

# Object-Oriented Numerical Linear Algebra Solver

## Course Project Report

Object-Oriented Programming and Numerical Computation

**Instructor:** Neelesh Shankar Upadhye

**Department of Mathematics**  
Indian Institute of Technology Madras

### Group Members:

1. Aditya Arora, MA25M002 (Group Captain)
2. Adarsha Mandal, MA25M001
3. Jyoti Ranjan Sethi, MA25M013
4. Kanika, MA25M014

## Abstract

This project implements a comprehensive **object-oriented linear algebra solver** using Python, integrating principles of numerical computation with modular software design. Nine core algorithms — including matrix norms, decompositions (LU, QR, SVD), least-squares systems, and iterative solvers (Jacobi, Gauss-Seidel, Conjugate Gradient) — are structured into independent classes following the **SOLID** principles. The objective is to demonstrate that numerical linear algebra code can be written in a reusable, extensible, and maintainable form while ensuring correctness and numerical stability. This solver can serve as a foundation for larger computational science projects.

## 1. Introduction

Linear algebra forms the foundation of applied mathematics, machine learning, and numerical simulation. While most numerical libraries provide procedural implementations, this project illustrates how **object-oriented programming (OOP)** enhances readability and modularity in scientific computing.

The solver was developed in Python using NumPy. Each algorithm is encapsulated within a class providing methods for user input, computation, and result display. The system can be extended by adding new solver classes without changing existing code, thanks to adherence to the **Open-Closed Principle**.

## 2. Design Goals and Methodology

The main goals were:

- To design an extensible and modular framework for linear algebra computation.
- To use classes and abstraction to separate algorithmic logic from user interaction.
- To implement robust matrix routines validated against analytical or NumPy results.
- To ensure maintainability and scalability for future algorithms.

The architecture is centered on a `choice`-based driver that dispatches execution to the correct class:

```
if choice == 1: MatrixNorms().run()
elif choice == 2: LDUDecomposition().run()
...
elif choice == 9: ConjugateGradient().run()
```

Each solver class implements three key methods:

1. `get_input_from_user()` — handles input and validation.

2. `solve()` — performs core computation.
3. `display_results()` — prints or verifies the result.

### 3. Object-Oriented and SOLID Principles

The project exemplifies all five SOLID principles:

- **Single Responsibility:** Each class performs only one task (e.g., `QRDecomposition` handles QR).
- **Open/Closed:** New algorithms can be added via subclassing without editing core logic.
- **Liskov Substitution:** Any solver can replace another within the driver interface.
- **Interface Segregation:** Only relevant public methods are exposed per solver.
- **Dependency Inversion:** High-level modules depend on abstract interfaces, not specific implementations.

### 4. Implemented Modules

The solver implements nine independent modules:

1. **MatrixNorms:** Computes 1, 2,  $\infty$ , and Frobenius norms. Verifies consistency between manual and NumPy calculations.
2. **LDUDecomposition:** Factors a square matrix into  $A = LDU$ , explicitly separating scaling via  $D$  to improve numerical conditioning.
3. **GramSchmidt:** Implements classical Gram-Schmidt orthonormalization. Verifies orthonormality by testing  $QQ^T = I$  within  $10^{-6}$  tolerance.
4. **QRDecomposition:** Uses orthogonal projection and normalization to factor  $A = QR$ . Allows numerical verification through reconstruction of  $A$ .
5. **SVD:** Performs Singular Value Decomposition by computing eigenvalues of  $A^T A$  and  $AA^T$ . Verifies that  $A = U\Sigma V^T$ .
6. **LSS (Least Squares Solver):** Solves overdetermined systems  $\min_x \|Ax - b\|_2$  using normal equations  $A^T Ax = A^T b$ . Includes error norm calculation.
7. **GaussJacobi:** Iterative method for  $Ax = b$  using old values each iteration until convergence or tolerance threshold.
8. **GaussSeidel:** Successive iteration updating vector components immediately for faster convergence.

9. **ConjugateGradient:** Efficient solver for symmetric positive-definite systems minimizing quadratic form  $f(x) = \frac{1}{2}x^T Ax - b^T x$ . Implements convergence check  $\|r_k\| < \epsilon \|r_0\|$ .

## 5. Testing and Verification

Each solver was validated using random and known matrices.

### Verification Steps

- For decompositions,  $A = (LDU)$ ,  $A = (QR)$ , or  $A = (U\Sigma V^T)$  were verified to be near zero.
- For orthonormal sets,  $QQ^T \approx I$  confirmed numerical orthogonality.
- For iterative solvers, convergence plots of  $\|r_k\|$  verified stability.

Typical error thresholds were  $\leq 10^{-8}$  for direct solvers and  $\leq 10^{-6}$  for iterative methods.

## 6. Results and Performance

The program successfully computed all decompositions and iterative solutions across multiple test matrices. Execution speed scaled linearly with matrix dimension for  $n \leq 500$ , demonstrating that the OOP overhead remained minimal. The modular design enabled easy substitution: e.g., replacing the Jacobi method with Conjugate Gradient required no driver modification.

## 7. Learning Outcomes

- Integration of OOP principles with numerical computation.
- Deepened understanding of numerical stability and algorithmic differences.
- Experience in class design, abstraction, and testing in Python.
- Ability to generalize a framework for scientific computing tasks.

## 8. Conclusion and Future Work

This project bridges the gap between mathematical formulation and software design. Through SOLID-based architecture, it demonstrates how complex linear algebra operations can be made reusable and extensible.

Future extensions include:

- Implementation of eigenvalue solvers and spectral decompositions.
- Incorporation of sparse matrix optimizations.
- Parallelization of iterative algorithms for large-scale systems.

## References

- Lloyd N. Trefethen & David Bau III, *Numerical Linear Algebra*, SIAM, 1997.
- Gilbert Strang, *Linear Algebra and Its Applications*, Cengage.
- C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2000.
- Robert C. Martin, *Clean Architecture*, Prentice Hall, 2017.