**CP.2.2.2.a, Sauer3**

Add two-step back substitution to your script from CP.2.2.1, and use it to solve the systems in EX.2.2.4.a.

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



**CP.2.2.2.a, Sauer3, solution, Langou**

See what was already done for **EX.2.2.4.a, Sauer3, solution, Langou**

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

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

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

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

```


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
```

---