```python
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 19 16:13:40 2019
HW4
@author: Ryan Dalby
"""
import numpy as np

def LUDecomposition(A):
    """
    Given an A matrix will return an L and U matrix
    """
    n = np.shape(A)[0] #assuming nxn matrix
    U = np.array(A) #make copy of A to store upper triangular form which is our U matrix
    L = np.eye(n)#makes empty diagonal array with ones in the diagonal to store factors fr

    #Get L and U matricies using naive gauss elimination steps of forward elimination and
    for k in range(n - 1):
        for i in range(k + 1, n):
            s = U[i,k] / U[k,k]
            L[i,k] = s
            for j in range(k, n):
                U[i,j] = U[i,j] - s * U[k,j]
    return [L,U]

def LUSolve(L, U, b):
    """
    Given the L and U decomposition matricies of a matrix A and a corresponding b, will sc
    """

    n = np.shape(L)[0] #assuming nxn matrix
    #Perform forward substitution (solve Ld = b for d)
    d = np.zeros(shape = (n,1), dtype='float') #create b vector of correct size
    d[0] = b[0]/L[0,0] #index first element out of b vector and set it to the solutin valu
    for i in range(0, n):
        s = 0.0
        for j in range(0,i):
            s = s + L[i,j] * d[j]
        d[i] = (b[i] - s) / L[i,i]


    #Perform back substitution (solve Ux = d for x)
    xSol = np.zeros(shape = (n,1), dtype='float') #create X vector of correct size
    xSol[n-1] = b[n-1] / U[n-1,n-1] #index last element of xSol and set it to the solution
    for i in range(n-1, -1, -1):
        s = 0.0
        for j in range(i+1, n):
            s = s + U[i,j] * xSol[j]
        xSol[i] = (d[i] - s) / U[i,i]
    return xSol
```

```
def LUDecomposeAndSolve(A, b):
    """
    Given A and b will solve Ax = b for x by decomposing A into L and U and solving for x
    Returns xAns and the L and U matricies
    """
    LU = LUDecomposition(A) #get L an U matricies that correspond with A
    xAns = LUSolve(LU[0], LU[1], b)
    return xAns, LU[0], LU[1]


#Do exercise 1b
print("Exercise 1b:\n")
A = np.array([[8,4,-1],[-2,5,1],[2,-1,6]], dtype='float')
b = np.array([[11],[4],[7]], dtype='float')
ans1b = LUDecomposeAndSolve(A, b)
print("x =\n{}\nL =\n{}\nU ={}\n".format(*ans1b))

#Do exercise 2b
print()
print("Exercise 2b:\n")

g = 9.81#m/s^2
m1 = 2#kg
m2 = 3#kg
m3 = 2.5#kg
A2 = np.array([[30, -20, 0], [-20,30,-10], [0,-10,10]], dtype='float')
b2b = np.array([[m1*g], [m2*g], [m3*g]], dtype='float')
L2,U2 = LUDecomposition(A2)
ans2b = LUSolve(L2,U2,b2b)
print("L =\n {}\nU =\n {}\n\nx1 = {}m, x2 = {}m, x3 = {}m\n".format(L2,U2, *ans2b))

print("Exercise 2c:\n")
m1 = 4#kg
m2 = 6#kg
m3 = 5#kg
b2c = np.array([[m1*g], [m2*g], [m3*g]], dtype='float')
ans2c = LUSolve(L2,U2,b2c)
print("x1 = {}m, x2 = {}m, x3 = {}m\n".format(*ans2c))


print()
print("Exercise 3a:\n")
A3 = np.array([[-3,1,12],[6,-1,-1],[6,9,1]])
b3 = np.array([[50], [3], [40]])
ans3a = np.linalg.solve(A3, b3)
print("Using numpy.linalg.solve\n x1 = {}, x2 = {}, x3 = {}".format(*ans3a))
print()

def gaussSeidel(A, b, tolerance, maxiters, initialGuesses):
    """
    Given A, b, a approximate relative error tolerance, max iterations, and a vector of n-
```

```
    will attempt to solve Ax=b for x
    Returns x
    """
    n = np.shape(A)[0] #assuming nxn matrix
    eA = 100 #arbitrary starting eA value
    currentXVals =  np.zeros(shape = (n,1), dtype='float') #create vector to hold our x va
    previousXVals = np.zeros_like(currentXVals, dtype='float')
    for i in range(1,n): #will put intial guesses in their correct spots(intial guesses ar
        currentXVals[i] = initialGuesses[i-1]

    count = 0 #loop number
    while eA > tolerance:
        for i in range(n):
            rowNorm = A[i,:] / A[i,i] #row normalization(A part)
            bNorm = b[i] / A[i,i] #row normalization(b part)
            sumRowNorm = 0 #will hold sum of the rowNorm * currentXVals excluding the x we
            for j in range(n):#will calculate sumRowNorm
                if i==j:
                    continue
                else:
                    sumRowNorm += rowNorm[j] * currentXVals[j]
            currentXVals[i] = bNorm - sumRowNorm #this is the rearranged equation solved f
        eA = np.linalg.norm((abs((currentXVals - previousXVals)/currentXVals) * 100)) #thi
        if count == maxiters:
            raise Exception("Max number of iterations were reached")
        count += 1
        previousXVals = np.array(currentXVals)
    return currentXVals

 #Rearranging Matrix order of equations to get convergence
A3 = np.array([[6,-1,-1],[6,9,1],[-3,1,12]])
b3 = np.array([[3], [40],[50]])

ans3b = gaussSeidel(A3,b3,.0005,50,[0,0])
print("Exercise 3b:\n")
print("Using Gauss Seidel\nx1 = {}, x2 = {}, x3 = {}".format(*ans3b))
```