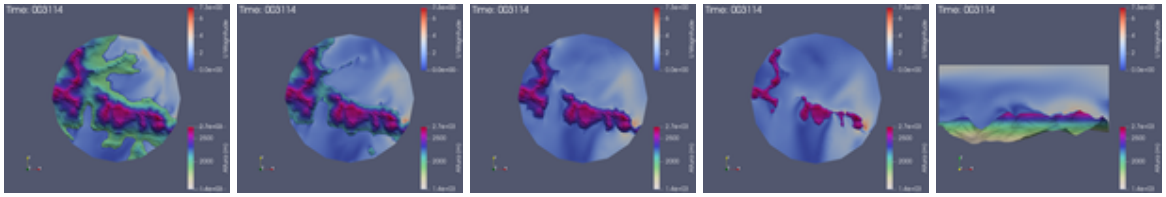
1900 m 2100 m 2300 m 2500 m Lateral Y01



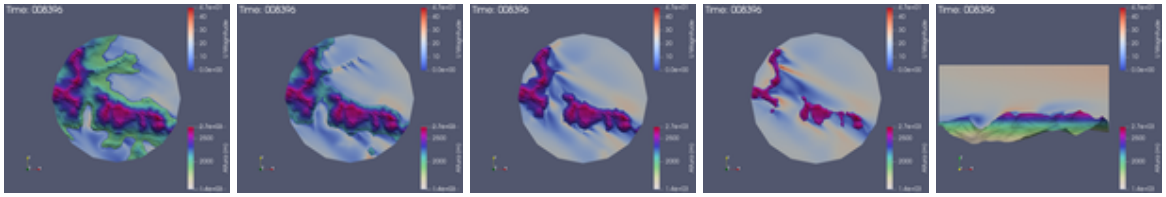
1900 m 2100 m 2300 m 2500 m Lateral Y01



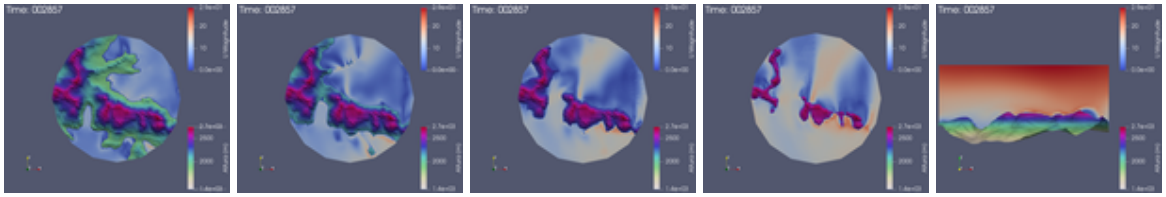
1900 m 2100 m 2300 m 2500 m Lateral Y01



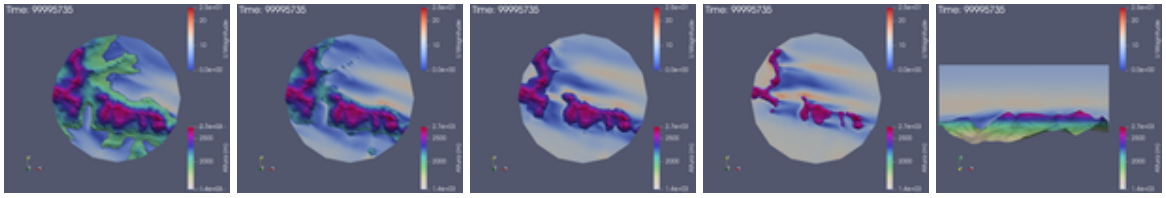
1900 m 2100 m 2300 m 2500 m Lateral Y01



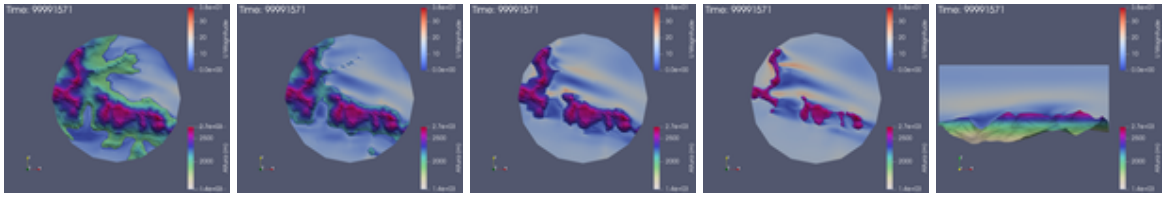
1900 m 2100 m 2300 m 2500 m Lateral Y01



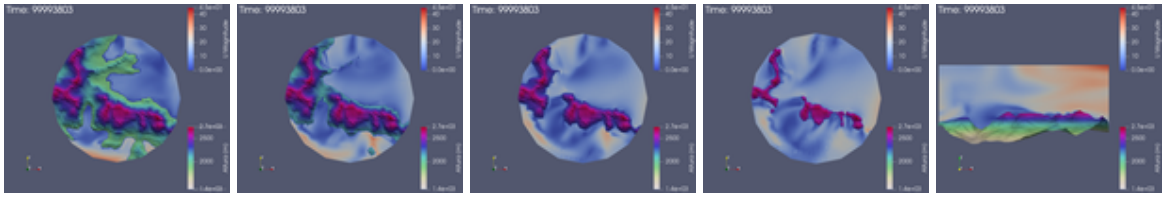
1900 m 2100 m 2300 m 2500 m Lateral Y01



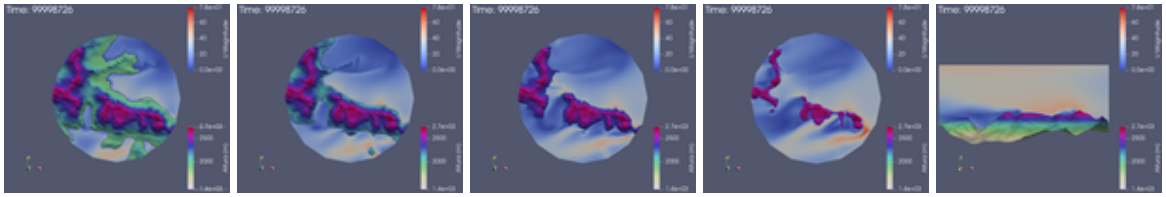
1900 m 2100 m 2300 m 2500 m Lateral Y01



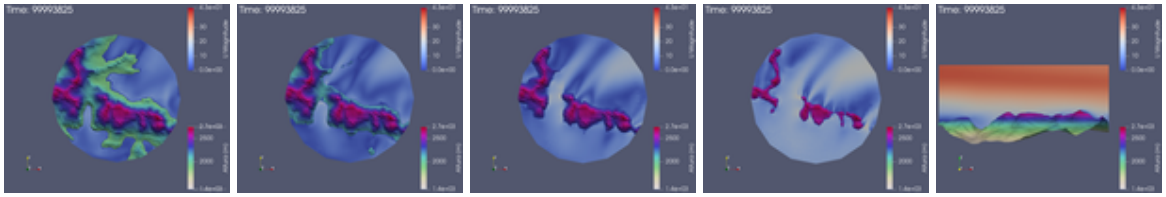
1900 m 2100 m 2300 m 2500 m Lateral Y01



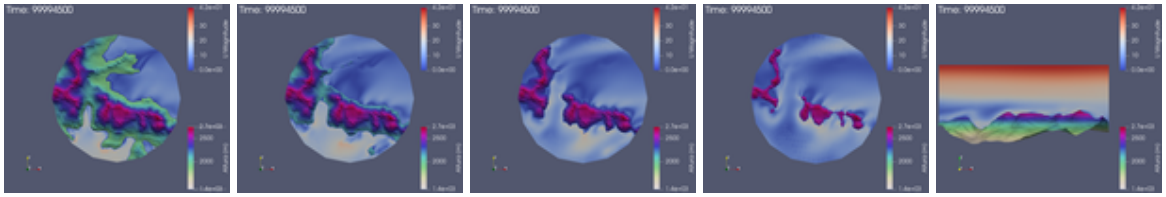
1900 m 2100 m 2300 m 2500 m Lateral Y01



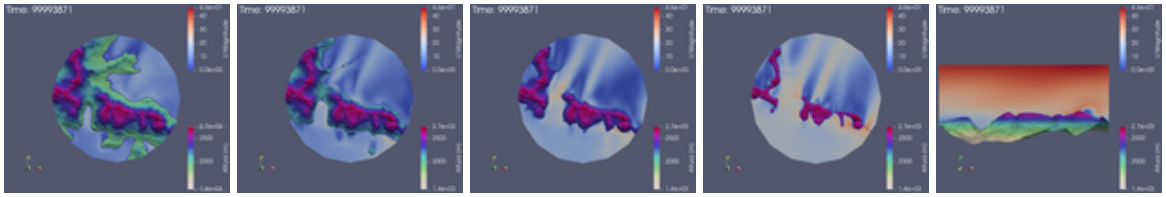
1900 m 2100 m 2300 m 2500 m Lateral Y01



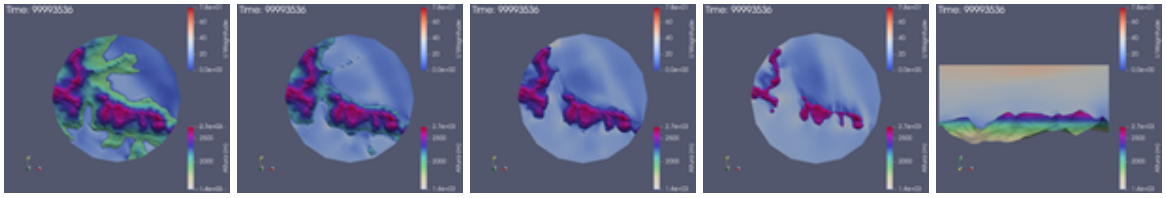
1900 m 2100 m 2300 m 2500 m Lateral Y01



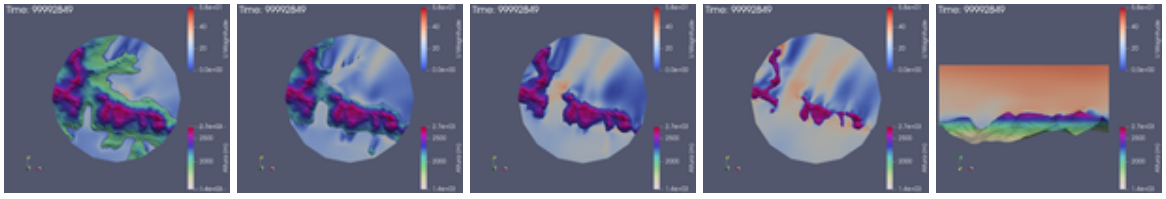
1900 m 2100 m 2300 m 2500 m Lateral Y01



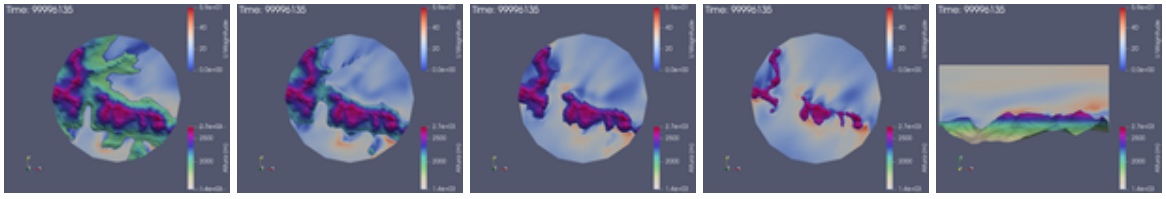
1900 m 2100 m 2300 m 2500 m Lateral Y01



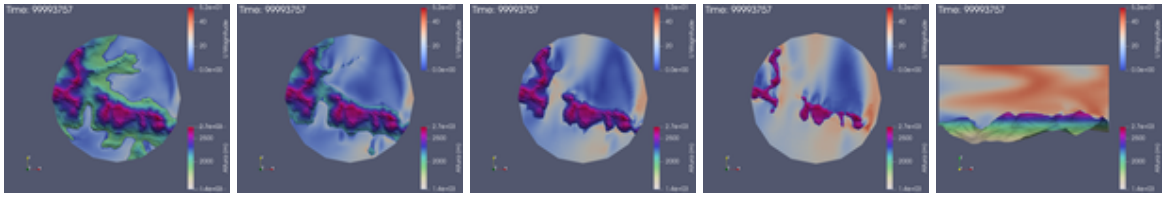
1900 m 2100 m 2300 m 2500 m Lateral Y01



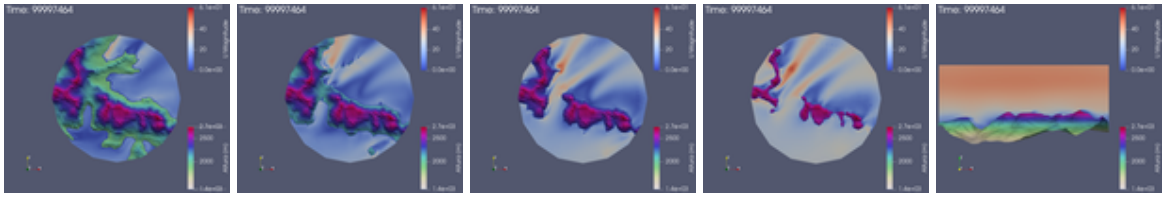
1900 m 2100 m 2300 m 2500 m Lateral Y01



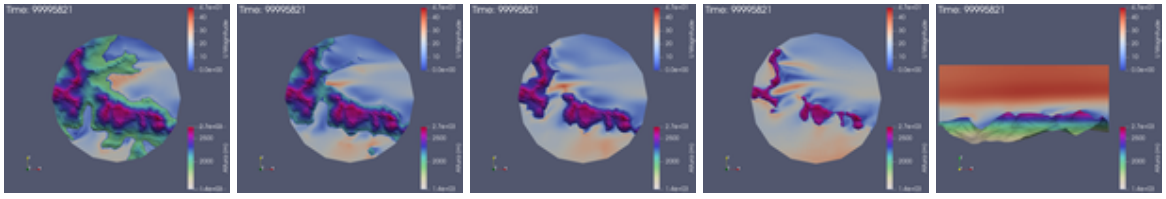
1900 m 2100 m 2300 m 2500 m Lateral Y01



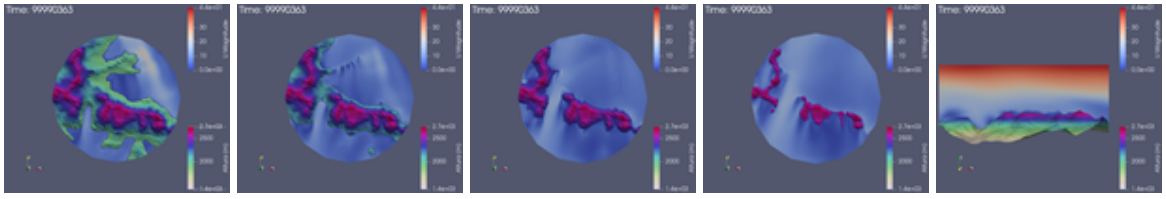
1900 m 2100 m 2300 m 2500 m Lateral Y01



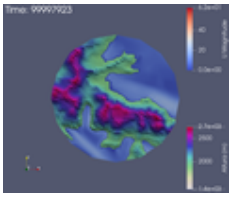
1900 m 2100 m 2300 m 2500 m Lateral Y01



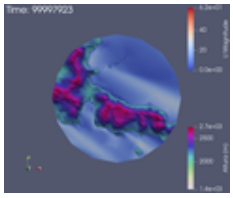
1900 m 2100 m 2300 m 2500 m Lateral Y01



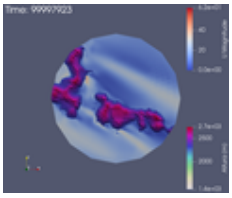
1900 m 2100 m 2300 m 2500 m Lateral Y01



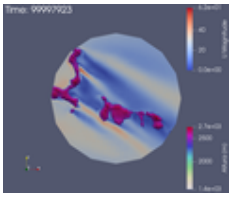
1900 m



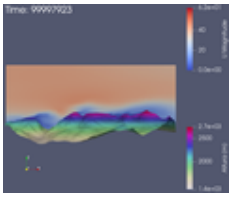
2100 m



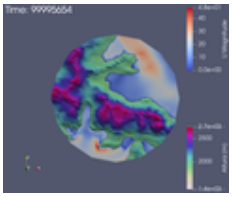
2300 m



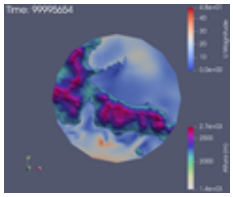
2500 m



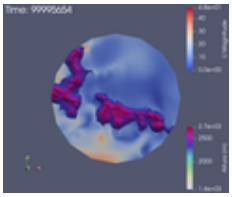
Lateral Y01



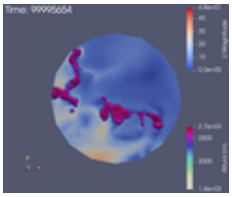
1900 m



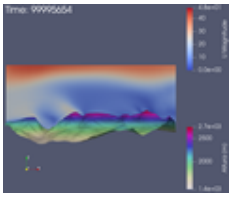
2100 m



2300 m



2500 m



Lateral Y01

Bibliografía

[1] Imran Akhtar, Osama A. Marzouk y Ali H. Nayfeh. «A van der Pol–Duffing Oscillator Model of Hydrodynamic Forces on Canonical Structures». En: *Journal of Computational and Nonlinear Dynamics* 4.4 (oct. de 2009). ISSN: 1555-1415. DOI: 10.1115/1.3192127. URL: https://asmedigitalcollection.asme.org/computationalnonlinear/article/doi/10.1115/1.3192127/424615/A-van-der-PolDuffing-Oscillator-Model-of.

[2] Imran Akhtar y col. «Hydrodynamics of a biologically inspired tandem flapping foil configuration». En: *Theoretical and Computational Fluid Dynamics* 21.3 (abr. de 2007), págs. 155-170. ISSN: 0935-4964. DOI: 10.1007/s00162-007-0045-2. URL: http://link.springer.com/10.1007/s00162-007-0045-2.

[3] Alonzo B y col. *From Numerical Weather Prediction outputs to accurate local surface wind speed : statistical modeling and forecasts*. Inf. téc. 2017.

[4] Johanna Orellana Alvear. *Árboles de decision y Random Forest*. URL: https://bookdown.org/content/2031/arboles-de-decision-parte-i.html (visitado 25-08-2020).

[5] Elsa Aristodemou y col. «How tall buildings affect turbulent air flows and dispersion of pollution within a neighbourhood». En: *Environmental Pollution* 233 (feb. de 2018), págs. 782-796. ISSN: 02697491. DOI: 10.1016/j.envpol.2017.10.041. URL: https://linkinghub.elsevier.com/retrieve/pii/S0269749117319322.

[6] Nadine Aubry. «On the hidden beauty of the proper orthogonal decomposition». En: *Theoretical and Computational Fluid Dynamics* 2.5-6 (ago. de 1991), págs. 339-352. ISSN: 0935-4964. DOI: 10.1007/BF00271473. URL: http://link.springer.com/10.1007/BF00271473.

[7] Henry P. Bakewell. «Viscous Sublayer and Adjacent Wall Region in Turbulent Pipe Flow». En: *Physics of Fluids* 10.9 (1967), pág. 1880. ISSN: 00319171. DOI: 10.1063/1.1762382. URL: https://aip.scitation.org/doi/10.1063/1.1762382.

[8] C. Bradford Barber, David P. Dobkin y Hannu Huhdanpaa. «The quickhull algorithm for convex hulls». En: *ACM Transactions on Mathematical Software (TOMS)* 22.4 (dic. de 1996), págs. 469-483. ISSN: 0098-3500. DOI: 10.1145/235815.235821. URL: http://dl.acm.org/doi/10.1145/235815.235821.

[9] D Bastine y col. «POD Analysis of a Wind Turbine Wake in a Turbulent Atmospheric Boundary Layer». En: *Journal of Physics: Conference Series* 524 (jun. de 2014), pág. 012153. ISSN: 1742-6596. DOI: 10.1088/1742-6596/524/1/012153. URL: https://iopscience.iop.org/article/10.1088/1742-6596/524/1/012153.

[10] Mark de Berg y col. *Computational Geometry*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-77973-5. DOI: 10.1007/978-3-540-77974-2. URL: http://link.springer.com/10.1007/978-3-540-77974-2.

[11] G Berkooz, P Holmes y J L Lumley. «The Proper Orthogonal Decomposition in the Analysis of Turbulent Flows». En: *Annual Review of Fluid Mechanics* 25.1 (ene. de 1993), págs. 539-575. ISSN: 0066-4189. DOI: 10.1146/annurev.fl.25.010193.002543. URL: http://www.annualreviews.org/doi/10.1146/annurev.fl.25.010193.002543.

[12] Christopher M. Bishop. «Sparse Kernel Machines». En: *Pattern recognition and machine learning*. Springer, 2006. Cap. 7, pág. 758. ISBN: 978-0387-31073-2. URL: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[13] Bert Blocken. «Computational Fluid Dynamics for urban physics: Importance, scales, possibilities, limitations and ten tips and tricks towards accurate and reliable simulations». En: *Building and Environment* 91 (sep. de 2015), págs. 219-245. ISSN: 03601323. DOI: 10.1016/j.buildenv.2015.02.015. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360132315000724.

[14] Kevin Q. Brown. «Voronoi diagrams from convex hulls». En: *Information Processing Letters* 9.5 (dic. de 1979), págs. 223-228. ISSN: 00200190. DOI: 10.1016/0020-0190(79)90074-7. URL: https://linkinghub.elsevier.com/retrieve/pii/0020019079900747.

[15] L. S. Caretto y col. «Two calculation procedures for steady, three-dimensional flows with recirculation». En: *Proceedings of the Third International Conference on Numerical Methods in Fluid Mechanics*. Berlin, Heidelberg: Springer Berlin Heidelberg, págs. 60-68. DOI: 10.1007/BFb0112677. URL: http://link.springer.com/10.1007/BFb0112677.

[16] Ian P. Castro y David D. Apsley. «Flow and dispersion over topography: A comparison between numerical and laboratory data for two-dimensional flows». En: *Atmospheric Environment* 31.6 (mar. de 1997), págs. 839-850. ISSN: 13522310. DOI: 10.1016/S1352-2310(96)00248-8. URL: https://linkinghub.elsevier.com/retrieve/pii/S1352231096002488.

[17] Ashvinkumar Chaudhari y col. «Large-Eddy Simulations for atmospheric boundary layer flows over complex terrains with applications in wind energy». En: *11th World Congress on Computational Mechanics, WCCM 2014, 5th European Conference on Computational Mechanics, ECCM 2014 and 6th European Conference on Computational Fluid Dynamics, ECFD 2014* Wccm Xi (2014), págs. 5205-5216.

[18] P. Cignoniz, C. Montaniz y R. Scopigno. «DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in Ed». En: (1997). URL: http://vcg.isti.cnr.it/publications/papers/dewall.pdf.

[19] Science Direct. *Energy Cascade*. URL: https://www.sciencedirect.com/topics/engineering/energy-cascade (visitado 05-09-2020).

[20] M. A. Donelan. «On the limiting aerodynamic roughness of the ocean in very strong winds». En: *Geophysical Research Letters* 31.18 (2004), pág. L18306. ISSN: 0094-8276. DOI: 10.1029/2004GL019460. URL: http://doi.wiley.com/10.1029/2004GL019460.

[21] Jimy Dudhia. «Numerical Study of Convection Observed during the Winter Monsoon Experiment Using a Mesoscale Two-Dimensional Model». En: *Journal of the Atmospheric Sciences* 46.20 (oct. de 1989), págs. 3077-3107. ISSN: 0022-4928. DOI: 10.1175/1520-0469(1989)046<3077:NSOCOD>2.0.CO;2. URL: http://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281989%29046%3C3077%3ANSOCOD%3E2.0.CO%3B2.

[22] Leif Enger. «Simulation of dispersion in moderately complex terrain—Part C. A dispersion model for operational use». En: *Atmospheric Environment. Part A. General Topics* 24.9 (ene. de 1990), págs. 2457-2471. ISSN: 09601686. DOI: 10.1016/0960-1686(90)90338-N. URL: https://linkinghub.elsevier.com/retrieve/pii/096016869090338N.

[23] Leif Enger y Darko Koračin. «Simulations of dispersion in complex terrain using a higher-order closure model». En: *Atmospheric Environment* 29.18 (sep. de 1995), págs. 2449-2465. ISSN: 13522310. DOI: 10.1016/1352-2310(95)00160-Z. URL: https://linkinghub.elsevier.com/retrieve/pii/135223109500160Z.

[24] European Centre for Medium-Range Weather Forecasts. *ERA-Interim Project*. Boulder CO, 2009. URL: https://doi.org/10.5065/D6CR5RD9.

[25] Joel H. Ferziger y Milovan Perić. *Computational Methods for Fluid Dynamics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. ISBN: 978-3-540-42074-3. DOI: 10.1007/978-3-642-56026-2. URL: http://link.springer.com/10.1007/978-3-642-56026-2.

[26] Eivind Fonn y col. «A step towards reduced order modelling of flow characterized by wakes using Proper Orthogonal Decomposition». En: *Energy Procedia* 137 (oct. de 2017), págs. 452-459. ISSN: 18766102. DOI: 10.1016/j.egypro.2017.10.369. URL: https://linkinghub.elsevier.com/retrieve/pii/S1876610217353468.

[27] Georg A. Grell. «Prognostic Evaluation of Assumptions Used by Cumulus Parameterizations». En: *Monthly Weather Review* 121.3 (mar. de 1993), págs. 764-787. ISSN: 0027-0644. DOI: 10.1175/1520-0493(1993)121<0764:PEOAUB>2.0.CO;2. URL: http://journals.ametsoc.org/doi/abs/10.1175/1520-0493%281993%29121%3C0764%3APEOAUB%3E2.0.CO%3B2.

[28] Georg A. Grell y Dezső Dévényi. «A generalized approach to parameterizing convection combining ensemble and data assimilation techniques». En: *Geophysical Research Letters* 29.14 (jul. de 2002), págs. 38-1-38-4. ISSN: 00948276. DOI: 10.1029/2002GL015311. URL: http://doi.wiley.com/10.1029/2002GL015311.

[29] Fernando F. Grinstein. «On integrating large eddy simulation and laboratory turbulent flow experiments». En: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367.1899 (jul. de 2009), págs. 2931-2945. ISSN: 1364-503X. DOI: 10.1098/rsta.2009.0059. URL: https://royalsocietypublishing.org/doi/10.1098/rsta.2009.0059.

[30] Weihong Guo, Xiao Liu y Xu Yuan. «Study on Natural Ventilation Design Optimization Based on CFD Simulation for Green Buildings». En: *Procedia Engineering* 121 (2015), págs. 573-581. ISSN: 18777058. DOI: 10.1016/j.proeng.2015.08.1036. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877705815027642.

[31] Steven R. Hanna y Ruixin Yang. «Evaluations of Mesoscale Models' Simulations of Near-Surface Winds, Temperature Gradients, and Mixing Depths». En: *Journal of Applied Meteorology* 40.6 (ene. de 2001), págs. 1095-1104. ISSN: 0894-8763. DOI: 10.1175/1520-0450(2001)040<1095:EOMMSO>2.0.CO;2. URL: http://journals.ametsoc.org/doi/abs/10.1175/1520-0450{\%}282001{\%}29040{\%}3C1095{\%}3AEOMMSO{\%}3E2.0.CO{\%}3B2.

[32] James R. Holton, Renata Dmowska y H. Thomas Rossby. *An Introduction to Dynamic Meteorology*. 4.ª ed. Burlington, MA: Elsevier Academic Press, 2004. ISBN: 978-0-12-354015-7.

[33] Song-You Hong, Jimy Dudhia y Shu-Hua Chen. «A Revised Approach to Ice Microphysical Processes for the Bulk Parameterization of Clouds and Precipitation». En: *Monthly Weather Review* 132.1 (ene. de 2004), págs. 103-120. ISSN: 0027-0644. DOI: 10.1175/1520-0493(2004)132<0103:ARATIM>2.0.CO;2. URL: http://journals.ametsoc.org/doi/abs/10.1175/1520-0493%282004%29132%3C0103%3AARATIM%3E2.0.CO%3B2.

[34] Song-You Hong, Yign Noh y Jimy Dudhia. «A New Vertical Diffusion Package with an Explicit Treatment of Entrainment Processes». En: *Monthly Weather Review* 134.9 (sep. de 2006), págs. 2318-2341. ISSN: 1520-0493. DOI: 10.1175/MWR3199.1. URL: https://journals.ametsoc.org/mwr/article/134/9/2318/67717/A-New-Vertical-Diffusion-Package-with-an-Explicit.

[35] Michael J. Iacono y col. «Radiative forcing by long-lived greenhouse gases: Calculations with the AER radiative transfer models». En: *Journal of Geophysical Research* 113.D13 (jul. de 2008), pág. D13103. ISSN: 0148-0227. DOI: 10.1029/2008JD009944. URL: http://doi.wiley.com/10.1029/2008JD009944.

[36] Timothy J. Baker. «Interpolation from a cloud of points». En: *IMR* (2003), págs. 55-63. URL: https://imr.sandia.gov/papers/imr12/baker03.pdf.

[37] Hrvoje Jasak, Aleksandar Jemcov y Zeljko Tukovic. «OpenFOAM : A C ++ Library for Complex Physics Simulations». En: *International Workshop on Coupled Methods in Numerical Dynamics* m.January (2007), págs. 1-20. URL: https://www.researchgate.net/publication/317267666_OpenFOAM_A_C_library_for_complex_physics_simulations.

[38] John S. Kain. «The Kain–Fritsch Convective Parameterization: An Update». En: *Journal of Applied Meteorology* 43.1 (ene. de 2004), págs. 170-181. ISSN: 0894-8763. DOI: 10.1175/1520-0450(2004)043<0170:TKCPAU>2.0.CO;2. URL: http://journals.ametsoc.org/doi/abs/10.1175/1520-0450%282004%29043%3C0170%3ATKCPAU%3E2.0.CO%3B2.

[39] Muhammad Saif Ullah Khalid y col. «Reduced-Order Modeling of Torque on a Vertical-Axis Wind Turbine at Varying Tip Speed Ratios». En: *Journal of Computational and Nonlinear Dynamics* 10.4 (jul. de 2015). ISSN: 1555-1415. DOI: 10.1115/1.4028064. URL: https://asmedigitalcollection.asme.org/computationalnonlinear/article/doi/10.1115/1.4028064/371023/ReducedOrder-Modeling-of-Torque-on-a-VerticalAxis.

[40] MMM: Mesoscale & Microscale Meteorology Laboratory. *Weather Research and Forecasting Model*. URL: https://www.mmm.ucar.edu/weather-research-and-forecasting-model (visitado 25-05-2020).

[41] M. Lateb y col. «On the use of numerical modelling for near-field pollutant dispersion in urban environments - A review». En: *Environmental Pollution* 208 (ene. de 2016), págs. 271-283. ISSN: 02697491. DOI: 10.1016/j.envpol.2015.07.039. URL: https://linkinghub.elsevier.com/retrieve/pii/S0269749115003723.

[42] Sang-Mi Lee y col. «Implementation of a Stable PBL Turbulence Parameterization for the Mesoscale Model MM5: Nocturnal Flow in Complex Terrain». En: *Boundary-Layer Meteorology* 119.1 (mayo de 2006), págs. 109-134. ISSN: 0006-8314. DOI: 10.1007/s10546-005-9018-4. URL: http://link.springer.com/10.1007/s10546-005-9018-4.

[43] Stefano Lorenzi y col. «POD-Galerkin method for finite volume approximation of Navier–Stokes and RANS equations». En: *Computer Methods in Applied Mechanics and Engineering* 311 (nov. de 2016), págs. 151-179. ISSN: 00457825. DOI: 10.1016/j.cma.2016.08.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0045782516308829.

[44] T. Marić, J. Höpken y K. G. Mooney. *The OpenFOAM Technology Primer*. 2014. DOI: 10.13140/2.1.2532.9600.

[45] Eli J. Mlawer y col. «Radiative transfer for inhomogeneous atmospheres: RRTM, a validated correlated-k model for the longwave». En: *Journal of Geophysical Research: Atmospheres* 102.D14 (jul. de 1997), págs. 16663-16682. ISSN: 01480227. DOI: 10.1029/97JD00237. URL: http://doi.wiley.com/10.1029/97JD00237.

[46] Carlos Montañes. «Development of the Flamelet-Generated Manifold model for the simulation of partially-premixed, non-adiabatic, laminar flames.» Tesis doct. Universidad de Zaragoza, 2015, pág. 150. URL: https://gfn.unizar.es/portfolio-items/development-of-the-flamelet-generated-manifold-model-for-the-simulation-of-partially-premixed-non-adiabatic-laminar-flames-by-carlos-montanes-bernal/?portfolioCats=11.

[47] H. Morrison, G. Thompson y V. Tatarskii. «Impact of Cloud Microphysics on the Development of Trailing Stratiform Precipitation in a Simulated Squall Line: Comparison of One- and Two-Moment Schemes». En: *Monthly Weather Review* 137.3 (mar. de 2009), págs. 991-1007. ISSN: 1520-0493. DOI: 10.1175/2008MWR2556.1. URL: https://journals.ametsoc.org/mwr/article/137/3/991/70576/Impact-of-Cloud-Microphysics-on-the-Development-of.

[48] Mikio Nakanishi e Hiroshi Niino. «An Improved Mellor–Yamada Level-3 Model: Its Numerical Stability and Application to a Regional Prediction of Advection Fog». En: *Boundary-Layer Meteorology* 119.2 (mayo de 2006), págs. 397-407. ISSN: 0006-8314. DOI: 10.1007/s10546-005-9030-8. URL: http://link.springer.com/10.1007/s10546-005-9030-8.

[49] Mikio NAKANISHI e Hiroshi NIINO. «Development of an Improved Turbulence Closure Model for the Atmospheric Boundary Layer». En: *Journal of the Meteorological Society of Japan* 87.5 (2009), págs. 895-912. ISSN: 0026-1165. DOI: 10.2151/jmsj.87.895. URL: http://joi.jlc.jst.go.jp/JST.JSTAGE/jmsj/87.895?from=CrossRef.

[50] National Centers for Environmental Prediction, National Weather Service, NOAA, U.S. Department of Commerce. *NCEP/NCAR Global Reanalysis Products, 1948-continuing*. Boulder CO, 1994. URL: https://rda.ucar.edu/datasets/ds090.0/.

[51] S. Omrani y col. «Natural ventilation in multi-storey buildings: Design process and review of evaluation tools». En: *Building and Environment* 116 (mayo de 2017), págs. 182-194. ISSN: 03601323. DOI: 10.1016/j.buildenv.2017.02.012. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360132317300720.

[52] OpenCFD. *OpenFOAM: User Guide: Reynolds Averaged Simulation (RAS)*. URL: https://www.openfoam.com/documentation/guides/latest/doc/guide-turbulence-ras.html (visitado 09-06-2020).

[53] OpenCFD. *OpenFOAM: User Guide: simpleFOAM*. URL: https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-incompressible-simpleFoam.html (visitado 09-06-2020).

[54] OpenFOAM. *Standard boundary conditions*. URL: https://www.openfoam.com/documentation/user-guide/standard-boundaryconditions.php (visitado 07-09-2020).

[55] J.M.L.M. Palma y col. «Linear and nonlinear models in wind resource assessment and wind turbine micro-siting in complex terrain». En: *Journal of Wind Engineering and Industrial Aerodynamics* 96.12 (dic. de 2008), págs. 2308-2326. ISSN: 01676105. DOI: 10.1016/j.jweia.2008.03.012. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167610508001037.

[56] Dora Pancheva y Plamen Mukhtarov. «Global Response of the Ionosphere to Atmospheric Tides Forced from Below: Recent Progress Based on Satellite Measurements». En: *Space Science Reviews* 168.1-4 (ene. de 2012), págs. 175-209. ISSN: 0038-6308. DOI: 10.1007/s11214-011-9837-1. URL: http://link.springer.com/10.1007/s11214-011-9837-1.

[57] F. Pedregosa y col. «Scikit-learn: Machine Learning in Python». En: *Journal of Machine Learning Research* 12 (2011), págs. 2825-2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[58] Russell Pflughaupt. *High-Dimensional Linear Data Interpolation*. Inf. téc. UCB/CSD-93-745. EECS Department, University of California, Berkeley, mayo de 1993. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/1993/6278.html.

[59] Pasha Piroozmand y col. «Coupled CFD framework with mesoscale urban climate model: Application to microscale urban flows with weak synoptic forcing». En: *Journal of Wind Engineering and Industrial Aerodynamics* 197 (feb. de 2020). ISSN: 01676105. DOI: 10.1016/j.jweia.2019.104059.

[60] Maziar Raissi, Paris Perdikaris y George E Karniadakis. «Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations». En: *Journal of Computational Physics* 378 (2019), págs. 686-707.

[61] Maziar Raissi, Paris Perdikaris y George Em Karniadakis. «Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations». En: *arXiv preprint arXiv:1711.10561* (2017).

[62] Maziar Raissi, Paris Perdikaris y George Em Karniadakis. «Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations». En: *arXiv preprint arXiv:1711.10566* (2017).

[63] F. J. Santos-Alamillos y col. «Analysis of WRF Model Wind Estimate Sensitivity to Physics Parameterization Choice and Terrain Representation in Andalusia (Southern Spain)». En: *Journal of Applied Meteorology and Climatology* 52.7 (jul. de 2013), págs. 1592-1609. ISSN: 1558-8424. DOI: 10.1175/JAMC-D-12-0204.1. URL: https://journals.ametsoc.org/jamc/article/52/7/1592/13698/Analysis-of-WRF-Model-Wind-Estimate-Sensitivity-to.

[64] Scikit-learn. *Choosing the right estimator*. URL: https://scikit-learn.org/stable/tutorial/machine{_}learning{_}map/index.html (visitado 25-08-2020).

[65] Scikit-learn. *Forests of randomized trees*. URL: https://scikit-learn.org/stable/modules/ensemble.html#forest (visitado 25-08-2020).

[66] Scikit-learn. *Nearest Neighbors*. URL: https://scikit-learn.org/stable/modules/neighbors.html (visitado 26-08-2020).

[67] Scikit-learn. *Standard Scaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (visitado 07-09-2020).

[68] Scikit-learn. *Stochastic Gradient Descent*. URL: https://scikit-learn.org/stable/modules/sgd.html#sgd-mathematical-formulation (visitado 28-08-2020).

[69] Scikit-learn. *Support Vector Machines*. URL: https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation (visitado 26-08-2020).

[70] M. Salman Siddiqui y col. «Reduced order model of offshore wind turbine wake by proper orthogonal decomposition». En: *International Journal of Heat and Fluid Flow* 82 (abr. de 2020). ISSN: 0142727X. DOI: 10.1016/j.ijheatfluidflow.2020.108554.

[71] Lawrence Sirovich. «Turbulence and the dynamics of coherent structures. I. Coherent structures». En: *Quarterly of Applied Mathematics* 45.3 (oct. de 1987), págs. 561-571. ISSN: 0033-569X. DOI: 10.1090/qam/910462. URL: http://www.ams.org/qam/1987-45-03/S0033-569X-1987-0910462-6/.

[72] W.C. Skamarock y col. «A Description of the Advanced Research WRF Model Version 4». En: *NCAR Technical Note NCAR/TN-475+STR* (2019), pág. 145. ISSN: 1477870X. DOI: 10.5065/1dfh-6p97. URL: http://library.ucar.edu/research/publish-technote.

[73] Alex J. Smola y Bernhard Schölkopf. *A tutorial on support vector regression*. 2004.

[74] Jiyun Song y col. «Natural ventilation in cities: the implications of fluid mechanics». En: *Building Research & Information* 46.8 (nov. de 2018), págs. 809-828. ISSN: 0961-3218. DOI: 10.1080/09613218.2018.1468158. URL: https://www.tandfonline.com/doi/full/10.1080/09613218.2018.1468158.

[75] P. SPALART y S. ALLMARAS. «A one-equation turbulence model for aerodynamic flows». En: *30th Aerospace Sciences Meeting and Exhibit*. Reston, Virigina: American Institute of Aeronautics y Astronautics, 1992. DOI: 10.2514/6.1992-439. URL: http://arc.aiaa.org/doi/10.2514/6.1992-439.

[76] Giovanni Stabile, Hermann G. Matthies y Claudio Borri. «A novel reduced order model for vortex induced vibrations of long flexible cylinders». En: *Ocean Engineering* 156 (mayo de 2018), págs. 191-207. ISSN: 00298018. DOI: 10.1016/j.oceaneng.2018.02.064. URL: https://linkinghub.elsevier.com/retrieve/pii/S0029801818302130.

[77] Giovanni Stabile y col. «POD-Galerkin reduced order methods for CFD using Finite Volume Discretisation: vortex shedding around a circular cylinder». En: *Communications in Applied and Industrial Mathematics* 8.1 (dic. de 2017), págs. 210-236. ISSN: 2038-0909. DOI: 10.1515/caim-2017-0011. URL: https://content.sciendo.com/view/journals/caim/8/1/article-p210.xml.

[78] A. Tallet y col. «A minimum residual projection to build coupled velocity–pressure POD–ROM for incompressible Navier–Stokes equations». En: *Communications in Nonlinear Science and Numerical Simulation* 22.1-3 (mayo de 2015), págs. 909-932. ISSN: 10075704. DOI: 10.1016/j.cnsns.2014.09.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S1007570414004535.

[79] Orkun Temel, Laurent Bricteux y Jeroen van Beeck. «Coupled WRF-OpenFOAM study of wind flow over complex terrain». En: *Journal of Wind Engineering and Industrial Aerodynamics* 174 (mar. de 2018), págs. 152-169. ISSN: 01676105. DOI: 10.1016/j.jweia.2018.01.002.

[80] Orkun Temel y col. «RANS closures for non-neutral microscale CFD simulations sustained with inflow conditions acquired from mesoscale simulations». En: *Applied Mathematical Modelling* 53 (ene. de 2018), págs. 635-652. ISSN: 0307904X. DOI: 10.1016/j.apm.2017.09.018.

[81] Gregory Thompson y col. «Explicit Forecasts of Winter Precipitation Using an Improved Bulk Microphysics Scheme. Part II: Implementation of a New Snow Parameterization». En: *Monthly Weather Review* 136.12 (dic. de 2008), págs. 5095-5115. ISSN: 1520-0493. DOI: 10.1175/2008MWR2387.1. URL: https://journals.ametsoc.org/mwr/article/136/12/5095/68204/Explicit-Forecasts-of-Winter-Precipitation-Using.

[82] Jonathan Tompson y col. «Accelerating Eulerian Fluid Simulation With Convolutional Networks». En: (jul. de 2016). arXiv: 1607.03597. URL: http://arxiv.org/abs/1607.03597.

[83] Abraham Albert Ungar. *Barycentric Calculus in Euclidean and Hyperbolic Geometry*. WORLD SCIENTIFIC, ago. de 2010. ISBN: 978-981-4304-93-1. DOI: 10.1142/7740. URL: https://www.worldscientific.com/worldscibooks/10.1142/7740.

[84] H. Versteeg y W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method Approach*. Prentice-Hall, 1995, pág. 272. ISBN: 978-0582218840.

[85] Alan Viladegut Farran. «Assessment of gas-surface interaction modelling for lifting body re-entry flight design». En: *TDX (Tesis Doctorals en Xarxa)* (dic. de 2017). URL: http://84.88.27.106/handle/10803/461893.

[86] Pauli Virtanen y col. «SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python». En: *Nature Methods* 17 (2020), págs. 261-272. DOI: https://doi.org/10.1038/s41592-019-0686-2.

[87] Marcel Vonlanthen, Jonas Allegrini y Jan Carmeliet. «Multiscale interaction between a cluster of buildings and the ABL developing over a real terrain». En: *Urban Climate* 20 (jun. de 2017), págs. 1-19. ISSN: 22120955. DOI: 10.1016/j.uclim.2017.02.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S2212095517300160.

[88] H. G. Weller y col. «A tensorial approach to computational continuum mechanics using object-oriented techniques». En: *Computers in Physics* 12.6 (1998), pág. 620. ISSN: 08941866. DOI: 10.1063/1.168744. URL: http://scitation.aip.org/content/aip/journal/cip/12/6/10.1063/1.168744.

[89] K. Willcox y J. Peraire. «Balanced Model Reduction via the Proper Orthogonal Decomposition». En: *AIAA Journal* 40.11 (nov. de 2002), págs. 2323-2330. ISSN: 0001-1452. DOI: 10.2514/2.1570. URL: https://arc.aiaa.org/doi/10.2514/2.1570.

[90] Nigel Wood. «Wind Flow Over Complex Terrain: A Historical Perspective and the Prospect for Large-Eddy Modelling». En: *Boundary-Layer Meteorology* 96.1-2 (ago. de 2000), págs. 11-32. ISSN: 0006-8314. DOI: 10.1023/A:1002017732694. URL: http://link.springer.com/10.1023/A:1002017732694.

[91] XGBoost. *Introduction to Boosted Trees*. URL: https://xgboost.readthedocs.io/en/latest/tutorials/model.html (visitado 25-08-2020).

[92] D. Xiao y col. «A reduced order model for turbulent flows in the urban environment using machine learning». En: *Building and Environment* 148 (ene. de 2019), págs. 323-337. ISSN: 03601323. DOI: 10.1016/j.buildenv.2018.10.035.

[93] Tetsuji Yamada y Katsuyuki Koike. «Downscaling mesoscale meteorological models for computational wind engineering applications». En: *Journal of Wind Engineering and Industrial Aerodynamics* 99.4 (abr. de 2011), págs. 199-216. ISSN: 01676105. DOI: 10.1016/j.jweia.2011.01.024.

[94] Frank J. Zajaczkowski, Sue Ellen Haupt y Kerrie J. Schmehl. «A preliminary study of assimilating numerical weather prediction data into computational fluid dynamics models for wind prediction». En: *Journal of Wind Engineering and Industrial Aerodynamics* 99.4 (abr. de 2011), págs. 320-329. ISSN: 01676105. DOI: 10.1016/j.jweia.2011.01.023.

[95] Tong Zhang. «Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms». En: *ICML 2004: PROCEEDINGS OF THE TWENTY-FIRST INTERNATIONAL CONFERENCE ON MACHINE LEARNING. OMNIPRESS*. 2004, págs. 919-926.