HW1 Resubmission

Lifan Yu lifany

September 2023

Question 1

Program:

```
### Question 1 & Question2 LDU
table0 = [[0, 7, 8, 0], [11, 14, 14, 11], [2, 0, 3, 0], [0, 0, 5, 1
                                          ], [0, 0, 0, 6]]
table1 = [[7, -1, 1], [1, 4, 3], [8, 1, 10]]
table2 = [[11, 14, 14, 11], [0, 7, 8, 0], [2, 0, 3, 0], [0, 0, 0, 6
                                          ], [0, 0, 5, 1]]
table3 = [[8, 3, 7], [3, 2, 0], [9, 5, 3]]

def LDU(table):
  #table = np.array(table)
  rownum = len(table)
  columnnum = len(table[0])
  L = [[0 for i in range(rownum)] for j in range(len(table))]
  for k in range(rownum):
    L[k][k] = 1

  P = []
  for this in range(rownum):
    to_append = [0 for t in range(rownum)]
    to_append[this] = 1
    P.append(to_append)


  # check for row exchanges
  # go column by column to check for zeros
  for c in range(columnnum):
    exchange = 0
    empty = 1
    for r in range(rownum):
      if table[r][c] != 0:
        #print(table[r][c])
        empty = 0
    # if column not all zero, there might be a need to swap rows
    if empty == 0:
      d = 0
      while d < rownum and table[c][d] == 0:
        d += 1
```

```
    # d: row number of the top non zero element in the column
    if d > c:
      # swap rows if there are nothing to cancel out this top non
                                              -zero element
      temp = []
      for ii in range(columnnum):
        temp.append(table[c][ii])
      table[c] = table[d]
      table[d] = temp

      temp2 = []
      for ii in range(columnnum):
        temp2.append(P[c][ii])
      P[c] = table[d]
      P[d] = temp2

      for l in range(c):  # swap correposnding rows in l but only
                                               below diagonal
        temp3 = L[c][l]
        L[c][l] = L[d][l]
        L[d][l] = temp3

print("table", table)
print("P", P)

for i in range(rownum):
  row = table[i]
  l_list = []
  for j in range(i):
    if row[j] != 0:
      m = row[j]/table[j][j]
      l_list.append(m)
      for k in range(columnnum):
        table[i][k] = table[i][k] - m*(table[j][k])
    else:
      l_list.append(0)
  for k in range(len(l_list)):
    L[i][k] = l_list[k]

print("L")
for i in range(len(L)):
  print(L[i])

D = [[0 for i in range(len(L))]for j in range(len(L))]

for i in range(min(rownum, columnnum)):
  if table[i][i] != 0:
    D[i][i] = table[i][i]
    for j in range(len(table[i])):
      table[i][j] = table[i][j]/D[i][i]

print("D")
for i in range(len(D)):
  print(D[i])

print("U")
for i in range(rownum):
```

```
    print(table[i])

  print("-"*40)


print("Question 1 testing")
LDU(table0)
print("a")
LDU(table1)
print("b")
LDU(table2)
print("c")
LDU(table3)
```

Test code:

```

# Testing the code
print("Question 1 testing")
LDU(table0)
print("a")
LDU(table1)
print("b")
LDU(table2)
print("c")
LDU(table3)
```

Output:

```
Question 1 testing
table [[11, 14, 14, 11], [0, 7, 8, 0], [2, 0, 3, 0], [0, 0, 5, 1],
                                  [0, 0, 0, 6]]
L
[1, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0.18181818181818182, -0.36363636363636365, 1, 0, 0]
[0, 0, 1.4864864864864864, 1, 0]
[0, 0, 0, 1.5102040816326532, 1]
D
[11, 0, 0, 0, 0]
[0, 7, 0, 0, 0]
[0, 0, 3.3636363636363638, 0, 0]
[0, 0, 0, 3.972972972972973, 0]
[0, 0, 0, 0, 0]
U
[1.0, 1.2727272727272727, 1.2727272727272727, 1.0]
[0.0, 1.0, 1.1428571428571428, 0.0]
[0.0, 0.0, 1.0, -0.5945945945945946]
[0.0, 0.0, 0.0, 1.0]
[0.0, 0.0, 0.0, 0.0]
----------------------------------------
a
table [[7, -1, 1], [1, 4, 3], [8, 1, 10]]
L
[1, 0, 0]
[0.14285714285714285, 1, 0]
[1.1428571428571428, 0.5172413793103448, 1]
D
```

```
[7, 0, 0]
[0, 4.142857142857143, 0]
[0, 0, 7.379310344827587]
U
[1.0, -0.14285714285714285, 0.14285714285714285]
[0.0, 1.0, 0.689655172413793]
[0.0, 0.0, 1.0]
----------------------------------------
b
table [[11, 14, 14, 11], [0, 7, 8, 0], [2, 0, 3, 0], [0, 0, 0, 6],
                                  [0, 0, 5, 1]]
L
[1, 0, 0, 0, 0]
[0, 1, 0, 0, 0]
[0.18181818181818182, -0.36363636363636365, 1, 0, 0]
[0, 0, 0, 1, 0]
[0, 0, 1.4864864864864864, 0.6621621621621622, 1]
D
[11, 0, 0, 0, 0]
[0, 7, 0, 0, 0]
[0, 0, 3.3636363636363638, 0, 0]
[0, 0, 0, 6, 0]
[0, 0, 0, 0, 0]
U
[1.0, 1.2727272727272727, 1.2727272727272727, 1.0]
[0.0, 1.0, 1.1428571428571428, 0.0]
[0.0, 0.0, 1.0, -0.5945945945945946]
[0.0, 0.0, 0.0, 1.0]
[0.0, 0.0, 0.0, 0.0]
----------------------------------------
c
table [[8, 3, 7], [3, 2, 0], [9, 5, 3]]
L
[1, 0, 0]
[0.375, 1, 0]
[1.125, 1.8571428571428572, 1]
D
[8, 0, 0]
[0, 0.875, 0]
[0, 0, 0]
U
[1.0, 0.375, 0.875]
[0.0, 1.0, -3.0]
[0.0, 0.0, 0.0]
----------------------------------------
```

2

Program:

```
### Question2 SVD
table1 = [[7, -1, 1], [1, 4, 3], [8, 1, 10]]
table2 = [[11, 14, 14, 11], [0, 7, 8, 0], [2, 0, 3, 0], [0, 0, 0, 6
                                    ], [0, 0, 5, 1]]
table3 = [[8, 3, 7], [3, 2, 0], [9, 5, 3]]
##################################################
##################################################
def SVD(table):
  U, Sigma, VT = np.linalg.svd(table)
  print("U", U, "Sigma", np.diag(Sigma), "V", VT.transpose(), "-"*
                                      40, sep = "\n")

print("a")
SVD(table1)
print("b")
SVD(table2)
print("c")
SVD(table3)
```

Output:

```
a
U
[[-0.38887445  0.76978599 -0.50616814]
 [-0.22418008 -0.61196298 -0.75844881]
 [-0.89359944 -0.18146855  0.41054746]]
Sigma
[[14.2766668   0.          0.        ]
 [ 0.          5.55941537  0.        ]
 [ 0.          0.          2.69623553]]
V
[[-0.7071046   0.5980468  -0.37728386]
 [-0.09816334 -0.61141438 -0.78519833]
 [-0.70026213 -0.51818191  0.49104019]]
----------------------------------------
b
U
[[-0.92731392  0.23785583 -0.24014796 -0.08741628  0.13491051]
 [-0.32653768 -0.71841854  0.21238742  0.50925057 -0.26982102]
 [-0.09609671 -0.11500782  0.07653054 -0.64892688 -0.74200782]
 [-0.08902678  0.60317161  0.62519865  0.35795854 -0.33053076]
 [-0.12767809 -0.22417987  0.70745337 -0.42869784  0.4991689 ]]
Sigma
[[27.05677454  0.          0.          0.        ]
 [ 0.          7.73282316  0.          0.        ]
 [ 0.          0.          4.1758625   0.        ]
 [ 0.          0.          0.          3.56322468]]
V
[[-0.38410515  0.30860637 -0.59594071 -0.63409779]
 [-0.56430077 -0.21970607 -0.44909512  0.65696839]
 [-0.61061886 -0.50218264  0.50382081 -0.34802557]
 [-0.40146293  0.77736989  0.43512393  0.21258112]]
```

```
----------------------------------------
c
U
[[-0.69924393  0.68659329 -0.19911699]
 [-0.20707103 -0.46111625 -0.86284031]
 [-0.68423645 -0.56210449  0.46460632]]
Sigma
[[1.53807062e+01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 3.66522525e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.84610916e-16]]
V
[[-0.80446843 -0.25906807 -0.53452248]
 [-0.38574666 -0.45644536  0.80178373]
 [-0.45169687  0.85119996  0.26726124]]
----------------------------------------
```

3

3

(a) $[A\ b]$ has rank $=3$, A has rank $=3$, (square, invertible)
$\det(A) \neq 0$. so it has unique solution

(b) $[A\ b]$ has rank $=2$, A has rank $=2$
$\det(A) = 0$, so not invertible, and b is in A's column space, thus it has multiple solutions

(c) $[A b]$ has rank $=3$, A has rank $=2$
$\det(A)=0$, So not invertible, and b not in column space of A, thus it has 0 solutions

For (b) and (c), they hav the same SVD solution because:

u_i, a column vector of U, is a unit eigen vector of AA^T. The first k columns of U are the k orthonormal basis of A's column space.

For $b \notin A$'s column space. although we can't solve $Ax=b$, We can solve $Ax=b'$ with $b' \in A$'s column space, and $|b-b'|^2$ as small as possible. for (c), $b-b' = \begin{pmatrix}12\\1\\7\end{pmatrix} - \begin{pmatrix}15\\14\\0\end{pmatrix} = \begin{pmatrix}-3\\-13\\7\end{pmatrix}$, and $A^T(b-b') = 0$, the difference vector is in the null of A, thus $(b-b') \perp$ column space of A. as (b) and (c) has the same b in A's column space, they have the same solution code

```
### Question3
# Input test matrix for SVD
input_matrix1 = [
    [7, -1, 1],
    [1, 4, 3],
    [8, 1, 10]
]

b1 = [2, 0, -20]

input_matrix2 = [
    [8, 3, 7],
    [3, 2, 0],
    [9, 5, 3]
]

b2 = [12, 1, 7]

b3 = [15, 14, 0]

def svd_solve(input_matrix, b, thresh):
  b = np.array(b)
  input_matrix = np.array(input_matrix)
  U, D, VT = np.linalg.svd(input_matrix) #resub
  print("U", U, "Sigma", np.diag(D), "V", VT.transpose(), sep = "\n
                                    ")
  # with unique exact solution
  if abs(D[-1]) >= thresh:
    x = np.linalg.solve(input_matrix, b)
    print("unique exact solution x", x, sep = "\n")
  # if it has many or zero solutions
  else:
    for i in range(len(D)):
      if abs(D[i]) >= thresh:
        D[i] = 1/D[i]
    x_bar = np.matmul(np.matmul(VT.transpose(), np.matmul(np.diag(D
                                      ), U.transpose())), b)
    Ax_bar = np.matmul(input_matrix, x_bar)
    # print("Ax_bar", Ax_bar, b, sep = "\n")
    flag = True
    for i in range(len(Ax_bar)):
      if abs(Ax_bar[i] - b[i]) >= thresh:
        flag = False
    # if there are zero solutions
    if not flag:
      print("There are 0 solutions, presenting SVD solution x_bar")
                                        #resub, add reason
      print("x_bar", x_bar, sep = "\n")
    else:
      x = np.linalg.solve(input_matrix, input_matrix[ :,0])
      x_bar_new = np.matmul(np.matmul(VT.transpose(), np.matmul(np.
                                      diag(D), U.transpose())),
                                      input_matrix[:, 0])  #resub
                                      , name clash
      x_n = np.subtract(x, x_bar_new) #resub, name clash
      print("There are multiple solutions") #resub, add reason
```

```
        print("x", x, "x_bar", x_bar, "all x", str(x_bar) + "+ k*" +
                                          str(x_n), sep = "\n")
    print("-"*40)


#U, D, V = np.linalg.svd(input_matrix)
#x = np.linalg.solve(input_matrix, b)
#print("U", U, "Sigma", np.diag(D), "V", V, "unique exact solution
                                   x", x, sep = "\n")

#print("-"*40)

#U2, D2, V2 = np.linalg.svd(input_matrix2)
#x2 = np.matmul(np.matmul(V2, np.matmul(np.diag(np.reciprocal(D2)),
                                    U2.transpose())), b2)
#Ax2 = np.matmul(input_matrix2, x2)
#print("U2", U2, "Sigma2", np.diag(D2), "V2", V2, "x2 bar", x2, "
                                   Ax2bar", Ax2, sep = "\n")

#print("-"*40)

t = 0.0001

print("a")
svd_solve(input_matrix1, b1, t)
print("b")
svd_solve(input_matrix2, b2, t)
print("c")
svd_solve(input_matrix2, b3, t)
```

Output:

```
a
U
[[-0.38887445  0.76978599 -0.50616814]
 [-0.22418008 -0.61196298 -0.75844881]
 [-0.89359944 -0.18146855  0.41054746]]
Sigma
[[14.2766668   0.          0.        ]
 [ 0.          5.55941537  0.        ]
 [ 0.          0.          2.69623553]]
V
[[-0.7071046   0.5980468  -0.37728386]
 [-0.09816334 -0.61141438 -0.78519833]
 [-0.70026213 -0.51818191  0.49104019]]
unique exact solution x
[ 1.  2. -3.]
----------------------------------------
b
U
[[-0.69924393  0.68659329 -0.19911699]
 [-0.20707103 -0.46111625 -0.86284031]
 [-0.68423645 -0.56210449  0.46460632]]
Sigma
[[1.53807062e+01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 3.66522525e+00 0.00000000e+00]
```

```
 [0.00000000e+00 0.00000000e+00 1.84610916e-16]]
V
[[-0.80446843 -0.25906807 -0.53452248]
 [-0.38574666 -0.45644536  0.80178373]
 [-0.45169687  0.85119996  0.26726124]]
There are multiple solutions
x
[ 1. -0.  0.]
x_bar
[ 0.42857143 -0.14285714  1.28571429]
all x
[ 0.42857143 -0.14285714  1.28571429]+ k*[ 0.28571429 -0.42857143 -
                                  0.14285714]
----------------------------------------
c
U
[[-0.69924393  0.68659329 -0.19911699]
 [-0.20707103 -0.46111625 -0.86284031]
 [-0.68423645 -0.56210449  0.46460632]]
Sigma
[[1.53807062e+01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 3.66522525e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 1.84610916e-16]]
V
[[-0.80446843 -0.25906807 -0.53452248]
 [-0.38574666 -0.45644536  0.80178373]
 [-0.45169687  0.85119996  0.26726124]]
There are 0 solutions, presenting SVD solution x_bar
x_bar
[ 0.42857143 -0.14285714  1.28571429]
----------------------------------------
```

5

Deriviation

⑤ Before transformation: $p_1 \cdots p_n$

After transformation: $q_1, \cdots q_n$

To transform the points, we have a rotation matrix and some translation. Denote rotation as matrix R, translation as t. $q_i = Rp_i + t$, for $i = 1, \cdots n$.

We need to find the minimum of

$$D = \sum_{i=1}^{n} \|(Rp_i + t) - q_i\|^2 \text{ with respect to } R \text{ and } t$$

taking $\frac{\partial D}{\partial t} = 0$:

$$2nt + 2R\sum_{i=1}^{n} p_i - 2\sum_{i=1}^{n} q_i = 0$$

$$2nt = 2\sum_{i=1}^{n} q_i - 2R\sum_{i=1}^{n} p_i$$

$$t = \frac{\sum_{i=1}^{n} q_i - R\sum_{i=1}^{n} p_i}{n}$$

$$t = \bar{q} - R\bar{p}$$

Now $D = \sum_{i=1}^{n} \|(\bar{q} - R\bar{p}) - (Rp_i - q_i)\|^2$

$$= \sum_{i=1}^{n} \| R\underbrace{(p_i - \bar{p})}_{x_i} - \underbrace{(q_i - \bar{q})}_{y_i}\|^2$$

denote: x_i y_i

Now $D = \sum_{i=1}^{n} \|Rx_i - y_i\|^2 = \sum_{i=1}^{n} (Rx_i - y_i)^T (Rx_i - y_i)$

$$= \sum_{i=1}^{n} (x_i^T R^T R x_i - x_i^T R^T y_i - y_i^T R x_i + y_i^T y_i)$$

As R is an orthogonal matrix and $\|R\| = 1$,

$$D = \sum_{i=1}^{n} (x_i^T x_i - 2x_i^T R^T y_i + y_i^T y_i)$$

To minimize D, we need to maximize $\sum_{i=1}^{n} x_i^T R^T y_i$

Which means maximizing $\sum_{i=1}^{n} Tr(R x_i y_i^T)$

let $X = \begin{pmatrix} x_1 & \cdots & x_n \\ | & & | \end{pmatrix}$ $\quad Y = \begin{pmatrix} y_1 & \cdots & y_n \\ | & & | \end{pmatrix}$

We maximize $Tr(R \underbrace{XY^T})$

Do SVD decomposition of XY^T :

$$Tr(RU\Sigma V^T) = Tr(\Sigma \overbrace{V^T R U}^{\text{denote as } A})$$

V, R, U are orthogonal matrices, thus $V^T R U$ is orthogonal as well. All its column vectors are orthonormal. That means all its column vectors a_i have $\|a_i\|^2 = 1$, and any $a_{ij} \leq 1$.

$$Tr(\Sigma A) = \sum_{i=1}^{3} \sigma_i a_{ii}$$

Only when all a_{ii} for $i = 1 \cdots 3$ are equal to 1 can we have maximum $Tr(\Sigma A)$ Thus $A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$A = V^T R U = I$$

$$R = VU^T$$

Therefore, $t = \bar{q} - VU^T \bar{p}$ $\quad$ where $\bar{q} = \frac{\sum_{i=1}^{n} q_i}{n}$, $\bar{p} = \frac{\sum_{i=1}^{n} p_i}{n}$

code

I have tested my approach in MATLAB by 1st generating two sets of points which are separated by a rotation and translation, followed by running my algorithm on the 2 set of points and see if the recovered rotation and translation matches the actual transformation between 2 sets of points

```
% HW1-Question5
clear all; close all
%% Generating the initial point set and the final point set after
                                  translating and rotating
% Randomly generating the rotation angles around the x-, y-, and z-
                                  axis
theta  = rand(3,1)*2*pi;
% Rotating Matrix around the X-axis
Rx = [1,0,0; ...
    0, cos(theta(1)), -sin(theta(1));...
    0, sin(theta(1)), cos(theta(1))];
% Rotating Matrix around the Y-axis
Ry = [cos(theta(2)),0, -sin(theta(2));...
    0,1,0; ...
    sin(theta(2)), 0, cos(theta(2))];
% Rotating Matrix around the Z-axis
Rz = [cos(theta(3)), -sin(theta(3)),0;...
    sin(theta(3)), cos(theta(3)),0;...
    0,0,1];
% Triaxial rotation matrix, rotates first around x, then around y,
                                  and finally around the z axis
R0 = Rz*Ry*Rx
DetofR0 = det(R0)

% Randomly generating translation vector
t0 = 8*(rand(3,1)-0.5)

%Randomly generating an initial point set of a rigid body
n = 4
P = rand(3,n)

% The final point set after translating and rotating
Q = R0 * P + t0*ones(1,n)

%% My  Scheme

% Compute the average centers of the initial point set and the
                                  final point set
p0 = mean(P,2);
q0 = mean(Q,2);

%  Coordinates of initial point set and the final point set
                                  Relative to the corresponding
                                  average centers
X = P - p0*ones(1,n);
Y = Q - q0*ones(1,n);

[U,S,V] = svd(X*(Y.'))
R = V*(U.')
t = q0 - R*p0

%  compute error to verify the scheme
difofR = R - R0
difoft = t - t0
```

Output:

```
    R0 =

   -0.1792   -0.6862    0.7050
    0.1239    0.6951    0.7081
   -0.9760    0.2143   -0.0396



t0 =

    3.1274
   -1.3267
    1.5900


n =

     8


P =

  Columns 1 through 7

    0.1978    0.5000    0.6099    0.8055    0.2399    0.4899    0.
                                      7127
    0.0305    0.4799    0.6177    0.5767    0.8865    0.1679    0.
                                      5005
    0.7441    0.9047    0.8594    0.1829    0.0287    0.9787    0.
                                      4711

  Column 8

    0.0596
    0.6820
    0.0424


Q =

  Columns 1 through 7

    3.5955    3.3463    3.2001    2.7162    2.4963    3.6143    2.
                                      9883
   -0.7541   -0.2905   -0.2132   -0.6964   -0.6604   -0.4562   -0.
                                     5569
    1.3740    1.1690    1.0931    0.9202    1.5446    1.1091    0.
                                      9830

  Column 8

    2.6786
   -0.8152
    1.6762

R =
```

```
   -0.1792   -0.6862    0.7050
    0.1239    0.6951    0.7081
   -0.9760    0.2143   -0.0396


t =

    3.1274
   -1.3267
    1.5900


difofR =

   1.0e-15 *

    0.0833         0    0.1110
    0.1388         0    0.2220
    0.1110    0.1665    0.2082


difoft =

   1.0e-15 *

         0
         0
   -0.2220
```

As the difofR and difoft are of order 10^{-15}, thus this proves that my approach works