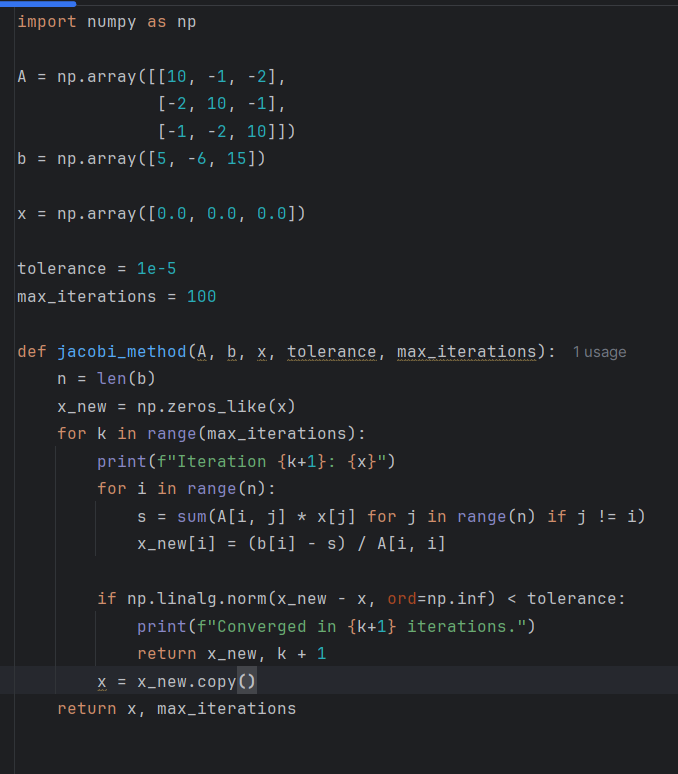
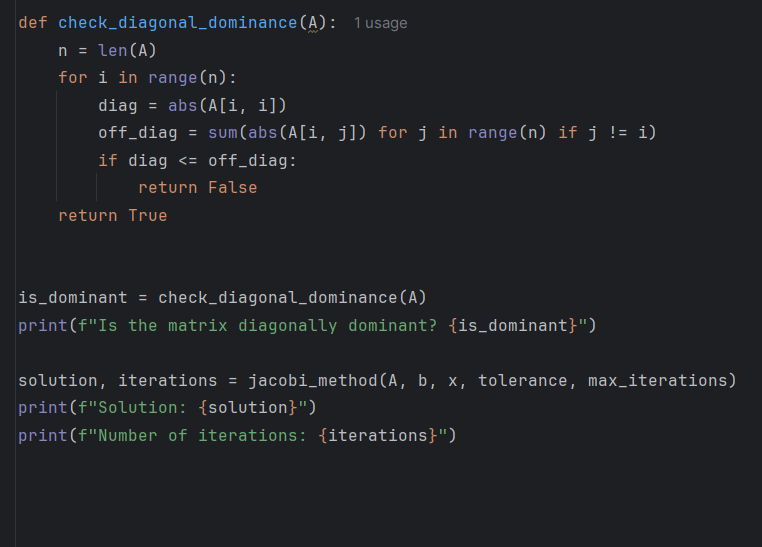
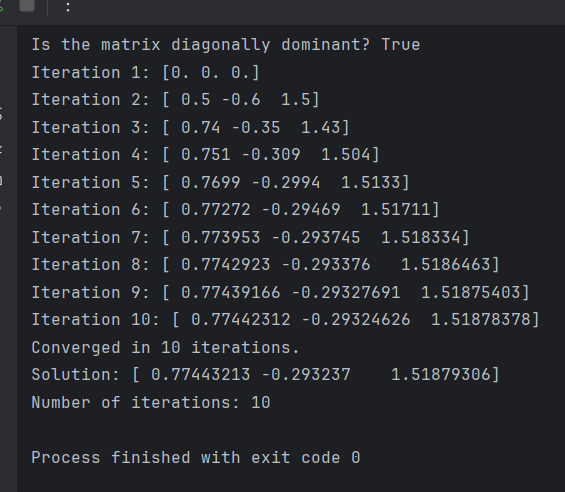
Aldongar Yerkin

SE 2318  
Task 1





Result:



Explanation:

Define the coefficient matrix A and right-hand side vector b.

Initialize an initial guess vector x as [0, 0, 0].

Check diagonal dominance by ensuring that the absolute value of each diagonal element is greater than the sum of the absolute values of all other elements in the same row.

Perform iterative updates using the Jacobi formula.

Stop the iterations when the difference between successive approximations (measured by the infinity norm) is less than the tolerance (10^-5).

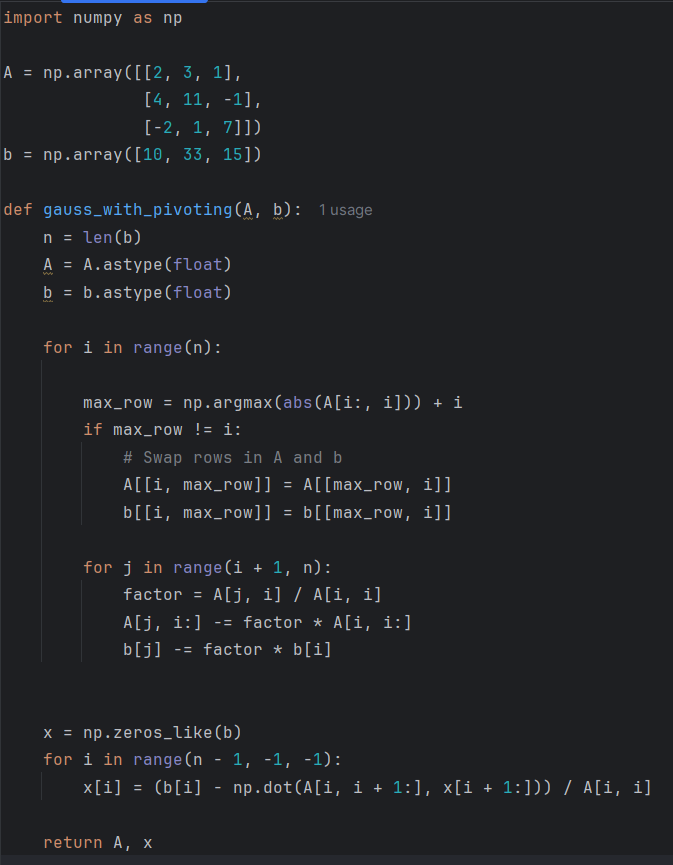
Print the results of each iteration for clarity.

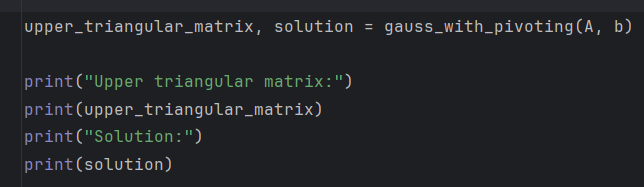
I use Jacobi method because:

It is an iterative approach suitable for large systems where direct methods (like Gaussian elimination) are computationally expensive.

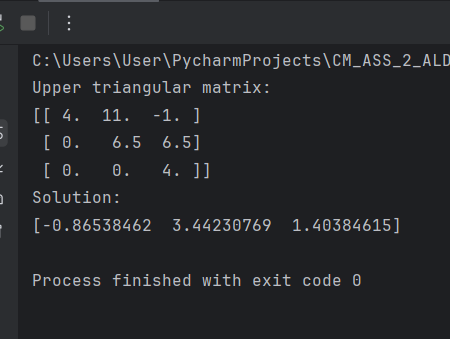
It relies on successive approximations, making it easier to implement for sparse systems.

Task 2





Result:



Explanation:

Define the coefficient matrix A and vector b.

Use partial pivoting.

For each column, find the row with the largest absolute value in the current column.

Swap rows to place this element as the pivot.

Perform forward elimination.

Subtract multiples of the pivot row from rows below it to create zeros below the pivot.

Perform back substitution.

Solve for the variables starting from the bottom row of the upper triangular matrix.

After forward elimination, the matrix is transformed into an upper triangular form where all elements below the diagonal are zero.

This simplifies solving the system through back substitution.

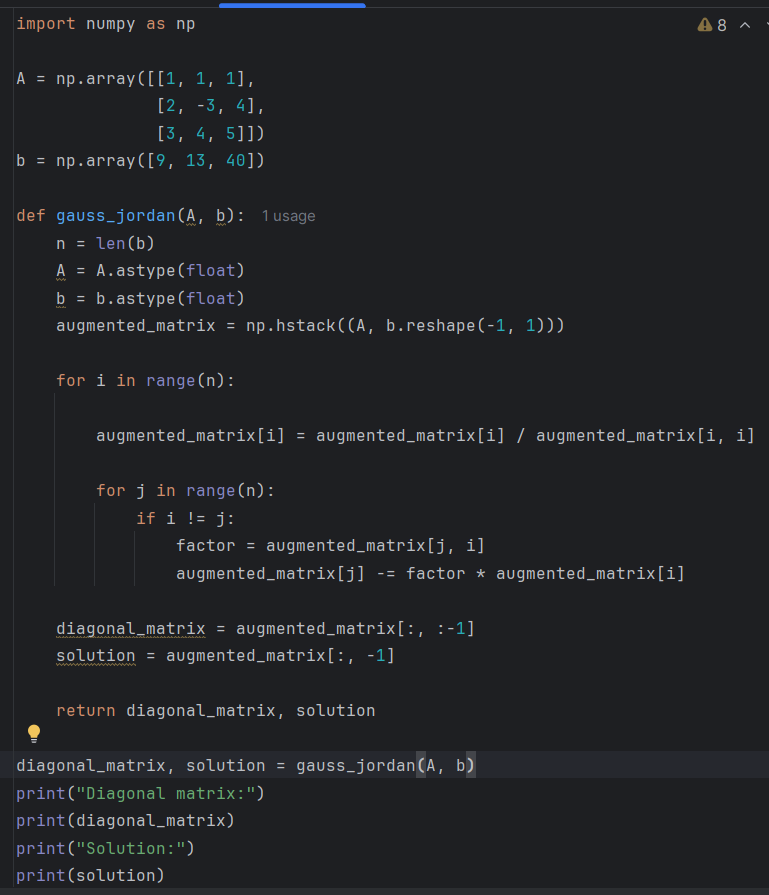
Pivoting reduces the effect of numerical errors and ensures accurate results, especially for systems with large coefficient differences.

I use pivoting because:

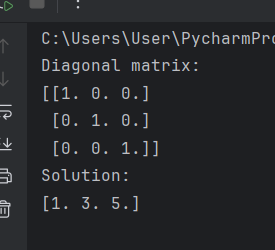
Without pivoting, very small or zero pivot elements can lead to division errors or significant rounding errors in floating-point arithmetic.

Pivoting ensures better numerical accuracy and stability.

Task 3:



Result:



Explanation:

Define the coefficient matrix A and vector b.

Form the augmented matrix by appending b to A.

For each row, normalize the diagonal element to 1 by dividing the row by the diagonal element.

Eliminate all other elements in the column by subtracting appropriate multiples of the current row from other rows.

Directly gives the solution without back substitution.

More systematic and suitable for finding the inverse of a matrix, as the same operations can transform the identity matrix into the inverse.

The diagonal matrix clearly shows the simplified system.

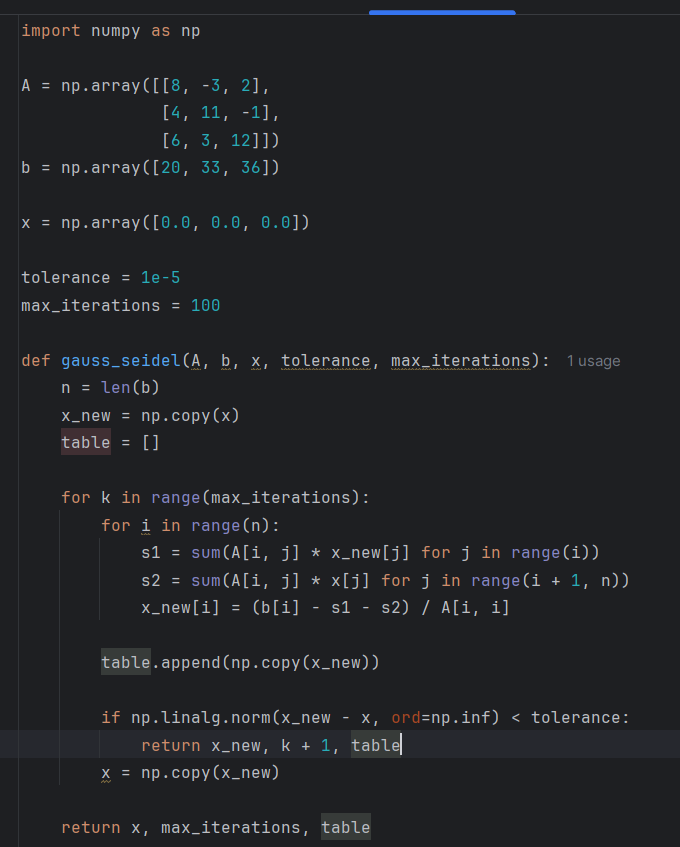
The method is more computationally expensive but systematic.

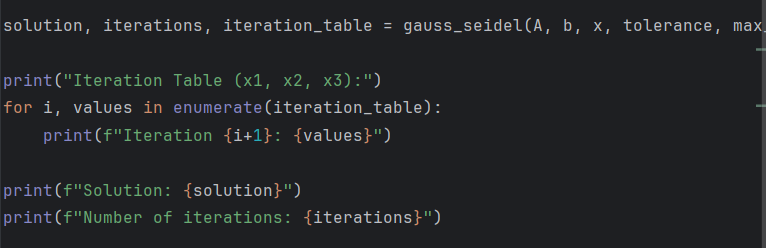
I use Gauss-Jordan because:

Unlike Gaussian elimination, Gauss-Jordan eliminates the need for back substitution.

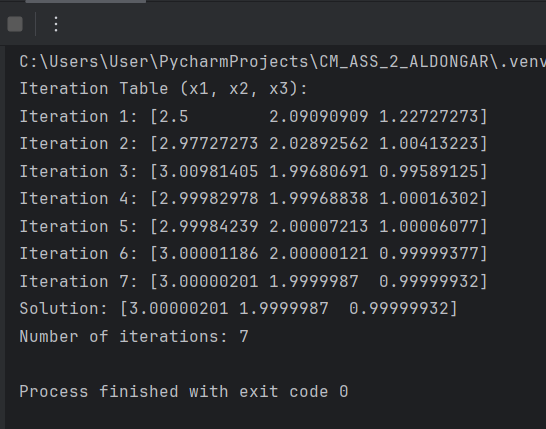
It directly reduces the system to a diagonal form (or reduced row-echelon form).

Task 4





Result:



Explanation:

Define the coefficient matrix A and vector b.

Initialize an initial guess vector x.

Iteratively update each variable using.

Store the values of all variables at each iteration for table output.

Stop when the difference between successive approximations is below the tolerance.

A smaller tolerance (10^-5 ) increases the number of iterations but ensures higher accuracy.

Larger tolerances lead to faster convergence but lower precision.

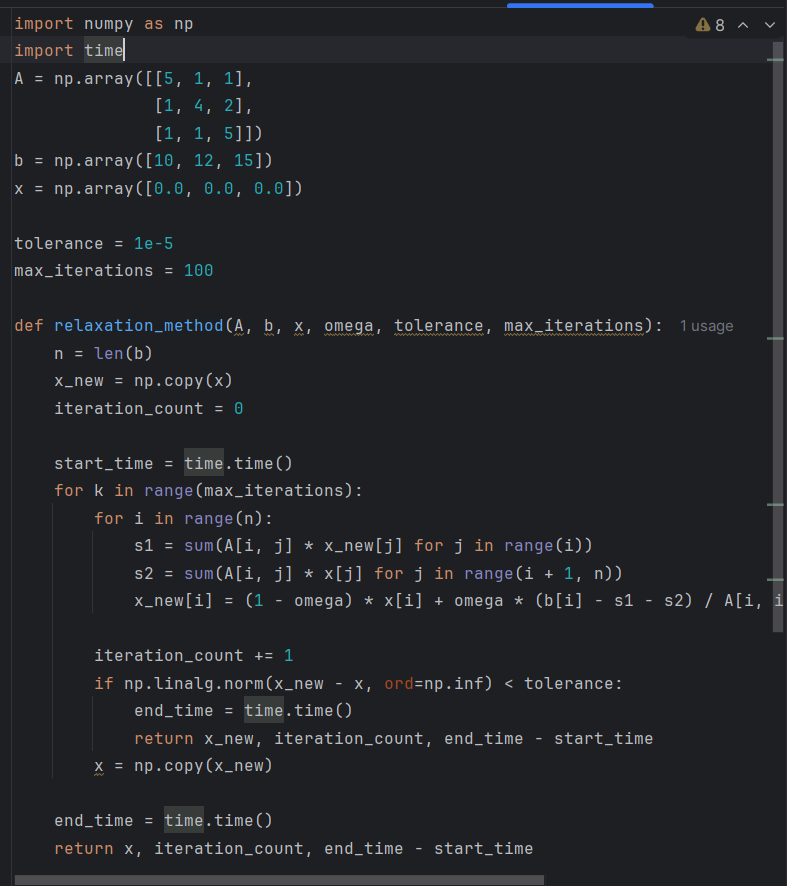
Outcome.

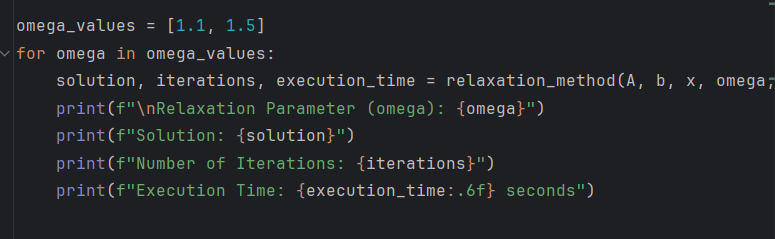
I use Gauss-Seidel because:

It is faster than the Jacobi method because it uses the latest updated values immediately within the same iteration.

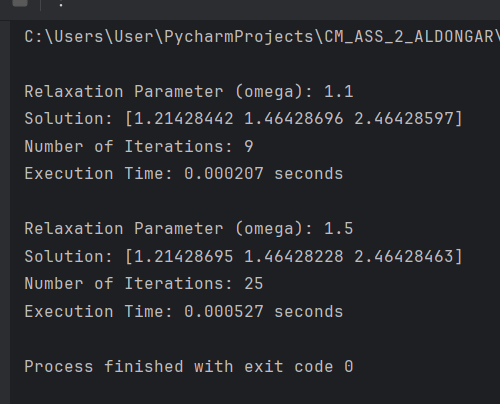
Suitable for diagonally dominant or symmetric positive-definite matrices.

Task 5





Result:



Explanation:

Solve the system iteratively using the relaxation formula.

Test for 𝜔 = 1.1 (slight acceleration) and ω=1.5 (higher acceleration).

Measure the number of iterations and execution time for each ω.

Lower ω values may converge more slowly.

Higher ω values may lead to divergence if too large.

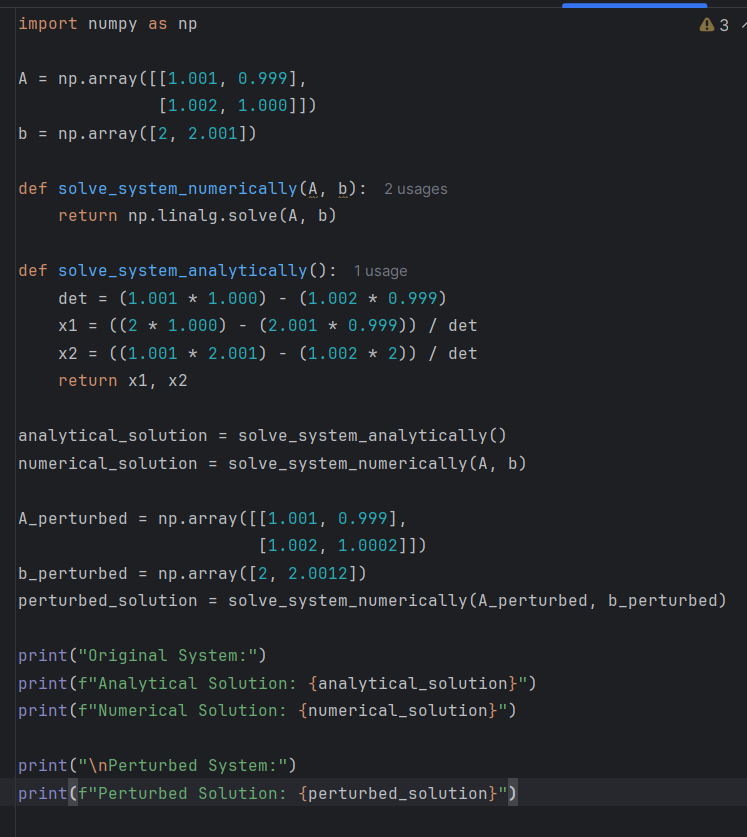
Solutions, iteration counts, and execution times are compared for both ω.

I use Relaxation because:

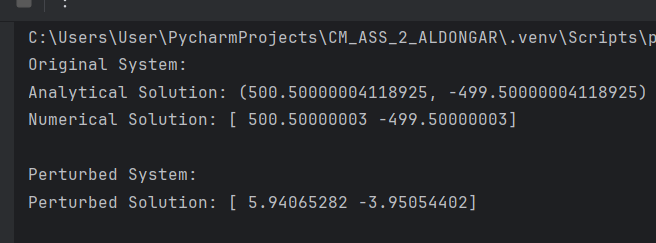
Relaxation accelerates convergence by introducing a weighted average of the current and previous values.

Optimal ω values lead to faster convergence.

Task 6



Result:



Explanation:

Solve analytically using the determinant and substitution formulas.

Solve numerically using NumPy’s linalg.solve.

Introduce small perturbations in the coefficients and observe the significant changes in the solution.

Analytical and numerical solutions match for the original system.

A small change in coefficients leads to drastically different solutions, illustrating the instability of ill-conditioned systems.