**DESIGN, IMPLEMENT, AND ANALYZE ALGORITHMS BY CREATING A SYSTEM TO FIND THE INVERSE OF A MATRIX IN RUNNING TIME COMPLEXITY OF O(n3).**

PROJECT REPORT

***by***

**A04297618: THRIVEEN ULLENDULA**

**A04298775: OM PREETHAM BANDI**

**A04298487: REDDY BHUVAN KORLAKUNTA**

**A04299831: SRAVYA SRI VIRIGINENI**

**School of Engineering & Computer Sciences**

**Texas A&M University – Corpus Christi**

**April 2023**

**BONAFIDE CERTIFICATE**

This is to certify that this project report entitled **“DESIGN, IMPLEMENT, AND ANALYZE ALGORITHMS BY CREATING A SYSTEM TO FIND THE INVERSE OF A MATRIX IN RUNNING TIME COMPLEXITY OF O(n3)”** submitted to **Texas A&M University – Corpus Christi**, is a bonafide record of work done by **“(A04297618) THRIVEEN ULLENDULA, (A04298775) OM PREETHAM BANDI, (A04298487) REDDY BHUVAN KORLAKUNTA, (A04299831) SRAVYA SRI VIRIGINENI”** under my supervision from “19th March 2023 to 19th April 2023.”

Dr. Minhua Huang

School of Engineering & Computer Sciences

Place: Corpus Christi, TX

Date: 19th April 2023

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
| 1. | Introduction | 1 |
| 2.  2.1  2.2  2.3 | Algorithms of the system  The functionality for each of the algorithms  The running time complexity analysis for each of the algorithms  Data Structures | 2 |
| 3  3.1  3.2  3.3 | Experiments  Generating the inverse of an attached matrix A  The time measured for obtaining the inverse on the attached matrix A by system.  Generating the inverse of a randomly generated matrix A | 4 |
| 4. | Pros & Cons | 6 |
| 5. | Further Improvements | 7 |

1. **INTRODUCTION**

This code performs LU decomposition with partial pivoting, matrix inversion, and determinant calculation for a given square matrix in Python.

**LU DECOMPOSITION**

The LU\_decomposition function takes a square matrix as input and returns the unit lower triangular matrix L, upper triangular matrix U, and permutation matrix P such that PA = LU.

It first initializes L and U to be matrices of zeros, and P to be the identity matrix.

Then, for each column j of the matrix, it finds the pivot row (the row with the largest absolute value in column j), and if necessary, swaps rows in the matrix and P and swaps rows in the previous columns of L.

It then sets the diagonal element of U to be the pivot element of the matrix, calculates the entries of L and U below the diagonal element of U, and subtracts their outer product from the remaining part of the matrix. Finally, it sets the diagonal elements of L to be ones.

**INVERSE OF A MATRIX**

The invert\_matrix function takes a square matrix as input and computes its inverse using LU decomposition with partial pivoting.

It first computes the LU decomposition of the matrix using the LU\_decomposition function.

Then, it solves the system of linear equations LUX = B for each column of the inverse matrix X, where B is the column vector of the identity matrix.

It uses forward substitution to solve Ly = b and backward substitution to solve Ux = y for each column of X. Finally, it returns the inverse matrix X.

The code also reads a square matrix from a CSV file and asks the user for the size of a new square matrix to test the functions on.

It generates a new square matrix of the specified size by extracting the first n rows and columns of the original matrix.

Finally, it prints the conditions to be satisfied by the input matrix for the functions to work correctly.

1. **ALGORITHMS OF THE SYSTEM:**

Here is a step-by-step explanation of the algorithm for finding the inverse of a matrix using LU decomposition with partial pivoting, as implemented in the Python code:

LU decomposition is a method used to factorize a matrix into a lower triangular matrix (L), an upper triangular matrix (U), and a permutation matrix (P). Once we have obtained the L, U, and P matrices, we can use them to solve linear systems of equations, which can be used to find the inverse of a matrix.

Given a square matrix A, we want to find the inverse A^-1. We use the LU decomposition to factorize A into PA = LU, where P is a permutation matrix, L is a lower triangular matrix, and U is an upper triangular matrix. Then, we solve the system of linear equations LUX = B for each column of the inverse matrix X, where B is the column vector of the identity matrix.

**2.1 The functionality for each of the algorithms**

* 1. **.1 Algorithm for LU decomposition with partial pivoting:**

**Input:** A n × n matrix

**Output:** L, U, and P matrices

1. Create zero matrices L and U of size n × n and permutation matrix P of size n × n with diagonal elements as 1.

2. For j=1 to n-1, do the following:

a. Find the row i with the largest absolute value in column j, starting from j.

b. If i != j, swap the rows i and j in the matrices A and P.

c. Divide the j-th row of A by the j-th element of A to get the j-th row of U.

d. Store the quotient in the L matrix at position (i,j).

e. For each row i > j, subtract L[i][j] times the j-th row of U from the i-th row of A.

3. Set the diagonal elements of L to 1.

**2.1.2 Algorithm for finding the inverse of a matrix using LU decomposition:**

**Input:** A n × n matrix

**Output:** The inverse of the matrix A

1. Compute the LU decomposition of A using the above algorithm.

2. Solve the system of linear equations LUX = B for each column of the inverse matrix X, where B is the column vector of the identity matrix.

3. Use forward substitution to solve Ly = b and backward substitution to solve Ux = y for each column of X.

4. Return the inverse matrix X.

* 1. **The running time complexity analysis for each of the algorithms**
* Finding the LU decomposition takes O(n^3) time since we perform Gaussian elimination on the matrix.
* Solving a system of linear equations using **forward and backward substitution** takes O(n^2) time (where the loop runs in range n and it has 3 further consequent n loops(n+n+n) so n\*3n => 3n^2).
* So finding the inverse of a matrix using LU decomposition takes O(n^3 + n^2) = O(n^3) time.
  1. **Data Structures**

We had used the data structure 2-D array to represent the matrix. This 2 D array elements represent the matrix entries.

***Finding the inverse of a matrix using LU decomposition involves the following steps:***

**LU Decomposition:** Decompose the original matrix into a lower triangular matrix L and an upper triangular matrix U such that A = LU. This can be done using Gaussian elimination.

**Solve for Y:** Solve the equation LY = I, where I is the identity matrix of the same size as A, for Y. This can be done using forward substitution.

**Solve for X:** Solve the equation UX = Y for X, where Y is the solution found in step 2. This can be done using backward substitution.

**Verify:** Multiply the original matrix A by the inverse matrix A^-1 to check if the result is the identity matrix I.

1. **EXPERIMENTS**

**3.1 Generating the inverse of the attached matrix A**

**A picture containing background pattern

Description automatically generated**

**A picture containing background pattern

Description automatically generated**

**3.2 The time measured for obtaining the inverse on the attached matrix A by your system.**

**Text

Description automatically generated with low confidence**

**3.3 Generating the inverse of a randomly generated matrix A**

**Background pattern

Description automatically generated**

**3.4 Screenshots of a randomly generated matrix and the inverse outputted by your system.**

**A picture containing background pattern

Description automatically generated**

**Background pattern

Description automatically generated**

1. **PROS & CONS**

**Pros:**

1. The LU decomposition method is computationally efficient and numerically stable.
2. It can be used to solve a system of linear equations for multiple right-hand side vectors without having to factorize the matrix again.
3. The L and U matrices can be used to compute the determinant of a matrix.

**Cons:**

1. The method may fail to find the LU decomposition if the matrix is singular or nearly singular.
2. The method requires O(n^2) memory to store the L and U matrices.
3. The method is not suitable for matrices that have many zeros, since the Gaussian elimination process may introduce large intermediate values.
4. **FURTHER IMPROVEMENTS**

* This code generates a random matrix with values ranging from 0 to 99. We check for the singularity of matrix later. As an improvement we can include the process of checking singularity in this random matrix generation function so that it will only generate the non-singular matrix.
* Next improvement could be instead of picking random value in matrix from 0 to 99 we can give the option to the user to enter the range he wants.
* Code complexity can be further by choosing a different method for finding inverse.
* The LU decomposition algorithm can be used to find the inverse of a matrix, but it may not always be the most efficient approach. An alternative algorithm for finding the inverse of a matrix is the Gauss-Jordan elimination method.
* The Gauss-Jordan elimination method involves transforming the original matrix into its reduced row echelon form, which is an equivalent matrix that has been simplified such that each row has a leading 1, and all entries below the leading 1 are zero. The reduced row echelon form of a matrix is unique, and if the original matrix is invertible, then its reduced row echelon form will be the identity matrix.
* To apply the Gauss-Jordan elimination method to find the inverse of a matrix, we start by augmenting the original matrix with the identity matrix of the same size. We then perform elementary row operations on the augmented matrix until the left half of the augmented matrix is reduced to the identity matrix. The right half of the augmented matrix will then be the inverse of the original matrix.