16811 Math Fundamentals of Robotics
Carnegie Mellon University
Fall 2020
September 17, 2020

Homework 1

1 Solutions

1. The code to this problem is run as:

   ```
   $ python q1.py --do_assert
   ```

   Here, the flag *-do_ assert* is optional and is used to verify if the results are correct, i.e. PA = LDU and Ax = b.

   I provided the following examples:

   Example 1:

   $$\begin{pmatrix} 1 & -2 & 1 \\ 1 & 2 & 2 \\ 2 & 3 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 9 \\ 2 \end{pmatrix} \quad \text{with solution: } x = \begin{pmatrix} 97 \\ 16 \\ -60 \end{pmatrix}$$

   Example 2:

   $$\begin{pmatrix} 2 & 3 & 3 \\ 1 & -2 & 1 \\ 3 & -1 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -4 \\ 3 \end{pmatrix} \quad \text{with solution: } x = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$$

   Example 3: (filename: Ab_3.txt)[1]

   $$\begin{pmatrix} 1 & 2 & 1 & 4 \\ 2 & 0 & 4 & 3 \\ 4 & 2 & 2 & 1 \\ -3 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 13 \\ 28 \\ 20 \\ 6 \end{pmatrix} \quad \text{with solution: } x = \begin{pmatrix} 3 \\ -1 \\ 4 \\ 2 \end{pmatrix}$$

   Example 4:[2]

[1] This example is taken from: https://slideplayer.com/slide/4598671/
[2] This example is taken from: https://www.youtube.com/watch?v=hwyNaFtZmoM

$$\begin{pmatrix} 2 & -1 & 4 & 1 & -1 \\ -1 & 3 & -2 & -1 & 2 \\ 5 & 1 & 3 & -4 & 1 \\ 3 & -2 & -2 & -2 & 3 \\ -4 & -1 & -5 & 3 & -4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 7 \\ 1 \\ 33 \\ 24 \\ -49 \end{pmatrix} \quad \text{with solution: } x = \begin{pmatrix} 1 \\ -2 \\ 3 \\ -4 \\ 5 \end{pmatrix}$$

Note: The code also allows to create a random example. Currently it is configured to create a matrix A^{nxn} and a vector b^{nx1} where n can have a value from 2 to 10 and random inner values v from -100 to 100.

To run this setting:

```
$ python q1.py --random --do_assert
```

2. I used the LDU decomposition code from question 1 for square, non-singular matrices. For matrices that do not satisfy this condition, I used a predefined function from the Scipy package in Python. For SVD, I also used a pre-defined functions from Scipy.

   The code for this question is run as:

   ```
   $ python q2.py
   ```

   Below, I provide the results from the code using floating point precision of 3.

   Decomposition of $A_1 = \begin{pmatrix} 10 & -10 & 0 \\ 0 & -4 & 2 \\ 2 & 0 & -5 \end{pmatrix}$

   LDU:

   $$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.2 & -0.5 & 1 \end{pmatrix} D = \begin{pmatrix} 10 & 0 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & -4 \end{pmatrix} U = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{pmatrix}$$

   SVD:

   $$U = \begin{pmatrix} -0.975 & 0.019 & -0.221 \\ -0.200 & -0.509 & 0.837 \\ -0.097 & 0.860 & 0.501 \end{pmatrix} S = \begin{pmatrix} 14.498 & 0.000 & 0.000 \\ 0.000 & 5.947 & 0.000 \\ 0.000 & 0.000 & 1.856 \end{pmatrix} V = \begin{pmatrix} -0.686 & 0.728 & 0.006 \\ 0.322 & 0.310 & -0.895 \\ -0.653 & -0.612 & -0.447 \end{pmatrix}$$

Decomposition of $A_2 = \begin{pmatrix} 5 & -5 & 0 & 0 \\ 5 & 5 & 5 & 0 \\ 0 & -1 & 4 & 1 \\ 0 & 4 & -1 & 2 \\ 0 & 0 & 2 & 1 \end{pmatrix}$

LDU:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & -0.1 & 1 & 0 \\ 0 & 0.4 & -0.66 & 1 \\ 0 & 0 & 0.44 & 0.20 \end{pmatrix} D = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 4.5 & 0 \\ 0 & 0 & 0 & 2.66 \end{pmatrix} U = \begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & 0.5 & 0 \\ 0 & 0 & 1 & 0.22 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

SVD:

$$U = \begin{pmatrix} 0.113 & 0.867 & 0.375 & -0.308 & -0.021 \\ -0.932 & 0.152 & 0.163 & 0.285 & 0.021 \\ -0.202 & 0.226 & -0.750 & -0.317 & -0.496 \\ -0.240 & -0.412 & 0.383 & -0.771 & -0.177 \\ -0.142 & 0.064 & -0.352 & -0.360 & 0.850 \end{pmatrix} S = \begin{pmatrix} 9.145 & 0 & 0 & 0 \\ 0 & 7.798 & 0 & 0 \\ 0 & 0 & 4.421 & 0 \\ 0 & 0 & 0 & 2.24 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.448 & -0.654 & -0.603 & -0.090 \\ 0.654 & -0.699 & 0.283 & -0.069 \\ 0.608 & 0.276 & -0.741 & -0.076 \\ -0.051 & 0.087 & 0.092 & -0.991 \end{pmatrix}$$

Decomposition of $A_3 = \begin{pmatrix} 1 & 1 & 1 \\ 10 & 2 & 9 \\ 8 & 0 & 7 \end{pmatrix}$

LDU (Result from code performs permutation by default):

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.8 & 1 & 0 \\ 0.1 & -0.5 & 1 \end{pmatrix} D = \begin{pmatrix} 10 & 0 & 0 \\ 0 & -1.6 & 0 \\ 0 & 0 & 0 \end{pmatrix} U = \begin{pmatrix} 1 & 0.2 & 0.9 \\ 0 & 1 & 0.125 \\ 0 & 0 & 0 \end{pmatrix} P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

LDU (Result obtained by hand keeping $P = I$):

Step 1: Pivot at Aij with i=1, j=1. Eliminate leading values of rows 2 and 3.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 10 & 1 & 0 \\ 8 & ? & 1 \end{pmatrix} A' = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -8 & -1 \\ 0 & -8 & -1 \end{pmatrix} \quad \begin{matrix} row2 = row2 - 10row1 \\ row3 = row3 - 8row1 \end{matrix}$$

Step 2: Pivot at Aij with i=2, j=2. Eliminate value of col=2 row=3.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 10 & 1 & 0 \\ 8 & 1 & 1 \end{pmatrix} A' = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -8 & -1 \\ 0 & 0 & 0 \end{pmatrix} \quad row3 = row3 - row2$$

Step 3: Find D from the diagonal of $A^{'}$ and U by normalizing $A^{'}$ with D.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 10 & 1 & 0 \\ 8 & 1 & 1 \end{pmatrix} D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & 0 \end{pmatrix} U = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0.125 \\ 0 & 0 & 0 \end{pmatrix}$$

SVD:

$$U = \begin{pmatrix} -0.087 & -0.571 & -0.816 \\ -0.786 & -0.464 & 0.408 \\ -0.612 & 0.677 & -0.408 \end{pmatrix} S = \begin{pmatrix} 17.283 & 0 & 0 \\ 0 & 1.513 & 0 \\ 0 & 0 & 0 \end{pmatrix} V = \begin{pmatrix} -0.743 & -0.096 & -0.662 \\ 0.134 & -0.991 & -0.007 \\ 0.656 & 0.094 & -0.749 \end{pmatrix}$$

3. The code used in this problem can be run as:

```
$ python q3.py
```

Below, I provide the results and analysis:

(a) $A = \begin{pmatrix} 1 & 1 & 1 \\ 10 & 2 & 9 \\ 8 & 0 & 7 \end{pmatrix} b = \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}$

- This system has more than one exact solution
- Thus, a solution can be represented as A$\hat{x}$ = b, with $\hat{x} = x + a \cdot xn$, where:
  - a is a scalar number
  - $x = \begin{pmatrix} 0.018 \\ 0.86 \\ 0.123 \end{pmatrix}$ and is the solution obtained through the SVD.
  - $xn = \begin{pmatrix} 0.656 \\ 0.094 \\ -0.749 \end{pmatrix}$ and represents the unit vector that spans the null space and corresponds to the last row of V^T from the SVD solution
- **Proof:** With $a = 1$, $\hat{x} = \begin{pmatrix} 0.673 \\ 0.953 \\ -0.626 \end{pmatrix}$. Then $A\hat{x} = \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} = b$

(b) $A = \begin{pmatrix} 1 & 1 & 1 \\ 10 & 2 & 9 \\ 8 & 0 & 7 \end{pmatrix}$ $b = \begin{pmatrix} 3 \\ 2 \\ 2 \end{pmatrix}$

- This system has zero solutions
- Here, $x = \begin{pmatrix} 0.018 \\ 0.86 \\ 0.123 \end{pmatrix}$ is the solution obtained through SVD. It represents the least square solution (i.e. $x = U\frac{1}{\Sigma}V^T$) of a projection of b' into the column space of A.
- **Proof:** $Ax = \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix} \neq b$

Analysis of (a) and (b): Both of the examples have the same SVD decomposition and least square solution because matrix A is the same. Since matrix A has a vector in the null space, a system may have many solutions only if $b \in colspace(A)$, otherwise it will have no solution. Thus, the difference between examples (a) and (b) is that in the former this condition is true, whereas in the latter it is not.

(c) $A = \begin{pmatrix} 10 & -10 & 0 \\ 0 & -4 & 2 \\ 2 & 0 & -5 \end{pmatrix}$ $b = \begin{pmatrix} 10 \\ 2 \\ 13 \end{pmatrix}$

- The system above has one unique solution given by $x = \begin{pmatrix} -1 \\ -2 \\ -3 \end{pmatrix}$
- **Proof:** $Ax = \begin{pmatrix} 10 \\ 2 \\ 13 \end{pmatrix} = b$

Note: The code also verifies the results and outputs the type of solution found.

4. **(a)** Matrix A makes the component in the direction of u collapse to 0, whereas the components in a direction perpendicular to u do not change. Below is the algebraic derivation.

   - Action over u:

$$\begin{aligned} Au &= (I - uu^T)u \\ &= Iu - uu^T u \quad (\text{where } u^T u = 1) \\ &= u - u \\ &= 0 \end{aligned}$$

   - Action over a vector v perpendicular to u:

$$\begin{aligned} Av &= (I - uu^T)v \\ &= Iv - uu^T v \quad (\text{where } u^T v = 0) \\ &= v \end{aligned}$$

In Figure 1, I show a graphic representation of the action of A over a vector x. Here, matrix A makes the component of u collapse to 0 but keeps the v component unchanged.

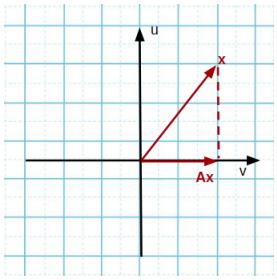


Figure 1: Geometric representation of the action of A.

(b) A householder matrix has eigenvalues ± 1. In the case of matrix A^{nxn}, $Au = 0$ and $Av = v$. This leaves one eigenvalue $= 0$ and $n - 1$ eigenvalues $= 1$ which correspond to the $n - 1$ independent vectors orthogonal to vector u.

Note: I reviewed [3, 4] to answer this question.

(c) The null space of A is spanned by vector u. In section (a) it was proven that $Au = 0$, and in (b) the eigenvalue with value $= 0$ is associated to this vector.

(d) From the derivation shown below $A^2 = A$:

$$\begin{aligned} A^2 &= (I - uu^T)^2 \\ &= I - 2uu^T + uu^T uu^T \quad \text{(where } u^T u = 1\text{)} \\ &= I - 2uu^T + uu^T \\ &= I - uu^T \\ &= A \end{aligned}$$

5. Below, I show the main steps followed to estimate the rigid transformation between two point sets. I used [1, 2] as reference to build the algorithm. The algorithm makes the following assumptions:

   - Both points sets have N points, with $N \geq 3$
   - Points in P correspond to points in Q

   Algorithm

   Step 1: Compute the centroids of the point sets P and Q:

   $$\bar{P} = \frac{1}{N}\sum_{i=1}^{N} p_i \quad \bar{Q} = \frac{1}{N}\sum_{i=1}^{N} q_i$$

   Step 2: Compute the displacement that makes both centroids coincident:

   $$t_{est} = \bar{Q} - \bar{P}$$

   Step 3: Compute a moment matrix W^{3x3} which encodes the rotation matrix between point sets P and Q. Then, find the SVD decomposition of W.

   $$W = (P - \bar{P})(Q - \bar{Q})^T$$
   $$W = U\Sigma V^T$$

   Step 4: The rotation matrix is obtained as.

   $$R_{est} = VU^T$$

   The estimated transformation between the two sets is $[R_{est}, t_{est}]$.

Code

The algorithm can be run as:

```
$ python q5.py
```

The code also allows to specify desired rotation, translation, number of points, and visualization of the point sets. To do so, run as:

```
$ python q5.py --my_params --vis --N n
               --yaw Y --pitch P --roll R
               --x x --y y --z z
```

In this setting, the only required parameter is $-my\_params$. By default $n = 3$ points and there is no rotation nor translation.

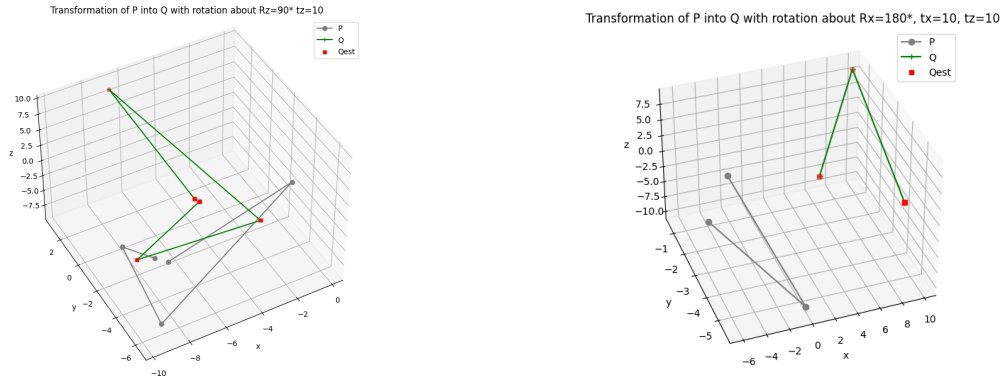
Examples

Figure 2 shows two results of the code. Plot (a) shows a 90 deg rotation about z and translation $z = 10$:

```
$ python q5.py --my_params --vis --N 5 --yaw 90 --z 10
```

Plot (b) shows a 180 deg rotation about x-$axis$ and translations $z = 10$, $x = 10$:

```
$ python q5.py --my_params --vis --N 3 --roll 180 --x 10 --z 10
```



(a) Rz = 90 degrees, tz = 10

(b) Rx = 180 degrees, tx = 10, tz = 10

Figure 2: Transformations from P to Q

References

[1] Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Second. ISBN 978-3-319-54413-7. Springer, 2017. Chap. 14, pp. 505–506.

[2] D.W. Eggert. *Estimating 3-D rigid body transformations: a comparison of four major algorithms*. 1997.

[3] *Eigenvalues of Householder matrix*. `https://math.stackexchange.com/questions/1345165/eigenvalues-of-householder-matrix`. [Online; accessed Sept-12-2020].

[4] *Householder transformation*. `https://en.wikipedia.org/wiki/Householder_transformation`. [Online; accessed Sept-12-2020].