Indonesian Physical Review
Volume 07 Issue 03, September 2024
P-ISSN: 2615-1278, E-ISSN: 2614-7904

Comparing scientific computing environments for simulating 2D non-buoyant fluid parcel trajectory under inertial oscillation: A preliminary educational study

Sandy H. S. Herho [1*], Iwan P. Anwar [2], Katarina E. P. Herho [3], Candrasa S. Dharma[4], Dasapta E. Irawan[5]

[1]*Department of Earth and Planetary Sciences, University of California, CA, USA.*
[2]*Oceanography Research Group, Bandung Institute of Technology, West Java, Indonesia.*
[3]*Department of Geological Engineering, Trisakti University, West Jakarta, DKI Jakarta, Indonesia.*
[4]*Naval Hydrographic and Oceanographic Center (Pushidrosal), Indonesian Navy (TNI AL), DKI Jakarta, Indonesia.*
[5]*Applied Geology Research Group, Bandung Institute of Technology, West Java, Indonesia.*

Corresponding Authors E-mail: sandy.herho@email.ucr.edu

Article Info

Article info:
Received: 30-04-2024
Revised: 18-07-2024
Accepted: 13-08-2024

Keywords:
Fluid parcel trajectories; Geophysical fluid dynamics; Inertial oscillations; Idealized models; Open-source programming languages

How To Cite:
S. H. S. Herho, I. P. Anwar, K. E. P. Herho, C. S. Dharma, D. E. Irawan, "Comparing scientific computing environments for simulating 2D non-buoyant fluid parcel trajectory under inertial oscillation: A preliminary educational study", Indonesian Physical Review, vol. 7, no. 3, p 451-468, 2024.

DOI:
https://doi.org/10.29303/ipr.v7i3.335

Abstract

This study presents a preliminary numerical investigation of the two-dimensional trajectory of a non-buoyant fluid parcel subjected to inertial oscillations and abrupt external forcing events. The simulations were implemented using Python, GNU Octave, R, Julia, and Fortran open-source scientific computing environments. By running 1,000 iterations in each environment, we evaluated the computational performance of these languages in tackling this idealized problem. The results, visualized through static plots and animation, validate the numerical model's ability to represent the fundamental physics governing fluid motion. Statistical analysis using the Kruskal-Wallis test and Dunn's post-hoc test with Bonferroni correction revealed that Fortran exhibits significantly faster execution times than other environments. However, the choice of programming language should also consider factors such as coding expertise, library availability, and scalability requirements. This study focuses on the performance of scientific computing environments within each language rather than the languages themselves. The observed execution times should be interpreted in the context of the specific environments used, as they often leverage optimized libraries written in lower-level languages. Despite the limitations of this work, such as the simplified 2D model and the use of a single hardware configuration, this study provides valuable insights into selecting appropriate computational tools. It contributes to educational resources for teaching idealized fluid dynamics models. Future studies could explore more complex scenarios, a more comprehensive range of programming environments, and the impact of different numerical schemes and physical parameterizations.

Introduction

The trajectory of neutrally buoyant fluid parcels in oscillating flows is a fundamental problem in geophysical fluid dynamics (GFD), with implications for oceanic and atmospheric processes. Inertial oscillations, resulting from the Earth's rotation, significantly influence the behavior of these fluid parcels [1]. Understanding the trajectories of non-buoyant particles subjected to oscillating flows is crucial for predicting the transport and dispersion of tracers in the oceans and atmosphere [2].

Geophysical fluids are characterized by vast spatial scales, complex dynamics, and the influence of rotational effects [3–5]. The Coriolis force, arising from the Earth's rotation, poses challenges in modeling fluid motions accurately. Inertial oscillations cause fluid parcels to oscillate around their mean trajectory, affecting transport and mixing processes in geophysical systems. Analytical solutions for fluid parcel trajectories under inertial oscillations are often limited to simplified cases or require restrictive assumptions. Consequently, numerical simulations have become essential for exploring the dynamics of these systems [6–8], particularly in two-dimensional (2D) scenarios where vertical motions are neglected. These idealized 2D models provide a starting point for understanding the fundamental physics governing fluid parcel trajectories and inform the development of more advanced models.

In this paper, we present a preliminary numerical investigation of the 2D trajectory of a non-buoyant fluid parcel under inertial oscillations. Although the problem setup is based on a textbook example [28], we focus on the computational analysis and comparison of different scientific computing environments for implementing the numerical solution. We aim to provide insights into the selection and performance of computational tools for simulating such idealized scenarios, emphasizing educational value.

We implement the simulation using open-source scientific computing environments in multiple programming languages: Python, GNU Octave, R, Julia, and Fortran. These environments are widely adopted in scientific computing due to their extensive library support and ease of use [9-11]. It is important to note that our comparison is not primarily focused on the programming languages themselves, but rather on the performance of the scientific computing environments within each language. This approach allows us to evaluate the suitability and efficiency of these environments for the specific problem at hand, considering factors such as library availability, ease of implementation, and computational performance [57, 58].

In this paper, we present a preliminary numerical investigation of the 2D trajectory of a non-buoyant fluid parcel under inertial oscillations. Although the problem setup is based on a textbook example [28], our focus is on the computational analysis and comparison of different scientific computing environments for implementing the numerical solution. We aim to provide insights into the selection and performance of computational tools for simulating such idealized scenarios, emphasizing educational value.

We implement the simulation using open-source scientific computing environments in multiple programming languages: Python, GNU Octave, R, Julia, and Fortran. These environments are widely adopted in scientific computing due to their extensive library support and ease of use [9-11]. It is important to note that our comparison is not primarily focused on the programming languages themselves, but rather on the performance of the scientific computing environments within each language. This approach allows us to evaluate

the suitability and efficiency of these environments for the specific problem at hand, considering factors such as library availability, ease of implementation, and computational performance [57, 58].

The choice of running the simulations for 1,000 iterations is based on a careful balance between statistical robustness and computational feasibility. While a larger number of iterations generally leads to more accurate and reliable results, it also increases the computational cost. We argue that 1,000 iterations provide a sufficiently large sample size for meaningful statistical analysis while maintaining a reasonable computational burden, especially considering the educational context of this study. This iteration count allows us to capture the essential dynamics of the system and demonstrate the performance differences between the scientific computing environments, which are key objectives of our work. Moreover, 1,000 iterations ensure that the simulation time remains manageable for students and researchers working with limited computational resources, making the study more accessible and reproducible in educational settings.

We argue that a detailed accuracy assessment against analytical solutions or across implementations is not necessary for the scope and objectives of this study. We focus on the comparative performance analysis and educational value of implementing the numerical solution in different programming environments. The employed numerical schemes, such as the semi-implicit scheme and finite difference discretization, are well-established and validated in the literature [13, 14]. We rely on their inherent accuracy and consistent implementation across the environment.

The objectives of this paper are threefold. First, we describe the numerical methods used to simulate the 2D trajectory of a non-buoyant fluid parcel under inertial oscillations. Second, we evaluate the computational performance of scientific computing environments in Python, GNU Octave, R, Julia, and Fortran in tackling this problem, offering insights for researchers and educators in the GFD community. Third, we contribute to the educational resources available for teaching and learning about idealized models in fluid dynamics.
This study aims to address a gap in literature by analyzing the numerical simulation of this idealized problem across multiple scientific computing environments, emphasizing performance evaluation and educational value. The insights gained will guide the selection of computational tools and support teaching and understanding fundamental concepts in GFD. While the choice of programming environment depends on various factors, our findings serve as a reference point for researchers and educators.

Theory and Calculation

To investigate the motion of a fluid parcel influenced by inertial oscillations in two dimensions, we start with the Navier-Stokes equations, which describe the conservation of momentum in fluid dynamics. The Navier-Stokes equations account for external forces, pressure gradients, and viscous forces within the fluid [17, 18]:

$$\rho(\partial_t \boldsymbol{u} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \mu \nabla^2 \boldsymbol{u} + \boldsymbol{f} \tag{1}$$

Where, ρ is the fluid density, $\boldsymbol{u} \equiv (\boldsymbol{u}, \boldsymbol{v})$ is the fluid velocity vector with components u and v in the x and y directions, p is the pressure, μ is the dynamic viscosity, and $\boldsymbol{f}$ represents body forces per unit volume. In 2D flow, the Navier-Stokes equations can

be separated into two components, one for momentum in the x-direction and one for the y-direction:

$$\rho\left(\partial_t u + u\,\partial_x u + v\,\partial_y u\right) = -\,\partial_x p + \mu\left(\partial_{xx}^2 u + \partial_{yy}^2 u\right) + f_x$$
$$\rho\left(\partial_t v + u\,\partial_x v + v\,\partial_y v\right) = -\,\partial_y p + \mu\left(\partial_{xx}^2 v + \partial_{yy}^2 v\right) + f_y \quad (2)$$

We make several assumptions to simplify the Navier-Stokes equations for practical applications in geophysical fluid dynamics [3]. The fluid is treated as a Newtonian fluid with constant viscosity, and the flow is assumed incompressible. We apply the continuum assumption, treating the fluid as a continuous medium, and consider the fluid properties to be isotropic. The no-slip boundary condition is imposed at solid boundaries, and external forces other than body forces are neglected unless expressly stated.

Focusing on 2D inertial oscillations, we further assume that the flow is predominantly horizontal ($w = 0$) with no vertical gradients ($\partial_z(.)$ terms are zero). This assumption suits scenarios where flow dynamics are confined to a thin layer, such as near the ocean surface or in the atmosphere away from significant topographical influences [19–22]. We consider environments where pressure gradient forces are negligible compared to the Coriolis and external forcing terms, which may occur in conditions with strong rotational effects or relatively uniform pressure fields.

The external forcings influencing the flow dynamics are defined as:

$$f_x = \partial_t u_f$$
$$f_y = \partial_t v_f \quad (3)$$

, where u_f and v_f represent uniform external forcings in space, such as wind stress or other steady influences. This assumption allows us to focus on the effect of external forcings without the complexities introduced by spatial variability.

In our simplified model, we neglect viscosity due to its minimal impact compared to inertial terms and assume irrotational flow, simplifying the momentum equations. The Navier-Stokes equations then reduce primarily to expressions influenced by inertial effects and the Coriolis force, introduced through the Coriolis parameter $\boldsymbol{f}$, twice the angular velocity of Earth's rotation [3]

$$\partial_t u = -f\,v + f_x$$
$$\partial_t v = -f\,u + f_y \quad (4)$$

These equations govern the inertial oscillations in a rotating reference frame, simplified to highlight the oscillatory behavior of fluid parcels. We focus on predicting the pathway of a non-buoyant fluid parcel, which further simplifies our assumptions by neglecting buoyancy effects [3]. The pathway of these fluid parcels is given by:

$$\dot{x} = U_0 + u \quad (5)$$

$$\dot{y} = V_0 + v$$

where U_0 and V_0 represent the ambient uniform flow and u and v are the velocity perturbations due to inertial oscillations.

While this model is highly simplified, it provides a valuable framework for understanding and predicting the movement of non-buoyant fluid parcels under specific geophysical conditions. A rigorous derivation would involve linearizing the Navier-Stokes equations, possibly introducing adjustments for external forcing terms, and considering geostrophic balance for large-scale geophysical flows. However, the current formulation captures the essential physics of inertial oscillations and is a valuable educational tool for exploring fluid parcel trajectories in idealized scenarios.

It is important to note that the assumptions made in this simplified model, such as neglecting viscosity and assuming irrotational flow, are not always valid in real-world geophysical flows. More complex models incorporating additional physical factors and spatial variability may be necessary for accurate predictions in certain situations. Nevertheless, this idealized model provides a solid foundation for understanding the fundamental dynamics of inertial oscillations and serves as a steppingstone towards more sophisticated numerical simulations in geophysical fluid dynamics.

Experimental Method

In this study, we employ two numerical techniques to predict the trajectory of non-buoyant fluid parcels in a rotating fluid system under the influence of inertial oscillations: the semi-implicit approach and the local rotation method. These methods offer distinct perspectives and computational strategies for modeling the Coriolis force and other relevant factors.

The semi-implicit approach utilizes the following numerical scheme to predict the fluid parcel trajectory, as described in equation 4:

$$u^{n+1} = \frac{(1 - \beta)u^n + \alpha v^n}{1 + \beta}$$
$$v^{n+1} = \frac{(1 - \beta)v^n - \alpha u^n}{1 + \beta} \quad (6)$$

where u^n and v^n are the velocities at the current time step n, u^{n+1} and v^{n+1} are the velocities at the next time step $n + 1$, and the parameters α and β are defined as $\alpha = \Delta t\, f$ and $\beta = \alpha/4$. This semi-implicit scheme efficiently integrates the inertial oscillation equations over time, providing accurate predictions of the fluid parcel's velocity components (u, v). To predict the x and y coordinates of a non-buoyant fluid parcel, we discretize the kinematic equation (equation 5) using finite differences:

$$\Delta x = \frac{\alpha v^n}{1 + \beta}\Delta t \quad (7)$$

$$\Delta y = -\frac{\alpha u^n}{1+\beta}\Delta t$$

The local rotation method simulates the Coriolis force using a velocity vector rotation:

$$u^{n+1} = \cos(\theta)u^n + \sin(\theta)u^n$$
$$v^{n+1} = \cos(\theta)v^n - \sin(\theta)u^n \quad (8)$$

where θ is determined based on the time step Δt and the Coriolis parameter f, given by $\theta = 2\sin^{-1}(\frac{\Delta t}{2} f)$. This method effectively captures the Coriolis effect by rotating the velocity components, aiding in predicting the trajectory of fluid parcels in a rotating system. By incorporating the finite difference approximation to equation 5, we obtain:

$$\Delta x = (\cos(\theta) - 1)\, u^n \Delta t + \sin(\theta) v^n \Delta t$$
$$\Delta y = \cos(\theta) v^n \Delta t - (\sin(\theta)+1) u^n \Delta t \quad (9)$$

In our simulation, we model the ambient flow as a uniform northeastward flow with values of $U_0 = 5$ cm/s and $V_0 = 5$ cm/s. The total simulation time is 6 days with Δt = 4320 seconds (approximately 1.2 days). Three abrupt events change the relative flow speed and direction. $(\Delta u\, f, \Delta v\, f)$, as explained in Table 1.

Table 1. Velocity disturbance parameters

time (days)	$\Delta \boldsymbol{u_f}$ (cm / s)	$\Delta \boldsymbol{v_f}$ (cm / s)
1	10	0
2	10	0
4	0	10

We implement these numerical schemes using various open-source computing environments widely used in earth science and geophysical fluid dynamics. Initially, we apply the schemes using Fortran, a language with a long history and extensive use in solving such problems [23–26]. Fortran remains a staple in general circulation models (GCMs) [27], and the 1995 version has been pivotal in many classic problem-solving scenarios within this domain [28, 29].

We also implement the solutions in Python, leveraging the NumPy library [30]. Python has gained prominence in scientific computing due to its versatility and extensive libraries, making it a popular choice for numerical simulations and statistical computations in earth sciences [31–35]. GNU Octave, an open-source alternative to MATLAB®, is explored as well, offering a cost-effective solution with a syntax familiar to MATLAB® users. This makes it an appealing option for GFD modelers and researchers interested in numerical computations [36–38].

Additionally, we investigate Julia, a computing environment gaining traction among GFD modelers due to its ability to match Fortran's speed while maintaining Python's ease of use.

Many researchers are considering transitioning from Fortran to Julia for ocean models, attracted by this combination of performance and user-friendliness [37, 39–43]. Lastly, we conduct numerical calculations in R, a popular choice within the atmospheric and oceanic sciences communities [44–47] due to its data analysis and visualization strengths.

The resulting simulation data is preserved in a structured format as a text file, ensuring accessibility and easy manipulation for further analysis and visualization. For visualization, we utilize the Matplotlib library [48] within the Python environment, generating static plots in Portable Network Graphics (.png) format and animations in Graphics Interchange Format (.gif). These widely supported formats ensure the accessibility and usability of the visualized data across different systems and applications.

To rigorously evaluate and compare the execution times across diverse programming languages, we employ Python code to execute each numerical solver 1,000 times, ensuring robust statistical sampling. Crucial metrics like execution time, return codes, standard output, and errors are captured using the Subprocess library. The resulting dataset is structured into a Pandas DataFrame [49] and exported to comma-separated values (.csv) for analysis.

For statistical comparison, we applied the Kruskal-Wallis test [50], a non-parametric method suitable for comparing multiple independent groups [51]. The Kruskal-Wallis test evaluates the null hypothesis that the medians of all groups are equal, indicating no significant difference in performance among programming languages. This test produces a test statistic H along with a p-value:

$$H = \frac{12}{N(N+1)} \sum_{j=1}^{k} \frac{R_j^2}{n_j} - 3(\mathrm{N}+1) \tag{10}$$

In this case, N is the total number of observations, k is the number of groups, n_j is the number of observations in the jth group, and R_j is the sum of ranks for the jth group. The degrees of freedom for the Kruskal-Wallis test is $df = k - 1$.

If the p-value from the Kruskal-Wallis test was below a pre-defined significance level $\alpha = 0.05$, we proceeded with Dunn's post-hoc test [52]. Dunn's test is used for pairwise comparisons between groups to identify which groups exhibit statistically significant differences in performance [53].

Dunn's test statistic for pairwise comparisons between groups i and j is given by:

$$Z_{ij} = \frac{|R_i - R_j| - (N(N+1))\,/\,12}{\sqrt{N(N+1)(N+2)\,/\,12}} \tag{11}$$

The critical value for Dunn's test was obtained using the Bonferroni adjustment, where the significance level α was divided by the number of pairwise comparisons m to control for multiple testing:

$$\alpha_{adjusted} = \frac{\alpha}{m} \tag{12}$$

Pairwise comparisons with $|Z_{ij}|$ exceeding the adjusted critical value indicate statistically significant differences between the corresponding groups. We performed these calculations automatically using the statistics module in the SciPy [54] and the scikit-posthoc [55] libraries

in the Python computing environment. This rigorous statistical approach ensured reliable insights into the computational performance of multiple computing environments for simulating a 2D fluid parcel trajectory over 1,000 iterations.

While the numerical schemes employed in this study are inspired by the work of Kämpf [28], we have adapted and expanded upon the original implementation to suit the specific objectives of our research. The primary focus of our study is to compare the performance of different programming languages and computing environments in simulating the trajectory of non-buoyant fluid parcels under inertial oscillations rather than to introduce novel numerical methods or to conduct a comprehensive accuracy assessment.

We argue that a detailed comparison of the simulation results against analytical solutions or across different implementations is not strictly necessary for the scope and goals of this study. The semi-implicit scheme and finite difference discretizations used in our simulations are well-established numerical methods that have been extensively validated and applied in geophysical fluid dynamics [63, 64]. These methods have provided accurate and reliable results for various problems, including the simulation of inertial oscillations [65, 66].

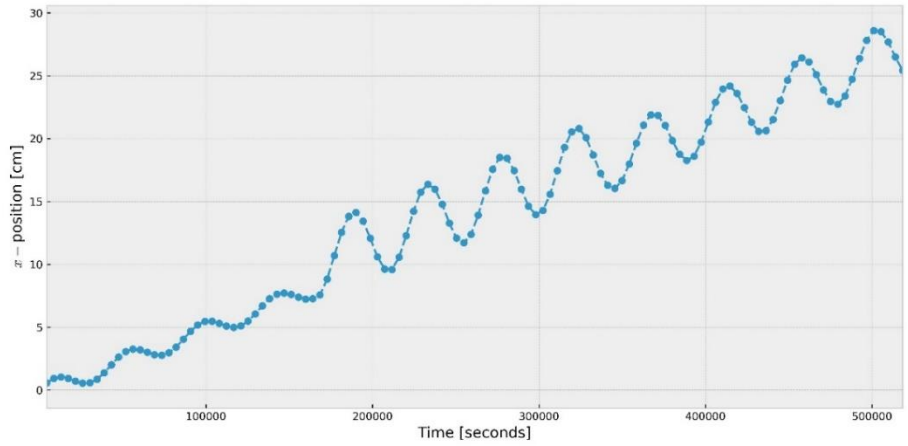
Moreover, the primary objective of this study is to evaluate the computational performance and educational value of implementing the numerical solution in different programming languages and environments. By focusing on the relative performance differences between the implementations rather than on absolute accuracy, we can gain valuable insights into the suitability and efficiency of each programming language for this specific problem.

It is important to acknowledge that this study's numerical schemes and model setup are simplified representations of the complex dynamics governing fluid parcel trajectories in real-world geophysical flows. The assumptions made, such as neglecting vertical motion and assuming a uniform ambient flow, limit the direct applicability of the results to more realistic scenarios. However, these simplifications are intentional and create an accessible and easily understood educational example highlighting the key concepts and challenges of simulating inertial oscillations.

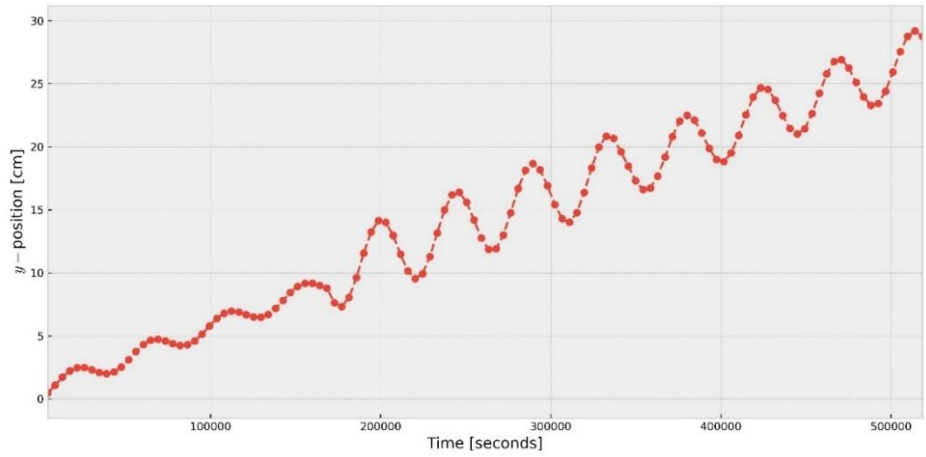
In summary, our experimental methodology builds upon the foundation laid by Kämpf [28] but adapts and extends the original implementation to align with the specific goals of our study. By focusing on the comparative performance of different programming languages and environments rather than on comprehensive accuracy assessments, we can provide valuable insights into the computational aspects of simulating fluid parcel trajectories under inertial oscillations while maintaining a strong emphasis on educational value and accessibility.

Result and Discussion

Simulations were conducted to model the trajectory of a non-buoyant fluid parcel under the combined influence of inertial oscillations induced by the Earth's rotation, uniform ambient northeasterly flow over six days, and abrupt disturbance events on days one, two, and four. The resulting trajectories, plotted against time in Figures 1a and 1b, exhibit oscillatory behavior with increasing amplitude due to Coriolis effects. The slight path deviations observed can be attributed to differences in numerical precision across the various programming environments.



(a)



(b)

Figure 1. Temporal evolution of a non-buoyant fluid parcel undergoing inertial oscillation and abrupt forcing events in (a) x and (b) y directions.

Figure 2 reveals a spiraling cyclical trajectory forming expanding loops consistent with the expected inertial oscillation pattern. The triangle markers represent the discretization points along the fluid parcel trajectory, effectively indicating the temporal evolution of the parcel's position. The distinct perturbations evident in the trajectory are likely caused by simulated disturbance events, which have the potential to amplify or dampen the oscillations and significantly impact parcel transport and dispersion. While we previously suggested that the increasing oscillation amplitudes might be due to numerical precision, this claim requires further investigation through simulations with varying time step sizes. Without such analysis, we refrain from attributing the amplitude growth to any specific factor and acknowledge the need for additional studies to draw definitive conclusions.

In this idealized scenario, the observed trajectory patterns validate the numerical models' ability to represent the fundamental rotational dynamics governing fluid motion. Although implementation differences across programming environments were minor in this case, they highlight the importance of numerical accuracy and algorithm design for faithfully representing intricate fluid behavior, which could be amplified under more complex

conditions. The consistency of the results across different environments reinforces the robustness of the underlying numerical methods and their suitability for educational purposes.

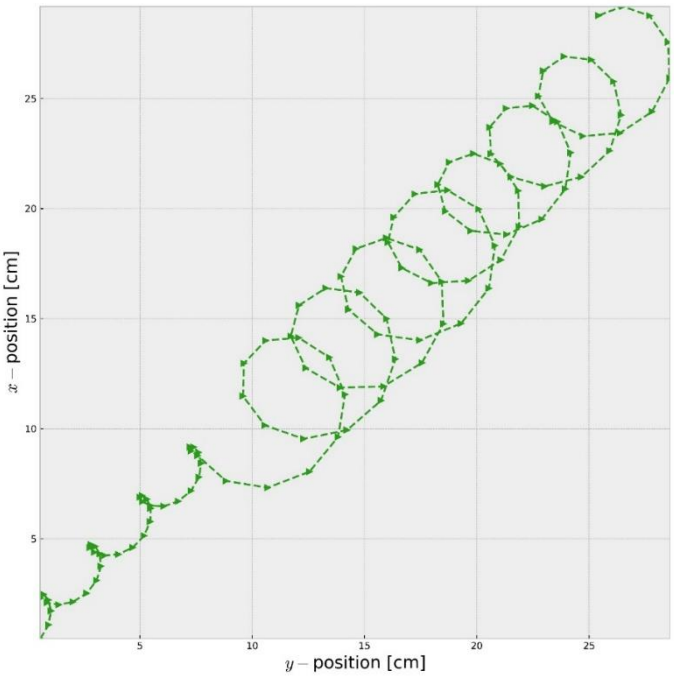


Figure 2. The trajectory of a fluid element is propelled within an ambient flow and undergoes inertial oscillation.

Figure 3 examines the execution times of various computing environments for simulating the trajectory of a fluid parcel under inertial oscillations in a 2D geophysical fluid system. Based on 1,000 simulations in each environment (Fortran, Python, Julia, GNU Octave, and R), Fortran is the leader in speed and consistency. The boxplots and statistical data support this observation, with Fortran exhibiting the lowest mean execution time of 0.01 seconds, closely aligned with its median of 0.01 seconds. Furthermore, Fortran demonstrates an exceptionally low standard deviation of 0.001 seconds, indicating remarkable consistency in performance.

In contrast, Python, R, Julia, and GNU Octave display more significant variability in execution times, as evidenced by their more extensive interquartile ranges (IQRs) and the presence of outliers. Among these environments, Python delivers a median execution time of 0.47 seconds, faster than R (1.01 seconds) and GNU Octave (0.52 seconds). Additionally, Python boasts a relatively small IQR of 0.064 seconds. However, Julia lags considerably with a median execution time of 3.8 seconds and a larger IQR of 1.14 seconds, indicating that half of the simulations in Julia took between 3.12 and 4.26 seconds to complete.

The standard deviations substantiate the variability in execution times across the interpreted languages. Julia exhibits the most significant standard deviation of 0.593 seconds, followed by GNU Octave (0.090 seconds) and Python (0.046 seconds). The presence of outliers in Julia (one at 6.86 seconds) and Octave (one at 1.69 seconds) reinforces this observation.

It is important to note that our comparison focuses on the performance of scientific computing environments within each programming language rather than the languages in isolation. While pure Julia has been shown to outperform pure Python in certain benchmarks, our study utilizes the scientific computing ecosystems available in each language, such as SciPy in Python and the Julia programming language itself. These ecosystems often leverage optimized libraries written in lower-level languages like C, which can significantly impact performance. Therefore, the observed execution times should be interpreted in the context of the specific scientific computing environments used, rather than as a direct comparison of the languages' inherent speeds.

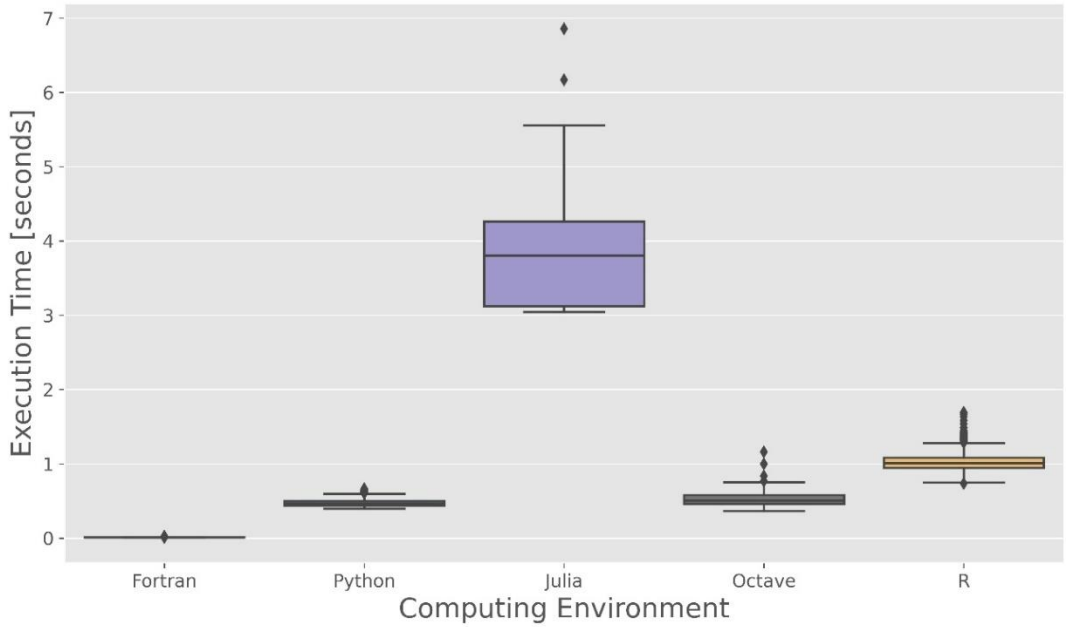


Figure 3. Box plots comparing the execution times of simulating the 2D trajectory of a non-buoyant fluid parcel under inertial oscillations using different open-source programming languages (Fortran, Python, Julia, GNU Octave, and R) over 1,000 iterations.

Regarding the statistical analysis, we reported p-values as zeros based on the results obtained from the Kruskal-Wallis test and Dunn's post-hoc test with Bonferroni correction. The Kruskal-Wallis test yielded a statistically significant result (p-value = 0.000, test statistic = 4577.973), indicating that at least one environment exhibits a median execution time significantly different from the others. Dunn's post-hoc test revealed that Fortran's execution times were statistically different from all other environments (Julia, Octave, Python, and R) at a significance level of $\alpha = 0.05$ (all p-values were 0.000). Furthermore, pairwise comparisons among the interpreted languages showed statistically significant differences in their median execution times

(all p-values were 0.000) (Fig. 4).

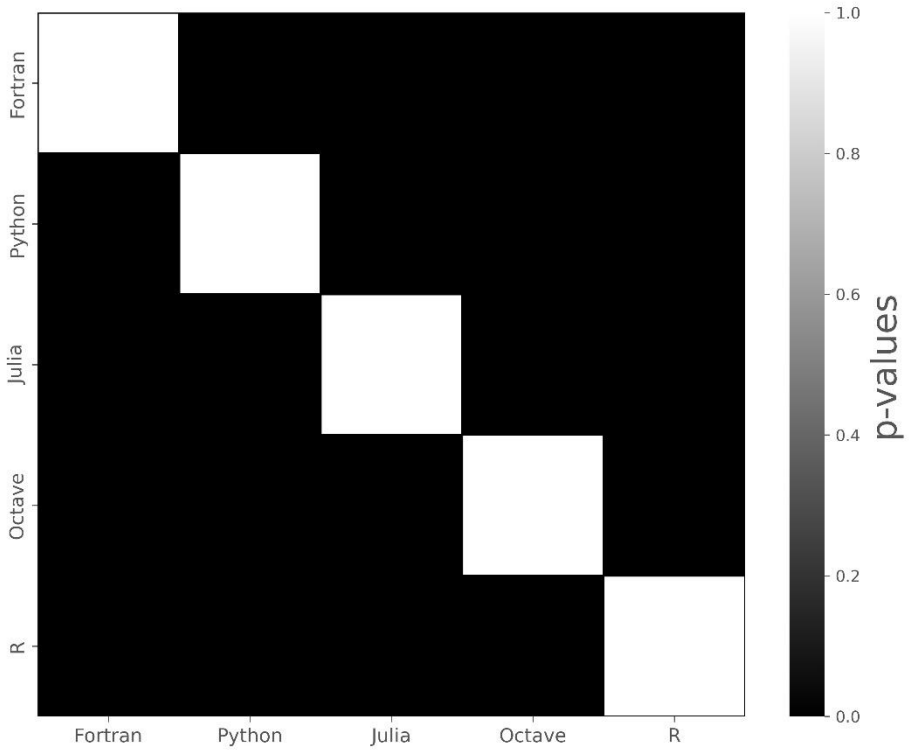


Figure 4. Box plots comparing the execution times of simulating the 2D trajectory of a non-buoyant fluid parcel under inertial oscillations using different open-source programming languages (Fortran, Python, Julia, GNU Octave, and R) over 1,000 iterations.

We acknowledge the importance of verifying the accuracy of these calculations and ensuring the appropriate use of statistical tests. In the revised manuscript, we provide a more detailed description of the statistical methods employed and their assumptions, as well as discuss the limitations of our analysis given the relatively small sample size. We emphasize the need for caution in interpreting the results, as they may not generalize to a wider range of scenarios or programming implementations.

It is crucial to recognize that our study was conducted on a single machine: a Fedora Linux 39 (Budgie) x86_64 system with a 20LB0021US ThinkPad P52s laptop equipped with an Intel i7-8550U (8) @4.000GHz CPU. This hardware limitation may impact the generalizability of our results, as performance characteristics could vary across different systems and architectures. Future studies could explore the performance of these programming environments on a wider range of hardware configurations to provide a more comprehensive understanding of their behavior.

Conclusion

The numerical simulation of the trajectory of a non-buoyant fluid parcel under inertial oscillation in a two-dimensional geophysical fluid system provides valuable insights into the fundamental physics governing fluid parcel transport and dispersion. By leveraging open-source scientific computing environments in Python, GNU Octave, R, Julia, and Fortran, this

study contributes to the reproducibility and transparency of scientific research and facilitates collaborative knowledge sharing within the geophysical fluid dynamics community.

The evaluation of computational performance across multiple programming environments revealed Fortran as the most efficient choice for simulating this idealized scenario. Statistical analysis, including the Kruskal-Wallis test and Dunn's post-hoc test with Bonferroni correction, confirmed that Fortran exhibited significantly faster execution times compared to the other environments. However, the selection of an appropriate programming language should also consider factors such as coding expertise, availability of specialized libraries, and scalability requirements.

It is crucial to recognize that our study focused on the performance of scientific computing environments within each language, rather than the languages in isolation. The observed execution times should be interpreted in the context of the specific environments used, as they often leverage optimized libraries written in lower-level languages. Therefore, the performance differences cannot be solely attributed to the inherent characteristics of the programming languages themselves.

While this study provides valuable insights into the performance characteristics of different scientific computing environments, it is important to acknowledge its limitations. The idealized nature of the problem setup, with simplified assumptions such as two-dimensional flow and uniform ambient conditions, may not fully capture the complexities of real-world geophysical fluid dynamics. Additionally, the focus on a single test case and the use of a single hardware configuration (a Fedora Linux 39 system with an Intel i7-8550U CPU) may limit the generalizability of the results.

Future studies could explore more complex scenarios, incorporate additional physical factors and spatial variability, and investigate the performance impact of different numerical schemes, grid resolutions, and physical parameterizations. Expanding the analysis to a wider range of programming environments and hardware configurations would provide a more comprehensive understanding of the performance landscape.

Despite these limitations, the present study serves as a valuable educational resource, introducing students and researchers to the process of implementing and comparing numerical simulations across multiple programming environments. By providing hands-on experience with a tractable problem, this work promotes a deeper understanding of the interplay between physical models, numerical methods, and computational tools in geophysical fluid dynamics.

In summary, this study contributes to the growing body of knowledge in geophysical fluid dynamics education and computational performance analysis. The insights gained can guide the selection of appropriate computational tools and inform future educational initiatives. While the choice of programming environment depends on various factors, our findings serve as a reference point for researchers and educators in the field. Further investigations and discussions on the optimal use of computational tools in geophysical fluid dynamics research and education are encouraged, considering the limitations and potential extensions of this preliminary study.

Acknowledgment

We extend our sincere gratitude to Andrew J. Ridgwell, Gayatri Mishra, and Sandra K. Turner for their invaluable discussions on geophysical fluid dynamics within oceanic realms, and to Faiz R. Fajary for insightful conversations regarding atmospheric dynamics. Their collective expertise greatly enriched our understanding and contributed significantly to this work. We are also grateful for the generous support from the Dean's Distinguished Fellowship at the University of California, Riverside (UCR) in 2023 and the ITB Research, Community Services, and Innovation Program (PPMI-ITB) in 2024, which made this research possible. The code and complete runtime dataset associated with this paper are accessible through our GitHub repository: https://github.com/sandyherho/inerOsci.

References

[1] K. Hasselmann, "Wave-driven inertial oscillations," Geophys. Astrophys. Fluid Dyn., vol. 1, no. 3–4, pp. 463–502, 1970, doi: 10.1080/03091927009365783.

[2] B. Voisin, "Buoyancy oscillations," J. Fluid Mech., vol. 984, p. A29, 2024, doi: 10.1017/jfm.2024.179.

[3] B. Cushman-Roisin and J.-M. Beckers, Introduction to geophysical fluid dynamics: physical and numerical aspects. Oxford: Academic Press, 2011.

[4] J. Pedlosky, Geophysical Fluid Dynamics. Berlin: Springer Science & Business Media, 2013.

[5] G. K. Vallis, "Geophysical fluid dynamics: whence, whither and why?," Proc. Math. Phys. Eng. Sci., vol. 472, no. 2192, p. 20160140, 2016, doi: 10.1098/rspa.2016.0140.K.

[6] J. D. Denton and W. N. Dawes, "Computational fluid dynamics for turbomachinery design," Proc. Inst. Mech. Eng., Part C, vol. 213, no. 2, pp. 107–124, 1998, doi: 10.1243/0954406991522211.

[7] R. Malki, A. J. Williams, T. N. Croft, M. Togneri, and I. Masters, "A coupled blade element momentum–Computational fluid dynamics model for evaluating tidal stream turbine performance," Appl. Math. Model., vol. 37, no. 5, pp. 3006–3020, 2013, doi: 10.1016/j.apm.2012.07.025.

[8] C. Windt, J. Davidson, and J. v Ringwood, "High-fidelity numerical modeling of ocean wave energy systems: A review of computational fluid dynamics-based numerical wave tanks," Renewable and Sustainable Energy Reviews, vol. 93, pp. 610–630, 2018, doi: 10.1016/j.rser.2018.05.020.

[9] A. Bhatt, T. Valentic, A. Reimer, L. Lamarche, P. Reyes, and R. Cosgrove, "Reproducible Software Environment: a tool enabling computational reproducibility in geospace sciences and facilitating collaboration," J. Space Weather Space Clim., vol. 10, p. 12, 2020, doi: 10.1051/swsc/2020011.

[10] L. Talirz et al., "Materials Cloud, a platform for open computational science," Sci. Data, vol. 7, no. 1, p. 299, 2020, doi: 10.1038/s41597-020-00637-5.

[11] M. Beg et al., "Using Jupyter for reproducible scientific workflows," Comput. Sci. Eng., vol. 23, no. 2, pp. 36–46, 2021, doi: 10.1109/MCSE.2021.3052101.

[12] N. Lazar, "Ockham's Razor," Wiley Interdiscip. Rev. Comput. Stat., vol. 2, no. 2, pp. 243–246, 2010, doi: 10.1002/wics.75.

[13] N. Jeevanjee, P. Hassanzadeh, S. Hill, and A. Sheshadri, "A perspective on climate model hierarchies," J. Adv. Model. Earth Syst., vol. 9, no. 4, pp. 1760–1771, 2017, doi: 10.1002/2017MS001038.

[14] D. Song et al., "Near-inertial oscillations in seasonal highly stratified shallow water," Estuar. Coast. Shelf Sci., vol. 258, p. 107445, 2021, doi: 10.1016/j.ecss.2021.107445.

[15] A. K. Mirza, H. F. Dacre, and C. H. B. Lo, "A case study analysis of the impact of a new free tropospheric turbulence scheme on the dispersion of an atmospheric tracer," Q. J. R. Meteorol. Soc., 2024, doi: 10.1002/qj.4681.

[16] F. J. Beron-Vera, "Nonlinear dynamics of inertial particles in the ocean: From drifters and floats to marine debris and Sargassum," Nonlinear Dynamics, vol. 103, no. 1, pp. 1–26, 2021, doi: 10.1007/s11071-020-06053-z.

[17] S. v Ershkov, E. Y. Prosviryakov, N. v Burmasheva, and V. Christianto, "Towards understanding the algorithms for solving the Navier–Stokes equations," Fluid Dyn. Res., vol. 53, no. 4, p. 44501, 2021, doi: 10.1088/1873-7005/ac10f0.

[18] J. W. Garvin, A student's guide to the Navier-Stokes equations. Cambridge: Cambridge University Press, 2023.

[19] Li, R., Chen, C., Dong, W., Beardsley, R.C., Wu, Z., Gong, W., Liu, Y., Liu, T., Xu, D.: Slope-intensified storm-induced near-inertial oscillations in the South China sea. J. Geophys. Res. Oceans p. 126, vol.3, 2020–016713, 2021, doi: 10.1029/2020JC016713.

[20] T. Hibiya, "A new parameterization of turbulent mixing enhanced over rough seafloor topography," Geophys. Res. Lett., vol. 49, no. 2, p. e2021GL096067, 2022, doi: 10.1029/2021GL096067.

[21] X. Luo, X. Huang, J. Fei, J. Wang, C. Li, and X. Cheng, "Role of Topography in Triggering Elevated Thunderstorms Associated With Winter Cold Fronts Over the Eastern Yunnan-Guizhou Plateau," J. Geophys. Res. Atmos., vol. 128, no. 8, p. e2023JD038640, 2023, doi: 10.1029/2023JD038640.

[22] Y. Zhu and X. Liang, "Near-inertial oscillations in the deep Gulf of Mexico," Deep-Sea Res. II: Top. Stud. Oceanogr., vol. 210, p. 105310, 2023, doi: 10.1016/j.dsr2.2023.105310.

[23] E. Price, J. Mielikainen, M. Huang, B. Huang, H.-L. A. Huang, and T. Lee, "GPU-accelerated longwave radiation scheme of the rapid radiative transfer model for general circulation models (RRTMG)," IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens., vol. 7, no. 8, pp. 3660–3667, doi: 10.1109/JSTARS.2014.2315771.

[24] M. Norman, J. Larkin, A. Vose, and K. Evans, "A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel," J. Comput. Sci., vol. 9, pp. 1–6, 2015, doi: 10.1016/j.jocs.2015.04.022.

[25] J. Ott, M. Pritchard, N. Best, E. Linstead, M. Curcic, and P. Baldi, "A Fortran-Keras deep learning bridge for scientific computing," Sci. Program., vol. 2020, pp. 1–13, 2020, doi: 10.1155/2020/8888811.

[26] W. A. Perkins, N. D. Brenowitz, C. S. Bretherton, and J. M. Nugent, "Emulation of cloud microphysics in a climate model," J. Adv. Model. Earth Syst., vol. 16, no. 4, p. e2023MS003851, 2024, doi: 10.1029/2023MS003851.

[27] I. M. Held and M. J. Suarez, "A Proposal for the Intercomparison of the Dynamical Cores of Atmospheric General Circulation Models," Bull. Am. Meteorol. Soc., vol. 75, no. 10, pp. 1825–1830, 1994, doi: 10.1175/1520-0477(1994)075%3C1825:APFTIO%3E2.0.CO;2.

[28] J. Kämpf, Ocean Modelling for Beginners: Using Open-Source Software. Berlin: Springer Science & Business Media, 2009.

[29] J. Kämpf, Advanced Ocean Modelling: Using Open-Source Software. Berlin: Springer Science & Business Media, 2010.

[30] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: a structure for efficient numerical computation," Comput. Sci. Eng., vol. 13, no. 2, pp. 22–30, 2011, doi: 10.1109/MCSE.2011.37.

[31] S. H. S. Herho and D. E. Irawan, "PY-METEO-NUM: Dockerized Python Notebook Environment for Portable Data Analysis Workflows in Indonesian Atmospheric Science Communities," Int. J. Data Sci., vol. 2, no. 1, pp. 38–46, 2021, doi: 10.18517/ijods.2.1.38-46.2021. S. H. S. Herho, "A Univariate Extreme Value Analysis and Change Point Detection of Monthly Discharge in Kali Kupang, Central Java, Indonesia," JOIV : Int. J. Inform. Visualization, vol. 6, no. 4, pp. 862–868, 2022, doi: 10.30630/joiv.6.4.953.

[32] S. H. S. Herho, "A Univariate Extreme Value Analysis and Change Point Detection of Monthly Discharge in Kali Kupang, Central Java, Indonesia," *JOIV : Int. J. Inform. Visualization*, vol. 6, no. 4, pp. 862–868, 2022, doi: 10.30630/joiv.6.4.953.

[33] Y.-K. Qian, "xinvert: A Python package for inversion problems in geophysical fluid dynamics," J. Open Source Softw., vol. 8, no. 89, p. 5510, 2023, doi: 10.21105/joss.05510.

[34] J. Yu, T. Mukerji, and P. Avseth, "rockphypy: An extensive Python library for rock physics modeling," SoftwareX, vol. 24, p. 101567, 2023, doi: 10.1016/j.softx.2023.101567.

[35] R. Suwarman et al., "imc-precip-iso: open monthly stable isotope data of precipitation over the Indonesian Maritime Continent," J. of Data, Inf. and Manag., pp. 1–12, 2024, doi: 10.1007/s42488-024-00116-1.

[36] J. B. D. Jaffrés, "GHCN-Daily: a treasure trove of climate data awaiting discovery," Comput. Geosci., vol. 122, pp. 35–44, 2019, doi: 10.1016/j.cageo.2018.07.003. O.

[37] Sulpis et al., "RADIv1: a non-steady-state early diagenetic model for ocean sediments in Julia and MATLAB/GNU Octave," Geosci. Model Dev., vol. 15, no. 5, pp. 2105–2131, 2022, doi: 10.5194/gmd-15-2105-2022.

[38] J. B. D. Jaffrés and J. L. Gray, "Chasing rainfall: estimating event precipitation along tracks of tropical cyclones via reanalysis data and in-situ gauges," Environ. Model. Softw., vol. 167, p. 105773, 2023, doi: 10.1016/j.envsoft.2023.105773.

[39] J. M. Perkel, "Julia: come for the syntax, stay for the speed," Nature, vol. 572, no. 7767, pp. 141–142, 2019, doi: 10.1038/d41586-019-02310-3.

[40] A. Ramadhan et al., "Oceananigans. jl: Fast and friendly geophysical fluid dynamics on GPUs," J. Open Source Softw., vol. 5, no. 53, 2020, doi: 10.21105/joss.02018.

[41] N. Constantinou, G. Wagner, L. Siegelman, B. Pearson, and A. Palóczy, "GeophysicalFlows. jl: Solvers for geophysical fluid dynamics problems in periodic domains on CPUs GPUs," J. Open Source Softw., vol. 6, no. 60, 2021, doi: 10.21105/joss.03053.

[42] S. Partee et al., "Using machine learning at scale in numerical simulations with SmartSim: An application to ocean climate modeling," J. Comput. Sci., vol. 62, p. 101707, 2022, doi: 10.1016/j.jocs.2022.101707.

[43] S. Bishnu, R. R. Strauss, and M. R. Petersen, "Comparing the Performance of Julia on CPUs versus GPUs and Julia-MPI versus Fortran-MPI: a case study with MPAS-Ocean (Version 7.1)," Geosci. Model Dev., vol. 16, no. 19, pp. 5539–5559, 2023, doi: 10.5194/gmd-16-5539-2023.

[44] B. Czernecki, A. Głogowski, and J. Nowosad, "Climate: An R package to access free in-situ meteorological and hydrological datasets for environmental assessment," Sustainability, vol. 12, no. 1, p. 394, 2020, doi: 10.3390/su12010394.

[45] S. H. S. Herho, F. Brahmana, K. E. P. Herho, and D. E. Irawan, "Does ENSO significantly affect rice production in Indonesia? A preliminary study using computational time-series approach," Int. J. Data Sci., vol. 2, no. 2, pp. 69–76, 2021, doi: 10.18517/ijods.2.2.69-76.2021.

[46] N. P. McKay, J. Emile-Geay, and D. Khider, "GeoChronR–an R package to model, analyze and visualize age-uncertain paleoscientific data," Geochronology, vol. 2020, pp. 1–33, 2020, doi: 10.5194/gchron-3-149-2021.

[47] M. D. Ashkezari et al., "Simons collaborative marine atlas project (Simons CMAP): an open-source portal to share, visualize, and analyze ocean data," Limnol. Oceanogr. Methods, vol. 19, no. 7, pp. 488–496, 2021, doi: 10.1002/lom3.10439.

[48] J. D. Hunter, “Matplotlib: A 2D graphics environment,” Comput. Sci. Eng., vol. 9, no. 03, pp. 90–95, 2007, doi: 10.1109/MCSE.2007.55.

[49] W. McKinney, “pandas: a foundational Python library for data analysis and statistics,” Python for High Performance and Scientific Computing, vol. 14, no. 9, pp. 1–9, 2011.

[50] W. H. Kruskal and W. A. Wallis, “Use of Ranks in One-Criterion Variance Analysis,” J. Am. Stat. Assoc., vol. 47, no. 260, pp. 583–621, 1952, doi: 10.1080/01621459.1952.10483441.

[51] E. Ostertagova, O. Ostertag, and J. Kováč, “Methodology and application of the Kruskal-Wallis test,” Appl. Mech., vol. 611, pp. 115–120, 2014, doi: 10.4028/www.scientific.net/AMM.611.115.

[52] O. J. Dunn, “Multiple comparisons using rank sums,” Technometrics, vol. 6, no. 3, pp. 241–252, 1964, doi: 10.1080/00401706.1964.10490181.

[53] G. D. Ruxton and G. Beauchamp, “Time for some a priori thinking about post hoc testing,” Behav. Ecol., vol. 19, no. 3, pp. 690–693, 2008, doi: 10.1093/beheco/arn020.

[54] P. Virtanen et al., “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” Nat. Methods, vol. 17, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.

[55] M. A. Terpilowski, “scikit-posthocs: Pairwise multiple comparison tests in Python,” J. Open Source Softw., vol. 4, no. 36, p. 1169, 2019, doi: 10.21105/joss.01169.

[57] K. J. Millman and M. Aivazis, "Python for Scientists and Engineers," Computing in Science Engineering, vol. 13, no. 2, pp. 9–12, Mar. 2011, doi: 10.1109/MCSE.2011.36.

[58] T. Kluyver et al., "Jupyter Notebooks – a publishing format for reproducible computational workflows," in Positioning and Power in Academic Publishing: Players, Agents and Agendas, 2016, pp. 87–90.

[63] R. Hallberg, "Using a resolution function to regulate parameterizations of oceanic mesoscale eddy effects," Ocean Model., vol. 72, pp. 92–103, 2013, doi: 10.1016/j.ocemod.2013.08.007.

[64] P. Bougeault and P. Lacarrere, "Parameterization of Orography-Induced Turbulence in a Mesobeta--Scale Model," Mon. Weather Rev., vol. 117, no. 8, pp. 1872–1890, 1989, doi: 10.1175/1520-0493(1989)117<1872:POOITI>2.0.CO;2.

[65] G. Danabasoglu, S. C. Bates, B. P. Briegleb, S. R. Jayne, M. Jochum, W. G. Large, S. Peacock, and S. G. Yeager, "The CCSM4 Ocean Component," J. Clim., vol. 25, no. 5, pp. 1361–1389, 2012, doi: 10.1175/JCLI-D-11-00091.1.

[66] G. Madec, "NEMO ocean engine," Institut Pierre-Simon Laplace (IPSL), France, No. 27, ISSN No 1288-1619, 2008, doi: 10.5281/zenodo.3248739.