
Open access

RISET
Geologi dan Pertambangan

https://jrisetgeotam.brin.go.id/index.php/jrisgeotam

Geology and Mining Research

Riset Geologi dan Pertambangan (2024) Vol.34, No.2, 131–152, DOI: 10.55981/risetgeotam.2024.1345

Research article

Adaptive preconditioning Krylov Subspace methods for efficient numerical groundwater flow modeling in steady-state conditions

Gumilar Utamas Nugraha, Hendra Bakti, Rachmat Fajar Lubis

Research Center for Limnology and Water Resources, National Research and Innovation Agency of Indonesia (PRLSDA-BRIN), Cibinong, Bogor 16911, Indonesia

Keywords:
numerical groundwater modeling
Krylov Subspace method
adaptive preconditioning
sparse matrices, steady-state simulation

Corresponding author:
Gumilar Utamas Nugraha
Email address: gumlabz@gmail.com

Article history
Received : 29 November 2024
Revised : 13 December 2024
Accepted : 18 December 2024

ABSTRACT

The numerical method plays an important role in groundwater flow modeling to solve linear equations with sparse matrices. This study evaluates the performance of the Krylov Subspace method with adaptive preconditioning compared to classical iterative methods, such as Gauss-Seidel, Jacobi, and Successive Over-Relaxation (SOR), in the simulation of steady-state groundwater flow on a 2D grid. The results show that the Krylov method with adaptive preconditioning provides the fastest execution time (0.0054 seconds) with minimal resource usage, such as CPU of 0.0% and RAM of only 0.175 MB. In contrast, the classic iterative method shows longer execution times and greater resource consumption. This study concludes that the Krylov Subspace method with adaptive preconditioning is the best solution for applications that require high efficiency in groundwater flow computing.

INTRODUCTION

Groundwater flow modeling plays an important role in water resource management and mitigation of environmental problems, such as groundwater pollution and development planning. The complexity of domain geometry and the heterogeneity of the hydraulic properties of media are often challenges that require efficient and accurate computing solutions. To solve groundwater flow equations in the form of linear systems, various numerical methods have been used, including iterative methods such as Gauss-Seidel, Jacobi, and Successive Over-Relaxation (SOR). However, these methods often face obstacles in the form of slow convergence, especially for large and sparse systems, as well as suboptimal computing resource requirements (Bair, 2016; Jackson, 2007; Nugraha et al., 2024; Remson, 1982; Benali, 2013).

In this context, the Krylov Subspace method with adaptive preconditioning is a promising approach because it is able to offer a faster and more efficient solution. This method works by utilizing iteratively constructed vector subspaces to speed up the completion process of linear systems. In addition, adaptive preconditioning helps improve the structure of the matrix thus supporting faster convergence. This approach has become particularly relevant in the simulation of steady-state groundwater flows, which often involve large sparse matrices (Downar and Joo, 2001a; Hashimoto and Nodera, 2022; Jolivet et al., 2021; Quillen and Ye, 2010a; Sabaté Landman et al., 2024; Downar and Joo, 2001a; Hashimoto and Nodera, 2022; Hashimoto and Nodera, 2022).

This research offers novelty by applying the Krylov Subspace method equipped with adaptive preconditioning to optimize the efficiency of computing time and system resource usage. The main focus is to evaluate the performance of this method compared to the classical iterative method through simulation on a 2D grid domain. Analytics include measurements of execution time, CPU usage, memory consumption, and power efficiency.

This study aims to develop a more efficient and fast solution for groundwater flow models based on linear sparse systems, evaluate the performance of the Krylov Subspace method with adaptive preconditioning compared to the classical iterative method, and provide recommendations for the best method based on the results of the analysis. Thus, this research is expected to contribute to the development of a superior numerical approach in groundwater flow modeling.

METHOD

Basic Theory of Groundwater Flow

Groundwater flow is the movement of water through a porous medium, which is influenced by factors such as permeability, porosity, and pressure gradient or water table. In the world of engineering and geology, groundwater flow models are used to understand how water moves in the ground, whether for water resource management, pollution studies, or development planning (Fitts, 2013; Jain, 2014; Kelbe et al., 2011).

Groundwater Flow Governing Equation

The governing equation of groundwater flow can be derived from Darcy's law, which states that the rate of water flow through soil pores is directly proportional to the pressure gradient and permeability of the soil (Todd and Mays, 2005).

For steady-state groundwater flows, the governing equation used is Laplace's equation in two-dimensional form (Domenico and Schwartz, 1990; Fetter, 2001):

$$\frac{\partial^2 H}{\partial x^2} + \frac{\partial^2 H}{\partial y^2} = 0$$

Where H is the water level (head) at a certain point, x and y is the spatial coordinate, the gradient H controls the direction and speed of groundwater flow.

Boundary Conditions

Boundary conditions are critical to solving these numerical models, as they provide the additional information needed to determine a unique solution. There are two main types of boundary conditions that are often used, namely Dirichlet boundary conditions and Neumann boundary conditions (Reilly and Harbaugh, 2004).

The Dirichlet boundary condition sets the value of the searched function, in this case the H value, at the domain boundary. That is, we specify a definite value for H along the boundaries of that domain. Meanwhile, the Neumann boundary condition sets the gradient value of the function along the domain boundary, just like the normal flow condition, where we set the rate of change H at that boundary, rather than the direct value of H. These two conditions help define the behavior of the solution at the edge area of the domain being analyzed.

Numerical Methods for Settlement

To solve the groundwater flow equation numerically, we use a discrete technique, which divides the physical domain into a grid or cell, and replaces the derivatives with finite differences (Wang and Anderson, 1982).

Finite Difference Method (FDM)

The Finite Difference Method (FDM) is the most commonly used numerical technique to solve groundwater flow equations. In FDM, continuous domains are replaced by discrete grids, and partial derivatives are calculated using a finite differential approach.

For one-dimensional steady-state streams, the discrete Laplace equation yields:

$$\frac{H_{i-1} - 2H_i + H_{i+1}}{(\Delta x)^2} = 0$$

Where Δx is the distance between the grids. This equation relates the value H_i at the grid points i with the value H at the neighbor next to it.

Finite Element Method (FEM)

This method is more flexible in handling complex geometries. With FEM, the domain is divided into small elements, and the flow equation is solved within each element by a variational approach.

This method is generally more appropriate for problems involving constraints or irregular geometry, although it is more complex in its implementation compared to FDM.

Finite Volume Method (FVM)

The Infinite Volume Method is an approach used to solve flow equations by paying more attention to the principle of mass conservation. In this method, the control volume is formed around each grid point, and the flow integral is calculated on the surface of that control volume.

Solving the System of Equations

After discrete of the governing equation, we will get a system of linear equations that must be solved using various numerical methods (Wang and Anderson, 1982).

Coefficient Matrix

The linear equation system of groundwater flow models is usually arranged in the form of a $A \cdot H = b$ matrix, where: A is the coefficient matrix, H is the vector of water level values on the grid, b is the result vector containing the values determined by the boundary conditions. This matrix is usually a sparse matrix because most of the elements outside the diagonal are zero, which reflects the nature of groundwater flow that is only affected by neighboring points.

Matrix Solutions

To solve the linear equation system, we can use two main approaches, namely the direct method and the iterative method. Direct methods, such as LU Decomposition or Gaussian Elimination, are typically used for relatively small systems of equations, as they compute the solution exactly in one step. On the other hand, iterative methods, such as Jacobi, Gauss-Seidel, or Successive Over-Relaxation (SOR), are more efficiently applied to large and sparse systems. In iterative methods, solution values are updated iteratively until convergence, with each iteration approaching a more accurate solution.

Krylov Subspace Method with Adaptive Preconditioning

The Krylov Subspace method with Adaptive Preconditioning is a numerical approach used to solve large and sparse systems of linear equations, especially in the context of scientific computing, such as physics simulations, optimization, or fluid dynamics problems. Basically, this method utilizes Krylov subspace and adaptive preconditioning techniques to improve the speed and efficiency of convergence of iterations in linear system completion (Aksoylu and Klie, 2009; Bujurke and Kantli, 2021; Downar and Joo, 2001b; Il'in, 2021; Kuether and Steyer, 2024; Quillen and Ye, 2010b; Sabaté Landman et al., 2024; Il'in, 2021).

Basic Krylov Subspace Method

Essentially, Krylov's method works with iterations to narrow down a solution in a subspace defined by an iteration of the initial coefficient A and vector matrix b. Krylov's concept of subspace arises from the observation that a good solution can be proximate in a space constructed from the repeated times of a matrix A with a vector b.

The Krylov subspace $K_m(A, b)$ is a vector space resulting from the first iteration m of the system $Ax = b$. In other words, $K_m(A, b)$ is the linear range of the vector $Ab, A^2b, \dots, A^{m-1}b$. Mathematically, this is expressed as:

$$K_m(A, b) = \text{span}\{b, Ab, A^2b, \dots, A^{m-1}b\}$$

This iteration process reveals that solutions can be approached in small subspaces without the need to search for solutions in the entire full-dimensional space. The Krylov subspace essentially reduces the dimension of the problem, making it easier to calculate compared to non-iterative methods.

This means $K_m(A, b)$ is a vector space constructed from all linear combinations from $bAbA^2b$ to $A^{m-1}b$. The greater the value m, the greater the dimensions of the Krylov subspace and the more possible linear combinations we can use to approach the solution x.

The procedure in Krylov's method begins with the selection of the initial vector, which is often the initial residual vector $r_0 = b - Ax_0$, which x_0 is the initial approximation of the solution sought. This initial vector is then used in the process of subspace repetition, where the vector is multiplied repeatedly by a matrix A. Each multiplication result in a row of vectors that form a Krylov subspace, which is used to approximate the solution of a linear system of equations.

After that, Krylov's method builds the solution as a linear combination of vectors contained in Krylov's subspace. The solution on the iteration m can be expressed as $x_m = x_0 + \sum_{i=1}^{m} \alpha_i v_i$, where v_i is the base vector in the Krylov subspace and α_i is the coefficient specified during the iteration process. This iteration process continues until the error or residual, which is counted as. $\|Ax_m - b\|$ reaches a fairly small value. This indicates that the solution x_m is sufficiently close to the actual solution x, and the iteration can be stopped once sufficient convergence has been achieved

Krylov methods such as GMRES (Generalized Minimal Residual), BiCGSTAB (Biconjugate Gradient Stabilized), and Conjugate Gradient (CG) are examples of methods that are often used. This study focuses

more on the GMRES (Generalized Minimum Residual) method. GMRES (Generalized Minimal Residual) is one of the iterative methods used to solve a system of linear equations in shapes, where is a usually large and sparse coefficient matrix (sparse), and bbb is a constant vector. GMRES is a very powerful method, especially for non-symmetrical and non-positive-definite linear equation systems, which makes it more flexible than other methods such as Conjugate Gradient (CG) which are only suitable for positive-definite symmetrical matrices $Ax = bA$.

GMRES works with the principle of building solutions in Krylov subspaces and choosing solutions that minimize residuals $r_m = b - Ax_m$, which means looking for solutions x_m in Krylov subspaces that minimize errors or distances between the results of the multiplication of the matrix A and the solution of approximation and constant vectors b.

The residual r_m in the second iteration m is the difference between b and the multiplication of the matrix A with the approximation solution x_m:

$$r_m = b - Ax_m$$

This residual norm is a measure of the precision of our solution to the system of equations:

$$\|r_m\| = \|b - Ax_m\|$$

In GMRES, we want the solution x_m to be as close to the actual solution x as possible, with the smallest residual possible.

The GMRES algorithm is an iterative method that builds a solution incrementally in Krylov space, defined by vectors resulting from iterations A on constant vectors b. Here are the steps of the GMRES algorithm.

The GMRES algorithm begins with the initialization stage, where we start with an initial solution estimate x_0, which is usually a zero vector or a better estimate if available. After that, the initial residual $r_0 = b - Ax_0$ is calculated by the formula, where b is the vector of the constant and A is the matrix of coefficients. Then, this initial residue is normalized so that a vector $v_1 = r_0/\|r_0\|$ is obtained, which will be used in the next steps.

Next, the process enters the stage of building the Krylov subspace. In each iteration, we start $m = 1,2,3, \ldots$ by counting the vectors $v_{m+1} = Av_m$ and then perform the orthogonality process on all the previously calculated vectors using the Gram-Schmidt method. This process aims to ensure that all vectors $v_1, v_2, \ldots, v_m$ remain orthogonal to each other, keeping the vector bases in Krylov's subspace preserved. After the orthogonality step, we form the Hessenberg matrix H_m, which has an upper trapezoidal shape, with the elements H_m defined by the products in the Krylov subspace.

Once the matrix H_m is formed, the next step is to solve the decomposed linear system with the equation $H_m y = \|r_0\|e_1$, where e_1 is the first column vector of the identity matrix and y is the solution vector containing the linear combination coefficients of the Krylov subspace base vector. From this step, we construct an approximation solution x_m in the m th-iteration, which is calculated as $x_m = x_0 + V_m y$, where V_m is a matrix consisting of columns $v_1, v_2, \ldots, v_m$ forming the base of the Krylov subspace.

The vector y discovered earlier gives the linear combination coefficient of the base. This iteration process continues until the residual $\|r_m\|$ reaches a sufficiently small value, or if the iteration limit has been reached. GMRES relies heavily on the Krylov subspace constructed by matrix iterations A on residual vectors r_0 . The matrix H_m is a Hessenberg matrix obtained through the decomposition of the QR on the matrix $[r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0]$, which plays an important role in controlling the iteration process. This matrix makes it easy to obtain solutions due to its simple shape. During the iteration process, we continue to ensure that the base vectors in the Krylov subspace remain orthogonal using Gram-Schmidt (or variants such as Modified Gram-Schmidt), which also reduces the likelihood of numerical regression, which is especially important when working with large and sparse matrices

Adaptive Preconditioning

Preconditioning aims to improve the quality of the matrix structure in a linear equation system, by making it "well-conditioned". This means that the eigenvalues of the matrix are more evenly distributed, which in turn favors faster convergence in iterative methods. An effective preconditioner can significantly reduce the number of iterations required to complete a linear equation system with the Krylov method.

In the system of equations $Ax = b$, we introduce a matrix of preconditioners M, so that the system is transformed into:

$$M^{-1}Ax = M^{-1}b$$

The main goal of preconditioning is to choose M in such a way that the matrix $M^{-1}A$ approaches the identity matrix I. With a more regular matrix structure, this system becomes easier to solve using iterative methods.

Adaptive preconditioning is a dynamic approach in which preconditioners M are updated or adjusted during an iteration of Krylov's method. The purpose of this adaptation is for the preconditioner to adapt to the changes in matrix characteristics that occur during the iteration, resulting in a more optimal approach and accelerating convergence.

The preconditioning adaptation process begins with selecting the initial preconditioner in the first iteration, which usually uses techniques such as ILU (Incomplete LU Factorization) or Jacobi preconditioner. Once the preconditioner is selected, its effectiveness evaluation is carried out every few iterations by checking for residuals or errors. If the residue does not decrease quickly, this indicates that the preconditioners currently in use are less effective in accelerating convergence.

If the evaluation shows that the preconditioner is not effective, then the next step is to update the preconditioner. This update was made to make the preconditioner more suitable for the characteristics of the matrix A that changed during the iteration. For example, if the matrix A exhibits increasingly strong asymmetric behavior, the preconditioner can be changed to accommodate that trait. The new preconditioner is then applied to the next iteration, and this process can be repeated as needed to achieve optimal convergence.

Some of the types of preconditioners that are often used in adaptive preconditioning are ILU (Incomplete LU Factorization), SSOR (Symmetric Successive Over-Relaxation), and Diagonal or Jacobi Preconditioner. ILU is one of the most commonly used methods, especially because it can be adapted to ongoing iterations. The ILU divides the matrix into two matrices, the lower (L) and upper (U) matrices, which are close to the LU decomposition of the original matrix, but retain only the most significant elements, such as the largest or most relevant values.

Another method used is SSOR, which combines the advantages of the Jacobi and Gauss-Seidel methods in a symmetrical form, making it ideal for symmetrical matrices. On the other hand, Diagonal or Jacobi Preconditioner is a simpler method, which uses only the diagonal elements of the matrix A as a preconditioner. This method is more suitable for matrices with a diagonal dominant structure.

In this study, the preconditioner used was ILU (Incomplete LU Factorization). ILU is a decomposition method used to produce preconditioners in the completion of linear equation systems, especially for sparse matrices. ILU works by factoring the initial matrix, but retaining only certain elements to keep the structure sparse, thus improving the computational speed and memory usage efficiency compared to full LU decomposition.

ILU aims to approach LU decomposition, where the matrix A is decomposed into a lower (L) and upper (U) matrix, yielding:

$$A \approx LU$$

In full LU decomposition $A = LU$, however, in ILU, only a portion of the elements of L and U are counted. ILU is "incomplete decomposition" because it only counts elements that are in a certain pattern, e.g. non-zero elements in a matrix A, while other elements are ignored.

With this approach, the number of elements that need to be stored is less, thus saving memory and speeding up the computing process. ILU's approach provides a huge advantage in efficiency, especially when working with large and sparse matrices.

Efficient Sparsity and Storage for efficiency solver Krylov Subspace Methods with Adaptive Preconditioning

To accelerate the execution of the Krylov Subspace Methods solver with Adaptive Preconditioning, several effective strategies can be implemented to improve performance in terms of execution time and memory efficiency. One of the important elements is the selection of the right preconditioner, which is adapted to the characteristics of the system being completed. With adaptive preconditioners, the resulting solution can be more efficient, as simpler preconditioners are usually faster, while more complex preconditioners can accelerate convergence under certain conditions. For example, Incomplete LU (ILU) or Incomplete Cholesky-based preconditioners often provide an optimal balance between compute time and convergence effectiveness. In large and sparse systems, multigrid preconditioners can also speed up convergence significantly, as they allow for faster calculations at various grid levels.

The selection of the optimal Krylov method is also important in adapting the method to the type of equation system. For example, in symmetrical and positively definitized systems, the Conjugate Gradient (CG) method is often more efficient than GMRES. Meanwhile, GMRES is great for non-symmetrical problems, although this method usually requires more memory to store subspace. To improve memory efficiency, GMRES with restart is a wise choice because it limits the size of the stored subspace, and reducing the dimensions of Krylov's subspace is another way to speed up the execution of the solver. Filtering out less significant components can reduce the size of the computing space, saving on computational costs without sacrificing accuracy.

Parallelization techniques are very useful for speeding up execution, especially in matrix operations such as matrix-vector multiplication or applying preconditioners. Many modern Krylov solvers support automatic parallelization; therefore, the use of OpenMP or MPI can speed up execution at both the thread and node levels. For very large systems, GPU acceleration with CUDA or OpenCL also provides a significant speed boost. Krylov's GPU-accelerated methods, such as GPU-accelerated GMRES or CG, can drastically cut execution times.

Efficiency in sparse matrix storage is essential to save memory and speed up matrix operations. Storage structures such as Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC) are helpful for large and sparse systems. Preconditioners should also be stored in efficient structures, such as block-based preconditioners or preconditioner iterations that only count the parts needed in a given iteration. In addition, adaptive regularization of Krylov preconditioners and solvers can accelerate convergence by selecting optimal parameters during the iteration process. Determining the appropriate convergence tolerance threshold can also speed up execution; Tolerances that are too tight will slow down convergence, while looser tolerances allow for faster execution if slightly lower accuracy is still acceptable.

Some Krylov methods and certain preconditioners can also produce better performance if the order of the elements in the matrix is changed. For example, techniques such as Cuthill-McKee ordering can be used to reduce the bandwidth of the matrix, which in turn increases the speed and effectiveness of the solver. In this study, the improvement of the solver execution efficiency will be focused on modifying sparsity and using storage more efficiently through the compressed sparse row (CSR) structure. Some

additional ways to improve memory and computing efficiency include using more efficient sparse storage structures, such as scipy.sparse to store matrices in lil_matrix or csr_matrix format, and ensuring sparse preconditioners are used with Krylov's methods (such as spilu).

In the implementation code, adjustments to focus on efficiency include storing the matrix in sparse format from the start by changing the lil_matrix format to csr_matrix, which is more efficient for matrix-vector operations. In addition, preconditioner optimization with ILU sparse can be done by using scipy.sparse.linalg.spilu to maximize the efficiency of preconditioners in the Krylov method.

'lil_matrix' is a type of storage structure in the form of a sparse matrix provided by the SciPy library, specifically for handling large matrices with many zero elements. The name 'lil_matrix' stands for List of Lists Matrix, as this matrix is stored in the form of a list of row lists. This storage approach makes it very easy to efficiently modify matrix elements, especially when non-zero elements need to be added or removed frequently.

Each row in the 'lil_matrix' is stored as a separate list. This allows for access and modification of elements on each row in a flexible way, making it suitable for situations where matrices require dynamic changes. Also, in terms of storage, 'lil_matrix' is more memory-efficient compared to full matrix storage as it only stores non-zero elements. This is what makes it ideal for handling large and sparse matrices.

In the process of building a new matrix, 'lil_matrix' is very useful because it provides good efficiency when the addition of non-zero elements is still ongoing. However, when the matrix is complete, 'lil_matrix' is less efficient for arithmetic operations such as matrix-vector multiplication because it is not optimized for fast computational processing. For operations like this, conversion to a more optimal sparse format such as 'csr_matrix' (Compressed Sparse Row) or 'csc_matrix' (Compressed Sparse Column) is preferred.

'csr_matrix', or Compressed Sparse Row Matrix, is a type of data structure in the form of a sparse matrix provided by the SciPy library. This structure is specifically designed to efficiently store and manipulate large matrices containing many zero elements, especially for computational operations such as matrix-vector multiplication or other linear operations. 'csr_matrix' is particularly useful in numerical processing, thanks to its high storage efficiency as well as its fast performance on certain operations.

One of the main advantages of 'csr_matrix' is memory-saving storage. In 'csr_matrix', only non-zero elements are stored, complete with their row and column indexes. This approach makes 'csr_matrix' much more efficient in memory usage compared to a dense matrix, where every element, including the zero one, must still be stored.

The 'csr_matrix' storage structure uses three main arrays: 'data', 'indices', and 'indptr'. The 'data' array stores the values of the non-zero elements in the matrix. The 'indices' array stores the column position for each non-zero element. Meanwhile, the 'indptr' array stores the initial index of each row in the 'data' array, allowing direct access to the elements on a particular row.

In addition to being memory-efficient, csr_matrix is also highly efficient for matrix-vector operations and line manipulation. Because it uses a compression structure, matrix-vector operations can be performed quickly. However, in contrast to the 'lil_matrix' (List of Lists Matrix) which is more flexible for adding new elements but less efficient for arithmetic operations, 'csr_matrix' is optimal for computational purposes.

However, 'csr_matrix' is less efficient in column operations because this structure is specifically designed for quick access to elements per row. For applications that require more column access, a format such as 'csc_matrix' (Compressed Sparse Column) is more appropriate.

Overall, 'csr_matrix' is an ideal choice for large and sparse matrices, especially when access to row elements, matrix-vector multiplication, or linear operations is often required. This structure provides

high efficiency in terms of storage and computing, making it particularly useful for a variety of numerical and computing applications.

RESULTS AND DISCUSSION

In this section, the results of groundwater flow simulations carried out using various iterative methods will be discussed and analyzed. The methods tested include Gauss-Seidel Iteration, Successive Over-Relaxation (SOR) Method, Symmetric Successive Over-Relaxation (SSOR) Method, Jacobi Method, and Krylov Subspace Method with Adaptive Preconditioning (GMRES). Each method has different characteristics regarding convergence, computational complexity, and efficiency of computing resources. The comparison is based on several performance indicators, namely computing time, CPU usage, RAM usage, and battery usage.

This simulation was carried out on a 2D grid with a size of 50x50, which represents a groundwater flow area. Boundary conditions are determined based on hydraulic parameters common to groundwater flow models, with a predetermined head value for each boundary. The formed linear system is completed using each method, and the performance results are measured using the system's resource monitoring function. The discussion in this chapter will evaluate each method to determine the most efficient and effective method in completing this model.

Model Data and Parameters Description

The operating system used in this study is Windows 11 Home Single Language version 23H2. The device used has Intel(R) Core(TM) i7-12700H "Alder Lake" 2.30 Ghz processor specifications, DDR5 RAM 16 GB 4800MHz, NVME Gen 4x4 SSD, Intel(R) Iris(R) Xe Graphics iGPU and GPU NVIDIA GeForce RTX 3060 Laptop GPU. The programming language used is Python Programming language with the Jupyter Notebook version 7.0.8 development environment contained in Anaconda Navigator version 2.5.2.

The model domain represents a two-dimensional aquifer with dimensions of 1000 meters in both the X and Y directions. The domain is discretized into a uniform grid of 50 cells in each direction, resulting in a grid spacing of 20.41 meters. Dirichlet boundary conditions are applied along all edges of the model: the hydraulic head is fixed at 10 meters on the left boundary, 5 meters on the right boundary, 8 meters on the top boundary, and 6 meters on the bottom boundary. The interior hydraulic head values are initialized to zero before the iterative process begins. At each iteration, the hydraulic head at each internal grid cell is updated as the average of its neighboring cells. Convergence is achieved when the maximum absolute difference between hydraulic head values in successive iterations falls below a tolerance of 10^{-4}, or after 1000 iterations if convergence is not reached. The aquifer is assumed to have homogeneous and isotropic properties, and the focus of this study is on computational performance rather than specific aquifer characteristics. The assumption of a homogeneous and isotropic aquifer underpins the numerical framework, ensuring that the hydraulic head distribution depends solely on the imposed boundary conditions and not on spatially varying material properties. By simplifying the model in this way, the experiment isolates the computational aspects of solving the groundwater flow equation, such as execution time, memory usage, and convergence behavior, without the influence of additional complexities introduced by variable aquifer parameters

During the simulation, execution time, memory usage, CPU usage, and battery usage are monitored. The execution time is recorded using a timer, while memory usage is measured by observing changes in memory before and after the simulation. CPU utilization is tracked during the process, and battery consumption is noted where applicable. To ensure the robustness of the results, the simulation is repeated 100 times, and the final hydraulic head distribution is recorded from the last simulation run.

Simulation Results for Each Method

Gauss-Seidel Iteration

Figure 1 shows the hydraulic head distribution in 2d steady-state groundwater flow using gauss-seidel Iteration. The execution time required by the Gauss-Seidel method to complete this calculation is 1.83 seconds. This shows that this method is relatively quick in achieving convergence to solve the problem of discrete groundwater flow. These results correspond to the characteristics of the Gauss-Seidel method, which is known to be efficient in problems where the matrix is dominant-diagonal, such as the cases in groundwater flow modeling that often uses structured grids. However, if the size of the problem gets larger or the system matrix is less than ideal, execution time can increase significantly, so more sophisticated methods may be necessary.

The relatively low CPU usage, which is 2.40%, shows that the Gauss-Seidel method does not overload the processor. This is because arithmetic operations in this method are quite simple and do not require complicated matrix factorization. Low CPU usage makes the Gauss-Seidel method suitable for use in systems with limited CPU resources or in systems that run multiple tasks simultaneously. However, at a much larger problem size, this CPU usage can increase because this method updates elements sequentially.

A memory usage of 49.36 MB indicates that the Gauss-Seidel method is efficient in terms of memory usage. This efficiency is largely due to the use of sparse matrix representations, which allow storage of large matrices with many zero elements without taking up a lot of memory. In contrast to direct solvers, which store the matrix in the form of dense, iterative methods such as Gauss-Seidel work in a more memory-efficient way, especially on sparse matrices. This makes it an ideal choice for large simulations on devices with memory limitations.

Battery usage of 0% indicates that the computation is completed in a short enough time that it does not cause a measurable power drop, or that this computation is performed on a device that has low power consumption. This low battery usage is also related to low CPU and memory usage during the iteration process. This efficiency is advantageous for portable devices, as the Gauss-Seidel method can solve problems like this without significantly draining battery power.

These results show that the Gauss-Seidel method excels as an efficient iterative solver in terms of computing, resource usage, and power consumption. Its performance is ideal for systems with limited computing resources, such as devices with limited processors and memory, or for applications that require power efficiency on mobile devices.

Although the Gauss-Seidel method is quite effective in solving this problem, it has limitations. In the case of larger, complex matrices, or if convergence is slow, these methods may not be as efficient as more sophisticated iterative methods, such as the Successive Over-Relaxation (SOR) method or the Krylov subspace method (e.g. GMRES or Conjugate Gradient). These methods generally have higher convergence speeds, but can require more CPU or memory, so their use needs to be tailored to computing needs and system conditions.

Overall, the Gauss-Seidel method provides a good balance between speed and resource efficiency for this groundwater flow problem. However, its effectiveness needs to be re-evaluated if the problem size increases or if it is used on larger, more complex simulations.

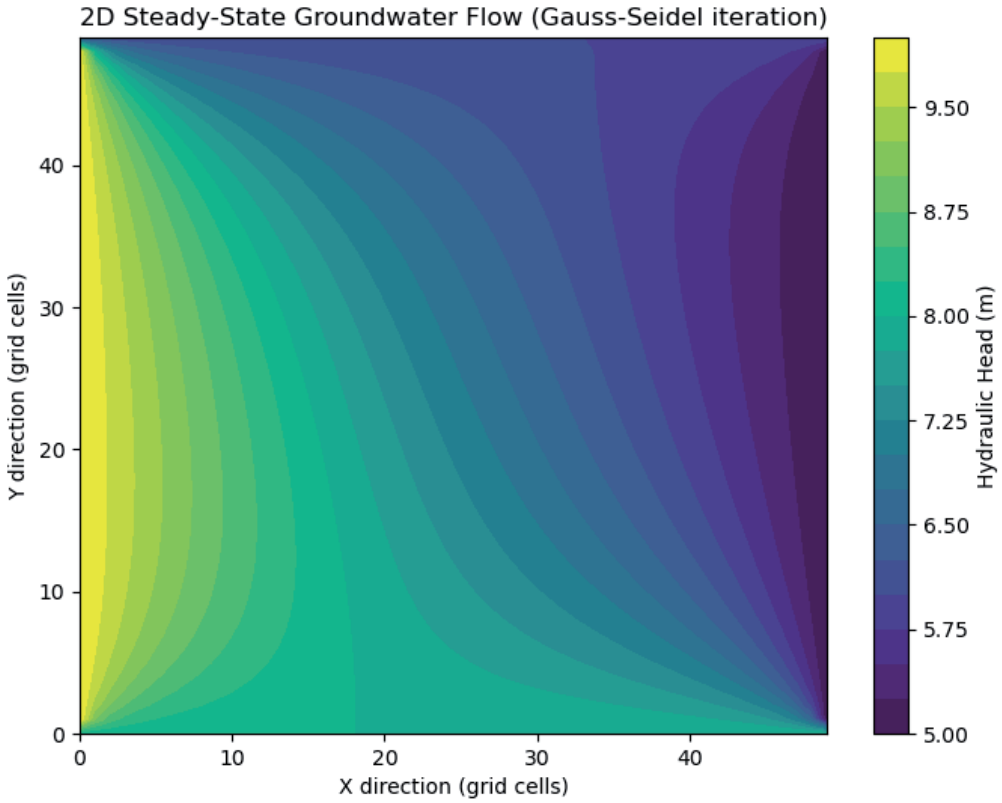

Figure 1. Hydraulic Head Distribution in 2D Steady-State Groundwater Flow using Gauss-Seidel Iteration Method.

Successive Over-Relaxation (SOR) Method

Figure 2 shows the hydraulic head distribution in 2d steady-state groundwater flow using SOR Iteration. In the performance test of the Successive Over-Relaxation (SOR) method to solve the two-dimensional steady-state groundwater flow equation, the results were obtained with an execution time of 1.45 seconds, CPU consumption of 0.70%, memory usage of 542.12 MB, and battery usage of 1%.

The relatively fast execution time, which is only 1.45 seconds, shows that the SOR method can speed up the iteration process to achieve the desired solution compared to standard iterative methods such as Gauss-Seidel. The advantage of SOR in accelerating convergence is based on the additional relaxation factor, which helps the solution get closer to the real value more quickly in a linear equation system.

The low CPU usage, which is only 0.70%, indicates that this method does not put too much strain on the processor during execution. This can potentially be advantageous in operating systems that require power efficiency or in systems with limited processor resources. On the other hand, memory usage reaches 542.12 MB, which is quite significant. This can be due to the use of data structures in the form of arrays or matrices that store information for two-dimensional grids in the simulation domain, thus requiring greater memory capacity especially at high grid resolutions.

Finally, battery usage of 1% indicates that this method slightly affects the power consumption of the device, especially if it is run on a laptop or battery-dependent device. This low battery usage can be considered quite efficient, so the SOR method can be a good choice for computing on portable devices, as long as the use of large enough memory is not an obstacle.

Overall, the SOR method offers a good balance between fast execution times and CPU efficiency, although it requires a larger memory capacity. The use of this method is suitable for numerical calculations with fast execution time requirements, especially in environmental or system simulations with limited computing resources.

In the comparative simulation of Gauss-Seidel and Successive Over-Relaxation (SOR) methods to solve the two-dimensional groundwater flow problem under solid conditions, the execution results show that there is a significant difference in the performance of these two methods in terms of execution time, CPU usage, memory usage, and battery usage.

The Gauss-Seidel method takes 2.15 seconds to execute, with 0.50% CPU usage, 1.12 MB memory usage, and no visible impact on battery percentage (0%). On the other hand, the SOR method results in a faster execution time of 1.45 seconds, with a slightly higher CPU usage of 0.70%, and a much larger memory usage of 542.12 MB. Battery usage in the SOR method also increased to 1%, which indicates additional power consumption compared to the Gauss-Seidel method.

In terms of execution time, the SOR method is more efficient because it uses an additional relaxation approach to accelerate solution convergence. This led to a reduction in execution time of almost 33% faster compared to the Gauss-Seidel method. SOR effectively reduces the number of iterations required to achieve convergence because it is able to "jump" further towards a solution, compared to Gauss-Seidel's more conservative iterative approach.

However, the memory usage in SOR is much higher (542.12 MB) than Gauss-Seidel (1.12 MB). This is likely due to the additional need to store more data in the relaxation iteration process, especially if there is the use of additional matrix or vector structures in the computational stage. Although SOR is faster, it is more memory-intensive and may be less ideal if applied to devices with limited memory or for larger problems.

CPU usage in SOR has also increased, namely by 0.70%, compared to Gauss-Seidel which only uses 0.50%. This suggests that SOR, while faster, requires slightly more intensive CPU processing, likely due to the additional relaxation calculations that need to be done at each iteration step. However, the difference in CPU usage is still relatively low so it is not a big obstacle for modern devices.

In terms of battery usage, SOR showed an increase of 1% compared to the Gauss-Seidel method which showed no impact on battery power. This may be due to the combination of higher CPU and memory usage in the SOR method, which consumes more energy in its computational process.

Overall, the SOR method is more effective in execution time than the Gauss-Seidel method due to its higher convergence speed. However, this method has drawbacks in higher memory consumption, as well as a small increase in CPU and battery usage. The choice between these two methods should consider the need for time efficiency or the limitations of available system resources. If time is a factor, SOR could be a better choice, while for applications that are more memory and power sensitive, Gauss-Seidel could be a better choice.

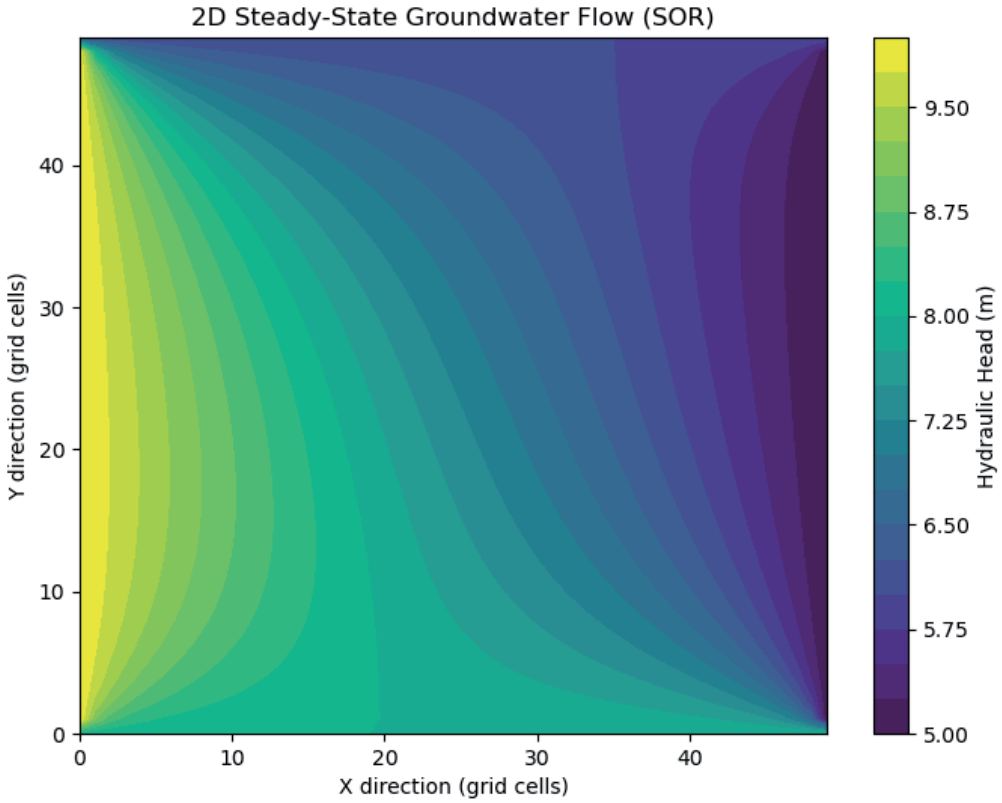

Figure 2. Hydraulic Head Distribution in 2D Steady-State Groundwater Flow using SOR Method.

Symmetric Successive Over-Relaxation (SSOR) Method

Figure 3 shows the hydraulic head distribution in 2d steady-state groundwater flow using SSOR Iteration. In the test of the Symmetric Successive Over-Relaxation (SSOR) method to solve the two-dimensional steady-state groundwater flow equation, the results were obtained with an execution time of 2.22 seconds, CPU consumption of 2.50%, memory usage of 25.05 MB, and no additional battery usage during execution.

The execution time of 2.22 seconds indicates that the SSOR method takes longer than the SOR method, but it can still be considered quite efficient. The SSOR method, which is an extension of SOR with a symmetrical approach, improves stability and convergence, especially for large and complex linear equation systems. Although more stable, this method tends to require a longer execution time due to the existence of two phases of iteration per step (forward and backward decomposition).

The CPU usage of 2.50% shows that the SSOR method is quite demanding in terms of processor load. This higher CPU usage indicates that the SSOR method involves more computational processes per iteration than the SOR method, along with its goal of improving the stability of the solution. In certain cases or devices, this can have an impact on an increase in processor temperature or power consumption.

In contrast, the memory usage of 25.05 MB is relatively low, indicating that the SSOR method is relatively memory-efficient although more stable. This can be an advantage in computing environments with memory limitations or when domain models require higher grid resolutions.

Finally, the absence of additional battery consumption in this test is an advantage for the SSOR method. This suggests that this method, although slightly more CPU-intensive, does not have a significant impact on the power of battery-powered devices.

Overall, the SSOR method provides a balance between stability and computational efficiency with low memory usage. This method is suitable for linear equation systems that require higher stability, especially when memory resources are limited or when simulations are to be performed with minimal risk to the stability of the results.

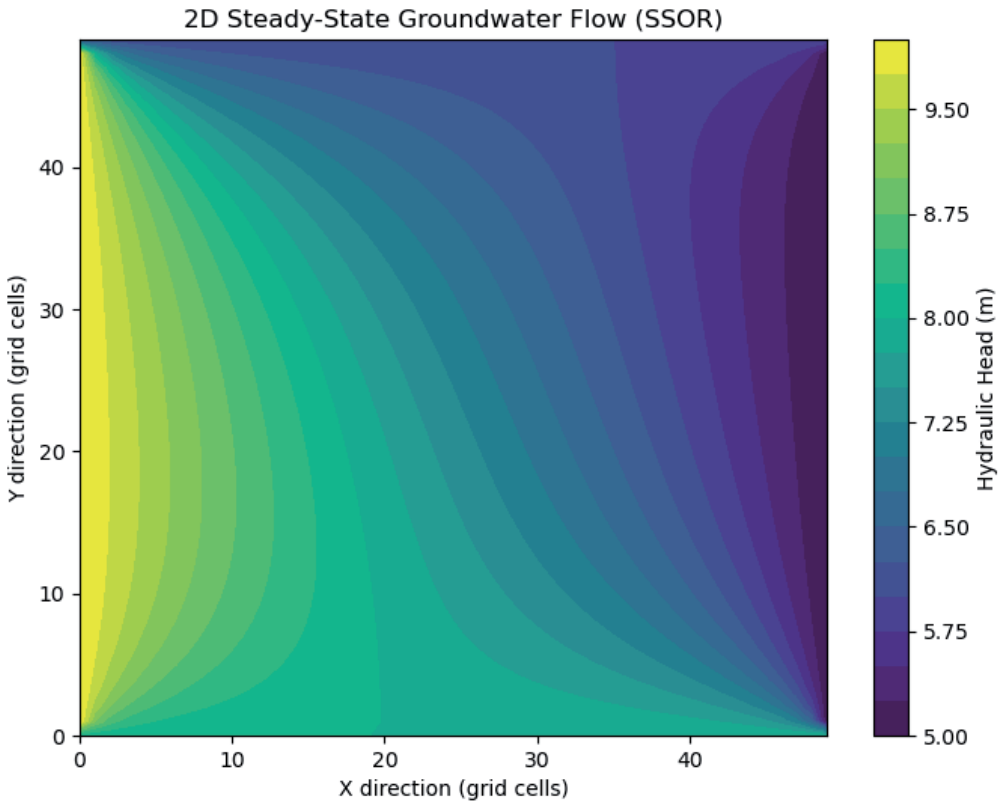

Figure 3. Hydraulic Head Distribution in 2D Steady-State Groundwater Flow using SSOR Method.

Jacobi Method

Figure 4 shows the hydraulic head distribution in 2d steady-state groundwater flow using Jacobi Iteration. Testing of the Jacobi method to solve a two-dimensional steady-state groundwater flow equation shows that the required execution time is 2.77 seconds, with a CPU usage of 2.20%, memory usage of 34.35 MB, and no additional battery usage during execution.

The relatively high execution time of 2.77 seconds suggests that the Jacobi method takes longer than some other iterative methods, such as SOR. The Jacobi method iterates by simultaneously updating the value of each point based on the value from the previous iteration, which requires more steps to achieve convergence. This generally results in longer execution times, especially in large or complex equation systems.

The CPU usage of 2.20% indicates that the Jacobi method requires a fairly high computational load. This larger CPU usage is due to the many iterations that the Jacobi method requires to approach a convergent solution, given the nature of this method that updates all elements in parallel in each iteration.

The memory usage of 34.35 MB is relatively higher than some other methods, such as SSOR. This is because the Jacobi method stores the results of each iteration separately before updating the solution, so memory usage increases to store temporary values. This can be a limitation in memory-constrained computing, especially if the grid size of the domain model is enlarged.

On the other hand, battery usage of 0% indicates that this method does not have a significant impact on power consumption on battery-powered devices. Although it uses a higher CPU and memory, the Jacobi method does not directly strain power, making it suitable for portable devices in certain contexts.

Overall, the Jacobi method tends to be slower and more intensive in terms of memory and CPU usage, but still has advantages in simplicity and parallelism. This method is more suitable for systems with simple computing needs or as a basic comparison method, but in applications that require efficient computing time, it is less optimal when compared to other iterative methods that are faster and resource-efficient.

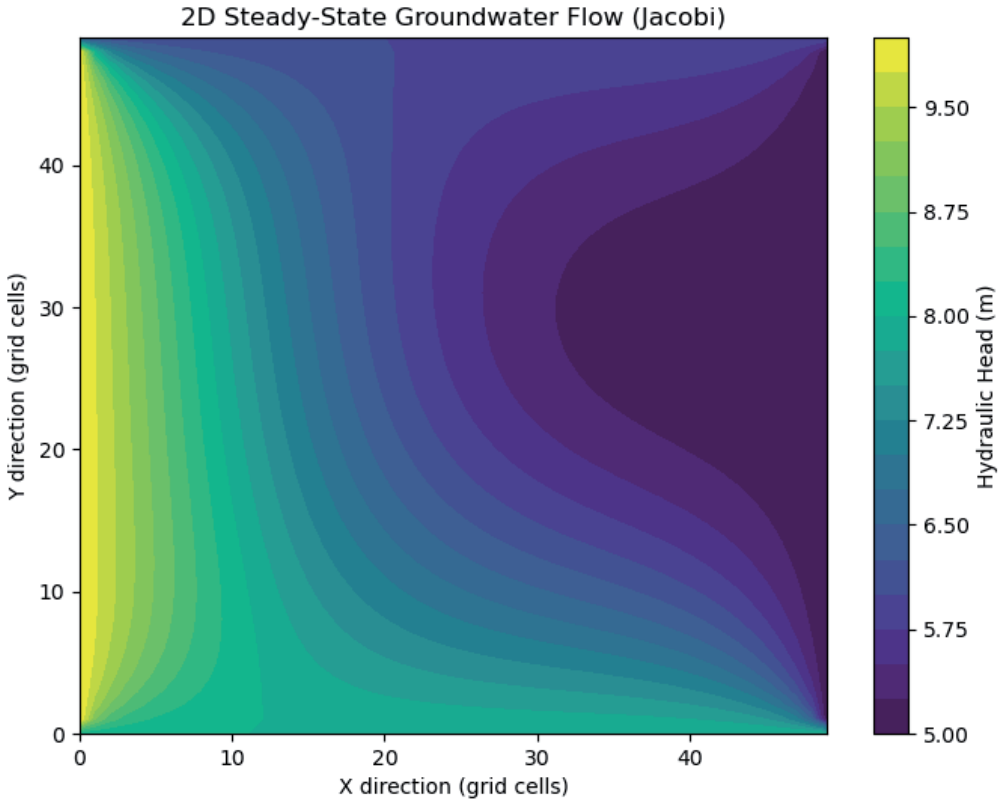

Figure 4. Hydraulic Head Distribution in 2D Steady-State Groundwater Flow using Jacobi Method.

Krylov Subspace Method with Adaptive Preconditioning (GMRES)

Figure 5 shows the hydraulic head distribution in 2d steady-state groundwater flow using Krylov Subspace Method with Adaptive Preconditioning (GMRES).The results obtained from the Krylov method with adaptive preconditioning show that this computational process is highly efficient in terms of execution time and resource usage. The execution time was recorded at only 0.0054 seconds, which is very short to complete the calculations on the large equation system that usually appears in two-dimensional groundwater flow simulations. This fast execution time indicates that the Krylov method with adaptive preconditioning is able to increase convergence and decrease the number of iterations required to achieve a solution with the desired tolerance level.

CPU usage during this process was recorded at 0.0%, which indicates that this method does not put a significant strain on the CPU. This is advantageous for applications on CPU-limited devices or for concurrent use with other processes, where there is no significant competition for CPU resources.

In terms of RAM usage, this method only requires about 0.175 MB of memory, which is very low. This low RAM usage shows efficiency in memory allocation for sparse matrix data storage and calculations in this method. With low RAM usage, this method is ideal for running on devices with limited memory capacity or in situations where multiple programs need to run simultaneously.

Battery power usage was recorded at 0%, indicating that this method does not consume significantly power or is at least so low that it has no visible effect on the percentage of battery power remaining. This is especially beneficial in field computing applications or the use of battery-powered devices.

Overall, the Krylov Method with adaptive preconditioning shows highly efficient performance with minimal resource usage. This makes this method suitable for use in simulations that require fast and effective completion of large equation systems, especially on devices with limited resource capacity.

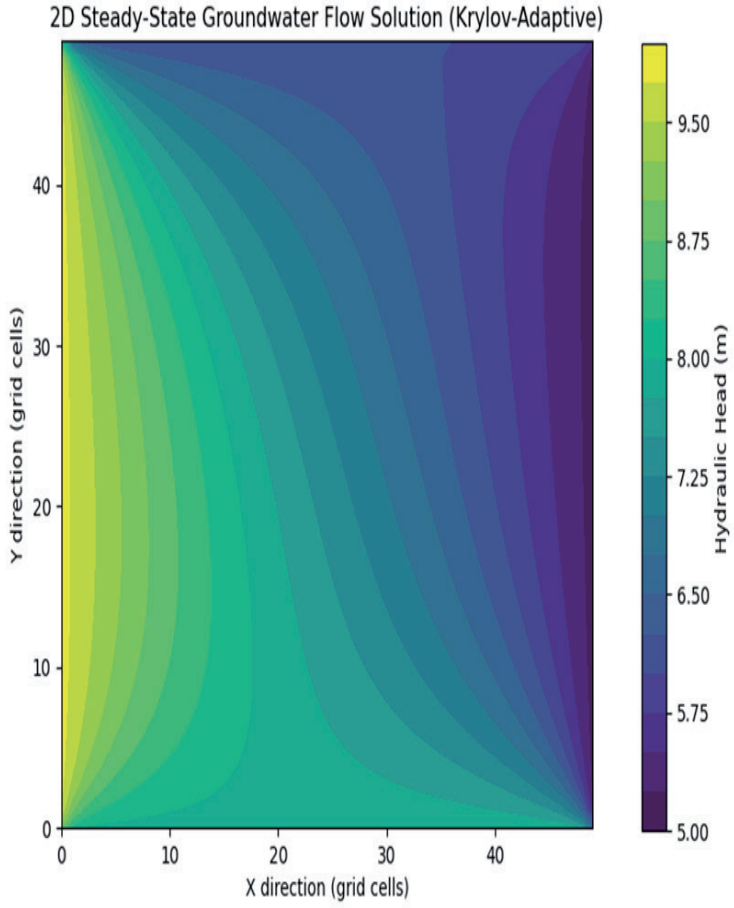

Figure 5. Hydraulic Head Distribution in 2D Steady-State Groundwater Flow using Krylov-Adaptive Method.

Method Performance Comparison

Based on the results of the performance comparison of several solution methods for two-dimensional groundwater flow problems, namely Gauss-Seidel, SOR, SSOR, Jacobi, and Krylov Method with Adaptive Preconditioning, we can analyze the aspects of computing time, CPU usage, RAM usage, and battery consumption (Figure 6).

In terms of compute time, the Krylov Method with Adaptive Preconditioning is the fastest, with an execution time of 0.0054 seconds. This time is much shorter than other methods, showing very high efficiency. The SOR method was second with a time of 1.45 seconds, followed by Gauss-Seidel with 1.83 seconds, SSOR with 2.22 seconds, and Jacobi who was the slowest with 2.77 seconds. This shows that in terms of processing speed, the Krylov Method is far superior and suitable for applications that require quick turnaround.

When viewed from the CPU usage, the Krylov Method with Adaptive Preconditioning again shows its superiority by recording CPU usage of 0.0%, which means that this method is very light for the processor. Meanwhile, the SOR method also has low CPU usage, which is 0.7%. On the other hand, the Gauss-Seidel, Jacobi, and SSOR methods each have higher CPU usage, ranging from 2.2% to 2.5%. With almost zero CPU consumption, the Krylov Method has a great advantage for running on systems that may have limited computing power or that are running other processes at the same time.

In terms of memory usage (RAM), the Krylov Method shows extraordinary efficiency with a requirement of only 0.175 MB. This makes it a very lightweight option when it comes to memory allocation. On the other hand, the SOR method requires the highest amount of RAM, at 542.12 MB, which indicates a trade-off between computing speed and memory usage. The Gauss-Seidel method (49.36 MB), Jacobi (34.35 MB), and SSOR (25.05 MB) showed more moderate RAM usage but still higher than the Krylov Method. With very low RAM usage, the Krylov Method is an ideal method to run on devices with limited memory.

Finally, from the battery consumption aspect, all methods, except SOR, show a battery consumption of 0%. The SOR method records a battery consumption of 1%, although this percentage is relatively small. This shows that these methods are overall power-efficient. However, the efficiency recorded by the Krylov Method with 0% battery power usage is also a significant advantage, especially when used on mobile devices or when battery power is limited.

Overall, this analysis shows that the Krylov Method with Adaptive Preconditioning is the most efficient method of all aspects: compute time, CPU usage, RAM usage, and battery consumption. Its excellent time efficiency makes it suitable for applications that require rapid completion of large equation systems, such as in complex groundwater flow modeling or in situations that require real-time computing. Although SOR is also relatively fast, the large RAM requirement is a limiting factor. Meanwhile, the Gauss-Seidel, Jacobi, and SSOR methods are comparatively slower with greater CPU and RAM usage. As such, the Krylov Method with Adaptive Preconditioning is the most ideal choice for practical applications on resource-constrained devices or in situations that require efficient and fast computing completion.

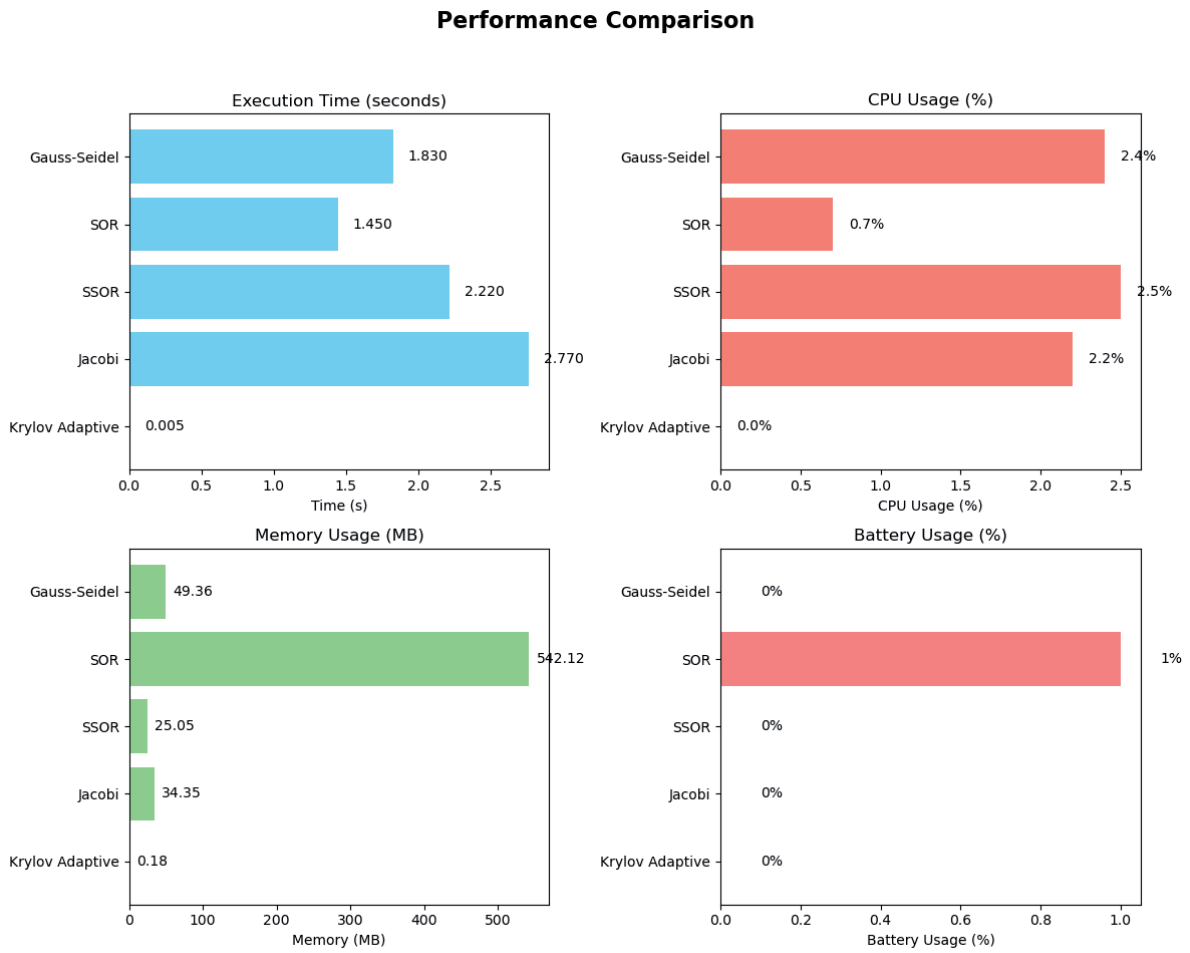

Figure 6. Performance Comparison of Numerical Methods in Groundwater Flow Modeling.

Uncertainty and sensitivity analysis

The uncertainty and sensitivity analysis was conducted on a 2D groundwater flow model using Monte Carlo simulations. The simulation involved 100 independent runs, each with randomized boundary conditions and solver tolerances. Boundary conditions for the left, right, top, and bottom boundaries were sampled uniformly from realistic, positive integer ranges ([5, 15], [1, 10], [6, 12], and [4, 10], respectively). Solver tolerance, a critical parameter influencing convergence speed and accuracy, was sampled from a uniform range (10^{-5} to 10^{-2}) to capture its impact on solver performance. Each setup was solved using the Krylov subspace method with adaptive preconditioning, recording the solution times and resulting hydraulic head distributions.

The analysis revealed that the Krylov solver's runtime exhibited moderate variability across the 100 samples. The average runtime was approximately 0.0037 seconds, with a standard deviation of 0.0011 seconds, indicating that the solver's performance was relatively robust under the tested variability of boundary conditions and tolerances. While the mean runtime suggests consistent performance, individual runtimes varied due to differences in the complexity of the system matrix A and the preconditioning effectiveness influenced by boundary conditions and tolerances.

Randomized boundary conditions significantly affected the hydraulic head distributions across the domain. Higher boundary values, particularly on the left or top edges, caused elevated hydraulic heads and steeper gradients within the flow domain. Conversely, lower boundary values resulted in flatter gradients. The solver's ability to adapt to these variations underscores its applicability to real-world systems where boundary conditions may be uncertain or spatially variable.

Solver tolerance played a critical role in determining runtime. Tighter tolerances (10^{-5}) increased runtime due to stricter convergence criteria, while looser tolerances (10^{-2}) reduced runtime but occasionally yielded less accurate solutions. This trade-off emphasizes the need to carefully select solver tolerances based on the required accuracy and computational resources. Sensitivity analysis revealed that boundary conditions had a more pronounced impact on solution distributions than solver tolerance. This indicates that the physical setup of the groundwater system is the dominant factor influencing hydraulic head patterns, while solver parameters primarily affect computational performance.

General Discussion and Implications of Results

Based on the results of the experiments that have been carried out, a comparison of the performance of various numerical methods in solving groundwater flow problems provides interesting insights into the advantages and disadvantages of each method, as well as its implications for real-time or large-scale applications.

Advantages and Disadvantages of Each Method

The Gauss-Seidel method has a fairly stable and efficient performance in memory usage. However, while it can provide consistent results, its execution time is longer compared to some other methods. This makes it less efficient in the context of modeling that requires fast computing times. On the other hand, SOR (Successive Over-Relaxation) provides better convergence speeds than Gauss-Seidel, with lower CPU consumption, but higher memory usage. This method can work better if the relaxation parameters are chosen correctly, even though its memory-consuming use is an obstacle in systems with limited memory. SSOR (Symmetric Successive Over-Relaxation) provides better stability than SOR, but requires more computing time and CPU resources. Jacobi is very simple in implementation and easy to parallelize, but it requires longer execution times and high memory consumption, making it less efficient for big problems. Finally, the Krylov Method with Adaptive Preconditioning shows outstanding performance with very fast execution times and very low resource consumption. Nonetheless, its more complex implementation makes it less ideal for users who are unfamiliar with preconditioning techniques.

Method Suitability with Real-Time or Large-Scale Applications

For applications that require fast and efficient calculations, such as in groundwater flow system modeling or other real-time applications, the Krylov Method with Adaptive Preconditioning is a very good choice. Its very short execution speed and low resource usage make it ideal for systems that require quick responses. However, in large-scale applications, methods such as SOR and SSOR can be an option if the tuning parameters are done carefully. This method is more suitable for large systems that can still tolerate longer computation times, especially if memory consumption is not a major issue. For applications that utilize parallel processors, Jacobi may have potential, given the ease of parallelization it has. Nonetheless, the Krylov method remains the best choice in environments with efficient memory and CPU requirements.

Implications for Further Research and Development

Further research can be focused on parameter optimization in SOR and SSOR methods to improve computational time efficiency and memory usage. Given that the selection of relaxation parameters greatly influences the performance of this method, further research in terms of the selection of more adaptive parameters would be very beneficial. In addition, the Krylov Method can be continuously developed by looking for more efficient preconditioners, to improve its convergence and performance on different types of problems. The use of adaptive preconditioners can be an important focus in this

study, so that this method can be increasingly competitive for a wider range of applications. For Jacobi's method, development in terms of parallelization can speed up execution and reduce computation time, especially on systems with parallel processors or GPUs. In addition, the development of energy-efficient algorithms will also be beneficial, especially in battery-based applications or mobile devices that require higher energy efficiency. By continuously improving the efficiency of parameters, parallelization, and utilization of the latest technologies, these methods can be further optimized for real-world applications.

The implementation of Krylov Subspace methods, while computationally efficient and robust for solving large sparse systems, presents notable challenges when applied to more complex or irregularly shaped systems. First, the convergence rate of Krylov Subspace methods is highly sensitive to the spectral properties of the coefficient matrix. In systems with highly irregular geometries or heterogeneous material properties, the eigenvalue distribution may become skewed, leading to slower convergence or, in extreme cases, divergence. This issue often necessitates advanced preconditioning strategies, which can be computationally expensive to construct and may not always guarantee significant improvements. Second, complex geometries often result in the discretization of governing equations that produce ill-conditioned matrices. The conditioning of these matrices directly impacts the stability and accuracy of the iterative solutions. While Krylov methods can mitigate some of these issues, their performance heavily depends on the quality of the preconditioner and the ability to retain numerical stability over iterations. Finally, the scalability of Krylov Subspace methods in massively parallel computing environments remains a challenge. Communication overhead and load balancing become critical as the number of processors increases, particularly in systems characterized by non-uniform grid distributions or anisotropic properties. Addressing these challenges requires sophisticated domain decomposition techniques and optimized parallel algorithms, which may add layers of complexity to the implementation.

Despite these limitations, Krylov Subspace methods remain a powerful tool for solving large-scale systems, provided that the inherent challenges are addressed through tailored computational strategies and problem-specific optimizations. Further research into adaptive preconditioners and advanced parallelization techniques is essential to extend the applicability of these methods to increasingly complex systems.

Overall, the results of these experiments show that the Krylov Method with Adaptive Preconditioning is the best choice for applications that require fast computing times and efficient use of resources. However, each method still has advantages that can be leveraged in the right context, depending on memory requirements, computing time, and overall system efficiency. Further research in the field of optimization and parallelization may expand the application of this method in large-scale problem-solving in the future.

Krylov subspace methods, showcased in this groundwater flow modeling study, have versatile applications beyond hydrology due to their efficiency in solving large sparse linear systems. In physics, they are widely employed for simulating quantum systems, solving heat transfer equations, and studying fluid dynamics, where matrices with high dimensionality often arise. In optimization, Krylov methods underpin iterative solvers for linear and nonlinear systems, such as those encountered in Newton-based algorithms and large-scale linear programming. They are also critical in structural engineering for finite element analysis, enabling efficient computation of stress-strain responses in complex structures. Furthermore, these methods are extensively used in computational electromagnetics for solving Maxwell's equations and in acoustics for modeling wave propagation. Their adaptability and computational efficiency make Krylov methods a cornerstone of scientific computing across disciplines.

CONCLUSION

Based on the results and discussion above, the GMRES method with adaptive preconditioning shows the best overall performance, but this method is more suitable for devices with adequate computing resources. SSOR and SOR can be used for applications on power-constrained devices without sacrificing too much compute time. The Gauss-Seidel and Jacobi methods are more suitable for models that do not require fast computing times.

REFERENCES

Aksoylu, B., Klie, H., 2009. A family of physics-based preconditioners for solving elliptic equations on highly heterogeneous media. Applied Numerical Mathematics 59, 1159–1186. https://doi.org/https://doi.org/10.1016/j.apnum.2008.06.002

Bair, E.S., 2016. Applied Groundwater Modeling-Simulation of Flow and Advective Transport. Groundwater. https://doi.org/10.1111/gwat.12464

Benali, A., 2013. Groundwater modelling: Towards an estimation of the acceleration factors of iterative methods via an analysis of the transmissivity spatial variability. Comptes Rendus Geoscience 345, 36–46. https://doi.org/https://doi.org/10.1016/j.crte.2012.11.001

Bujurke, N.M., Kantli, M.H., 2021. Numerical Simulation of Influence of Surface Features on the Elastohydrodynamic Lubrication of Sliding Line Contact Using Krylov Subspace Method. Computational Mathematics and Modelling 32, 198–220. https://doi.org/10.1007/s10598-021-09526-x

Domenico, P.A., Schwartz, F.W., 1990. Physical and chemical hydrogeology.

Downar, T.J., Joo, H.G., 2001a. A preconditioned Krylov method for solution of the multi-dimensional, two fluid hydrodynamics equations. Ann Nucl Energy 28, 1251–1267. https://doi.org/https://doi.org/10.1016/S0306-4549(00)00124-9

Downar, T.J., Joo, H.G., 2001b. A preconditioned Krylov method for solution of the multi-dimensional, two fluid hydrodynamics equations. Ann Nucl Energy 28, 1251–1267. https://doi.org/https://doi.org/10.1016/S0306-4549(00)00124-9

Fetter, C.W., 2001. Applied Hydrogeology Fourth Edition, Applied Hydrogeology. https://doi.org/0-13-088239-9

Fitts, C.R., 2013. Groundwater Science, Groundwater Science. https://doi.org/10.1016/C2009-0-62950-0

Hashimoto, Y., Nodera, T., 2022. A preconditioning technique for Krylov subspace methods in RKHSs. J Comput Appl Math 415, 114490. https://doi.org/https://doi.org/10.1016/j.cam.2022.114490

Il'in, V.P., 2021. Iterative Preconditioned Methods in Krylov Spaces: Trends of the 21st Century. Computational Mathematics and Mathematical Physics 61, 1750–1775. https://doi.org/10.1134/S0965542521110099

Jackson, J.M., 2007. Hydrogeology and groundwater flow model, central catchment of bribie island, southeast queensland 1–134.

Jain, S., 2014. Groundwater, in: Springer Geology. https://doi.org/10.1007/978-81-322-1539-4_9

Jolivet, P., Roman, J.E., Zampini, S., 2021. KSPHPDDM and PCHPDDM: Extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners. Computers & Mathematics with Applications 84, 277–295. https://doi.org/https://doi.org/10.1016/j.camwa.2021.01.003

Kelbe, B.E., Taylor, R.H., Haldorsen, S., 2011. Groundwater hydrology, in: Ecology and Conservation of Estuarine Ecosystems: Lake St Lucia as a Global Model. https://doi.org/10.1017/CBO9781139095723.010

Kuether, R.J., Steyer, A., 2024. Large-scale harmonic balance simulations with Krylov subspace and preconditioner recycling. Nonlinear Dyn 112, 3377–3398. https://doi.org/10.1007/s11071-023-09171-6

Nugraha, G.U., Bakti, H., Lubis, R.F., Nur, A.A., 2024. Jakarta groundwater modeling: a review. Appl Water Sci 14, 186. https://doi.org/10.1007/s13201-024-02168-5

Quillen, P., Ye, Q., 2010a. A block inverse-free preconditioned Krylov subspace method for symmetric generalized eigenvalue problems. J Comput Appl Math 233, 1298–1313. https://doi.org/https://doi.org/10.1016/j.cam.2008.10.071

Quillen, P., Ye, Q., 2010b. A block inverse-free preconditioned Krylov subspace method for symmetric generalized eigenvalue problems. J Comput Appl Math 233, 1298–1313. https://doi.org/https://doi.org/10.1016/j.cam.2008.10.071

Reilly, T.E., Harbaugh, A.W., 2004. Guidelines for Evaluating Ground-Water Flow Models. Scientific Investigations Report 2004-5038 37. https://doi.org/10.1017/CBO9781107415324.004

Remson, I., 1982. Introduction to Groundwater Modeling: Finite Difference and Finite Element Methods. Eos, Transactions American Geophysical Union 63, 778. https://doi.org/https://doi.org/10.1029/EO063i037p00778-02

Sabaté Landman, M., Jiang, J., Zhang, J., Ren, W., 2024. Augmented flexible Krylov subspace methods with applications to Bayesian inverse problems. Linear Algebra Appl. https://doi.org/https://doi.org/10.1016/j.laa.2024.05.007

Todd, D.K., Mays, L.W., 2005. Groundwater hydrology. Wiley

Wang, H.F., Anderson, M.P., 1982. Introduction to Groundwater Modeling: Finite Difference and Finite Element Methods.