Universidad Internacional de La Rioja
Escuela Superior de Ingeniería y Tecnología
Máster en Ingeniería Matemática y Computación

Procesos Gaussianos aplicados al estudio de gases magnetizados

Trabajo Fin de Máster

Presentado por: Ortiz López, Diego Fernando

Director/a: Luna Molina, Francisco Javier

Ciudad: Quito, Ecuador
Fecha: Septiembre 2019

Resumen

El coste computacional para generar campos escalares y vectoriales con los métodos de simulación actuales es muy alto, debido a que se utiliza principalmente el método de diferencias finitas que discretiza y resuelve las ecuaciones de Navier-Stokes en forma diferencial. Por tanto, obtener una solución convergente requiere, por lo general, realizar simulaciones que usan multiprocesadores.

Por este motivo, el presente trabajo busca desarrollar e implementar métodos que permitan generar instancias específicas de la evolución de los gases magnetizados a un menor costo computacional a partir del empleo de los Procesos Gaussianos (en adelante, GP) como una herramienta para generar campos escalares y vectoriales. Para este fin, se programa un código para describir gases magnetizados con características de divergencia y rotacional nulo, aprovechando las propiedades de los GP.

En primer lugar, se propone una metodología que permite comprender el desarrollo del código partiendo del estudio de las propiedades de los GP y la importancia del *Kernel* para describir distintas propiedades. A continuación, se estudian los campos escalares y se extiende su esquema a campos vectoriales en $\mathbb{R}^2$. Finalmente, se desarrollan los *Kernels* que cumplen las propiedades de divergencia nula y rotacional nulo.

Los resultados muestran que los campos generados cumplen con las condiciones de divergencia nula y rotacional nulo. Además, los resultados son comparados con simulaciones generadas con el código *PLUTO*. En lo principal, el código generado en el presente trabajo disminuye prácticamente a la mitad el tiempo en el que se generan las simulaciones *PLUTO*

Palabras Clave: Procesos Gaussianos, *Kernel*, Campos vectoriales, divergencia, rotacional, campos magnéticos.

Abstract

The computational cost to generate scalar and vector fields with the current simulation methods is very high, because it mainly uses the finite difference method that discretizes and solves the Navier-Stokes equations in a differential form. Therefore, obtaining a convergent solution generally requires simulations using multiprocessors.

For this reason, the present work seeks to develop and implement methods that can generate specific instances of the evolution of magnetized gases at a lower computational from the use of the Gaussian Processes (GP) as a tool to generate scalar and vector fields. For this purpose, a code is programmed to describe magnetized gases with divergence free and curl free, taking advantage of the properties of GP.

Firstly, a methodology has been proposed to understand the development of the code based on the study of the properties of the GP and the importance of the Kernel to describe different properties. After that, scalar fields have been studied and their scheme have been extended to vector fields in $\mathbb{R}2$. Finally, Kernel has been developed so that satisfies the properties of divergence free and curl free.

The results show that the generated fields fulfill the divergence and curl conditions. Finally, the results are compared with simulations from PLUTO's code. In the main, the code generated in the present work almost halves the time in which the PLUTE simulations are generated.

Keywords: Gaussian processes, *Kernel*, vector fields, divergence, curl, magnetic fields.

AGRADECIMIENTOS

Me gustaría agradecer a todas las personas que han tomado parte para que este trabajo se haya podido llevar a cabo. Agradecerles por su interés, esfuerzo y buenos deseos. Espero sepan que me siento muy agradecido por toda su paciencia, aprecio, enseñanzas y por sobre todo su amistad.

Citar en primer lugar a Federico Zertuche y Wladimir Banda, por su tiempo, interés, dedicación y paciencia para que este trabajo se realice de manera adecuada y cumpliendo con los objetivos planteados para la investigación.

Luego, quiero dedicar una líneas para mis amigos que han sido soporte con sus comentarios, sus bromas y sobre todo preocupándose por mí. Por todo lo que han aportado y criticado: Edwin Quizhpi, Wilson Arroyo y Cristian Aguirre.

Finalmente agradezco a los seres que me dieron la vida y que han apoyado cada paso tomado para mi formación tanto espiritual como académica; entregando todo su amor y dedicación para mí, Eduardo y Blanquita, mis padres. Citar a mis hermanos, Edison y Sandra, siempre recordarles el apoyo que son y que seré para ustedes. También quiero dedicar este trabajo a mis sobrinos Rihanna, Nicolas y Jeremías; espero que su curiosidad los lleve tan lejos como ustedes deseen.

Contenido

Índice de figuras

Capítulo 1. Introducción

Calentar un gas puede ionizar sus moléculas o átomos convirtiéndolo en un plasma. Es decir, los plasmas son gases ionizados que interactúan con campos electromagnéticos. Este es el estado de agregación de la materia que está presente en, prácticamente, todo el universo de manera abundante (Gibbon, 2017).

Existen dos tipos de modelos fundamentales para el estudio del plasma: los modelos de fluidos o hidrodinámicos en los que se supone que el plasma se comporta como un fluido ideal (Skiba, 2008) y los modelos cinéticos donde se consideran las interacciones a escalas infinitesimales (Chen, 1984).

En la actualidad, existen varias herramientas numéricas que permiten modelar la dinámica y evolución de plasmas o gases magnetizados. Entre estas herramientas están las simulaciones de dinámica de fluidos (CFD, Computational Fluid Dynamics)(Federrath et al., 2011), las simulaciones de Monte Carlo (Lewis & Austin, 1996) y los estudios basados en análisis de Fourier (E., Shaaban, El-Naggar, & Shaaban, 2011).

Las CFD son las herramientas más empleadas para modelar la dinámica y evolución de plastas o gases magnetizados, de entre las comentadas anteriormente, ya que en ellas se combinan elementos físicos con métodos numéricos. En estas simulaciones se utiliza principalmente el método de diferencias finitas que discretiza y resuelve las ecuaciones de Navier-Stokes en forma diferencial. Sin embargo, obtener una solución convergente requiere, por lo general, realizar simulaciones que usan multiprocesadores y esto provoca que dichas simulaciones sean computacionalmente costosas. Por este motivo, es necesario implementar métodos que permitan generar instancias específicas de la evolución de los gases magnetizados a un menor costo computacional.

En este contexto, en el presente trabajo se recurre en a los Procesos Gaussianos (GP, Gaussian Processes) ya que estos permiten generar campos vectoriales y/o escalares que representen estados específicos de los gases magnetizados sin la necesidad de realizar una simulación CFD. Un GP permite definir una distribución de probabilidades sobre un espacio de funciones específico. Las características de las funciones de este espacio están completamente descritas por el *Kernel* del proceso (Alvarez, 2011; Alvarez, Rosasco, & Lawrence, 2012).

Los GP son utilizados actualmente en campos del conocimiento que, en general, tienen en común la hipótesis de no tener acceso a información secundaria o complementaria del fenómeno y que, por tanto, son campos en los que no se pueden aplicar modelos para realizar predicciones individuales para cada variable (Jidling, Wahlström, Wills, & Schön, 2017). Por ejemplo, en teoría electromagnética, las fuentes magnéticas en entornos complejos se pueden modelizar a través de un GP que explota las propiedades de divergencia nula y rotacional nulo en los campos magnéticos (Wahlström, Kok, Schön, & Gustafsson, 2013). Los GP también son usados en el monitoreo de fenómenos espaciales (temperatura y precipitación) con redes de sensores inalámbricos, ya que permiten determinar la posición óptima de dichos sensores, tal que sean informativos y se comuniquen de manera eficiente entre sí (Krause, Guestrin, Gupta, & Kleinberg, 2008).

1.1 Motivación

Como se explica en la sección anterior, los GP tienen aplicación en diferentes áreas de la ciencia y tecnología, dado que son útiles para crear campos vectoriales y escalares con los que se pueden representar estados específicos en la evolución de los gases magnetizados, sin la necesidad de resolver ecuaciones de magnetohidrodinámica.

Adicionalmente, al poder instanciar una de las etapas de la evolución de los gases magnetizados, los GP se convierten en generadores de condiciones iniciales para simulaciones y construcciones de modelos más complejos en los que encontrar una solución analítica es casi imposible; por ejemplo, si un proceso requiere una simulación de CFD en medios turbulentos, las condiciones iniciales se tomarían del resultado generado por los GP.

En el presente trabajo se busca programar dos tipos de GP a través de la definición de distribuciones sobre los espacios de campos vectoriales con divergencia y rotacional nulos, restricciones motivadas por propiedades físicas que adquieren los gases que interactúan en campos magnéticos.

Además, con el presente trabajo se pretende producir una herramienta (código) que permita generar campos escalares y vectoriales, en estado estacionario, que describan gases turbulentos en medios incomprensibles, campos magnéticos con divergencia nula y flujos altamente compresivos con rotacional nulo.

1.2 Planteamiento del trabajo

En el presente trabajo se aborda el estudio de gases magnetizados a través de la implementación de una herramienta alternativa que permite analizar las características de este fenómeno y considerar diferentes escenarios mediante la utilización de GP, ya que estos son un recurso apropiado para simular algunas instancias de la evolución de los gases magnetizados e incluso sus resultados pueden considerarse como condiciones iniciales para realizar otras simulaciones.

La herramienta se implementa en la librería *GPY* de *Python*, que es un lenguaje de programación utilizado a nivel científico, libre de licencias y que permite la contribución constante de varios investigadores. La herramienta implementada se centra en el cálculo de un *kernel* que, al ser una medida de similitud entre objetos, permite crear estimaciones de funciones de densidad que a su vez son la base de los GP.

1.2.1 Estructura del trabajo

El presente trabajo se divide en 5 capítulos. El primer capítulo consiste en la introducción. En el capítulo 2 se presenta el estado del arte con el desarrollo teórico de los GP, sus principales aplicaciones y la extensión a otros campos del conocimiento. En el capítulo 3, se plantean los objetivos y se describe la metodología utilizada. En el capítulo 4 se expone el desarrollo de la contribución mostrando todos los procesos realizados para lograr los objetivos planteados en el TFM. Finalmente, en el capítulo 5 se muestran las conclusiones y las líneas de investigaciones futuras que se han identificado durante y tras el desarrollo del trabajo.

Capítulo 2. Contexto y Estado del Arte

En la actualidad existen varias técnicas que permiten la simulación de gases que interactúan con campos magnéticos. En el caso ideal, se tienen modelos que se basan en simulaciones de dinámica de fluidos (denominadas simulaciones magnetohidrodinámicas) que requieren elevados costos computacionales, ya que es necesario resolver las ecuaciones de Euler o Navier-Stokes combinadas con las ecuaciones de Maxwell y se requiere establecer condiciones iniciales y de frontera que dependen del problema en estudio; a esto se lo conoce como un problema de Riemann (Soler & Gamazo, 2015).

El presente trabajo se enfoca en desarrollar una herramienta a partir de técnicas numéricas basadas en GP (Duvenaud, 2014) y no de técnicas CFD, que permite construir, visualizar y describir campos con condiciones conocidas que pueden, posteriormente, usarse para estudios de instancias específicas de la evolución de gases y campos magnéticos, sin necesidad de desarrollar simulaciones multiprocesador.

A continuación, se definen algunos conceptos necesarios para facilitar la comprensión del presente trabajo.

2.1 Procesos Estocásticos

Las instancias de evolución de las interacciones entre gases y campos magnéticos pueden considerarse como sistemas en estado estacionario representados por un conjunto de variables de estado. Estos sistemas evolucionan en el tiempo de acuerdo a leyes de movimiento que, en el caso de estudio, vienen dadas por las ecuaciones de conservación de masa, momento lineal, energía y flujo magnético. Por tanto, se define a X_t como el estado del sistema al tiempo t. Así, el sistema es una colección de variables aleatorias en la que dichas variables no son independientes entre sí. Esta colección de variables aleatorias es la definición de proceso estocástico y las distintas formas en la que pueden relacionarse las variables unas con otras son lo que distinguen a los procesos entre sí.

Los conceptos desarrollados en la presente sección se basan en Žitković et (2010) y Rincón (2012). Formalmente, un *proceso estocástico* es una colección de variables aleatorias $\{X_t : t \in T\}$ parametrizada por un conjunto T, llamado *espacio parametral*, en donde las variables toman valores en un conjunto S llamado *espacio de estados*.

El espacio T puede tomar valores discretos o continuos, definiendo de esta manera procesos a tiempo discreto o continuo, respectivamente. El *espacio de estados* S puede ser subconjunto de $\mathbb{R}^n$; sin embargo, también se pueden definir espacios más generales dependiendo del problema estudiado.

Un *proceso estocástico* también puede definirse como una función aleatoria:

$$X: T \times \Omega \to S \tag{1}$$

Tal que a la pareja (t, ω) se le asocia el *valor o estado* $X(t, \omega)$ y que usualmente se escribe como $X_t(\omega)$, lo que indica que para valor de $t \in T$ el valor de $\omega \to X_t(\omega)$ es una variable aleatoria, mientras que para un $\omega \in \Omega$ fijo, la función $t \to X_t(\omega)$ es llamada *trayectoria o realización del proceso*.

La aplicación de los procesos estocásticos dependerá de las consideraciones que se tomen para el espacio T, el espacio de estados S, las características de las trayectorias, las relaciones y características entre las distintas variables aleatorias que definen los diferentes tipos de procesos estocásticos.

A continuación, se definen la *media* y la *autocovarianza* de un proceso estocástico, respectivamente:

$$m_X(t) = E(X_t) \tag{2}$$

$$\Gamma_X\,(r, t) = Cov(X_r, X_t) = E\left(\big(X_r - m_X(r)\big)\big(X_t - m_X\,(t)\big)\right) \tag{3}$$

La varianza del proceso X se define por:

$$\sigma_X^2(t) = \Gamma_X(t, t) = Var(X_t) \tag{4}$$

Se dice que un proceso estocástico a tiempo continuo $\{X_t: t \geq 0\}$ es un GP si para cualquier colección finita de tiempos $t_1, \dots, t_n$ el vector $\left(X_{t_1}, \dots, X_{t_n}\right)$ tiene una distribución normal o Gaussiana Multivariante.

En un GP, la media $m_X(t)$ y la autocovarianza $\Gamma_X(r,t)$ determinan las distribuciones del proceso. En el caso del presente estudio, la evolución en el tiempo viene dada por las leyes de conservación mencionadas anteriormente. Por lo que este tipo de procesos están definidos por las funciones de media y covarianzas. El movimiento Browniano es un ejemplo de este tipo de procesos (Rincón, 2012).

2.2 Proceso Gaussiano (GP)

Para cualquier conjunto T, un GP es un conjunto de variables aleatorias $(Z_t: t \in T)$ tal que $\forall n \in \mathbb{N}, \forall t_1 \dots, t_n \in T$ el vector $\left(Z_{t_1}, \dots, Z_{t_n}\right)$ es gaussiano.

Es decir, un GP es una generalización de la función de distribución de probabilidad gaussiana (Fairbrother, Nemeth, Rischard, & Brea, 2018). Mientras que una distribución de probabilidad describe variables aleatorias que son escalares o vectoriales, un GP es una distribución sobre el espacio de funciones $f(x)$ definido por las funciones de media $\left(\mathbb{E}[f(x)] = \mu(x)\right)$ y la matriz de varianzas-covarianzas del proceso $(cov\,[f(x), f(x')] = K(x,x') = \Sigma$ denominándose *Kernel* a la función $K(x,x')$).

Existen distintas formas de notación para un GP; sin embargo, en el presente trabajo se utilizan indistintamente las siguientes:

$$f \sim \mathcal{GP}(\mu, K)\ o\ f \sim \mathcal{GP}(\mu, \Sigma) \qquad (5)$$

Los GP son ampliamente utilizados en regresión (Alvarez, 2011), clasificación (Sarmiento, Fondón, Velasco, Qaisar, & Aguilera, 2014,), filtrado e interpolación de datos (Krause et al., 2008). La idea clave detrás de los GP es que una función se puede modelar utilizando una distribución Gaussiana multivariante infinitamente dimensional. En otras palabras, cada punto en el espacio de entrada está asociado con una variable aleatoria y la distribución conjunta de estos se modela como un GP multivariante (Alvarez, 2011; Fairbrother et al., 2018; Sarmiento et al., 2014).

A diferencia de muchas técnicas, los GP proporcionan información sobre la distribución de los posibles resultados (Duvenaud, 2014), en lugar de una única estimación de máxima verosimilitud. También son capaces de producir modelos que recogen características del fenómeno estudiado mediante las funciones $\mu(x)$ y $K(x,x')$.

En la práctica, se fija el valor de μ en cero y se trabaja con la función de covarianzas K, que es típicamente una función simétrica y no negativa (i.e. $K(x, x') \geq 0$) y puede ser interpretada como una medida de similitud entre dos puntos del espacio de características similares (Murphy, 2012).

Duvenaud (2014), menciona que la estructura que se desea capturar por un GP está completamente determinada por su *kernel*, puesto que en este se generaliza el modelo y permite extrapolar a nuevos datos.

Debido a que el presente trabajo se centra en el diseño de un *kernel* que recoja las características del fenómeno estudiado, a continuación se presentan las definiciones necesarias para el desarrollo de este.

2.2.1 *Kernel*

Un *kernel* (también llamado función de covarianza, función de *kernel* o *kernel* de covarianza), es una función definida positiva de dos entradas x y x' (vectores definidos en un espacio euclídeo); sin embargo, un *kernel* también se puede definir mediante gráficos, imágenes, entradas discretas o categóricas e incluso texto.

Los GP utilizan el *kernel* para definir la covarianza entre dos funciones (Duvenaud, 2014).

$$cov[f(x), f(x')] = k(x, x') \quad (6)$$

2.2.2 El *kernel* para expresar estructuras

En el *kernel,* el investigador coloca las hipótesis del modelo a estimar. Por ejemplo, se puede suponer que la función objeto de estudio posee infinitas derivadas, es decir, f es de clase C^{∞}, o que tiene alguna periodicidad o tendencia (Duvenaud, 2014). Por lo que, para codificar las características de la función objeto de estudio se utilizará un tipo de *kernel* específico.

Existen muchas formas de definir un *kernel* (Manning, Raghavan, & Schütze, 2008). En la Figura 1, se presentan los *kernels* más utilizados para procesos de regresión y clasificación.

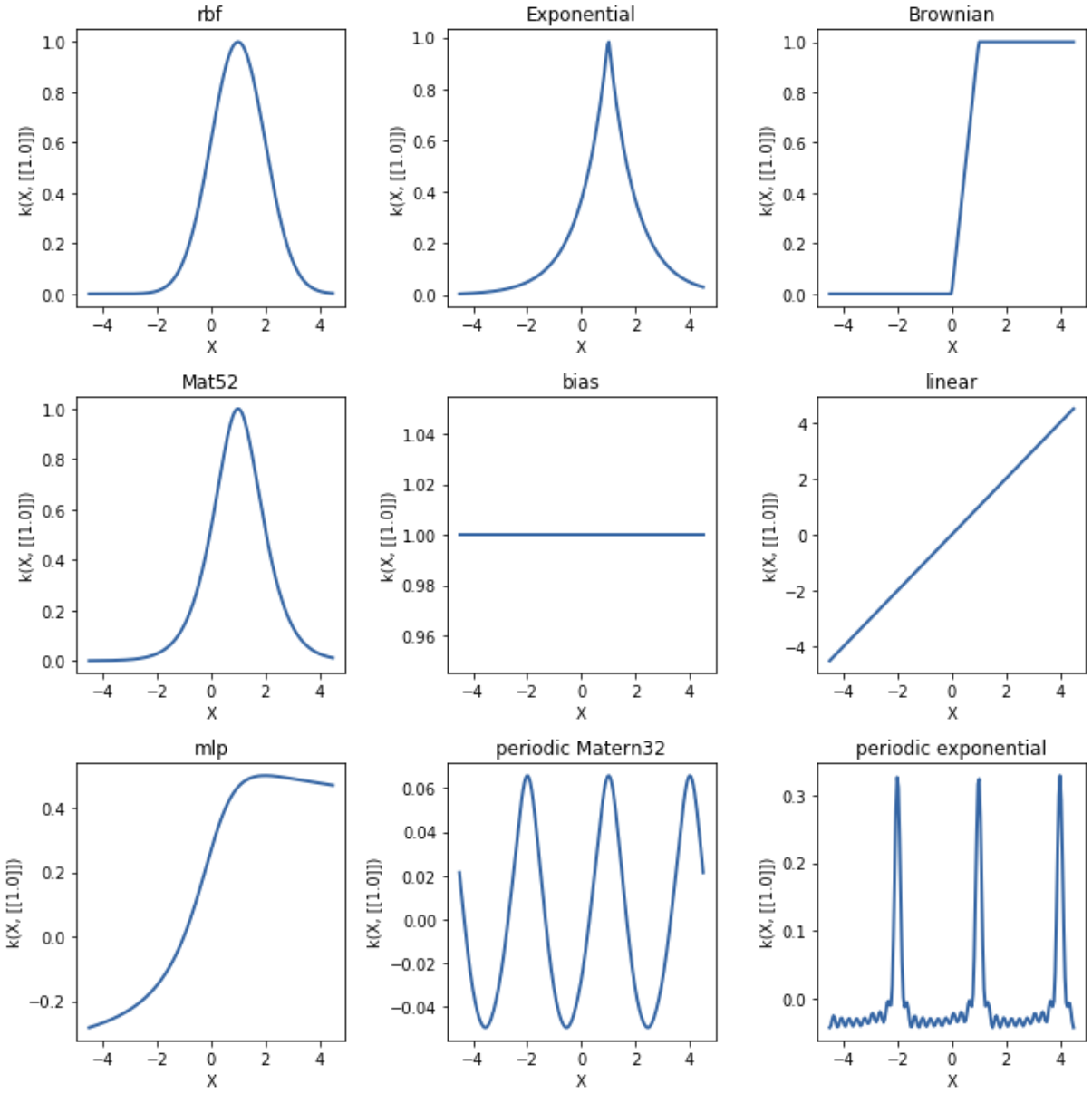


Figura 1. Ejemplos de *kernel*

donde los *kernels* se pueden describir como:

- RBF (radial basis function): Exponencial cuadrática (squared – exponencial (SE))

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \quad (7)$$

- Exponential: *kernel* con función exponencial

$$k(x, x') = \sigma^2 \exp\left(-\frac{x - x'}{2\ell^2}\right) \quad (8)$$

- Brownian: *kernel* browniano

$$k(x, x') = \sigma^2 \min(x, x') \quad (9)$$

- Mater52: *kernel* con función Mater de *Python*

$$k(x, x') = \sigma^2 \left(1 + \sqrt{5}\, r + \frac{5}{3} r^2\right) \exp\left(-\sqrt{5}\, r\right) \; con\; r = \frac{(x - x')^2}{\ell} \quad (10)$$

- Bias: *kernel* sesgado

$$k(x, x') = \sigma^2 \quad (11)$$

- Lienar: *kernel* lineal

$$K(x, x') = \sum_{i=1}^{m} \sigma_i^2 x_i\, x'_i \quad (12)$$

- MLP (Multi layer perceptron): *kernel* generado por la función sen^{-1}

$$K(x, x') = \sigma^2 \sin^{-1}\left(\frac{w x^t x' + b}{\sqrt{w x^t + b + 1}\sqrt{w (x')^t + b + 1}}\right) \quad (13)$$

- Periodic: *kernel* periódico.

$$K(x, x') = \sigma^2 \, exp\left(-\frac{2 \sin^2\left(\frac{\pi(x - x')}{p}\right)}{\ell^2}\right) \quad (14)$$

- *Kernel* combinado: Es el resultado de realizar operaciones entre diferentes *kernels*; por ejemplo, al multiplicar un *kernel* periódico por uno exponencial se obtiene un *kernel* denominado *periodic exponential*.

En algunos de los *kernels* expuestos, existe la presencia del hiperparámetro ℓ. Este hiperparámetro cumple el rol de escalas de longitud característica; es decir, representa el valor hasta el cual el *kernel* se debe mover a lo largo de un eje, en el espacio de entrada para que los valores de la función sean no correlacionados (Rasmussen & Williams, 2006)

Así, si se varía ℓ (la correlación que existe entre las variables), se obtiene el siguiente efecto:

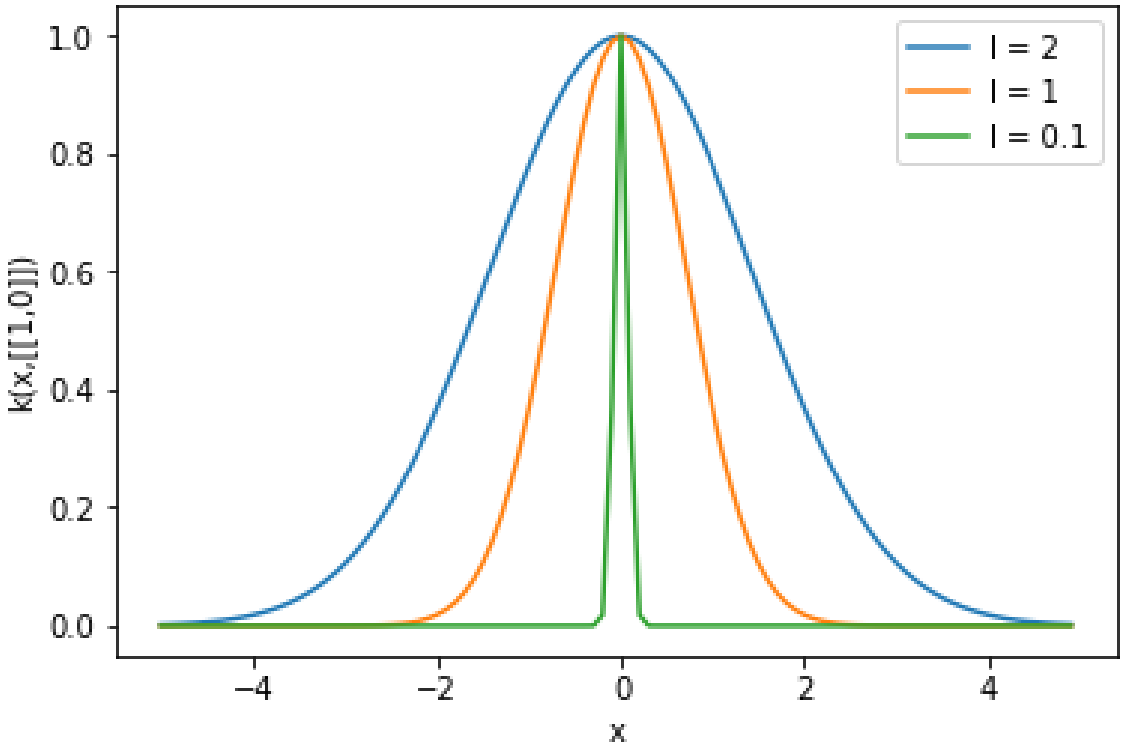


Figura 2. Efecto producido al variar el hiperparámetro ℓ

Esto indica que a menor ℓ, menos puntos correlacionados con la variable representada en el eje x.

Por otro lado, una de las ventajas de utilizar el *kernel* es que se pueden combinar (mediante adición o multiplicación) para generar nuevas estructuras que recojan comportamientos como, por ejemplo la estructura periódica de los datos, el grado del polinomio o la variación creciente, entre otros.

Mediante la multiplicación de dos *kernels* se pueden obtener propiedades de la función que no necesariamente son paramétricas. A continuación, se listan algunos ejemplos:

- Al multiplicar n *kernels* del tipo lineal se puede obtener el grado de un polinomio del mismo orden.

  Luego, si se multiplica n veces el mismo *kernel* lineal, se obtiene:

$$K_n(x,y) = \prod_{j=1}^{n} K_j(x,y) \quad (15)$$

  Por ejemplo, si se toma el producto de dos *kernels* lineales, gráficamente se tiene:

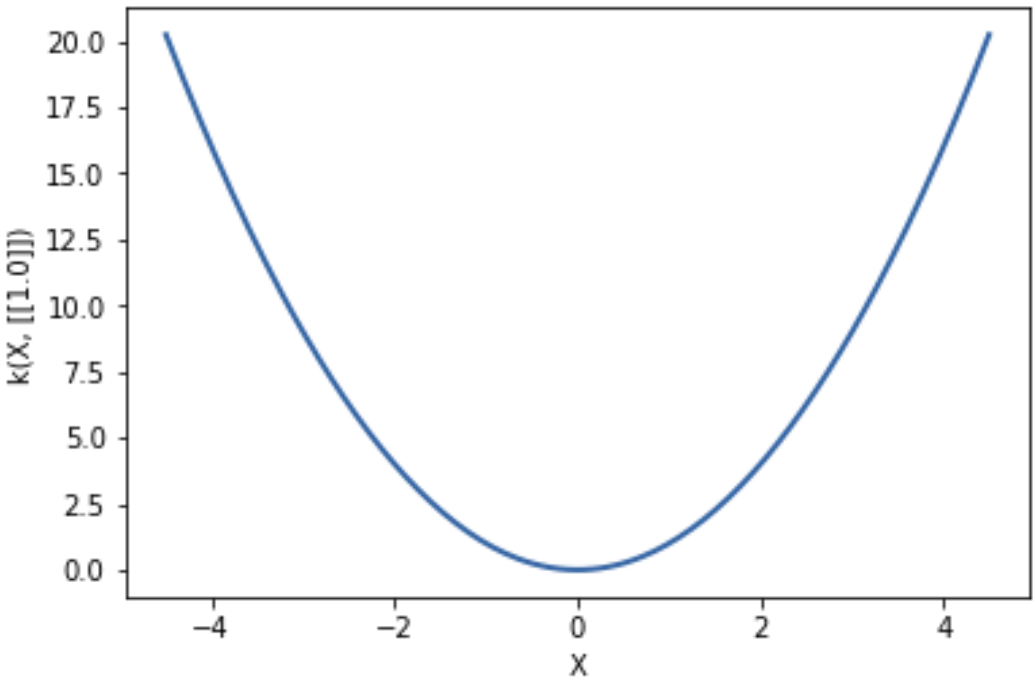


Figura 3. *Kernel* resultante del producto de dos *kernels* lineales

- Se puede obtener la estructura periódica local multiplicando el *kernel* periódico por el *kernel* RBF.

- Para recoger la estructura creciente se puede multiplicar cualquier *kernel* por un *kernel* lineal.

Aprovechando las propiedades anteriormente descritas, se tiene una forma fácil y flexible de construir modelos multidimensionales. En principio, existen *kernels* que permiten representar cualquier función continua (Micchelli, Xu, & Zhang, 2006).

Es decir, se pueden generar modelos a partir de operaciones sobre el *kernel* aplicando álgebra lineal; de esta manera, los GP son cerrados bajo transformaciones lineales (Jidling et al., 2017). Por ejemplo: Sea A un operador lineal y sea $p(f) = \mathcal{GP}(f, \mu, \Sigma)$ un GP, se puede realizar una transformación del tipo $p(Af) = \mathcal{GP}\left(Af, A\mu, A\Sigma\mathrm{A}^{\mathrm{T}}\right)$ de tal forma que el resultado también sea un GP.

En la Figura 4 se presentan tres GP generados a partir de los *kernels* mencionados con anterioridad.

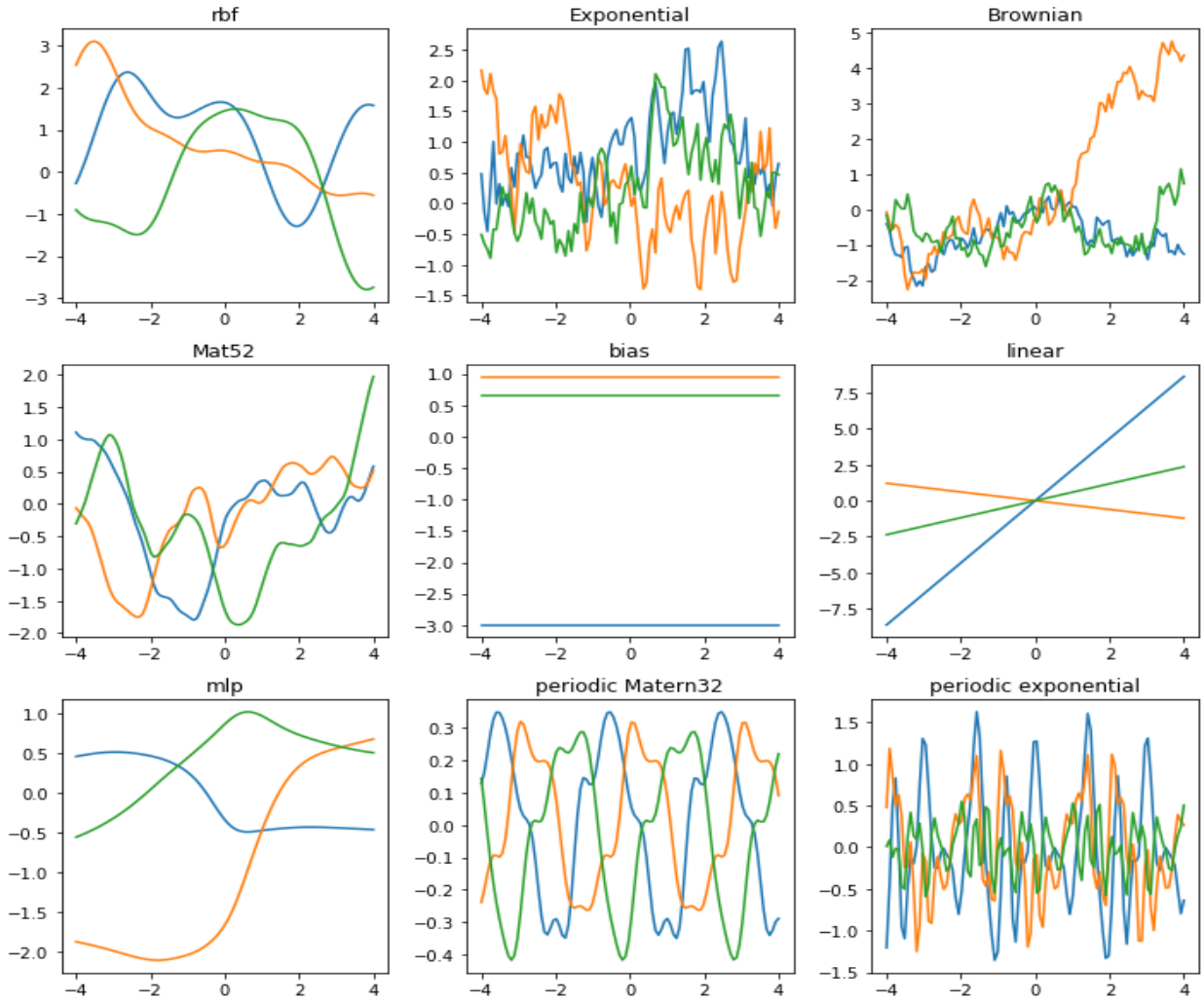


Figura 4. Ejemplo de tres GP para cada *kernel*

Para el caso en que se realizan operaciones con *kernels*, la Figura 5 muestra tres GP generados de forma aleatoria a partir del *kernel* representado en la Figura 3.

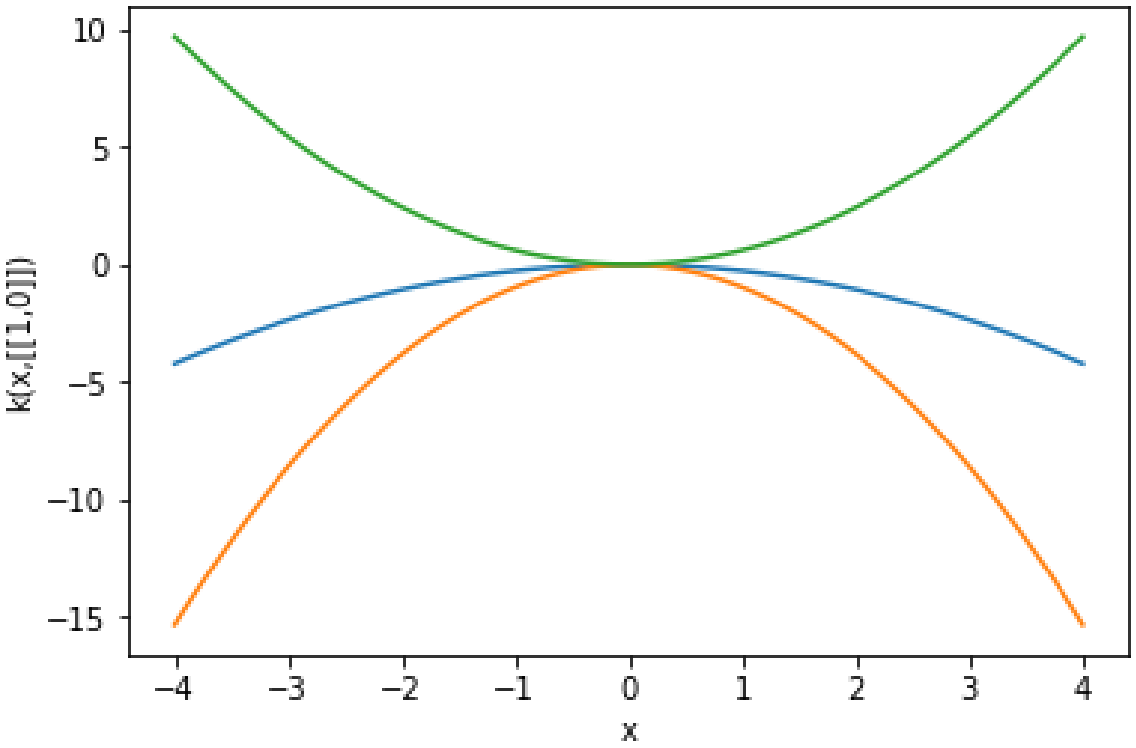


Figura 5. GP aleatorios a partir del producto de dos *kernels* lineales

Al igual que en el caso de trabajar con un *kernel* simple, también se puede realizar una evaluación del efecto que produce variar ℓ cuando se trata de un *kernel* combinado (ver Figura 6).

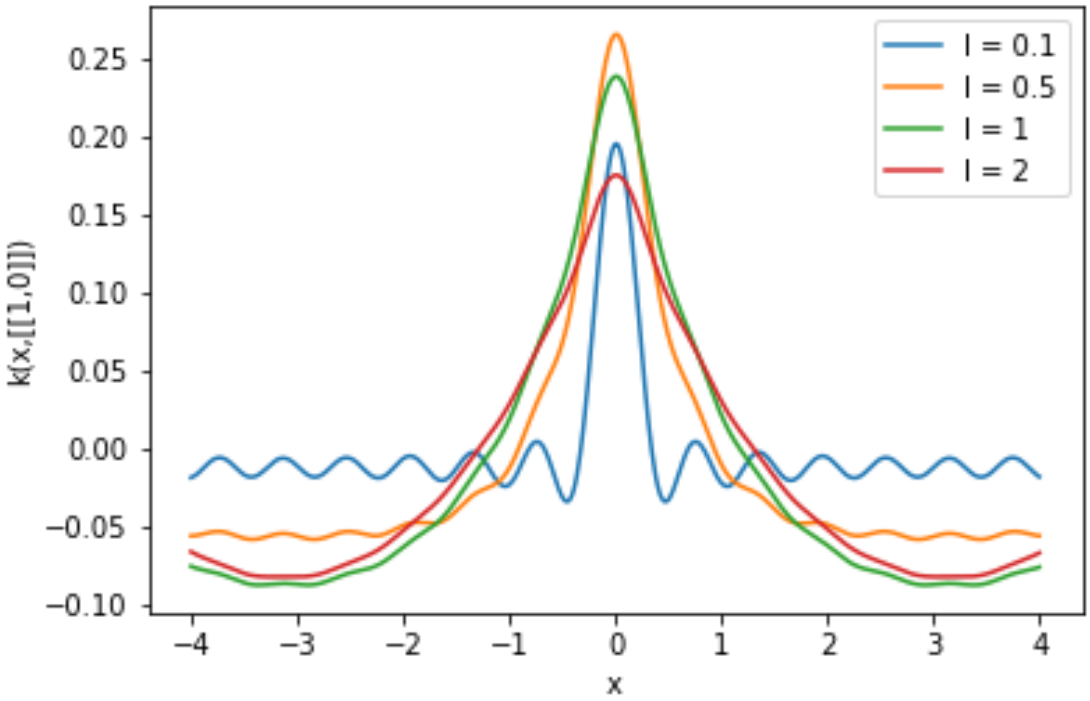


Figura 6. Efecto de variar ℓ en un *kernel* combinado

2.2.3 Modelos con GP

Las propiedades de los GP descritas anteriormente hacen que los investigadores pongan estos una importante atención sobre ellos e incluso hayan demostrado la utilidad de los mismos en una amplia gama de aplicaciones (Wan et al., 2007).

La clave para modelar fenómenos con GP depende de la apropiada elección de las funciones μ y Σ; sin embargo, la construcción de estas funciones para generar un modelo que obedezca a una determinada condición de equilibrio o ley de conservación no es trivial (Jidling et al., 2017). Para resolver este inconveniente se suelen agregar variables ficticias para estudiar los puntos de interés teniendo de esta manera una fácil implementación pero provocando que incremente la dimensión del problema y que las restricciones deseadas no se apliquen a los puntos de interés.

Otra forma de ajustar el modelo con GP al fenómeno estudiado es construir la función de medias y covarianzas; por ejemplo, puede encontrarse una función de covarianzas para modelar campos vectoriales y que recoja las características de curvatura y divergencia nula del proceso (Wahlström et al., 2013).

Por otra parte, este tipo de modelos se pueden extender a modelos multidimensionales multiplicando los *kernels* definidos para cada entrada.

2.3 Campos Vectoriales

Para modelar matemáticamente campos físicos tales como el campo electromagnético y los campos de presión y densidad de fluidos, es útil trabajar con campos vectoriales ya que son funciones que permiten representar una dirección y una magnitud en cada punto del espacio o de una superficie. Un campo vectorial en $\mathbb{R}^n$ es una función $F: D \subset \mathbb{R}^n \to \mathbb{R}^n$ que asigna a cada punto $x = (x_1, \dots, x_n) \in D$ un vector $F(x)$. Así pues, todo campo vectorial en $\mathbb{R}^n$ tiene n componentes $F(x) = \big(F_1(x), \dots, F_n(x)\big)$, cada uno de los cuales es un campo escalar.

Los campos vectoriales tienen una gran aplicación en la representación de fenómenos físicos donde aparecen como cantidades cuantificables; por ejemplo, al estudiar el movimiento de un fluido en una tubería es más práctico señalar la velocidad con la que pasa el fluido en cada punto de la tubería que estudiar cada molécula por separado. Puesto que da una mejor visión del fenómeno estudiado, debido a que el momento lineal es aditivo en la aproximación clásica. Una representación gráfica aproximada de un ejemplo de campo vectorial puede observarse en la Figura 7.

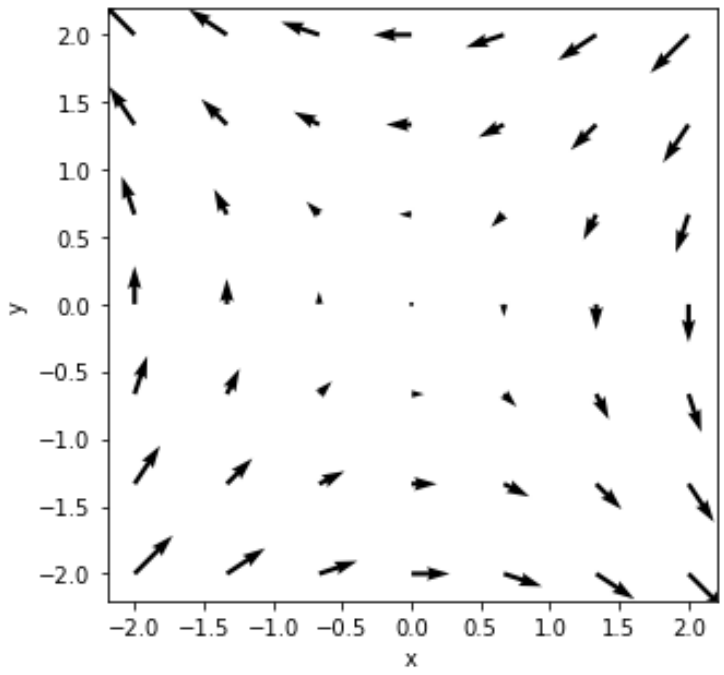


Figura 7. Ejemplo de Campo Vectorial

En física, los campos vectoriales que tienen más aplicación son los *conservativos*. Se dice que un campo vectorial es conservativo si existe una función f tal que $F = \nabla f$. En tal caso, f se llama función potencial de F.

2.3.1 Divergencia y rotacional de un campo vectorial

La divergencia y el rotacional tienen un papel fundamental en la interpretación física de campos como el electromagnetismo y la mecánica de fluidos. Dado un campo vectorial F, se define la *divergencia* de F como la función escalar dada por:

$$div(F) = \nabla \cdot F$$
$$\text{con}, \nabla = (\partial_1, \dots, \partial_n) = \left(\frac{\partial}{\partial x_1}, \dots., \frac{\partial}{\partial x_n}\right) \quad (16)$$

Un campo vectorial se dice *solenoidal* si $div(F) = 0$.

Dado un campo vectorial F, se define el *rotor* (o rotacional) como:

$$rot(F) = \nabla \times F$$
$$\text{con}, \nabla = (\partial_1, \dots, \partial_n) = \left(\frac{\partial}{\partial x_1}, \dots., \frac{\partial}{\partial x_n}\right) \quad (17)$$

Un campo vectorial se dice *irrotacional* si $rot(F) = 0$.

A la divergencia, al rotacional y al laplaciano se les denomina ***operadores diferenciales***; estos operadores tienen gran importancia en distintas disciplinas puesto que al combinar derivadas se pueden construir ecuaciones diferenciales y mediante las que se pueden representar leyes físicas.

Los gases magnetizados (plasmas ideales) pueden ser caracterizados por los campos vectoriales con ciertas invariantes diferenciales, donde las derivadas de las componentes escalares están acopladas por alguna relación. Entonces, dado que las leyes físicas pueden ser escritas como ecuaciones diferenciales, estas podrían incorporarse en las estructuras de covarianza como se determinó para los GP. De este modo, a pesar de su naturaleza aleatoria, todas las simulaciones y la función media de los GP se adhieren estrictamente a estas leyes físicas (Macêdo & Castro, 2008).

2.4 Librerías *Python*

Python es un lenguaje de programación del tipo *opensource* que busca que el usuario trabaje más rápidamente ya que es multiparadigma que soporta orientación a objetos,

programación imperativa e inclusive funcional; e, integre sus sistemas de manera más eficiente dado que es multiplataforma.

El presente trabajo se realiza sobre las bases desarrolladas en las librerías de *Python* denominadas *Numpy* y *GPy*.

A continuación se realiza una pequeña explicación de cada una de las librerías utilizadas.

2.4.1 Librería *Numpy*

La librería *Numpy* aporta un valor agregado para el trabajo de vectores y matrices, y permite tener un directorio de varias funciones matemáticas para operar vectores o matrices. Travis Oliphant creó NumPy en 2005, basándose en lo que había desarrollado en *Numeric.*

2.4.2 Librería *GPy*

GPy es una librería escrita en *Python* para el trabajo con GP desarrollado y mantenido por el grupo de aprendizaje automático de *The University of Sheffield*.

En esta librería se encuentran implementados algunos algoritmos de machine learning basados en GP; actualmente, se encuentra disponible la versión 1.0.7 de la librería *GPy*.

Capítulo 3. Objetivos y metodología de trabajo

El objetivo del presente trabajo fin de máster es desarrollar una herramienta flexible y modular que permita modelar campos escalares y vectoriales con restricciones motivadas por procesos físicos que ocurren en gases magnetizados y turbulentos a una fracción del costo computacional de una simulación CFD.

Para ello, el TFM se ha desarrollado en tres etapas, de acuerdo a los objetivos específicos de dicho trabajo:

1. Programar un *kernel* de tipo: $k(x, y) \coloneqq \phi(x)'\phi(y)$.
2. Implementar un *kernel* para funciones a valores en $\Re^n$.
3. Generar campos vectoriales con rotacional y/o divergencia nulos.

El aporte del presente trabajo es implementar un *kernel* que puede utilizarse para generar campos vectoriales en lenguaje *Python*. Para ello, se ha seguido el siguiente proceso:

1. Programar un *kernel* con divergencia nula.
2. Programar un *kernel* con rotacional nulo.
3. Simular y verificar las salidas para los procesos definidos en los pasos anteriores.
4. Limpiar, comentar y documentar el código.

3.1 Metodología de trabajo

El aporte del presente trabajo es la implementación de un programa que permita estudiar campos vectoriales, en particular, de aquellos con condiciones de divergencia nula y/o rotacional nulo.

Por tanto, la metodología utilizada en dicho trabajo ha sido la que se describe a continuación:

1. **Simular campos vectoriales con coordenadas independientes**. En este punto se describe el desarrollo teórico de un campo vectorial que va del espacio de los reales al plano ($\mathbb{R} \longrightarrow \mathbb{R}^2$). Luego es necesario encontrar la distribución conjunta de dos coordenadas diferentes.

2. **Simular campos vectoriales con coordenadas independientes del plano en el plano.** En este caso, se implementa el código correspondiente para simular campos

vectoriales con coordenadas independientes, pero tomando funciones que van de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$.

3. **Simular campos vectoriales con coordenadas dependientes del plano en el plano.** Se desarrolla de manera teórica y se implementa el código de campos vectoriales cuando existe dependencia lineal de las coordenadas en análisis.

4. **Simular campos vectoriales con divergencia nula.** Se simulan instancias específicas de gases magnetizados incompresibles y campos magnéticos, con divergencia nula.

5. **Simular campos vectoriales con rotacional nulo.** Se simulan instancias de gases magnetizados compresivos, que tienen rotacional nulo.

6. **Resultados.** Los campos escalares y vectoriales que se obtienen como salidas del código son comparados con simulaciones de gases magnetizados y son usados como condiciones iniciales de simulaciones CFD futuras.

Capítulo 4. Desarrollo de la contribución

En las siguientes secciones se desarrollan los pasos citados en la metodología de trabajo:

4.1 Simulación de campos vectoriales con coordenadas independientes

Para comprender mejor la simulación de campos vectoriales, se consideran en primer lugar los campos vectoriales más simples y se van explicando posteriormente los más complejos. Por tanto, lo primero que se describe es cómo se genera un campo vectorial de coordenadas independientes cuyo conjunto de partida es un espacio escalar y cuyo conjunto imagen es el plano.

4.1.1 Simulación de campos vectoriales con coordenadas independientes de $\mathbb{R} \longrightarrow \mathbb{R}^2$

Sea,

$$f: \begin{matrix} \mathbb{R} \longrightarrow & \mathbb{R}^2 \\ t \longmapsto & f(t) = \begin{bmatrix} f_1(t) \\ f_2(t) \end{bmatrix} \end{matrix}, \quad (18)$$

un GP con matriz de covarianzas:

$$cov\left(\begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix}, \begin{bmatrix} f_1(y) \\ f_2(y) \end{bmatrix}\right) = \begin{bmatrix} cov\big(f_1(x), f_1(x)\big) & cov\big(f_1(x), f_2(y)\big) \\ cov\big(f_2(x), f_1(y)\big) & cov\big(f_2(x), f_2(y)\big) \end{bmatrix}, \quad (19)$$

Donde la covarianza de f está definida como: cov: $\mathbb{R}\ x\ \mathbb{R} \longrightarrow \mathbb{R}^{2\ x\ 2}$.

Si se consideran dos coordenadas del campo vectorial: $[f(s), f(t)]$ tal que la distribución conjunta de las coordenadas sigue una distribución Gaussiana y que, la media del proceso es cero, el proceso es definido completamente por la covarianza:

$$K = \begin{bmatrix} cov\big(f(s), f(s)\big) & cov\big(f(s), f(t)\big) \\ cov\big(f(t), f(s)\big) & cov\big(f(t), f(t)\big) \end{bmatrix}, \quad (20)$$

$$\text{con,} \begin{bmatrix} f(s) \\ f(t) \end{bmatrix} = \begin{bmatrix} f_1(s) \\ f_2(s) \\ f_1(t) \\ f_2(t) \end{bmatrix}. \quad (21)$$

Ahora, si se expresa (20) en términos de las coordenadas de f, se tiene:

$$K = \left[\begin{array}{cc|cc} cov(f_1(s), f_1(s)) & cov(f_1(s), f_2(s)) & cov(f_1(s), f_1(t)) & cov(f_1(s), f_2(t)) \\ cov(f_2(s), f_1(s)) & cov(f_2(s), f_2(s)) & cov(f_2(s), f_1(t)) & cov(f_2(s), f_2(t)) \\ \hline cov(f_1(t), f_1(s)) & cov(f_1(t), f_2(s)) & cov(f_1(t), f_1(t)) & cov(f_1(t), f_2(t)) \\ cov(f_2(t), f_1(s)) & cov(f_2(t), f_2(s)) & cov(f_2(t), f_1(t)) & cov(f_2(t), f_2(t)) \end{array}\right], \quad (22)$$

Se puede observar en (22) que existen dos tipos de términos:

- $cov(f_1(s), f_2(s))$ la dependencia entre coordenadas se da en la misma entrada s.
- $cov(f_1(s), f_2(t))$ la dependencia entre coordenadas se da en diferentes entradas s y t.

Dado que, en los supuestos, se considera que las coordenadas son independientes, se puede reorganizar el vector de entrada dado en (21) como:

$$\begin{bmatrix} f(s) \\ f(t) \end{bmatrix} = \begin{bmatrix} f_1(s) \\ f_1(t) \\ f_2(s) \\ f_2(t) \end{bmatrix}, \quad (23)$$

entonces la matriz de covarianzas se convierte en una matriz de bloque – diagonal, expresada de la siguiente manera:

$$K = \left[\begin{array}{cc|cc} cov(f_1(s), f_1(s)) & cov(f_1(s), f_1(t)) & 0 & 0 \\ cov(f_1(t), f_1(s)) & cov(f_1(t), f_1(t)) & 0 & 0 \\ \hline 0 & 0 & cov(f_2(s), f_2(s)) & cov(f_2(s), f_2(t)) \\ 0 & 0 & cov(f_2(t), f_2(s)) & cov(f_2(t), f_2(t)) \end{array}\right], \quad (24)$$

Cada bloque – diagonal de la covarianza modela la estructura de cada coordenada en el tiempo. Los términos que están fuera de la diagonal principal modelan la dependencia que existe entre coordenadas.

En *Python*, se consideran dos *kernels* para la simulación de los parámetros; el primero es un *RBF* y el segundo es un *Matern32*, presentados en la sección 2.2.2. En la Figura 8 se presenta el código para generar una matriz de covarianzas bloque – diagonal.

```
##generación de los kernels

k1 = GPy.kern.RBF(input_dim=1,lengthscale=0.2)
k2 = GPy.kern.Matern32(1, 0.5, 0.2)

##Matrices por cada kernel

C1 = k1.K(t,t)
C2 = k2.K(t,t)

##Matriz resultante

C = np.kron([[1,0],[0,0]], C1) + np.kron([[0,0],[0,1]], C2)
```

Figura 8. Código para generar una matriz de covarianzas bloque – diagonal

La matriz descrita en esta sección, con el código indicado, se puede observar gráficamente en la Figura 9; en dicha figura se representa una matriz de orden $20\,x20$ y las regiones en negro representan la parte nula de la matriz.

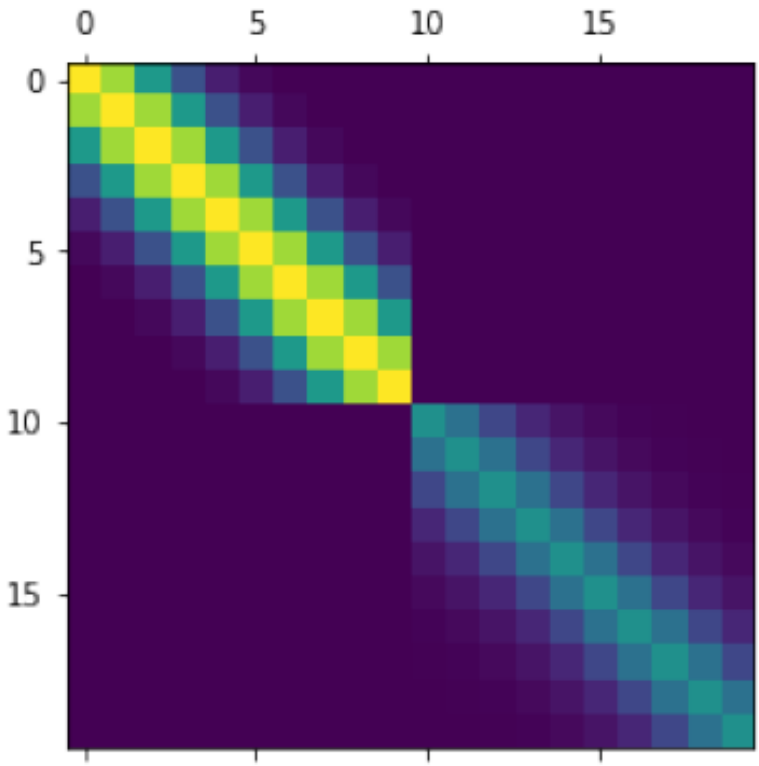


Figura 9. Representación gráfica de una matriz de covarianzas bloque – diagonal

Ahora, también es necesario generar las simulaciones de cada coordenada como un GP para la matriz de covarianzas obtenida; la implementación en *Python* de dichas simulaciones, se muestra en la Figura 10.

```
## semilla de simulación para replica de resultados

np.random.seed(0)

## variable aleatoria Z = (Z_1, Z_2) con distribución normal
## transposición de variable

Z = np.random.multivariate_normal(mu,C,m)
Z = Z.T

## Gráfica de un GP; t → Z_1

plt.figure(1)
for i in range(5):
 plt.plot(t,Z[0:nt,i])
plt.show()

## Gráfica de un GP; t → Z_2

plt.figure(2)
for i in range(5):
 plt.plot(t,Z[nt:nt+nt,i])
plt.show()
```

Figura 10. Código para generar simulaciones con la matriz de covarianzas calculada

Gráficamente, los GP generados por cada uno de los *kernels* ocupados son:

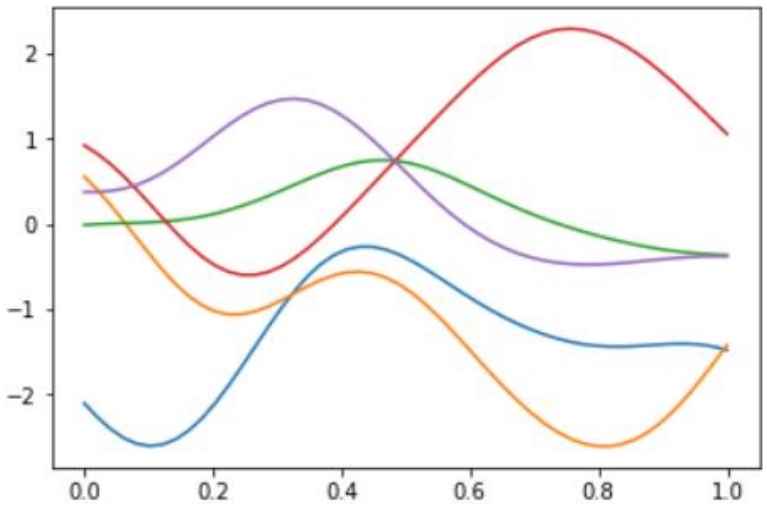


Figura 11. GP con *kernel* RBF

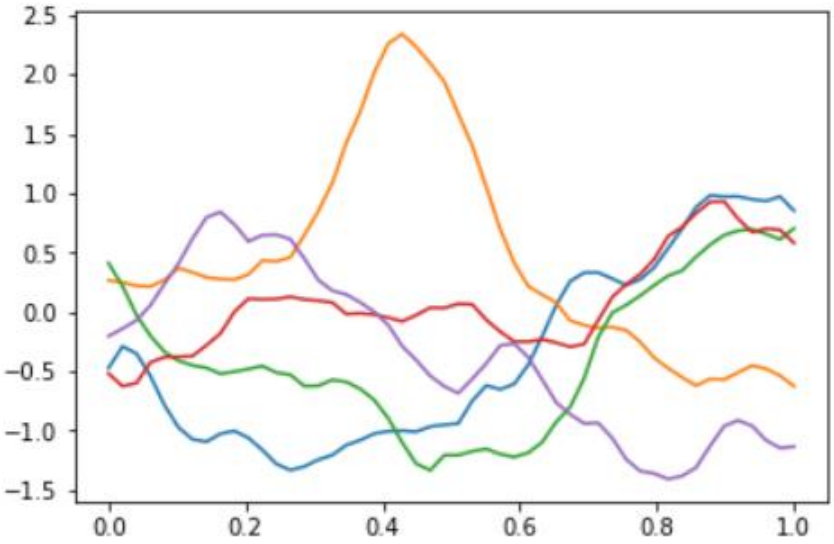


Figura 12. GP con *kernel* Matern32

Una posible aplicación de este tipo de GP es la generación de caminos útiles para aprendizaje de gráficos, textos en general, incluidos los manuscritos, rutas, etc. Dichos caminos útiles podrían generarse, por ejemplo, como se muestra en Figura 13.

```
## Generar GP de t → Z_1 con kernel RBF

a=Z[0:nt,i]

## Generar GP de t → Z_2 con kernel Matern32

b= Z[nt:nt+nt,i]

## generar gráfico
plt.plot(a[0],b[0],'r.')
```

Figura 13. Código para la generación de caminos aleatorios

Además, un ejemplo visual de los caminos aleatorios generados con el código presentado, se muestra en la Figura 14.

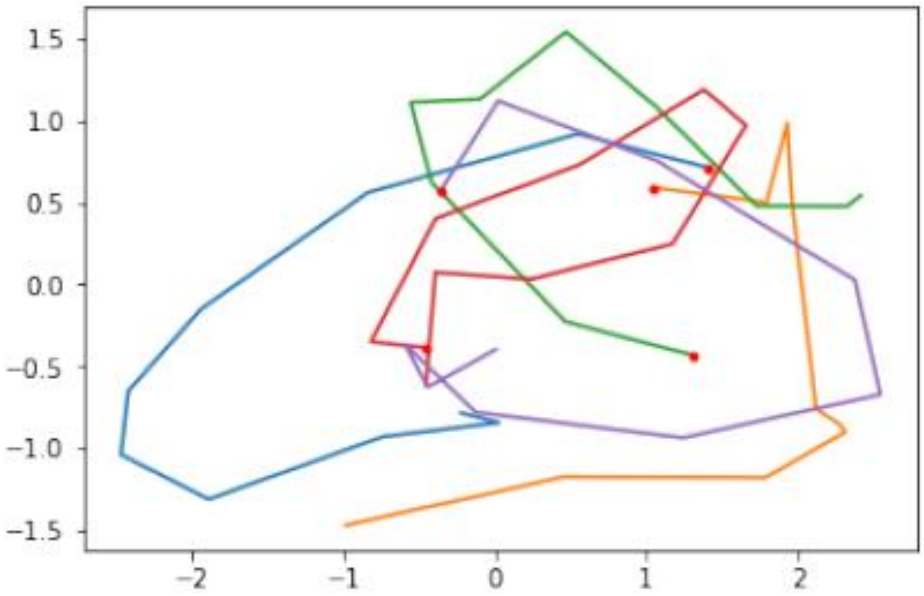


Figura 14. Representación del GP Z_1 vs Z_2

4.1.2 Simulación de campos vectoriales con coordenadas independientes de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

En el caso de la simulación de campos vectoriales con coordenadas independientes que van de un plano en otro plano, el desarrollo teórico es similar al presentado en la sección anterior. A continuación, se muestra la implementación realizada en *Python* y los resultados obtenidos.

Así, es necesario definir la malla en la que se va a trabajar y ordenar las coordenadas una debajo de otra como se describe en el desarrollo teórico; para esto, se utiliza el código que puede apreciarse en la Figura 15.

```
## se define la dimensión del espacio de salida

d = 2

## se determina los puntos en la malla

nxGrid = 8

## se generan los ejes de trabajo

x = np.linspace(0., 1., nxGrid)
y = np.linspace(0., 1., nxGrid)

## generar todos los puntos de la malla 1 a 1

L = []
```

```
for i in range(nxGrid):
 for j in range(nxGrid):
 L.append([x[i], y[j]])

## malla generada en forma de vector

X = np.reshape(L, (nxGrid*nxGrid, d))
```

Figura 15. Generación de la malla de trabajo

Ahora, asumiendo que el comportamiento de las coordenadas es similar, por ser estas independientes, se toma un *kernel* y se procede a simular y a graficar la matriz de covarianzas obtenida (ver Figura 16 y Figura 17):

```
##generar un kernel de dimensión 2 y con el parámetro ℓ deseado

k = GPy.kern.RBF(input_dim=2, lengthscale=0.2)

## se genera la matriz de covarianzas

S = k.K(X, X)

## productor de Kroneker para generar un GP de variables independientes
Sigma = np.kron([[1,0.],[0.,1]], S)
```

Figura 16. Código para generar una matriz de covarianzas bloque – diagonal de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

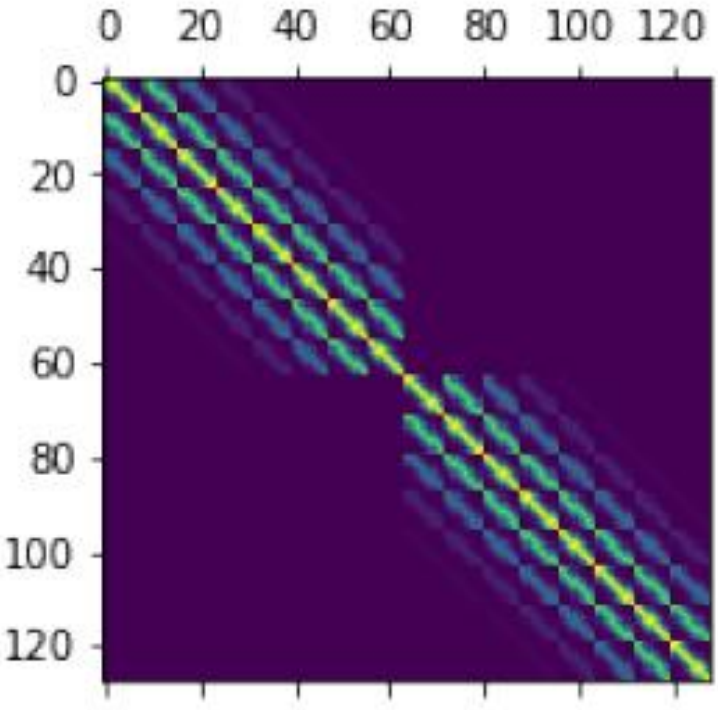


Figura 17. Representación gráfica de una matriz de covarianzas bloque – diagonal de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

Cabe mencionar que en este tipo de matrices la diagonal inversa es nula; es por esto que se puede ver una sombra en la Figura 17 que va en esa dirección.

Adicionalmente, se realiza la simulación de dos campos vectoriales aleatorios con coordenadas independientes para poder tener una idea visual de por qué son útiles para modelar, por ejemplo, los campos magnetizados. El código para realizar la simulación de dos campos vectoriales aleatorios puede observarse en la Figura 18 y en la Figura 19 se presentan dichos campos.

```
##definir la semilla

np.random.seed(0)

##generar GP con coordenadas independientes en el plano

mu = np.zeros(d*nxGrid*nxGrid)
Z = np.random.multivariate_normal(mu, Sigma, 2)

##Separar los GP en sus componentes

dx0 = Z[0][0:nxGrid*nxGrid]
dy0 = Z[0][nxGrid*nxGrid:d*nxGrid*nxGrid]

dx1 = Z[1][0:nxGrid*nxGrid]
dy1 = Z[1][nxGrid*nxGrid:d*nxGrid*nxGrid]

##generar malla de gráfico en el plano

xx, yy = np.meshgrid(x, x)

##grafico del campo vectorial

ax0.quiver(xx, yy, dx0, dy0)
ax1.quiver(xx, yy, dx1, dy1)
```

Figura 18. Código para generar dos campos vectoriales aleatorios con coordenadas independientes de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

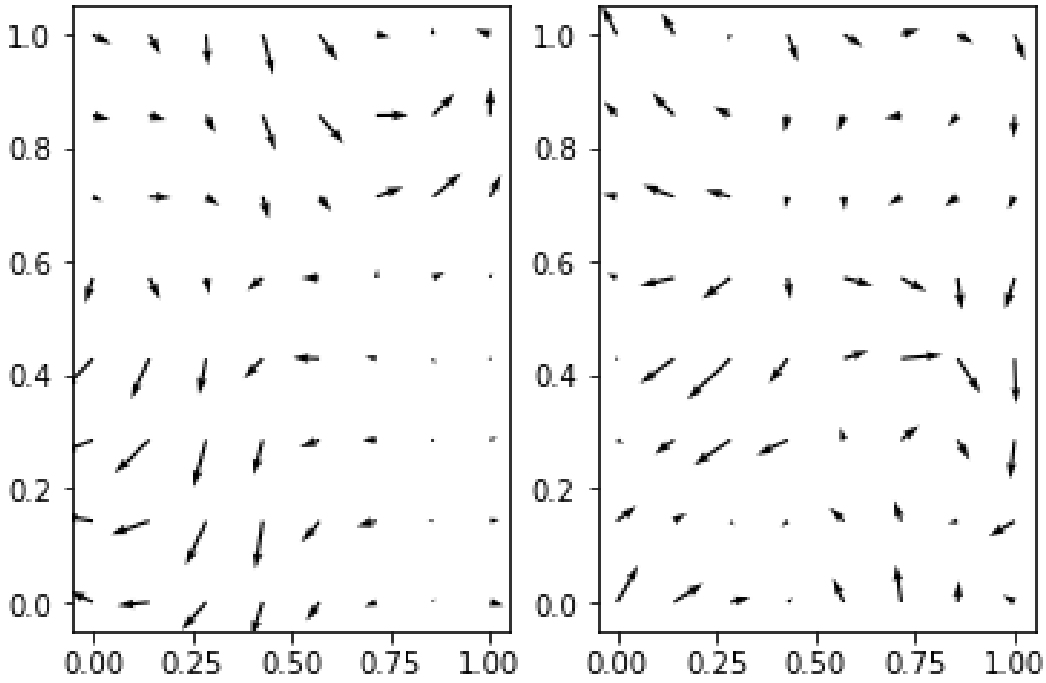


Figura 19. Representación dos campos vectoriales aleatorios con coordenadas independientes de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

4.2 Simulación de campos vectoriales con coordenadas dependientes de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

Al igual que en el caso de las coordenadas independientes, se va a realizar el desarrollo teórico de la aplicación para luego exponer la implementación en *Python*.

Entonces, sea

$$f: \begin{matrix} \mathbb{R}^2 \longrightarrow & \mathbb{R}^2 \\ x \longmapsto & f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} \end{matrix}, \tag{25}$$

un GP, donde $f_1(x) = a_1 u(x)$, $f_2(x) = a_2 u(x)$ y $u(x) \sim GP\big(0, k(x,x)\big)$ y con matriz de covarianzas:

$$K = \begin{bmatrix} a_1^2 cov\big(u(x),u(x)\big) & a_1 a_2 cov\big(u(x),u(x)\big) \\ a_2 a_1 cov\big(u(x),u(x)\big) & a_2^2 cov\big(u(x),u(x)\big) \end{bmatrix} = \mathrm{k(x,x)} \left(\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} [a_1 a_2] \right) = k(x,x) a a' \tag{26}$$

donde la covarianza de f está definida como: cov: $\mathbb{R}\ x\ \mathbb{R} \longrightarrow \mathbb{R}^{2\ x\ 2}$.

Ahora, si se consideran n puntos $(x_1, \dots, x_n)$, la matriz de covarianzas es:

$$\begin{bmatrix} k(x_1,x_1)aa' & \dots & k(x_1,x_n)aa' \\ \vdots & \ddots & \vdots \\ k(x_n,x_1)aa' & \dots & k(x_n,x_n)aa' \end{bmatrix} =: \begin{bmatrix} k(x_1,x_1) & \dots & k(x_1,x_n) \\ \vdots & \ddots & \vdots \\ k(x_n,x_1) & \dots & k(x_n,x_n) \end{bmatrix} \otimes aa' \qquad (27)$$

A continuación, se realiza la implementación en *Python*. Para esto es necesario definir la malla en la que se va a trabajar y ordenar las coordenadas una debajo de otra como se describe en el desarrollo teórico; para esto, se utiliza el mismo código que el presentado en la Figura 15.

Ahora, asumiendo que el comportamiento de las coordenadas está dado por los coeficientes $a1$ y $a2$ (por ser dependientes), se toma un *kernel* y se procede a simular y graficar la matriz de covarianzas obtenida (ver Figura 20 y Figura 21):

```
##generar kernel de dimensión 2 y ℓ deseado

k = GPy.kern.RBF(input_dim=2, lengthscale=0.2)
Sigma = k.K(X, X)

##definir las components del vector a

a1 = 2
a2 = 1
a = np.array([[a1], [a2]])

##generar la matriz de covarianzas asociadas

K = np.kron(Sigma, a.dot(a.T))
```

Figura 20. Código para generar una matriz de covarianzas bloque – diagonal de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

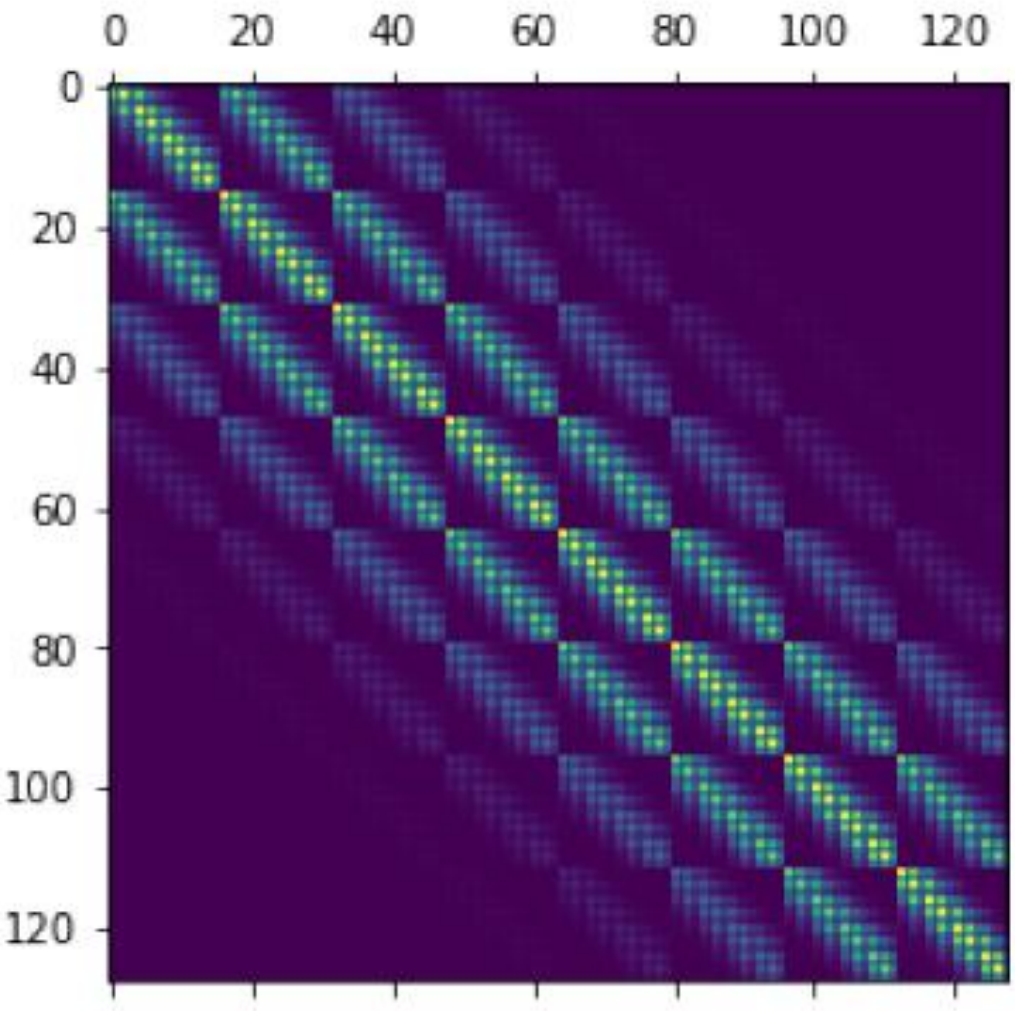


Figura 21. Representación gráfica de una matriz de covarianzas bloque – diagonal de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

Adicionalmente, se realiza la simulación de dos campos vectoriales aleatorios con coordenadas dependientes para poder tener una idea visual de por qué son útiles para modelar, por ejemplo, flujos turbulentos. El código para esta simulación consta en la Figura 22 y gráficamente, se presenta en la Figura 23:

```
Np.random.seed(0)

mu = np.zeros(d*nxGrid*nxGrid)
Z = np.random.multivariate_normal(mu, K, 2)
Z.shape

dx0 = Z[0][0:nxGrid*nxGrid]
dy0 = Z[0][nxGrid*nxGrid:d*nxGrid*nxGrid]

dx1 = Z[1][0:nxGrid*nxGrid]
```

```
dy1 = Z[1][nxGrid*nxGrid:d*nxGrid*nxGrid]

xx, yy = np.meshgrid(x, x)


fig, (ax0, ax1) = plt.subplots(1, 2)

ax0.quiver(xx, yy, dx0, dy0)
ax1.quiver(xx, yy, dx1, dy1)
```

Figura 22. Código para generar dos campos vectoriales aleatorios con coordenadas dependientes de $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$

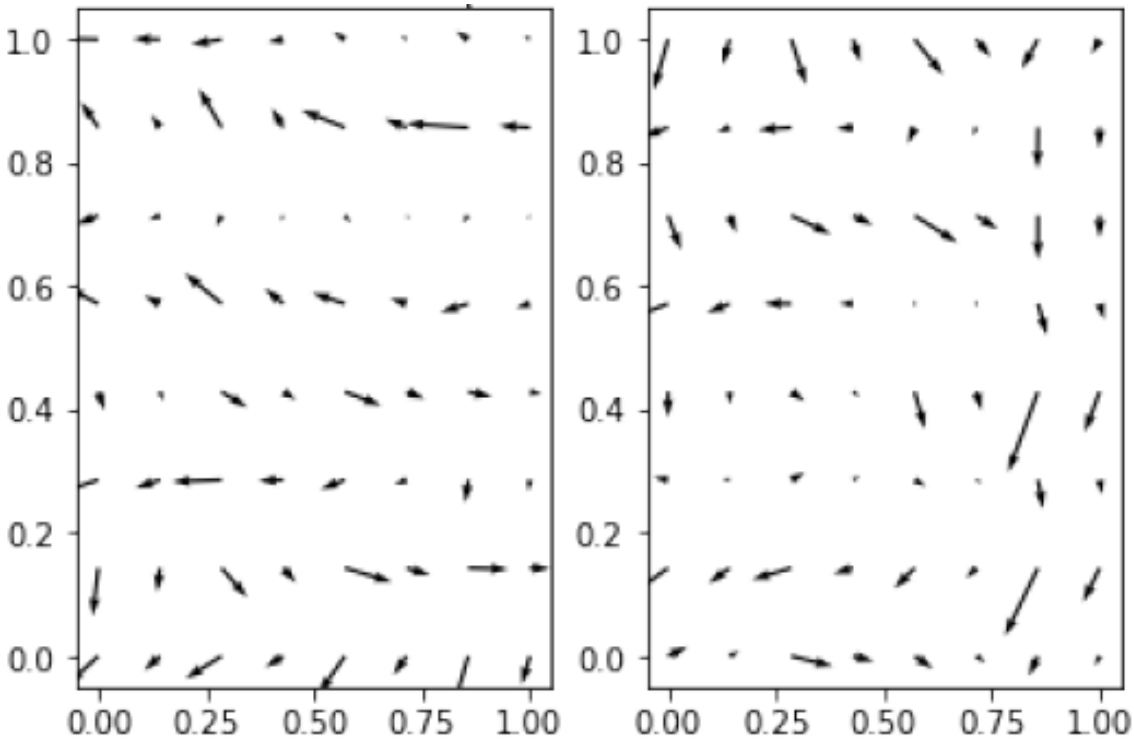


Figura 23. Representación dos campos vectoriales aleatorios con coordenadas dependientes de $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$

4.3 Simulación de campos vectoriales con divergencia nula

Wahlström (2013) presenta el *kernel* para simular campos vectoriales con divergencia nula, la función descrita puede ser obtenida al aplicar el concepto de divergencia al *kernel* escalar RBF, obteniendo de esta manera una función de covarianzas que asegura que las funciones obtenidas de un GP con este *kernel* tengan divergencia nula. Es decir:

$$k_{div}(x,y) = \begin{bmatrix} \partial_{x_2 y_2} & -\partial_{x_2 y_1} \\ -\partial_{x_1 y_2} & \partial_{x_1 y_1} \end{bmatrix} k(x,y), x = [x_1, x_2] \text{ e } y = [y_1, y_2] \in \mathbb{R}^2 \quad (28)$$

Ahora, si se toma el *kernel* exponencial al cuadrado, se la siguiente manera:

$$\sigma_f^2 \, e^{-\frac{1}{2\ell^2}||x-y||^2} \quad (29)$$

entonces,

$$\partial_{y_1} = \frac{s^2}{\ell^2}(x_1 - y_1) \, e^{-\frac{1}{2\ell^2}||x-y||^2} \quad (30)$$

$$\partial_{y_2} = \frac{s^2}{\ell^2}(x_2 - y_2) \, e^{-\frac{1}{2\ell^2}||x-y||^2} \quad (31)$$

$$\partial_{x_2 y_2} = s^2 \, e^{-\frac{1}{2\ell^2}||x-y||^2} \left(\frac{1}{\ell^2} - \frac{1}{\ell^4}(x_2 - y_2)^2 \right) \quad (32)$$

$$\partial_{x_1 y_1} = s^2 \, e^{-\frac{1}{2\ell^2}||x-y||^2} \left(\frac{1}{\ell^2} - \frac{1}{\ell^4}(x_1 - y_1)^2 \right) \quad (33)$$

$$-\partial_{x_2 y_1} = s^2 \, e^{-\frac{1}{2\ell^2}||x-y||^2} \left(\frac{1}{\ell^4}(x_1 - y_1)(x_2 - y_2) \right) \quad (34)$$

$$-\partial_{x_1 y_2} = s^2 \, e^{-\frac{1}{2\ell^2}||x-y||^2} \left(\frac{1}{\ell^4}(x_2 - y_2)(x_1 - y_1) \right) \quad (35)$$

Luego, tomando (32), (33), (34) y (35), se forma la matriz:

$$k_{div}(x,y) = \left(\begin{bmatrix} \frac{1}{\ell^2} - \frac{1}{\ell^4}(x_2 - y_2)^2 & \frac{1}{\ell^4}(x_1 - y_1)(x_2 - y_2) \\ \frac{1}{\ell^4}(x_1 - y_1)(x_2 - y_2) & \frac{1}{\ell^2} - \frac{1}{\ell^4}(x_1 - y_1)^2 \end{bmatrix} s^2 \, e^{-\frac{1}{2\ell^2}||x-y||^2} \right) \quad (36)$$

Wahlström (2013), convierte la matriz anterior, en la siguiente forma funcional:

$$K_{div}(x,x') = \sigma_f^2 e^{-\frac{\|x-x'\|^2}{2\ell^2}} \left(\left(\frac{x-x'}{\ell}\right)\left(\frac{x-x'}{\ell}\right)^T + \left(n_y - 1 - \frac{\|x-x'\|^2}{\ell^2}\right) I_{n_y} \right) \quad (37)$$

Donde n_y es la dimensión de la salida y ℓ y σ_f son los llamados hiperparámetros que pueden ser distintos para cada dimensión.

Ahora, se muestra la implementación del *kernel* en *Python*, se toma (37) como referencia:

```
def divFreeKern(x, y, l):
'''
input:
- x e y dos vectores fila en R^n.
output:
- k(x, y) div free kernel matriz R^nxn.
'''
# definir los hiperparámetros ℓ y sigma (alpha en este código)

alpha=1
l=l

##operaciones previas
n = len(x)
d = x - y
normd = np.sum(d**2)

##evaluar primer término del kernel final

T1 = (alpha**2)*np.e**(-normd/(2*(l**2)))

##evaluar primer sumando del Segundo término del kernel final

In = np.eye(n, n)
v = np.reshape(d, (n, 1))
vt = np.transpose(v)
ddT = v.dot(vt)

## evaluar Segundo sumando del Segundo término del kernel final

nod = np.repeat(1, n)
nod[1:n] = -1
NOD = toeplitz(nod)

## calcular el Segundo término del kernel final
```

```
T2 = (1/l**2)*ddT+(n-1-normd/(l**2))*(In)

##obtener matriz final

return(T1*T2)
```

Figura 24. Generación *kernel* con divergencia nula

Para verificar el funcionamiento del programa de manera adecuada, primero es necesario definir la malla en la que se va a trabajar y ordenar las coordenadas una debajo de otra como se muestra en la Figura 15. Ahora, sobre esta malla se aplica el *kernel* desarrollado para obtener la matriz de covarianzas (ver Figura 25 y Figura 26):

```
l=0.3
K=divFreeKern(X[:,0],X[:,1],l)
divFreeK=K
```

Figura 25. Código para generar una matriz con divergencia nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$

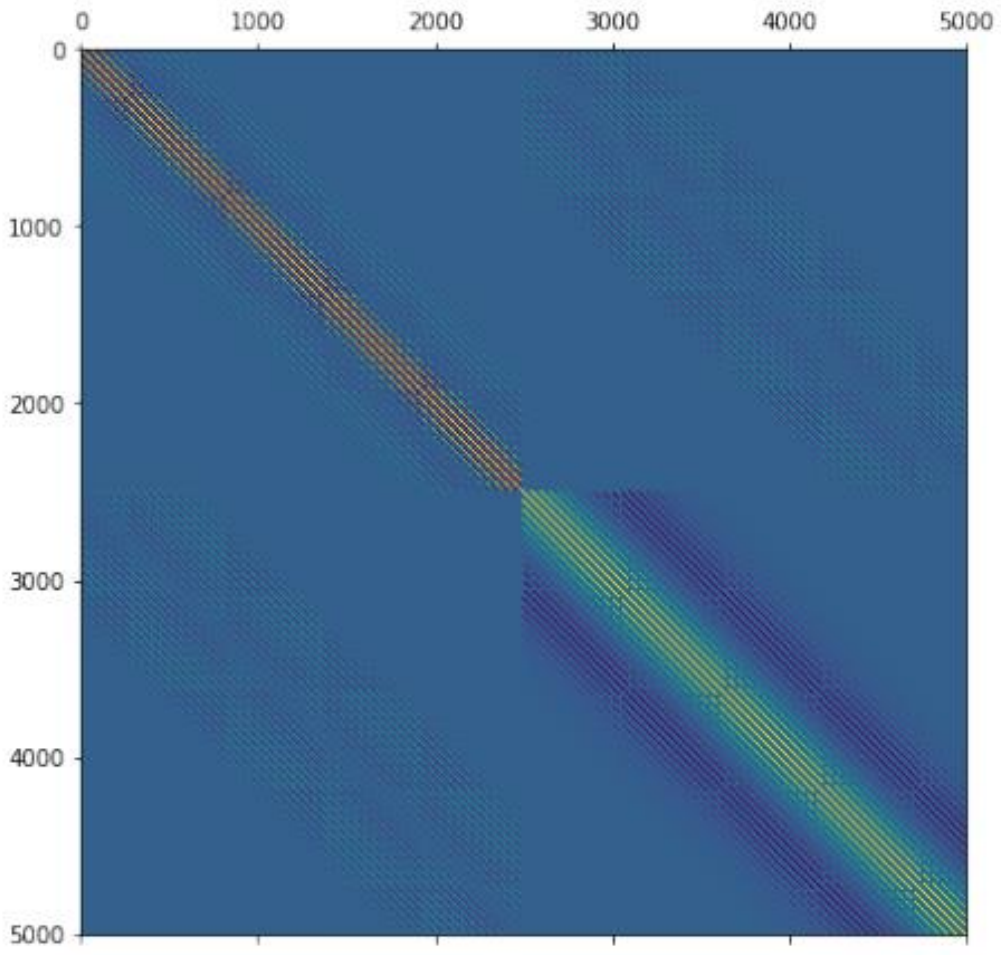


Figura 26. Representación gráfica del *kernel* con divergencia nula $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$, con $l = 0{,}3$

Una de las validaciones a realizar con esta simulación es determinar que la matriz es definida positiva; para esto, se muestran los valores propios de la matriz para verificar que todos son positivos (ver Figura 27):

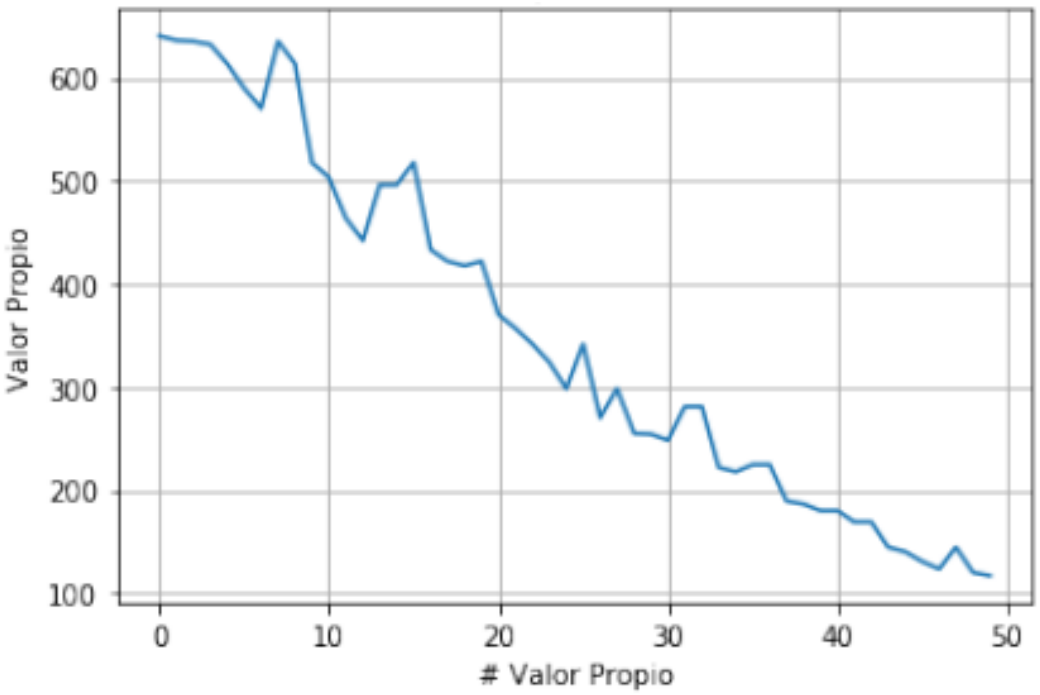


Figura 27. Representación gráfica de los primeros 50 valores propios del *kernel* con divergencia nula

Ahora, ya que el *kernel* generado cumple con las características necesarias de divergencia nula, se realiza la simulación de campos vectoriales aleatorios con coordenadas dependientes. El código para esta simulación consta en la Figura 28 y en la Figura 29 puede observarse dicha simulación.

```
## generar la semilla para replicar los datos

np.random.seed(0)


##generar parámetros para la generación del kernel correspondiente


mu = np.zeros(d*nxGrid)
Z1 = np.random.multivariate_normal(mu, divFreeK, 4)
```

```
##generar las componentes del campo vectorial

dx0 = Z1[0][0:nxGrid*nxGrid]
dy0 = Z1[0][ nxGrid*nxGrid:d*nxGrid*nxGrid]
```

Figura 28. Código para generar campos vectoriales aleatorios con divergencia nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$

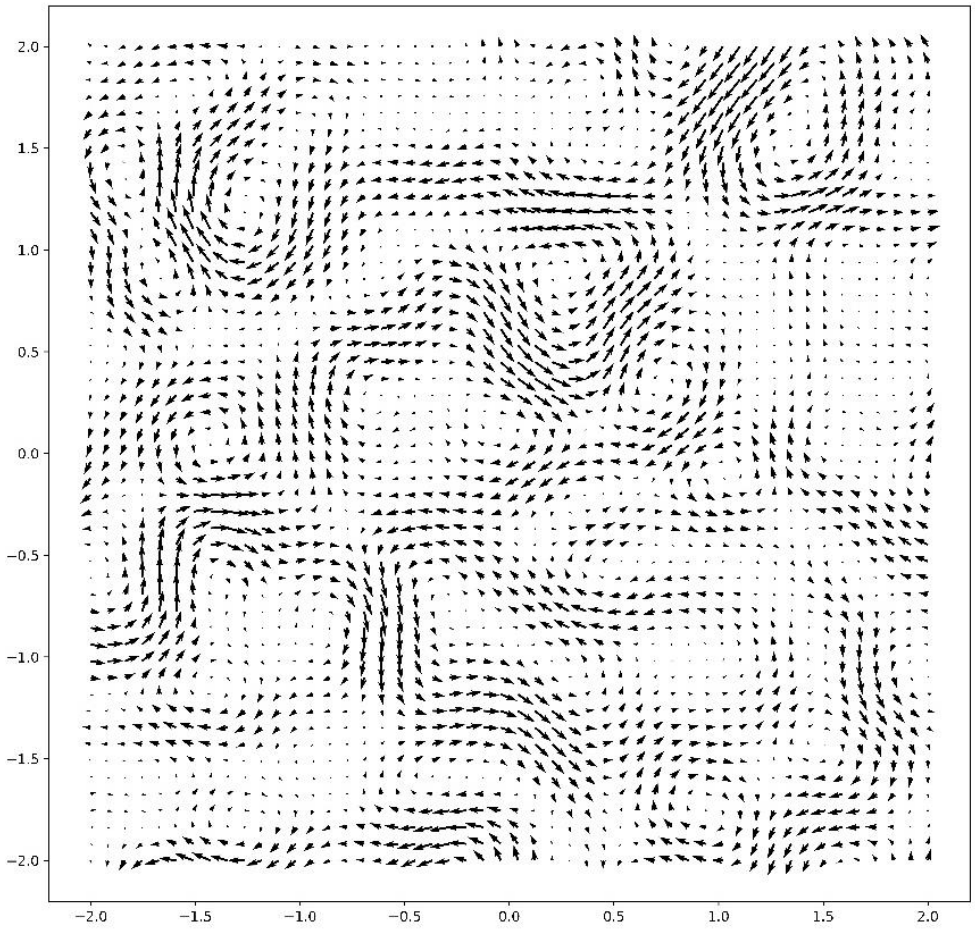


Figura 29. Representación de campos vectoriales aleatorios con divergencia nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$, con $l = 0{,}3$

Cabe recalcar que los campos vectoriales con divergencia nula, físicamente pueden representar campos de velocidad en fluidos incompresibles o campos magnéticos (por su naturaleza) solenoidal (Wahlström et al., 2013).

La característica principal de un campo con divergencia nula (cero) es que carece de monopolos o puntos de acumulación dentro del gráfico (que actúan como fuentes o sifones). Estos campos se observan, por ejemplo, en agua moviéndose en un canal, aire a velocidades mucho más bajas que la del sonido, en campos magnéticos ya que estos son siempre dipolares; etc. (Wahlström et al., 2013).

Una forma de comprobar que un campo vectorial generado posee las características de divergencia nula es aplicar diferencias finitas de segundo orden sobre dicho campo para obtener una aproximación de la divergencia del campo. De la mencionada operación, se obtiene la Figura 30, en la que se puede observar que el campo simulado posee una divergencia que está entre los valores de $-0{,}5352$ y $0{,}4762$ (cercanos a 0). Para la salida gráfica se ha utilizado el programa VisIT, que es una herramienta de visualización *opensource*.

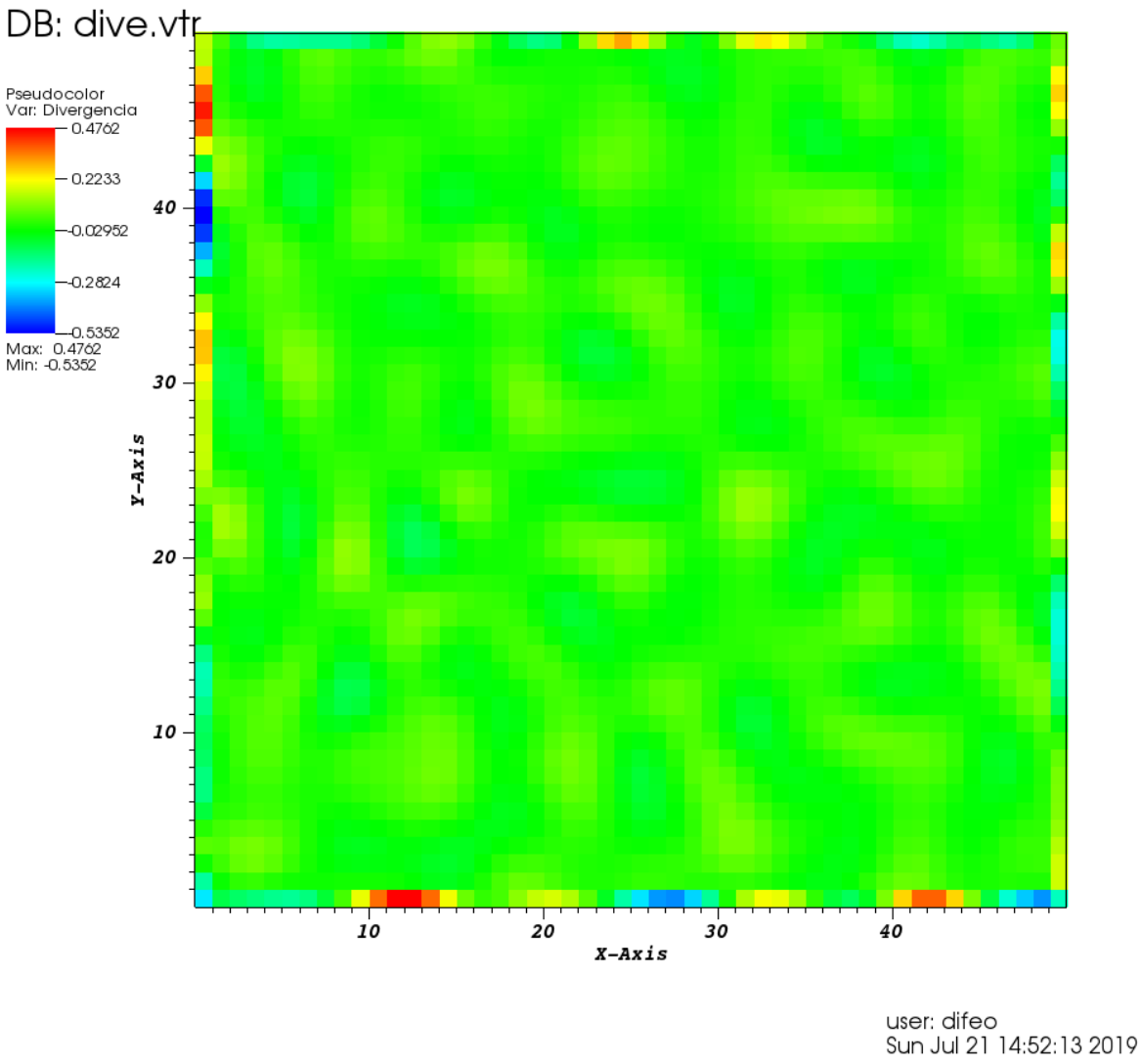


Figura 30. Representación de campos vectoriales aleatorios con divergencia nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$, con $l = 0{,}3$

Por otro lado, este campo vectorial garantiza que la divergencia de este sea 0; sin embargo, no restringe de ninguna manera el rotacional dentro de la simulación.

En la Figura 31, se muestra la representación del rotacional por diferencias finitas del campo vectorial generado. Se puede observar que el rotacional está entre los valores de -7.288 y 7.069; es decir, no se encuentra restringida la propiedad rotacional del campo.

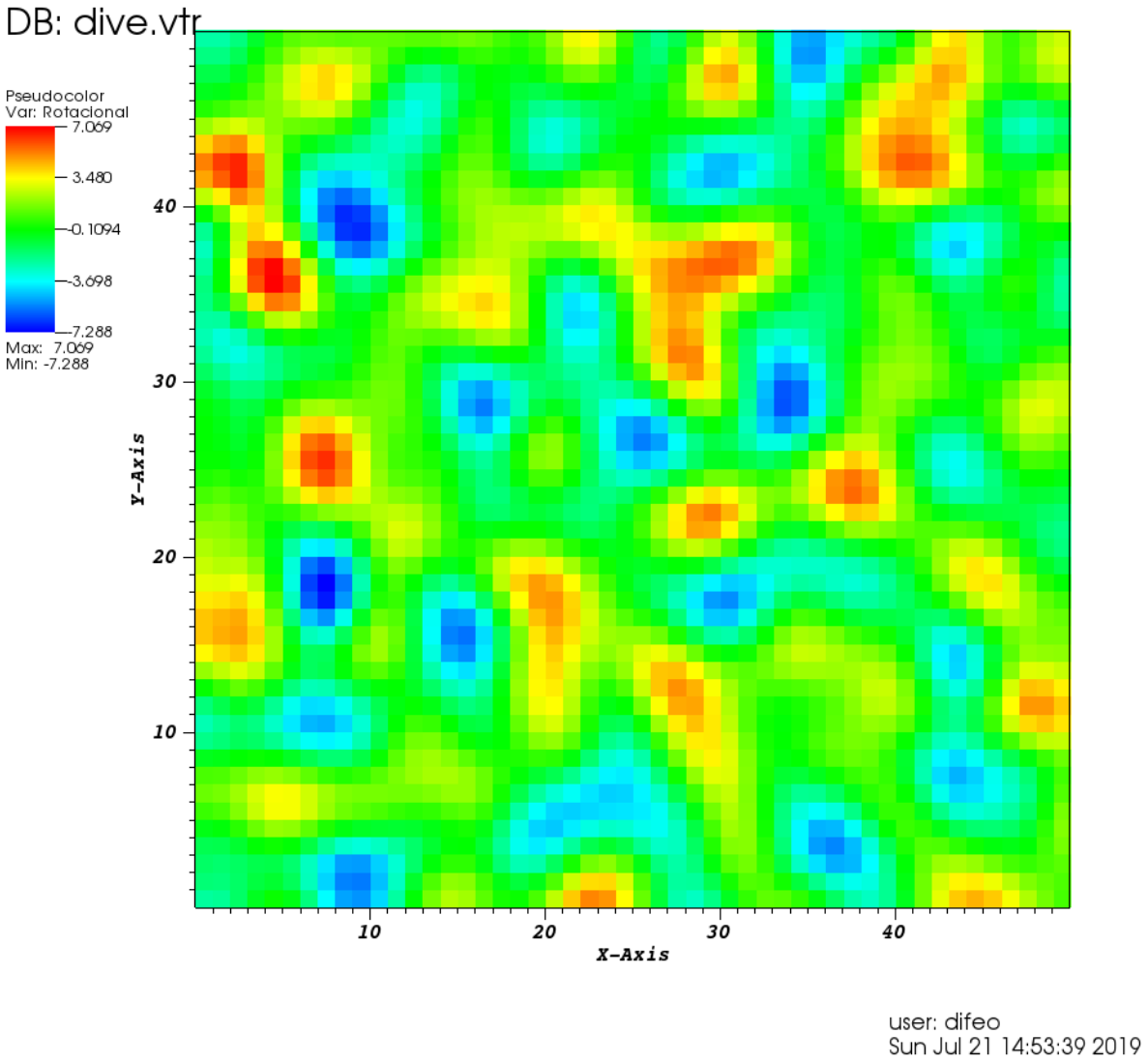


Figura 31. Representación del rotacional de campos vectoriales aleatorios con divergencia nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$, con $l = 0{,}3$

4.4 Simulación de campos vectoriales con rotacional nulo

Wahlström (2013), presenta el *kernel* para simular campos vectoriales con rotacional nulo, la función descrita puede ser obtenida al aplicar el concepto de rotacional al *kernel* escalar RBF, obteniendo de esta manera una función de covarianzas que asegura que las funciones obtenidas de un GP con este *kernel* tengan rotacional nulo:

$$K_{curl}(x,y) = \begin{bmatrix} \partial_{x_1 y_1} & \partial_{x_1 y_2} & \partial_{x_1 y_3} \\ \partial_{x_2 y_1} & \partial_{x_2 y_2} & \partial_{x_2 y_3} \\ \partial_{x_3 y_1} & \partial_{x_3 y_2} & \partial_{x_3 y_3} \end{bmatrix}, x = [x_1, x_2, x_3], y = [y_1, y_2, y_3] \in \mathbb{R}^3 \quad (38)$$

Al igual que en el caso de la divergencia nula, se considera el *kernel* exponencial:

$$\sigma_f^2 e^{-\frac{\|x-y\|^2}{2\ell^2}} \quad (39)$$

Y aplicando el cálculo similar al realizado para la divergencia, se obtiene la siguiente generalización:

$$K_{curl}(x,x') = \frac{\sigma_f^2}{\ell^2} e^{-\frac{\|x-x'\|^2}{2\ell^2}} \left(I_n - \left(\frac{x-x'}{\ell}\right)\left(\frac{x-x'}{\ell}\right)^T \right), x \in \mathbb{R}^n \quad (40)$$

Donde n es la dimensión de la salida, ℓ y σ_f son los llamados hiperparámetros que pueden ser distintos para cada dimensión.

Aunque se puede generar un código similar al utilizado en el caso de la divergencia nula, en el presente trabajo se ha desarrollado un código alternativo que produce buenos resultados (ver Figura 32):

```
def curlFreeKern(x, y, l):
'''
input:
- x e y dos vectores fila en R^n.
output:
- k(x, y) curl free kernel matriz R^nxn.
'''
## definir hiperparámetros y dimensión del espacio de salida

alpha=1
l=l
n = len(x)

##ordenar los vectores uno debajo del otro

xrf=np.matlib.repmat(x,n,1)
xrc=xrf.T
dxx1=xrf-xrc
dxx=dxx1**2
```

```
yrf=np.matlib.repmat(y,n,1)
yrc=yrf.T
dyy1=yrf-yrc
dyy=dyy1**2

z=np.zeros(n)
zrf=np.matlib.repmat(z,n,1)
zrc=zrf.T
dzz1=zrf-zrc
dzz=dzz1**2

dxy=dxx1*dyy1
dxz=dxx1*dzz1
dyz=dyy1*dzz1

##calcular el primer térmmino del kernel final

t1=(alpha**2)*np.e**(-1/2*(dxx+dyy+dzz)/l**2)
t1=np.kron([[1,1,1],[1,1,1],[1,1,1]],t1)

##calcular los términos internos del kernel final

t11=np.kron([[1,0,0],[0,0,0],[0,0,0]], 1/l**2-dxx/l**4)
t22=np.kron([[0,0,0],[0,1,0],[0,0,0]], 1/l**2-dyy/l**4)
t33=np.kron([[0,0,0],[0,0,0],[0,0,1]], 1/l**2-dzz/l**4)

t12=np.kron([[0,1,0],[1,0,0],[0,0,0]], -dxy/l**4)
t13=np.kron([[0,0,1],[0,0,0],[1,0,0]], -dxz/l**4)

t23=np.kron([[0,0,0],[0,0,1],[0,1,0]], -dyz/l**4)
t2=t11+t22+t33+t12+t13+t23

##calcular el kernel final

K=t1*t2
return(K)
```

Figura 32. Generación del *kernel* con curvatura nula de $\mathbb{R}^3 \rightarrow \mathbb{R}^3$

Ahora, se aplica *kernel* a la malla definida y se procede a construir, simular y graficar la matriz de covarianzas obtenida:

```
X=np.array([[1, 0],[0, 1]])
X[0]

l=0.3
K = curlFreeKern(X[:,0],X[:,1],l)
```

Figura 33. Código para generar un *kernel* con curvatura nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$, con $l = 0{,}3$

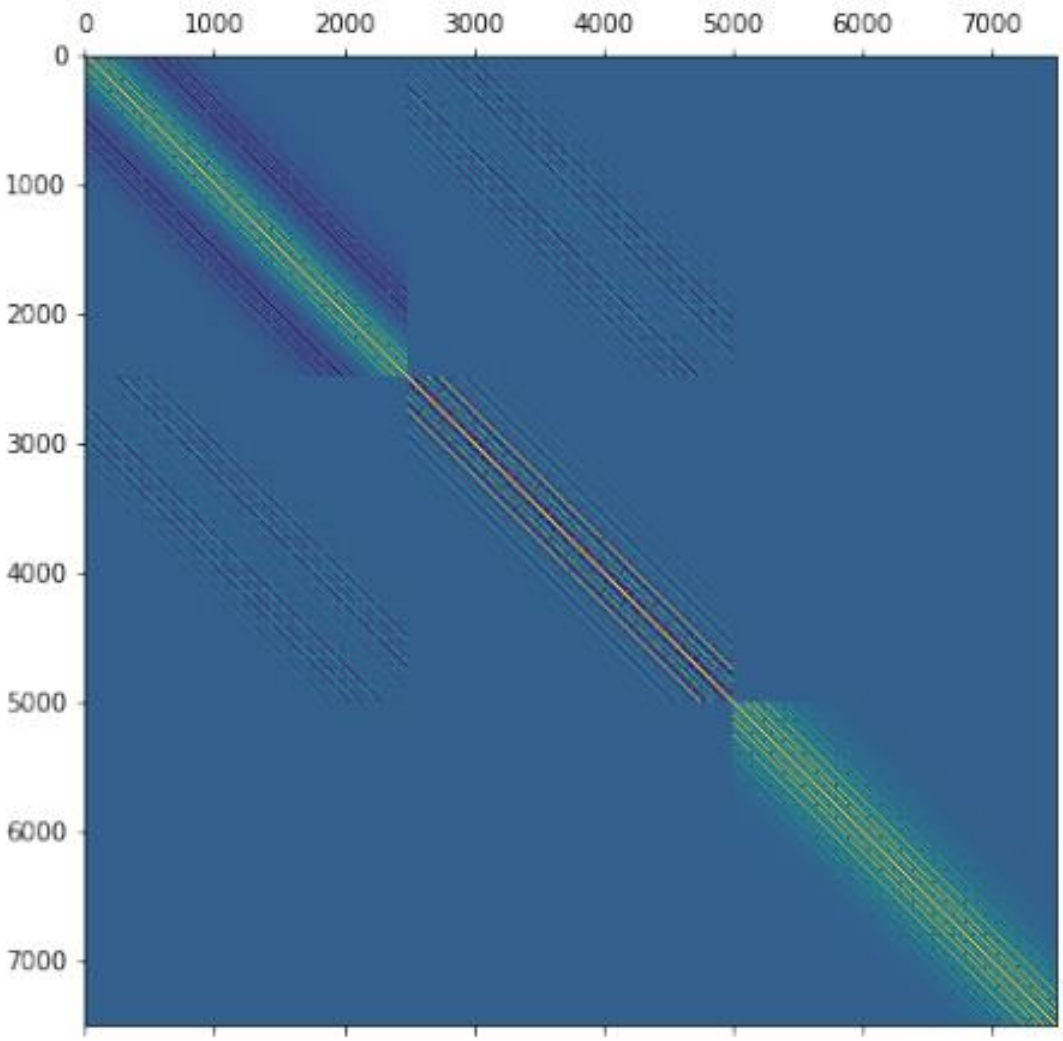


Figura 34. Representación gráfica de un *kernel* con curvatura nula $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$, con $l = 0{,}3$

Al igual que en el caso de la divergencia nula, se presentan los valores propios de la matriz que representa el *kernel* para verificar que todos sean mayores o iguales a cero, condición para verificar que sea definida positiva:

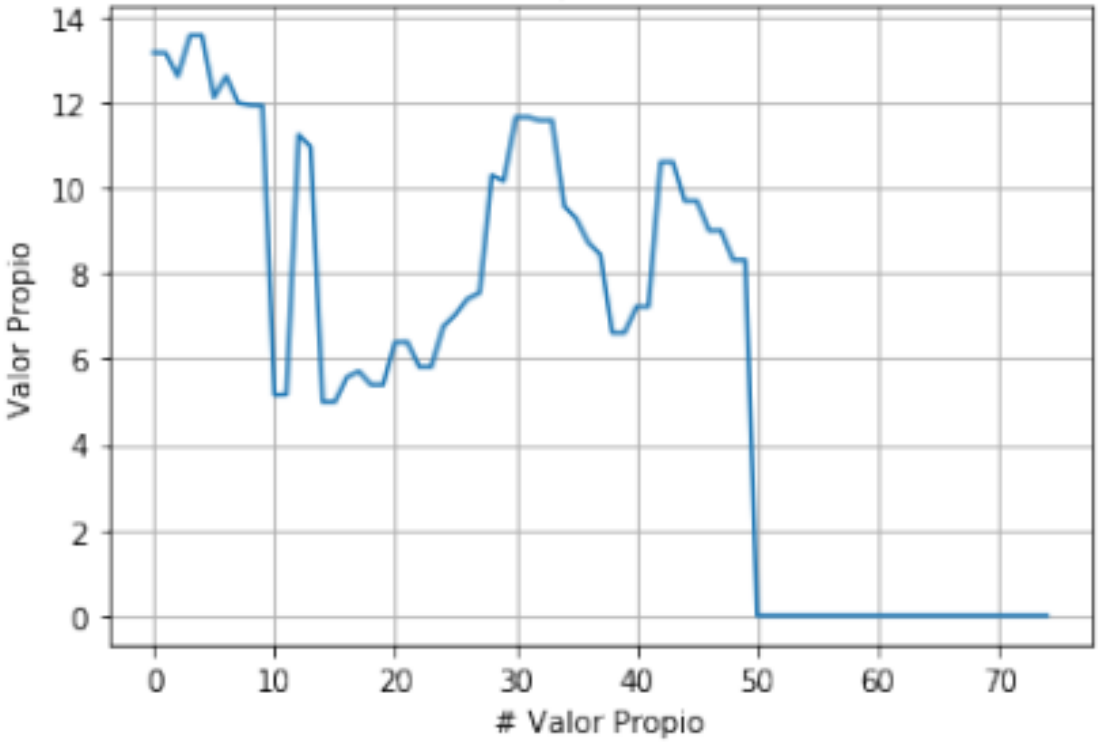


Figura 35. Valores propios de *kernel* con curvatura nula

Siguiendo la metodología presentada para el caso de la divergencia nula, se presentan a continuación el código correspondiente (Figura 36) y la visualización de campos vectoriales dependientes con curvatura nula (Figura 37):

```
##definir semilla de generación para poder replicar resultados

np.random.seed(0)

##definir el kernel con curvarura nula

mu = np.zeros((d+1)*nxGrid*nxGrid)
Z2 = np.random.multivariate_normal(mu, curlFreeK, 4)

##tomar todos los puntos de la grilla

npoints=nxGrid

##generar los campos vectoriales

dx0 = Z2[0][0:nxGrid*nxGrid]
dy0 = Z2[0][nxGrid*nxGrid:d*nxGrid*nxGrid]
dz0 = Z2[0][d*nxGrid*nxGrid:(d+1)*nxGrid*nxGrid]
```

Figura 36. Código para generar dos campos vectoriales aleatorios con rotacional nulo de $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$, con $l = 0{,}3$

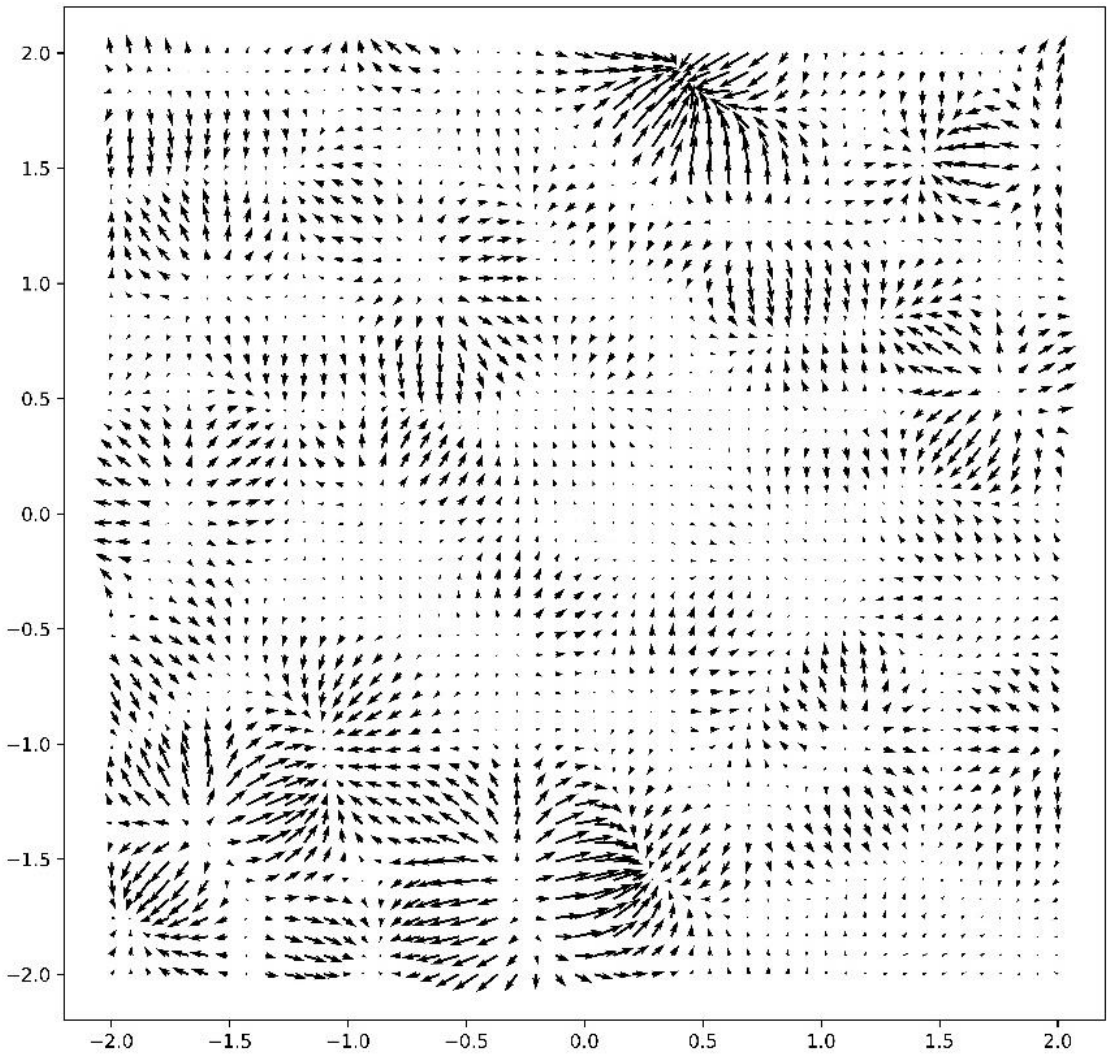


Figura 37. Representación de campos vectoriales aleatorios con rotacional nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^{\mathbf{2}}$, con $l = 0{,}3$

Los campos con rotacional nulo representan físicamente a campos de fuerza compresivos (Wahlström et al., 2013). Un campo con esta propiedad es aquel que no tiene tendencia de giro; este tipo de campos se pueden observar en flujos perfectamente laminares y también en flujos altamente compresivos, donde las fuerzas generan sitios de alta presión y densidad y otras regiones de vacío (sin depositar vorticidad). Un ejemplo es una región del medio interestelar que ha sido barrida por ondas de choque. Si las ondas de choque son supersónicas, la presión en ciertas regiones del gas puede incrementar la presión y densidad en varios órdenes de magnitud y dejar atrás zonas de rarefacción (baja densidad) (Banda-Barragán, Federrath, Crocker, & Bicknell, 2018; Banda-Barragán et al., 2019).

Al igual que para la divergencia, se procede a comprobar que el campo posee su rotacional nulo, utilizando diferencias finitas de segundo orden (ver Figura 38). En este caso, los valores del rotacional están entre -0.5367 y 0.6437 (aproximadamente, 0).

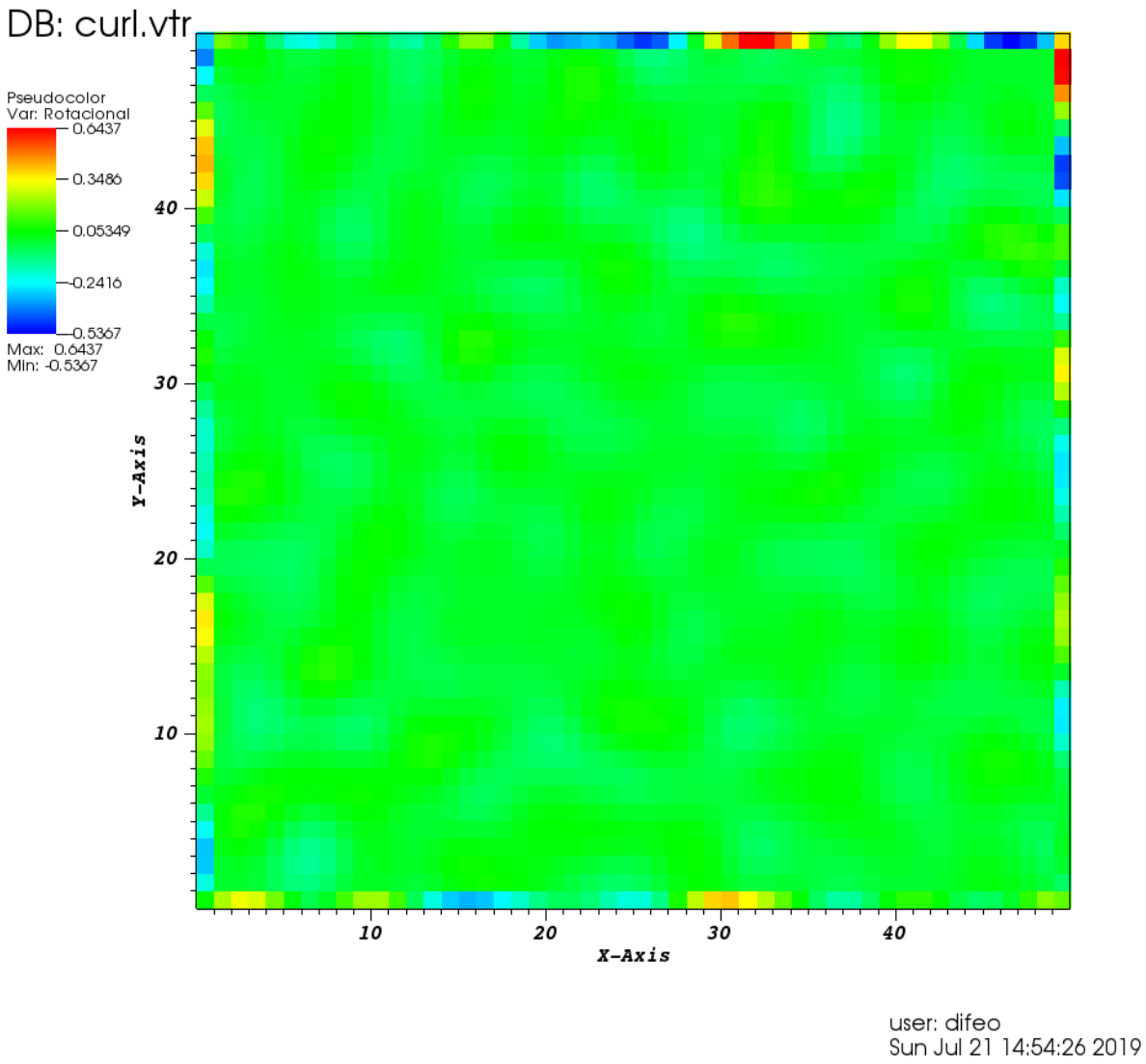


Figura 38. Representación del campo vectorial aleatorio con rotacional nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$, con $l = 0{,}3$

Dado que en este campo se asegura que el rotacional sea nulo, no se restringe la divergencia de ninguna manera; por esto, la divergencia está entre los valores de -10.850 y 8.534 (ver Figura 39).

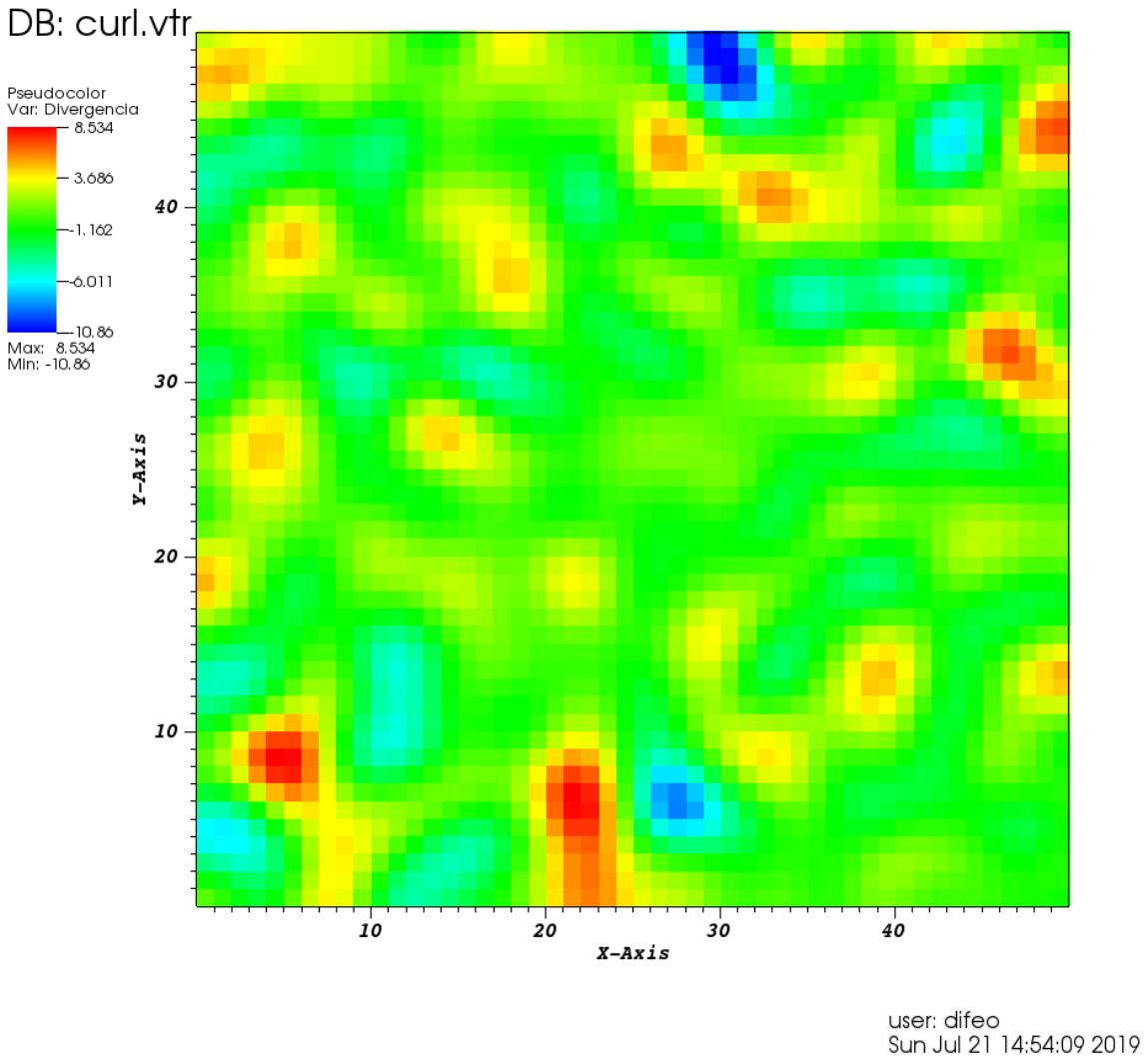


Figura 39. Representación de la divergencia de campos vectoriales aleatorios con rotacional nula de $\mathbb{R}^2 \longrightarrow \mathbb{R}^2$, con $l = 0{,}3$

4.5 Comparación de simulaciones

Adicionalmente a lo expuesto en las simulaciones de campos vectoriales con divergencia y rotacional nulo, en los que se puede observar que cumplen con las condiciones necesarias en cada caso y que la herramienta implementada logra generar dichos campos, se procede a realizar una validación adicional considerando dos aspectos:

El primer aspecto a considerar es el tiempo que tarda la herramienta en generar los campos solicitados, en comparación con el tiempo que tardaría otra herramienta en hacerlo; en este caso, la herramienta que se ha tomado PLUTO (ver Mignone et al., 2007).

El segundo aspecto es la consideración de un campo magnético inicialmente uniforme, sujeto a fuerzas de agitación siguiendo el modelo de Federrath et al. (Federrath, Roman-Duval, Klessen, Schmidt, & Mac Low, 2009).

4.5.1 Comparación de tiempos

Para comparar los tiempos de generación de los campos vectoriales, se tomaron como referencia las simulaciones obtenidas a través de PLUTO. Así, mientras que en PLUTO la simulación tarda un total de 169 segundos, en el caso de la herramienta implementada, toma en promedio 82 segundos (ver Figura 40):

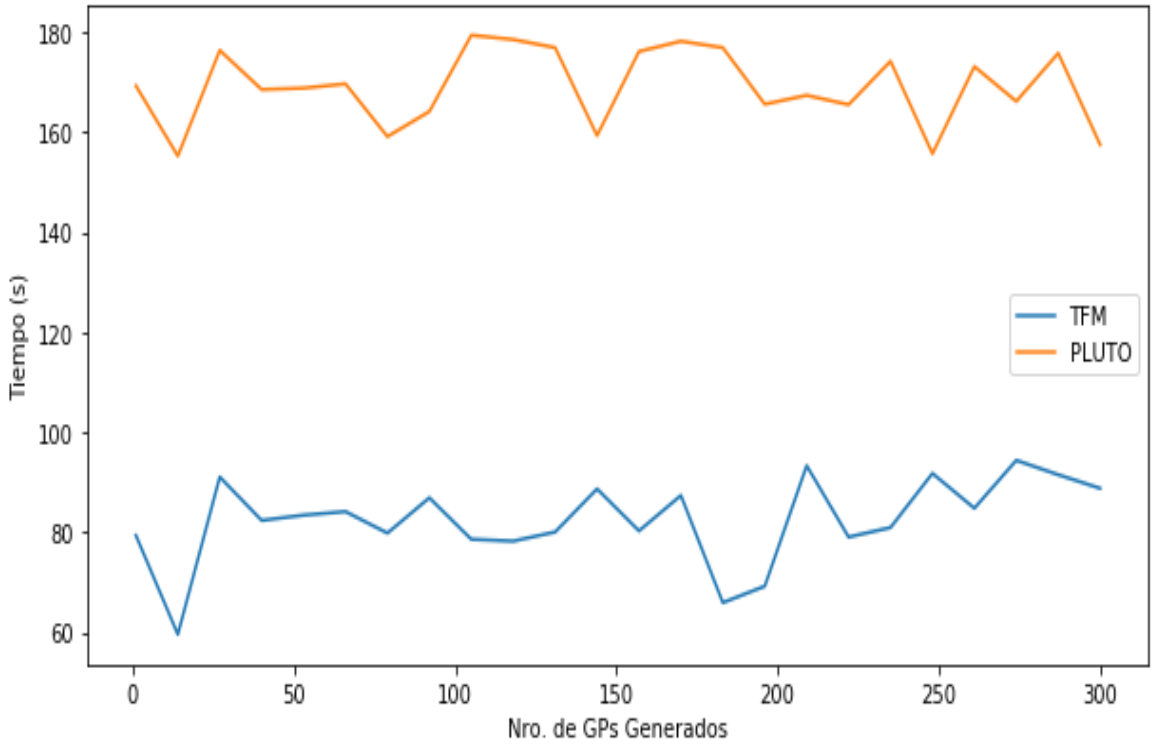


Figura 40. Comparación del tiempo en la generación de campos vectoriales entre PLUTO y el código generado en el TFM

4.5.2 Comparación de simulaciones de un campo magnético

En este caso, se toma un campo magnético inicialmente uniforme, pero sujeto a fuerzas de agitación, para verificar cómo reacciona la herramienta desarrollada ante este fenómeno; se simula el mismo campo en PLUTO para comparar los resultados.

Con el programa PLUTO, en primer lugar se genera un campo uniforme.

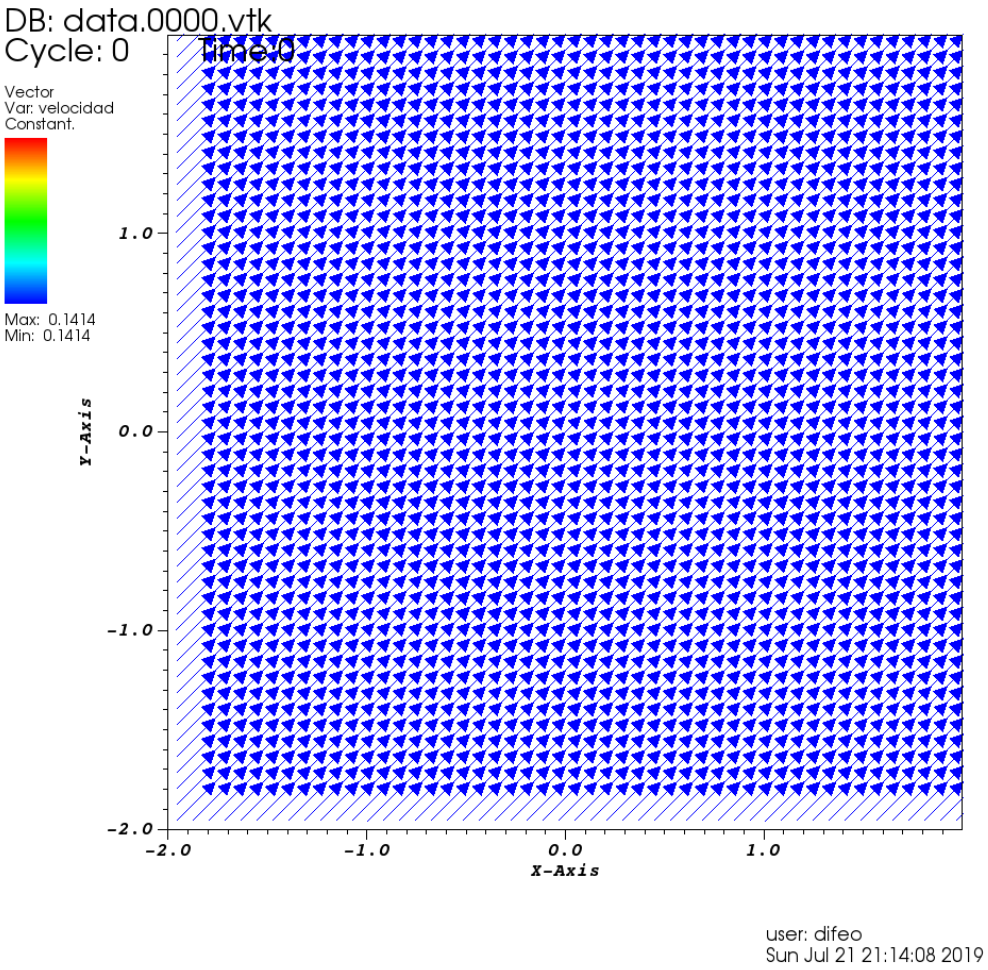


Figura 41. Campo uniforme generado con PLUTO

Luego, es necesario introducir fuerzas de agitación en las ecuaciones de momento y energía como funciones que varían en espacio y tiempo; de esta manera, se generan modos de turbulencia en el gas:

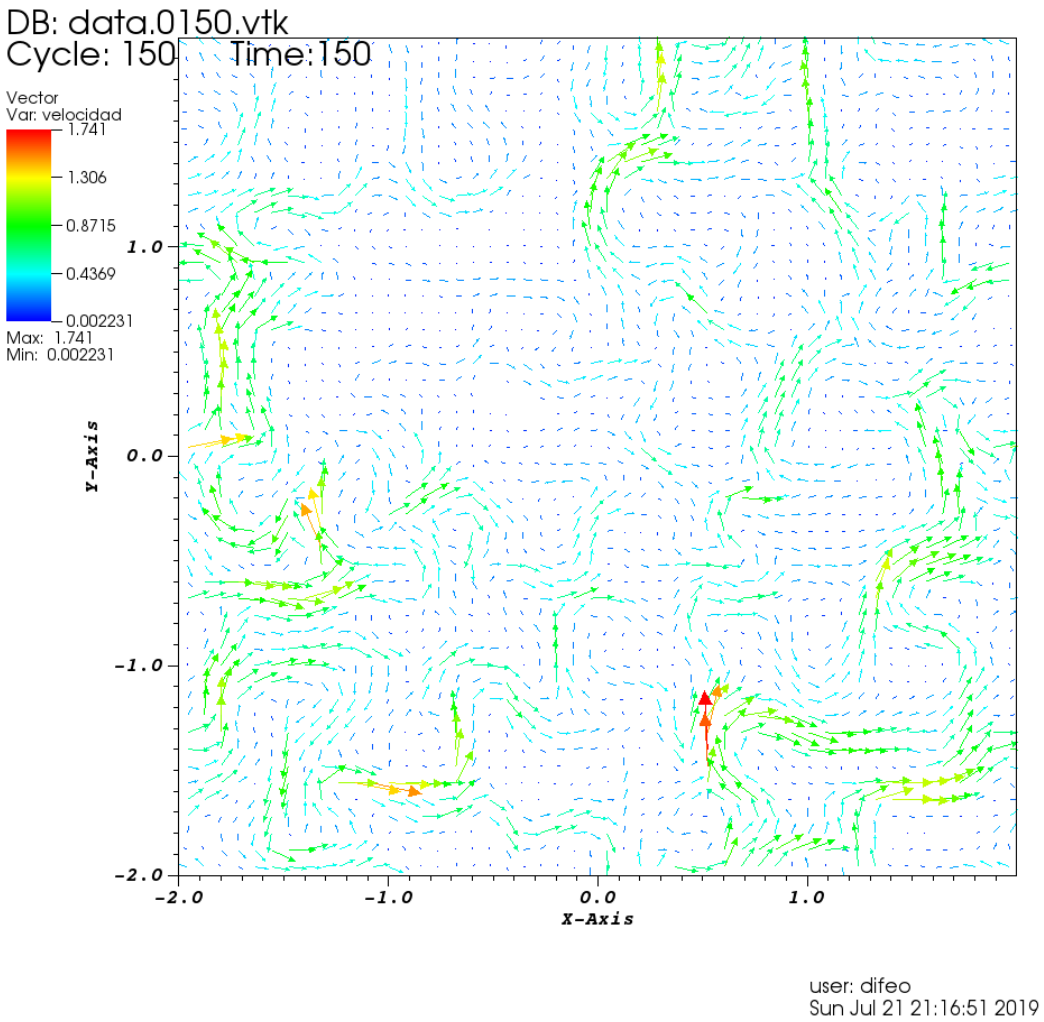


Figura 42. Campo con turbulencia desarrollada en $t = 150$

Finalmente, se obtienen las aproximaciones para la divergencia y el rotacional:

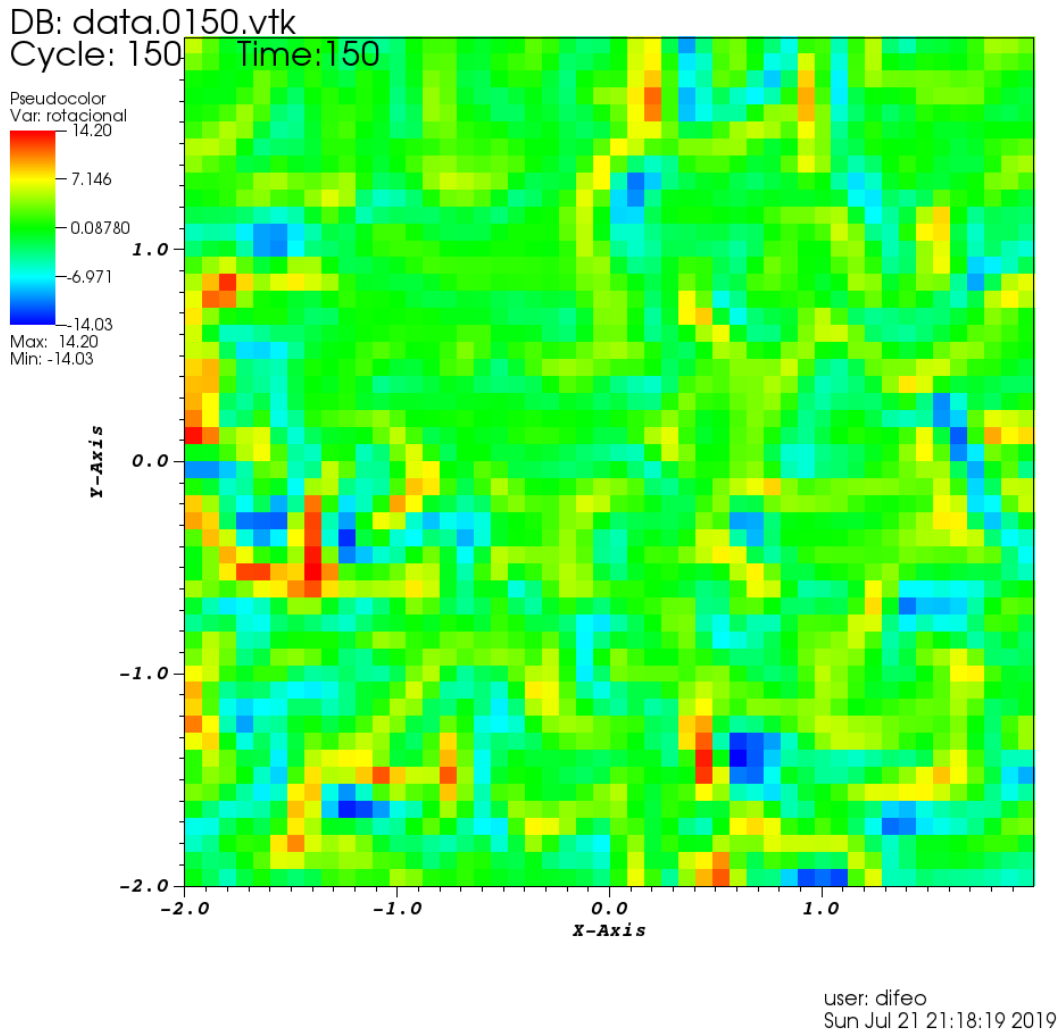


Figura 43. Rotacional del campo vectorial generado por PLUTO en $t = 150$

En la Figura 43 se puede observar que el rotacional del campo vectorial está entre -14.003 y 14.200; es decir, no cumple la característica de que el rotacional sea nulo. Por otro lado, en la Figura 44, se muestra la estimación de la divergencia; en este caso, aunque los valores son menores, se encuentran entre -2.699 y 3.465 y tampoco se cumple el supuesto de divergencia nula.

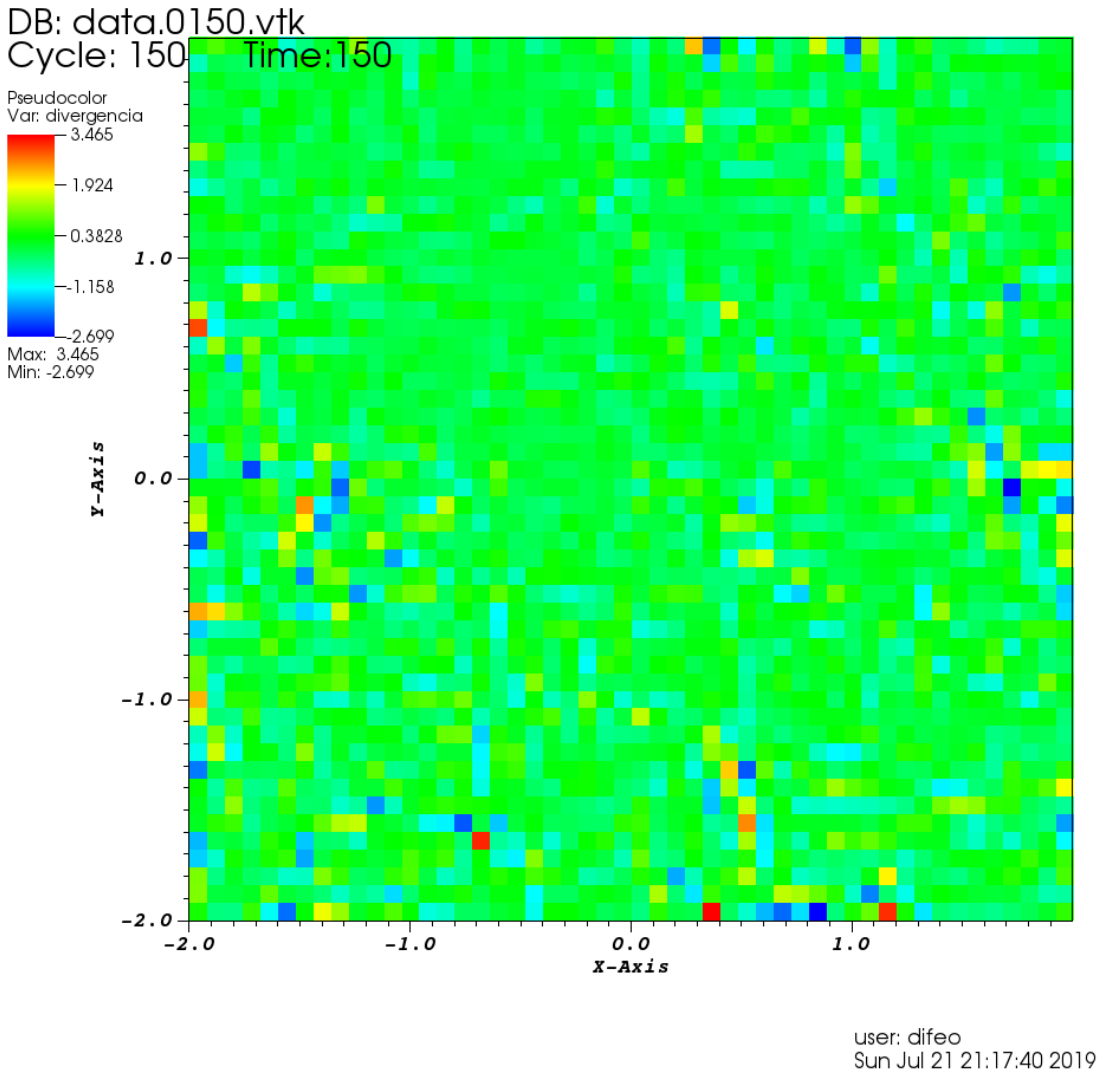


Figura 44. Divergencia del campo vectorial generado con PLUTO en $t = 150$

Ahora, con la herramienta generada en este proyecto, no es necesario realizar los pasos previos de generar un campo uniforme y someterlo a interacciones con fuerzas, sino que los resultados de divergencia y rotacional son inmediatos; la estimación de la divergencia nula y del rotacional nulo se presentan a continuación:

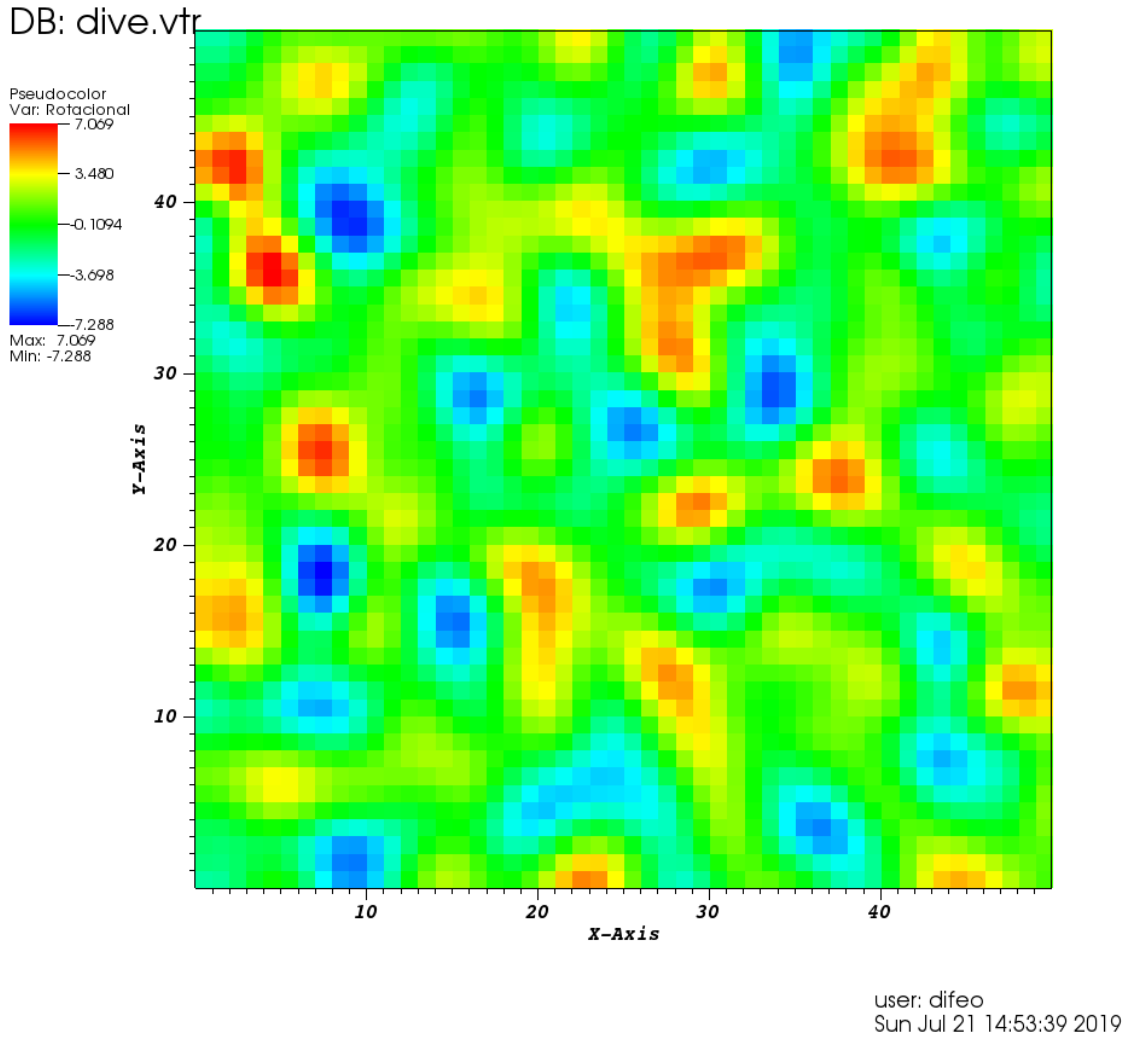


Figura 45. Rotacional del campo vectorial generado con la herramienta desarrollada en el proyecto

En las Figura 45 y Figura 46 se puede observar que, con la herramienta generada en este proyecto, los valores tanto del rotacional como de la divergencia son menores a los que se estimaron con PLUTO (ver Figura 43 y Figura 44); por ejemplo, el rotacional está entre -7.288 y 7.069 y la divergencia entre -0.5352 y 0.4762. Aunque en este caso el rotacional no es nulo, se puede concluir que la divergencia sí lo es; por tanto, con la herramienta desarrollada en el presente proyecto se ha logrado obtener de manera adecuada la característica de divergencia nula que debe cumplir el campo en análisis.

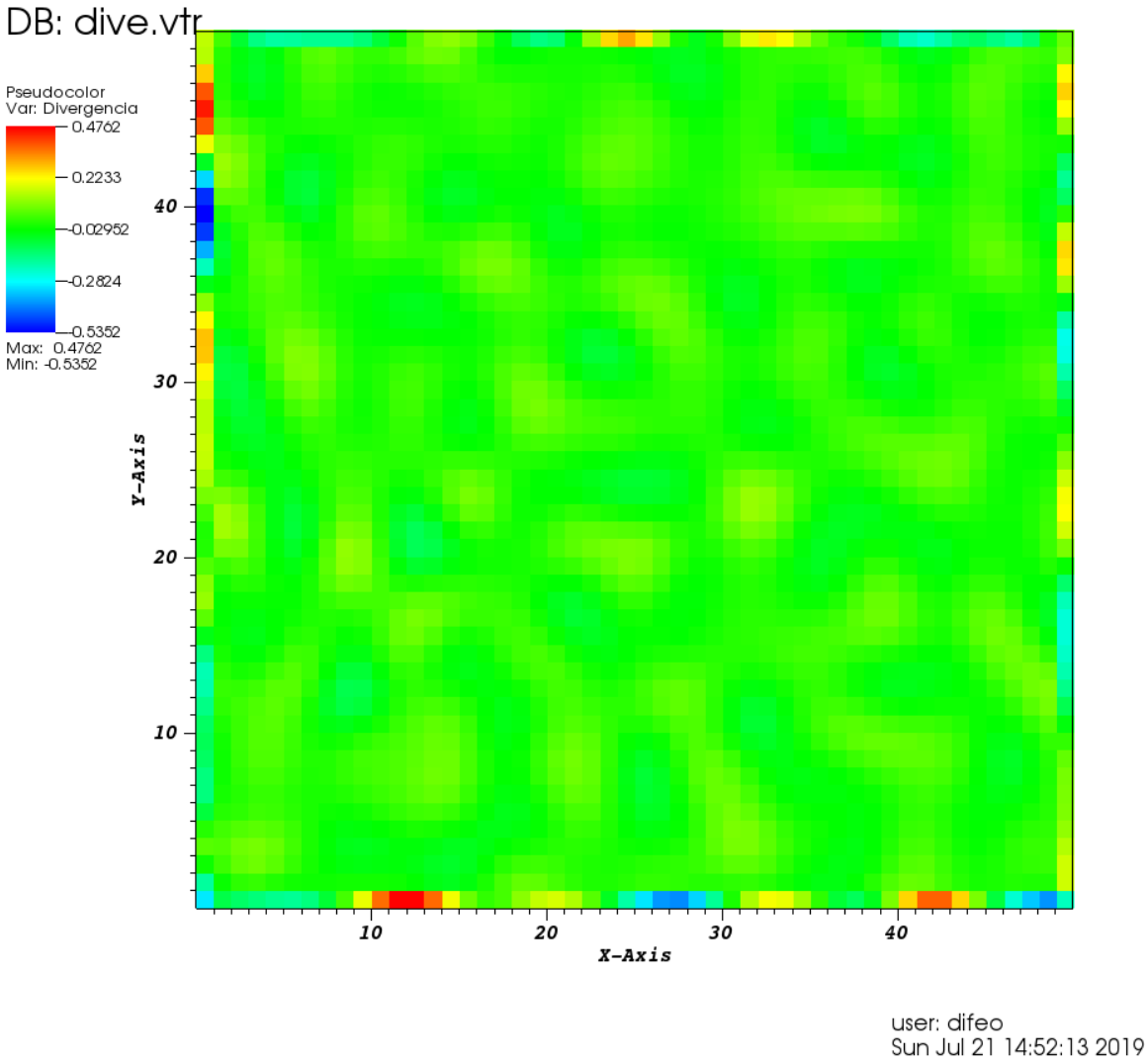


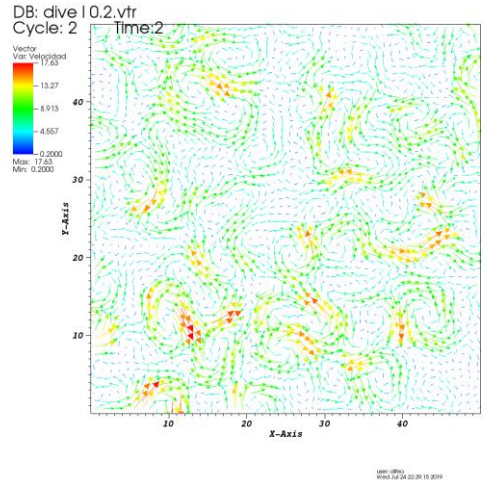
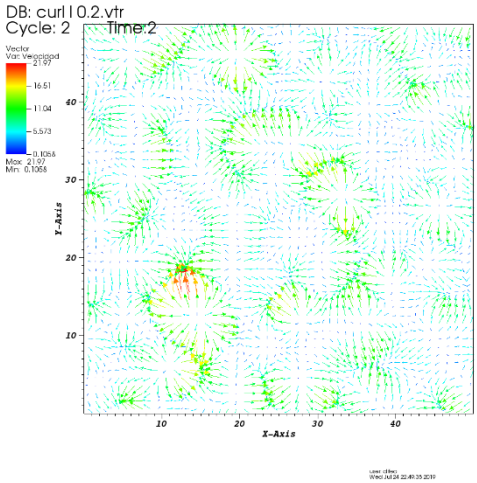
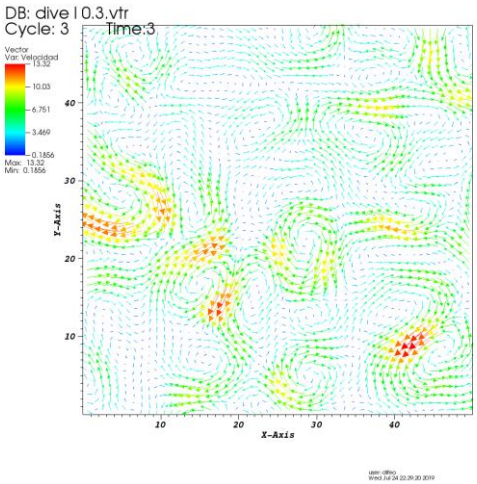
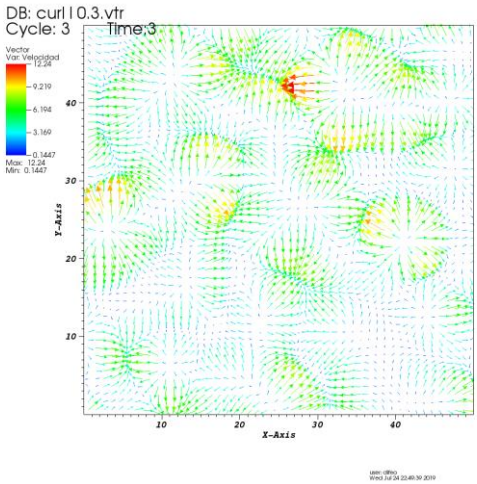
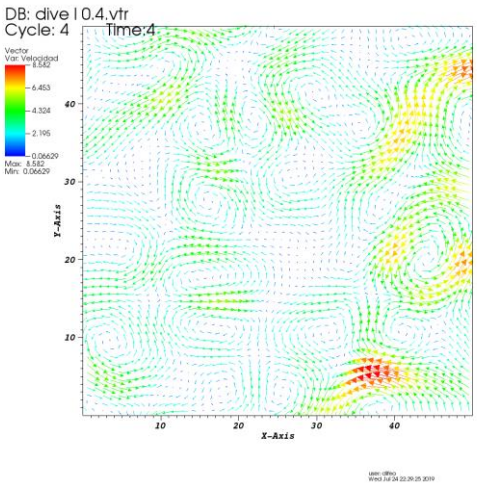
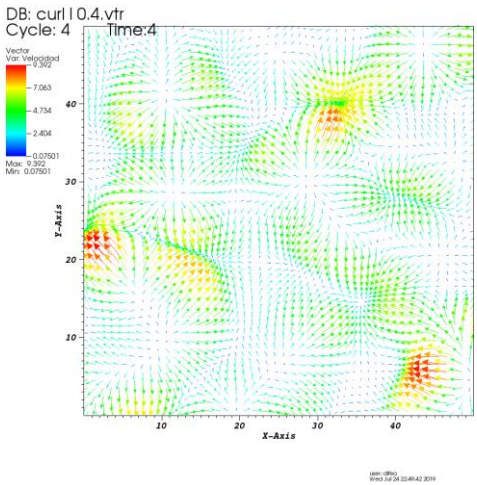
Figura 46. Divergencia del campo vectorial generado con la herramienta desarrollada en el proyecto

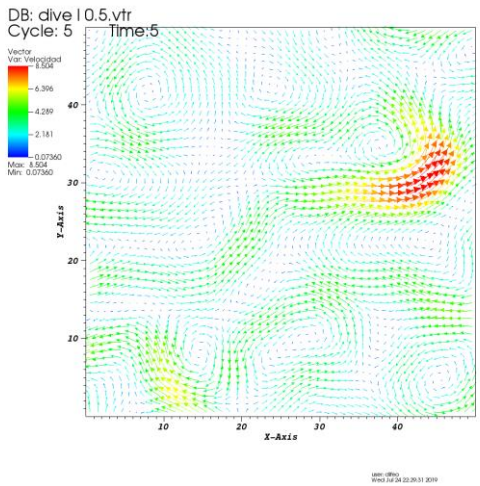
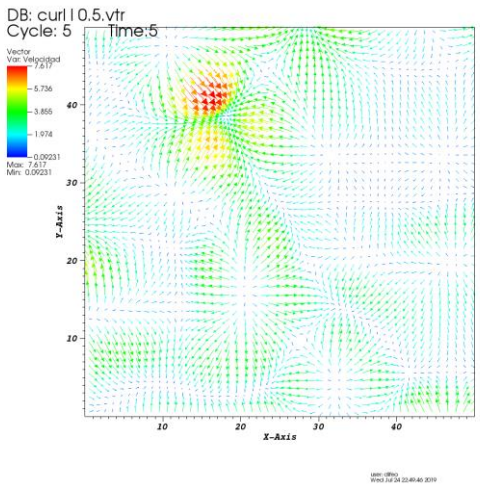
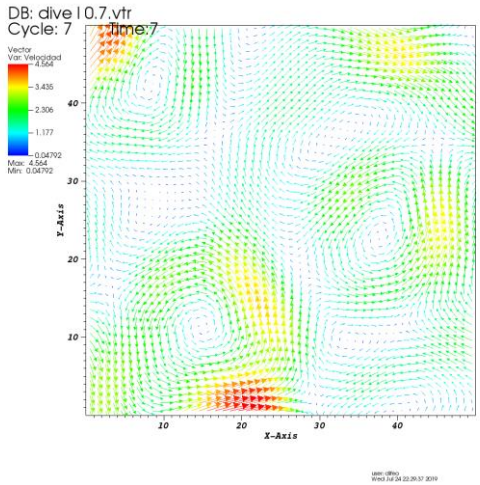
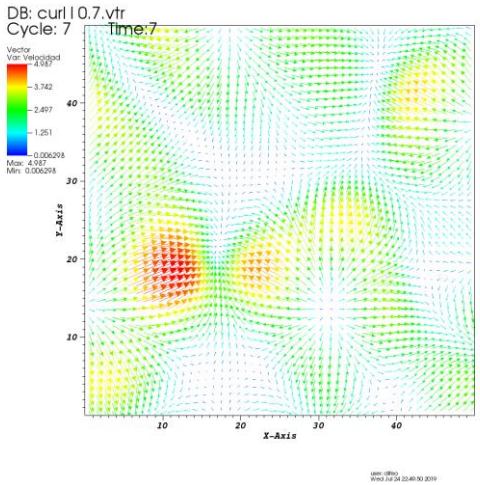
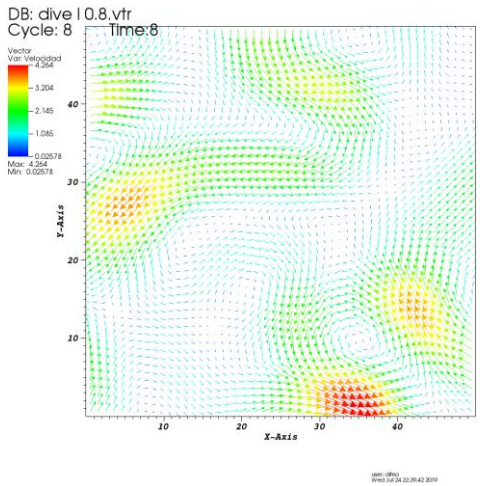
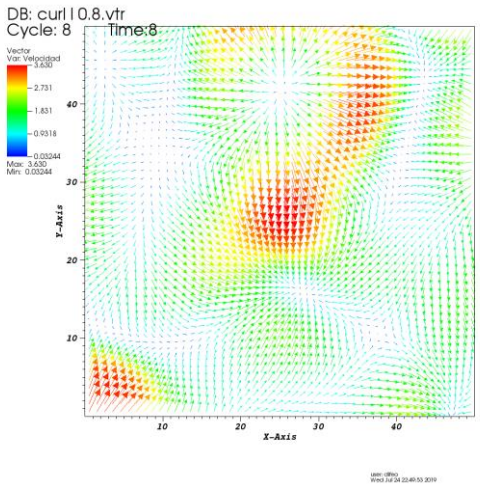
4.6 Simulaciones con variación de ℓ

Es importante señalar que, dentro de la herramienta implementada, se considera el valor del hiperparámetro ℓ como un input arbitrario en la generación del *kernel* y, por tanto, del campo vectorial asociado pero se mantiene fijo durante todo el proceso.

Por tanto, es importante verificar, cómo cambian los campos vectoriales y sus características (divergencia nula y rotacional nulo) cuando se cambia ℓ. A continuación, se presentan unos campos vectoriales con diferentes valores del hiperparámetro.

En primer lugar, se presentan los campos vectoriales generados con divergencia nula y rotacional nulo:

ℓ	Divergencia Nula	Rotacional Nulo
0.2		
0.3		
0.4		

0.5	 DB: dive l 0.5.vtr Cycle: 5 Time:5 X-Axis Y-Axis	 DB: curl l 0.5.vtr Cycle: 5 Time:5 X-Axis Y-Axis
0.7	 DB: dive l 0.7.vtr Cycle: 7 Time:7 X-Axis Y-Axis	 DB: curl l 0.7.vtr Cycle: 7 Time:7 X-Axis Y-Axis
0.8	 DB: dive l 0.8.vtr Cycle: 8 Time:8 X-Axis Y-Axis	 DB: curl l 0.8.vtr Cycle: 8 Time:8 X-Axis Y-Axis

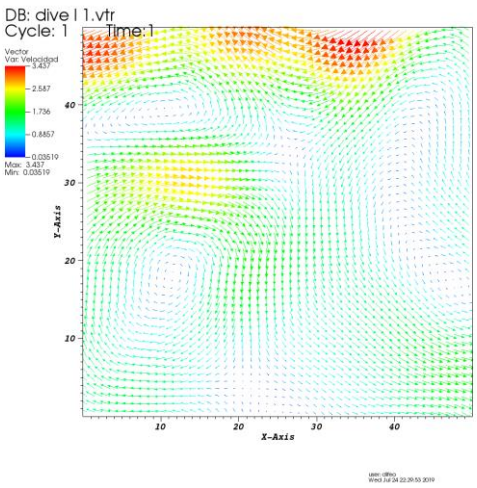
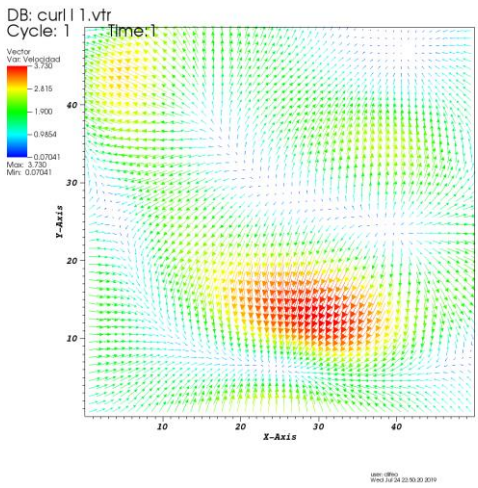
1		

Figura 47. Simulación de campos con divergencia nula y rotacional nulo para distintos valores de ℓ

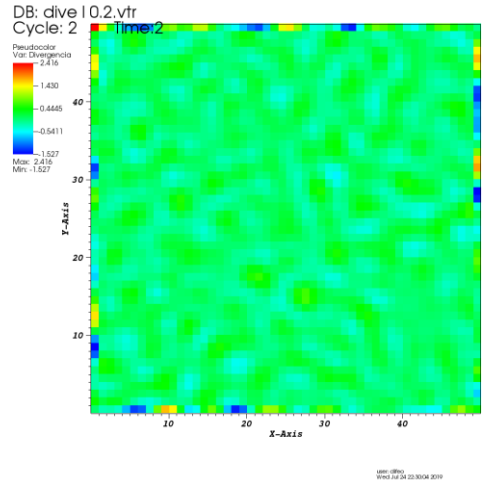
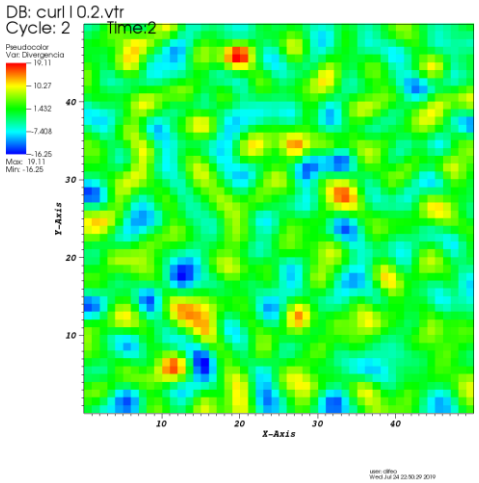
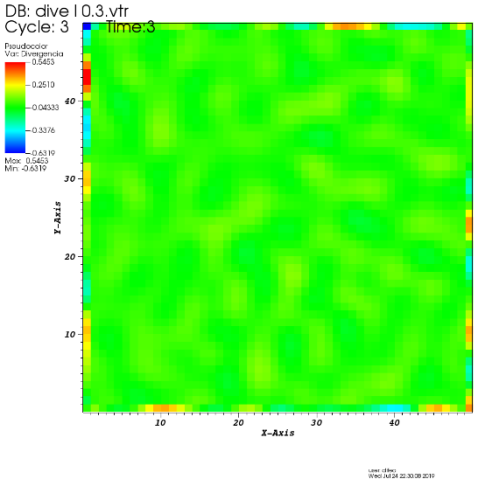
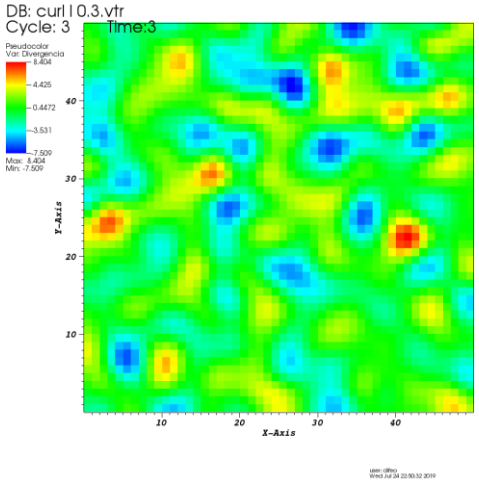
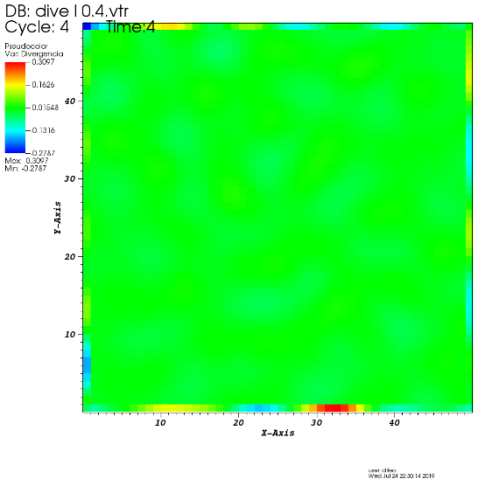
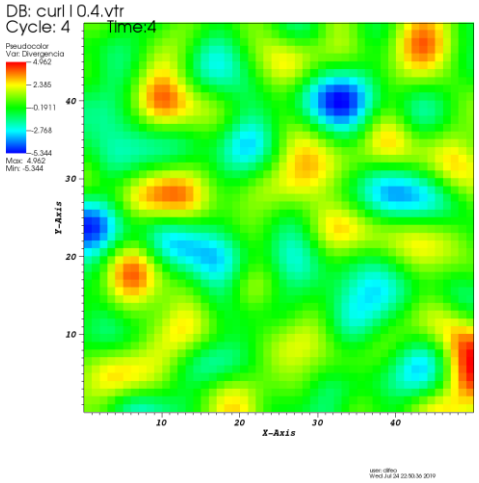
Como se puede observar en la Figura 47, los campos generados para cada uno de los valores de ℓ cumplen con las características visuales que se esperan en cada tipo de campos; esto es:

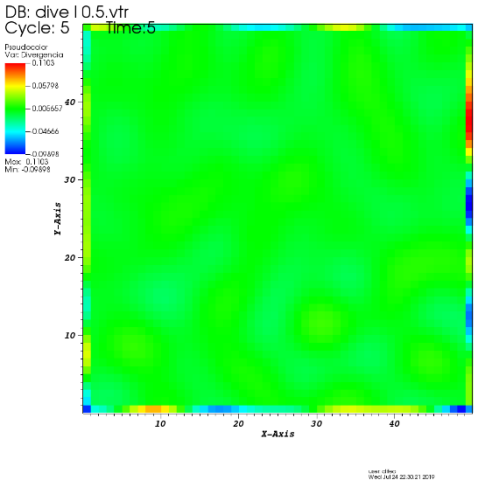
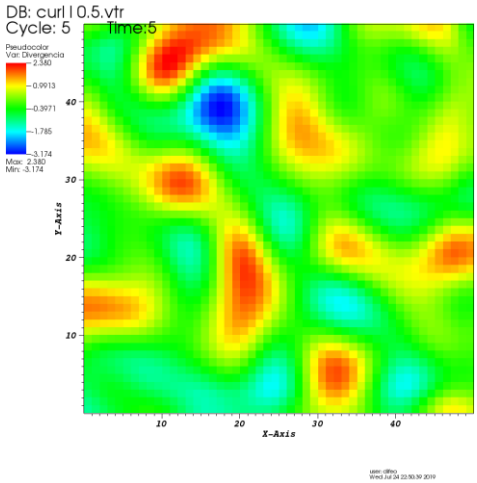
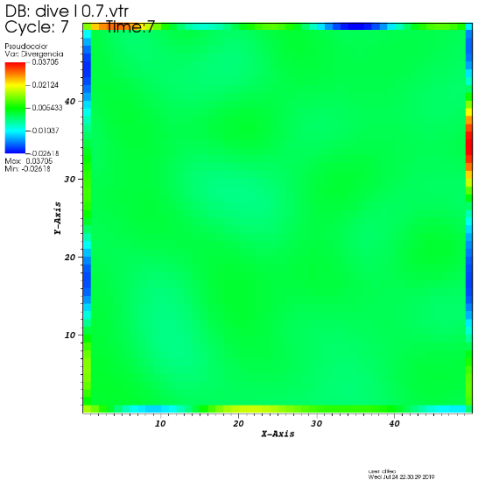
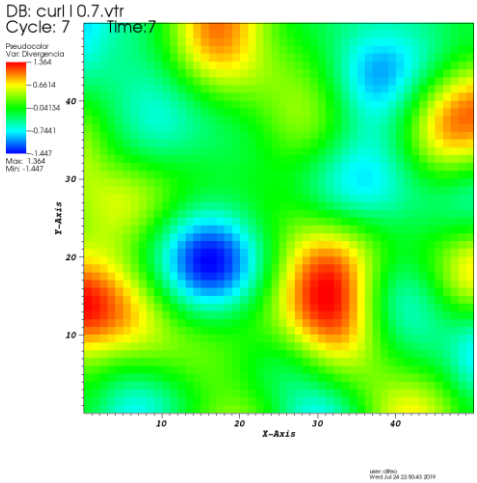
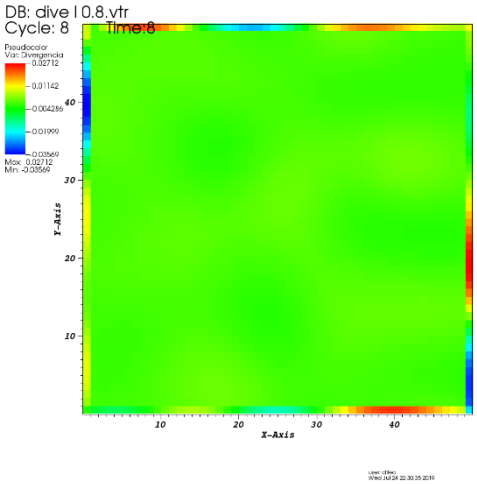
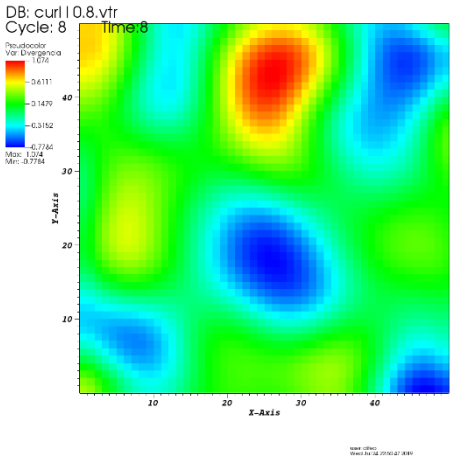
- Remolinos en los campos de divergencia nula.
- Puntos convergentes (acumulación) en los campos con rotacional nulo.

En los siguientes apartados se muestran las aproximaciones de la divergencia y del rotacional para cada uno de los campos vectoriales generados.

4.6.1 Divergencia de los campos vectoriales

A parir de los campos vectoriales presentados en la Figura 47, se realiza la estimación de la divergencia de dichos campos. Cabe recalcar que los campos vectoriales con divergencia nula, no tienen restricciones sobre el rotacional.

ℓ	Divergencia Nula	Rotacional Nulo
0.2		
0.3		
0.4		

0.5	 DB: dive l 0.5.vtr Cycle: 5 Time:5 Pseudocolor Var: Divergencia 0.1163 0.05798 0.006663 -0.04666 -0.09598 Max: 0.1163 Min: -0.09598 Y-Axis X-Axis	 DB: curl l 0.5.vtr Cycle: 5 Time:5 Pseudocolor Var: Divergencia 2.380 0.9913 -0.3971 -1.785 -3.174 Max: 2.380 Min: -3.174 Y-Axis X-Axis
0.7	 DB: dive l 0.7.vtr Cycle: 7 Time:7 Pseudocolor Var: Divergencia 0.03705 0.02124 0.005433 -0.01037 -0.02618 Max: 0.03705 Min: -0.02618 Y-Axis X-Axis	 DB: curl l 0.7.vtr Cycle: 7 Time:7 Pseudocolor Var: Divergencia 1.364 0.6614 -0.04134 -0.7441 -1.447 Max: 1.364 Min: -1.447 Y-Axis X-Axis
0.8	 DB: dive l 0.8.vtr Cycle: 8 Time:8 Pseudocolor Var: Divergencia 0.02712 0.01142 -0.004286 -0.01999 -0.03569 Max: 0.02712 Min: -0.03569 Y-Axis X-Axis	 DB: curl l 0.8.vtr Cycle: 8 Time:8 Pseudocolor Var: Divergencia 1.078 0.6111 0.1479 -0.3152 -0.7784 Max: 1.078 Min: -0.7784 Y-Axis X-Axis

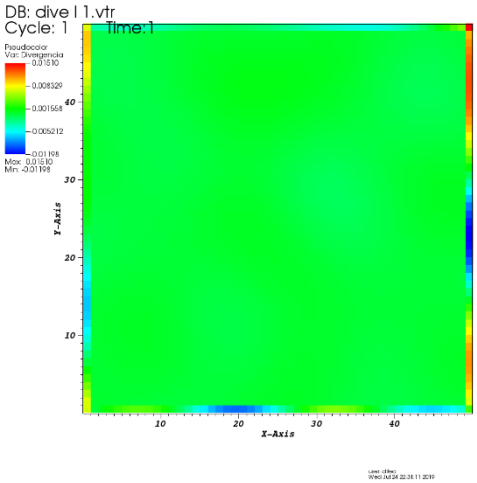
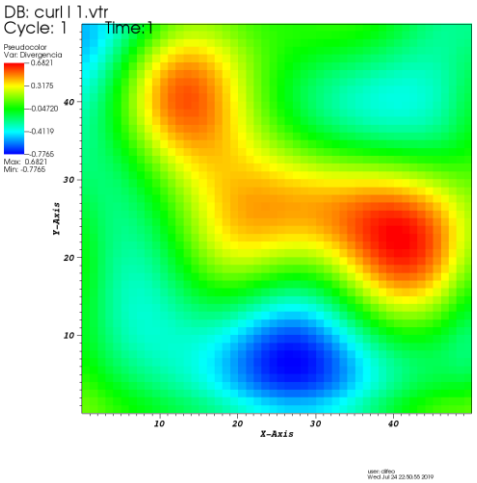
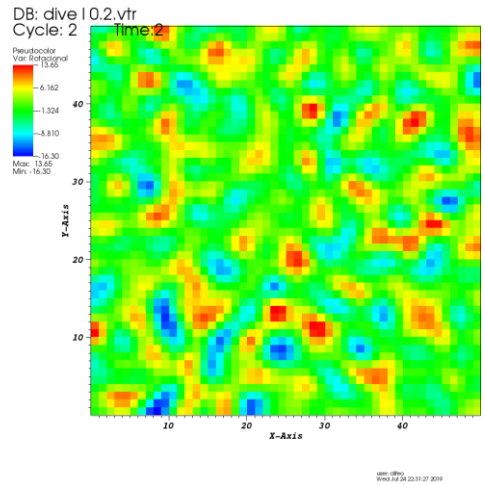
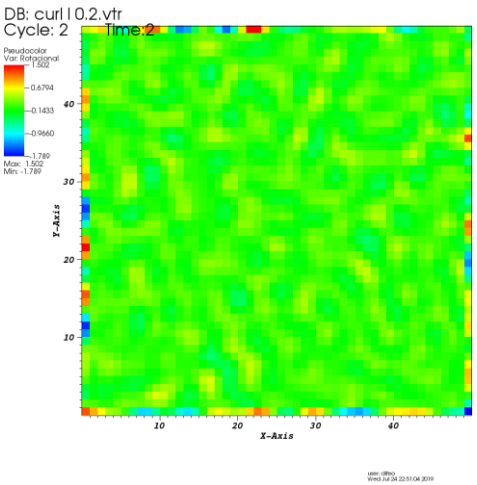
1		

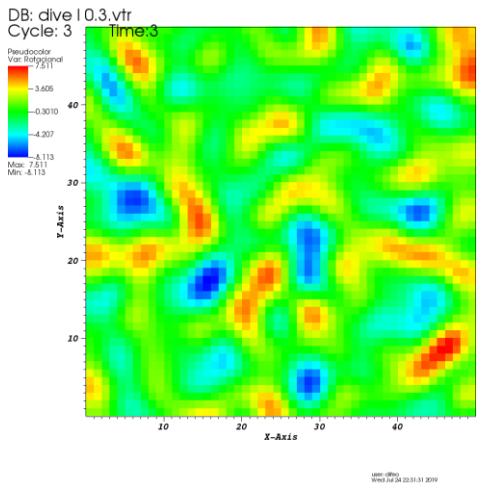
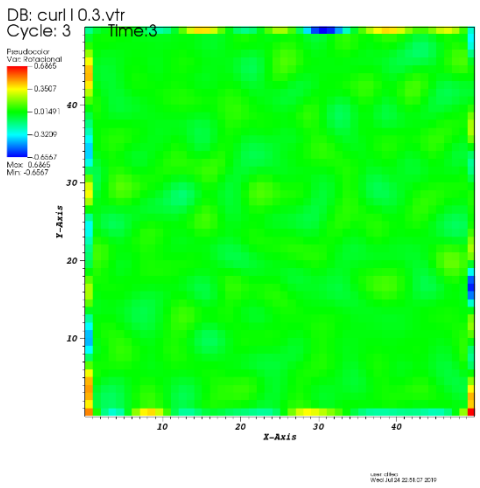
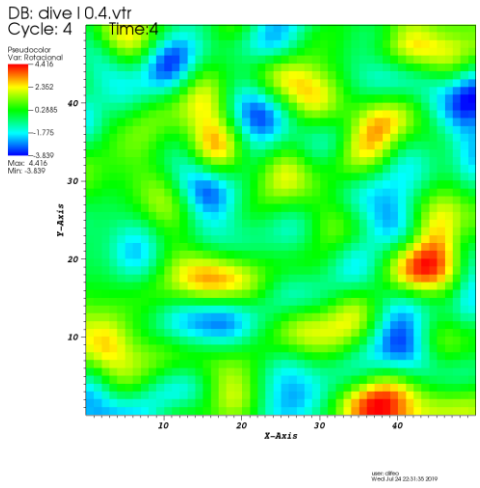
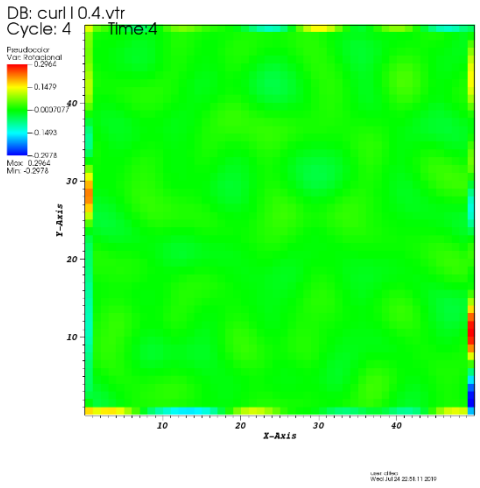
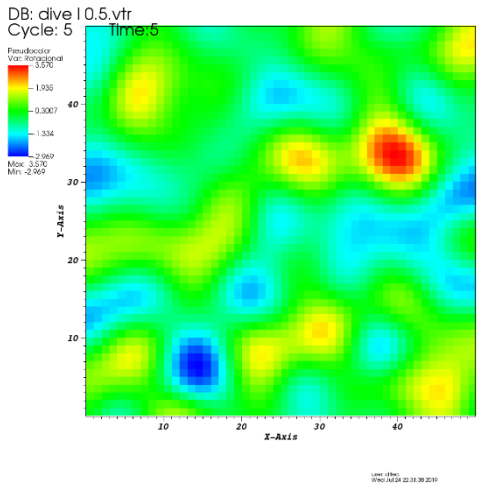
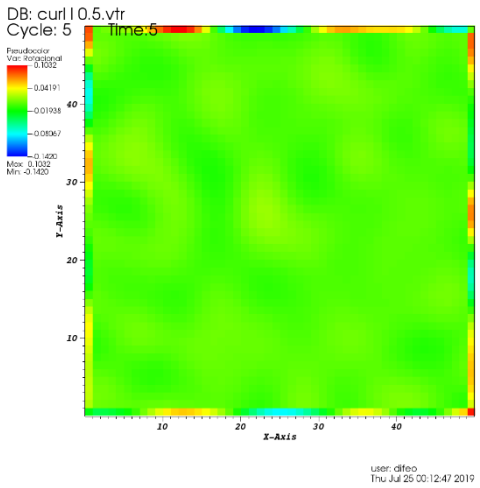
Figura 48. Divergencia de campos vectoriales con divergencia nula y rotacional nulo

En la Figura 48 se puede observar que para todos los valores de ℓ la divergencia está en promedio de 0 para los campos vectoriales con divergencia nula; sin embargo, esto no se cumple cuando el rotacional es nulo ya que varían entre 0,6821 y -0,7765.

4.6.2 Rotacional de los campos vectoriales

En esta sección se presentan los valores calculados para los rotacionales para cada uno de los campos vectoriales generados. La Figura 49 muestra dichos resultados.

ℓ	Divergencia Nula	Rotacional Nulo
0.2		

0.3	 DB: dive I 0.3.vtr Cycle: 3 Time:3 Pseudocolor Var: Rotacional 7.811 3.606 -0.3010 -4.207 -8.113 Max: 7.811 Min: -8.113 Y-Axis X-Axis	 DB: curl I 0.3.vtr Cycle: 3 Time:3 Pseudocolor Var: Rotacional 0.6665 0.3507 0.01491 -0.3299 -0.6657 Max: 0.6665 Min: -0.6657 Y-Axis X-Axis
0.4	 DB: dive I 0.4.vtr Cycle: 4 Time:4 Pseudocolor Var: Rotacional 4.416 2.352 0.2885 -1.775 -3.839 Max: 4.416 Min: -3.839 Y-Axis X-Axis	 DB: curl I 0.4.vtr Cycle: 4 Time:4 Pseudocolor Var: Rotacional 0.2964 0.1479 0.0007077 -0.1493 -0.2978 Max: 0.2964 Min: -0.2978 Y-Axis X-Axis
0.5	 DB: dive I 0.5.vtr Cycle: 5 Time:5 Pseudocolor Var: Rotacional 3.570 1.935 0.3007 -1.334 -2.969 Max: 3.570 Min: -2.969 Y-Axis X-Axis	 DB: curl I 0.5.vtr Cycle: 5 Time:5 Pseudocolor Var: Rotacional 0.1032 0.04191 -0.01938 -0.08067 -0.1420 Max: 0.1032 Min: -0.1420 Y-Axis X-Axis user: difeo Thu Jul 25 00:12:47 2019

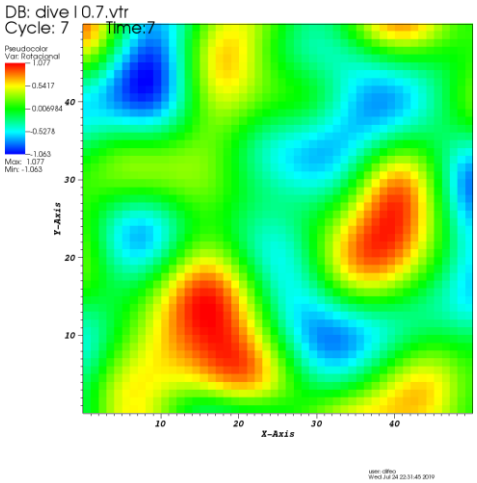
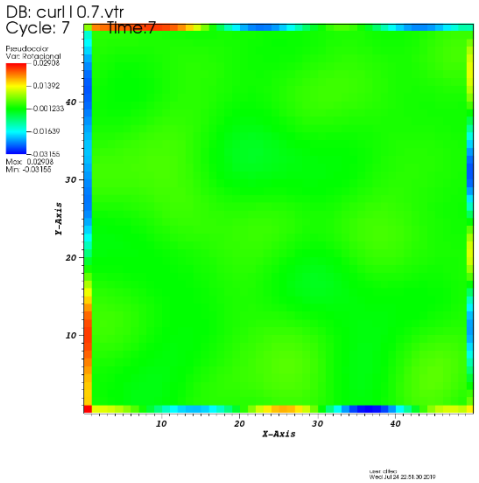
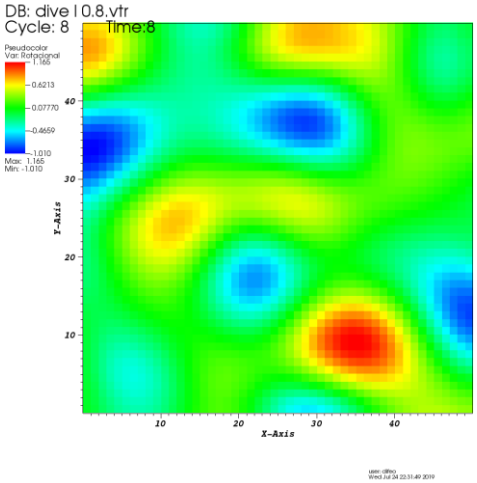
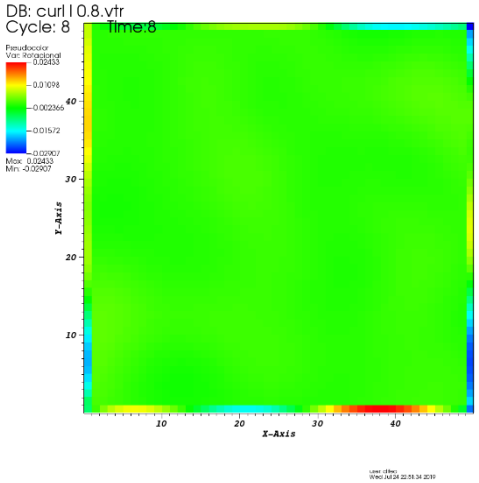
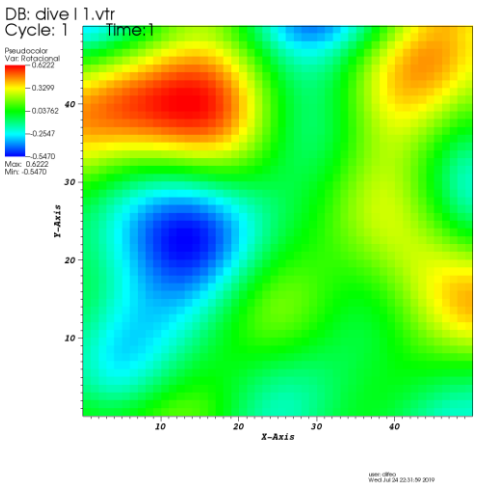
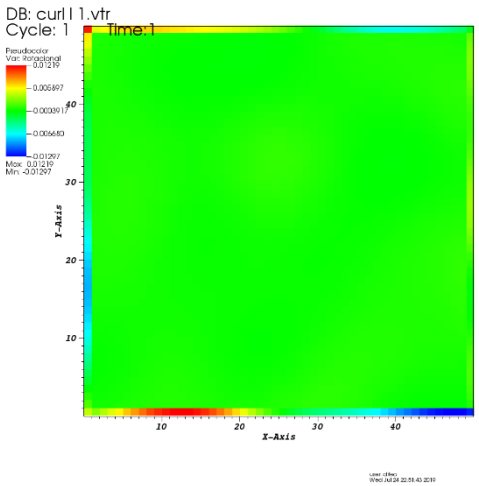
0.7		
0.8		
1		

Figura 49. Rotacional de los campos vectoriales con divergencia nula y rotacional nulo

Como se puede observar, se cumple el supuesto de que el rotacional es aproximadamente 0 para los campos vectoriales con rotacional nulo y dado que no se impone ninguna restricción al rotacional de los campos con divergencia nula, el rotacional de estos campos no llega al valor 0.

A manera de conclusión, se puede decir que, el código implementado en el trabajo genera de manera adecuada campos con características de rotacional nulo y de divergencia nula por separado.

Capítulo 5. Conclusiones y trabajos futuros

Conclusiones

En el presente documento se ha planteado una metodología de trabajo que permite generar campos vectoriales con características de divergencia y rotacional nulo aprovechando las características de los procesos gaussianos, que permiten que se puedan realizar varias simulaciones cambiando la semilla para la generación de números aleatorios.

El método planteado en el presente trabajo ha sido programado en Python porque es un software de acceso libre y ello permitiría la modificación del código desarrollado más fácilmente que si se hubieran empleado otros softwares. Además, las funciones programadas trabajan conjuntamente con las librerías *Numpy* y *GPy* de Python.

En los mapas de divergencia, la herramienta implementada proporciona valores promedio de 0, sin considerar los extremos del gráfico (ya que los valores de los extremos se deben a errores de aproximación por utilizar diferencias de segundo orden en el algoritmo), y en los campos de rotacional, también. Esto concuerda con lo que se aprecia de forma visual en los campos divergentes donde se observan remolinos y en los campos rotacionales donde se observan puntos convergentes.

Por tanto, las salidas obtenidas por la herramienta implementada, en el caso de divergencia nula, representan campos magnéticos en cualquier tipo de gas magnetizado ideal y campos de velocidad en cualquier fluido incompresible. Por otro lado, las salidas que se obtienen cuando se simula la característica de rotacional nulo representan campos de aceleración/fuerza de gases magnetizados altamente compresivos, como por ejemplo los que son sujetos a choques supersónicos.

Se ha verificado que la herramienta implementada produce resultados similares a los obtenidos por otras metodologías de simulación, concretamente el *software* PLUTO. Por tanto, se puede considerar que la herramienta desarrollada en el presente trabajo es válida para simular campos vectoriales con divergencia nula y/o rotacional nula.

En cuanto a los tiempos de simulación, se ha podido constatar que, dentro de que estos son bastante similares a los obtenidos con la simulación generada por PLUTO, son algo menores, lo que se traduce en que la herramienta desarrollada es capaz de mejorar los tiempos de computación y, por tanto, el coste computacional.

Trabajos futuros

En el presente trabajo se han construido campos vectoriales en espacio real. Los campos tienen perturbaciones y al fijar hiperparámetro ℓ se está fijando el tamaño de esas perturbaciones en el dominio. Sería conveniente que se analizase la posibilidad de generar campos vectoriales cuando se hace variar dicho hiperparámetro ya que, por las propiedades algebraicas de la divergencia, al sumar campos con distinto ℓ pero con divergencia nula, el resultado es un campo con divergencia nula pero con perturbaciones a diferentes escalas.

En ese mismo contexto, los campos generados para el presente trabajo no son periódicos, de modo que sería interesante analizar cómo imponer periodicidad al generar campos vectoriales para determinar las características que puedan aportar.

Los campos generados en el presente trabajo se pueden utilizar dentro de las simulaciones CFD, tomándolos como condiciones iniciales para analizar fenómenos más complejos. Además, es importante el poder extender la herramienta a nivel de usuario para que sea éste el que pueda escoger el kernel que desee estudiar y poder generar las condiciones de rotacional y divergencia nulos, con objeto de que pueda obtener mejores estimaciones de los fenómenos a estudiar.

Además, con la herramienta desarrollada durante el presente trabajo se puede generar un banco de datos de campos vectoriales con las características de divergencia nula y/o rotacional nulo, de tal manera que se pueden instanciar las condiciones iniciales de algunos fenómenos físicos para realizar,por ejemplo, estimaciones energéticas. Por lo que esto también podría ser una línea de trabajo futuro.

Por otro lado, para el cálculo de la divergencia y rotacional de los campos vectoriales se ha utilizado el programa VisIT, que tiene un algoritmo basado en diferencias finitas de segundo orden; por tanto, es importante buscar la manera de implementar estos cálculos dentro de la herramienta desarrollada en el presente trabajo, de tal manera que se obtengan los valores sin necesidad de una herramienta adicional.

Referencias

Alvarez, M. (2011). Convolved Gaussian process priors for multivariate regression with applications to dynamical systems. *Science*, (November). Retrieved from ftp://ftp.dcs.shef.ac.uk/home/neil/thesisAlvarez.pdf

Alvarez, M., Rosasco, L., & Lawrence, N. (2012). *Kernels for Vector-Valued Functions : a Review*. 1–37.

Banda-Barragán, W. E., Federrath, C., Crocker, R. M., & Bicknell, G. V. (2018). Filament formation in wind-cloud interactions- II. Clouds with turbulent density, velocity, and magnetic fields. *Monthly Notices of the Royal Astronomical Society*, *473*(3), 3454–3489. https://doi.org/10.1093/mnras/stx2541

Banda-Barragán, W. E., Zertuche, F. J., Federrath, C., Del Valle, J. G., Brüggen, M., & Wagner, A. Y. (2019). On the dynamics and survival of fractal clouds in galactic winds. *Monthly Notices of the Royal Astronomical Society*, *486*(4), 4526–4544. https://doi.org/10.1093/mnras/stz1040

Chen, F. F. (1984). *Introduction to plasma physics and controlled fusion* (Vol. 1). Springer.

Duvenaud, D. K. (2014). Automatic Model Construction with Gaussian Processes. Retrieved from https://www.cs.toronto.edu/~duvenaud/thesis.pdf

E., G. S., Shaaban, K. A., El-Naggar, M. E. E., & Shaaban, M. (2011). La Microespectroscopía de infrarojo con transformada de Fourier (FTIRM) en el estuido de sistemas biológicos. *Revista Latinoamericana de Química*, *39*(1–2), 62–74. Retrieved from http://www.scielo.org.mx/scielo.php?pid=S0370-59432011000100006&script=sci_arttext

Fairbrother, J., Nemeth, C., Rischard, M., & Brea, J. (2018). *GaussianProcesses.jl: A Nonparametric Bayes package for the Julia Language*. Retrieved from http://arxiv.org/abs/1812.09064

Federrath, C., Chabrier, G., Schober, J., Banerjee, R., Klessen, R. S., & Schleicher, D. R. G. (2011). Mach Number Dependence of Turbulent Magnetic Field Amplification:

Solenoidal versus Compressive Flows. *Phys. Rev. Lett.*, *107*(11), 114504. https://doi.org/10.1103/PhysRevLett.107.114504

Federrath, C., Roman-Duval, J., Klessen, R., Schmidt, W., & Mac Low, M.-M. (2009). *Comparing the statistics of interstellar turbulence in simulations and observations: Solenoidal versus compressive turbulence forcing*. https://doi.org/10.1051/0004-6361/200912437

Gibbon, P. (2017). *Introduction to Plasma Physics*. https://doi.org/10.5170/CERN-2016-001.51

Jidling, C., Wahlström, N., Wills, A., & Schön, T. B. (2017). Linearly constrained Gaussian processes. *Advances in Neural Information Processing Systems (NIPS)*. Retrieved from http://arxiv.org/abs/1703.00787

Krause, A., Guestrin, C., Gupta, A., & Kleinberg, J. (2008). *Near-optimal sensor placements: maximizing information while minimizing communication cost*. 2–10. https://doi.org/10.1109/ipsn.2006.244031

Lewis, G. M., & Austin, P. H. (1996). Jp4.16. *Compute*.

Macêdo, I., & Castro, R. (2008). Learning divergence-free and curl-free vector fields with matrix-valued kernels. *IMPA Preprint*. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Learning+divergence-free+and+curl-free+vector+fields+with+matrix-valued+kernels#0

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval Introduction. In *Computational Linguistics* (Vol. 35). https://doi.org/10.1162/coli.2009.35.2.307

Micchelli, C. A., Xu, Y., & Zhang, H. (2006). Universal Kernels. *Journal of Machine Learning Research*, *7*, 2651–2667.

Mignone, A., Bodo, G., Massaglia, S., Matsakos, T., Tesileanu, O., Zanni, C., & Ferrari, A. (2007). PLUTO: A Numerical Code for Computational Astrophysics. *The Astrophysical Journal Supplement Series*, *170*(1), 228–242. https://doi.org/10.1086/513316

Murphy, K. (2012). Machine Learning. In *SpringerReference*.

https://doi.org/10.1007/SpringerReference_35834

Rasmussen, C. E., & Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. In *The Journal of Machine Learning Research* (Vol. 11). https://doi.org/10.1142/S0129065704001899

Rincón, L. (2012). *Introducción a los procesos estocásticos Circuito Exterior de CU*.

Sarmiento, A., Fondón, I., Velasco, M., Qaisar, A., & Aguilera, P. (2014). *Modelo de Mezcla de Gaussianas Generalizadas para Segmentación de Melanomas*. (November).

Skiba, Y. (2008). Introducción a la dinámica de fluidos. *UNAM*, (September), 629.

Soler, J., & Gamazo, P. (2015). *Solución del problema de riemann para las ecuaciones de flujo poco compresible*. (July). https://doi.org/10.13140/RG.2.1.4397.8088

Wahlström, N., Kok, M., Schön, T. B., & Gustafsson, F. (2013). Modeling magnetic fields using Gaussian processes. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 3522–3526. https://doi.org/10.1109/ICASSP.2013.6638313

Wan, Y. Q., Fan, L. M., & Qi, Y. (2007). Nonlinear characteristics analysis for steel wire isolators. *Zhendong Yu Chongji/Journal of Vibration and Shock*, *26*(7), 46–49. https://doi.org/10.1111/1368-423X.00003

Anexos

Anexo A.

A.1 *Kernel* Divergencia Nula

```
def divFreeKern(x, y, l):
'''
input:
- x e y dos vectores fila en R^n.
output:
- k(x, y) div free kernel matriz R^nxn.
'''
# definir los hiperparámetros ℓ y sigma (alpha en este código)

alpha=0.3
l=l

##operaciones previas
n = len(x)
d = x - y
normd = np.sum(d**2)

##evaluar primer término del kernel final

T1 = (alpha**2)*np.e**(-normd/(2*(l**2)))

##evaluar primer sumando del Segundo término del kernel final

In = np.eye(n, n)
v = np.reshape(d, (n, 1))
vt = np.transpose(v)
ddT = v.dot(vt)

## evaluar Segundo sumando del Segundo término del kernel final

nod = np.repeat(1, n)
nod[1:n] = -1
NOD = toeplitz(nod)

## calcular el Segundo término del kernel final

T2 = (1/l**2)*ddT+(n-1-normd/(l**2))*(In)

##obtener matriz final
```

```
return(T1*T2)
```

A.2 *Kernel* Rotacional Nulo

```
def curlFreeKern(x, y, l):
'''
input:
- x e y dos vectores fila en R^n.
output:
- k(x, y) curl free kernel matriz R^nxn.
'''
## definir hiperparámetros y dimensión del espacio de salida

alpha=0.3
l=l
n = len(x)

##ordenar los vectores uno debajo del otro

xrf=np.matlib.repmat(x,n,1)
xrc=xrf.T
dxx1=xrf-xrc
dxx=dxx1**2

yrf=np.matlib.repmat(y,n,1)
yrc=yrf.T
dyy1=yrf-yrc
dyy=dyy1**2

z=np.zeros(n)
zrf=np.matlib.repmat(z,n,1)
zrc=zrf.T
dzz1=zrf-zrc
dzz=dzz1**2

dxy=dxx1*dyy1
dxz=dxx1*dzz1
dyz=dyy1*dzz1

##calcular el primer térmmino del kernel final

t1=(alpha**2)*np.e**(-1/2*(dxx+dyy+dzz)/l**2)
t1=np.kron([[1,1,1],[1,1,1],[1,1,1]],t1)

##calcular los términos internos del kernel final

t11=np.kron([[1,0,0],[0,0,0],[0,0,0]], 1/l**2-dxx/l**4)
t22=np.kron([[0,0,0],[0,1,0],[0,0,0]], 1/l**2-dyy/l**4)
```

```
t33=np.kron([[0,0,0],[0,0,0],[0,0,1]], 1/l**2-dzz/l**4)

t12=np.kron([[0,1,0],[1,0,0],[0,0,0]], -dxy/l**4)
t13=np.kron([[0,0,1],[0,0,0],[1,0,0]], -dxz/l**4)

t23=np.kron([[0,0,0],[0,0,1],[0,1,0]], -dyz/l**4)
t2=t11+t22+t33+t12+t13+t23

##calcular el kernel final

K=t1*t2
return(K)
```