Ting Fung Lam 2924629375

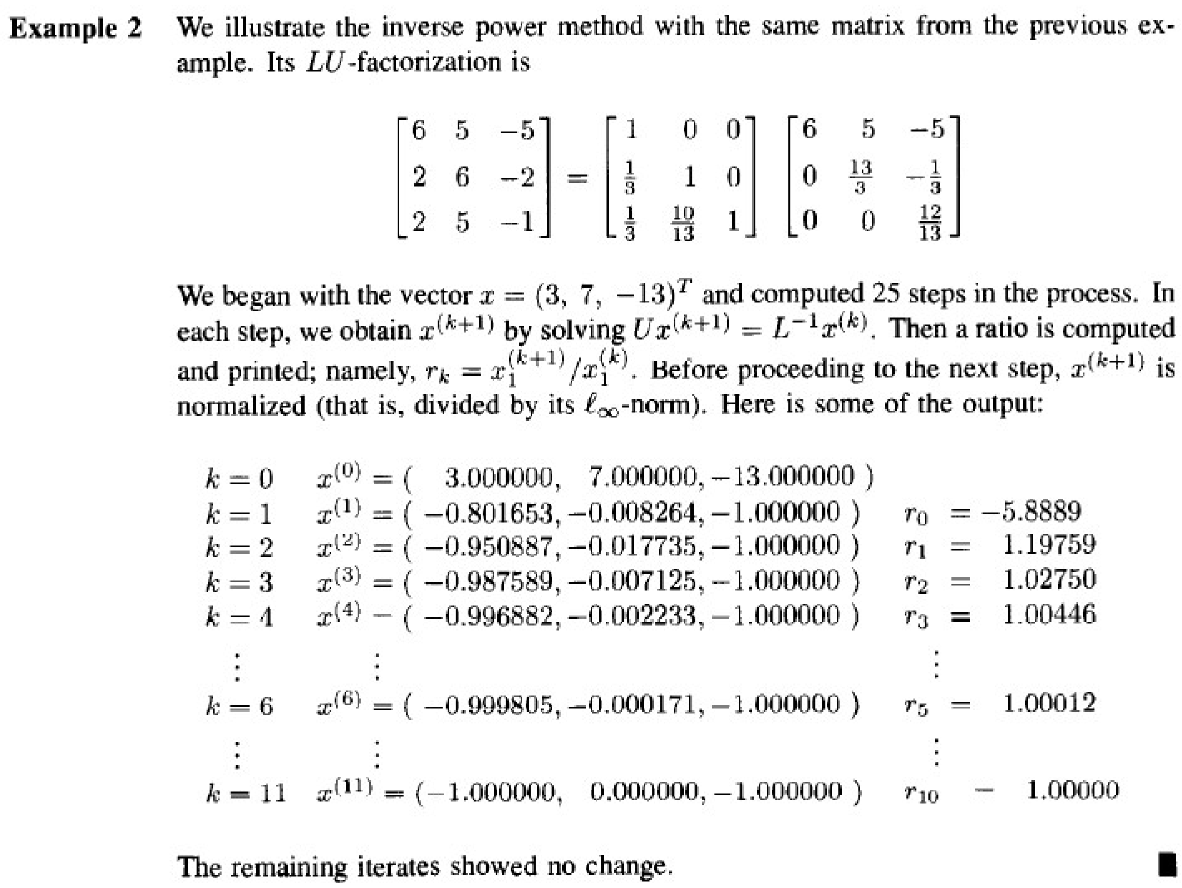
Feb 29 2020

Programming HW#7 (Due: March 6th, 11:59 PM):
Implement the algorithm of shifted inverse power method (using your own LU decomposition codes which you did before). And then run your own code to recover all the computational procedures in Example 2 on Page 233.

The results of the program matches with the example

[1]:
```python
from IPython.display import display, Image
i = Image(filename='image.jpg')
i
```

[1]:



Example 2 We illustrate the inverse power method with the same matrix from the previous example. Its LU-factorization is

$$\begin{bmatrix} 6 & 5 & -5 \\ 2 & 6 & -2 \\ 2 & 5 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 1 & 0 \\ \frac{1}{3} & \frac{10}{13} & 1 \end{bmatrix} \begin{bmatrix} 6 & 5 & -5 \\ 0 & \frac{13}{3} & -\frac{1}{3} \\ 0 & 0 & \frac{12}{13} \end{bmatrix}$$

We began with the vector $x = (3,\ 7,\ -13)^T$ and computed 25 steps in the process. In each step, we obtain $x^{(k+1)}$ by solving $Ux^{(k+1)} = L^{-1}x^{(k)}$. Then a ratio is computed and printed; namely, $r_k = x_1^{(k+1)}/x_1^{(k)}$. Before proceeding to the next step, $x^{(k+1)}$ is normalized (that is, divided by its ℓ_∞-norm). Here is some of the output:

$$\begin{array}{lllll}
k=0 & x^{(0)} = (\ \ 3.000000,\ \ 7.000000, -13.000000\ ) & & & \\
k=1 & x^{(1)} = (\ -0.801653, -0.008264, -1.000000\ ) & r_0 & = & -5.8889 \\
k=2 & x^{(2)} = (\ -0.950887, -0.017735, -1.000000\ ) & r_1 & = & 1.19759 \\
k=3 & x^{(3)} = (\ -0.987589, -0.007125, -1.000000\ ) & r_2 & = & 1.02750 \\
k=4 & x^{(4)} = (\ -0.996882, -0.002233, -1.000000\ ) & r_3 & = & 1.00446 \\
\vdots & \vdots & \vdots & & \\
k=6 & x^{(6)} = (\ -0.999805, -0.000171, -1.000000\ ) & r_5 & = & 1.00012 \\
\vdots & \vdots & \vdots & & \\
k=11 & x^{(11)} = (-1.000000,\ \ 0.000000, -1.000000\ ) & r_{10} & = & 1.00000
\end{array}$$

The remaining iterates showed no change. ■

[2]:
```python
import numpy as np


def luDecomposition(A):
    """
    LU decomposition
    :param A: Input matrix
    :return: Matrices L and U
    """
    n = np.size(A, 0)
    L = np.zeros([n, n])
    U = np.zeros([n, n])
    for k in range(0, n):
        L[k, k] = 1
        U[k, k] = (A[k, k] - np.dot(L[k, 0:k], U[0:k, k])) / L[k, k]
        for j in range(k, n):
            U[k, j] = (A[k, j] - np.dot(L[k, 0:k], U[0:k, j])) / L[k, k]
        for i in range(k, n):
            L[i, k] = (A[i, k] - np.dot(L[i, 0:k], U[0:k, k])) / U[k, k]
    return L, U


def inversePower(L, U, x, M):
    for k in range(0, M):
        y = np.dot(np.dot(np.linalg.inv(U), np.linalg.inv(L)), x)
        r = y[0]/x[0]
        x = y / np.max(abs(y))
        print("k = " + str(k + 1) + " x = " + str(x) + " r = " + str(r))


if __name__ == '__main__':
    A = np.array([[6, 5, -5], [2, 6, -2], [2, 5, -1]])
    x = np.array([3, 7, -13])
    L, U = luDecomposition(A)
    M = 25
    inversePower(L, U, x, M)
```

```
k = 1 x = [-0.80165289 -0.00826446 -1.        ] r = -5.388888888888889
k = 2 x = [-0.95088677 -0.01773533 -1.        ] r = 1.1975945017182132
k = 3 x = [-0.98758906 -0.0071248  -1.        ] r = 1.027498804399809
k = 4 x = [-0.99688198 -0.00223266 -1.        ] r = 1.0044604763012954
k = 5 x = [-9.99219244e-01 -6.32958164e-04 -1.00000000e+00] r = 1.0007401114672227
k = 6 x = [-9.99804721e-01 -1.70634888e-04 -1.00000000e+00] r = 1.0001232606845578
k = 7 x = [-9.99951174e-01 -4.47180434e-05 -1.00000000e+00] r = 1.000020540915539
k = 8 x = [-9.99987793e-01 -1.15222182e-05 -1.00000000e+00] r = 1.000003423415603
k = 9 x = [-9.99996948e-01 -2.93763580e-06 -1.00000000e+00] r = 1.0000005705673138
k = 10 x = [-9.99999237e-01 -7.43920002e-07 -1.00000000e+00] r = 1.000000095094498
k = 11 x = [-9.99999809e-01 -1.87565012e-07 -1.00000000e+00] r = 1.0000000158490814
k = 12 x = [-9.99999952e-01 -4.71554110e-08 -1.00000000e+00] r = 1.0000000026415137
k = 13 x = [-9.99999988e-01 -1.18328784e-08 -1.00000000e+00] r = 1.0000000004402523
k = 14 x = [-9.99999997e-01 -2.96555723e-09 -1.00000000e+00] r = 1.0000000000733755
k = 15 x = [-9.99999999e-01 -7.42612279e-10 -1.00000000e+00] r = 1.0000000000122293
k = 16 x = [-1.00000000e+00 -1.85856941e-10 -1.00000000e+00] r = 1.0000000000020384
k = 17 x = [-1.00000000e+00 -4.64982497e-11 -1.00000000e+00] r = 1.0000000000003397
k = 18 x = [-1.00000000e+00 -1.16302662e-11 -1.00000000e+00] r = 1.0000000000000566
k = 19 x = [-1.00000000e+00 -2.90856228e-12 -1.00000000e+00] r = 1.0000000000000095
k = 20 x = [-1.00000000e+00 -7.27348737e-13 -1.00000000e+00] r = 1.0000000000000016
k = 21 x = [-1.00000000e+00 -1.81910043e-13 -1.00000000e+00] r = 1.0000000000000004
k = 22 x = [-1.00000000e+00 -4.55468996e-14 -1.00000000e+00] r = 1.0000000000000002
k = 23 x = [-1.00000000e+00 -1.14352972e-14 -1.00000000e+00] r = 1.0000000000000002
k = 24 x = [-1.00000000e+00 -2.90045765e-15 -1.00000000e+00] r = 1.0
k = 25 x = [-1.00000000e+00 -7.77156117e-16 -1.00000000e+00] r = 1.0000000000000002
```