```
import numpy as np

array = np.array([[8, 4, -1, 11],
                  [-2, 5, 1, 4],
                  [2, -1, 6, 7]])


def lu_decomposition(A):
    n = len(A)
    # Create zero matrices for L and U
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    # Decomposing matrix into Upper and Lower triangular matrices
    for i in range(n):
        #Find upper triangle
        for k in range(i, n):
            sum_upper = sum(L[i][j] * U[j][k] for j in range(i))
            U[i][k] = A[i][k] - sum_upper

        #Find lower triangle
        for k in range(i, n):
            if i == k:
                L[i][i] = 1  # Diagonal as 1
            else:
                sum_lower = sum(L[k][j] * U[j][i] for j in range(i))
                L[k][i] = (A[k][i] - sum_lower) / U[i][i]

    return L, U

def __forward_substitution(L, U, b):
    n = len(L)
    d = np.zeros(n)
    d[0] = b[0]
    #forward substitution
    for i in range(1, n):
        d[i] = b[i] - (sum(L[i][j]*d[j] for j in range(0, i)))

    #backwards substitution
    x = np.zeros(n)
    x[n-1] = d[n-1]/U[n-1][n-1]
    for i in range(n-2, -1, -1):
        x[i] = (d[i] - sum(U[i][j]*x[j] for j in range(i+1, n)))/U[i][i]

    return x

def solve(A):
    b = A[:, -1]
    A = A[:, :-1]
    L, U = lu_decomposition(A)
    return __forward_substitution(L, U, b)

print(f'Solving {array}')
x = solve(array)
print(f'x1 = {x[0]}\nx2 = {x[1]}\nx3 = {x[2]}')
```