Exploring LU Decomposition in Linear Algebra

Mohammed El Amine Jouibi

MTH 2303 03
Dr. El Mostafa Kalmoun
April 17, 2024

1 Abstract

This project delves into LU decomposition, a fundamental concept in linear algebra. LU decomposition involves factoring a matrix into the product of a lower triangular matrix (L) and an upper triangular matrix (U). The objective of this project is to thoroughly understand LU decomposition, implement it from scratch, and analyze its applications. Through this exploration, we aim to provide a comprehensive understanding of LU decomposition and its significance in various mathematical and computational contexts.

2 Introduction

Linear algebra is a branch of mathematics that deals with vector spaces and linear mappings between these spaces. LU decomposition, also known as LU factorization, is a method used to solve systems of linear equations and compute matrix inverses. In this project, we aim to explore LU decomposition, its theoretical underpinnings, and practical applications. Specifically, I will provide code that implements the LU decomposition, and a detailed explanation of what LU decomposition is and how it works mathematically and it's usefulness.

3 Problem Statement or Project Description

The primary goal of this project is to understand LU decomposition thoroughly. We seek to address the following questions:

1. What is LU decomposition, specifically using the Doolittle method, and how does it work?
2. What are the applications of LU decomposition in solving systems of linear equations and computing matrix inverses?
3. How can LU decomposition be implemented from scratch using basic linear algebra operations?
4. What are the benefits of using the Doolittle method without Gaussian Elimination?

Understanding LU decomposition is crucial as it forms the basis for various computational algorithms and numerical methods in engineering, physics, computer science, and other disciplines.

4 Methodology or Approach

My approach involves studying the theoretical foundations of LU decomposition, including the mathematical principles behind it. We will then implement LU decomposition from scratch using the Python programming language, using the Doolittle method/algorithm.

4.1 Doolittle's Method

This is the specific algorithm employed for LU decomposition in the code. Doolittle's method factors the matrix A into the product of a lower triangular matrix L and an upper triangular matrix U, with partial pivoting performed using the permutation matrix P to ensure numerical stability.

LU Decomposition (Doolittle's method)

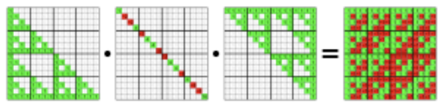
- Matrix decomposition

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

4.1.1 Permutation Matrix P

This matrix represents the row permutations needed during LU decomposition to perform partial pivoting. It ensures that the largest elements are placed along the diagonal of U, which helps in maintaining numerical stability.

5 Implementation or Execution


LDU decomposition of aWalsh matrix

I implemented LU decomposition in Python, starting with the basic algorithm and gradually refining it for efficiency and numerical stability. I first encountered problems when running the code, but after numerous trials and errors, and through rigorous testing and debugging, I overcame those challenges and developed a fully functioning implementation of LU decomposition. The main steps taken were:

Matrix Multiplication Function: This function multiplies two matrices. It uses list comprehensions to perform matrix multiplication.

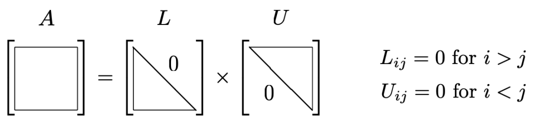
Pivoting Matrix Function: This function creates a pivoting matrix that rearranges the identity matrix such that the largest element of each column of the input matrix (M) is placed on the diagonal of M. It iterates over each column of M, finds the row with the largest absolute value element in that column, and swaps that row with the current row.

LU Decomposition Function: This function performs LU decomposition using Doolittle's method. It first initializes matrices L and U with zeros. It calculates the pivoting matrix (P) and the multiplied matrix (PA) by multiplying P and A. Then, it iterates over the columns of the input matrix A to calculate the elements of L and U. For each column, it computes the elements of U using the formula provided in comments and computes the elements of L accordingly. Finally, it returns the matrices P, L, and U.

Main Execution: The main code initializes a sample matrix A. It calls the LU decomposition function to compute P, L, and U matrices. Then, it prints out A, P, L, and U matrices. Additionally, it checks if the original matrix A can be reconstructed by multiplying L and U matrices and prints the result.

6 Results:

Mathematical aspect: A very general LU decomposition using Doolittle's method can be represented as follows:



Given matrix A = $\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$

Initialize matrices L and U with zeros.

For j = 0 to n-1: $\begin{cases} U_{jj} = A_{jj} - \sum_{k=0}^{j-1} U_{kj} L_{jk} \\ L_{ij} = \frac{1}{U_{jj}}(A_{ij} - \sum_{k=0}^{j-1} U_{kj} L_{ik}) \text{ for } i \geq j \end{cases}$

Return L and U

Now, implementing this specifically in the code (shown later):

Given matrix A = $\begin{pmatrix} 7 & 3 & -1 & 2 \\ 3 & 8 & 1 & -4 \\ -1 & 1 & 4 & -1 \\ 2 & -4 & -1 & 6 \end{pmatrix}$

Initializing matrices L and U with zeros.

Next, Computing the pivot matrix P such that PA = LU .

For j = 0 to 3: $\begin{cases} U_{jj} = A_{jj} - \sum_{k=0}^{j-1} U_{kj} L_{jk} \\ L_{ij} = \frac{1}{U_{jj}}(A_{ij} - \sum_{k=0}^{j-1} U_{kj} L_{ik}) \text{ for } i \geq j \end{cases}$

The resulting matrices are: L $= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4286 & 1 & 0 & 0 \\ -0.1429 & 0.2125 & 1 & 0 \\ 0.2857 & -0.3913 & -0.3082 & 1 \end{pmatrix}$, U $=$ $\begin{pmatrix} 7 & 3 & -1 & 2 \\ 0 & 6.7143 & 1.4286 & -4.8571 \\ 0 & 0 & 3.5536 & 0.9821 \\ 0 & 0 & 0 & 1.038 \end{pmatrix}$

Checking if A $=$ LU by multiplying L and U and comparing with A:

Computing $\mathrm{A_{result}} = LU$

If $\mathrm{A_{result}} = A$, then the LU decomposition is correct.

$$\mathrm{A_{result}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.4286 & 1 & 0 & 0 \\ -0.1429 & 0.2125 & 1 & 0 \\ 0.2857 & -0.3913 & -0.3082 & 1 \end{pmatrix} \times \begin{pmatrix} 7 & 3 & -1 & 2 \\ 0 & 6.7143 & 1.4286 & -4.8571 \\ 0 & 0 & 3.5536 & 0.9821 \\ 0 & 0 & 0 & 1.038 \end{pmatrix}$$

$$\mathrm{A_{result}} = \begin{pmatrix} 7 & 3 & -1 & 2 \\ 3 & 8 & 1 & -4 \\ -1 & 1 & 4 & -1 \\ 2 & -4 & -1 & 6 \end{pmatrix}$$

7 Results

The usage of the Doolittle method presents various advantages, offering improved stability, particularly when dealing with matrices containing extremely large or small values. Furthermore, once the LU decomposition is computed, it can be reused for solving multiple linear systems, inversion of the matrix, and determination of the determinant, all of which are performed with greater efficiency compared to Gaussian elimination.

The results demonstrate the effectiveness of the Doolittle method when used in LU decomposition in solving systems of linear equations and computing matrix inverses. However, It is clear that in order for this algorithm to work, one needs to have at each step $a_{n,n}^{(n-1)} \neq 0$. If this assumption fails at some point, one needs to interchange n-th row with another row below it before continuing. This is why the LU decomposition in general looks like$P^{-1}A = LU$.

8 Conclusion

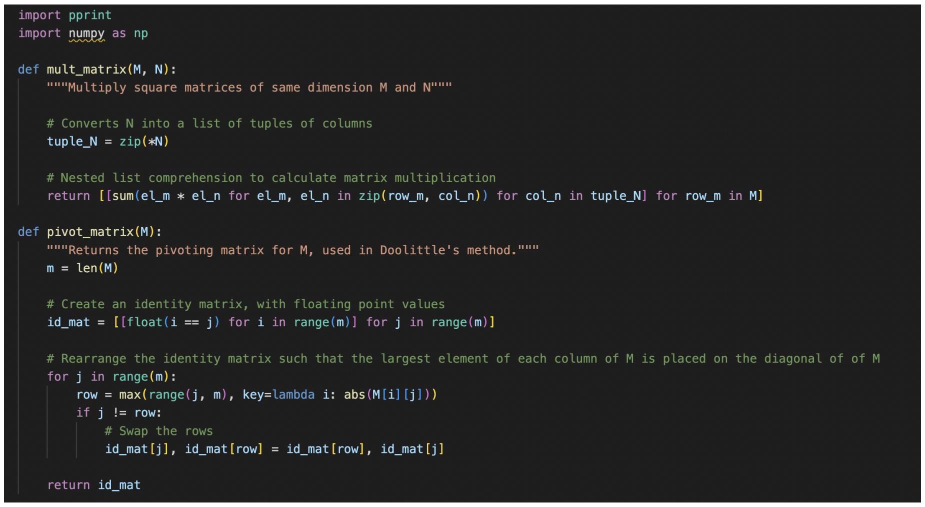
In conclusion, this project provided a comprehensive exploration of LU decomposition in linear algebra, specifically using the Doolittle's method. We achieved our objectives of understanding LU decomposition, implementing it from scratch, and analyzing its applications. This project proved the usefulness of the Doolittle algorithm, and its numerous benefits. LU decomposition serves as a powerful tool in various fields, facilitating efficient and accurate solutions to linear algebraic problems. If we extend this project, several promising directions can be pursued for future work. Firstly, optimizing the performance of LU decomposition could be pursued, exploring parallelization techniques or algorithmic refinements to enhance computational efficiency. Additionally, addressing numerical stability concerns by investigating pivoting strategies or regularization techniques could improve the reliability of solutions, especially for matrices with high condition numbers.

9 References

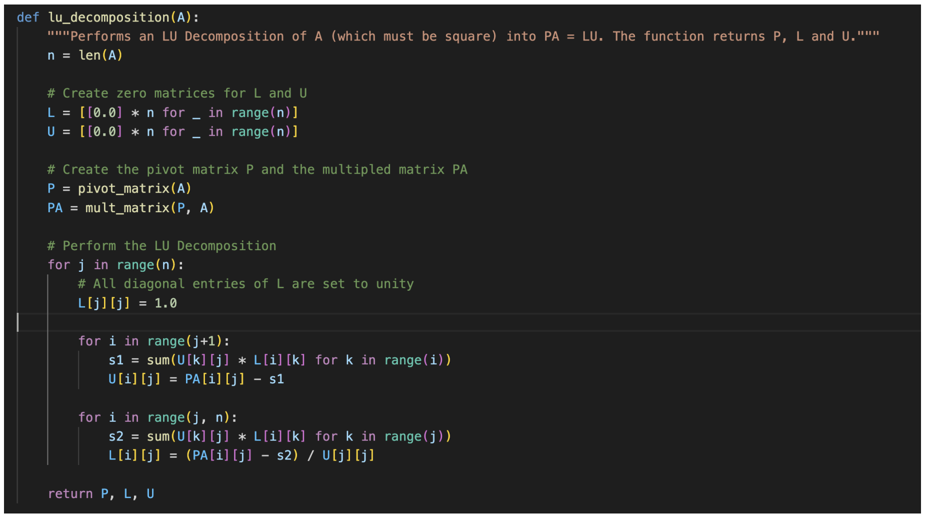
Jack, Rob. “Lecture 5: Gaussian Elimination / LU Decomposition.” University of Cambridge. 2019.

“Lu Decomposition.” Wikipedia, Wikimedia Foundation, 11 Mar. 2024, en.wikipedia.org/wiki/LU$_decomposition$.

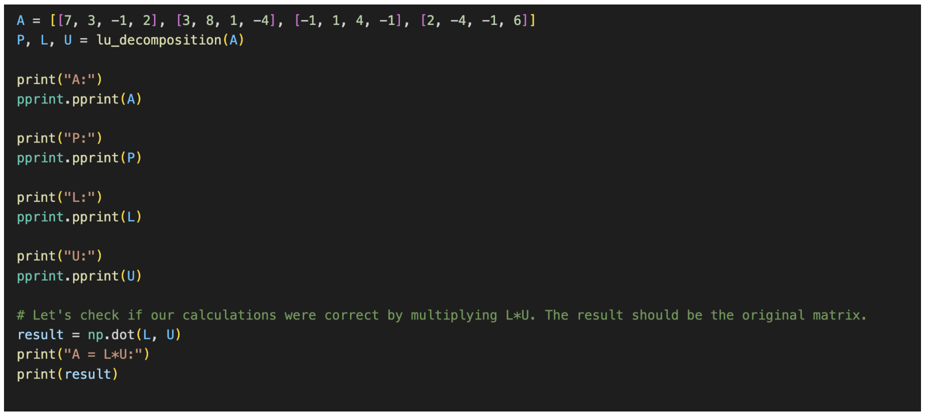
10 Appendices



```python
import pprint
import numpy as np

def mult_matrix(M, N):
    """Multiply square matrices of same dimension M and N"""

    # Converts N into a list of tuples of columns
    tuple_N = zip(*N)

    # Nested list comprehension to calculate matrix multiplication
    return [[sum(el_m * el_n for el_m, el_n in zip(row_m, col_n)) for col_n in tuple_N] for row_m in M]

def pivot_matrix(M):
    """Returns the pivoting matrix for M, used in Doolittle's method."""
    m = len(M)

    # Create an identity matrix, with floating point values
    id_mat = [[float(i == j) for i in range(m)] for j in range(m)]

    # Rearrange the identity matrix such that the largest element of each column of M is placed on the diagonal of of M
    for j in range(m):
        row = max(range(j, m), key=lambda i: abs(M[i][j]))
        if j != row:
            # Swap the rows
            id_mat[j], id_mat[row] = id_mat[row], id_mat[j]

    return id_mat
```



```python
def lu_decomposition(A):
    """Performs an LU Decomposition of A (which must be square) into PA = LU. The function returns P, L and U."""
    n = len(A)

    # Create zero matrices for L and U
    L = [[0.0] * n for _ in range(n)]
    U = [[0.0] * n for _ in range(n)]

    # Create the pivot matrix P and the multipled matrix PA
    P = pivot_matrix(A)
    PA = mult_matrix(P, A)

    # Perform the LU Decomposition
    for j in range(n):
        # All diagonal entries of L are set to unity
        L[j][j] = 1.0

        for i in range(j+1):
            s1 = sum(U[k][j] * L[i][k] for k in range(i))
            U[i][j] = PA[i][j] - s1

        for i in range(j, n):
            s2 = sum(U[k][j] * L[i][k] for k in range(j))
            L[i][j] = (PA[i][j] - s2) / U[j][j]

    return P, L, U
```



```python
A = [[7, 3, -1, 2], [3, 8, 1, -4], [-1, 1, 4, -1], [2, -4, -1, 6]]
P, L, U = lu_decomposition(A)

print("A:")
pprint.pprint(A)

print("P:")
pprint.pprint(P)

print("L:")
pprint.pprint(L)

print("U:")
pprint.pprint(U)

# Let's check if our calculations were correct by multiplying L*U. The result should be the original matrix.
result = np.dot(L, U)
print("A = L*U:")
print(result)
```