Application of Linear Algebra and Matrix Decomposition in Optimizing Sudoku Solutions

Muhammad Fithra Rizki, 13523049[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523049@std.stei.itb.ac.id

***Abstract*—Sudoku is one of the games that can sharpen the brain of its players. Sudoku is divided into several levels so that there are easy sudoku levels and very difficult levels to solve. However, many players still find it difficult to find sudoku solutions, both at the easiest and most difficult levels. To make it easier for players, a system can be created to solve the sudoku game. This system will see the existing sudoku problem and can immediately provide the correct solution to the problem.**

***Keywords*—Sudoku, Matrix Decomposition, Linear Programming, Matrix**

I. Introduction

Sudoku, also known as Number Place, is a brain-sharpening puzzle game invented in 1973 by a Swiss mathematician named Leonhard Ueler. Sudoku itself was historically not created for a game, but rather a mathematical problem. Ueler, as the inventor, created a problem where he wanted to arrange different sets of numbers in rows and columns.

The concept was then developed by an American architect named Howard Grans who published modern sudoku in a magazine in 1979 called *Dell Pencil Puzzles and Word Games*. From here, the sudoku game became increasingly popular worldwide.

The history of the name sudoku began when Japan had a magazine, called Nikoli magazine, in 1986. A phrase was created, namely "Suuji wa dokushin ni kagiru" which means the number must be single, according to the rules of playing sudoku which must have different numbers in a row and column. But the name was too long so it had to be shortened to "Su" which means numbers and "Doku" which means single. However, in fact in that year, sudoku was not a popular game in Japan. This made Nikoli make a rule, namely that these numbers must be in a symmetrical pattern. Then, sudoku was popular with many people.

Sudoku became more global when a retired judge, Wayne Gould, created a sudoku program called Pappocom Sudoku which mass-produced various sudoku problems in order to attract the American and British markets. The British newspaper, *The Times*, was introduced to the sudoku game at that time and appeared in the newspaper on November 12, 2004. The same thing was done on an American newspaper, *The Conway Daily* Sun. This made Wayne Gould one of the most influential people in the world by *Time Magazine* at that time. Now, an international sudoku day is made which falls on September 9 because it matches the shape of sudoku, which is 9x9.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Fig 1.1 A sudoku layout generated by GNU program, Su Doku Solver (Source : Tim Stellmach)

II. Theoretical Basis

A. Matrix

A matrix of size m x n is a matrix that has m rows and n columns.

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Fig 2.1 Matrix m x n
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

The size of the matrix itself is called as an order, If the m and the n have the same value, the matrix will be called a square matrix with the value of order n.

A submatrix of a matrix is a matrix obtained by deleting certain rows or columns from a main matrix.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 2 & 1 & 6 & 8 \\ 2 & 4 & 5 & 9 & 1 & 3 \\ 8 & 2 & 4 & 3 & 2 & 1 \\ 3 & 7 & 6 & 4 & 1 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 2 & 1 \\ 4 & 5 & 9 \\ 2 & 4 & 3 \end{bmatrix}, C = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 8 \\ 3 \end{bmatrix}, D = [3 \quad 7 \quad 6 \quad 4 \quad 1 \quad 2].$$

Fig 2.2 Example of submatrix. B, C, and D are submatrix of A

(Source : Review Matrix Slides by Dr. Rinaldi Munir)

The leading diagonal of a square matrix (n x n matrix) is the set of values of the matrix A_{ij} with the value of i equal to the value of j. The leading diagonal of a matrix can be determined if and only if the matrix is square, in other words, an m x n matrix (with m not equal to n), does not have a principal diagonal.

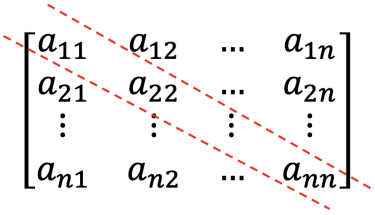


Fig 2.3 The leading diagonal of matrix n x n
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

Matrices have several special operations that can be applied to one or more matrices. The operations are as follows:

1. Matrix Addition/Subtraction

Matrix has the property of addition, that is, if there are two matrices, addition is done by adding the values in the matrix with the same column and row positions. For example, C is the addition of matrices A and B, then $A_{ij} + B_{ij} = C_{ij}$.

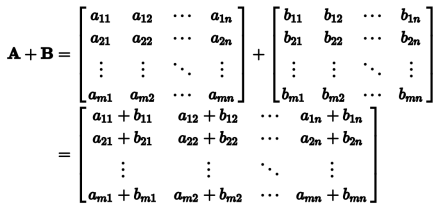


Fig 2.4 Example of two matrices addition
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

Subtraction in matrix has the same system as addition. For example, C is the addition of matrices A and B, then $A_{ij} - B_{ij} = C_{ij}$.

2. Matrix Multiplication

Matrix multiplication is divided into two types, namely multiplication between two matrices and multiplication of a matrix by a scalar.

Multiplication of a matrix by a scalar is done by multiplying the scalar value of each element in the matrix. For example, there is a scalar value c multiplied by a matrix M, then each element M_{ij} in the matrix multiplied by c produces cM_{ij}.

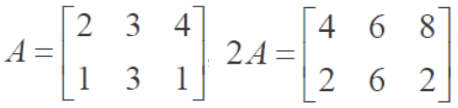
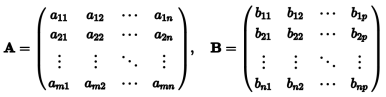


Fig 2.4 Example of multiplication of a matrix by a scalar
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

Then, there is multiplication between two matrices. This multiplication is done by multiplying each row of one matrix by each column of the other matrix and then adding them. For example, there are matrices A and B that are multiplied into matrix C. There is also a special rule for matrix multiplication, where the number of columns of matrix A must be the same as the number of rows of matrix B. If matrix A has an order of m x n and matrix B has an order of n x p, then matrix C will have an order of m x p. So, each row in matrix A is multiplied by each column in matrix B and added for each of each multiplication.



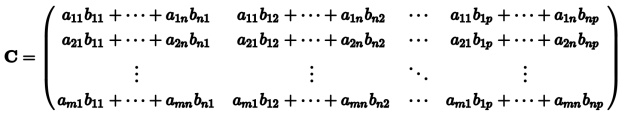


Fig 2.5 Multiplication of 2 matrices
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

3. Linear Combination of Matrix

A matrix multiplication can also be viewed as a linear combination. For example, there is a matrix A of size m x n and a matrix x of size n x 1. Each column of the matrix A will be multiplied by the value of the matrix x starting from x_1 to x_n.

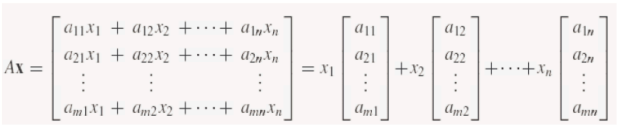


Fig 2.6 Example of Linear Combination of Matrix
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

4. Transpose Matrix

Matrix transpose is an operation performed by exchanging elements in a matrix, namely the value m_{ij} will exchange itself with the element m_{ji} . If a matrix has an order of m x n, after the transpose operation is performed it will have a change in order to n x m. For example, a matrix A has a transpose matrix A^T containing the elements that have been exchanged.

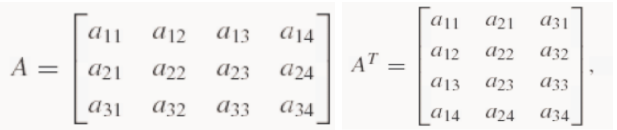


Fig 2.7 Example of Transpose Matrix
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

The transpose matrix also has some special properties that are similar to ordinary matrices as follows :

a. $(A^T)^T = A$
b. $(A + B)^T = A^T + B^T$
c. $(A - B)^T = A^T - B^T$
d. $(kA)^T = kA^T$
e. $(AB)^T = B^T A^T$

5. Elementary Row Operations

Elementary Row Operations are commonly performed on augmented matrices, namely a matrix that is a combination of 1 or more linear equation systems with the last column being the value of each of the linear equation systems.

$$\begin{aligned} x_1 + 3x_2 - 6x_3 &= 9 \\ 2x_1 - 6x_2 + 4x_3 &= 7 \\ 5x_1 + 2x_2 - 5x_3 &= -2 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 1 & 3 & -6 & 9 \\ 2 & -6 & 4 & 7 \\ 5 & 2 & -5 & -2 \end{bmatrix}$$

Fig 2.8 Example of Augmented Matrix
(Source : Review Matrix Slides by Dr. Rinaldi Munir)

Elementary Row Operations itself consists of three operations, including :

a. Multiplication of a row by a nonzero constant
b. Exchange of two rows
c. Addition of a row with multiples of another row

B. Matrix Decomposition

Matrices can be decomposed using many methods, but in this study only three types of decomposition will be used, namely LU decomposition, QR decomposition, and Singular Value Decomposition (SVD).

1. LU Decomposition

LU decomposition is a way to factor a matrix A into two parts, namely L and U, so that the equation A = LU is formed. There is two methods for decomposing the matrix, as follows :

a. LU-Gaussian Method

This method generally uses the Gaussian elimination method to form L and U. Here is how to do LU decomposition with LU-Gaussian:

1. Express A as A = IA
2. Operate the Gaussian elimination method on matrix A to form matrix U. The multiplier m_{ij} is placed on l_{ij} in matrix I. If the operation performed is a row swap, also perform the swap on the multipliers in matrix I.
3. After all the processes are completed, the I matrix on the left will change to the L matrix and the A matrix on the right will change to the U matrix.

b. Crout Reduction Method

This method is generally used based on the similarity between two matrices.

$$LU = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + u_{22} & l_{21}u_{13} + u_{23} \\ l_{31}u_{13} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + u_{33} \end{bmatrix} = A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$u_{11} = a_{11}, \quad u_{12} = a_{12}, \quad u_{13} = a_{13}$$

$$l_{21}u_1 = a_{21} \rightarrow l_{21} = \frac{a_{21}}{u_{11}}$$

$$l_{31}u_{11} = a_{31} \rightarrow l_{31} = \frac{a_{31}}{u_{11}}$$

$$\begin{aligned} l_{21}u_{12} + u_{22} &= a_{22} \rightarrow u_{22} = a_{22} - l_{21}u_{12} \\ l_{21}u_{13} + u_{23} &= a_{23} \rightarrow u_{23} = a_{23} - l_{21}u_{13} \end{aligned}$$

Fig 2.9 Form of the crout reduction method
(Source : Dekomposisi LU Slides by Dr. Rinaldi Munir)

From the existing decomposition form, a pattern is found for u and l in this method, so the method is to calculate the row elements starting from 1-k for rows then columns alternately.

Both methods will produce the same results for L and U.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ m_{21} & 1 & 0 & 0 \\ m_{31} & m_{32} & 1 & 0 \\ m_{41} & m_{42} & m_{43} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

Fig 2.10 LU Decomposition
(Source : Dekomposisi LU Slides by Dr. Rinaldi Munir)

2. QR Decomposition

QR decomposition is a technique for factoring an m x n matrix into two parts, namely Q and R. With the equation A = QR, then Q is an orthonormal matrix and R is an upper triangular matrix.

There is a method to produce Q and R, namely the Gram-Schmidt method. This method will view matrix A as a collection of column matrices a_1 to a_n. Then, there is a special calculation to find Q and R so that the decomposition is formed.

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{a}_1, \quad \mathbf{e}_1 = \frac{\mathbf{u}_1}{||\mathbf{u}_1||}, \\ \mathbf{u}_2 &= \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{e}_1)\mathbf{e}_1, \quad \mathbf{e}_2 = \frac{\mathbf{u}_2}{||\mathbf{u}_2||}. \\ \mathbf{u}_{k+1} &= \mathbf{a}_{k+1} - (\mathbf{a}_{k+1} \cdot \mathbf{e}_1)\mathbf{e}_1 - \cdots - (\mathbf{a}_{k+1} \cdot \mathbf{e}_k)\mathbf{e}_k, \quad \mathbf{e}_{k+1} = \frac{\mathbf{u}_{k+1}}{||\mathbf{u}_{k+1}||}. \end{aligned}$$

Fig 2.11 u and e calculation
(Source : Dekomposisi QR Slides by Dr. Rinaldi Munir)

$$\mathbf{A} \quad \mathbf{Q} \quad \mathbf{R}$$

$$\begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{bmatrix} \begin{bmatrix} \mathbf{e}_1 \cdot \mathbf{a}_1 & \mathbf{e}_1 \cdot \mathbf{a}_2 & \mathbf{e}_1 \cdot \mathbf{a}_3 \\ 0 & \mathbf{e}_2 \cdot \mathbf{a}_2 & \mathbf{e}_2 \cdot \mathbf{a}_3 \\ 0 & 0 & \mathbf{e}_3 \cdot \mathbf{a}_3 \end{bmatrix}$$

Orthogonal Unit vectors — Upper Diagonal Matrix

Fig 2.12 QR Decomposition
(Source : Dekomposisi QR Slides by Dr. Rinaldi Munir)

3. Singular Value Decomposition (SVD)

If there is a matrix A and the matrix has rank k and order m x n, then the matrix can be decomposed as follows:

$$A = U\Sigma V^T = [\mathbf{u}_1 \ \mathbf{u}_2 \ \cdots \ \mathbf{u}_k | \mathbf{u}_{k+1} \ \cdots \ \mathbf{u}_m] \left[\begin{array}{ccc|c} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k \\ \hline 0_{(m-k)\times k} & & & \end{array}\right] \left[\begin{array}{c|c} 0_{k\times(n-k)} & \\ \hline 0_{(m-k)\times(n-k)} & \end{array}\right] \left[\begin{array}{c} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_k^T \\ \hline \mathbf{v}_{k+1}^T \\ \vdots \\ \mathbf{v}_n^T \end{array}\right]$$

Fig 2.13 Singular Value Decomposition
(Source : Singular Value Decomposition (SVD) Bag. 1 by Dr. Rinaldi Munir)

The equation has the following description :

a. U is m x m matrix, Σ is m x n matrix, and V is n x n matrix
b. U contains the left singular vectors of matrix A.
c. V contains the right singular vectors of matrix A.
d. Σ contains the singular values of A arranged diagonally.

There are 2 ways to do SVD, the first one is from the theorem in the picture below.

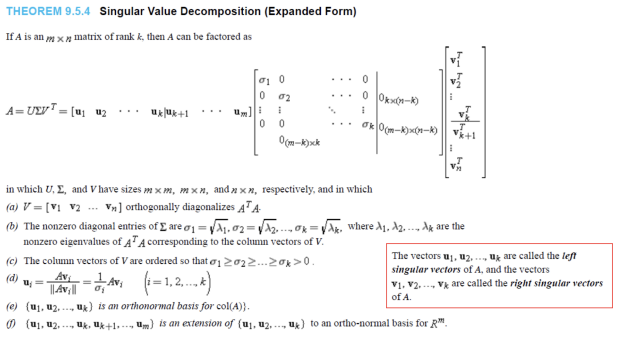


Fig 2.14 Theorem to do SVD
(Source : Singular Value Decomposition (SVD) Bag. 1 by Dr. Rinaldi Munir)

Another way is to calculate the left singular vector and the right singular vector separately, as follows:

1. Find the right singular vectors from v_1 to v_n corresponding to the eigenvalues of A^TA. Normalize v_1 to v_n by dividing each component of the vector by the length of the vector. Then, we will obtain the matrix V and transpose it. Rank(A) = k = the number of nonzero eigenvalues of A^TA
2. Determine the left singular vectors u_1 to u_k with the equation below and normalize every u.

$$\mathbf{u}_i = \frac{A\mathbf{v}_i}{\|A\mathbf{v}_i\|} = \frac{1}{\sigma_i}A\mathbf{v}_i \quad , i = 1, 2, \ldots, k$$

3. Expand u if n > k to form an orthonormal basis for R^m
4. Form a matrix of size m x n whose diagonal elements are the nonzero singular values of matrix A in ascending order. The singular values in are the square roots of the nonzero eigenvalues of A^TA.

C. Matrix in Sudoku

A sudoku matrix is a square matrix of order n with $n = m^2$ ($m \geq 2$). A sudoku matrix satisfies the following rules or properties:

a. Each row in the matrix contains positive numbers 1 through n exactly once

$$S = \begin{bmatrix} 9 & 4 & 7 & 2 & 5 & 8 & 1 & 3 & 6 \\ 1 & 2 & 3 & 4 & 6 & 7 & 9 & 8 & 5 \\ 6 & 5 & 8 & 1 & 9 & 3 & 7 & 2 & 4 \\ 8 & 9 & 5 & 6 & 4 & 2 & 3 & 7 & 1 \\ 7 & 6 & 4 & 9 & 3 & 1 & 2 & 5 & 8 \\ 3 & 1 & 2 & 8 & 7 & 5 & 4 & 6 & 9 \\ 4 & 8 & 9 & 7 & 2 & 6 & 5 & 1 & 3 \\ 2 & 3 & 6 & 5 & 1 & 9 & 8 & 4 & 7 \\ 5 & 7 & 1 & 3 & 8 & 4 & 6 & 9 & 2 \end{bmatrix}$$

Fig 2.15 Row in sudoku matrix
(Source : https://lib.ui.ac.id/detail.jsp?id=20289497)

b. Each column in the matrix contains positive numbers 1 through n exactly once

$$S = \begin{bmatrix} 9 & 4 & 7 & 2 & 5 & 8 & 1 & 3 & 6 \\ 1 & 2 & 3 & 4 & 6 & 7 & 9 & 8 & 5 \\ 6 & 5 & 8 & 1 & 9 & 3 & 7 & 2 & 4 \\ 8 & 9 & 5 & 6 & 4 & 2 & 3 & 7 & 1 \\ 7 & 6 & 4 & 9 & 3 & 1 & 2 & 5 & 8 \\ 3 & 1 & 2 & 8 & 7 & 5 & 4 & 6 & 9 \\ 4 & 8 & 9 & 7 & 2 & 6 & 5 & 1 & 3 \\ 2 & 3 & 6 & 5 & 1 & 9 & 8 & 4 & 7 \\ 5 & 7 & 1 & 3 & 8 & 4 & 6 & 9 & 2 \end{bmatrix}$$

Fig 2.16 Column in sudoku matrix
(Source : https://lib.ui.ac.id/detail.jsp?id=20289497)

c. Each submatrix of order $n^{1/2}$ that is arranged in a regular manner contains positive numbers 1 through n exactly once

$$S = \begin{bmatrix} 9 & 4 & 7 & 2 & 5 & 8 & 1 & 3 & 6 \\ 1 & 2 & 3 & 4 & 6 & 7 & 9 & 8 & 5 \\ 6 & 5 & 8 & 1 & 9 & 3 & 7 & 2 & 4 \\ 8 & 9 & 5 & 6 & 4 & 2 & 3 & 7 & 1 \\ 7 & 6 & 4 & 9 & 3 & 1 & 2 & 5 & 8 \\ 3 & 1 & 2 & 8 & 7 & 5 & 4 & 6 & 9 \\ 4 & 8 & 9 & 7 & 2 & 6 & 5 & 1 & 3 \\ 2 & 3 & 6 & 5 & 1 & 9 & 8 & 4 & 7 \\ 5 & 7 & 1 & 3 & 8 & 4 & 6 & 9 & 2 \end{bmatrix}$$

Fig 2.17 Submatrix in sudoku matrix
(Source : https://lib.ui.ac.id/detail.jsp?id=20289497)

In accordance with the properties of the sudoku matrix, there are several equations that can be applied as follows :

a. Sum of the i-th row

$$\sum_{j=1}^{n} S_{i,j} = \frac{n(n+1)}{2}, 1 \leq i \leq n$$

b. Sum of the j-th column

$$\sum_{i=1}^{n} S_{i,j} = \frac{n(n+1)}{2}, 1 \leq j \leq n$$

D. Linear Programming

Linear Programming is Linear Algebra with two additional ideas, namely inequality and minimization. The initial equation is still based on the matrix equation Ax = b, but the solution required is a non-negative value. If there are many non-negative x solutions for Ax = b, then linear programming will take the x* that requires the least cost.

The cost is $c_1x_1 + \cdots + c_nx_n$. The winning vector x^* is the nonnegative solution of $Ax = b$ that has smallest cost.

Fig 2.18 Solution from linear programming
(Source : math.mit.edu)

III. Implementation Method

This section will explain how the program that has been created in python language. Primarily, the program runs by taking input from the sudoku game in the form of a 9x9 matrix. Then, the matrix will be formed into a constraint matrix and solved with the linear programming function.

The interesting thing in this experiment is the use of

matrix decomposition in the solution search process. So, after the constraint matrix is created, the matrix will be decomposed with three types of decomposition, namely LU decomposition, QR, and Singular Value Decomposition (SVD). After that, the sudoku matrix problem can be solved with linear programming.

A. Initializing Constraints Matrix

To solve the 9x9 sudoku problem, the sudoku matrix must first be made in the form of constraints, namely in matrix A and column vector b.

In solving sudoku problems, it is known that the rule is that each row and each column must have numbers 1 to 9 with the number appearing only 1 time. From this rule, constraints need to be formed in the solution. Matrix A in the constraint will have the first 81 rows which are the constraint representation of the row rules in sudoku. Then, the column rules also produce something similar, in matrix A will have 81 additional rows that come from the sudoku matrix rules in the column.

Sudoku rules are not only limited to rows and columns. There are also rules in the 3x3 submatrix of the matrix. So, because of the submatrix rules, matrix A will have 81 additional rows as a constraint representation of the rules. Then, the thing used in forming this constraint is also from the rule that each square in sudoku can only be filled with 1 number. This adds 81 final rows to the A matrix which represent the constraints of the rule.

```python
def initialize_constraints():
    A = []
    b = []

    # Row Constraint
    for i in range(9):
        for num in range(1, 10):
            row_constraint = [0] * 729
            for j in range(9):
                row_constraint[i * 81 + j * 9 + (num - 1)] = 1
            A.append(row_constraint)
            b.append(1)

    # Column Constraint
    for j in range(9):
        for num in range(1, 10):
            col_constraint = [0] * 729
            for i in range(9):
                col_constraint[i * 81 + j * 9 + (num - 1)] = 1
            A.append(col_constraint)
            b.append(1)

    # Submatrix Constraint
    for box_row in range(3):
        for box_col in range(3):
            for num in range(1, 10):
                submatrix_constraint = [0] * 729
                for i in range(3):
                    for j in range(3):
                        r = box_row * 3 + i
                        c = box_col * 3 + j
                        submatrix_constraint[r * 81 + c * 9 + (num - 1)] = 1
                A.append(submatrix_constraint)
                b.append(1)

    # Cell Constraint
    for i in range(9):
        for j in range(9):
            cell_constraint = [0] * 729
            for num in range(1, 10):
                cell_constraint[i * 81 + j * 9 + (num - 1)] = 1
            A.append(cell_constraint)
            b.append(1)
    return np.array(A), np.array(b)
```

Fig 3.1 Implementation code of initialization constraints

From the code, the result is obtained where A is a 324 x 729 matrix with each row representing one constraint and b is a 324 x 1 dimensional column vector with all its elements equal to 1 (constraints are met if Ax = b).

B. Constraint Matrix Decomposition

The Constraints Matrix obtained from the first algorithm will then be decomposed in three ways, namely with LU, QR, and SVD. By importing scipy.linalg, the LU, QR, and SVD functions can be used instantly.

If decomposition is carried out with LU decomposition, the value that will be used is the U matrix because the U matrix stores important values from the constraint matrix A. Then, if decomposition is carried out with QR decomposition, the value used is the R matrix for the same reason as the LU decomposition, namely the R matrix stores important values from the constraint matrix A. Finally, if we use decomposition using the SVD method, the value that will be used is ΣV^T, because Σ contains singular values, which determine how significant each constraint is. Small values indicate small contributions. V^T provides a direct representation of the solution space of constraint A.

```python
def decompose_constraints(A, method):
    if method == "qr":
        Q, R = qr(A, mode='economic')
        return R  # Constraint representation with R
    elif method == "lu":
        L, U = lu(A)
        return U  # Constraint representation with U
    elif method == "svd":
        U, S, Vt = svd(A, full_matrices=False) # Constraint representation with S and Vt
        return np.diag(S) @ Vt
    else:
        print("Decomposition method invalid. Use 'qr', 'lu', or 'svd'.")
```

Fig 3.2 Implementation code of constraint matrix decomposition

C. Implementation Code for Sudoku Solving

The implementation of the code in this sudoku solver begins with a function call for constraint initiation and the values of A and b are obtained. Then, matrix A will be decomposed with the three types of decompositions that have been discussed. In order to get the appropriate results, the column vector b will also be adjusted to the matrix A that has been decomposed. For QR decomposition, b is adjusted by transposing Q multiplied by b. For LU decomposition, b is adjusted by inverse L multiplied by the transpose of the permutation multiplied by b. For SVD, b is adjusted by multiplying U transpose and b.

Then, it is necessary to check the rank consistency between the matrix A and the decomposed A. If there is no match, then the values in matrix A may disappear and the operation will not be valid to continue. It is necessary to create a bound to limit each variable x with the numbers that have been filled in the sudoku problem. For example, if a cell is already filled with a number, then the value of x in that cell is limited between 1-1.

The last thing to do is to use linear programming to find the required results. The $A_{decomposed} \cdot x = b_{decomposed}$ system will be solved with the constraints that have been made. Then, the results of the operation are converted into a 9 x 9 x 9 matrix and continued with the return to the sudoku format.

```python
def solve_sudoku(matrix, method="qr"):
    A, b = initialize_constraints()

    # Validating or preprocessing with decomposition
    print(("Using (method.upper()) Decomposition for validation...")
    A_decomposed = decompose_constraints(A, method=method)
    if method == "qr":
        Q, R = qr(A, mode='economic')
        b_decomposed = Q.T @ b  # b value adjustment for QR
    elif method == "lu":
        P, L, U = lu(A)
        b_decomposed = np.linalg.inv(L) @ (P.T @ b)  # b value adjustment for LU
    elif method == "svd":
        U, S, Vt = svd(A, full_matrices=False)
        A_decomposed = np.diag(S) @ Vt
        b_decomposed = U.T @ b  # b value adjustment for SVD
    else:
        print("Decomposition method invalid!")

    # Check if the decomposition valid
    if np.linalg.matrix_rank(A) != np.linalg.matrix_rank(A_decomposed):
        print("Decomposition causes loss of information on constraints!")
    bounds = []
    for i in range(9):
        for j in range(9):
            if matrix[i, j] != 0:
                num = matrix[i, j]
                for k in range(1, 10):
                    if k == num:
                        bounds.append((1, 1))
                    else:
                        bounds.append((0, 0))
            else:
                bounds.extend([(0, 1)] * 9)
    c = [0] * 729
    res = linprog(c, A_eq=A_decomposed, b_eq=b_decomposed, bounds=bounds, method="highs")

    if res.success:
        solution = np.array(res.x).reshape((9, 9, 9))
        solved_sudoku = np.argmax(solution, axis=2) + 1
        return solved_sudoku
    else:
        print("Sudoku can't be solved with the given constraints.")
```

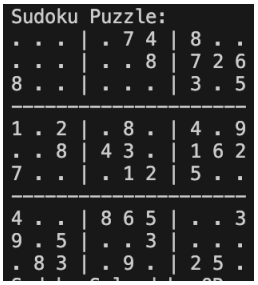
Fig 3.3 Implementation code of sudoku solver

IV. Implementation Testing and Result Discussion

A. Unit Testing

From the implementation of the code, it was decided to conduct testing using sudoku problems from the sudoku.com website. On the website, several levels of difficulty are given, namely easy, medium, hard, expert, master, and extreme. In this test, only three levels will be tested, including easy, master, and extreme.

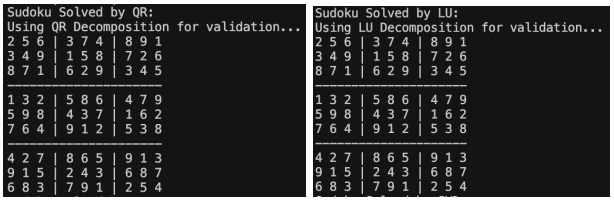
1. Easy Level



```
Sudoku Puzzle:
. . . | . 7 4 | 8 . .
. . . | . . 8 | 7 2 6
8 . . | . . . | 3 . 5
---------------------
1 . 2 | . 8 . | 4 . 9
. . 8 | 4 3 . | 1 6 2
7 . . | . 1 2 | 5 . .
---------------------
4 . . | 8 6 5 | . . 3
9 . 5 | . . 3 | . . .
. 8 3 | . 9 . | 2 5 .
```

Fig 4.1 Easy level on sudoku.com (recreated in the program)

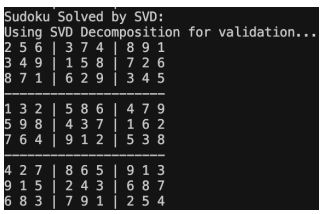
From the sudoku, solutions were found in three ways as follows:



```
Sudoku Solved by QR:
Using QR Decomposition for validation...
2 5 6 | 3 7 4 | 8 9 1
3 4 9 | 1 5 8 | 7 2 6
8 7 1 | 6 2 9 | 3 4 5
---------------------
1 3 2 | 5 8 6 | 4 7 9
5 9 8 | 4 3 7 | 1 6 2
7 6 4 | 9 1 2 | 5 3 8
---------------------
4 2 7 | 8 6 5 | 9 1 3
9 1 5 | 2 4 3 | 6 8 7
6 8 3 | 7 9 1 | 2 5 4
```



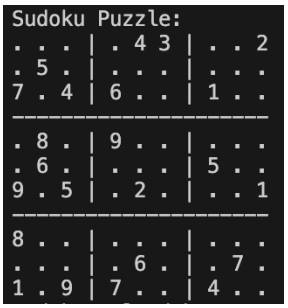
```
Sudoku Solved by LU:
Using LU Decomposition for validation...
2 5 6 | 3 7 4 | 8 9 1
3 4 9 | 1 5 8 | 7 2 6
8 7 1 | 6 2 9 | 3 4 5
---------------------
1 3 2 | 5 8 6 | 4 7 9
5 9 8 | 4 3 7 | 1 6 2
7 6 4 | 9 1 2 | 5 3 8
---------------------
4 2 7 | 8 6 5 | 9 1 3
9 1 5 | 2 4 3 | 6 8 7
6 8 3 | 7 9 1 | 2 5 4
```



```
Sudoku Solved by SVD:
Using SVD Decomposition for validation...
2 5 6 | 3 7 4 | 8 9 1
3 4 9 | 1 5 8 | 7 2 6
8 7 1 | 6 2 9 | 3 4 5
---------------------
1 3 2 | 5 8 6 | 4 7 9
5 9 8 | 4 3 7 | 1 6 2
7 6 4 | 9 1 2 | 5 3 8
---------------------
4 2 7 | 8 6 5 | 9 1 3
9 1 5 | 2 4 3 | 6 8 7
6 8 3 | 7 9 1 | 2 5 4
```

Fig 4.2 Solutions for easy level

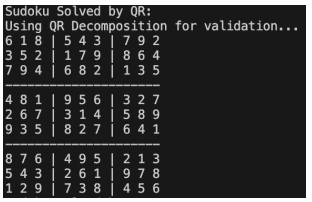
2. Master Level



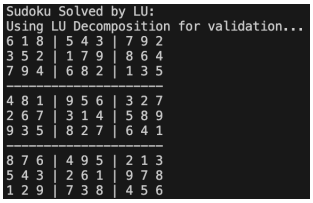
```
Sudoku Puzzle:
. . . | . 4 3 | . . 2
. 5 . | . . . | . . .
7 . 4 | 6 . . | 1 . .
---------------------
. 8 . | 9 . . | . . .
. 6 . | . . . | 5 . .
9 . 5 | . 2 . | . . 1
---------------------
8 . . | . . . | . . .
. . . | . 6 . | . 7 .
1 . 9 | 7 . . | 4 . .
```

Fig 4.3 Master level on sudoku.com (recreated in the program)

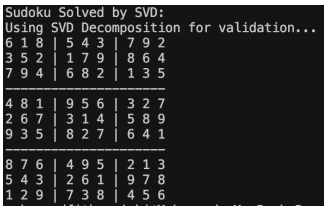
From the sudoku, solutions were found in three ways as follows:



```
Sudoku Solved by QR:
Using QR Decomposition for validation...
6 1 8 | 5 4 3 | 7 9 2
3 5 2 | 1 7 9 | 8 6 4
7 9 4 | 6 8 2 | 1 3 5
---------------------
4 8 1 | 9 5 6 | 3 2 7
2 6 7 | 3 1 4 | 5 8 9
9 3 5 | 8 2 7 | 6 4 1
---------------------
8 7 6 | 4 9 5 | 2 1 3
5 4 3 | 2 6 1 | 9 7 8
1 2 9 | 7 3 8 | 4 5 6
```



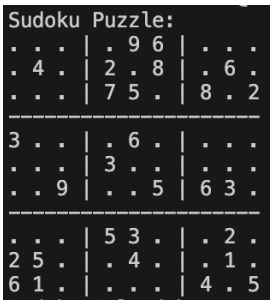
```
Sudoku Solved by LU:
Using LU Decomposition for validation...
6 1 8 | 5 4 3 | 7 9 2
3 5 2 | 1 7 9 | 8 6 4
7 9 4 | 6 8 2 | 1 3 5
---------------------
4 8 1 | 9 5 6 | 3 2 7
2 6 7 | 3 1 4 | 5 8 9
9 3 5 | 8 2 7 | 6 4 1
---------------------
8 7 6 | 4 9 5 | 2 1 3
5 4 3 | 2 6 1 | 9 7 8
1 2 9 | 7 3 8 | 4 5 6
```



```
Sudoku Solved by SVD:
Using SVD Decomposition for validation...
6 1 8 | 5 4 3 | 7 9 2
3 5 2 | 1 7 9 | 8 6 4
7 9 4 | 6 8 2 | 1 3 5
---------------------
4 8 1 | 9 5 6 | 3 2 7
2 6 7 | 3 1 4 | 5 8 9
9 3 5 | 8 2 7 | 6 4 1
---------------------
8 7 6 | 4 9 5 | 2 1 3
5 4 3 | 2 6 1 | 9 7 8
1 2 9 | 7 3 8 | 4 5 6
```

Fig 4.4 Solutions for master level

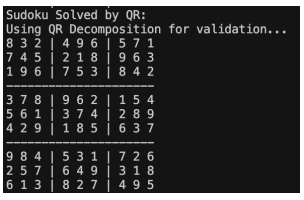
3. Extreme Level



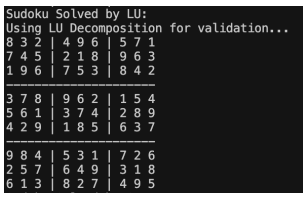
```
Sudoku Puzzle:
. . . | . 9 6 | . . .
. 4 . | 2 . 8 | . 6 .
. . . | 7 5 . | 8 . 2
---------------------
3 . . | . 6 . | . . .
. . . | 3 . . | . . .
. . 9 | . . 5 | 6 3 .
---------------------
. . . | 5 3 . | . 2 .
2 5 . | . 4 . | . 1 .
6 1 . | . . . | 4 . 5
```

Fig 4.5 Extreme level on sudoku.com (recreated in the program)

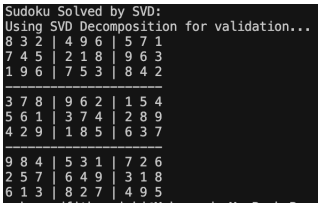
From the sudoku, solutions were found in three ways as follows:



```
Sudoku Solved by QR:
Using QR Decomposition for validation...
8 3 2 | 4 9 6 | 5 7 1
7 4 5 | 2 1 8 | 9 6 3
1 9 6 | 7 5 3 | 8 4 2
---------------------
3 7 8 | 9 6 2 | 1 5 4
5 6 1 | 3 7 4 | 2 8 9
4 2 9 | 1 8 5 | 6 3 7
---------------------
9 8 4 | 5 3 1 | 7 2 6
2 5 7 | 6 4 9 | 3 1 8
6 1 3 | 8 2 7 | 4 9 5
```



```
Sudoku Solved by LU:
Using LU Decomposition for validation...
8 3 2 | 4 9 6 | 5 7 1
7 4 5 | 2 1 8 | 9 6 3
1 9 6 | 7 5 3 | 8 4 2
---------------------
3 7 8 | 9 6 2 | 1 5 4
5 6 1 | 3 7 4 | 2 8 9
4 2 9 | 1 8 5 | 6 3 7
---------------------
9 8 4 | 5 3 1 | 7 2 6
2 5 7 | 6 4 9 | 3 1 8
6 1 3 | 8 2 7 | 4 9 5
```



```
Sudoku Solved by SVD:
Using SVD Decomposition for validation...
8 3 2 | 4 9 6 | 5 7 1
7 4 5 | 2 1 8 | 9 6 3
1 9 6 | 7 5 3 | 8 4 2
---------------------
3 7 8 | 9 6 2 | 1 5 4
5 6 1 | 3 7 4 | 2 8 9
4 2 9 | 1 8 5 | 6 3 7
---------------------
9 8 4 | 5 3 1 | 7 2 6
2 5 7 | 6 4 9 | 3 1 8
6 1 3 | 8 2 7 | 4 9 5
```

Fig 4.6 Solutions for extreme level

B. Result Discussion

From the existing testing results, it is indeed likely that all solutions work well. With the tested levels of easy, master, and extreme, it can be concluded that the implementation of the code can complete levels that have not been tested as well. However, looking at how the code works, there are several things and shortcomings that can interfere with the performance of the code, including :

1. Computational Efficiency

The resulting constraints matrix has a very large size, which is 729 x 729. This large size makes the program require a lot of time and memory. For sudoku work, of course this approach is not very efficient compared to other methods for finding sudoku solutions.

2. Handling of Singular Matrices

The matrix formed from the sudoku problem may be a singular matrix. However, the singular matrix may not be solved with the code created in this study. Some operations in decomposition and finding the results require the inverse operation of the matrix itself. So, if the existing sudoku matrix is singular or has a determinant value of 0, then the calculation will be invalid. Information on constraints may also overlap or even be lost.

3. Not So Good Optimization for Sudoku

The "highs" method in linear programming is not designed for integer programming like sudoku, but for general linear optimization. This makes it possible for inaccurate results and long processing. In addition, the code that is made is not actually designed specifically for solving sudoku, but rather for generic linear optimization.

V. Conclusion

The conclusion obtained from this study is that matrix decomposition, be it LU decomposition, QR decomposition, or Singular Value Decomposition (SVD), can be used to help solve sudoku problems, but still requires assistance from one of the developments of linear algebra, linear programming. However, solving Sudoku games with decomposition turns out to be less effective, especially in terms of time and memory used, compared to using other methods, such as backtracking.

VI. Appendix

The complete sudoku solver program and other functions used can be found below.

https://github.com/fithrarzk/Makalah-Algeo

Below is a video demonstration and explanation of this project.

https://youtu.be/CRveIHkcH7g?feature=shared

VII. Acknowledgment

All praise and gratitude are offered to the presence of the Almighty God, Allah Subhanahu wa Ta'ala, who has given the author the opportunity to complete the paper entitled "Application of Linear Algebra and Matrix Decomposition in Optimizing Sudoku Solutions". In addition, the author would like to express his deepest gratitude to the lecturer in charge of the Linear Algebra and Geometry course, Ir. Rila Mandala, M.Eng., Ph.D., for the lessons and motivation that have been given during the lecture.

References

[1] S. Salamah, "Sifat-Sifat Matriks Sudoku," Depok : Fakultas Matematika dan Ilmu Pengetahuan Universitas Indonesia, 2012.

[2] R. Munir, "Review Matriks," IF2123 Aljabar Linier dan Geometri. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf [Accessed 28 December 2024]

[3] R. Munir, "Singular Value Decomposition (SVD) (Bagian 1)," IF2123 Aljabar Linier dan Geometri. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf [Accessed 28 December 2024]

[4] R. Munir, "Dekomposisi LU," IF2123 Aljabar Linier dan Geometri. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-23-Dekomposisi-LU-2023.pdf [Accessed 28 December 2024]

[5] R. Munir, "Dekomposisi QR," IF2123 Aljabar Linier dan Geometri. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2024-2025/Algeo-23b-Dekomposisi-QR-2024.pdf [Accessed 28 December 2024]

Statement of Originality

I hereby declare that this paper is my own writing, not an adaptation, or translation of someone else's paper, and not plagiarized.

Bandung, 27 Desember 2024

Muhammad Fithra Rizki - 13523049