CS 111 (S19): Homework 3

Due by 6:00pm Monday, April 22

NAME and PERM ID No.: Chen Li, 5468137 (replace with yours)

UCSB EMAIL: chenli@ucsb.edu (replace with yours)

1. How many arithmetic operations (total of additions, subtractions, multiplications, divisions) are required to do each of the following? (You can omit lower-order terms in n.)

1a. Compute the sum of two n-vectors?

$$n$$

1b. Compute the product of an n-by-n matrix with an n-vector?

$$n * (n + (n-1)) = 2n^2 - n$$

1c. Compute the product of two n-by-n matrices?

$$n^2(n + (n-1)) = 2n^3 - n^2$$

1d. Solve an n-by-n upper triangular linear system $Ux = y$?

$$2\sum_{i=1}^{n-1} i + n = (n-1)n + n = n^2$$

2. Suppose A and B are n-by-n matrices, with A nonsingular, and c is an n-vector. Describe the steps you would use to *efficiently* compute the product $A^{-1}Bc$. Describe a different, less efficient, sequence of steps.

Efficient: realizing that $AA^{-1}Bc = Bc$, then $A^{-1}Bc$ is the solution to $Ax = Bc$, so we can do gaussian elimination to solve x, which is the answer. In this way, we can avoid computing inverse, which take roughly 3 time more steps than gaussian elimination when matrix is large.

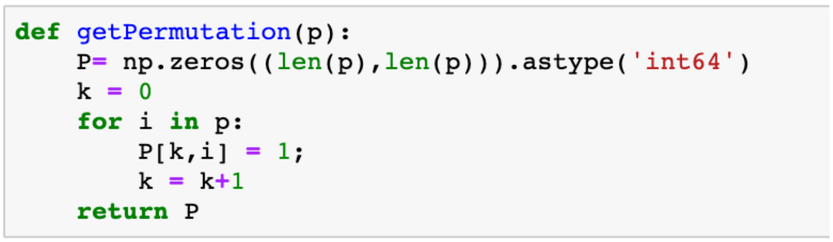
Inefficient: inverse A, then calculate $A^{-1}Bc$, which take roughly 3 time more steps than previous one.

3. Suppose that A is a square, nonsingular, nonsymmetric matrix, b is an n-vector, and that you have called

`L, U, p = LUfactor(A)`

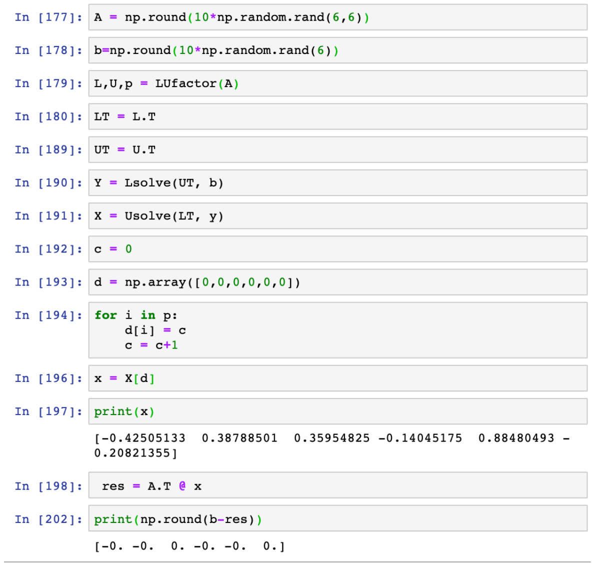
(using the routine from the lecture files). Now suppose you want to solve the system $A^Tx = b$ (not $Ax = b$) for x. Show how to do this using calls to `Lsolve()` and `Usolve()`, without modifying either of those routines or calling `LUfactor()` again. Test your method in `numpy` on a randomly generated 6-by-6 matrix (see `np.random.rand()`).

Get Permutation matrix P from p
PA = LU
$A = P^TLU$
$A^T = U^TL^TP$
$A^Tx = U^TL^TPx = b$
let Px = y
$A^Tx = U^TL^Ty = b$
therefore, U^T is a lower triangle matrix and L^T is a upper triangle matrix
then, $y = Usolve(L.T, Lsolve(U.T, b[p])$
and $x = P.Ty$



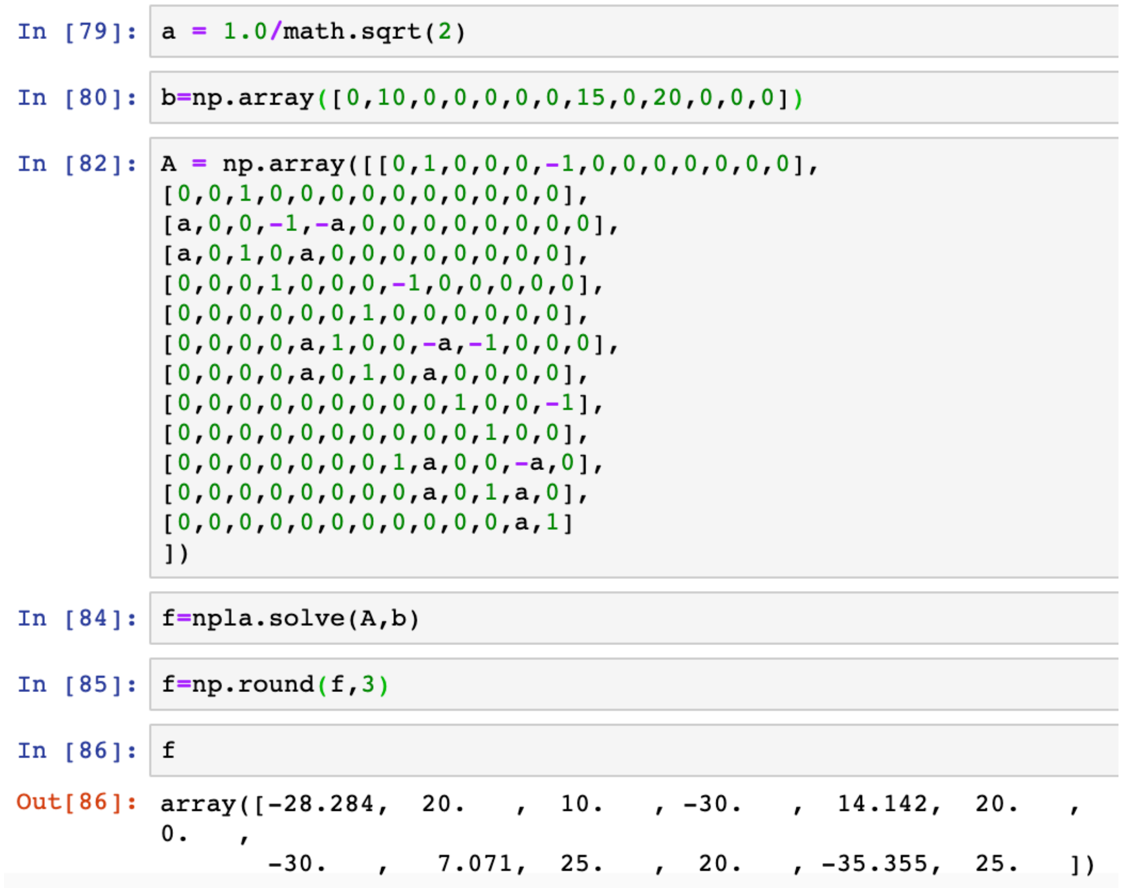
```
def getPermutation(p):
    P= np.zeros((len(p),len(p))).astype('int64')
    k = 0
    for i in p:
        P[k,i] = 1;
        k = k+1
    return P
```

here, lower p is permuting array, and upper P is permutation matrix that get returned



```
In [177]: A = np.round(10*np.random.rand(6,6))

In [178]: b=np.round(10*np.random.rand(6))

In [179]: L,U,p = LUfactor(A)

In [180]: LT = L.T

In [189]: UT = U.T

In [190]: Y = Lsolve(UT, b)

In [191]: X = Usolve(LT, y)

In [192]: c = 0

In [193]: d = np.array([0,0,0,0,0,0])

In [194]: for i in p:
              d[i] = c
              c = c+1

In [196]: x = X[d]

In [197]: print(x)

          [-0.42505133  0.38788501  0.35954825 -0.14045175  0.88480493 -
          0.20821355]

In [198]:  res = A.T @ x

In [202]: print(np.round(b-res))

          [-0. -0.  0. -0. -0.  0.]
```

4. Do problem 2.3 on pages 32–33 of the NCM book, showing the `numpy` code you use and its output. Note: To understand intuitively what the problem means by "assume that joint 1 is rigidly fixed both horizontally and vertically and that joint 8 is fixed vertically," think of the truss as a (2-dimensional) drawbridge across a river, with the left end being a hinge and the right end lying on the ground.



```
In [79]: a = 1.0/math.sqrt(2)

In [80]: b=np.array([0,10,0,0,0,0,0,15,0,20,0,0,0])

In [82]: A = np.array([[0,1,0,0,0,-1,0,0,0,0,0,0,0],
         [0,0,1,0,0,0,0,0,0,0,0,0,0],
         [a,0,0,-1,-a,0,0,0,0,0,0,0,0],
         [a,0,1,0,a,0,0,0,0,0,0,0,0],
         [0,0,0,1,0,0,0,-1,0,0,0,0,0],
         [0,0,0,0,0,0,1,0,0,0,0,0,0],
         [0,0,0,0,a,1,0,0,-a,-1,0,0,0],
         [0,0,0,0,a,0,1,0,a,0,0,0,0],
         [0,0,0,0,0,0,0,0,0,1,0,0,-1],
         [0,0,0,0,0,0,0,0,0,0,1,0,0],
         [0,0,0,0,0,0,0,1,a,0,0,-a,0],
         [0,0,0,0,0,0,0,0,a,0,1,a,0],
         [0,0,0,0,0,0,0,0,0,0,0,a,1]
         ])

In [84]: f=npla.solve(A,b)

In [85]: f=np.round(f,3)

In [86]: f

Out[86]: array([-28.284,  20.   ,  10.   , -30.   ,  14.142,  20.   ,
         0.   ,
                -30.   ,   7.071,  25.   ,  20.   , -35.355,  25.   ])
```

5. Consider the linear system

$$\begin{pmatrix} \alpha & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 3\alpha + 2 \\ 3 \end{pmatrix},$$

for some $\alpha < 1$. Clearly the solution is $(x_0, x_1)^T = (1, 2)^T$. For each value of $\alpha = 10^{-4}, 10^{-8}, 10^{-16}, 10^{-20}$, solve this system using the routines `LUfactor()`, `Lsolve()`, and `Usolve()` from `LUsolve.ipynb` in the lecture files. For each α, do this twice, first with `pivoting = True` in `LUfactor()` and then with `pivoting = False`. Show your `numpy` code and its output. Comment on your results.

```
In [69]: def p5(power):
             a = 10 ** power
             A = np.array([[a,1],[1,1]])
             b=np.array([a+2,3])
             LU1 = LUfactor(A, True)
             LU2 = LUfactor(A, False)
             y1 = Lsolve(LU1[0], b[LU1[2]])
             y2 = Lsolve(LU2[0], b[LU2[2]])
             x1 = Usolve(LU1[1], y1)
             x2 = Usolve(LU2[1], y2)
             print("when alpha = 10 to the", power, ":\n","with pivoting
```

```
In [70]: for power in [-4,-8,-16,-20]:
             p5(power)
```

```
when alpha = 10 to the -4 :
 with pivoting x =  [1. 2.]
 without pivoting x= [1. 2.]

when alpha = 10 to the -8 :
 with pivoting x =  [1. 2.]
 without pivoting x= [0.99999999 2.        ]

when alpha = 10 to the -16 :
 with pivoting x =  [1. 2.]
 without pivoting x= [4.4408921 2.        ]

when alpha = 10 to the -20 :
 with pivoting x =  [1. 2.]
 without pivoting x= [0. 2.]
```

```
In [ ]: #when pivoting is true, the result is more accurate no matter
        #how alpha change since it enhance numerical stability.
        #However, when pivoting is false, the answer start to fluctuate
        #as alpha get smaller because numerical stability is no ensured
```

by selete the pivot with maxium value in the column, we can enhanve stability

6. Recall that a symmetric matrix A is *positive definite* (SPD for short) if and only if $x^T Ax > 0$ for every nonzero vector x.

6a. Find a 2-by-2 matrix A that (1) is symmetric, (2) is not singular, and (3) has all its elements greater than zero, but (4) is *not* SPD. Show a nonzero vector x such that $x^T Ax < 0$.

$$A = \left(\begin{array}{cc} 1 & 2 \\ 2 & 1 \end{array} \right), x = (1, -1), \Rightarrow x^T Ax = -2$$

6b. Let B be a nonsingular matrix, of any size, not necessarily symmetric. Prove that the matrix $A = B^T B$ is SPD.

symmetric: $A^T = (B^T B)^T = B^T (B^T)^T = B^T B = A$
positive definite: $x^T Ax = x^T B^T Bx = (Bx)^T (Bx) = y^T y$, which is a inner product of a non-zero vector y (since B is nonsingular and x is non-zero), alway larger than 0.