CS 111 (S22): Homework 2

Due Monday, April 11th by 11:59 PM

NAME and PERM ID No.: Amey Walimbe, 8594210

UCSB EMAIL: amey@ucsb.edu

Additional Instructions: Please enter the LaTeX command \newpage at the end of each of your answers in your LaTeX source code. This will put each answer on at least one page and make it easier to grade on Gradescope.

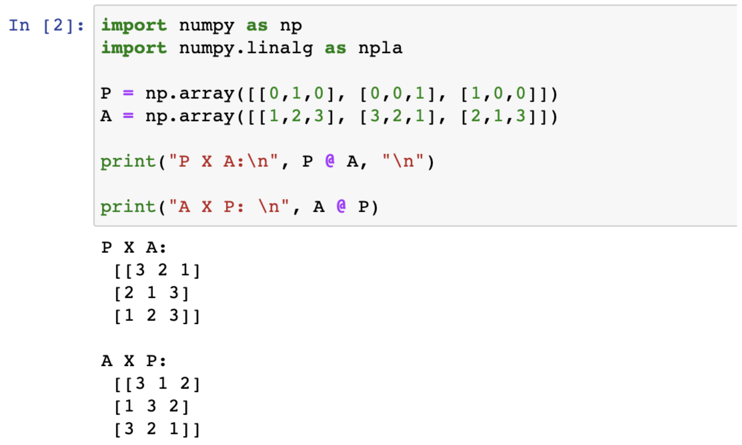
1. *(30 pts)* The following three statements are all **false**. For each one, give a counterexample consisting of a 3-by-3 matrix or matrices (to show that they are indeed false), and show the Python computation that proves that the statement fails (i.e. snapshot or text copy of the actual code and also the aftermath of its execution).

   A good way to solve this problem (other than thinking about Linear Algebra theories) can start with trial-and-error computations on Python: you should get comfortable with this language and environment, which in turn will help with your intuitive thinking about these sorts of problems.

   - If P is a permutation matrix and A is any matrix, then $PA = AP$.
     Answer:

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{pmatrix}$$



```
In [2]: import numpy as np
        import numpy.linalg as npla

        P = np.array([[0,1,0], [0,0,1], [1,0,0]])
        A = np.array([[1,2,3], [3,2,1], [2,1,3]])

        print("P X A:\n", P @ A, "\n")

        print("A X P: \n", A @ P)
```

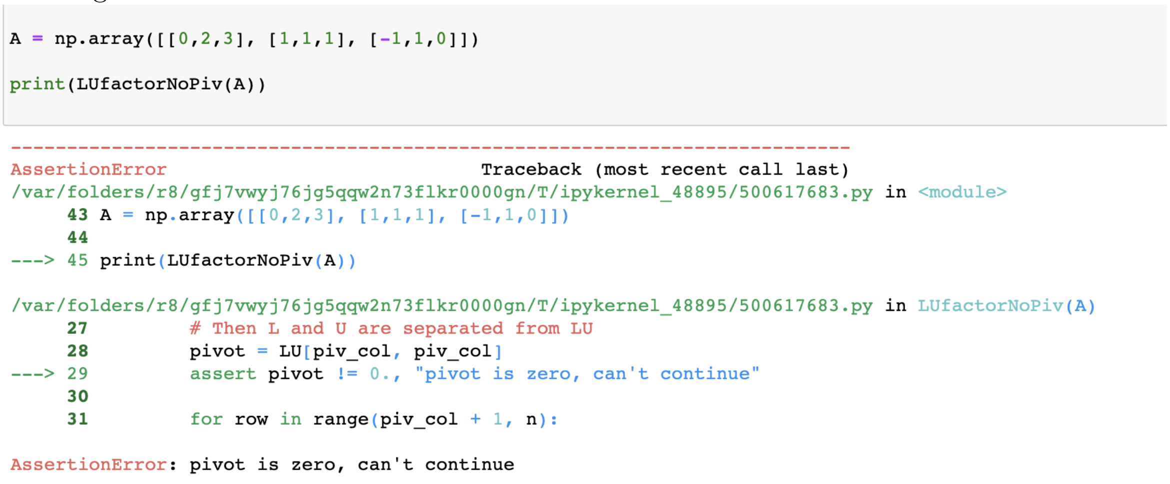
```
P X A:
 [[3 2 1]
 [2 1 3]
 [1 2 3]]

A X P:
 [[3 1 2]
 [1 3 2]
 [3 2 1]]
```

   - If matrix A is nonsingular, then it has a factorization $A = LU$ where L is lower triangular and U is upper triangular.
     Answer:

Let A be the matrix:

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

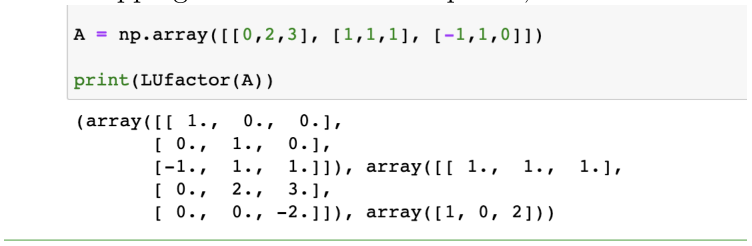
When we try to factor this matrix, we realize that there is a zero in the first pivot position, making it impossible to use the general equation to transform the matrix into upper-triangular form.



```
A = np.array([[0,2,3], [1,1,1], [-1,1,0]])

print(LUfactorNoPiv(A))
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
/var/folders/r8/gfj7vwyj76jg5qqw2n73flkr0000gn/T/ipykernel_48895/500617683.py in <module>
     43 A = np.array([[0,2,3], [1,1,1], [-1,1,0]])
     44 
---> 45 print(LUfactorNoPiv(A))

/var/folders/r8/gfj7vwyj76jg5qqw2n73flkr0000gn/T/ipykernel_48895/500617683.py in LUfactorNoPiv(A)
     27         # Then L and U are separated from LU
     28         pivot = LU[piv_col, piv_col]
---> 29         assert pivot != 0., "pivot is zero, can't continue"
     30 
     31         for row in range(piv_col + 1, n):

AssertionError: pivot is zero, can't continue
```

In order to counter this, we need to include a permutation matrix that swaps the rows. This swapping allows for non-zero pivots, and we can then proceed with factoring.



```
A = np.array([[0,2,3], [1,1,1], [-1,1,0]])

print(LUfactor(A))
```

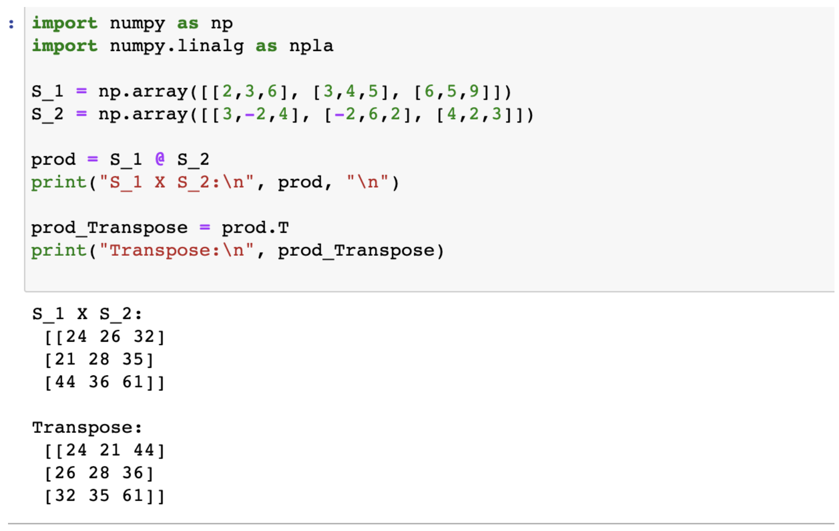
```
(array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [-1.,  1.,  1.]]), array([[ 1.,  1.,  1.],
       [ 0.,  2.,  3.],
       [ 0.,  0., -2.]]), array([1, 0, 2]))
```

From the image, we can see that the permutation matrix is represented by array([1, 0, 2]), which shows how the rows had to be swapped.

- The product of two symmetric matrices is a symmetric matrix.
  Answer:

$$S_1 = \begin{pmatrix} 2 & 3 & 6 \\ 3 & 4 & 5 \\ 6 & 5 & 9 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} 3 & -2 & 4 \\ -2 & 6 & 2 \\ 4 & 2 & 3 \end{pmatrix}$$



```
import numpy as np
import numpy.linalg as npla

S_1 = np.array([[2,3,6], [3,4,5], [6,5,9]])
S_2 = np.array([[3,-2,4], [-2,6,2], [4,2,3]])

prod = S_1 @ S_2
print("S_1 X S_2:\n", prod, "\n")

prod_Transpose = prod.T
print("Transpose:\n", prod_Transpose)
```

```
S_1 X S_2:
 [[24 26 32]
 [21 28 35]
 [44 36 61]]

Transpose:
 [[24 21 44]
 [26 28 36]
 [32 35 61]]
```

2. *(20 pts)* Write the following matrix in the form $A = LU$, where L is a unit lower triangular matrix (that is, a lower triangular matrix with ones on the diagonal) and U is an upper triangular matrix. You can check your answer using Python, but for this exercise, you need to also show the steps you took to get to your answer with some explanation to go with them. In particular, if you have to use pivoting, then I need you to also tell me what the permutation matrix you used was.

$$A = \begin{pmatrix} 5 & 3 & 3 \\ 3 & 5 & 3 \\ 3 & 3 & 5 \end{pmatrix}$$

- Step 1. Eliminate first element in row index 1.
  $r_1 = r_1 - (3/5)r_0$, so $m = 3/5$:

$$\begin{pmatrix} 5 & 3 & 3 \\ 0 & 16/5 & 6/5 \\ 3 & 3 & 5 \end{pmatrix}$$

- Step 2. Eliminate first element in row index 2.
  $r_2 = r_2 - (3/5)r_0$, so $m = 3/5$:

$$\begin{pmatrix} 5 & 3 & 3 \\ 0 & 16/5 & 6/5 \\ 0 & 6/5 & 16/5 \end{pmatrix}$$

- Step 3. Eliminate second element in row index 2.
  $r_2 = r_2 - (6/16)r_1$, so $m = 3/8$:

$$\begin{pmatrix} 5 & 3 & 3 \\ 0 & 16/5 & 6/5 \\ 0 & 0 & 2 \end{pmatrix}$$

Now, what we have left is U. In order to form L, we take all the multipliers and place them into their correct lower triangular spots.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 3/5 & 1 & 0 \\ 3/5 & 3/8 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 5 & 3 & 3 \\ 0 & 16/5 & 6/5 \\ 0 & 0 & 11/4 \end{pmatrix}$$

So, writing in the form $A = LU$

$$\begin{pmatrix} 5 & 3 & 3 \\ 3 & 5 & 3 \\ 3 & 3 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3/5 & 1 & 0 \\ 3/5 & 3/8 & 1 \end{pmatrix} \begin{pmatrix} 5 & 3 & 3 \\ 0 & 16/5 & 6/5 \\ 0 & 0 & 11/4 \end{pmatrix}$$

3. *(20 pts)* Similar to the above exercise, write this matrix in the form $A = LU$. Again, you can check your answer using Python, but for this exercise, you need to also show the steps you took to get to your answer with some explanation to go with them. In particular, if you have to use pivoting, then I need you to also tell me what the permutation matrix you used was.

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

- Step 1. Swap Row 0 and 2, because row 0 has a zero in the pivot position, and so does row 3. Their swapped positions now allow for non-zeroes in pivot positions.

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

- Step 2. Eliminate first element in row index 1.
  $r_1 = r_1 - (-1)r_0$, so $m = -1$:

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

- Step 2. Eliminate first element in row index 2.
  $r_2 = r_2 - (0)r_1$, so $m = 0$:

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

- Step 3. Eliminate second element in row index 2.
  $r_2 = r_2 - (1)r_1$, so $m = 1$:

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

Now, what we have left is U. In order to form L, we take all the multipliers and place them into their correct lower triangular spots.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} -1 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

So, writing in the form $A = LU$

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

The permutation matrix used: (Swapped row 2 and 0)

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

4. *(30 pts)* Write `Usolve()`, analogous to `Lsolve()` in the class lecture file `lect04_A=LU.ipynb` (this is the demo file used for **Lecture 4**), to solve an upper triangular system $Ux = y$. Your submission can just be the code for the *function definition* of `Usolve()`.

   Obviously, you should also check your work beforehand - this is best done in the `lect04_A=LU.ipynb` file (provided for you in the same place you got these instructions). To best utilize this, you should run the .ipynb file on Jupyter Notebook.

   Warning: Notice that, unlike in `Lsolve()`, the diagonal elements of U don't have to be equal to one. Test your answer, both by itself and with `LUsolve()`, and turn in the result.

   Hint: *(You don't have to use this hint, but if you do, here it is...)* Loops can be run backward in Python, say from $n-1$ down to 0, by writing

```
for i in reversed(range(n)):
```

   Answer:

```
In [31]: def Usolve(U, y):
             """Backward solve an upper triangular system Ux = y for x
             Parameters:
               U: the matrix, must be square, upper triangular, with nonzeros on the diagonal
               y: the right-hand side vector
             Output:
               x: the solution vector to U @ x == y

             """
             m,n = U.shape # m = row, n = col
             assert m == n, "matrix L must be square"

             x = y.astype(np.float64).copy()

             for col in reversed(range(n)):
                 x[col] /= U[col, col]
                 x[:col] -= x[col] * U[:col, col]
             return x
```