EX.2.2.2.a, Sauer3

Find the LU factorization of the given matrix. Check by matrix multiplication.

$$\text{(a)} \begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix}$$

EX.2.2.2.a, Sauer3, solution, Langou

Colab link: `https://colab.research.google.com/drive/1Av8Vn_gwXXKtLulKxGETXHEuK_ER_ldL`

We perform the LU factorization

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix} \begin{matrix} R_1 \leftarrow R_1 - 2R_0 \\ R_2 \leftarrow R_1 - R_0 \\ \rightsquigarrow \end{matrix} \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

We obtain

$$\boxed{L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} 3 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}}$$

We can check that

- U is upper triangular
- L is lower unit triangular
- L times U is A, indeed

$$\begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

not needed for full credit.

We often store the matrices L and U in place of A. In the end, the unit diagonal of L is not stored. (Because this is all ones, we do not need to store these ones, we know there are here.) And then the zeros in the upper part of L are not stored, and the the zeros in the lower part of U are not stored. It looks something like this:

$$\begin{pmatrix} 3 & 1 & 2 \\ 2 & 1 & 0 \\ 1 & 0 & 3 \end{pmatrix}$$

where U is in blue, and L is in red.

The algorithm would run as follows:

$$\begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix} \begin{matrix} R_1 \leftarrow R_1 - 2R_0 \\ R_2 \leftarrow R_1 - 1R_0 \\ \rightsquigarrow \end{matrix} \begin{pmatrix} 3 & 1 & 2 \\ 2 & 1 & 0 \\ 1 & 0 & 3 \end{pmatrix} \begin{matrix} R_2 \leftarrow R_2 + 0R_1 \\ \rightsquigarrow \end{matrix} \begin{pmatrix} 3 & 1 & 2 \\ 2 & 1 & 0 \\ 1 & 0 & 3 \end{pmatrix}$$

not needed for full credit.

```
import numpy as np
import copy
```

```
A = np.array([  [  3.,   1.,   2. ],
                [  6.,   3.,   4. ],
                [  3.,   1.,   5. ] ] )
```

```
# using our bucket algorithm 'lu_no_pivoting'

L, U = lu_no_pivoting( A )

print("L=\n",L)
print("U=\n",U)

# check that A = LU
print("\nL @ U=\n", L @ U)
print("A=\n",A)
print("\n|| A - LU ||_oo / || A ||_oo = ",\
 np.linalg.norm(A - L@U,np.infty)/np.linalg.norm(A,np.infty) )
```

```
L=
 [[1. 0. 0.]
 [2. 1. 0.]
 [1. 0. 1.]]
U=
 [[3. 1. 2.]
 [0. 1. 0.]
 [0. 0. 3.]]

L @ U=
 [[3. 1. 2.]
 [6. 3. 4.]
 [3. 1. 5.]]
A=
 [[3. 1. 2.]
 [6. 3. 4.]
 [3. 1. 5.]]

|| A - LU ||_oo / || A ||_oo =  0.0
```

```
# by hand
U = copy.deepcopy(A)
L = np.eye(3)
print(np.concatenate((L, U, L@U), axis=1))
```

```
print("|| A - LU ||_oo / || A ||_oo = ",\
 f"{np.linalg.norm(A - L@U,np.infty)/np.linalg.norm(A,np.infty):.2e}" )
```

```
[[1. 0. 0. 3. 1. 2. 3. 1. 2.]
 [0. 1. 0. 6. 3. 4. 6. 3. 4.]
 [0. 0. 1. 3. 1. 5. 3. 1. 5.]]
|| A - LU ||_oo / || A ||_oo =  0.00e+00
```

```
# step 1

L[1,0] = 2.; U[1,:] = U[1,:] - L[1,0] * U[0,:]
L[2,0] = 1.; U[2,:] = U[2,:] - L[2,0] * U[0,:]

print(np.concatenate((L, U, L@U), axis=1))
print("|| A - LU ||_oo / || A ||_oo = ",\
 f"{np.linalg.norm(A - L@U,np.infty)/np.linalg.norm(A,np.infty):.2e}" )
```

```
[[1. 0. 0. 3. 1. 2. 3. 1. 2.]
 [2. 1. 0. 0. 1. 0. 6. 3. 4.]
 [1. 0. 1. 0. 0. 3. 3. 1. 5.]]
|| A - LU ||_oo / || A ||_oo =  0.00e+00
```