EX.2.2.4.a, Sauer3

Solve the system by finding the LU factorization and then carrying out the two-step back substitution

$$\text{(a)}\ \begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 3 \end{pmatrix}$$

Note 1: the LU factorization for this matrix was found in EX.2.2.2.a.
Note 2: the two-step back substitution for this linear system of equations is done using Python in CP.2.2.2.a.

EX.2.2.4.a, Sauer3, solution, Langou

Colab link: `https://colab.research.google.com/drive/1YgoHxlDmVqVcQf22NcB4OX17sSTPS4tL`

Not allowed but good to see:
With a little bit of eye-balling, we see that a solution is:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.$$

Solution starts now:
From EX.2.2.2.a, we have the LU factorization of A:

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

So we want to solve $Ax = b$ which is:

$$\begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 3 \end{pmatrix}$$

Since $A = LU$, this is equivalent to solving $LUx = b$:

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 3 \end{pmatrix}$$

Firstly, we solve $Ly = b$ with a forward solve (to get y). $Ly = b$ is

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 3 \end{pmatrix}$$

and so we get (solving for y_0 first, and substituing, and solving for y_1, and then solving for y_2):

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 3 \end{pmatrix}$$

Secondly and finally, We solve $Ux = y$ with a backward solve (to get x). $Ux = y$ is

$$\begin{pmatrix} 3 & 1 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 3 \end{pmatrix}$$

and so we get (solving for x_2 first, and substituing, and solving for x_1, and then solving for x_0):

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.$$

We obtain

$$\boxed{\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix}.}$$

Anything after this is **not needed** for full credit.

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

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

b = np.array([  [  0. ],
                [  1. ],
                [  3. ] ] )
```

```
# solve Ax=b with np.linalg.solve
# and check that Ax is b

x = np.linalg.solve(A, b)

print("With np.linalg.solve, we find x = \n", x)

print("\nWe check that Ax is b. Indeed [ Ax, b ] =\n",\
 np.concatenate((A@x, b), axis=1) )

print("\nrelative backward error: || b - Ax  ||_oo / || b ||_oo =\n",\
 np.linalg.norm(b-A@x,np.infty)/np.linalg.norm(b,np.infty) )
```

```
With np.linalg.solve, we find x =
 [[-1.]
 [ 1.]
 [ 1.]]

We check that Ax is b. Indeed [ Ax, b ] =
 [[0. 0.]
 [1. 1.]
 [3. 3.]]
```

```
relative backward error: || b - Ax  ||_oo / || b ||_oo =
 0.0
```

```python
# using our bucket algorithms
#   `lu_no_pivoting`
#   `forward__substitution`
#   `backward_substitution`

L, U = lu_no_pivoting( A )
y = forward__substitution( L, b )
x = backward_substitution( U, y )

print("With our algorithms, we find\n")
print("y=\n",y)
print("x=\n",x)

print("\nWe check that Ax is b. Indeed [ Ax, b ] =\n",\
 np.concatenate((A@x, b), axis=1) )

print("\nrelative backward error: || b - Ax  ||_oo / || b ||_oo =\n",\
 np.linalg.norm(b-A@x,np.infty)/np.linalg.norm(b,np.infty) )
```

```
With our algorithms, we find

y=
 [[0.]
 [1.]
 [3.]]
x=
 [[-1.]
 [ 1.]
 [ 1.]]

We check that Ax is b. Indeed [ Ax, b ] =
 [[0. 0.]
 [1. 1.]
 [3. 3.]]

relative backward error: || b - Ax  ||_oo / || b ||_oo =
 0.0
```