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

Comparative Analysis of Scaling Techniques for AIM Matrices in Reservoir Flow Equations

Pedro C. da S. Lara[1]
CEFET-RJ, Petrópolis, RJ

José R. P. Rodrigues[2]
CENPES, Petrobras, Rio de Janeiro, RJ

Cesar A. C. Perez[3]
CMG, Calgary, Canada

Luiz M. Carvalho[4]
UERJ, Rio de Janeiro, RJ

Paulo Goldfeld[5]
UFRJ, Rio de Janeiro, RJ

Abstract. Scaling improves the performance of iterative solvers for large, sparse linear systems by enhancing conditioning and balancing row and column magnitudes. This work highlights a tailored scaling approach for linear systems arising from Adaptive Implicit Method (AIM) discretization in multiphase porous media flow equations. The method focuses on conditioning implicit equations to reduce computational costs and improve solver efficiency. We compare this approach with general algorithms, such as Sinkhorn-Knopp and Ruiz, using the GMRES solver. Results show that the tailored method enhances convergence and outperforms general-purpose techniques in reservoir simulation, highlighting the benefits of problem-specific scaling strategies in large-scale simulations.

Keywords. Scaling Techniques, Iterative Solvers, Reservoir Simulation, Adaptive Implicit Method

1 Introduction

Scaling is a widely used technique to enhance the performance of iterative methods when solving large, sparse linear systems. The primary goal of scaling is to transform the original system into an equivalent one where the rows and columns have similar magnitudes, leading to a better-conditioned system. This improves the convergence behavior of iterative solvers, which is crucial for efficiently solving large-scale problems. In addition to improving convergence, proper scaling allows the use of low-precision arithmetic, a growing area of interest in scientific computing. Low-precision arithmetic can offer significant performance gains, especially with modern hardware, such as GPUs, which are optimized for such operations [1]. Several scaling techniques have been proposed in the literature, including row and column scaling, diagonal scaling, and the more general methods like the Sinkhorn-Knopp [6] and Ruiz [2] algorithms, which aim to balance the norms of rows and columns to achieve a more stable system.

In the specific context of reservoir simulation, proper scaling plays a critical role in the efficiency of solving the discretized equations that model multiphase flow through porous media. As the

[1]pedro.lara@cefet-rj.br
[2]jrprodrigues@petrobras.com.br
[3]cesar.conopoima@cmgl.ca
[4]luizmc@ime.uerj.br
[5]goldfeld@matematica.ufrj.br

complexity of reservoir models increases, so do the size and condition of the resulting linear systems, which can severely affect the performance of solvers. Several researchers have investigated the impact of scaling techniques on the performance of iterative solvers for reservoir simulation, with a focus on improving the conditioning of the matrix and reducing the number of iterations required for convergence.

This work shows a scaling approach tailored to the linear systems arising from the Adaptive Implicit Method (AIM) discretization of multiphase porous media flow equations. This scaling approach addresses the necessary decoupling of implicit and explicit variables in the system. Specifically, the method involves scaling the resulting linear system of implicit variables, with particular emphasis on improving the conditioning of the implicit equations. This method provides a systematic way of improving the conditioning of the matrix, making it more amenable to fast convergence in reservoir simulation applications. This technique promises to enhance the performance of numerical solvers, ultimately reducing computational costs and improving the accuracy of reservoir simulations.

Furthermore, we will compare the scaling approach presented in this work with more general algorithms, such as the Sinkhorn-Knopp and Ruiz methods, using the GMRES [5] solver. These algorithms aim to balance the norms of the rows and columns, achieving a well-conditioned system that facilitates faster convergence and potentially more robust numerical behavior. By analyzing the performance of tailored scaling method alongside these established algorithms within the GMRES solver, we seek to highlight the specific advantages and applicability of our approach for reservoir simulation. Although this scaling strategy may benefit other problems involving large linear systems. Examples include geomechanics, multiphysics simulations, and inverse problems in geophysics, where matrix conditioning and variable coupling structures are also critical.

2 Description of AIM System

We are interested in the solution of the linear system $Jx = b$ where J is the Jacobian matrix resulting from the application of Newton's method to the nonlinear set of algebraic equations resulting from the AIM discretization of the multiphase porous media flow equations. For each cell in the discretized domain, a set of n equations is solved simultaneously and the corresponding unknowns are the pressure p and $n-1$ transport variables X_i, such as saturations and/or compositions. This grouping of equations and variables related to a single cell naturally suggests a block structure for the Jacobian matrix J, where each block J_{kj} corresponds to a small $n \times n$ submatrix associated with cell interactions. In the AIM discretization, pressure is always treated implicitly, while the transport variables can be either explicit or implicit. The decision on whether a cell should have implicit treatment only for pressure or for all unknowns is based on the flow conditions and varies along the simulation both in space and time. The implicit/explicit treatment determines the sparsity of the J_{kj} blocks. While the diagonal blocks J_{kk} are always dense, the columns associated with the explicit unknowns are zero in J_{kj} for $k \neq j$. As it will be explained in the next section, this internal structure of the blocks can be explored to decouple implicit and explicit variables.

3 Decoupling Explicit Variables

Recalling the discussion in the previous section, one block-row of the AIM matrix for an explicit cell with two implicit and two explicit neighbors can be depicted as

DOI: 10.5540/03.2026.012.01.0263

$$\begin{array}{c} \begin{array}{ccc} & \text{E} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} x & 0 & 0 \\ x & 0 & 0 \\ x & 0 & 0 \end{bmatrix} \end{array} \cdots \begin{array}{c} \begin{array}{ccc} & \text{I} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{ccc} & \text{Diag - E} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{ccc} & \text{E} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} x & 0 & 0 \\ x & 0 & 0 \\ x & 0 & 0 \end{bmatrix} \end{array} \cdots \begin{array}{c} \begin{array}{ccc} & \text{I} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \end{array} . \tag{1}$$

The pressure implicit variable is only coupled to the explicit ones through the nonzeros of the first row of the diagonal block. Zeroing out those entries decouples pressure (implicit) from explicit variables:

$$\begin{array}{c} \begin{array}{ccc} & \text{E} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & 0 & 0 \\ x & 0 & 0 \\ x & 0 & 0 \end{bmatrix} \end{array} \cdots \begin{array}{c} \begin{array}{ccc} & \text{I} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & * & * \\ x & x & x \\ x & x & x \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{ccc} & \text{Diag - E} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & 0 & 0 \\ x & x & x \\ x & x & x \end{bmatrix} \end{array} \begin{array}{c} \begin{array}{ccc} & \text{E} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & 0 & 0 \\ x & 0 & 0 \\ x & 0 & 0 \end{bmatrix} \end{array} \cdots \begin{array}{c} \begin{array}{ccc} & \text{I} & \\ p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & * & * \\ x & x & x \\ x & x & x \end{bmatrix} \end{array} . \tag{2}$$

In order to describe this operation mathematically, let k be the index for the block-row corresponding to a cell with explicit treatment. Separating out the pressure from the explicit variables, J_{kk} can be written as

$$J_{kk} = \begin{bmatrix} \alpha_k & v_k^T \\ u_k & A_k \end{bmatrix}, \tag{3}$$

where α_k is a scalar, u_k and v_k are $n-1$ vectors and A_k is an order $n-1$ matrix. α_k and u_k contain the derivatives with respect to pressure, while v_k and A_k contain the derivatives with respect to the explicit variables. Define S_k as

$$S_k = \begin{bmatrix} 1 & \tilde{v}_k^T \\ 0 & I_{n-1} \end{bmatrix} \text{ such that } S_k J_{kk} = \begin{bmatrix} \beta_k & 0 \\ u_k & A_k \end{bmatrix}, \tag{4}$$

where $\tilde{v}_k = -A_k^{-T} v_k$, $\beta_k = \alpha_k + \tilde{v}_k^T u_k$, I_t is the order-t identity matrix, and A_k is assumed invertible. The matrix S_k enables the decoupling between implicit and explicit variables. More precisely, the system $Jx = b$ becomes:

$$\begin{bmatrix} J_{ii} & 0 \\ J_{ei} & J_{ee} \end{bmatrix} \begin{bmatrix} x_i \\ x_e \end{bmatrix} = \begin{bmatrix} b_i \\ b_e \end{bmatrix}. \tag{5}$$

where subscripts i and e refer to the implicit and explicit equations and variables, respectively: x_i is the vector with the implicit unknowns, x_e is the vector with the explicit unknowns and so forth. To see that this is indeed the case, let D_S be the block-diagonal matrix where each block is $D_{S_{kk}} = S_k$ if cell k is treated explicitly, and $D_{S_{kk}} = I_n$ otherwise (implicit treatment) and P be the permutation matrix that reorders implicit unknowns first. Linear system $Jx = b$ is equivalent to $(PD_S JP^T)(Px) = PD_S b$ which will be in the desired form (5). J_{ii} has a block structure with blocks of sizes $n \times n$, $n \times 1$, $1 \times n$ and 1×1, since block rows and columns associated with cells with explicit treatment will have size one, corresponding to pressure which is treated implicitly, while the ones corresponding to cells with implicit treatment will have size n. J_{ee} is a block-diagonal matrix, whose blocks are the order $n-1$ matrices A_k from (3), and J_{ei} contains blocks of sizes $(n-1) \times 1$ and $(n-1) \times n$ corresponding to the dependencies between the explicit equations and the implicit variables.

Solution of the decoupled system (5) proceeds in two steps: Solve $J_{ii} x_i = b_i$ and $x_e = J_{ee}^{-1}(b_e - J_{ei} x_i)$. Due to the block-diagonal structure of J_{ee}, step 2 can be easily accomplished and requires low computational effort. On the other hand, step 1 is the most computationally demanding and requires the use of specialized iterative techniques to achieve high efficiency. In this context, a proper scaling of J_{ii} can play a role, and this is the motivation to consider the application of the row and column scaling algorithms described in the next section.

We close this section mentioning that several references in the literature use J_{kk}^{-1} to decouple implicit from explicit variables. Using the inverse of the diagonal block does provide the desired

decoupling, as depicted below for the same explicit block-row with two implicit and two explicit neighbors example as before:

$$\begin{array}{c} \text{E} \\ \begin{array}{ccc} p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & 0 & 0 \\ * & 0 & 0 \\ * & 0 & 0 \end{bmatrix} \end{array} \cdots \begin{array}{c} \text{I} \\ \begin{array}{ccc} p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \end{array} \begin{array}{c} \text{Diag - E} \\ \begin{array}{ccc} p & X_1 & X_2 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{c} \text{E} \\ \begin{array}{ccc} p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & 0 & 0 \\ * & 0 & 0 \\ * & 0 & 0 \end{bmatrix} \end{array} \cdots \begin{array}{c} \text{I} \\ \begin{array}{ccc} p & X_1 & X_2 \end{array} \\ \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \end{array} . \tag{6}$$

However, it unnecessarily modifies the explicit equations, resulting in a much more expensive procedure. Indeed, while applying S_k to a vector is an $O(n)$ operation, multiplying by the inverse of diagonal block requires $O(n^2)$ operations. In our experience, for problems where the linear systems are not very hard to solve and a relatively small number of iterations is needed, the cost of this decoupling step can be substantial, particularly when the percentage of cells with explicit treatment is large.

Using J_{kk}^{-1} also scales the resulting pressure equations, with ones appearing in the diagonal. We note that this could be easily achieved without modifying the explicit equations by employing

$$\tilde{S}_k = \begin{bmatrix} \beta_k^{-1} & \beta_k^{-1}\tilde{v}_k^T \\ 0 & I \end{bmatrix} \tag{7}$$

instead of S_k. Since $\begin{bmatrix} \beta_k^{-1} & \beta_k^{-1}\tilde{v}_k^T \end{bmatrix}$ is the first row of J_{kk}^{-1}, this approach results in the same implicit matrix as multiplying by the inverse diagonal block.

4 Iterative Methods for Matrix Equilibration

This section explores the principles and steps involved in both the Sinkhorn-Knopp and Ruiz algorithms. We will compare this iterative process with the method shown in this paper. R. Sinkhorn and P. Knopp [6] proposed an iterative process aimed at transforming a non-negative matrix A into an approximately doubly stochastic matrix. A doubly stochastic matrix is one in which all rows and columns sum to 1, and all elements are non-negative. The output of this process consists of two diagonal matrices, R and C, such that $R \cdot A \cdot C$ is approximately doubly stochastic. This algorithm has been applied across various fields, including Optimal Transport, Image Normalization, Economic Equilibrium, and Scaling for Iterative Linear Methods [3]. The Sinkhorn-Knopp theorem states that, given a nonnegative matrix A, a necessary and sufficient condition for the existence of a doubly stochastic matrix $P = R \cdot A \cdot C$, is that A has total support. Furthermore, if this matrix P exists, it is unique. Moreover, the matrices R and C are also unique up to a scalar multiple if and only if A is fully indecomposable [6]. The application of the Sinkhorn-Knopp algorithm as a preconditioner stems from its ability to balance the rows and columns of a matrix, which can significantly improve the condition number and, in turn, enhance the convergence rate of iterative solvers. By scaling the matrix to achieve this equilibrium, the algorithm mitigates the numerical instability often caused by large variations in the magnitudes of the matrix elements. This balanced structure can reduce error propagation, which is especially beneficial for ill-conditioned matrices commonly encountered in numerical simulations and optimization problems.

Additionally, the preconditioning effect helps align the eigenvalues, further improving the efficiency of methods such as GMRES or Conjugate Gradient. In general, the Sinkhorn-Knopp algorithm as a preconditioner offers a straightforward yet powerful tool to achieve faster and more stable convergence in various computational applications [2, 6].

The Ruiz Algorithm [2] is an iterative method designed to balance the norms of the row and column of a matrix, originally using the norm $\|\cdot\|_\infty$. By iteratively scaling each row and column to achieve approximately equal norms, this algorithm effectively reduces the condition number of the matrix, making it a valuable tool to improve the performance of numerical solvers [3]. The

DOI: 10.5540/03.2026.012.01.0263

Ruiz Algorithm is widely applied in areas such as sparse linear systems, where balanced matrices improve the efficiency of iterative solvers; optimization, where scaling enhances the reliability of convergence in algorithms; and machine learning, where it is used in preconditioning large datasets to facilitate more robust training and convergence in gradient-based methods [3]. Given a matrix A of size $N \times N$, the Ruiz Algorithm aims to adjust A such that the maximum norm $\|\cdot\|_\infty$ of each row and each column approaches uniformity. The algorithm starts with identity matrices R and C, which hold the scaling factors for rows and columns, respectively. During each iteration, diagonal matrices $\mathcal{R}$ and $\mathcal{C}$ are computed, with their diagonal entries given by the inverse square roots of the row and column norms. These matrices are applied to update R and C, and the process continues until convergence.

5 Results

This section presents a comparative analysis using twelve AIM matrices derived from two realistic simulations: Unisim and Olympus. The results are summarized in Tables 1 and 2. The Unisim matrices originate from the Unisim-III benchmark model, a synthetic yet realistic pre-salt reservoir model developed by the UNISIM research group at the University of Campinas (UNICAMP).[6]. The Olympus matrices are derived from the Olympus benchmark model, developed by the Group of Technology in Energy and Petroleum (GTEP) at PUC-Rio[7]. This model evaluates geomechanical effects using realistic multiphysics conditions. All experiments were conducted using implementations in C++, running on a system equipped with an Intel Core i9-14900HX processor and 64GB of RAM. The GMRES iterative solver and the associated preconditioners were implemented using the Eigen C++ library[8].

Let A be the original system matrix and b the corresponding right-hand side (RHS). Define $\tilde{A}$ and $\tilde{b}$ as the scaled matrix and RHS, respectively. Let x denote the exact solution and $\tilde{x}$ the solution to $\tilde{A}x = \tilde{b}$ obtained via GMRES. For the comparative analysis, we evaluate the following metrics, with the column identifiers as shown in Tables 1 and 2: the number of GMRES iterations to reach a specified tolerance (#Iter); the name of the scaling used, with "NoScale" indicating the original matrix (Scaling); the relative error $|x - \tilde{x}|/|x|$ computed using BiCGSTAB [7] with a tolerance of 1e-16 (Sol. Error); the final preconditioned relative residual norm $|P\tilde{A}\tilde{x} - P\tilde{b}|/|P\tilde{b}|$ (Prec. Res. Norm); the final relative residual norm $|\tilde{A}\tilde{x} - \tilde{b}|/|\tilde{b}|$ (Res. Norm); and the number of iterations of the scaling algorithm, applicable only to Sinkhorn-Knopp and Ruiz methods (Scal. #Iter). For these experiments, the preconditioner used was Incomplete LU decomposition with threshold [4].

The choice of tolerance in scaling algorithms significantly affects both computational cost and solution accuracy. In this study, a fixed tolerance of 1e-3 was used for the SK and Ruiz algorithms to balance efficiency and robustness. Stricter tolerances can improve matrix conditioning and reduce GMRES iterations but increase the cost of the scaling phase, while looser tolerances may reduce scaling effort at the expense of GMRES performance. Empirically, the 1e-3 setting provided a practical trade-off, improving convergence without excessive overhead. Based on the results shown in Tables 1 and 2, we can conclude that the Row and SK scalings produced very similar results, especially in terms of the number of GMRES iterations and the error relative to the system's solution. We conducted a computational time analysis of the scaling methods for the Olympus 1 and Unisim 1 matrices. All reported times represent the average over 100 runs to mitigate fluctuations. While each iteration of the Sinkhorn-Knopp (SK) method is more efficient than the others, it requires a significantly larger number of iterations. For instance, SK required 1084

[6] https://www.unisim.cepetro.unicamp.br/benchmarks/br/unisim-iii

[7] https://gtep.com.br/index.php/reservoir-geomechanics/

[8] https://eigen.tuxfamily.org/

Table 1: Results for Unisim using GMRES preconditioned with ILUT with a tolerance of 1e-10, while scaling algorithms were tested with a tolerance of 1e-3. The matrix has an order of 230,408 with 1,397,085 non-zero elements.

#	Scaling	#Iter	Sol. Error	Prec. Res. Norm	Res. Norm	Scal. #Iter
1	NoScale	7	7.8192e-11	2.3167e-11	1.4577e-09	–
	Row	3	2.6400e-12	1.9312e-12	1.4805e-12	–
	SK	3	1.1425e-12	2.2292e-13	2.5045e-13	3149
	Ruiz	4	9.1314e-10	9.8552e-13	7.8145e-11	14
2	NoScale	8	1.2889e-11	4.5220e-12	1.1684e-12	–
	Row	3	1.2841e-11	7.4004e-13	3.8744e-13	–
	SK	3	1.2841e-11	1.6469e-13	4.4404e-14	2936
	Ruiz	5	1.1241e-08	6.7093e-11	6.9467e-10	14
3	NoScale	9	6.3618e-12	4.6089e-12	9.5871e-11	–
	Row	4	2.2297e-13	2.0323e-13	2.4374e-13	–
	SK	4	3.5927e-13	1.5897e-14	3.1020e-14	2795
	Ruiz	6	3.2120e-08	4.3786e-11	1.3563e-08	14
4	NoScale	10	1.0686e-10	3.1484e-11	2.0771e-10	–
	Row	4	4.1286e-12	2.9097e-12	4.7411e-12	–
	SK	4	9.9079e-11	3.1977e-11	1.5416e-10	2606
	Ruiz	9	2.4082e-09	3.8407e-12	1.0255e-09	14
5	NoScale	10	2.6039e-10	1.9038e-11	1.4750e-10	–
	Row	4	1.0334e-10	5.2727e-11	7.1679e-11	–
	SK	5	3.1398e-12	4.0953e-13	1.3891e-12	2014
	Ruiz	11	8.3328e-09	2.4544e-11	9.7388e-10	14
6	NoScale	10	3.8428e-10	6.3508e-11	6.2494e-11	–
	Row	5	2.5423e-11	2.4244e-11	2.1204e-11	–
	SK	5	3.1232e-11	1.4140e-11	4.9682e-11	1724
	Ruiz	12	8.0534e-09	3.5254e-11	4.8041e-09	14

iterations to converge for the Olympus 1 matrix, with an average time of 2.23 ms per iteration, resulting in a total execution time of 2421.65 ms. For the Unisim 1 matrix, SK required 3.22 ms per iteration, totaling 10139.78 ms. In contrast, the Ruiz method required only 13 and 14 iterations for Olympus 1 and Unisim 1, respectively, with per-iteration times of 9.29 ms and 11.84 ms (for total times of 120.74 ms and 165.76 ms). The Row scaling method, which is a direct method, achieved the lowest total execution times: 22.88 ms for Olympus 1 and 26.39 ms for Unisim 1. For context, each GMRES iteration took approximately 11.38 ms (Olympus 1) and 8.33 ms (Unisim 1). These results confirm that, in addition to reducing the number of GMRES iterations, the Row technique provides significant computational efficiency.

6 Conclusion

This work presents a tailored scaling approach for linear systems arising from the Adaptive Implicit Method (AIM) discretization in multiphase porous media flow equations. The method was explained in detail and compared with general-purpose scaling algorithms. Our results demonstrate that the tailored strategy improves the conditioning of implicit equations, leading to enhanced solver performance and reduced computational costs in reservoir simulation applications.

The results indicate that the Row scaling approach effectively reduces the number of GMRES iterations required for convergence while maintaining low solution error and residual norms. In particular, the method highlighted in this work, consistently outperformed general algorithms in terms of efficiency, achieving faster GMRES convergence with fewer computational effort compared to Sinkhorn-Knopp, while also providing more robust conditioning than the Ruiz method. These findings highlight the importance of domain-specific scaling techniques in large-scale simulations, where computational efficiency and accuracy are critical.

DOI: 10.5540/03.2026.012.01.0263

Table 2: Results for Olympus using GMRES preconditioned with ILUT with a tolerance of 1e-6, while scaling algorithms were tested with a tolerance of 1e-3. The matrix has an order of 199,057 with 1,380,877 non-zero elements.

#	Scaling	#Iter	Exact Sol. Error	GMRES Prec. Error	GMRES Error	Scal. #Iter
1	NoScale	16	5.5721e-07	5.9585e-07	1.5599e-05	–
	Row	12	1.0938e-06	7.7398e-07	6.6876e-07	–
	SK	13	1.6609e-06	5.7377e-07	1.6578e-06	1084
	Ruiz	18	1.2800e-06	6.0068e-07	9.5553e-06	13
2	NoScale	21	1.3821e-06	8.4714e-07	4.9526e-05	–
	Row	18	1.1521e-06	9.6338e-07	1.2249e-06	–
	SK	18	1.7612e-06	6.6996e-07	4.2827e-06	1087
	Ruiz	22	1.5785e-06	8.7252e-07	1.8781e-05	13
3	NoScale	27	1.7714e-06	9.1594e-07	1.2999e-04	–
	Row	22	8.9344e-07	8.2157e-07	1.4038e-06	–
	SK	23	2.1875e-06	6.9634e-07	5.5338e-06	1088
	Ruiz	29	1.7659e-06	7.1836e-07	6.5063e-05	13
4	NoScale	42	2.3337e-06	8.5617e-07	7.4607e-05	–
	Row	30	7.4950e-07	5.8878e-07	2.9671e-06	–
	SK	30	2.5215e-06	9.6747e-07	2.8685e-05	1088
	Ruiz	42	1.1178e-06	8.0780e-07	6.1023e-05	13
5	NoScale	51	5.0916e-06	8.1285e-07	1.7686e-04	–
	Row	41	2.1862e-06	8.2383e-07	2.2555e-06	–
	SK	41	3.0160e-06	9.9793e-07	1.4265e-05	1089
	Ruiz	50	3.1641e-06	9.4062e-07	6.8470e-05	13
6	NoScale	55	1.4236e-06	8.0144e-07	3.6464e-06	–
	Row	44	1.3846e-06	9.0887e-07	2.0565e-07	–
	SK	42	5.1367e-07	6.0111e-07	5.4134e-06	1089
	Ruiz	52	8.5355e-07	8.9788e-07	2.4286e-05	13

References

[1] N. J. Higham, S. Pranesh, and M. Zounon. “Squeezing a Matrix into Half Precision, with an Application to Solving Linear Systems”. In: **SIAM Journal on Scientific Computing** 41.4 (2019), A2536–A2551. ISSN: 1064-8275. DOI: 10.1137/18M1229511.

[2] P. A. Knight and D. Ruiz. “A fast algorithm for matrix balancing”. In: **IMA J. of Num. Analysis** 33.3 (2012), pp. 1029–1047. ISSN: 0272-4979. DOI: 10.1093/imanum/drs019.

[3] P. A. Knight and D. Ruiz. “A symmetry preserving algorithm for matrix scaling”. In: **SIAM Journal on Matrix Analysis and Applications** 35.3 (2013), pp. 931–955. ISSN: 0895-4798. DOI: 10.1137/110825753.

[4] Y. Saad. “ILUT: A dual threshold incomplete LU factorization”. In: **Numer. Linear Algebra Appl.** 1 (1994), pp. 387–402. ISSN: 1070-5325. DOI: 10.1002/nla.1680010405.

[5] Y. Saad and M. H. Schultz. “GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: **SIAM J. Sci. Stat. Comput.** 7.3 (1986), pp. 856–869. ISSN: 0196-5204. DOI: 10.1137/0907058.

[6] R. Sinkhorn and P. Knopp. “Concerning nonnegative matrices and doubly stochastic matrices”. In: **Pacific Journal of Mathematics** 21.2 (1967), pp. 343–348. ISSN: 0030-8730. DOI: 10.2140/pjm.1967.21.343.

[7] H. A. van der Vorst. “Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems”. In: **SIAM Journal on Scientific and Statistical Computing** 13.2 (1992), pp. 631–644. ISSN: 0196-5204. DOI: 10.1137/0913035.

