HOMEWORK 2

LIAM DUNGAN

Problem 1.

To verify the two matrices are inverses of each other, we can multiply and show the product is $I_{n\times n}$ identity matrix.

i.e.

$(I + m_k e_k^T)(I - m_k e_k^T) = I$

$(I + m_k e_k^T)(I - m_k e_k^T)$
$= I^2 + Ime_k^T - Ime_k^T + (me_k^T)^2 \quad \#by\, distribution$
$= I^2 + (me_k^T)^2 \qquad \#by\, cancelling\, like\, terms$
$= I^2 \qquad \#note\, that\, (me_k^T)^2 = (me_k^T)(me_k^T) = 0$
$= I \qquad \#note\, the\, identity\, matrix\, multiplied\, by\, itself\, equals\, itself\;\, so\, I^2 = I$

$L_k M_K = M_k L_k = I$

Thus, L_k is the inverse of matrix M_k

Problem 2.

```
x =  [[ 0.09  0.38 -1.29  0.06  0.05  0.04  0.29 -0.49  0.58]]

 L =
[[ 0.05  0.    0.    0.    0.    0.   -0.    0.    0.  ]
 [ 0.03 -0.04  0.    0.    0.    0.   -0.    0.    0.  ]
 [ 0.07  0.19 -0.11 -0.   -0.   -0.    0.   -0.    0.  ]
 [-0.03  0.01 -0.04  0.16 -0.   -0.    0.   -0.    0.  ]
 [ 0.02  0.01 -0.02 -0.07  0.07  0.   -0.    0.    0.  ]
 [-0.01 -0.01  0.   -0.07  0.07  0.06 -0.    0.    0.  ]
 [-0.02  0.01  0.04 -0.03 -0.05  0.08 -0.05 -0.    0.  ]
 [-0.02  0.01  0.01  0.06 -0.04 -0.   -0.01 -0.01  0.  ]
 [-0.17  0.16  0.27  0.78 -1.56  1.41 -2.09 -0.37  1.  ]]

 U =
[[ 1.   -1.52 -0.45  0.32  0.66 -1.19  0.8  -0.89 -0.04]
 [ 0.    1.   -0.14  0.68  3.09 -1.74  2.54  1.69 -0.13]
 [-0.    0.    1.   -2.61 -8.2   6.08 -8.14 -1.24  0.26]
 [-0.    0.    0.    1.   -0.48 -1.31  1.46 -4.33  0.23]
 [ 0.   -0.   -0.    0.    1.   -0.54  0.61  1.28 -0.1 ]
 [-0.    0.    0.   -0.   -0.    1.   -1.16  0.03  0.1 ]
 [-0.    0.    0.   -0.   -0.    0.    1.    0.53  0.04]
 [-0.    0.    0.   -0.   -0.    0.   -0.    1.    0.13]
 [ 0.   -0.   -0.    0.    0.   -0.    0.   -0.   -0.32]]
```

Date: Oct. 17, 2017.

```python
np.set_printoptions(precision=2)
np.set_printoptions(suppress=True)

def LU(A,b):
    s = A.shape[0]
    t = A.shape[1]
    if s != t:
        print "Invalid: Matrix is not square"
        return

    p = A.shape[0]
    M = np.ndarray( shape = (p,p,p) )
    L = np.ndarray( shape = (p,p,p) )

    Lfinal = np.identity(p)
    Ufinal = np.identity(p)

    for i in range(0,p-1):
        M[i] = np.identity(p)
        M[i][i,i]=np.float(1)/A[i,i]

        for j in range(i+1,p):
            M[i][j,i]=(np.float(-1)*A[j,i])/A[i,i]

        A=M[i].dot(A)
        L[i] = np.linalg.inv(M[i])
        Lfinal = Lfinal.dot(L[i])

    Lfinal = np.linalg.inv(Lfinal)
    Ufinal = np.linalg.inv(A)

    print "\nx = ", Ufinal.dot(Lfinal.dot(b))
    print "\n L = \n", Lfinal
    print "\n U = \n", Ufinal

def main():
    A = np.matrix([[21. ,32. ,14. ,8. ,6. ,9. ,11. ,3. ,5 ],
        [17. ,2. ,8. ,14. ,55. ,23. ,19. ,1. ,6 ],
        [41. ,23. ,13. ,5. ,11. ,22. ,26. ,7. ,9 ],
        [12. ,11. ,5. ,8. ,3. ,15. ,7. ,25. ,19 ],
        [14. ,7. ,3. ,5. ,11. ,23. ,8. ,7. ,9 ],
        [2. ,8. ,5. ,7. ,1. ,13. ,23. ,11. ,17 ],
        [11. ,7. ,9. ,5. ,3. ,8. ,26. ,13. ,17 ],
        [23. ,1. ,5. ,19. ,11. ,7. ,9. ,4. ,16 ],
        [31. ,5. ,12. ,7. ,13. ,17. ,24. ,3. ,11]])
    b = np.array([2. ,5. ,7. ,1. ,6. ,9. ,4. ,8. ,3.])
    LU(A,b)

if __name__== "__main__":
    main()
```

Problem 3.

From $c\|x\|_a \le \|x\|_b \le d\|x\|_a$ we can say $c'\|x\|_a \le \|x\|_b \le d'\|x\|_a$ and therfore,
$\frac{\|Ax\|_a}{\|x\|_a} \le \frac{d'\|Ax\|_b}{c'\|x\|_b}$
From these statements,
$\frac{c'\|Ax\|_b}{d'\|x\|_b} \le \frac{\|Ax\|_a}{\|x\|_a} \le \frac{d'\|Ax\|_b}{c'\|x\|_b}$
Moving forword we use the definition of $\|A\|$,
$\|A\|_a = max_{x\neq 0}\frac{\|Ax\|_a}{\|x\|_a} \le max_{x\neq 0}\frac{d'\|Ax\|_b}{c'\|x\|_b}$
$\quad and$
$\|A\|_a = max_{x\neq 0}\frac{\|Ax\|_a}{\|x\|_a} \ge max_{x\neq 0}\frac{c'\|Ax\|_b}{d'\|x\|_b}$
Thus,
$c' * max_{x\neq 0}\frac{\|Ax\|_a}{\|x\|_a} \le max_{x\neq 0}\frac{\|Ax\|_b}{\|x\|_b} \le d' * max_{x\neq 0}\frac{\|Ax\|_a}{\|x\|_a}$
$\Longrightarrow c'\|A\|_a \le \|A\|_b \le d'\|A\|_a$

Therefore, the induced matrix norms are, in fact, also equavalent.

Problem 4.

```
[nbp-77-43:desktop liamdungan$ python 4.py
x =  [-0.9999179  2.000371  -2.9986966  3.9980645]


r =  [  7.5101852e-06  -2.8371811e-05  -9.5367432e-06   2.1338462e-06]
Improved:  [-1.         1.9999999 -3.0000002  4.0000004]


r =  [ -4.7875801e-09  -4.7439244e-09  -1.9645086e-08  -7.4142008e-09]
Improved:  [-1.  2. -3.  4.]


r =  [  5.6843419e-13   5.1727511e-12  -9.0949470e-13   1.2647661e-12]
Improved:  [-1.  2. -3.  4.]
```

After 3 repeats, no more improvement is observable.
full code in 4.py

```
def solution(A, b, x):
    n=A.shape[0]
    bCopy = deepcopy(b)
    gaussian(np.matrix.copy(A),bCopy,x)

    r = np.zeros(n)
    thisX = np.zeros(n)
    newX = np.zeros(n)
    for i in range(0,n):
        thisX[i] = x[i]
    print "x = ", x, "\n\n"

    for temp in range(0,3): #no change observed after 3
        aCopy = np.matrix.copy(A)
        Ax = aCopy.dot(thisX)
        for i in range(0, n):
            r[i] = np.float32(b[i] - Ax[0,i])

        print "r = ", r
        z = np.zeros(n)
        gaussian(aCopy, r, z)

        for add in range(0, n):
            newX[add] = thisX[add] + z[add]
        print"Improved: ",newX

        print "\n\n"
        for i in range(0, n):
            thisX[i] = newX[i]


def main():
    A = np.matrix( [[21.0, 67.0, 88.0, 73.0],
                    [76.0, 63.0, 7.0, 20.0],
                    [0.0, 85.0, 56.0, 54.0],
                    [19.3, 43.0, 30.2, 29.4]])

    b = np.array([141.0, 109.0, 218.0, 93.7])
    x = np.zeros(4)
    solution(A, b, x)


if __name__== "__main__":
  main()
```

Problem 5.

As k increases from 1-10, the value of ε decreases and the computed solution becomes less accurate.

```
k: 1 ,  x =
[[ 1.]
 [ 1.]]

k: 2 ,  x =
[[ 1.]
 [ 1.]]

k: 3 ,  x =
[[ 1.]
 [ 1.]]

k: 4 ,  x =
[[ 0.99999999]
 [ 1.        ]]

k: 5 ,  x =
[[ 1.00000008]
 [ 1.        ]]

k: 6 ,  x =
[[ 0.99986686]
 [ 1.        ]]

k: 7 ,  x =
[[ 0.99920072]
 [ 1.        ]]

k: 8 ,  x =
[[ 2.22044605]
 [ 1.        ]]

k: 9 ,  x =
[[ 0.]
 [ 1.]]

k: 10 , x =
[[ 0.]
 [ 1.]]
```

```
import numpy as np
import math

#Problem 5
def forward(A, b, n):
    for r in range(0, n-1):
        for i in range(r+1, n):
            temp = A[i,r] / A[r,r]
            for j in range(r, n):
                A[i,j] = A[i,j] - temp * A[r,j]

            b[i] = b[i] - temp * b[r]
    return A, b


def back(a, b, n):
    x = np.zeros((n,1))
    x[n-1] = b[n-1] / a[n-1, n-1]
    for r in range(n-2, -1, -1):
        sums = b[r]
        for j in range(r+1, n):
            sums = sums - a[r,j] * x[j]

        x[r] = sums / a[r,r]
    return x


def gaussian(A, b):
    n = A.shape[0]
    if any(np.diag(A)==0):
        print "Error: diagnol elements zero"
        return
    A, b = forward(A, b, n)
    return back(A, b, n)



def main():
    for k in range(1, 11):
        epsilon = math.pow(10, -2 * k)
        A = np.array([[epsilon,1,],[1,1,],])
        b = np.array([1+epsilon,2,])
        x = gaussian(A, b)
        print "k:",k,",\tx = \n", x
        print ""

if __name__ == '__main__':
    main()
```