# DOOLITTLE

# ALGORITHM

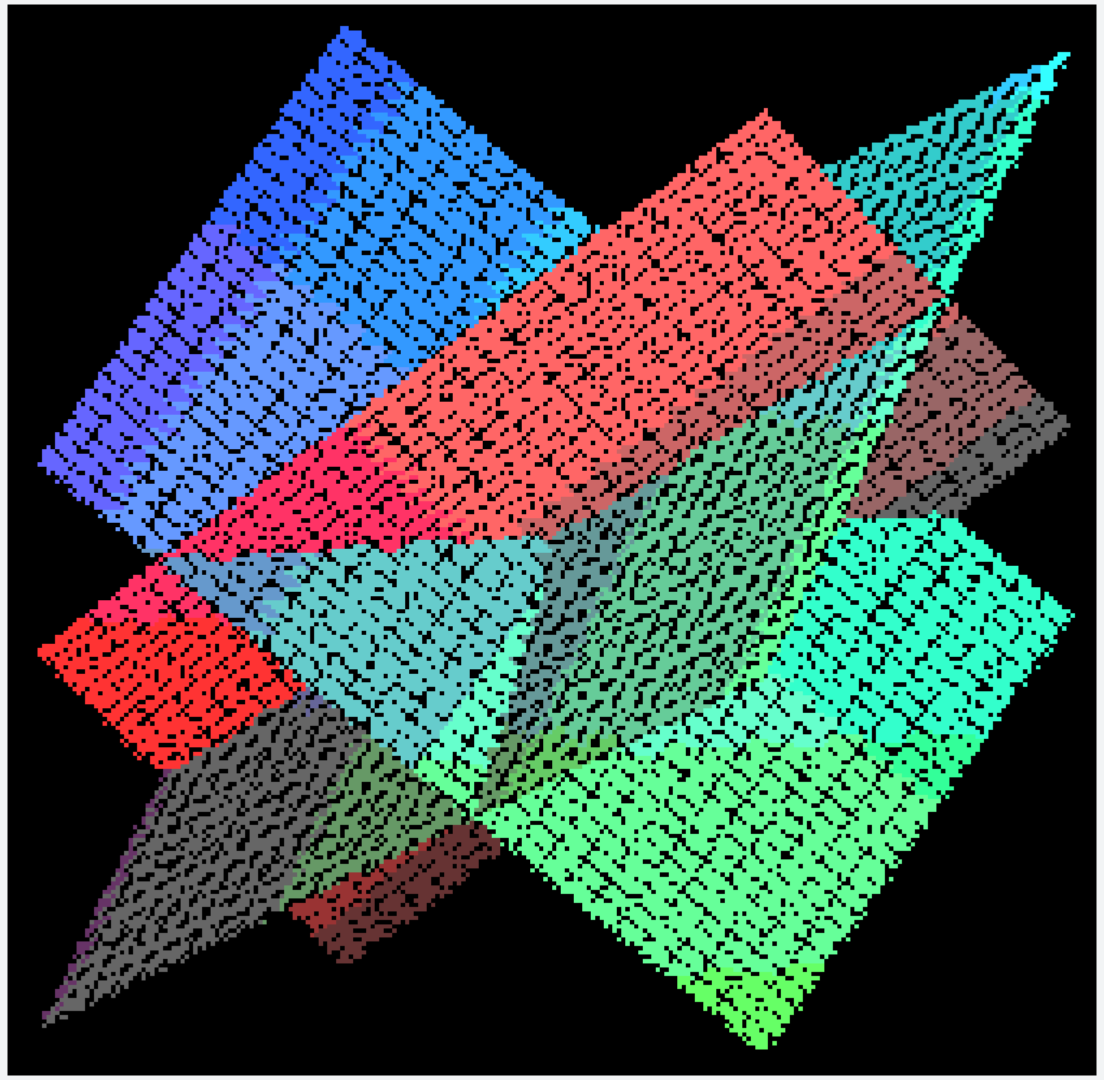
## LU DECOMPOSITION

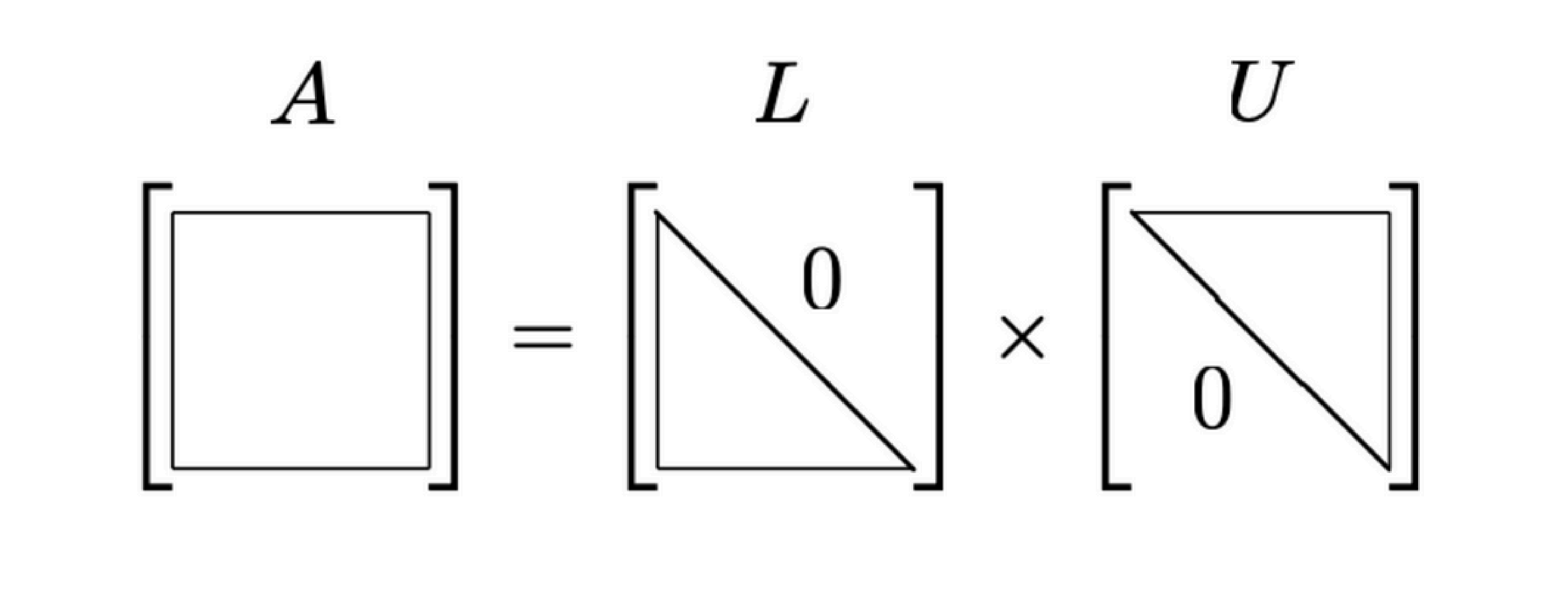
MOHAMMED EL AMINE
JOUIBI

# TABLE OF CONTENT

# LU DECOMPOSITION

In linear algebra, we often encounter systems of linear equations represented by matrices. These systems can arise from various real-world scenarios, such as circuit analysis, structural engineering, and optimization problems.






Suppose we have the system of equations
AX = B.
The LU decomposition is another approach designed to exploit triangular systems.
We suppose that we can write
A = LU
where L is a lower triangular matrix and U is an upper triangular matrix. Our aim is to find L and
U and once we have done so we have found an LU decomposition of A.

# DOLITTLE ALGORITHM

Doolittle’s method provides an alternative way to factor A into an LU decomposition without going through the hassle of Gaussian Elimination.

For a general n×n matrix A, we assume that an LU decomposition exists, and write the form of L and U explicitly. We then systematically solve for the entries in L and U from the equations that result from the multiplications necessary for A=LU.

**Terms of U matrix are given by:**

$$\forall j$$

$$i = 0 \rightarrow U_{ij} = A_{ij}$$

$$i > 0 \rightarrow U_{ij} = A_{ij} - \sum_{k=0}^{i-1} L_{ik} U_{kj}$$

**And the terms for L matrix:**

$$\forall i$$

$$j = 0 \rightarrow L_{ij} = \frac{A_{ij}}{U_{jj}}$$

$$j > 0 \rightarrow L_{ij} = \frac{A_{ij} - \sum_{k=0}^{j-1} L_{ik} U_{kj}}{U_{jj}}$$

$$
\begin{bmatrix} A00 & A01 & A02 \\ A10 & A11 & A12 \\ A20 & A21 & A22 \end{bmatrix}
=
\overbrace{\begin{bmatrix} 1 & 0 & 0 \\ L10 & 1 & 0 \\ L20 & L21 & 1 \end{bmatrix}}^{\text{Lower Triangular}}
\underbrace{\begin{bmatrix} U00 & U01 & U02 \\ 0 & U11 & U12 \\ 0 & 0 & U22 \end{bmatrix}}_{\text{Upper Triangular}}
$$

# BENEFITS OF USING DOOLITTLE ALGORITHM

Doolittle's algorithm with partial pivoting minimizes the potential for division by very small numbers

**NUMERICAL STABILITY**

It has a lower computational cost compared to other methods for LU decomposition, particularly for dense matrices.

**EFFICIENCY**

The algorithm is conceptually straightforward and easy to implement

**SIMPLICITY**

# CODE IMPLEMENTATION:

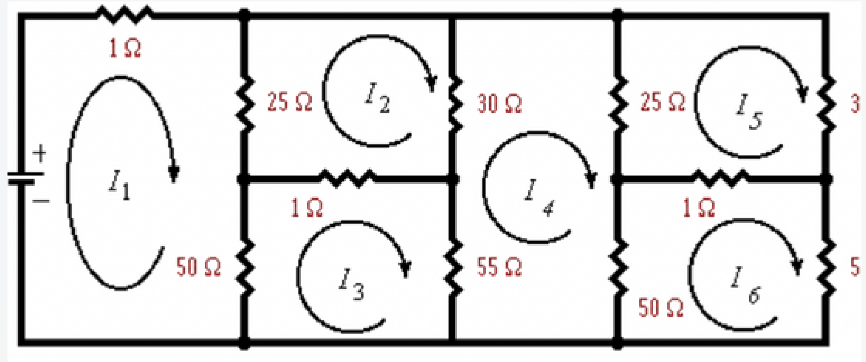
```python
def lu_decomposition(A):
    """Performs an LU Decomposition of A (which must be square) into PA = LU. The function returns P, L and U."""
    n = len(A)

    # 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
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:
```

```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)
```

# METHOD APPLICATION

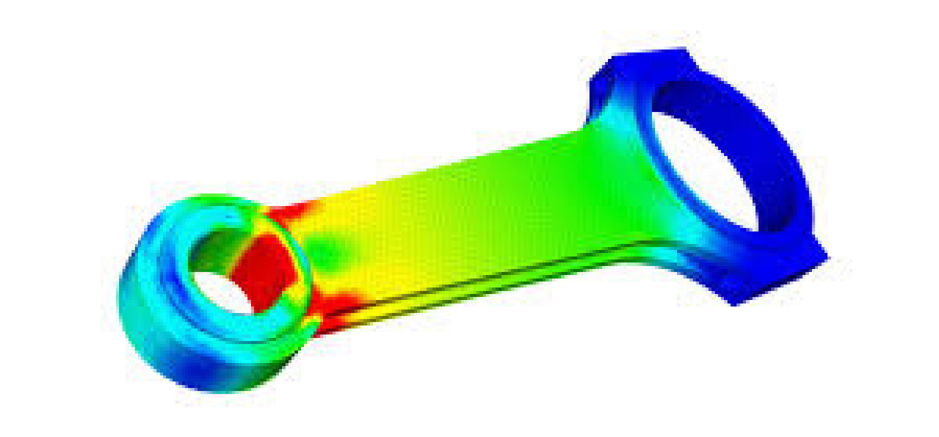
## circuit analysis




In circuit analysis, LU decomposition is essential for solving systems of linear equations derived from Kirchhoff's laws. By decomposing the coefficient matrix into lower and upper triangular matrices, LU decomposition enables efficient and stable solutions to these equations, facilitating the determination of voltages and currents in complex electrical circuits. Its use ensures accurate circuit analysis, aiding in circuit design, optimization, and troubleshooting processes.

## *Finite Element Analysis*

In structural engineering and other fields using FEA, LU decomposition is utilized to solve the system of equations resulting from discretization of partial differential equations. It enables the efficient solution of large-scale linear systems representing complex structural and mechanical problems.



# THANK YOU