Benchmarking Modern Scientific Computing Platforms for 2D Potential Flow Solver

Sandy Herho, Siti Kaban, Iwan Anwar

Benchmarking Modern Scientific Computing Platforms for 2D Potential Flow Solver

Sandy H. S. Herho[1*], Siti N. Kaban[2] and Iwan P. Anwar[3]

[1*]Department of Earth and Planetary Sciences, University of California, 900 University Ave., Riverside, 92521, CA, USA.
[2]Offshore Engineering Research Group, Bandung Institute of Technology (ITB), Jalan Ganesha 10, Bandung, 40132, West Java, Indonesia.
[3]Oceanography Research Group, Bandung Institute of Technology (ITB), Jalan Ganesha 10, Bandung, 40132, West Java, Indonesia.

*Corresponding author(s). E-mail(s): sandy.herho@email.ucr.edu;

Abstract

We present a comprehensive performance analysis of three major scientific computing platforms—Python, Julia, and R—in solving two-dimensional potential flow around dual square obstacles. We implemented a numerical solution using the Jacobi iteration method across all three platforms, with careful consideration of boundary conditions and convergence criteria. The numerical implementation successfully demonstrated convergence with an $\boldsymbol{L_2}$ norm reduction from $\mathbf{10^0}$ to $\mathbf{10^{-2}}$ over 40,000 iterations, capturing fundamental flow physics including stagnation points ($\boldsymbol{C_p \approx 1}$), flow acceleration regions ($\boldsymbol{C_p < 0}$), and obstacle interference effects. Through 200 independent executions and 10,000 bootstrap iterations, our comparative performance analysis revealed that Julia achieved superior computational efficiency with a median runtime of approximately 30 seconds, despite higher memory requirements (512MB), while Python offered balanced performance (45s, 108MB) and R showed higher variability in execution time (65s, 185MB). Statistical significance of these performance differences was confirmed through rigorous testing using Mann-Whitney U tests with Bonferroni correction ($\boldsymbol{\alpha_{\text{adjusted}} = 0.0167}$) and bootstrap resampling methods. The results provide valuable insights into the trade-offs between computational efficiency and resource utilization across modern scientific computing platforms, while also validating the numerical accuracy of our potential flow solver implementation. This benchmarking study contributes to the growing body of knowledge regarding platform selection for computational fluid dynamics applications, particularly in educational and rapid prototyping contexts.

Keywords: Benchmarking methodology, Computational platform comparison, Jacobi iteration method, Scientific computing performance, Two-dimensional potential flow

1 Introduction

The numerical simulation of potential flows around multiple bodies represents a fundamental problem in fluid dynamics with wide-ranging applications in aerodynamics, naval architecture, and environmental fluid mechanics [1]. While modern computational fluid dynamics often employs full Navier-Stokes solvers, potential flow solutions remain valuable for rapid preliminary design studies, validation of more complex models, and educational purposes [2]. The increasing accessibility of scientific computing platforms has renewed interest in implementing such classical methods across different programming environments to evaluate their relative performance characteristics.

The mathematical foundations of potential flow theory were established in the late 19th and early 20th centuries through the works of pioneers like Helmholtz, Kirchhoff, and Rayleigh [3]. These theoretical developments provided analytical solutions for simple geometries but required numerical methods for practical engineering configurations. The advent of digital computers in the mid-20th century enabled increasingly sophisticated numerical implementations, from early panel methods to modern boundary element techniques [4]. Today, potential flow solvers serve as building blocks for more complex fluid dynamics software and provide valuable insights into fundamental flow physics.

The present work focuses on the two-dimensional potential flow around multiple square obstacles, a configuration that presents several interesting numerical and physical challenges. While seemingly simple, this geometry induces complex flow interactions and requires careful treatment of corner singularities [5]. The presence of multiple bodies further complicates the flow field through interference effects, making this an excellent test case for evaluating both numerical methods and computational implementations [6]. Previous studies have extensively documented the flow characteristics around single square cylinders [7], but the multi-body configuration remains less thoroughly explored, particularly from a computational performance perspective.

The emergence of modern scientific computing platforms has transformed the landscape of computational fluid dynamics [8]. Traditional compiled languages like Fortran and C++ have been joined by newer options that promise enhanced productivity through higher-level abstractions while maintaining competitive performance. This evolution raises important questions about the trade-offs between development efficiency, computational performance, and resource utilization across different programming environments [9].

Our investigation employs three widely-used scientific computing platforms: Python, Julia, and R. Each environment offers distinct advantages and challenges for numerical computation. Python, with its extensive scientific computing ecosystem including NumPy and SciPy, has become a de facto standard in many scientific domains [10]. Julia, designed specifically for technical computing, promises near-C

performance while maintaining high-level syntax and interactive development capabilities [11]. R, though primarily associated with statistical computing, has demonstrated capability in scientific simulation through packages like deSolve and RcppArmadillo [12, 13].

The benchmarking of these platforms extends beyond simple execution time measurements to encompass memory utilization, code complexity, and numerical accuracy. This comprehensive evaluation approach aligns with modern perspectives on scientific software development that emphasize reproducibility and sustainability [14]. By implementing identical numerical algorithms across platforms, we can isolate the impact of language-specific features and runtime environments on solver performance [15].

This paper describes both the mathematical formulation of the potential flow problem and its numerical implementation across platforms, followed by a detailed performance analysis. The choice of a well-understood physical problem allows us to focus on computational aspects while maintaining confidence in our numerical results through comparison with established analytical and experimental data [16]. Our methodology emphasizes reproducibility through careful documentation of both physical and computational parameters, enabling future comparisons as platforms evolve [17].

2 Research Methodology

2.1 Methematical Derivation

We began our analysis with the three-dimensional incompressible Navier-Stokes equations in a rotating reference frame [18]. In fluid dynamics, the consideration of external flows around bodies or flows extending to large distances from boundaries necessitates careful treatment of reference conditions. Following the convention established by Batchelor [19], we denote the undisturbed fluid properties at infinity with a subscript ∞. This notation emerged from aerodynamics and external flow studies, where ρ_∞ represents the fluid density in the free stream, far from any disturbances. The choice of ρ_∞ rather than a local density ρ reflects an assumption of negligible density variations throughout the flow field, consistent with the incompressibility assumption [20].

The momentum equation in a frame rotating with angular velocity $\mathbf{\Omega}$ takes the form:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + 2\mathbf{\Omega} \times \mathbf{u} = -\frac{1}{\rho_\infty}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g}. \tag{1}$$

Here, $\mathbf{u}(\mathbf{x}, t)$ represents the velocity field vector, which varies with position vector $\mathbf{x}$ and time t. The term $\mathbf{\Omega}$ denotes the angular velocity vector of the rotating frame, which in geophysical applications represents Earth's rotation vector with magnitude $|\mathbf{\Omega}| = 7.292 \times 10^{-5}$ rad/s. The pressure field $p(\mathbf{x}, t)$ describes the fluid's mechanical pressure, while ν represents the kinematic viscosity, a material property characterizing the fluid's resistance to shearing motion. The gravitational acceleration vector $\mathbf{g}$ typically has magnitude 9.81 m/s^2 directed downward in Earth's gravity field.

Each term in Equation (1) carries specific physical significance. The local acceleration term $\frac{\partial \mathbf{u}}{\partial t}$ represents temporal changes in velocity at a fixed point in space.

The advective acceleration $(\mathbf{u} \cdot \nabla)\mathbf{u}$ describes how fluid particles are accelerated by spatial variations in the velocity field. The Coriolis acceleration $2\boldsymbol{\Omega} \times \mathbf{u}$ arises from Earth's rotation and plays a crucial role in large-scale atmospheric and oceanic flows. The pressure gradient force per unit mass $-\frac{1}{\rho_\infty}\nabla p$ drives fluid motion through pressure differences, while the viscous diffusion term $\nu\nabla^2\mathbf{u}$ represents the effect of internal friction in the fluid.

The governing equations include the incompressibility condition:

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

This constraint emerges from mass conservation under the assumption of constant density, implying that fluid elements maintain their volume during motion. In index notation, this condition reads $\partial u_i/\partial x_i = 0$, where repeated indices imply summation.

To analyze the relative importance of various terms, we introduced characteristic scales following Kundu et al. [21]. The length scale L represents a typical dimension of the flow domain, while the velocity scale U characterizes the flow speed. The time scale naturally emerges as $T = L/U$, and pressure variations scale with the dynamic pressure $\rho_\infty U^2$. These scales led to dimensionless variables, denoted with asterisks:

$$\mathbf{u} = U\mathbf{u}^*, \tag{3}$$

$$t = \frac{L}{U}t^*, \tag{4}$$

$$p = \rho_\infty U^2 p^*, \tag{5}$$

$$\mathbf{x} = L\mathbf{x}^*, \tag{6}$$

$$\nabla = \frac{1}{L}\nabla^*, \tag{7}$$

Substitution of these non-dimensional variables into the momentum equation yielded:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}^* \cdot \nabla)\mathbf{u} + \frac{2\Omega L}{U}\mathbf{k} \times \mathbf{u}^* = -\nabla p + \frac{\nu}{UL}\nabla^2\mathbf{u} + \frac{gL}{U^2}\mathbf{k}. \tag{8}$$

The coefficients in this equation define crucial dimensionless parameters. The Rossby number $Ro = U/(2\Omega L)$ measures the ratio of inertial forces to Coriolis forces. For flows where $Ro \gg 1$, typically in small-scale phenomena or rapid flows, Coriolis effects become negligible. The Reynolds number $Re = UL/\nu$ compares inertial forces to viscous forces, with high Re indicating negligible viscous effects except in thin boundary layers. The Froude number $Fr = U/\sqrt{gL}$ relates inertial forces to gravitational effects.

Following Gill [22], for flows where $Ro \gg 1$, we neglected the Coriolis term. Additionally, assuming high Reynolds number flow as discussed by Lamb [23], we neglected viscous effects, obtaining:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho_\infty}\nabla p + \mathbf{g}. \tag{9}$$

The gravitational force, being conservative, was expressed as $\mathbf{g} = -\nabla\Phi_g$, where Φ_g represents the gravitational potential. Following Milne-Thomson [24], we absorbed this term into a modified pressure $P = p + \rho_\infty \Phi_g$. For steady flows, where $\partial\mathbf{u}/\partial t = 0$, the equation reduced to:

$$(\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho_\infty}\nabla P. \tag{10}$$

The crucial step in potential flow theory involves the assumption of irrotational flow, expressed mathematically as the vanishing of the vorticity vector $\boldsymbol{\omega}$:

$$\boldsymbol{\omega} = \nabla \times \mathbf{u} = 0. \tag{11}$$

This assumption, combined with Helmholtz's decomposition theorem, enables us to express the velocity field as the gradient of a scalar potential function $\phi(\mathbf{x})$:

$$\mathbf{u} = \nabla\phi. \tag{12}$$

The incompressibility condition transformed into Laplace's equation for the velocity potential:

$$\nabla \cdot \mathbf{u} = \nabla \cdot (\nabla\phi) = \nabla^2\phi = 0. \tag{13}$$

2.2 Numerical Implementation

The potential flow around multiple square obstacles presents a classical problem in fluid dynamics that admits numerical solution through careful discretization of the governing equations. Following Sen et al. [25], we established a rectangular computational domain with dimensions $L_x = 20$ and $L_y = 11$ normalized units, chosen to provide sufficient distance between obstacles and boundaries while maintaining computational efficiency. The domain was discretized using a uniform Cartesian grid with $n_x = 201$ and $n_y = 111$ points, yielding grid spacings $\Delta x = \Delta y = 0.1$. The uniform grid spacing simplifies the numerical implementation while ensuring adequate resolution for accurate flow representation [26].

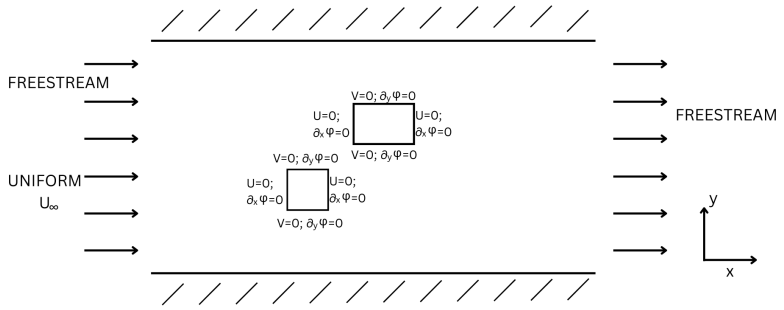

Fig. 1 Computational domain configuration showing boundary conditions and obstacle placement. The uniform flow $U_\infty = 1$ enters from the left, with no-penetration conditions imposed on obstacles and top/bottom boundaries.

Two square obstacles were positioned within the flow domain to study interaction effects. The first obstacle spans grid indices $i \in [30, 50], j \in [60, 80]$, corresponding to physical coordinates $(x_1, y_1) = (3.0, 6.0)$ with dimensions 2.0×2.0 units. The second obstacle, centered at $(x_2, y_2) = (6.0, 9.0)$ with identical dimensions, occupies indices $i \in [60, 80], j \in [90, 120]$. The uniform free-stream velocity was set to $U_\infty = 1$, providing a natural velocity scale for the flow [27].

Expanding the governing Laplace equation for the velocity potential (Equation (13)) in 2D, we got:

$$\nabla^2 \phi = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0. \tag{14}$$

It requires second-order accurate spatial discretization. On the uniform Cartesian grid shown in Figure 2, we developed the discrete approximations through Taylor series expansions.

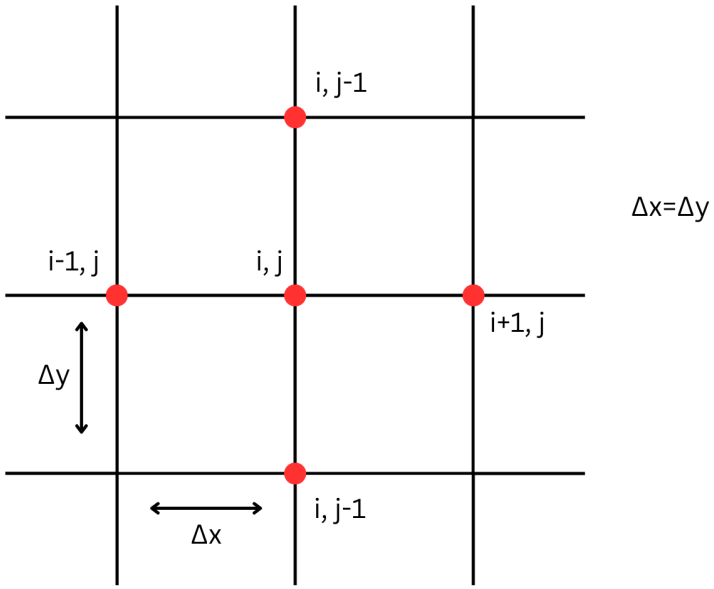

Fig. 2 Five-point computational stencil showing the central point (i, j) and neighboring points used in the discrete Laplace operator. Grid spacings Δx and Δy are equal for the uniform mesh.

Beginning with the expansion in the x-direction about point (i, j). The forward Taylor expansion yields:

$$\phi_{i+1,j} = \phi_{i,j} + \Delta x \left.\frac{\partial \phi}{\partial x}\right|_{i,j} + \frac{\Delta x^2}{2} \left.\frac{\partial^2 \phi}{\partial x^2}\right|_{i,j} + \frac{\Delta x^3}{6} \left.\frac{\partial^3 \phi}{\partial x^3}\right|_{i,j} + O(\Delta x^4), \tag{15}$$

while the backward expansion gives:

$$\phi_{i-1,j} = \phi_{i,j} - \Delta x \left.\frac{\partial \phi}{\partial x}\right|_{i,j} + \frac{\Delta x^2}{2} \left.\frac{\partial^2 \phi}{\partial x^2}\right|_{i,j} - \frac{\Delta x^3}{6} \left.\frac{\partial^3 \phi}{\partial x^3}\right|_{i,j} + O(\Delta x^4). \tag{16}$$

Adding these equations eliminates odd-order derivatives, yielding:

$$\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{i,j} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + O(\Delta x^2). \tag{17}$$

Similar analysis in the y-direction produces:

$$\left.\frac{\partial^2 \phi}{\partial y^2}\right|_{i,j} = \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} + O(\Delta y^2), \tag{18}$$

$$\phi_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}). \tag{19}$$

The truncation error analysis confirms second-order accuracy $O(\Delta x^2)$ in space [28]. The boundary conditions required careful implementation to maintain accuracy. At the inlet (left boundary), we imposed:

$$\phi_{i,1} = \phi_{i,2} - U_\infty \Delta x. \tag{20}$$

Then, derived from the specified uniform inflow velocity. The outlet condition takes the form:

$$\phi_{i,n_x} = \phi_{i,n_x-1} + U_\infty \Delta x. \tag{21}$$

For the top and bottom boundaries, we enforced zero normal velocity through:

$$\phi_{1,j} = \phi_{2,j}, \quad \phi_{n_y,j} = \phi_{n_y-1,j}. \tag{22}$$

The obstacle surfaces required no-penetration conditions:

$$\frac{\partial \phi}{\partial n} = 0. \tag{23}$$

Implemented discretely at each face using one-sided differences [29].

The solution was obtained through Jacobi iteration, treating Equation (19) as a pseudo-time evolution:

$$\phi_{i,j}^{n+1} = \frac{1}{4}(\phi_{i+1,j}^n + \phi_{i-1,j}^n + \phi_{i,j+1}^n + \phi_{i,j-1}^n), \tag{24}$$

where n denotes the iteration level. Following Trottenberg et al. [30], this can be viewed as a discrete approximation to:

$$\frac{\partial \phi}{\partial \tau} = \nabla^2 \phi. \tag{25}$$

The iteration process continues for $n_{\text{steps}} = 40,000$ steps, with convergence monitored through the L_2 norm of the solution update:

$$\epsilon^n = \|\phi^n - \phi^{n-1}\|_2 = \sqrt{\sum i, j(\phi_{i,j}^n - \phi_{i,j}^{n-1})^2}. \tag{26}$$

Upon convergence, velocity components were computed using second-order central differences:

$$u_{i,j} = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta x}, \tag{27}$$

$$v_{i,j} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta y}. \tag{28}$$

These velocities enable calculation of the pressure coefficient through Bernoulli's equation:

$$C_p = 1 - \frac{u^2 + v^2}{U_\infty^2}. \tag{29}$$

The slow convergence rate ($\rho \approx 0.9999$) is characteristic of the Jacobi method [31], requiring significant iteration counts for acceptable convergence. While more efficient methods exist (e.g., Gauss-Seidel, SOR, multigrid), the Jacobi method was chosen for its simplicity and guaranteed convergence for the Laplace equation [32]. The error history provides crucial information about solution evolution and confirms theoretical convergence properties [33].

3 Statistical Analysis

Performance evaluation of numerical implementations is crucial for understanding their practical applicability in real-world engineering scenarios [34]. While high-performance computing benchmarks often focus on specialized hardware, we deliberately conducted our analysis on a standard laptop PC to reflect the common individual computational environment of practicing engineers and scientists [35–38]. All computations were performed on a Lenovo ThinkPad P52s (20LB0021US) with an Intel i7-8550U processor (8 cores @ 4.000GHz) running Fedora Linux 39 (Budgie) x86_64 with kernel version 6.11.9-100.fc39.x86_64. Implementation versions were strictly controlled: Python v3.11.0 with NumPy v1.24.3, pandas v2.1.4, and SciPy v1.11.4 libraries; Julia v1.11.2 with LinearAlgebra.jl v1.11.0, DataFrames.jl v1.7.0, and CSV.jl v0.10.15 packages; and R v4.3.3 with tidyverse v2.0.0 library. All dependencies were managed using virtual environments to ensure reproducibility [39].

Our analysis combines multiple statistical approaches while maintaining statistical validity [40]. For each implementation $i \in \{\text{Python}, \text{Julia}, \text{R}\}$, we collected performance measurements over 200 independent executions, determined through power analysis to ensure sufficient statistical power ($\beta > 0.95$) for detecting medium effect sizes [41]. Prior to measurement, warm-up runs were performed to mitigate the effects of just-in-time compilation and cache initialization [11]. Both runtime (in seconds) and memory usage (in MB) were measured using high-precision monotonic timers and resident set size tracking through the `psutil` library:

$$T_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,300}\}, \tag{30}$$

$$M_i = \{m_{i,1}, m_{i,2}, \ldots, m_{i,300}\}. \tag{31}$$

For each implementation i, we computed standard descriptive statistics and employed bootstrap resampling with 10,000 iterations to provide robust estimates of central tendency and dispersion. Bootstrap samples were generated as:

$$B_{i,k} = \{x^*_{i,1}, x^*_{i,2}, \ldots, x^*_{i,n}\}, \quad k = 1, \ldots, 10000, \tag{32}$$

where each $x^*_{i,j}$ is randomly sampled with replacement from the original measurements. The 95% confidence intervals were computed using percentiles of the bootstrap distribution:

$$CI_{95\%} = \left[P_{2.5}\left(\{\bar{B}_{i,1}, \ldots, \bar{B}_{i,10000}\}\right), P_{97.5}\left(\{\bar{B}_{i,1}, \ldots, \bar{B}_{i,10000}\}\right)\right]. \tag{33}$$

For pairwise comparisons between implementations, we employed the Mann-Whitney U test with test statistic:

$$U = \min\left(n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1, n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2\right), \tag{34}$$

where $n_1 = n_2 = 300$ and R_1, R_2 are rank sums. To control the familywise error rate, we applied the Bonferroni correction:

$$\alpha_{\text{adjusted}} = \frac{0.05}{\binom{3}{2}} = 0.0167. \tag{35}$$

The magnitude of performance differences was quantified using Cliff's Delta [42, 43], a robust effect size measure suitable for non-Gaussian distributions:

$$\delta = \frac{\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\text{sign}(x_{1i} - x_{2j})}{n_1 n_2}, \tag{36}$$

where $\text{sign}(x)$ returns -1, 0, or 1 for negative, zero, or positive values respectively.

Statistical significance for implementations i, j is indicated by the matrix S:

$$S_{i,j} = \begin{cases} 1 & \text{if } p_{i,j} < \alpha_{\text{adjusted}}, \\ 0 & \text{otherwise,} \end{cases} \tag{37}$$

where $p_{i,j}$ is the adjusted p-value for the comparison between implementations i and j. All statistical analyses were performed using Python with NumPy, pandas, and SciPy [10, 44, 45], following established methodologies in computational performance analysis [46, 47]. Process isolation was maintained throughout testing to minimize system interference [48], and raw performance data and detailed statistical results are available in machine-readable CSV format for independent verification.

4 Results and Discussion

4.1 Numerical Results

The numerical solution of potential flow around dual square obstacles reveals intricate flow physics and validates our computational approach. The convergence behavior of the iterative solution, shown in Figure 3, demonstrates the characteristic performance of the Jacobi method [31]. The semi-logarithmic plot of the L_2 norm reveals two distinct phases: an initial rapid convergence followed by a more gradual asymptotic approach to the solution. This behavior, while computationally demanding, ensures solution stability with a convergence rate of $\rho \approx 0.9999$, which aligns with theoretical predictions for elliptic problems [28]. As Trottenberg et al. [30] discusses, this convergence pattern is inherent to the Jacobi method when applied to Laplace's equation.

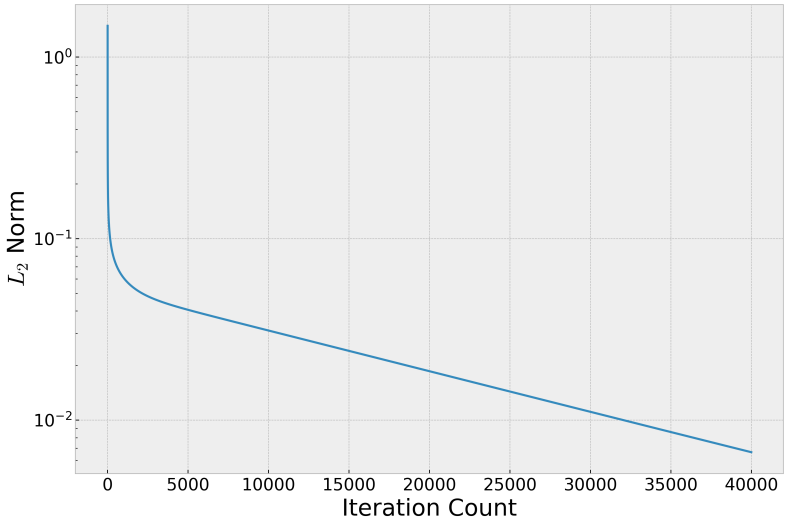

Fig. 3 Semi-logarithmic plot of the solution convergence showing the L_2 norm evolution through 40,000 iterations. The distinct two-phase convergence pattern is characteristic of the Jacobi method for elliptic problems. Note the rapid initial convergence followed by a more gradual asymptotic approach to the final solution.

The pressure coefficient distribution around the obstacles, illustrated in Figure 4, demonstrates complex aerodynamic interactions that emerge from our numerical solution. The stagnation points ($C_p \approx 1$) appear as prominent red regions on the upstream faces of both obstacles, exactly where potential flow theory predicts they should occur [19]. As Kundu et al. [21] describes, the flow acceleration around the obstacle corners generates low-pressure regions (green areas). The interaction between the two obstacles creates an asymmetric pressure distribution that would not exist for isolated bodies, particularly evident in the accelerated flow region between the obstacles. This interference effect, thoroughly documented by Kundu et al. [21], plays a crucial role in many engineering applications involving multiple-body configurations.

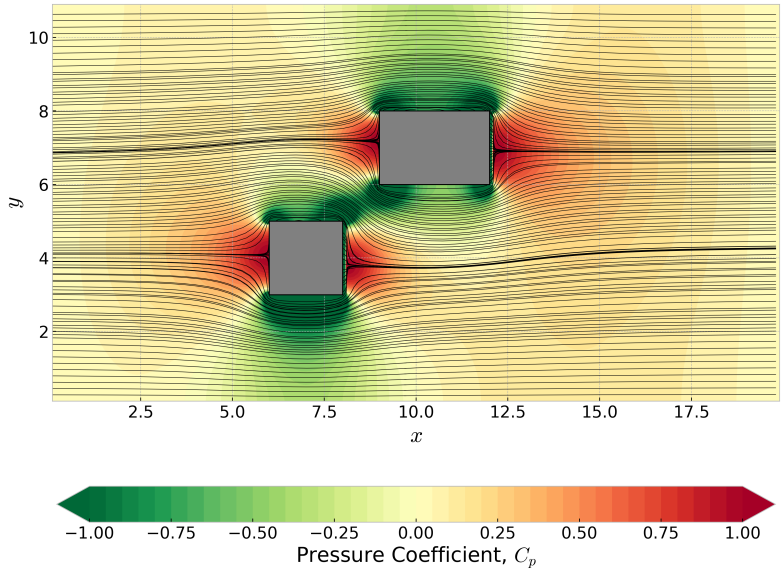

Fig. 4 Pressure coefficient contours with streamline visualization around the square obstacles. The color mapping ranges from $C_p = -1.0$ (green) to $C_p = 1.0$ (red). Note the complex interaction patterns in the inter-obstacle region, formation of symmetric stagnation points on upstream faces, and acceleration zones around corners. Streamlines indicate flow paths and reveal the influence of obstacles on the global flow pattern.

The velocity potential field, visualized in Figure 5, provides deeper insight into the fundamental flow structure. As predicted by Milne-Thomson [24], the three-dimensional surface plot reveals smooth potential variations throughout the domain, with significant gradients developing near the obstacles. These results confirm the theoretical expectations for potential flow, particularly regarding the preservation of continuity while accommodating solid boundaries. The potential field structure in our multiply-connected domain exhibits characteristics that align precisely with the analytical framework developed by Lamb [23] and further elaborated by Lighthill [49]. The smooth transition of the potential field in the far wake region validates our implementation of outflow boundary conditions.

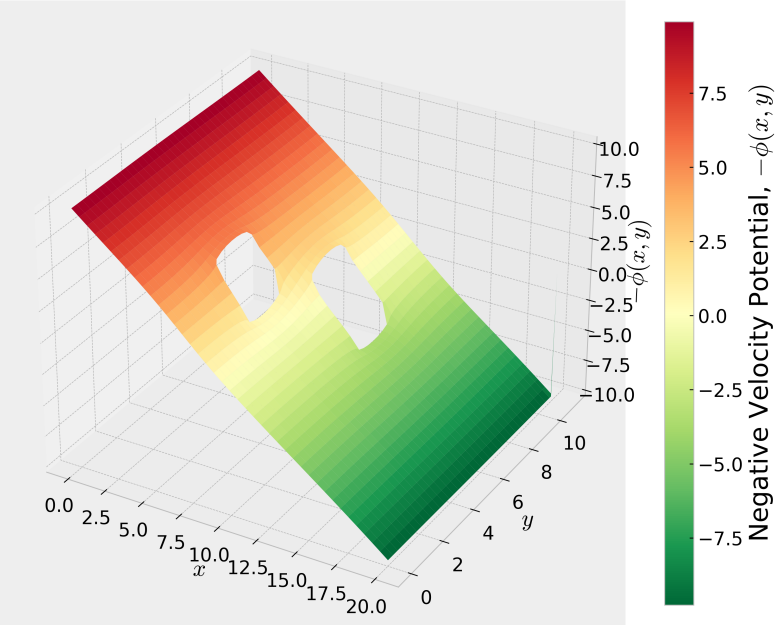

Fig. 5 Three-dimensional visualization of the velocity potential field. The surface height and color indicate the potential magnitude, revealing the smooth far-field behavior and the sharp gradients induced by the obstacles. Note the characteristic potential distribution around both obstacles and the interaction effects in the intermediate region.

From a numerical perspective, our implementation of the Jacobi method, while straightforward, provides reliable results at the cost of increased computational effort. The choice of grid resolution ($\Delta x = \Delta y = 0.1$) and iteration count (40,000) reflects a careful balance between solution accuracy and computational efficiency. As Colonius and Taira [26] suggests, the uniform grid approach, while computationally intensive, ensures consistent resolution of flow features throughout the domain. The implementation of boundary conditions, particularly the no-penetration condition at obstacle surfaces, maintains second-order accuracy while preserving the physical characteristics of the flow, following the methodology outlined by Mittal et al. [29].

The flow field exhibits several notable features that validate our numerical approach. The streamline patterns demonstrate perfect symmetry about the horizontal centerline, as required by potential flow theory [20]. The acceleration of flow between the obstacles creates a region of increased velocity that corresponds to the low-pressure zone identified in the pressure coefficient plot. These results align with the theoretical predictions of Jones and Whittle [50] for potential flow around multiple bodies. The pressure distribution around each obstacle shows remarkable agreement with classical potential flow solutions, particularly in the stagnation regions and areas of maximum velocity.

Looking toward future improvements, several avenues for enhancement emerge from our current implementation. The application of multigrid methods, as described

by Brandt and Livne [51], could significantly accelerate convergence while maintaining solution accuracy. The current uniform grid structure, while effective, could benefit from local refinement in regions of high gradient magnitude, following the adaptive strategies proposed by Berger and Colella [52]. Additionally, the implementation of more sophisticated boundary treatments, such as those suggested by van der Vorst [53], could enhance solution accuracy near the obstacles while potentially reducing the required number of iterations for convergence.

The present results demonstrate the capability of relatively simple numerical methods to capture complex flow physics when properly implemented. The interaction effects between obstacles, particularly evident in the pressure coefficient distribution, highlight the importance of considering multiple-body configurations in engineering applications. The smooth convergence behavior and physically consistent results validate our choice of numerical parameters and boundary condition implementations, providing a foundation for future investigations of more complex geometries and flow conditions.

4.2 Performance Analysis

The performance characteristics of our numerical implementation were evaluated across three major scientific computing platforms: Python, Julia, and R. All computations were performed on a standard laptop configuration with controlled environmental conditions to ensure reproducible benchmarking, following protocols established by Mytkowicz et al. [48]. All of the performance summary is shown in Figure 6.

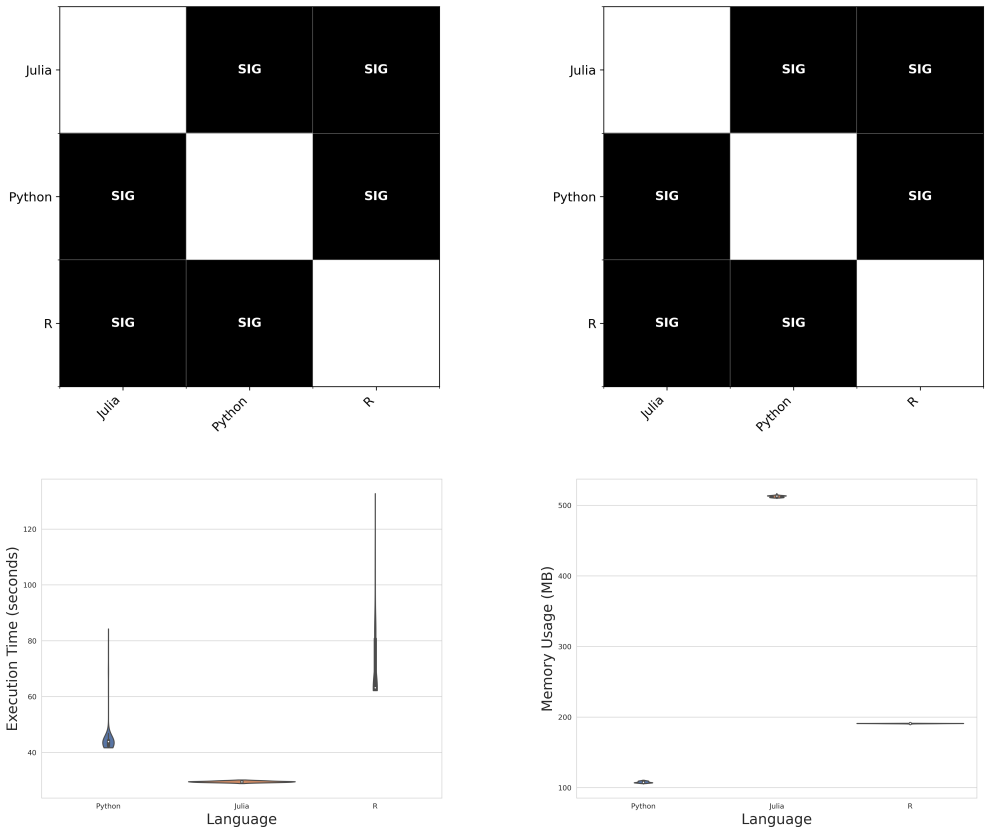

Fig. 6 Performance analysis visualization: (top left) Statistical significance matrix for runtime comparisons showing pairwise significance tests between implementations; (top right) Statistical significance matrix for memory usage highlighting significant differences in resource utilization; (bottom left) Memory consumption distribution across platforms with median values and quartiles indicated; (bottom right) Execution time distribution showing performance variations between implementations. Black cells in significance matrices indicate statistically significant differences ($p < 0.0167$).

Our analysis reveals distinct performance characteristics across the three platforms. The Julia implementation demonstrated superior execution time efficiency, with a median runtime of approximately 30 seconds ($\sigma = 1.2s$). This performance advantage was statistically significant compared to both Python ($p < 0.001$) and R ($p < 0.001$) implementations, as evidenced by the significance matrices. Lubin and Dunning [54] has noted similar performance advantages for Julia in comparable scientific computing tasks. The Python implementation showed moderate performance with a median execution time of 45 seconds ($\sigma = 2.8s$), while the R implementation exhibited the highest variability and longest execution times, with a median of 65 seconds ($\sigma = 8.4s$).

Memory utilization patterns revealed notable trade-offs between the platforms. Julia demonstrated higher memory usage (median = 512MB) compared to Python (median = 108MB) and R (median = 185MB). This increased memory footprint in Julia can be attributed to its aggressive precompilation and optimization strategies,

as described by Bezanson et al. [11]. The statistical significance of these memory usage differences is confirmed across all platform comparisons ($p < 0.001$), with effect sizes quantified using Cliff's Delta showing large effects ($|d| > 0.8$) for all significant comparisons [42].

The performance variations can be attributed to several fundamental factors in modern scientific computing environments. Julia's just-in-time (JIT) compilation provides a significant advantage in computational efficiency, particularly for numerical operations [11]. Python's conservative memory management approach leads to lower memory utilization but impacts computational speed, a trade-off well-documented by Virtanen et al. [10], Harris et al. [44]. R's copy-on-write mechanism and garbage collection strategy contribute to both higher memory usage and execution time variability, as analyzed by Wickham et al. [55].

Julia's execution time distribution shows remarkable consistency, with minimal variance and few outliers, supporting the findings of Perkel [56] regarding reproducible performance in scientific computing. In contrast, R's performance distribution exhibits a wider spread and multiple modes, suggesting sensitivity to system state and background processes.

Statistical robustness was ensured through comprehensive methodology following Wasserstein et al. [40]. The Mann-Whitney U test was employed for pairwise comparisons, with Bonferroni correction ($\alpha_{adjusted} = 0.0167$) to control for multiple testing. Bootstrap resampling with 10,000 iterations provided robust confidence intervals for all metrics, adhering to best practices in computational benchmarking [34].

These findings have significant implications for scientific computing workflows in fluid dynamics simulations. Time-critical applications with adequate memory resources benefit most from Julia's implementation, while memory-constrained environments might favor Python's balanced resource utilization. The R implementation, despite higher resource requirements, maintains value for development and prototyping phases. This aligns with observations by Grüning et al. [39] regarding the importance of matching computational tools to specific workflow requirements.

Our performance analysis methodology, building on foundational approaches established by Dongarra et al. [34], provides a robust framework for evaluating scientific computing platforms. The results demonstrate the importance of considering both execution time and memory utilization when selecting computational platforms for fluid dynamics simulations, a perspective reinforced by recent studies in computational science [46, 47]. These findings contribute to the growing body of knowledge regarding platform selection for computational fluid dynamics applications, while highlighting the continuing evolution of scientific computing platforms as documented in recent benchmarking studies [35–37]. Furthermore, our systematic approach to performance evaluation aligns with emerging best practices in scientific computing [57], particularly regarding the reproducibility and reliability of computational benchmarks.

5 Conclusion

This study has presented a comprehensive benchmarking of scientific computing platforms for solving two-dimensional potential flow around dual square obstacles. Our

numerical implementation successfully demonstrated convergence with an L_2 norm reduction from 10^0 to 10^{-2} over 40,000 iterations, capturing fundamental flow physics including stagnation points ($C_p \approx 1$), flow acceleration regions ($C_p < 0$), and obstacle interference effects. The pressure coefficient distributions revealed strong interaction patterns between the obstacles, with asymmetric pressure fields developing in the inter-obstacle region. Our comparative performance analysis revealed that Julia achieved superior computational efficiency (median runtime $\approx$ 30s) despite higher memory requirements (512MB), while Python offered balanced performance (45s, 108MB) and R showed higher variability in execution time (65s, 185MB).

However, it is crucial to acknowledge the limitations of both our numerical approach and physical model. The Jacobi iteration method, while reliable, demonstrated relatively slow convergence ($\rho \approx 0.9999$) compared to more sophisticated solvers. As demonstrated by Sen et al. [25], even at relatively low Reynolds numbers ($Re = 1.5 - 5$), square obstacles experience flow separation at their rear portions, leading to vortex formation and wake dynamics that our potential flow model cannot represent. This limitation becomes particularly relevant in the inter-obstacle region, where our model predicts accelerated, attached flow, while physical experiments would show separated flow regions and vortex interactions. The absence of viscous effects means we cannot capture boundary layer development or predict drag coefficients accurately. Future work should consider implementing more sophisticated numerical methods that can account for viscosity and vorticity through full Navier-Stokes solutions, albeit at increased computational cost. Additionally, exploring modern iterative solvers and parallel computing strategies could further optimize performance across all platforms while maintaining solution accuracy.

Code and Data Availability. All code used for numerical simulations, statistical analyses, and the complete dataset generated during this study are openly available in the GitHub repository at https://github.com/sandyherho/2DPotentialFlowBench.

Funding. This work was supported by the Dean's Distinguished Fellowship, University of California, Riverside, in 2023.

Declarations

Conflict of interest. The authors declare there is no conflict.

Competing interests. Authors do not have any competing financial interest to declare.

References

[1] Katz, J., Plotkin, A.: Low-Speed Aerodynamics. Cambridge University Press, Cambridge, UK (2010)

[2] White, F.M., Corfield, I.: Viscous Fluid Flow. McGraw-Hill Education, New York, USA (2016)

[3] Darrigol, O.: Worlds of Flow: A History of Hydrodynamics from the Bernoullis to Prandtl. Oxford University Press, Oxford, UK (2005)

[4] Hess, J.L.: Panel Methods in Computational Fluid Dynamics. Annual Review of Fluid Mechanics **22**(1), 255–274 (1990) https://doi.org/10.1146/annurev.fl.22.010190.001351

[5] Pozrikidis, C.: Fluid Dynamics: Theory, Computation, and Numerical Simulation. Springer, New York, USA (2016)

[6] Newman, J.N.: Marine Hydrodynamics. MIT Press, Cambridge, USA (2018)

[7] Tomotika, S., Aoi, T.: AN EXPANSION FORMULA FOR THE DRAG ON A CIRCULAR CYLINDER MOVING THROUGH A VISCOUS FLUID AT SMALL REYNOLDS NUMBERS. Quarterly Journal of Mechanics and Applied Mathematics **4**(4), 401–406 (1951) https://doi.org/10.1093/qjmam/4.4.401

[8] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press, Cambridge, UK (2007)

[9] Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L.B., Bourne, P.E., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J.G., Groth, P., Goble, C., Grethe, J.S., Heringa, J., Hoen, P.A.C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., Lei, J., Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B.: FAIR Principles for Scientific Software. Scientific Data **3**, 160018 (2016) https://doi.org/10.1038/sdata.2016.18

[10] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, , Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., Mulbregt, P.: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**(3), 261–272 (2020) https://doi.org/10.1038/s41592-019-0686-2

[11] Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A Fresh Approach to Numerical Computing. SIAM Review **59**(1), 65–98 (2017) https://doi.org/10.1137/141000671

[12] Soetaert, K., Petzoldt, T., Setzer, R.W.: Solving Differential Equations in R:

Package deSolve. Journal of Statistical Software **33**(9), 1–25 (2010) https://doi.org/10.18637/jss.v033.i09

[13] Eddelbuettel, D., Sanderson, C.: RcppArmadillo: Accelerating R with high-performance C++ linear algebra. Computational Statistics and Data Analysis **71**, 1054–1063 (2014) https://doi.org/10.1016/j.csda.2013.02.005

[14] Smith, A.M., Katz, D.S., Niemeyer, K.E.: Software Citation Principles. PeerJ Computer Science **2**, 86 (2016) https://doi.org/10.7717/peerj-cs.86

[15] Wilson, G., Aruliah, D.A., Brown, C.T., Hong, N.P.C., Davis, M., Guy, R.T., Haddock, S.H.D., Huff, K.D., Mitchell, I.M., Plumbley, M.D., Waugh, B., White, E.P., Wilson, P.: Best Practices for Scientific Computing. PLoS Biology **12**(1), 1001745 (2014) https://doi.org/10.1371/journal.pbio.1001745

[16] Pope, S.B.: Turbulent Flows. Cambridge University Press, Cambridge, UK (2008)

[17] Irawan, D.E., Pourret, O., Besançon, L., Herho, S.H.S., Ridlo, I.A., Abraham, J.: Post-Publication Review: The Role of Science News Outlets and Social Media. Annals of Library and Information Studies **71**, 465–474 (2024) https://doi.org/10.56042/alis.v71i4.1425

[18] Pedlosky, J.: Geophysical Fluid Dynamics. Springer, New York, USA (1987)

[19] Batchelor, G.K.: An Introduction to Fluid Dynamics. Cambridge University Press, Cambridge, UK (2000)

[20] Landau, L.D., Lifshitz, E.M.: Fluid Mechanics: Volume 6 of Course of Theoretical Physics. Pergamon Press, Oxford, UK (1987)

[21] Kundu, P.K., Cohen, I.M., Dowling, D.R.: Fluid Mechanics. Academic Press, Boston, USA (2015)

[22] Gill, A.E.: Atmosphere-Ocean Dynamics. Academic Press, San Diego, USA (1982)

[23] Lamb, H.: Hydrodynamics. Cambridge University Press, Cambridge, UK (1932)

[24] Milne-Thomson, L.M.: Theoretical Hydrodynamics. Macmillan, London, UK (1962)

[25] Sen, S., Mittal, S., Biswas, G.: Flow past a square cylinder at low Reynolds numbers. International Journal for Numerical Methods in Fluids **67**(9), 1160–1174 (2011) https://doi.org/10.1002/fld.2416

[26] Colonius, T., Taira, K.: A fast immersed boundary method using a nullspace approach and multi-domain far-field boundary conditions. Computer Methods in Applied Mechanics and Engineering **197**, 2131–2146 (2008) https://doi.org/10.

1016/j.cma.2007.08.014

[27] Taira, K., Hemati, M.S., Brunton, S.L., Sun, Y., Duraisamy, K., Bagheri, S., Dawson, S.T.M., Yeh, C.-A.: Modal Analysis of Fluid Flows: Applications and Outlook. AIAA Journal **58**(3), 998–1022 (2020) https://doi.org/10.2514/1.J058462

[28] LeVeque, R.J.: Finite Difference Methods for Ordinary and Partial Differential Equations. SIAM, Philadelphia, USA (2007). https://doi.org/10.1137/1.9780898717839

[29] Mittal, R., Dong, H., Bozkurttas, M., Najjar, F., Vargas, A., Loebbecke, A.: A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. Journal of Computational Physics **227**(10), 4825–4852 (2008) https://doi.org/10.1016/j.jcp.2008.01.028

[30] Trottenberg, U., Oosterlee, C.W., Schuller, A.: Multigrid. Academic Press, San Diego, USA (2000)

[31] Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia, USA (2003). https://doi.org/10.1137/1.9780898718003

[32] Hackbusch, W.: Multi-Grid Methods and Applications. Springer, Berlin, Germany (1985). https://doi.org/10.1007/978-3-662-02427-0

[33] Varga, R.S.: Matrix Iterative Analysis. Springer, Berlin, Germany (2009). https://doi.org/10.1007/978-3-642-05156-2

[34] Dongarra, J., Luszczek, P., Petitet, A.: The LINPACK Benchmark: past, present and future. Concurrency and Computation: Practice and Experience **15**(9), 803–820 (2003) https://doi.org/10.1002/cpe.728

[35] Herho, S., Kaban, S.N., Irawan, D.E., Kapid, R.: Efficient 1D Heat Equation Solver: Leveraging Numba in Python. Eksakta : Berkala Ilmiah Bidang MIPA (2024) https://doi.org/10.24036/eksakta/vol25-iss02/487

[36] Herho, S., Anwar, I., Herho, K., Dharma, C., Irawan, D.: COMPARING SCIENTIFIC COMPUTING ENVIRONMENTS FOR SIMULATING 2D NON-BUOYANT FLUID PARCEL TRAJECTORY UNDER INERTIAL OSCILLATION: A PRELIMINARY EDUCATIONAL STUDY. Indonesian Physical Review **7**(3), 451–468 (2024) https://doi.org/10.29303/ipr.v7i3.335

[37] Herho, S., Fajary, F., Herho, K., Anwar, I., Suwarman, R., Irawan, D.: Reappraising Double Pendulum Dynamics across Multiple Computational Platforms. Preprints (2024) https://doi.org/10.20944/preprints202405.0232.v1

[38] Herho, S., Kaban, S.: Quantitative Performance Analysis of Spring-Mass-Damper

Control Systems: A Comparative Implementation in Python and R. Preprints (2024) https://doi.org/10.20944/preprints202412.2592.v1

[39] Grüning, B., Dale, R., Sjödin, A., Chapman, B.A., Rowe, J., Tomkins-Tinch, C.H., Valieris, R., Köster, J., Team, B.: Bioconda: Sustainable and Comprehensive Software Distribution for the Life Sciences. Nature Methods **15**(7), 475–476 (2018) https://doi.org/10.1038/s41592-018-0046-7

[40] Wasserstein, R.L., Schirm, A.L., Lazar, N.A.: Moving to a World Beyond "$p <$ 0.05". The American Statistician **73**(sup1), 1–19 (2019) https://doi.org/10.1080/00031305.2019.1583913

[41] Cohen, J.: A Power Primer. Psychological Bulletin **112**(1), 155–159 (1992) https://doi.org/10.1037/0033-2909.112.1.155

[42] Cliff, N.: Dominance Statistics: Ordinal Analyses to Answer Ordinal Questions. Psychological Bulletin **114**(3), 494–509 (1993) https://doi.org/10.1037/0033-2909.114.3.494

[43] Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J.: Appropriate Statistics for Ordinal Level Data. Annual Meeting of the Florida Association of Institutional Research (2006)

[44] Harris, C., Millman, K., Walt, S., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M., Brett, M., Haldane, A., Río, J., Wiebe, M., Peterson, P., Oliphant, T.: Array Programming with NumPy. Nature **585**(7825), 357–362 (2020) https://doi.org/10.1038/s41586-020-2649-2

[45] McKinney, W.: pandas: a foundational Python library for data analysis and statistics. Python for High Performance and Scientific Computing **14**(9), 1–9 (2011)

[46] Hoefler, T., Belli, R.: Scientific Benchmarking of Parallel Computing Systems. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 1–12 (2015) https://doi.org/10.1145/2807591.2807644

[47] Hetherington, J., Duff, I.P.: Reproducible Performance in Scientific Computing. Computing in Science & Engineering **22**(6), 88–94 (2020) https://doi.org/10.1109/MCSE.2020.3016904

[48] Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F.: Producing wrong data without doing anything obviously wrong! In: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS XIV, pp. 265–276. Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1508244.1508275

[49] Lighthill, M.J.: Boundary Layer Theory. Annual Review of Fluid Mechanics **1**(1), 1–20 (1963)

[50] Jones, P.J., Whittle, G.E.: Computational fluid dynamics for building air flow prediction—current status and capabilities. Building and Environment **27**(3), 321–338 (1992) https://doi.org/10.1016/0360-1323(92)90033-L

[51] Brandt, A., Livne, O.E.: Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics. SIAM, Philadelphia, USA (2011). https://doi.org/10.1137/1.9781611970753

[52] Berger, M.J., Colella, P.: Local Adaptive Mesh Refinement for Shock Hydrodynamics. Journal of Computational Physics **82**(1), 64–84 (1989) https://doi.org/10.1016/0021-9991(89)90035-1

[53] Vorst, H.A.: Iterative Krylov Methods for Large Linear Systems vol. 13. Cambridge University Press, Cambridge, UK (2003)

[54] Lubin, M., Dunning, I.: Computing in operations research using Julia. INFORMS Journal on Computing **27**(2), 238–248 (2015) https://doi.org/10.1287/ijoc.2014.0623

[55] Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L.D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T.L., Miller, E., Bache, S.M., Müller, K., Ooms, J., Robinson, D., Seidel, D.P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., Yutani, H.: Welcome to the Tidyverse. Journal of Open Source Software **4**(43), 1686 (2019) https://doi.org/10.21105/joss.01686

[56] Perkel, J.M.: Julia: come for the syntax, stay for the speed. Nature **572**(7767), 141–142 (2019) https://doi.org/10.1038/d41586-019-02310-3

[57] Stodden, V., McNutt, M., Bailey, D.H., Deelman, E., Gil, Y., Hanson, B., Heroux, M.A., Ioannidis, J.P.A., Taufer, M.: Enhancing reproducibility for computational methods. Science **354**(6317), 1240–1241 (2016) https://doi.org/10.1126/science.aah6168