1. I implemented LDU decomposition following the Crout LU decomposition algorithm, and then extracting D from U in the final step. See Appendix for code walkthrough, and separate Python file to run/test the code on its own.

2. (a) *PA=LDU decomposition (from my code):*

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ 0.75 & 1 & 0 \\ 0.5 & 0.462 & 1 \end{pmatrix}$$

$$D = \begin{pmatrix} 4 & 0 & 0 \\ 0 & -3.25 & 0 \\ 0 & 0 & -6.462 \end{pmatrix} \quad U = \begin{pmatrix} 1 & 1.75 & 0 \\ 0 & 1 & -0.308 \\ 0 & 0 & 1 \end{pmatrix}$$

SVD decomposition (from `np.linalg.svd()`):

$$U = \begin{pmatrix} 0.831. & 0.364 & -0.421 \\ 0.321 & 0.304 & 0.897 \\ 0.454 & -0.88 & 0.136 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 9.331 & 0 & 0 \\ 0 & 5.79 & 0 \\ 0 & 0 & 1.555 \end{pmatrix}$$

$$V = \begin{pmatrix} 0.557 & 0.79 & -0.258 \\ 0.105 & 0.241 & 0.965 \\ 0.824 & -0.564 & 0.051 \end{pmatrix}$$

(b) *PA=LDU decomposition (from my code):*

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix} \quad U = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

SVD decomposition (from `np.linalg.svd()`):

$$U = \begin{pmatrix} -0.851 & 0 & 0 & -0.526 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -0.851 & 0 & 0 & -0.526 \\ 0 & -0.526 & 0 & 0 & 0.851 \\ -0.526 & 0 & 0 & 0.851 & 0 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 1.618 & 0 & 0 & 0 & 0 \\ 0 & 1.618 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0.618 & 0 \\ 0 & 0 & 0 & 0 & 0.618 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.851 & 0 & 0 & 0 & -0.526 \\ 0 & -0.851 & 0 & -0.526 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0.526 & 0 & 0 & 0 & -0.851 \\ 0 & 0.526 & 0 & -0.851 & 0 \end{pmatrix}$$

(c) *PA=LDU decomposition (from my code):*

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad L = \begin{pmatrix} 1 & 0 & 0 \\ 0.667 & 1 & 0 \\ 0.333 & 0.5 & 1 \end{pmatrix}$$

$$D = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 0.667 & 0 \\ 0 & 0 & 3.5 \end{pmatrix} \quad U = \begin{pmatrix} 1 & 0.667 & 1.667 \\ 0 & 1 & 2.5 \\ 0 & 0 & 1 \end{pmatrix}$$

SVD decomposition (from `np.linalg.svd()`):

$$U = \begin{pmatrix} -0.586 & -0.044 & -0.809 \\ -0.623 & -0.614 & 0.485 \\ -0.518 & 0.788 & 0.332 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 9.791 & 0 & 0 \\ 0 & 1.416 & 0 \\ 0 & 0 & 0.361 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.364 & -0.3 & -0.882 \\ -0.806 & -0.373 & 0.459 \\ 0.467 & -0.878 & 0.106 \end{pmatrix}$$

3. (a) First, I find the determinant of A to check if it is singular.

$$\begin{aligned}\det(A) &= 2\begin{vmatrix}1 & 2\\ 5 & 5\end{vmatrix} - 1\begin{vmatrix}2 & 2\\ 5 & 5\end{vmatrix} + 3\begin{vmatrix}2 & 1\\ 5 & 5\end{vmatrix}\\ &= 2(5-10) - 1(10-10) + 3(10-5)\\ &= 5\end{aligned}$$

Since $\det(A)$ is nonzero, A is nonsingular, and there exists one unique solution to the system. I use $PA = LDU$ decomposition to solve the nonsingular system $Ax = b$. I use my Python implementation from (1) to find P, L, D, U.

$$P = \begin{pmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{pmatrix} \quad L = \begin{pmatrix}1 & 0 & 0\\ 0.4 & 1 & 0\\ 0.4 & 1 & 1\end{pmatrix} \quad D = \begin{pmatrix}5 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{pmatrix} \quad U = \begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}$$

By substituting $A = LDU$ and $b = P^{-1}b$ (with $P^{-1} = P$ for permutation matrices), we get the system $LDU = Pb$. Rewriting as $Ly = Pb$ with $y = DUx$, I solve for y as follows.

$$Ly = Pb$$

$$\begin{pmatrix}1 & 0 & 0\\ 0.4 & 1 & 0\\ 0.4 & 1 & 1\end{pmatrix}\begin{pmatrix}y_1\\ y_2\\ y_3\end{pmatrix} = \begin{pmatrix}0 & 0 & 1\\ 0 & 1 & 0\\ 1 & 0 & 0\end{pmatrix}\begin{pmatrix}10\\ -10\\ 0\end{pmatrix} = \begin{pmatrix}0\\ -10\\ 10\end{pmatrix}$$

$$\begin{aligned} y_1 &= 0\\ 0.4y_1 + y_2 = -10 &\longrightarrow (0) + y_2 = -10 \longrightarrow y_2 = -10\\ 0.4y_1 + y_2 + y_3 = 10 &\longrightarrow (0) + (-10) + y_3 = 10 \longrightarrow y_3 = 20\end{aligned}$$

Now that we know y, we substitute back in to get x.

$$Ux = D^{-1}y$$

$$\begin{pmatrix}1 & 1 & 1\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix} = \begin{pmatrix}0.2 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}0\\ -10\\ 20\end{pmatrix}$$

$$\begin{aligned} x_1 + x_2 + x_3 = 0 &\longrightarrow x_1 + (10) + (20) = 0 \longrightarrow x_1 = -30\\ x_2 &= 10\\ x_3 &= 20\end{aligned}$$

$$\textbf{Solution:} \qquad x = \begin{pmatrix} -30 \\ 10 \\ 20 \end{pmatrix}$$

(b) Once again, I find the determinant of A to check if it is singular.

$$\begin{aligned} \det(A) &= 8\begin{vmatrix} 2 & -6 \\ 2 & 1 \end{vmatrix} - 14\begin{vmatrix} 2 & -6 \\ 1 & 1 \end{vmatrix} + 0\begin{vmatrix} 2 & 2 \\ 1 & 2 \end{vmatrix} \\ &= 8(2+12) - 14(2+6) + 0(4-2) \\ &= 0 \end{aligned}$$

Because $\det(A) = 0$, A is singular, and no unique solution to the system exists. I reduce the system into reduced row echelon form with Gaussian elimination investigate if the system has zero (overconstrained) or infinite (underconstrained) solutions.

$$\left(\begin{array}{ccc|c} 8 & 14 & 0 & 6 \\ 2 & 2 & -6 & 5 \\ 1 & 2 & 1 & 1 \end{array}\right) \xrightarrow{\text{swap rows 1 \& 3}} \left(\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 2 & 2 & -6 & 5 \\ 8 & 14 & 0 & 6 \end{array}\right) \begin{matrix} \\ -2r_1 \\ -8r_1 \end{matrix}$$

$$\longrightarrow \left(\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -2 & -8 & 3 \\ 0 & -2 & -8 & 2 \end{array}\right) \begin{matrix} \\ \\ -r_2 \end{matrix}$$

$$\longrightarrow \left(\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -2 & -8 & 3 \\ 0 & 0 & 0 & -1 \end{array}\right)$$

Row 3 of this final augmented matrix implies that $0x_1 + 0x_2 + 0x_3 = -1$, which is impossible to satisfy. Thus, **this system has no solution**, and I will find the least-squares approximation via SVD.

SVD decomposition (from `np.linalg.svd()`*):*

$$U = \begin{pmatrix} -0.974 & 0.161 & -0.162 \\ -0.187 & -0.969 & 0.162 \\ -0.131 & 0.188 & 0.973 \end{pmatrix} \quad \Sigma = \begin{pmatrix} 16.532 & 0 & 0 \\ 0 & 6.059 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.502 & -0.863 & 0.06 \\ -0.077 & 0.113 & 0.991 \\ -0.862 & 0.492 & -0.123 \end{pmatrix}$$

Here we see again that Σ is singular, with only two nonzero singular values on the diagonal, and thus cannot be inverted. Thus, we approximate Σ^{-1} as diagonal matrix $\tilde{\Sigma^{-1}}$ defined as

$$\tilde{\Sigma}^{-1}_{i,i} = \begin{cases} \frac{1}{\Sigma_{i,i}} & \Sigma_{i,i} \neq 0 \\ 0 & \Sigma_{i,i} = 0 \end{cases}$$

$$\tilde{\Sigma^{-1}} = \begin{pmatrix} 0.06 & 0 & 0 \\ 0 & 0.165 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Our best approximation $\bar{x}$ is then found as

$$\begin{aligned} Ax &= b \\ x &= A^{-1}b \\ x &= (U\Sigma V^T)^{-1}b \\ \tilde{x} &= V(\tilde{\Sigma^{-1}})U^T b \end{aligned}$$

$$\tilde{x} = \begin{pmatrix} -0.502 & -0.863 & 0.06 \\ -0.077 & 0.113 & 0.991 \\ -0.862 & 0.492 & -0.123 \end{pmatrix} \begin{pmatrix} 0.06 & 0 & 0 \\ 0 & 0.165 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} -0.974 & -0.187 & -0.131 \\ 0.161 & -0.969 & 0.188 \\ -0.162 & 0.162 & 0.973 \end{pmatrix} \begin{pmatrix} 6 \\ 5 \\ 1 \end{pmatrix}$$

Approximate Solution: $\tilde{x} = \begin{pmatrix} 0.736 \\ -0.037 \\ 0.06 \end{pmatrix}$

(c) Once again, I use the determinant to check if A is singular.

$$\begin{aligned} \det(A) &= 4\begin{vmatrix} 2 & -6 \\ 2 & 1 \end{vmatrix} - 7\begin{vmatrix} 2 & -6 \\ 1 & 1 \end{vmatrix} + 0\begin{vmatrix} 2 & 2 \\ 1 & 2 \end{vmatrix} \\ &= 4(2+12) - 7(2+6) + 0(4-2) \\ &= 0 \end{aligned}$$

As in (3b), $\det(A) = 0$, A is singular, and no unique solution to the system exists. This makes sense intuitively as the only difference between this coefficient matrix and that in (3b) is the first row of A, which is scaled down by a factor of 2, which does not affect the rank. To determine if the new system is under- or onverconstrained we must consider the coefficient matrix's relation to b.

I once again perform Gaussian elimination on the augmented matrix.

$$\left(\begin{array}{ccc|c} 4 & 7 & 0 & 18 \\ 2 & 2 & -6 & -12 \\ 1 & 2 & 1 & 8 \end{array}\right) \xrightarrow{\text{swap rows 1 \& 3}} \left(\begin{array}{ccc|c} 1 & 2 & 1 & 8 \\ 2 & 2 & -6 & -12 \\ 4 & 7 & 0 & 18 \end{array}\right) \begin{array}{l} \\ -2r_1 \\ -4r_1 \end{array}$$

$$\longrightarrow \left(\begin{array}{ccc|c} 1 & 2 & 1 & 8 \\ 0 & -2 & -8 & -28 \\ 0 & -1 & -4 & -14 \end{array}\right) \begin{array}{l} \\ \\ -\frac{1}{2}r_2 \end{array}$$

$$\longrightarrow \left(\begin{array}{ccc|c} 1 & 2 & 1 & 1 \\ 0 & -2 & -8 & 3 \\ 0 & 0 & 0 & 0 \end{array}\right)$$

This time, the final row of the augmented matrix implies the equation $0x_1 + 0x_2 + 0x_3 = 0$, which holds for all values of x_i. Thus, the system is underconstrained, and has infinite solutions in 1 degree of freedom.

Find particular solution:

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 1 \\ -2x_2 - 8x_3 &= 3 \end{aligned}$$

Choose $x_3 = 0$.

$$x_1 + 2x_2 + 0 = 1 \longrightarrow x_1 = 1 - 2x_2 \longrightarrow x_1 = 1 - 2(-1.5) \longrightarrow x_1 = 4$$
$$-2x_2 - 8(0) = 3 \longrightarrow x_2 = -1.5$$

Find homogeneous solution:

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 0 \\ -2x_2 - 8x_3 = 0 &\longrightarrow x_2 = -4x_3 \end{aligned}$$

$$x_1 + 2(-4x_3) + x_3 = 0 \longrightarrow x_1 = 7x_3$$

Solution Set: $$\bar{x} = \begin{pmatrix} 4 \\ -1.5 \\ 0 \end{pmatrix} + \lambda \begin{pmatrix} 7 \\ -4 \\ 1 \end{pmatrix}$$

4. (a) Use Gaussian elimination to rewrite in reduced row-echelon form.

$$A = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 1 & 6 \end{pmatrix} \; {\scriptstyle +r_1} \longrightarrow \begin{pmatrix} 1 & 2 & 0 \\ 0 & 3 & 6 \end{pmatrix} \; {\scriptstyle \times \frac{1}{3}} \longrightarrow \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix} \; {\scriptstyle -2r_2} \longrightarrow \begin{pmatrix} 1 & 0 & -4 \\ 0 & 1 & 2 \end{pmatrix}$$

$rank(A) = 2 < 3 = ndim(A)$, so there is one free variable of A (and $ndim(N(A) = 1)$.

$$\begin{pmatrix} 1 & 0 & -4 \\ 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{aligned} x_1 - 4x_3 = 0 &\longrightarrow x_1 = 4x_3 \\ x_2 + 2x_3 = 0 &\longrightarrow x_2 = -2x_3 \\ &\quad\; x_3 = x_3 \end{aligned}$$

$$\mathbf{x} = x_3 \begin{pmatrix} 4 \\ -2 \\ 1 \end{pmatrix} \text{for any value of } x_3.$$

In other words, $[4 \;\; -2 \;\; 1]^T$ is a basis for $nullspace(A)$.

(b) Use Gaussian elimination to rewrite in reduced row-echelon form.

$$A = \begin{pmatrix} 2 & 1 \\ 3 & 2 \end{pmatrix} \; {\scriptstyle \times \frac{1}{2}} \longrightarrow \begin{pmatrix} 1 & \frac{1}{2} \\ 3 & 2 \end{pmatrix} \; {\scriptstyle -3r_1} \longrightarrow \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{1}{2} \end{pmatrix} \; {\scriptstyle -r_2} \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}$$

$rank(A) = 2 = 2 = ndim(A)$, so there are no free variables of A (and $ndim(N(A) = 0)$. Thus, the nullspace consists of only the zero vector, $[0 \;\; 0]^T$.

5.

$$A = \begin{pmatrix} 1 & 1 & 3 \\ 1 & 2 & 4 \\ 1 & 3 & x \end{pmatrix} \qquad b = \begin{pmatrix} 2 \\ 3 \\ y \end{pmatrix}$$

(a) The system will have a unique solution iff $\det(A) \neq 0$.

$$\begin{aligned} \det(A) &= 1 \begin{vmatrix} 2 & 4 \\ 3 & x \end{vmatrix} - 1 \begin{vmatrix} 1 & 4 \\ 1 & x \end{vmatrix} + 3 \begin{vmatrix} 1 & 2 \\ 1 & 3 \end{vmatrix} \\ &= 1(2x - 12) - 1(x - 4) + 3(3 - 2) \\ &= x - 5 \end{aligned}$$

The system will have a unique solution for $x \neq 5$.

(b) The system will have no solution only in the case that $\det(A) = 0$, which occurs where $x = 5$.

$$A^* = \left(\begin{array}{ccc|c} 1 & 1 & 3 & 2 \\ 1 & 2 & 4 & 3 \\ 1 & 3 & 5 & y \end{array}\right) \begin{array}{c} \\ -r_1 \\ -r_1 \end{array} \longrightarrow \left(\begin{array}{ccc|c} 1 & 1 & 3 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 2 & 2 & y-2 \end{array}\right) \begin{array}{c} \\ \\ -2r_2 \end{array}$$

$$\longrightarrow \left(\begin{array}{ccc|c} 1 & 1 & 3 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & y-4 \end{array}\right) \begin{array}{c} \\ -r_2 \\ \end{array} \longrightarrow \left(\begin{array}{ccc|c} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & y-4 \end{array}\right)$$

$$\begin{aligned} x_1 + 2x_3 &= 1 \\ x_2 + x_3 &= 1 \\ 0x_1 + 0x_2 + 0x_3 &= y - 4 \end{aligned}$$

The third equation is only consistent if the RHS is 0, so the system has no solution for all $y \neq 4$.

(c) The system will have infinitely many solutions only in the case that $\det(A) = 0$, which occurs where $x = 5$. We know that the system has no solution for all $y \neq 4$, so let us consider the final case, where $y = 4$.

$$\left(\begin{array}{ccc|c} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & (4)-4 \end{array}\right) = \left(\begin{array}{ccc|c} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right)$$

$$\begin{aligned} x_1 + 2x_3 &= 1 \\ x_2 + x_3 &= 1 \\ 0x_1 + 0x_2 + 0x_3 &= 0 \end{aligned}$$

This time, the system is underconstrained, as we have two (meaningful) equations (a.k.a. $rank(A^*) = 2$) and 3 unknowns ($ncols(A) = 3$). I define x_3 as our free variable, and thus solutions can be found for any value of x_3 (i.e. there are infinite solutions).

Find particular solution for $x_3 = 0$:

$$\begin{array}{rcl} x_1 & = & 1 \\ x_2 & = & 1 \\ x_3 & = & 0 \end{array} \longrightarrow \mathbf{x}_p = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

Find homogeneous solution:

$$x_1 + 2x_3 = 0 \longrightarrow x_1 = -2x_3$$
$$x_2 + x_3 = 0 \longrightarrow x_2 = -x_3$$
$$\mathbf{x}_h = \begin{pmatrix} -2x_3 \\ -x_3 \\ x_3 \end{pmatrix}$$

Solution Set: $\mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + \lambda \begin{pmatrix} -2 \\ -1 \\ 1 \end{pmatrix}$

6. Our system is represented by $\mathbf{M}\vec{u} = \vec{d}$ where $\mathbf{M}$ is a 6×6 matrix, $\vec{d}$ is a 6×1 vector, and $\vec{u} = [r_{11}\ r_{12}\ r_{21}\ r_{22}\ t_1\ t_2]^T$ is a 6×1 vector representing an unknown affine transformation from image $\mathbf{P}$ to $\mathbf{P'}$ such that

$$\mathbf{P'} = \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{P}\ .$$

Let $p_i = [x_i\ y_i\ 1]^T$ represent point i (represented in homogeneous coordinates) in the image $\mathbf{P}$, and $p_i' = [x_i'\ y_i'\ 1]^T$ represent the corresponding point in image $\mathbf{P'}$. Let $\mathbf{p} = [p_1\ p_2\ p_3]$ and $\mathbf{p'} = [p_1'\ p_2'\ p_3']$. We can write the transformation of these corresponding points as

$$\mathbf{p'} = \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{p}\ .$$

which expands into

$$\begin{pmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x_1' & x_2' & x_3' \\ y_1' & y_2' & y_3' \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} r_{11}x_1 + r_{12}y_1 + t_1 & r_{11}x_2 + r_{12}y_2 + t_1 & r_{11}x_3 + r_{12}y_3 + t_1 \\ r_{21}x_1 + r_{22}y_1 + t_2 & r_{21}x_2 + r_{22}y_2 + t_2 & r_{21}x_3 + r_{22}y_3 + t_2 \\ 1 & 1 & 1 \end{pmatrix}$$

This can be rewritten as the set of linear equations

$$
\begin{aligned}
r_{11}x_1 + r_{12}y_1 + t_1 &= x'_1 \\
r_{21}x_1 + r_{22}y_1 + t_2 &= y'_1 \\
r_{11}x_2 + r_{12}y_2 + t_1 &= x'_2 \\
r_{21}x_2 + r_{22}y_2 + t_2 &= y'_2 \\
r_{11}x_3 + r_{12}y_3 + t_1 &= x'_3 \\
r_{21}x_3 + r_{22}y_3 + t_2 &= y'_3
\end{aligned}
$$

I then factor out our non-unknowns to achieve the form $\mathbf{M}\vec{u} = \vec{d}$.

$$
\begin{pmatrix}
x_1 & y_1 & 0 & 0 & 1 & 0 \\
0 & 0 & x_1 & y_1 & 0 & 1 \\
x_2 & y_2 & 0 & 0 & 1 & 0 \\
0 & 0 & x_2 & y_2 & 0 & 1 \\
x_3 & y_3 & 0 & 0 & 1 & 0 \\
0 & 0 & x_3 & y_3 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
r_{11} \\ r_{12} \\ r_{21} \\ r_{22} \\ t_1 \\ t_2
\end{pmatrix}
=
\begin{pmatrix}
x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3
\end{pmatrix}
$$

Appendix

Problem 1: Code and Explanation

Code developed in Jupyter Notebook.

Python Imports

[1]:
```
import numpy as np
```

Define helper functions

[2]:
```
##############################################################################
########################### UTILITY FUNCTIONS ################################
##############################################################################

def swap_rows(M,row1,row2):
    '''
    Swap the rows of 2D matrix M in place.
    '''
    M[[row1,row2]] = M[[row2,row1]]

def find_pivot_row(X,n=0):
    '''
    Get index of row with the largest (by magnitude) element in column n.
    '''
    return np.argsort(-np.abs(X[n:,n]))[0] + n


def perm_mat(N,row1,row2):
    '''
    Return the permutation matrix that Swaps row1 and row2 for an NxN matrix.
    '''
    P = np.eye(N)
    swap_rows(P,row1,row2)
    return P



##############################################################################
################## TESTING FUNCTIONS #########################################
##############################################################################


def sp_ldu(A):
    '''
```

```
    Perform lu decomposition from built-in scipy.lu() function, then extract
    D using my factor_out_D to get ground-truth p,l,d,u.
    Note: scipy considers the decomposition to be A = PLDU, which is why we
    use the inverse of the output p from lu().
    '''
    from scipy.linalg import lu
    inv_p,l,du = lu(A)
    p = np.linalg.inv(inv_p)
    d,u = factor_out_D(du)
    return p,l,d,u

def test_decomp(A,P,L,D,U):
    '''
    Test that P,L,D,U extracted by my functions match those calculated
    by the built-in scipy.lu() function (applied in sp_ldu()).
    '''
    p,l,d,u = sp_ldu(A)
    assert (p == P).all()
    assert (l == L).all()
    assert (d == D).all()
    assert (u == U).all()
    print("Success!")

def print_LDU(A,P,L,D,U):
    '''
    Print results of decomposition for debugging.
    '''
    print('A')
    print(A)
    print()
    print('P')
    print(P)
    print('DU')
    print(np.matmul(D,U))
    print('L')
    print(L)
    print('D')
    print(D)
    print('U')
    print(U)
    print()
    print('PA')
    print(np.matmul(P,A))
    print('LDU')
```

```
    print(np.matmul(L,np.matmul(D,U)))
```

Define matrices A,B,C from problem 2 parts (a), (b), and (c), respectively. These matrices will be used to test my decomposition against the built-in scipy LU decomposition, as well as for answering the LDU decomposition part of problem 2.

I explicitly set the datatype to double, otherwise they are implicitly cast as integers.

[3]:
```
## Matrices from hw1 problem 2

A = np.array(
    [[4, 7, 0],
     [3, 2, 1],
     [2, 2, -6]
    ],
    dtype='double'
)

B = np.array(
    [[1,0,0,0,1],
     [0,0,1,0,0],
     [0,1,0,1,0],
     [0,1,0,0,0],
     [1,0,0,0,0],
    ],
    dtype='double'
)


C = np.array(
    [[2, 2, 5],
     [3, 2, 5],
     [1, 1, 5]
    ],
    dtype='double'
)
```

Define functions for each of the main "steps" of LDU decomposition. In the permutation step, we perform partial-pivoting by rearranging the rows such that we never hit a 0 pivot. To do this, for each pivot, we swap the row with the highest magnitude potential pivot into the pivot row before moving to the elimination step. This pivoting is reflected in transformations of P and L.

In the elimination step, we perform basic Gaussian elimination to eliminate all elements in

the pivot column in lower rows to 0 through elementary operations, eventually building the upper-triangular matrix DU. The inverse of these operations is stored in L.

In the DU factoring step, we factor out D and U from DU such that U is upper-triangular with 1s on the diagonal, and D is a diagonal matrix. This occurs by dividing each row of DU by the pivot value, and storing the original pivot values on the diagonal of D.

[4]:
```
##############################################################################
########################## MAIN STEPS ########################################
##############################################################################


def permutation_step(DU,L=None,P=None,n=0):
    '''
    Permute X to place highest magnitude pivot of X in current row (n) by␣
 ↪swapping,
    and permute L to keep row correspondence and lower triangular form. Record
    permutation in P.

    PA = LDU
    this step permutation: Pn
    (Pn P)A = (Pn L Pn)(Pn DU)
    '''
    if P is None:
        P = np.eye(len(DU))
    if L is None:
        L = np.eye(len(DU))

    pivot = find_pivot_row(X=DU,n=n)
    if pivot != n:
        # Find permutation for this step
        Pn = perm_mat(len(DU),pivot,n)
        # Update total permutation matrix -> Pn(PA)
        P = np.matmul(Pn,P)
        DU = np.matmul(Pn,DU)
        L = np.matmul(Pn,np.matmul(L,Pn))

    return P,L,DU


def elimination_step(DU,L=None,n=0):
    '''
    Perform Gaussian elimination on rows below pivot row (n) such that
    the only nonzero element of the pivot column is the pivot.
```

```
    '''
    if L is None:
        L = np.eye(len(DU))

    for nn in range(n+1,len(DU)):
        if DU[n,n] != 0:
            factor = DU[nn,n]/DU[n,n]
            DU[nn] -= factor*DU[n]  # eliminate element in pivot column by␣
↪subtracting
                                    # scaled pivot row from each lower␣
↪row
            L[nn,n] = factor        # Store (inverse) elementary operations in L

    return L,DU

def factor_out_D(DU):
    '''
    DU should be an upper triangular matrix. This function will
    factor out the elements on the diagonal so they are set to 1.
    The diagonal elements are returned in D, the remaining upper
    triangular matrix (with 1s on the diag) in U.
    '''
    diag = np.diag(DU)  # extract the diagonal of DU
    D = np.diag(diag)  # make into a diagonal matrix
    U = DU / diag[:,None]  # factor out D from DU

    return D,U
```

Define a function to perform the steps of the LDU decomposition algorithm, calling the steps from above. Permuation and elimation are called for each pivot to extract P, L, and DU from the input matrix, and then the final factoring step is called to extract D and U from DU.

The decomposition is run for the matrices defined in Problem 2 (a), (b), and (c), and tested against the (slightly modified) built-in scipy implementation. The purpose of the modification is effectively only to turn the LU decomposition into an LDU decomposition and define P as a LHS permutation matrix instead of a RHS one.

[5]:
```
def LDU(A):
    # P and L begin as identity matrices
    P = np.eye(len(A)) # assuming square matrix
    L = np.eye(len(A))
    DU = A.copy() # this matrix will have L factored out until it is DU
```

```
    for n in range(len(A-1)):
        P,L,DU = permutation_step(DU=DU,L=L,P=P,n=n)
        L,DU   = elimination_step(DU=DU,L=L,n=n)
    D,U = factor_out_D(DU)

    return P, L, D, U

X = [A,B,C]
for x in X:
    P,L,D,U = LDU(x)
    test_decomp(A=x,P=P,L=L,D=D,U=U) # Check against built-in method
```

[Result]:
```
Success!
Success!
Success!
```

All tests passed (implying P, L, D, U from my algorithm all match their ground-truth counterparts from scipy) for all 3 input matrices.