

# Avaliação de Desempenho da Equação do Calor: Uma Análise Comparativa entre Python e Julia

ENZO LISBÔA PEIXOTO, Universidade Federal do Rio Grande do Sul, Brasil

NATHAN MATTES, Universidade Federal do Rio Grande do Sul, Brasil

PEDRO SCHOLZ SOARES, Universidade Federal do Rio Grande do Sul, Brasil

Este trabalho apresenta uma análise de desempenho comparativa entre as linguagens Python e Julia na resolução numérica da Equação do Calor utilizando o Método das Diferenças Finitas em 1D, 2D e 3D. O estudo foi conduzido em um ambiente controlado utilizando contêineres Docker e isolamento de recursos de hardware (CPU Pinning) para garantir a reprodutibilidade. Os resultados demonstram a superioridade de desempenho da linguagem Julia em tarefas intensivas de CPU, com speedups significativos em relação à implementação pura em Python, embora com um consumo de memória superior. A análise estatística, incluindo regressão linear e gráficos de estabilidade, valida o comportamento assintótico esperado dos algoritmos e destaca a importância do controle de ruído do sistema operacional em benchmarks de computação.

CCS Concepts: • **General and reference** → **Measurement; Empirical studies.**

Additional Key Words and Phrases: Equação do Calor, Diferenças Finitas, Python, Julia, Benchmarking, Docker

## ACM Reference Format:

Enzo Lisbôa Peixoto, Nathan Mattes, and Pedro Scholz Soares. 2025. Avaliação de Desempenho da Equação do Calor: Uma Análise Comparativa entre Python e Julia. *J. ACM* 1, 1, Article 1 (November 2025), 6 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 Introdução

A simulação numérica de fenômenos físicos, como a difusão de calor, é uma tarefa computacionalmente intensiva que exige eficiência tanto do hardware quanto do software. Historicamente, linguagens compiladas como C e Fortran dominam este cenário. No entanto, linguagens de alto nível como Python e Julia ganharam popularidade pela facilidade de desenvolvimento. Este trabalho investiga o trade-off entre facilidade de uso e desempenho, comparando implementações iterativas da Equação do Calor nas duas linguagens.

## 2 Metodologia

### 2.1 O Problema Computacional

A Equação do Calor foi discretizada utilizando o Método das Diferenças Finitas (FDM) com esquema explícito no tempo. Foram implementados três dimensões:

- **1D:** Complexidade  $O(L)$ , onde  $L$  é o tamanho do domínio linear.

---

Authors' Contact Information: Enzo Lisbôa Peixoto, [elpeixoto@inf.ufrgs.br](mailto:elpeixoto@inf.ufrgs.br), Universidade Federal do Rio Grande do Sul, Porto Alegre, Rio Grande do Sul, Brasil; Nathan Mattes, [nmattes@inf.ufrgs.br](mailto:nmattes@inf.ufrgs.br), Universidade Federal do Rio Grande do Sul, Porto Alegre, Rio Grande do Sul, Brasil; Pedro Scholz Soares, [pssoares@inf.ufrgs.br](mailto:pssoares@inf.ufrgs.br), Universidade Federal do Rio Grande do Sul, Porto Alegre, Rio Grande do Sul, Brasil.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

- **2D:** Complexidade  $O(L^2)$ , representando uma malha quadrada.
- **3D:** Complexidade  $O(L^3)$ , representando um volume cúbico.

## 2.2 Design Experimental

Foi utilizado um projeto fatorial completo considerando os fatores:

- **Linguagem:** Python vs. Julia.
- **Dimensão:** 1D, 2D, 3D.
- **Tamanho do Problema ( $L$ ):** Três níveis (Low, Mid, High) ajustados para cada dimensão para manter tempos de execução mensuráveis.

Cada configuração foi executada 30 vezes ('REPLICACOES = 30') para garantir significância estatística e permitir a análise de variabilidade.

## 3 Ambiente e Ferramentas

### 3.1 Hardware e Controle de Ruído

Para minimizar a interferência do Sistema Operacional e garantir resultados determinísticos, foram aplicadas as seguintes técnicas de controle de ambiente:

- **CPU Pinning:** Os experimentos foram isolados em um núcleo físico específico ('-cpuset-cpus="0"') via Docker.
- **Frequency Governor:** O governador da CPU foi fixado em performance para evitar oscilações de \*clock\*.
- **Turbo Boost:** Desativado para mitigar variações térmicas (\*thermal throttling\*).
- **Inibição de Sistema:** Utilização de systemd-inhibit para prevenir suspensão ou hibernação durante a execução.

### 3.2 Software e Automação

O fluxo de trabalho foi totalmente automatizado para garantir reprodutibilidade ("One script to rule them all\*"):

- **Docker:** Contêineres isolados garantiram que as versões das linguagens e bibliotecas fossem idênticas em todas as execuções.
- **Orquestração:** Um makefile que gerenciou a construção da imagem, a geração do plano de execução aleatorizado e a coleta de dados.
- **Análise:** Scripts em R foram utilizados para processar os logs brutos e gerar visualizações estatísticas.

## 4 Resultados Finais

### 4.1 Tempo de Execução

Os resultados indicam uma diferença expressiva de desempenho. A Figura ?? apresenta a distribuição do tempo de execução para as três dimensões.

Os experimentos demonstram que o Julia oferece tempos de resposta substancialmente menores que o Python para a mesma lógica algorítmica. A validação estatística apresentada na Tabela 1 confirma a consistência dos dados: o alto valor de  $R^2$  em todos os casos indica que a variação do tempo é explicada quase integralmente pelo aumento do tamanho do problema ( $L$ ), seguindo a complexidade polinomial teórica.

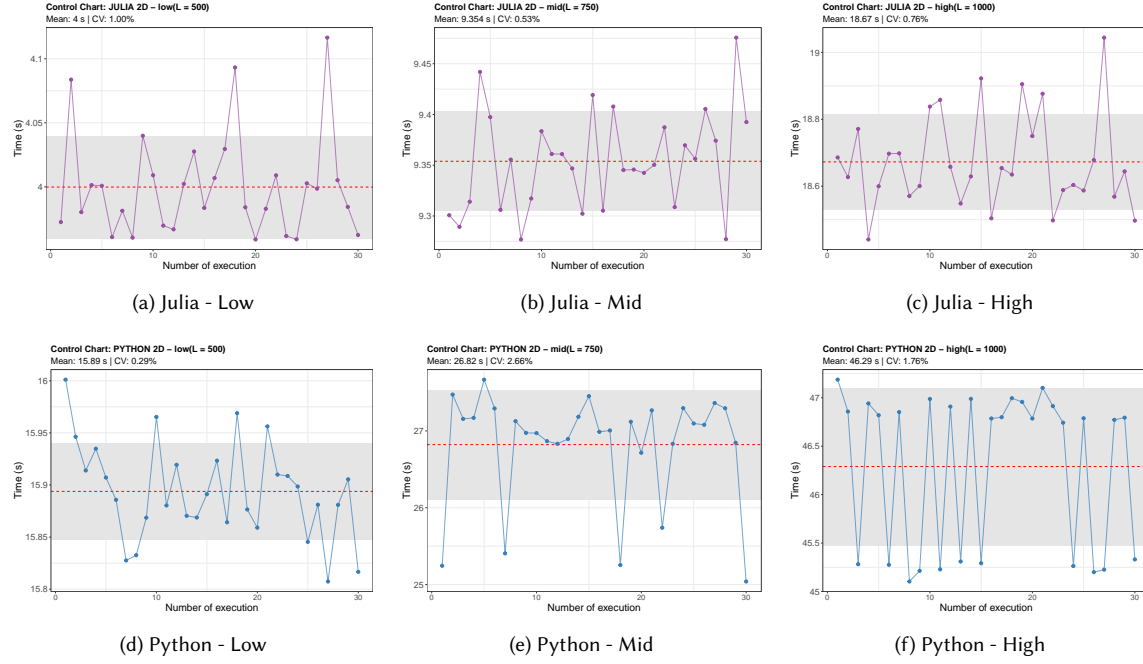


Fig. 1. Gráficos de Estabilidade para a dimensão 2D. A linha superior apresenta os resultados para Julia e a inferior para Python. Embora apresentados apenas os dados de 2D, o comportamento de estabilidade mostram o controle do experimento e a diferença da ordem de grandeza do tempo entre Julia e python. Graficos similares para 1d e 2d podem ser encontrados nos anexos.

Tabela 1. Coeficiente de Determinação ( $R^2$ ) para Regressão Linear do Tempo de Execução.

Linguagem	Dimensão	$R^2$	$R^2_{adj}$
Python	1D	0.9631	0.9627
Python	2D	0.9720	0.9717
Python	3D	0.9773	0.9771
Julia	1D	0.9924	0.9923
Julia	2D	0.9760	0.9758
Julia	3D	0.9675	0.9671

## 4.2 Consumo de Memória

A análise de memória revela o custo do desempenho. A Figura 3 e a Tabela ?? mostram que o Julia consome consistentemente mais memória RAM que o Python.

A análise do consumo de memória revela um claro trade-off entre tempo e espaço. Enquanto o Julia oferece um desempenho temporal (velocidade) muito superior, ele exige uma alocação de memória RAM significativamente maior que o Python em todas as dimensões testadas.

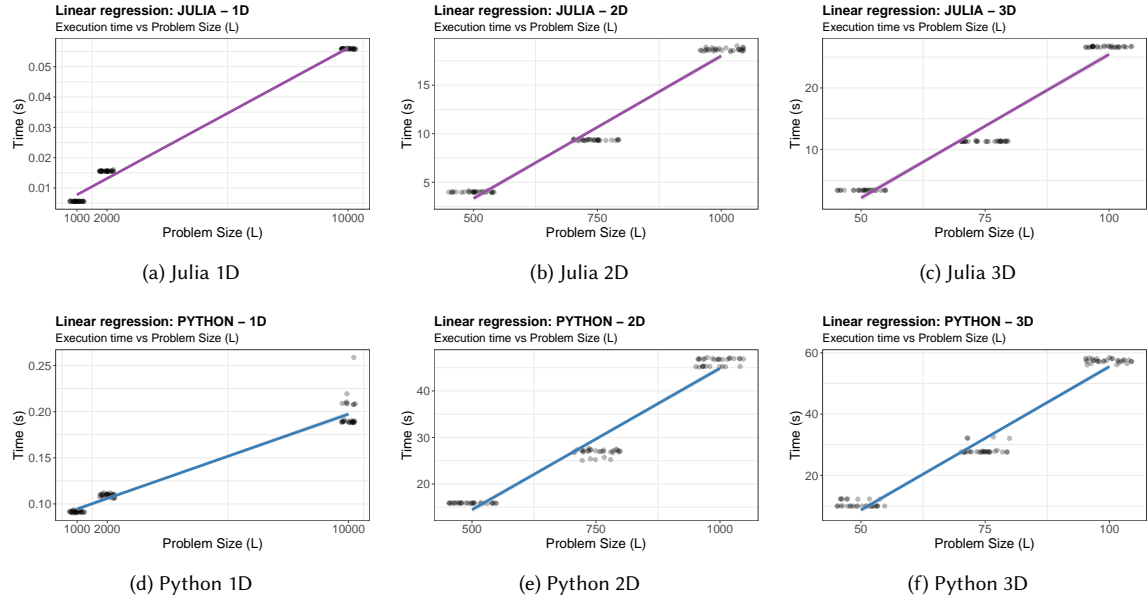


Fig. 2. Regressão Linear do Tempo de Execução ( $t_{exec}$ ) versus Tamanho do Problema ( $L$ ).%

Tabela 2. Regressão Linear: Pico de Memória

Linguagem	Dimensão	$R^2$	$R^2_{adj}$
Python	1D	<b>1.0000</b>	1.0000
Python	2D	<b>0.9908</b>	0.9907
Python	3D	<b>0.9661</b>	0.9657
Julia	1D	<b>1.0000</b>	1.0000
Julia	2D	<b>0.9908</b>	0.9907
Julia	3D	<b>0.9667</b>	0.9663

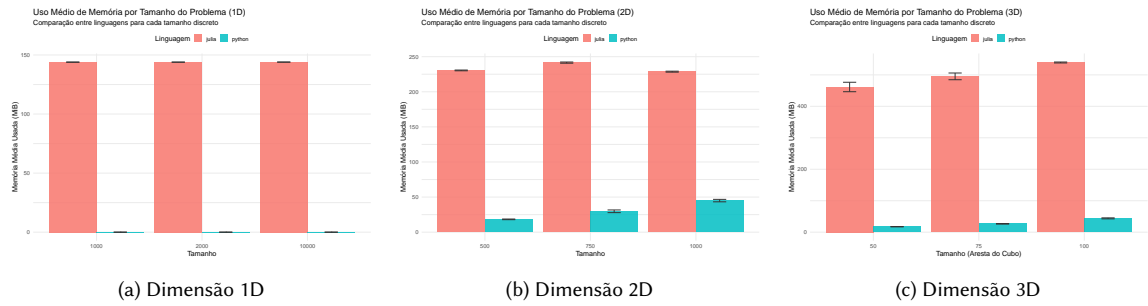


Fig. 3. Comparação do consumo de memória entre Python e Julia. Observa-se que, em todas as dimensões, a alocação de memória do Julia é superior, refletindo o custo do JIT e da gestão de tipos.

## 5 Discussão

### 5.1 Compilação JIT vs Interpretação

A vantagem massiva do Julia deve-se à sua compilação *Just-In-Time* (JIT) baseada em LLVM. Ao encontrar um laço for tipado, o Julia gera instruções de máquina otimizadas, similares a C/C++. O Python, por sua vez, realiza a verificação de tipos ('type checking') e o dispatch de métodos dinamicamente a cada iteração do laço, o que introduz um overhead massivo em algoritmos numéricos iterativos ( $O(N)$  verificações).

### 5.2 Trade-off Memória-Velocidade

Confirmou-se que não existe opção perfeita. O desempenho superior do Julia exige uma alocação de memória mais agressiva para manter as estruturas de dados e o código compilado. Para sistemas com restrição severa de memória, uma implementação Python otimizada (via C-extensions) poderia ser preferível, apesar da complexidade de desenvolvimento.

## 6 Conclusão

Este estudo demonstrou que, para computação científica baseada em simulações iterativas, a linguagem Julia oferece uma combinação superior de produtividade (sintaxe de alto nível) e desempenho (velocidade de execução), superando Python.

Do ponto de vista metodológico, o trabalho evidenciou que a medição precisa de desempenho em sistemas modernos exige um controle rigoroso do ambiente. O isolamento de processos via Docker e o controle de frequência da CPU foram fundamentais para reduzir o ruído experimental e obter dados consistentes e homoscedásticos.

Recomenda-se para trabalhos futuros a exploração de paralelismo (multithreading e GPU) para avaliar se a lacuna de desempenho pode ser reduzida.

## Referências

- [1] Rafal Ablamowicz and Bertfried Fauser. 2007. *CLIFFORD: a Maple 11 Package for Clifford Algebra Computations, version 11*. Retrieved February 28, 2008 from <http://math.tntech.edu/rafal/cliff11/index.html>
- [2] Patricia S. Abril and Robert Plant. 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan. 2007), 36–44. doi:10.1145/1188913.1188915
- [3] Sten Andler. 1979. Predicate Path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL '79)*. ACM Press, New York, NY, 226–236. doi:10.1145/567752.567774
- [4] David A. Anisi. 2003. *Optimal Motion Control of a Ground Vehicle*. Master's thesis. Royal Institute of Technology (KTH), Stockholm, Sweden.
- [5] Sam Anzaroot and Andrew McCallum. 2013. *UMass Citation Field Extraction Dataset*. Retrieved May 27, 2019 from <http://www.iesl.cs.umass.edu/data/data-umasscitationfield>
- [6] Sam Anzaroot, Alexandre Passos, David Belanger, and Andrew McCallum. 2014. *Learning Soft Linear Constraints with Application to Citation Field Extraction*. arXiv:1403.1349 doi:10.48550/arXiv.1403.1349
- [7] R. Baggett, M. Simecek, C. Chambellan, K. Tsui, and M. Fraune. 2025. Fluidity in the Phased Framework of Technology Acceptance: Case Study to Gain a Holistic Understanding of (Older Adult) Participant Advancement Through Acceptance Phases with Mobile Telepresence Robots. *Robotics Aut. Systems*. Manuscript submitted for review.
- [8] Lutz Bornmann, K. Brad Wray, and Robin Haunschild. 2019. *Citation concept analysis (CCA)—A new form of citation analysis revealing the usefulness of concepts for other researchers illustrated by two exemplary case studies including classic books by Thomas S. Kuhn and Karl R. Popper*. arXiv:1905.12410 [cs.DL]
- [9] Kenneth L. Clarkson. 1985. *Algorithms for Closest-Point Problems (Computational Geometry)*. Ph. D. Dissertation. Stanford University, Palo Alto, CA. UMI Order Number: AAT 8506171.
- [10] Jacques Cohen (Ed.). 1996. Special issue: Digital Libraries. *Commun. ACM* 39, 11 (Nov. 1996).
- [11] Sarah Cohen, Werner Nutt, and Yehoshua Sagie. 2007. Deciding equivalences among conjunctive aggregate queries. *J. ACM* 54, 2, Article 5 (April 2007), 50 pages. doi:10.1145/1219092.1219093

- [12] Bruce P. Douglass, David Harel, and Mark B. Trakhtenbrot. 1998. Statecharts in use: structured analysis and object-orientation. In *Lectures on Embedded Systems*, Grzegorz Rozenberg and Frits W. Vaandrager (Eds.). Lecture Notes in Computer Science, Vol. 1494. Springer-Verlag, London, 368–394. doi:10.1007/3-540-65193-4\_29
- [13] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago, Chapter The title of the chapter, 127–238. doi:10.1007/3-540-09237-4
- [14] Ian Editor (Ed.). 2008. *The title of book two* (2nd. ed.). University of Chicago Press, Chicago, Chapter 100, 25–137. doi:10.1007/3-540-09237-4
- [15] Matthew Van Gundy, Davide Balzarotti, and Giovanni Vigna. 2007. Catch me, if you can: Evading network signatures with web-based polymorphic worms. In *Proceedings of the first USENIX workshop on Offensive Technologies (WOOT '07)*. USENIX Association, Berkley, CA, Article 7, 9 pages.
- [16] Torben Hagerup, Kurt Mehlhorn, and J. Ian Munro. 1993. Maintaining Discrete Probability Distributions Optimally. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science, Vol. 700)*. Springer-Verlag, Berlin, 253–264.
- [17] David Harel. 1978. *LOGICS of Programs: AXIOMATICS and DESCRIPTIVE POWER*. MIT Research Lab Technical Report TR-200. Massachusetts Institute of Technology, Cambridge, MA.
- [18] David Harel. 1979. *First-Order Dynamic Logic*. Lecture Notes in Computer Science, Vol. 68. Springer-Verlag, New York, NY. doi:10.1007/3-540-09237-4
- [19] Lars Hörmander. 1985. *The analysis of linear partial differential operators. III*. Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. viii+525 pages. Pseudodifferential operators.
- [20] Lars Hörmander. 1985. *The analysis of linear partial differential operators. IV*. Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. vii+352 pages. Fourier integral operators.
- [21] Markus Kirschmer and John Voight. 2010. Algorithmic Enumeration of Ideal Classes for Quaternion Orders. *SIAM J. Comput.* 39, 5 (Jan. 2010), 1714–1747. doi:10.1137/080734467
- [22] Donald E. Knuth. 1997. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms (3rd. ed.)*. Addison Wesley Longman Publishing Co., Inc., Boston.
- [23] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY.
- [24] Barack Obama. 2008. A more perfect union. Video. Retrieved March 21, 2008 from <http://video.google.com/videoplay?docid=6528042696351994555>
- [25] Poker-Edge.Com. 2006. Stats and Analysis. Retrieved June 7, 2006 from <http://www.poker-edge.com/stats.php>
- [26] R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. <https://www.R-project.org/>
- [27] Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. 2010. A library-based synthesis methodology for reversible logic. *Microelectron. J.* 41, 4 (April 2010), 185–194.
- [28] Mehdi Saeedi, Morteza Saheb Zamani, Mehdi Sedighi, and Zahra Sasanian. 2010. Synthesis of Reversible Circuit Using Cycle-Based Approach. *J. Emerg. Technol. Comput. Syst.* 6, 4 (Dec. 2010), 12 pages.
- [29] Joseph Scientist. 2009. The fountain of youth. Patent No. 12345, Filed July 1st., 2008, Issued Aug. 9th., 2009.
- [30] Stan W. Smith. 2010. An experiment in bibliographic mark-up: Parsing metadata for XML export. In *Proceedings of the 3rd. annual workshop on Librarians and Computers (LAC '10, Vol. 3)*, Reginald N. Smythe and Alexander Noble (Eds.). Paparazzi Press, Milan Italy, 422–431.
- [31] Asad Z. Spector. 1990. Achieving application requirements. In *Distributed Systems* (2nd. ed.), Sape Mullender (Ed.). ACM Press, New York, NY, 19–33. doi:10.1145/90417.90738
- [32] Harry Thornburg. 2001. *Introduction to Bayesian Statistics*. Retrieved March 2, 2005 from <http://ccrma.stanford.edu/~jos/bayes/bayes.html>, archived at [<https://web.archive.org/web/20240505055615/https://ccrma.stanford.edu/~jos/bayes/bayes.html>]
- [33] TUG. 2017. *Institutional members of the T<sub>E</sub>X Users Group*. Retrieved May 27, 2017 from <http://www.tug.org/instmemb.html>
- [34] Boris Veytsman. 2017. *acmart—Class for typesetting publications of ACM*. Retrieved May 27, 2017 from <http://www.ctan.org/pkg/acmart>