# Report: Solving Linear Systems with Gaussian Elimination and Image Compression using Singular Value Decomposition

**Introduction**

To tackle various practical problems with efficient and elegant solution, I’ve utilized two fundamental numerical methods taught in the class: **Gaussian Elimination without Pivoting** for solving systems of linear equations and **Singular Value Decomposition (SVD)** for image compression in Python environment. Both algorithms are explored in depth, focusing on their methodologies, implementation strategies, computational complexities, and performance considerations such as truncation errors and convergence rates.

**Part 1: Gaussian Elimination Without Pivoting**

**1.1 Methodology**

Gaussian Elimination is a standard procedure for solving systems of linear equations as we discussed. The method transforms a set of equations into an upper triangular form through a series of basic row operations such as row addition and constant multiplication, along with a straightforward application of back substitution to find the solution vector (Golub & Van Loan, 2013).

We have two primary sections in the algorithms:

**Forward Elimination**: Firstly, we eliminate the variables via converting the coefficient matrix into an upper triangular matrix. It involves selecting pivot elements and performing row operations to nullify the entries below the pivots. As required, we will not perform any row-change operation in the section.

**Back Substitution**: Once the matrix is in the upper triangular form, we will solve the value of variables starting from the last equation and moving upwards.

In this implementation, Gaussian Elimination is performed without pivoting, meaning no row exchanges are conducted to handle zero or small pivot elements. It is established that this approach simplifies the algorithm but introduces potential numerical stability issues (Higham, 2002), but in our testing with regular small square matrices with random elements, it works just fine.

**1.2 Implementation**

My implementation of Gaussian Elimination in python without pivoting is quite straight forward:

1. **Input Validation**: Firstly, verify that the coefficient matrix is square and that its dimensions match the right-hand side vector .
2. **Forward Elimination**: iterate through each pivot row; check for each pivot element , and make sure that they are non-zero. Since we are not using pivoting in the implementation, encountering a zero pivot simply stops the loop and raises an error message. If the pivot is non-zero, the algorithm will eliminate the entries below the pivot by subtracting a multiple of the pivot row from the subsequent rows.
3. **Back Substitution**: after obtaining the upper triangular form, the algorithm initializes a solution vector with zeros using . Then iteratively solves for each variable starting from the last row, using the computed value from previous iteration to determine the current variable's value.

We ensure that the system is transformed into a form suitable to compute solutions efficiently and accurately through back substitution, when there are no zero pivots encountered.

* 1. **Complexity Analysis**

**Time Complexity** : for a matrix, each pivot operation involves computations, and the producer is looped for times, leading to a time complexity of in the forward elimination. For back substitution: solving the upper triangular system requires operations regarding substitution, multiplication, and additions. Overall, dominated by the forward elimination, the total time complexity would be .

**Space Complexity** :The primary storage needs are specifically for the coefficient matrix and the right-hand side vector , requiring and space respectively. Additional space is utilized for intermediate computations and the solution vector . Summing them up brings us a space complexity of .

**1.4 Error Handling and Numerical Stability**

Implementing Gaussian Elimination without pivoting introduces difficulties regarding the handling of zero pivot and certain numerical stability.

The absence of row-change operations makes zero-pivot element () non-applicable to our algorithms. When meeting one, we will have to return and raise an error message. It is possible to utilize pivoting strategies to improve our algorithm, yet it requires more complicated handling of results when there exist non-deterministic solutions, especially when the number of pivots is smaller than the dimension and we obtain infinite solutions. Nonetheless, strategies such as partial or full pivoting, are typically employed in practice to rearrange the rows and avoid zero or small pivot elements, thereby enhancing the algorithm's robustness (Strang, 2009).

Without using pivoting, the method is susceptible to cases that lead to numerical instability. Especially in cases where pivot elements are small, causing large rounding errors during row operations and undermining the accuracy of the solution. This instability is particularly pronounced in ill-conditioned systems where the coefficient matrix has a high condition number (Golub & Van Loan, 2013).

In the end, while Gaussian Elimination without pivoting offers a simple and direct approach to solving linear systems, it has a relatively bad numerical stability and robustness. Careful handling of the system's properties is essential to ensure accurate and reliable solutions in my implementation.

**Part 2: Singular Value Decomposition for Image Compression**

* 1. **Methodology**

**Singular Value Decomposition (SVD)** is a fundamental matrix factorization technique we learned, decomposing a matrix into three constituent matrices:

Here, is an orthogonal matrix containing the left singular vectors; is an diagonal matrix containing the singular values arranged in decreasing order; and is the transpose of an orthogonal matrix containing the right singular vectors. In our objective of image compression, is the grayscale image matrix, with each entry corresponding to a pixel's intensity. By retaining only the top singular values and their corresponding singular vectors, we can construct a **low-rank approximation** , where consist of the first columns of , the first singular values, and the first rows of , respectively. Using this method, we may significantly reduce the storage requirements without losing essential features of the grayscale picture (Strang, 2009).

* 1. **Implementation**

We first prepare the matrix and transform the image into desirable form: the grayscale image will be converted into a matrix representation where each element represents the intensity of a corresponding pixel.Then, we calculate and , which are symmetric matrices instrumental in determining the singular values and vectors.

Next, we obtain the dominant eigenvalues and their corresponding eigenvectors of and .using power iteration method. With the deflation technique to modify the matrices, enabling the subsequent extraction of the next dominant eigenvalues and eigenvectors.

We try contrasting matrices , , and . Obtaining the singular values are derived as the square roots of the eigenvalues of , then obtain the eigenvectors of to form the columns of .Each column of is computed by normalizing the product with respect to the corresponding singular value , and is the -th right singular vector.

Low-rank approximation and image restoration is also a major objective in the implementation.We select by determining the number of singular values to remain based on a predefined rate of singular values' summation.Then we can use the truncated matrices and reconstruct the approximated image matrix . In the end we will also evaluate the quality of the compressed image by comparing with the original image matrix .

**2.3 Complexity Analysis**

**Time Complexity** : When using the power iteration method, especially when combined with deflation for multiple eigenvalues, we are using computing power intensively. For each eigenvalue, the method involves operations, where and are the dimensions of the matrix. What’s more, calculating and requires and operations respectively. So the time complexity: is approximately , particularly when we are dealing with a large number of singular values and vectors.

**Space Complexity** : Original Image Matrix : Requires space, while the symmetric Matrices and : Each requires and space respectively. We also have eigenvectors and singular values that require minor additional storage. Summing everything up, we have a space complexity dominated by the term , we approximate it as as it depending on the image size and matrix dimensions.

* 1. **Truncation Error and Convergence**

Based on the formula in the lecture, the truncation error in low-rank approximations is quantified by the Frobenius norm of the difference between the original matrix and its approximation :

Here, are the singular values in descending order, and is the rank of . We can retain a higher number of singular values to reduce the truncation error, resulting in a more accurate approximation of the original image yet with larger computation intensity.

For the power iteration method, the ratio of the dominant eigenvalue to the second-largest eigenvalue will influence the convergence rate at a great level. A larger ratio ensures faster convergence, while closely spaced eigenvalues slow down the convergence.After extracting each eigenvalue and eigenvector, the utility of deflation modifies the matrix to facilitate the extraction of subsequent eigenvalues, maintaining numerical stability and ensuring convergence for each eigenvalue.

However, for matrices with multiple closely spaced eigenvalues, the power iteration method may require a significant greater number of iterations to achieve accurate results. This makes balancing the number of singular values retained essential in order to obtain an optimal compression level and truncation error.

|  |  |  |
| --- | --- | --- |
| Metric | Gaussian Elimination Without Pivoting | Singular Value Decomposition (SVD) |
| Time Complexity |  |  |
| Space Complexity |  |  |
| Numerical Stability | Low without pivoting | Generally stable but dependent on eigenvalue separation |
| Error Handling | Halts on zero pivots | Truncation introduces approximation errors |
| Convergence Characteristics | Deterministic with fixed operations | Iterative with convergence dependent on eigenvalue distribution |

**Conclusion**

In conclusion, gaussian elimination without pivoting is a deterministic algorithm tailored for solving linear systems. A straightforward implementation makes it suitable for small to medium-sized problems. However, the lack of pivoting compromises its numerical stability, making it relatively less reliable for ill-conditioned systems (Strang, 2009).

On the other hand, SVD offers a versatile framework for matrix factorization, enabling applications like image compression through low-rank approximations. While indeed computationally intensive, especially without optimized library functions, SVD provides robust mechanisms for dimensionality reduction and data approximation. Its iterative nature introduces complexities in convergence behavior, particularly in matrices with closely spaced singular values (Golub & Van Loan, 2013).

Both algorithms exemplify the profound impact of applied mathematics in computational problem-solving, each meets specific needs within the vast scenarios of scientific and engineering disciplines.

**References**

Friedman, J., Hastie, T., & Tibshirani, R. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

Golub, G. H., & Van Loan, C. F. (2013). *Matrix Computations*. Johns Hopkins University Press.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357-362.

Higham, N. J. (2002). *Accuracy and Stability of Numerical Algorithms*. SIAM.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90-95.

Numerical Recipes in C: The Art of Scientific Computing by William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery.

Patterson, D., & Starke, J. (1993). *Matrix Computations*. Cambridge University Press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press.

Strang, G. (2009). *Introduction to Linear Algebra*. Wellesley-Cambridge Press.