Proceeding Series of the Brazilian Society of Computational and Applied Mathematics, v. 12, n. 1, 2026.

Trabalho apresentado no XLIV CNMAC, Fundação Getulio Vargas - Rio de Janeiro - RJ, 2025.

Proceeding Series of the Brazilian Society of Computational and Applied Mathematics

Soluções Numéricas da Equação de Poisson: Eficiência do Método de Jacobi com Devito e NumPy

Gabriel de O. Portes[1], Thiago B. Ikeda[2], Gilcilene S. de Paulo[3]
FCT/UNESP, Presidente Prudente, SP

Neste trabalho, analisamos as soluções numéricas da equação de Poisson por meio do método iterativo clássico de Jacobi, explorando duas abordagens distintas de implementação computacional: uma implementação manual em NumPy [2], com controle direto dos *loops* e esquemas numéricos, sem otimizações avançadas; e uma implementação simbólica utilizando o Devito [4], que gera código otimizado automaticamente a partir de expressões diferenciais definidas com o SymPy e que também utiliza NumPy internamente para operações auxiliares.

Consideramos o problema (1) definido em $\Omega = [-1, 1] \times [-1, 1]$:

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2, & \text{em } (x, y) \in \Omega, \\ u(x, y) = 0, & \text{para } (x, y) \in \partial\Omega. \end{cases} \tag{1}$$

A discretização do domínio considera espaçamentos uniformes em x e y da seguinte forma: $x_i = x_0 + ih$ e $y_j = y_0 + jh$, $i, j = 1, \ldots, N-1$, tal que h é o espaçamento espacial e N é a quantidade de subintervalos em ambas as direções, sendo x_0, y_0, x_N, $y_N \in \partial\Omega$.

O problema (1) foi aproximado pela equação discreta (2) que representa o método iterativo de Jacobi [1, 3], implementado de duas formas: manualmente com NumPy e por meio da geração de código otimizado fornecida pelo Devito.

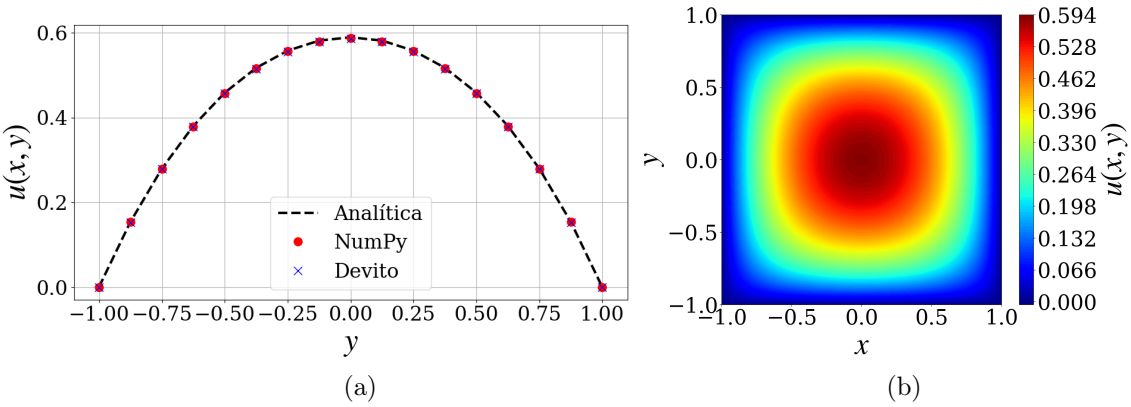
$$U_{i,j}^{(k+1)} = \frac{1}{4}(U_{i-1,j}^{(k)} + U_{i+1,j}^{(k)} + U_{i,j-1}^{(k)} + U_{i,j+1}^{(k)}) + \frac{h^2}{2}, \tag{2}$$

tal que $U_{i,j} \approx u(x_i, y_j)$ e o índice k representa a iteração.

O critério de parada $\left(\sum_{i,j=1}^{N-1} \left(U_{i,j}^{(k+1)} - U_{i,j}^{(k)}\right)^2\right)^{\frac{1}{2}} < 10^{-6}$ foi adotado para obtenção das soluções numéricas e estas foram verificadas pela comparação com a solução analítica do problema (1): $u(x, y) = 1 - y^2 - \frac{32}{\pi^3} \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)^3} senh\left(\frac{(2n+1)\pi}{2}\right) cosh\left(\frac{(2n+1)\pi x}{2}\right) cos\left(\frac{(2n+1)\pi y}{2}\right)$, em que consideramos 100 termos da série. A Figura 1 (a) traz uma análise qualitativa dos resultados numéricos obtidos pelo Devito e NumPy, com $N = 65$, $x = 0$ e $-1 \leq y \leq 1$. A Figura 1 (b) apresenta o gráfico de cores da solução analítica. Os gráficos de cores das soluções numéricas não foram incluídos, pois ficaram muito semelhantes.

Com o objetivo de avaliarmos a eficiência do método numérico a partir das distintas formas de implementação, realizamos uma análise comparativa através do refinamento de malha. A Tabela 1 apresenta a acurácia de cada solução numérica considerando o erro relativo na norma de Frobenius, além do desempenho das duas bibliotecas em termos de iterações e tempo de execução. Todas as simulações foram realizadas na plataforma Google Colaboratory.

[1] gabriel.portes@unesp.br
[2] t.ikeda@unesp.br
[3] gilcilene.sanchez@unesp.br



(a) (b)

Figura 1: (a) Comparação da solução analítica com soluções numéricas obtidas por NumPy e Devito, com $N = 65$, $x = 0$ e $-1 \leq y \leq 1$, e (b) solução analítica com 100 termos da série. Fonte: dos autores.

Tabela 1: Acurácia das soluções numéricas por NumPy e Devito e os seus respectivos desempenhos.

N	NumPy			Devito		
	Iterações	Erro	Tempo	Iterações	Erro	Tempo
5	37	$4.76 \cdot 10^{-2}$	0.623 ms	20	$4.85 \cdot 10^{-2}$	24.8 ms
9	155	$1.28 \cdot 10^{-2}$	12.724 ms	83	$1.41 \cdot 10^{-2}$	30.1 ms
17	594	$3.27 \cdot 10^{-3}$	0.212 s	327	$4.94 \cdot 10^{-3}$	30.23 ms
33	2244	$8.39 \cdot 10^{-4}$	3.235 s	1261	$3.05 \cdot 10^{-3}$	35.14 ms
65	8413	$2.44 \cdot 10^{-4}$	49.042 s	4649	$3.86 \cdot 10^{-3}$	43.84 ms
129	31361	$1.29 \cdot 10^{-4}$	25.82 min	16572	$6.81 \cdot 10^{-3}$	0.163 s
257	–	–	> 4 h	58883	$1.67 \cdot 10^{-2}$	1.12 s

As soluções numéricas obtidas pelo Devito apresentaram erros relativos ligeiramente maiores. Contudo, o número de iterações e o tempo de processamento foram consideravelmente menores, especialmente para malhas mais finas. Isso evidencia a eficiência da geração de código otimizado e as vantagens de ferramentas como o Devito, que dispensam a necessidade de otimização manual.

Agradecimentos

À FAPESP, Processo n.º 2024/03241-3, e à CAPES, Processo n.º 88887.993946/2024-00.

Referências

[1] J. A. Cuminato e M. Meneguette Jr. **Discretização de Equações Diferencias Parciais: Técnica de Diferenças Finitas**. 1a. ed. Rio de Janeiro: SBM, 2013. ISBN: 9788583370055.

[2] C. R. Harris et al. "Array programming with NumPy". Em: **Nature** 585.7825 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[3] R. J. Leveque. **Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-state and Time-dependent Problems**. 2a. ed. Filadélfia, EUA: Society for Industrial e Applied Mathematics, 2007. ISBN: 9780898716290.

[4] F. Luporini et al. "Architecture and Performance of Devito, a System for Automated Stencil Computation". Em: **ACM Trans. Math. Softw.** 46.1 (2020). DOI: 10.1145/3374916.