ME EN 2450 | Ryan Dalby | Assignment HW04

Exercise 9)

$$8x_1 + 4x_2 - x_3 = 11$$
$$-2x_1 + 5x_2 + x_3 = 4$$
$$2x_1 - x_2 + 6x_3 = 7$$

$$\begin{bmatrix} 8 & 4 & -1 \\ -2 & 5 & 1 \\ 2 & -1 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 4 \\ 7 \end{bmatrix}$$

[U] → gauss elimination

$$A = \begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & 0.75 \\ 2 & -1 & 6 \end{bmatrix} \quad b = \begin{bmatrix} 11 \\ 6.75 \\ 7 \end{bmatrix}$$

$S = \frac{-2}{8} = -1/4 = L_{21}$ → This step we eliminate A_{21}

$-2 - (-1/4)(8) = 0$ $\quad 4 - (-1/4)(11) = 6.75$

$5 - (-1/4)(4) = 6$

$1 - (-1/4)(-1) = 0.75$

$$A = \begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & 0.75 \\ 0 & -2 & 6.25 \end{bmatrix} \quad b = \begin{bmatrix} 11 \\ 6.75 \\ 4.25 \end{bmatrix}$$

$S = \frac{2}{8} = 1/4 = L_{31}$ → This step we eliminate A_{31}

$2 - (1/4)(8) = 0$ $\quad 7 - (1/4)(11) = 4.25$

$-1 - (1/4)(4) = -2$

$6 - (1/4)(-1) = 6.25$

$$A = \begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & 0.75 \\ 0 & 0 & 6.5 \end{bmatrix} \quad b = \begin{bmatrix} 11 \\ 6.75 \\ 6.5 \end{bmatrix}$$

$S = \frac{-2}{6} = -1/3 = L_{32}$ → This step we eliminate A_{32}

$-2 - (-1/3)(6) = 0$ $\quad 4.25 - (-1/3)(6.75) = 6.5$

$6.25 - (-1/3)(0.75) = 6.5$

$$U = \begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & 0.75 \\ 0 & 0 & 6.5 \end{bmatrix} \quad d = \begin{bmatrix} 11 \\ 6.75 \\ 6.5 \end{bmatrix}$$

$S_1 = -1/4 \quad d_1 = 11$

$S_2 = 1/4 \quad d_2 = 6.75$

$S_3 = -1/3 \quad d_3 = 6.5$

$U\vec{x} = \vec{d}$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ 1/4 & -1/3 & 1 \end{bmatrix}$$

$L \quad d = b$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ 1/4 & -1/3 & 1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 4 \\ 7 \end{bmatrix}$$

Once we have L and U only have to do following steps below:

$U \quad x = d$

$$\begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & 0.75 \\ 0 & 0 & 6.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 6.75 \\ 6.5 \end{bmatrix}$$

$d_1 = 11$

$-1/4\, d_1 + d_2 = 4$

$d_3 = 6.75$

$1/4\, d_1 - 1/3\, d_2 + d_3 = 7$

$d_3 = 6.5$

$6.5x_3 = 6.5 \quad x_3 = 1$

$6x_2 + 0.75x_3 = 6.75 \quad x_2 = 1$

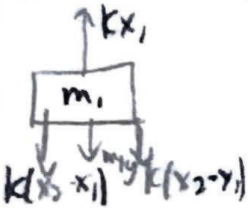
$8x_1 + 4x_2 - x_3 = 11 \quad x_1 = 1$

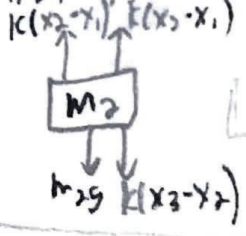
$$\vec{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Exercise 1 b) See code

c) By using the LU decomposition method in code a solution of $\vec{x}=\begin{bmatrix}1\\1\\1\end{bmatrix}$ was found. This matches the solution of the hand calculated solution of part a. In part a L and U were also calculated and in the code I outputted the L and U matricies found using the coded method, both the hand calculated and coded method get an L matrix of $\begin{bmatrix}1&0&0\\-1/4&1&0\\1/4&-1/3&1\end{bmatrix}$ and a U matrix of $\begin{bmatrix}8&4&-1\\0&6&0.75\\0&0&6.5\end{bmatrix}$

Exercise 2 a) $k = 10\ kg/s^2$ $g = 9.81\ m/s^2$





$+\uparrow \sum F_y = 0 = -2k(x_2-x_1) - m_2g - k(x_3-x_2)$

$+\uparrow \sum F_y = 0 = kx_1 - 2k(x_2-x_1) - m_1g$

$+\uparrow \sum F_y = k(x_3-x_2) - m_3g$

$m_1g = kx_1 - 2k(x_2-x_1)$

$m_1g = kx_1 - 2kx_2 + 2kx_1$

$m_2g = 2k(x_2-x_1) - k(x_3-x_2)$

$m_2g = 2kx_2 - 2kx_1 - kx_3 + kx_2$

$kx_3 - kx_2 = m_3g$

$3kx_1 - 2kx_2 + 0 = m_1g$

$-2kx_1 + 3kx_2 - kx_3 = m_2g$

$0 - kx_2 + kx_3 = m_3g$

$$\underbrace{\begin{bmatrix}30&-20&0\\-20&30&-10\\0&-10&10\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}}_{\vec{x}}=\underbrace{\begin{bmatrix}m_1g\\m_2g\\m_3g\end{bmatrix}}_{\vec{b}}$$

b) See code

c) See code

Exercise 3

a) See code

b) See code

$-3x_1 + x_2 + 12x_3 = 50$

$6x_1 - x_2 - x_3 = 3$

$6x_1 + 9x_2 + x_3 = 40$

$A=\begin{bmatrix}-3&1&12\\6&-1&-1\\6&9&1\end{bmatrix}$ $b=\begin{bmatrix}50\\3\\40\end{bmatrix}$

Rearrange to converge: $A=\begin{bmatrix}6&-1&-1\\6&9&1\\-3&1&12\end{bmatrix}$ $\begin{bmatrix}3\\40\\50\end{bmatrix}$

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW4/HW4.py', wdir='C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW4')
Exercise 1b:

x =
[[1.]
 [1.]
 [1.]]
L =
[[ 1.          0.          0.        ]
 [-0.25        1.          0.        ]
 [ 0.25       -0.33333333  1.        ]]
U =[[ 8.    4.   -1.  ]
 [ 0.    6.    0.75]
 [ 0.    0.    6.5 ]]


Exercise 2b:

L =
 [[ 1.          0.          0.        ]
 [-0.66666667  1.          0.        ]
 [ 0.         -0.6         1.        ]]
U =
 [[ 30.         -20.           0.        ]
 [  0.          16.66666667 -10.        ]
 [  0.           0.           4.        ]]

x1 = [7.3575]m, x2 = [10.05525]m, x3 = [12.50775]m

Exercise 2c:

x1 = [14.715]m, x2 = [20.1105]m, x3 = [25.0155]m


Exercise 3a:

Using numpy.linalg.solve
 x1 = [1.69736842], x2 = [2.82894737], x3 = [4.35526316]

Exercise 3b:

Using Gauss Seidel
x1 = [1.69736821], x2 = [2.82894748], x3 = [4.3552631]

In [2]:
```

```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 so
    """

    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))
```