PYTHON EN EL DESARROLLO DE LA COMPUTACIÓN CIENTÍFICA – CASO DE APLICACIÓN: CÁLCULO DEL COMPONENTE TOROIDAL DEL FLUJO DE UN FLUIDO INCOMPRESIBLE EN COORDENADAS ESFÉRICAS

PYTHON INTO THE SCIENTIFIC COMPUTING DEVELOPMENT – IMPLEMENTING CASE: CALCULATION OF THE TOROIDAL COMPONENT OF A FLOW IN SPHERICAL COORDINATES

Diego Andrés Peña Arcos, Ph.D.
Doctor en Geofísica, Observaciones Nacional ON/MCTIC (Brasil).
Docente de la Facultad de Ingenierías de la Universidad ECOTEC, Ecuador.
dpena@ecotec.edu.ec

ARTÍCULO DE REFLEXIÓN
Recibido: 11 de abril de 2019.
Aceptado: 3 de mayo de 2019.

RESUMEN

La computación científica se ha convertido en una de las principales herramientas para el desarrollo de la ciencia actual, convirtiéndose en una necesidad su aprendizaje y utilización. A pesar de esto, no todos los científicos son expertos programadores y sus investigaciones puede verse afectadas por esta limitante computacional. En este escenario, Python aparece como un lenguaje de programación de alto rendimiento, versátil, “open source” y de fácil aprendizaje y uso. El objetivo de este estudio es mostrar el uso de Python para el desarrollo de un código que permita el cálculo del componente toroidal del flujo de un fluido incompresible en coordenadas esféricas. Para esto se usaron librerías de Python especializadas en la solución de sistemas lineares en 2D, logrando un código simple y elegante que consigue hacer los cálculos necesarios con un nivel de error casi insignificante en un tiempo de procesamiento muy corto. Por su simplicidad, esta codificación puede ser bastante útil para procesos complejos como dinamos planetarios o flujos de aire en la atmósfera.

Palabras clave: computación científica, programación, Python, descomposición Poloidal-Toroidal, dinámica de fluidos.

ABSTRACT

Scientific Computing has become one of the most important tools for the development of the present-day science. Learning it and Using it is now a necessity for the scientists. Nevertheless, not all the scientists are specialized programmers and their works may be affected because this limitation. In this scenario, Python appears as a high-performance programming language, versatile, open source, easy to learn and use. The main objective of this study is to show the capability of Python into the development of a code that allows to calculate the toroidal component of the flow of an incompressible fluid in spherical coordinates. Several Python packages specialized in the solution of 2D linear systems were used to achieve a simple but elegant code able to do the necessary calculations with an insignificant numeric error level in a short processing time. By its simplicity, this way to code could be very useful in the analysis of complex process like planetary dynamos or atmospheric air flows.

Keywords: scientific computing, Python, Poloidal-Toroidal decomposition, fluid dynamics, programming

INTRODUCCIÓN

Tradicionalmente se considera que el proceso de desarrollo científico se divide en dos aproximaciones, una teórica y una experimental, que en conjunto establece las bases del conocimiento permitiendo consolidar los nuevos descubrimientos y trazando los caminos que la ciencia debe seguir en el futuro. Durante las últimas décadas, los avances en la tecnología y la investigación han venido ofreciendo grandes cantidades de datos, de forma tal que fue necesario el uso de herramientas computacionales para su procesamiento, haciendo de la computación una parte fundamental de la ciencia. Actualmente, el trabajo computacional es un complemento importante tanto para los experimentos como para la teoría y una considerable cantidad de artículos involucran cálculos numéricos, simulaciones y modelos computacionales (Weiss 2017).

Ya en los años 80, la computación científica empieza a popularizarse en áreas como las ciencias exactas y humanas, en el arte con problemas cada vez más complejos que van desde la comprensión del planeta en el que vivimos, el comportamiento de la población, simulaciones de mercados, codificación del genoma, teoría del color y del sonido, etc.

Esta variedad de temas y formas de abordar las investigaciones mostraron que la computación científica no puede más ser solo una computación numérica, es necesario que esta sea versátil: debe poder lidiar con grandes conjuntos de datos, ofrecer estructuras de datos mucho más ricas que solo números, interactuar con bases de datos y aplicaciones web, manejar datos en varios formatos, permitir trabajo colaborativo en grupos y que tenga una documentación sencilla (Perez et al., 2011).

A pesar de todas estas ventajas, una de las mayores limitantes en el uso de la computación científica por parte de los científicos es que ellos no tienen el tiempo suficiente para aprender complejos lenguajes de programación o intrincadas herramientas computacionales para el desarrollo de su trabajo. Fortran y C++ fueron los lenguajes por excelencia por muchos años debido a la rapidez en el procesamiento de datos, pero con el paso de los años, el tiempo que dedica a la programación se volvió cada vez más valioso que el tiempo de cómputo haciendo necesario desarrollar nuevos lenguajes que fueran más fáciles de aprender y usar. En este contexto aparecen herramientas como MATLAB, IDL, Mathematica y Maple, que permiten interactuar-explorar-visualizar los procesos en un marco de programación-compilación-ejecución (Perez et al., 2011).

Por otro lado, una de las características fundamentales de la ciencia, principalmente en la ciencia experimental, es que tanto los métodos usados como los resultados obtenidos estén sujetos a ser replicados y reproducidos. Es esta la garantía de que los resultados obtenidos son relevantes para su campo de estudio y que estudios derivados de estos pueden ser desarrollados por otros equipos en diferentes partes del mundo. La ciencia computacional no está fuera de esta máxima, y prestigiosas revistas, ya exigen a los autores proveer acceso a los códigos fuente de sus investigaciones (Peng, 2011).

En este contexto aparece Python, un lenguaje de programación de alto nivel que ofrece un balance entre simplicidad y flexibilidad sin sacrificar rendimiento (Oliphant, 2007). Python se considera un lenguaje de programación *interprete* (capaz de analizar y ejecuta otros programas), que permite al usuario elegir entre programación orientada a objetos, programación imperativa y programación funcional y *multiplataformas*. Python puede considerarse un “ecosistema computacional” que puede suplir todas las necesidades de la ciencia moderna. Este lenguaje es fácil de aprender y usar por mantener una sintaxis intuitiva que permite al investigador expresar, explorar y visualizar sus ideas a través de diferentes interfaces que se van ajustando a su trabajo.

Python también ha sido ampliamente utilizado en ciencias de la Tierra (Sáenz et al., 2002) a través del desarrollo de librerías, códigos, aplicaciones en áreas como sismología (Wassermann et al., 2013), métodos electromagnéticos (Werthmüller, D. 2017), magnetotelúrica (Krieger & Peacock, 2014), gravimetría (Hector & Hinderer, 2016) procesamiento y organización de datos geofísicos (Uieda, 2018) y modelado e inversión (Uieda et al., 2014; Rücker et al., 2017) de estos datos, entre otras. En el área de magnetohidrodinámica existen librerías como Spacepy-A (Morley at al., 2010) que permiten la solución de las ecuaciones de Maxwell para diferentes escenarios dinámicos espaciales.

Teniendo en cuenta todas las facilidades que Python ofrece, el presente estudio explorará las capacidades que este lenguaje de programación ofrece para el desarrollo de códigos relacionados con la descomposición poloidal-toroidal de un campo vectorial, herramienta muy útil en magnetohidrodinámica y mecánica de fluidos para analizar fenómenos como campos magnéticos o flujos de fluidos incompresibles en geometrías esféricas. Este artículo se presenta de la siguiente forma: en la sección 2 se hará una revisión teórica donde se expondrán algunos de los conceptos y significado físico del fenómeno a estudiar, así como las herramientas de Python que serán usadas. En la sección 3 se describirá la metodología a usarse, las características de los datos y modelos y el procedimiento a seguir. Finalmente, en las secciones 4 y 5 se mostrarán los resultados y las conclusiones a las que se llega a partir de ellos, respectivamente.

REVISIÓN TEÓRICA

Descomposición Poloidal-Toroidal

La descomposición poloidal-toroidal es una forma restringida de la descomposición de Helmholtz generalmente usada en el análisis en coordenadas esféricas de campos vectoriales solenoidales, i.e. campos magnéticos y fluidos incompresibles (Chandrasekhar, 1961). En mecánica de fluidos y en magnetohidrodinámica es muy útil descomponer de esta manera un campo vectorial ya que el componente poloidal contiene toda la parte radial del campo, mientras que en el componente toroidal se encuentra toda su corriente radial (Gubbins & Roberts, 1987).

En el caso de flujo de metal liquido en la superficie del núcleo de la Tierra, este puede considerarse como el flujo de un fluido incompresible $\vec{u}$ en la superficie de un cascarón esférico de radio *R* (radio del núcleo de la Tierra), entonces el campo vectorial de su

velocidad tangencial (flujo en 2 dimensiones sobre la superficie de la esfera) $\vec{u}_h$ puede expresarse en términos de los potenciales Ψ y Φ que representan sus componentes toroidal y poloidal, respectivamente:

$$\vec{u}_h = \nabla \times \Psi\hat{r} + \nabla_h\Phi \quad (1)$$

donde $\hat{r}$ es el vector unitario en la dirección radial y $\nabla_h = \nabla - \frac{\partial}{\partial r}$.

El flujo toroidal es no-divergente, de forma que su potencial se representa por una función de corriente (streamfunction), mientras que el flujo poloidal se origina en procesos de entrada y salida de fluido (Ver Figura 1).

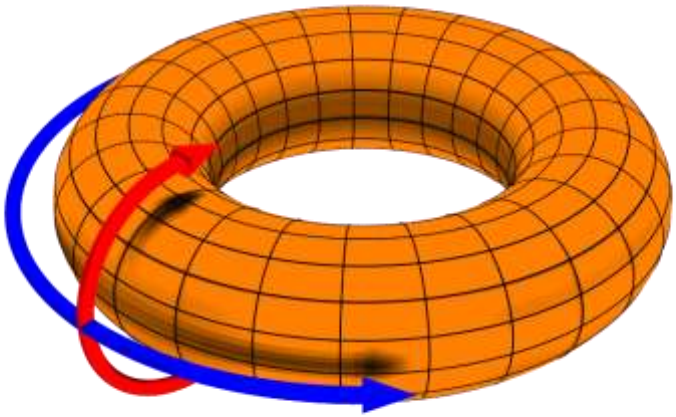

Figura 1: Diagrama que describe los componentes poloidal (flecha roja) y toroidal (flecha azul) de un flujo. El componente poloidal se desplaza en la dirección latitudinal entrando y saliendo de la superficie, mientras que el toroidal se desplaza en la dirección longitudinal siempre constante (fuente: Wikipedia).

Vorticidad

La vorticidad es una magnitud física que cuantifica la rotación de un fluido. Matemáticamente, la vorticidad puede definirse como el rotacional de un campo de velocidades:

$$\omega = \nabla \times \vec{u} \tag{2}$$

La vorticidad de un flujo bidimensional es siempre perpendicular al plano del flujo y puede considerarse como una cantidad escalar. Adicionalmente, la función de corriente puede ser expresada en términos de la vorticidad usando la siguiente ecuación de Poisson:

$$\nabla_h^2 \Psi = \omega_r \tag{3}$$

donde ∇_h^2 es el componente horizontal del operador laplaciano y $\omega_r = \nabla_h \times \vec{u}_h$ denominada *vorticidad radial*. En coordenadas esféricas la ecuación (3) sería:

$$\frac{1}{r^2 \sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta \frac{\partial \Psi}{\partial\theta}\right) + \frac{1}{r^2 \sin^2\theta}\frac{\partial^2 \Psi}{\partial\phi^2} = \omega_r \tag{4}$$

2.1.2. Solución a la ecuación de Poisson

La ecuación de Poisson es una ecuación diferencial parcial elíptica que aparece como una generalización de la *ecuación de Laplace* con múltiples usos en problemas de ingeniería, física teórica, ciencias de la Tierra y espaciales.

No es posible obtener una solución exacta de la ecuación (4), por eso es necesario usar un método numérico que permita el cálculo de la mejor aproximación a su solución. Para este caso se utiliza el método de diferencias finitas FDM (Morton & Myers, 2005), que ofrece expresiones para el cálculo numérico de las derivadas espaciales de la función de corriente. Considerando que la derivada numérica de una función puede ser calculada por:

$$f'(x) \approx f\left(x + \frac{1}{2}h\right) - f\left(x - \frac{1}{2}h\right)$$

y la segunda derivada

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

donde *h* es el espaciado entre los puntos de la función. Usando el FDM, la ecuación (4) puede ser escrita como:

$$\frac{\Psi(i+1,j)-2\Psi(i,j)+\Psi(i-1,j)}{\Delta^2}+\frac{1}{\tan\theta(i,j)}\frac{\Psi(i+1,j)-\Psi(i-1,j)}{2\Delta}+$$
$$\frac{1}{\sin^2\theta(i,j)}\frac{\Psi(i,j+1)-2\Psi(i,j)+\Psi(i,j-1)}{\Delta^2}=R^2\omega_r(i,j)$$

La descripción matemática de esta sección puede ser revisada en Peña (2017, p 85), llegando hasta el sistema lineal que puede ser expresado de forma matricial como:

$$\bar{\bar{A}}\bar{\psi}=\bar{\omega} \tag{5}$$

donde $\bar{\bar{A}}$, también llamada matriz sensibilidad, es una matriz cuadrada que almacena las características propias del sistema; $\bar{\psi},\bar{\omega}$ son vectores que representan la función de corriente Ψ y la vorticidad radial $R^2\omega_r$ en todos los puntos de la superficie esférica. Finalmente, la solución del sistema linear se calcula de la siguiente forma:

$$\bar{\psi}=\bar{\bar{A}}^{-1}\bar{\omega} \tag{6}$$

Librería CVOXPT

Python ofrece una variedad casi ilimitada de librerías para la computación científica que facilita los procesos de una forma sencilla y elegante. Una de estas librerías es la CVOXPT, que se define como una librería software libre de optimización convexa (búsqueda de la mejor solución por medio de funciones convexas) basada en Python que funciona como una excelente herramienta de modelado para problemas de optimización de grandes cantidades de datos con geometrías definidas.

Su característica más destacable es la posibilidad de organizar los grupos de datos en dos objetos matriciales: matrices densas y matrices dispersas. Estos objetos ocupan una menor cantidad de memoria comparados con otros en Python, permitiendo realizar cálculos más complejos con un menor gasto computacional.

La librería CVXOPT se organiza en los siguientes módulos:

- **cvxopt.blas:** Interfaz para cálculo de doble-precisión real y complejo basado en BLAS (The BLAS Interface).

- **cvxopt.lapack:** Interfaz para cálculo de doble-precisión real y complejo de ecuaciones lineales y rutinas de eigenvalores basados en LAPACK (The LAPACK Interface).
- **cvxopt.fftw:** Una interfaz opcional para rutinas de transformada discreta basadas en FFTW (Discrete Transforms).
- **cvxopt.amd:** Interfaz de rutinas de aproximación de mínimo grado basado en AMD (Matrix Orderings).
- **cvxopt.umfpack:** Interfaz para soluciones de Sistemas Lineales dispersos (General Linear Equations).
- **cvxopt.cholmod:** Interfaz para soluciones de sistemas dispersos Cholesky basado en CHOLMOD (Positive Definite Linear Equations).
- **cvxopt.solvers:** Rutinas de optimización convexa e interfaces opcionales para soluciones basadas en GLPK, MOSEK y DSDP5 (Cone Programming and Nonlinear Convex Optimization).
- **cvxopt.modeling:** Rutinas para especificar y resolver programas lineales y problemas de optimización convexa con diagramas lineales de costo y funciones de frontera (Modeling).

MATERIALES Y MÉTODOS

Con la finalidad de mostrar la eficiencia del uso de Python como lenguaje de alto nivel para la solución numérica de la ecuación de Poisson definida en la ecuación (4) y la posterior obtención del componente toroidal del flujo de un fluido incompresible en una superficie esférica, el presente estudio siguió la siguiente metodología: elegir la mejor interfaz para las características del problema, desarrollar el código que obtenga la solución numérica y finalmente, medir la eficiencia de la solución en modelos sintéticos.

Módulo umfpack

El módulo escogido para la solución numérica de la ecuación (6) fue el módulo **umfpack** que permite resolver sistemas lineales dispersos de la forma $X = A^{-1}B$ a través de la función *linsolve*. La matriz sensibilidad A puede ser modificada para minimizar el efecto de los bordes latitudinales y longitudinales propio de problemas con geometría esférica. Para los bordes latitudinales se calculó el valor de lo que vendría ser los polos usando

el promedio de los puntos vecinos. Para los bordes longitudinales se usó una aproximación cíclica.

Modelos sintéticos

Una forma bastante útil de medir la eficiencia de un código es usando modelos sintéticos de función de corriente, de forma que sea posible obtener una expresión exacta y su aproximación y hacer una comparación punto a punto, determinando así, el error vinculado.

Para este estudio fueron usadas 2 configuraciones de modelos sintéticos:

- **Psi_1:** $\Psi = \sin\theta \cos\phi$
- **Psi_2:** $\Psi = \sin\theta \cos\theta \cos\phi + \sin^2\theta \cos\theta \cos 2\phi$

donde θ y ϕ representan co-latitud y longitud, respectivamente. Los puntos en la superficie de la esfera se toman en un arreglo de 1°x1° que va de $0° \leq \theta \leq 180°$ y $0° \leq \phi \leq 360°$.

Estos modelos representan características dinámicas diferentes que permiten hacer una evaluación profunda de las soluciones. Los mapas correspondientes a estos modelos pueden apreciarse en la parte izquierda de la Figura 2.

Procedimiento

Inicialmente se calcula la función de corriente Ψ_f de la siguiente forma:

1. A partir del modelo sintético se calcula la vorticidad radial usando la ecuación (3).
2. La vorticidad radial se convierte en el input del código para obtener una solución numérica a la ecuación (4) Ψ_f.
3. Se compara el Ψ_f con el Ψ original y se calcula el error de aproximación.

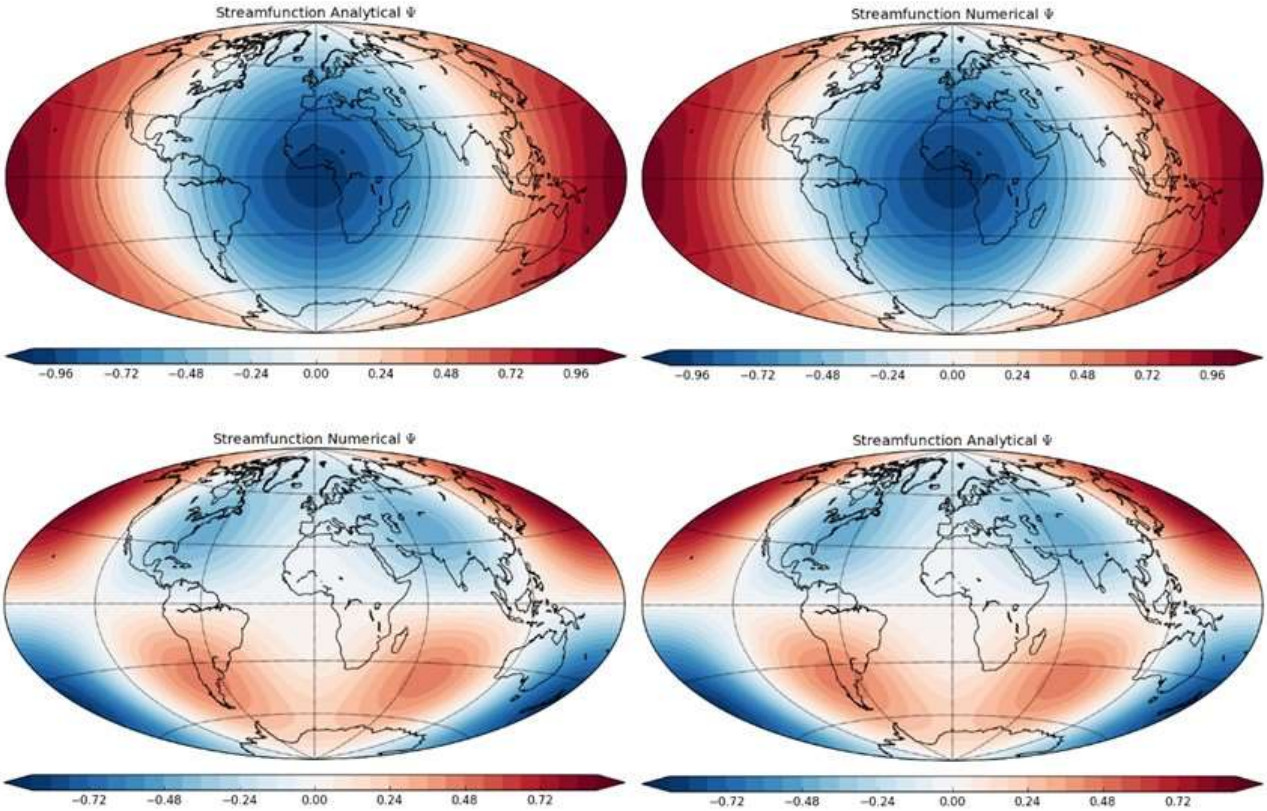

Figura 2: Mapas globales de las funciones de corriente sintética Psi_1 (arriba) y Psi_2 (abajo). A la izquierda se encuentra el modelo original y a la derecha el calculado numéricamente usando el código desarrollado en este estudio. Los mapas fueron realizados para este estudio usando la librería matplotlib.basemap. Los continentes son graficados como referencia.

RESULTADOS

Para obtener el componente toroidal del flujo de un fluido incompresible es necesario resolver numéricamente el problema descrito por la ecuación de Poisson definida en la ecuación (4) en coordenadas esféricas. Una vez definidas las características del problema se procede a diseñar el código usando las librerías de Python previamente descritas en este estudio para resolver el sistema lineal (6) a través de una optimización convexa. El código se muestra en la Figura 3 y puede ser encontrado en el repositorio de Github en https://github.com/diegupena/cajitafeliz/blob/master/Streamfunction.ipynb .

In [1]:
```
import numpy as np
from numpy import *
from cvxopt import matrix, spmatrix, umfpack, mul, sqrt, div
from scipy.stats.stats import pearsonr
```

In [2]:
```
R = 3485e+0                                 # Earth's core radius
# FMD step
h = deg2rad(1)                              # in radians
# Coordinates
Theta,Phi = linspace(h,pi-h,179),linspace(0,2*pi,361)
N,M       = len(Theta),len(Phi)
Ph,Th     = np.meshgrid(Phi, Theta)         # Ph,Th Lon-Lat Grid [N,M]
# Sin,Cos,Tan
sth,cth,tth = sin(Th),cos(Th),tan(Th)
```

In [3]:
```
#--------------------#
# Sensivity Matrix A #
#--------------------#
# Equation factors
a = (-2/h**2)*(1+(sth)**-2)
b = ( 1/h**2)+(2*h*tth)**-1
c = ( 1/h**2)-(2*h*tth)**-1
d = ( 1/h**2)*(sth)**-2
P = ( 2/h**2)
#b[N-1],c[0] = 0,0
b[0],b[N-1] = P,0
c[0],c[N-1] = 0,P
av = matrix(a,(1,N*M))
bv = matrix(b,(1,N*M))
cv = matrix(c,(1,N*M))
dv = matrix(d,(1,N*M))
# A Matrix
MD = spmatrix(av,            range(N*M),       range(N*M)                    )
B  = spmatrix(bv[0:N*M-1],   range(N*M-1),     range(1,N*M),       (N*M,N*M))
C  = spmatrix(cv[1:N*M],     range(1,N*M),     range(N*M-1),       (N*M,N*M))
DU = spmatrix(dv[0:N*M-(N)], range(N*M-N),     range(N,N*M),       (N*M,N*M))
EU = spmatrix(dv[0:N],       range(N),         range(N*M-2*N,N*M-N),(N*M,N*M))
ED = spmatrix(dv[N:N*M],     range(N,N*M),     range(N*M-N),       (N*M,N*M))
DD = spmatrix(dv[N*M-N:N*M], range(N*M-N,N*M), range(N,2*N),       (N*M,N*M))
A = MD+B+C+DU+DD+EU+ED
```

In [4]:
```
#-----------#
# Inversion #
#-----------#
Wr = (2/R**2)*cth                           # Radial vorticity
V = matrix(((-R**2)*Wr),(N*M,1))

# Linear System Solver
umfpack.linsolve(A,V)                       # Psi = A^-1*V
Y = matrix(V,(N,M))                         # numerical solution for Psi
#print Y
```

Figura 3: Código para la obtención del componente toroidal del flujo de un fluido incompresible en coordenadas esféricas.

El código define una malla de 181x361 puntos sobre una superficie esférica. Siguiendo el procedimiento descrito en la sección 3.3 se calculan las aproximaciones de las funciones de corriente sintéticas. El código desarrollado resuelve un sistema lineal de 64.800 ecuaciones por el mismo número de incógnitas en un tiempo de procesamiento de menos de 0.1 s y con un error numérico de aproximación encontrado entre la Ψ original y la Ψ_f menor 0.008%. El código consigue reproducir tanto la amplitud como las características estructurales de los modelos sintéticos sin importar el tipo de función que se use (compare en Figura 2: Psi_1 y Psi_2). Los mapas de los modelos sintéticos se muestran en la parte derecha de la Figura 2.

CONCLUSIONES

El papel que juega la computación científica en el desarrollo de la ciencia ha sido más que probado en las últimas décadas, llegando a ser parte importante de estudios de las más diversas naturalezas. Dentro de este contexto Python, como lenguaje de programación de última generación, ha mostrado ser una herramienta útil y versátil para el diseño, ejecución, evaluación de proyectos científicos debido a su característica esencial: Intuitivo sin perder su alto rendimiento. Python es sencillo de aprender y usar ofreciendo una sintaxis natural que permite a los investigadores, que no son programadores especializados, expresar y explorar sus ideas de forma más directa que con otros lenguajes tradicionales (Lin, 2012).

En este caso de estudio fue usado Python para desarrollar un código que permita el cálculo del componente toroidal de un campo vectorial. En áreas como mecánica de fluidos y magnetohidrodinámica, la descomposición poloidal-toroidal de un campo resulta una herramienta bastante eficaz para el análisis de sistemas complejos. El código desarrollado consiguió calcular numéricamente la solución al sistema lineal definido por una ecuación de Poisson a través de un proceso de optimización convexa con un nivel de error numérico de 0.008% y en un tiempo de procesamiento de casi 0.1s. Teniendo en cuenta el tamaño del problema (un sistema de 64.800 ecuaciones e incógnitas) se puede concluir que este código ofrece una solución excelente y rápida, probando la eficiencia de la programación en Python en la resolución de este tipo de problemas.

Soluciones a este tipo de procesos (descomposición poloidal-toroidal) mediante una programación sencilla e intuitiva como la que Python ofrece, pueden convertirse en catapultas para estudios más complejos en Física. Actualmente, este tipo de descomposición se usa para modelar campos magnéticos planetarios (Peña et al., 2016; 2018) y estelares (Lehmann et al., 2017), mecanismo de dinamos planetarios, flujos de plasma en estrellas (Hole & Appel, 2007), flujos de aire en la atmosfera terrestre (Shikuo & Shida, 1997), entre otras aplicaciones.

REFERENCIAS BIBLIOGRÁFICAS

Bladel, J. (1958). On Helmholtz's Theorem in Finite Regions. Midwestern Universities Research Association

Chandrasekhar, S. (1961). Hydrodynamic and hydromagnetic stability. *International Series of Monographs on Physics*. Oxford: Clarendon. See discussion on page 622.

Gubbins, D., Roberts, P. H., (1987), “Magnetohydrodynamics of the Earth’s core”. In: A, J. (Ed.), Geomagnetism, v. 2, Academic Press, pp. 1–183, London.

Hector, B., & Hinderer, J. (2016). pyGrav, a Python-based program for handling and processing relative gravity data. *Computers & geosciences*, *91*, 90-97.

Hole, M. J., & Appel, L. C. (2007). Fourier decomposition of magnetic perturbations in toroidal plasmas using singular value decomposition. *Plasma Physics and Controlled Fusion*, *49*(12), 1971.

Krieger, L., & Peacock, J. R. (2014). MTpy: A Python toolbox for magnetotellurics. *Computers & geosciences*, *72*, 167-175.

Lehmann, L. T., Jardine, M. M., Vidotto, A. A., Mackay, D. H., See, V., Donati, J. F., ... & Petit, P. (2016). The energy budget of stellar magnetic fields: comparing non-potential simulations and observations. *Monthly Notices of the Royal Astronomical Society: Letters*, *466*(1), L24-L28.

Lin, J. W. (2012). "Why Python Is the Next Wave in Earth Sciences Computing." Bulletin of the American Meteorological Society, 93(12), pp. 1823–1824.

Morley, S. K., Welling, D. T., Koller, J., Larsen, B. A., & Henderson, M. G. (2010). *Spacepy-A Python-based library of tools for the space sciences* (No. LA-UR-10-04308; LA-UR-10-4308). Los Alamos National Lab. (LANL), Los Alamos, NM (United States).

K.W. Morton y D.F. Mayers, *Numerical Solution of Partial Differential Equations, An Introduction*. Cambridge University Press, 2005.

Oliphant, T. E. (2007). "Python for Scientific Computing," in *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10-20, doi: 10.1109/MCSE.2007.58

Peng, R. D. (2011). Reproducible Research in Computational Science. *Science* 334, Issue 6060, pp. 1226-1227 DOI: 10.1126/science.1213847

Peña, D., Amit, H., & Pinheiro, K. J. (2016). Magnetic field stretching at the top of the shell of numerical dynamos. *Earth, Planets and Space*, *68*(1), 78.

Peña, D. A. (2017). Magnetic field stretching at the top of the Earth's core. (Tesis de Doctorado). Observatório Nacional ON/MCTIC, Rio de Janeiro, Brasil

Peña, D., Amit, H., & Pinheiro, K. J. (2018). Deep magnetic field stretching in numerical dynamos. *Progress in Earth and Planetary Science*, *5*(1), 8.

Perez, F., B. E. Granger, and J. D. Hunter. (2011). "Python: An Ecosystem for Scientific Computing." *Computing in Science & Engineering*, vol. 13, no. 2, pp. 13-21.

Rücker, C., Günther, T., & Wagner, F. M. (2017). pyGIMLi: An open-source library for modelling and inversion in geophysics. *Computers & Geosciences*, *109*, 106-123.

Sáenz, J., Zubillaga, J., & Fernández, J. (2002). Geophysical data analysis using Python. *Computers & geosciences*, *28*(4), 457-465.

Shikuo, L., & Shida, L. (1997). Toroidal-Poloidal Decomposition and Beltrami Flows in Atmosphere Motions [J]. *SCIENTIA ATMOSPHERICA SINICA*, *2*.

Uieda, L., V. C. Oliveira Jr., A. Ferreira, H. B. Santos, and J. F. Caparica Jr., 2014, Fatiando a Terra: A Python package for modeling and inversion in geophysics: figshare, http://dx.doi. org/10.6084/m9.figshare.1115194.

Uieda, L. (2018). Verde: Processing and gridding spatial data using Green's functions. Journal of Open Source Software, 3(29), 957. doi:10.21105/joss.00957

Wassermann, J. M., Krischer, L., Megies, T., Barsch, R., & Beyreuther, M. (2013, December). Obspy: A python toolbox for seismology. In *AGU Fall Meeting Abstracts*.

Weiss, C. J. (2017) Scientific Computing for Chemists: An Undergraduate Course in Simulations, Data Processing, and Visualization. *Journal of Chemical Education. 94* (5), 592-597. DOI: 10.1021/acs.jchemed.7b00078

Werthmüller, D. (2017). An open-source full 3D electromagnetic modeler for 1D VTI media in Python: empymod. *Geophysics*, *82*(6), WB9-WB19.