Pregunta 9

April 25, 2019

Autor:
Lazaro Camasca Edson
Analisis Numerico
Universidad Nacional de Ingenieria
Resuelva el sistema Ax = b utilizando factorización LU

```
In [1]: from toolNick import *
        import numpy as np
```

```
toolNick se ha importado correctamente.
```

0.0.1 Algoritmo para la factoriazacion LU

```
In [2]: def elim_LU(a, b, p = False, v = False):
            #v nos muestra el procedimiento detallado de la eliminacion
            #p se utiliza si requiere pivotacion total al inicio

            A_b = np.c_[a, b] # Matriz aumentada
            (fil, col) = A_b.shape #Guarda el #filas y #columnas
            line = "=============================================="
            print("Matriz aumentada [A|b] al inicio:\n{}\n{}".format(A_b,line))

            if p:
                A_b = pivoteo_Total(A_b, v)

            A = A_b[:, :col - 1] #A sera todo menos la ultima columna
            b = A_b[:, col - 1] #b sera solo la ultima columna

            #Creamos una lista para almacenar las matrices L y P
            L_list = []
            P_list = []  #Matrices de permutacion

            for i in range(fil): # se condiera dim(a)
                if A[i, i] == 0 :
                    P = pivoteo(A, i)

                    if v : print("P_{}:\n{}".format(i+1,P))
```

```
                A = P @ A
                P_list.append(P)

            else: #En caso de no permutar agragamos la identidad
                P_list.append(np.identity(fil))

            #Obtenemos el L_i
            L_i = get_L(A, i, v)
            L_list.append(L_i)

            A = L_i @ A

            #mostrar L(i) * (A)
            if v : print("L_{} * A:\n{}\n{}".format(i+1,A,line))


        U = A
        L,P = get_L_and_P_LU(L_list, P_list)
        print("L_:\n{}\nU_:\n{}\nP_:\n{}".format(L,U,P))

        Pb = P @ b
        #Tenemos L, U, P, b

        # Ax = b  &  PA = LU
        # PAx = Pb -> LUx = Pb
        # Ly = Pb  hallamos y  ->  Ux = y  hallamos x

        y = resolverMTriangularInf(L, Pb)
        x = resolverMTriangularSup(U, y)
        return x
```

0.0.2 Definiendo las matrices A y b

```
In [3]: A = np.array([[2, 1, 1, 0],
                      [4, 3, 3, 1],
                      [8, 7, 9, 5],
                      [6, 7, 9, 8]])

        print("Matriz de coeficientes A:\n", A); (fil,col) = A.shape
        b = np.array([1, 8, 30, 41]); b.reshape(fil,1)
        print("Matriz b: \n", b)
```

```
Matriz de coeficientes A:
 [[2 1 1 0]
 [4 3 3 1]
 [8 7 9 5]
 [6 7 9 8]]
```

```
Matriz b:
 [ 1  8 30 41]
```

0.0.3 Resolviendo el sistema por LU

```
In [4]: elim_LU(A,b,v=True)
```

```
Matriz aumentada [A|b] al inicio:
[[ 2  1  1  0  1]
 [ 4  3  3  1  8]
 [ 8  7  9  5 30]
 [ 6  7  9  8 41]]
=============================================
alpha_1:
[[0.]
 [2.]
 [4.]
 [3.]]
alpha_1 * e_1:
[[0. 0. 0. 0.]
 [2. 0. 0. 0.]
 [4. 0. 0. 0.]
 [3. 0. 0. 0.]]
L_1:
[[ 1.  0.  0.  0.]
 [-2.  1.  0.  0.]
 [-4.  0.  1.  0.]
 [-3.  0.  0.  1.]]
L_1 * A:
[[2. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 3. 5. 5.]
 [0. 4. 6. 8.]]
=============================================
alpha_2:
[[0.]
 [0.]
 [3.]
 [4.]]
alpha_2 * e_2:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 3. 0. 0.]
 [0. 4. 0. 0.]]
L_2:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
```

```
 [ 0. -3.  1.  0.]
 [ 0. -4.  0.  1.]]
L_2 * A:
[[2. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 0. 2. 2.]
 [0. 0. 2. 4.]]
=============================================
alpha_3:
[[0.]
 [0.]
 [0.]
 [1.]]
alpha_3 * e_3:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 1. 0.]]
L_3:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0. -1.  1.]]
L_3 * A:
[[2. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 0. 2. 2.]
 [0. 0. 0. 2.]]
=============================================
alpha_4:
[[0.]
 [0.]
 [0.]
 [0.]]
alpha_4 * e_4:
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
L_4:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
L_4 * A:
[[2. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 0. 2. 2.]
```

```
 [0. 0. 0. 2.]]
==============================================
L_:
[[1. 0. 0. 0.]
 [2. 1. 0. 0.]
 [4. 3. 1. 0.]
 [3. 4. 1. 1.]]
U_:
[[2. 1. 1. 0.]
 [0. 1. 1. 1.]
 [0. 0. 2. 2.]
 [0. 0. 0. 2.]]
P_:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

```
Out[4]: array([-1.,  2.,  1.,  3.])
```