NUMERICAL ANALYSIS

PROJECT – 9

Name – Mrinal Saini	Roll No. – BS1920	Semester – II

LU Decomposition: A non-singular matrix A has LU decomposition if it can be written as

$$A = LU$$

Where L is a lower triangular matrix and U is an upper triangular matrix. Further, if U has $1's$ on its diagonal it is called **crout's decomposition.**

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

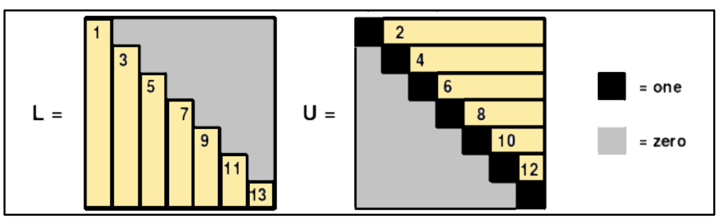
Crout's Decomposition Algorithm: By definition of matrix multiplication we have

$$\boldsymbol{a_{ij} = \sum_{k=1}^{n} l_{ik}u_{kj}}$$

Since, $l_{ik} = 0$ if $k > i$, and $u_{kj} = 0$ if $k < j$. Therefore, the above sum if effectively

$$\boldsymbol{a_{ij} = \sum_{k=1}^{\min\{i,j\}} l_{ik}u_{kj}}$$

Now, computations are done to find the $l_{ij}'s$ and the $u_{ij}'s$ and the order in which the computations are done are shown in the diagram below:



To find $l_{i1}'s$ consider

$$\boldsymbol{a_{i1} = l_{i1}u_{11} = l_{i1}}$$

Since diagonal entries of U are 1. Once $l_{i1}'s$ are computed, $u_{1i}'s$ can be computed by considering

$$\boldsymbol{u_{1i} = \frac{a_{1i}}{l_{11}}}$$

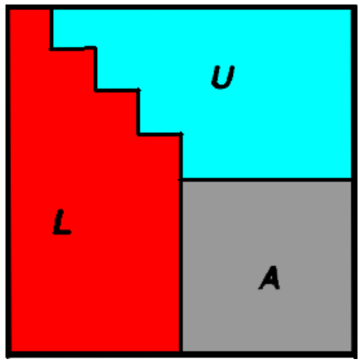
Next compute $l_{i2}'s$ and after that $u_{2i}'s$ and so on. The general formulas to compute $l_{ij}'s$ and $u_{ij}'s$ are

$$\boldsymbol{l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}} \qquad (\boldsymbol{i \geq j})$$

$$\boldsymbol{u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}}{l_{ii}}} \qquad (\boldsymbol{i < j})$$

Efficient Implementation of Crout's Decomposition: Notice that L and U have nonzero elements at different locations. The only place where both has nonzero elements is the diagonal, where U has only $1's$. So, we do not need to explicitly store the diagonal entries of U. This lets us store L and U in a single nxn matrix. Also, observe that a_{ij} for $i < j$ is required to compute only u_{ij}. Similarly, a_{ij} for $i \geq j$ is required to compute only l_{ij}. Thus, once u_{ij} is computed (for $i < j$) we can throw away a_{ij}. Similarly, for the case $i \geq j$. This suggests that we overwrite A with L and U.

Here is how the algorithm overwrites A:



Solving System of Equations using LU Decomposition: If $A = LU$ the $AX = B$ can be solved as follows. First write the system as two triangular units

$$LX' = B, \text{ where } X' = UX$$

Being triangular the system can be solved by forward and backward substitution Apply forward substitution to solve for X' from first equation and then apply backward substitution to solve for X from the second equation.

Project: Implement the efficient version of Crout's decomposition. Your software should also be able to solve a system $AX = B$ by forward and backward substitution.

Python Code:

```python
# PROJECT - 9

## Importing Libraries
import numpy as np

## Crout's Decomposition
def crouts_decomposition(A,n):
    for i in range(0,n):
        for j in range(i,n):                #overwrites the column of A with the column of L
            for k in range(0,i):
                A[j,i]=A[j,i]-A[j,k]*A[k,i]
        for j in range(i+1,n):              #overwrites the row of A with the row of U
            for k in range(0,i):
                A[i,j]=A[i,j]-A[i,k]*A[k,j]
        if A[i,i]==0:
            for j in range(i+1,n):
                if A[i,j]!=0:
                    return i
        else:
            for j in range(i+1,n):
                A[i,j]=A[i,j]/A[i,i]

    return A
```

```python
## Solving the System of Equations
def solve(A,B,n):
    for i in range(0,n):                        #computes X' where X' = UX
        for k in range(0,i):
            B[i]=B[i]-B[k]*A[i,k]
        if A[i,i]==0:
            if B[i]!=0:
                return 0                        #returns 0 if division of non-zero number by zero is encountered
            else:
                B[i]=0                          #puts zero in place of B[i] if division of zero by zero is encountered
        else:
            B[i]=B[i]/A[i,i]

    for i in range(n-1,-1,-1):                  #computes X
        for k in range(n-1,i,-1):
            B[i]=B[i]-B[k]*A[i,k]
    return B


## Taking the matrix input from the user
n=int(input("Enter the order of coefficient matrix "))
print("Enter the values of the coefficient matrix (row-wise and separated by space)")
A=np.matrix(list(map(float,input().split()))).reshape(n,n)
print("Enter the constants (separated by space)")
B=np.array(list(map(float,input().split())))

## Computing Crout's Decomposition and Solving the system of Equations
print("\nOBJECTIVE : To solve the system of linear equations Ax=B using LU Decomposition Method")
print("where A is the coefficient matrix = ")
print(A)
print("and B is the rhs vector = ", B, "\n")
A = crouts_decomposition(A,n)
if type(A)==int:
    print("ERROR\nCrout's Decomposition does not exist")
else:
    print("A after efficient implementation = ")
    print(A)
    print("(with the part above diagonal being U and the rest being L such that A = LU)\n")
    B = solve(A,B,n)
    if type(B)==int:
        print("System is Inconsistent")
    else:
        print("System is Consistent\nSolution to above system = ",B)
```

Output:

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\Notes\Numerical Analysis\Projects\Project9_LUDecomposition\Project9_LUDecomposition.py
Enter the order of coefficient matrix 4
Enter the values of the coefficient matrix (row-wise and separated by space)
1 3 3 3 5 8 1 2 7 4 3 0 3 5 1 2
Enter the constants (separated by space)
13 17 17 12

OBJECTIVE : To solve the system of linear equations Ax=B using LU Decomposition Method
where A is the coefficient matrix =
[[1. 3. 3. 3.]
 [5. 8. 1. 2.]
 [7. 4. 3. 0.]
 [3. 5. 1. 2.]]
and B is the rhs vector =  [13. 17. 17. 12.]

A after efficient implementation =
[[  1.           3.           3.           3.        ]
 [  5.          -7.           2.           1.85714286]
 [  7.         -17.          16.           0.66071429]
 [  3.          -4.           0.           0.42857143]]
(with the part above diagonal being U and the rest being L such that A = LU)

System is Consistent
Solution to above system =  [1. 1. 2. 1.]
>>>
```