Math 271.1: Exercise 2 (#1)

INSTRUCTION: Solving a Linear System Using LU Factorization

(a) Write a program that performs LU factorization (without pivoting)
(b) Solve for x using forward and backward substitution
(c) Verify your solution by comparing with the direct solver output

```
import numpy as np # for numerical operations
from IPython.display import display # for displaying outputs nicely
```

Initialize the system A and b

- Matrix A initialization
- Vector b initialization

```
# Creates coefficient matrix A
A = np.array([
    [2, -1, 1, 3],
    [4, 1, 0, 1],
    [-2, 5, 3, -1],
    [1, 0, 2, 4]
], dtype=float)

display(A)
```

```
array([[ 2., -1.,  1.,  3.],
       [ 4.,  1.,  0.,  1.],
       [-2.,  5.,  3., -1.],
       [ 1.,  0.,  2.,  4.]])
```

```
# Creates right-hand side vector b
b = np.array([8, 7, 1, 10], dtype=float)

display(b)
```

```
array([ 8.,  7.,  1., 10.])
```

(a) LU Factorization without pivoting

- we take note of the number of rows in matrix A (indicates the elimination steps we need)
- variables:
  - L for Lower Triangular
  - Upper Triangular matrix

Logic:

- Start with identity matrix (diagonals of A)
- Start U as a copy of A
- Implemeny elimination process:
  - For each pivot column k and for each i below the pivot, compute the multiplier (mul)
  - Update L using the multiplier
  - Update U by eliminating entries below diagonal

No pivoting use because there's no row swapping and the rows are processes in their original order

```
n = A.shape[0]
L = np.eye(n) #Starts with identity matrix of size n, then gets filled during elimination
U = A.copy().astype(float) # initialized as A then will get updated through elimination

for k in range(n-1): # For each pivot column
    for i in range(k+1, n): # For each row below pivot
```

```
        if U[k, k] == 0:
            raise ValueError("Zero pivot encountered.")

        # Compute for the multiplier
        mul = U[i, k] / U[k, k]

        # Eliminate entry in U
        L[i, k] = mul
        U[i, k:n] = U[i, k:n] - mul * U[k, k:n]

display(L, U)
```

```
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [ 2.        ,  1.        ,  0.        ,  0.        ],
       [-1.        ,  1.33333333,  1.        ,  0.        ],
       [ 0.5       ,  0.16666667,  0.275     ,  1.        ]])
array([[ 2.        , -1.        ,  1.        ,  3.        ],
       [ 0.        ,  3.        , -2.        , -5.        ],
       [ 0.        ,  0.        ,  6.66666667,  8.66666667],
       [ 0.        ,  0.        ,  0.        ,  0.95      ]])
```

Check: L @ U should equal to the matrix A

```
# Check if L @ U equals original A
print("\nL @ U = A:")
LU_product = L @ U
print(LU_product)

print("\nOriginal A:")
print(A)

print("\nCheck:", np.allclose(LU_product, A))
```

```

L @ U = A:
[[ 2.00000000e+00 -1.00000000e+00  1.00000000e+00  3.00000000e+00]
 [ 4.00000000e+00  1.00000000e+00  0.00000000e+00  1.00000000e+00]
 [-2.00000000e+00  5.00000000e+00  3.00000000e+00 -1.00000000e+00]
 [ 1.00000000e+00 -2.77555756e-17  2.00000000e+00  4.00000000e+00]]

Original A:
[[ 2. -1.  1.  3.]
 [ 4.  1.  0.  1.]
 [-2.  5.  3. -1.]
 [ 1.  0.  2.  4.]]

Check: True
```

(b) Solving x using forward and backward substitution

- solve Ly = b using forward substitution
- Solve Ux = y using backward substitution
- Compare with linalg.solve() for verification

```
## forward substitution to solve Ly = b
n = len(b)
y = np.zeros(n)

for i in range(n):
    sum_val = np.dot(L[i, :i], y[:i])
    y[i] = (b[i] - sum_val) / L[i, i]
    print(f"Iteration {i}: y[{i}] = ({b[i]} - {sum_val}) / {L[i,i]} = {y[i]}")

print("\nResult y:", y)
```

```
Iteration 0: y[0] = (8.0 - 0.0) / 1.0 = 8.0
Iteration 1: y[1] = (7.0 - 16.0) / 1.0 = -9.0
Iteration 2: y[2] = (1.0 - -20.0) / 1.0 = 21.0
Iteration 3: y[3] = (10.0 - 8.275) / 1.0 = 1.7249999999999996

Result y: [ 8.    -9.    21.     1.725]
```

```
## backward substitution to solve Ux = y
n = len(y)
```

```
    x = np.zeros(n)

    for i in range(n-1, -1, -1):
        sum_val = np.dot(U[i, i+1:], x[i+1:])
        x[i] = (y[i] - sum_val) / U[i, i]
        print(f"Iteration {i}: x[{i}] = ({y[i]} - {sum_val}) / {U[i,i]} = {x[i]}")

    print("\nResult x:", x)
```

```
Iteration 3: x[3] = (1.7249999999999996 - 0.0) / 0.9499999999999997 = 1.8157894736842106
Iteration 2: x[2] = (21.0 - 15.736842105263158) / 6.666666666666666 = 0.7894736842105264
Iteration 1: x[1] = (-9.0 - -10.657894736842106) / 3.0 = 0.5526315789473687
Iteration 0: x[0] = (8.0 - 5.684210526315789) / 2.0 = 1.1578947368421053

Result x: [1.15789474 0.55263158 0.78947368 1.81578947]
```

Verify: Solution

- Shows computed x solution
- Verifies that Ax = b
- Compare it with the Numpy's direct solver

```
print("Solution x: ", x)
print("Check Ax = b:", np.allclose(A @ x, b))
print("Verification:", np.allclose(x, np.linalg.solve(A, b)))
```

```
Solution x:  [1.15789474 0.55263158 0.78947368 1.81578947]
Check Ax = b: True
Verification: True
```