LU_decomposition

January 29, 2020

LU Decomposition *Author : Satrya Budi Pratama*

Define : y = **A**x

using this method we can achive the **LU** given **A**

1 Pseudocode

1. Generated **A** squared matrix using random value > 1 given the ordo(n)
2. Copy **A** to **U**
3. Create the **L** as Identity Matrix
4. Iterate over cols and iterate over cols but skip first row
    - calculate divider/factorize by divide current element with top element on same cols
    - calculate gaussian elimination by **current rows** of U - (divider) * **first rows** of U, replace the result as new rows of **U**
    - put divider on that **L** on the index rows,cols
5. Now we get **U** and **L**, check **L** dot **U** is same as **A**

Notes : sometimes this method doesn't have solution on LU Decomposition because the random of **A**, we need to check if **A** can be solved or unsolved first, and throw an error if it unsolved.

[1]:
```
import numpy as np
np.set_printoptions(precision=3) # print 3 decimal
```

[2]:
```
ordo = 5 # this mean A_5x5, we can change this
```

[3]:
```
# numpy.random.randint(min_val,max_val,(<num_rows>,<num_cols>))
A = np.random.randint(1,100,(ordo,ordo)).astype("float")
#A = np.array([[1 , -2],[3,2]])
A
```

[3]:
```
array([[ 6., 21., 42., 72., 90.],
       [66., 12., 94., 39., 46.],
       [48., 45., 33., 51., 27.],
       [28., 33., 98., 10., 11.],
       [14., 27., 26., 27., 58.]])
```

[4]:
```
rows, cols = np.shape(A) # get num of rows and col
# iterate over the columns idx 0..col-1
U = np.copy(A)
L = np.identity(rows) # create identiy matrix for L
# iterate over the rows and cols

# loop over the cols, skip last cols
for i in range(0,cols-1):
    print('\n')
    print('Step {}'.format(i+1))
    print('-----------------------')
    # loop over the rows, skip first row
    for j in range(i+1,rows):
        factorize = U[j][i]/U[i][i] # current element divide by upper element␣
↪on same cols
        print('[cols {0}] current/upper : {1:.2f}/{2:.2f} = {3:.2f}'.
↪format(i,U[j][i],U[i][i],factorize))
        print('[rows {0} : {1}]'.format(i,U[i]))
        U[j] = U[j] - factorize * U[i] # do gaussian elimination
        print('[rows {0} : {1}]'.format(j,U[j]))

        L[j][i] = factorize # set the factorize to L

    print('-----------------------')
    # show the step
    print('A{} : \n {}'.format(i,U))
    print('L{} : \n {}'.format(i,L))
```

```
Step 1
-----------------------
[cols 0] current/upper : 66.00/6.00 = 11.00
[rows 0 : [ 6. 21. 42. 72. 90.]]
[rows 1 : [   0. -219. -368. -753. -944.]]
[cols 0] current/upper : 48.00/6.00 = 8.00
[rows 0 : [ 6. 21. 42. 72. 90.]]
[rows 2 : [   0. -123. -303. -525. -693.]]
[cols 0] current/upper : 28.00/6.00 = 4.67
[rows 0 : [ 6. 21. 42. 72. 90.]]
[rows 3 : [   0.  -65.  -98. -326. -409.]]
[cols 0] current/upper : 14.00/6.00 = 2.33
[rows 0 : [ 6. 21. 42. 72. 90.]]
[rows 4 : [   0.  -22.  -72. -141. -152.]]
-----------------------
A0 :
 [[   6.   21.   42.   72.   90.]
```

```
 [   0. -219. -368. -753. -944.]
 [   0. -123. -303. -525. -693.]
 [   0.  -65.  -98. -326. -409.]
 [   0.  -22.  -72. -141. -152.]]
L0 :
 [[ 1.     0.     0.     0.     0.   ]
 [11.     1.     0.     0.     0.   ]
 [ 8.     0.     1.     0.     0.   ]
 [ 4.667  0.     0.     1.     0.   ]
 [ 2.333  0.     0.     0.     1.   ]]


Step 2
-----------------------
[cols 1] current/upper : -123.00/-219.00 = 0.56
[rows 1 : [   0. -219. -368. -753. -944.]]
[rows 2 : [   0.       0.     -96.315 -102.082 -162.808]]
[cols 1] current/upper : -65.00/-219.00 = 0.30
[rows 1 : [   0. -219. -368. -753. -944.]]
[rows 3 : [   0.       0.      11.224 -102.507 -128.817]]
[cols 1] current/upper : -22.00/-219.00 = 0.10
[rows 1 : [   0. -219. -368. -753. -944.]]
[rows 4 : [  0.      0.    -35.032 -65.356 -57.169]]
-----------------------
A1 :
 [[   6.      21.      42.      72.      90.   ]
 [   0.    -219.    -368.    -753.    -944.   ]
 [   0.       0.     -96.315 -102.082 -162.808]
 [   0.       0.      11.224 -102.507 -128.817]
 [   0.       0.     -35.032  -65.356  -57.169]]
L1 :
 [[ 1.     0.     0.     0.     0.   ]
 [11.     1.     0.     0.     0.   ]
 [ 8.     0.562  1.     0.     0.   ]
 [ 4.667  0.297  0.     1.     0.   ]
 [ 2.333  0.1    0.     0.     1.   ]]


Step 3
-----------------------
[cols 2] current/upper : 11.22/-96.32 = -0.12
[rows 2 : [   0.       0.     -96.315 -102.082 -162.808]]
[rows 3 : [   0.       0.       0.    -114.403 -147.79 ]]
[cols 2] current/upper : -35.03/-96.32 = 0.36
[rows 2 : [   0.       0.     -96.315 -102.082 -162.808]]
[rows 4 : [  0.      0.      0.    -28.227   2.048]]
-----------------------
A2 :
```

```
 [[   6.      21.      42.      72.      90.   ]
 [   0.    -219.    -368.    -753.    -944.   ]
 [   0.       0.     -96.315 -102.082 -162.808]
 [   0.       0.       0.    -114.403 -147.79 ]
 [   0.       0.       0.     -28.227    2.048]]
L2 :
 [[ 1.     0.     0.     0.     0.   ]
 [11.     1.     0.     0.     0.   ]
 [ 8.     0.562  1.     0.     0.   ]
 [ 4.667  0.297 -0.117  1.     0.   ]
 [ 2.333  0.1    0.364  0.     1.   ]]


Step 4
-----------------------
[cols 3] current/upper : -28.23/-114.40 = 0.25
[rows 3 : [   0.       0.       0.    -114.403 -147.79 ]]
[rows 4 : [ 0.     0.     0.     0.    38.512]]
-----------------------
A3 :
 [[   6.      21.      42.      72.      90.   ]
 [   0.    -219.    -368.    -753.    -944.   ]
 [   0.       0.     -96.315 -102.082 -162.808]
 [   0.       0.       0.    -114.403 -147.79 ]
 [   0.       0.       0.       0.      38.512]]
L3 :
 [[ 1.     0.     0.     0.     0.   ]
 [11.     1.     0.     0.     0.   ]
 [ 8.     0.562  1.     0.     0.   ]
 [ 4.667  0.297 -0.117  1.     0.   ]
 [ 2.333  0.1    0.364  0.247  1.   ]]
```

[5]:
```
print('{}'.format(L))
```

```
[[ 1.     0.     0.     0.     0.   ]
 [11.     1.     0.     0.     0.   ]
 [ 8.     0.562  1.     0.     0.   ]
 [ 4.667  0.297 -0.117  1.     0.   ]
 [ 2.333  0.1    0.364  0.247  1.   ]]
```

[6]:
```
print('{}'.format(U))
```

```
[[   6.      21.      42.      72.      90.   ]
 [   0.    -219.    -368.    -753.    -944.   ]
 [   0.       0.     -96.315 -102.082 -162.808]
 [   0.       0.       0.    -114.403 -147.79 ]
 [   0.       0.       0.       0.      38.512]]
```

2 Testing

We can test, with L.U :

1. **A = L U**
2. y = **A**x
3. y = **L U** x to solve x, first we search b , using this formula two back-substitution :
    - y = U x
    - b = L y
4. **y = inverse(A) x**

2.1 1. Prove A = LU

[7]:
```
# first testing A = LU
A_lu = np.round(np.dot(L,U)) # ceil up the element of matrix;
A_lu
```

[7]:
```
array([[ 6., 21., 42., 72., 90.],
       [66., 12., 94., 39., 46.],
       [48., 45., 33., 51., 27.],
       [28., 33., 98., 10., 11.],
       [14., 27., 26., 27., 58.]])
```

[8]:
```
A
```

[8]:
```
array([[ 6., 21., 42., 72., 90.],
       [66., 12., 94., 39., 46.],
       [48., 45., 33., 51., 27.],
       [28., 33., 98., 10., 11.],
       [14., 27., 26., 27., 58.]])
```

Compare **A** result using LU Decomposition with the **A** original one

[9]:
```
A_lu == A # all true means proved A = LU
```

[9]:
```
array([[ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True]])
```

2.2 2. Prove y = Ax

[10]:
```
# generate random y
y = np.random.randint(1,100,(1,ordo)).astype("float") # generate random y
y = y[0]
y
```

[10]:
```
array([43., 23., 97., 69.,  5.])
```

[11]:
```
# two back-substitution
def calculateX(U, Y):
    X = np.zeros(len(Y))
    for i in reversed(range(len(U))):
        X[i] = (Y[i] - sum(U[i][i+1:] * X[i+1:])) / U[i][i]
    return X

def calculateLY(L, B):
    Y = np.zeros(len(B))
    for i in range(len(L)):
        Y[i] = B[i] - sum(L[i] * Y)
    return Y
```

[12]:
```
b = calculateLY(L, y) # calculate b using Ly
b
```

[12]:
```
array([  43.   , -450.   ,    5.74 ,    2.564,  -52.848])
```

We can ge x using two-back substitution

[13]:
```
x = calculateX(U,b) # calculate X using Ub
x
```

[13]:
```
array([-0.537,  1.271,  0.405,  1.75 , -1.372])
```

[14]:
```
# prove y = Ax
y_rest = A.dot(x)
y_rest
```

[14]:
```
array([43., 23., 97., 69.,  5.])
```

[15]:
```
y
```

[15]:
```
array([43., 23., 97., 69.,  5.])
```

Compare y result using two back subtitution with the original one

[16]:
```
np.round(y_rest) == np.round(y) # all true means proved y = Ax
```

[16]: array([True, True, True, True, True])

2.3 3. Prove y = LUx

[17]:
```
y_rest_LU = L.dot(U).dot(x)
```

[18]:
```
y_rest_LU
```

[18]: array([43., 23., 97., 69., 5.])

[19]:
```
y
```

[19]: array([43., 23., 97., 69., 5.])

Compare y result with the original one

[20]:
```
np.round(y_rest_LU) == np.round(y) # all true means proved y = LUx
```

[20]: array([True, True, True, True, True])

2.4 4. Prove x = inverse(A) * y

[21]:
```
# using inverse
A_inv = np.linalg.inv(A)
A_inv
```

[21]:
```
array([[-0.015,  0.015,  0.006, -0.013,  0.01 ],
       [-0.008, -0.016,  0.012,  0.01 ,  0.017],
       [ 0.005,  0.001, -0.006,  0.011, -0.008],
       [ 0.02 , -0.003,  0.012, -0.   , -0.034],
       [-0.004,  0.005, -0.01 , -0.006,  0.026]])
```

get x using inverse dot product y

[22]:
```
# prove x = A-1 * y
x_res = A_inv.dot(y)
```

[23]:
```
x_res
```

[23]: array([-0.537, 1.271, 0.405, 1.75 , -1.372])

[24]:
```
x
```

[24]: array([-0.537, 1.271, 0.405, 1.75 , -1.372])

Compare x_result using inverse with x original from two-back subtitution

[25]:
```
np.round(x_res) == np.round(x) # all true means proved x = A-1 * y
```

[25]:
```
array([ True,  True,  True,  True,  True])
```