1 Exercise 1

No "final solutions" are provided for these types of questions, since the whole point of them is to encourage you to express *briefly* but *clearly* and *in your own words* what you understand. As explained in the directions, definitions taken from text books or the internet do not reflect a good understanding of these terms, nor do extremely long explanations. Equations do not express the meaning of these, nor do literal word translations of equations show that you know what they mean. Instead, we are looking for clear evidence that you understand what each term means. Possible definitions for each term is provided below:

(a) Direct elimination: A method to solve a system of equations that uses row operations to perform forward elimination in order to solve the system using backwards substitution.

(b) *LU* Factorization: Solving method that factors the initial matrix into lower and upper matrices in order to simplify the solving the system into forward and backwards substitutions.

(c) Inverse matrix: A matrix such that any square matrix multiplied by its inverse gives the identity matrix.

(d) Symmetric matrix: Any matrix that if you switch the rows and the columns you do not change the matrix.

(e) Transpose: A matrix operation that flips a matrix along its diagonal, i.e. switches its rows and columns.

(f) Permutation: Matrix operations that change the rows of an another matrix by multiplying it by a set of permutation matrices which are composed of the rows of the identity matrix.

(g) Inner product: A matrix operation that gives the projection of one matrix onto another.

(h) Singular matrix: A square matrix without an inverse.

2 Exercise 2

Solve the following system of equations, showing all relevant steps and stating the method used to solve the system (e.g., LU factorization, direct numerical solution, matrix elimination, etc.,). Furthermore, if a system has no solution state it has no solution and briefly describe why it has no solution.

(a) The following system of equations can be solved using the inverse matrix approach, namely

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{1a}$$

therefore we need to obtain A^{-1}. We can determine the inverse of matrix $\mathbf{A}$ using the augmented matrix method. Where the matrix $\mathbf{A}$ is augmented with the identity matrix $\mathbf{I}$, and row operations are performed until the left hand side of the augmented matrix is the identity matrix. Starting with the augmented matrix

$$\left[\begin{array}{ccc|ccc} 1 & 3 & 1 & 1 & 0 & 0 \\ 4 & 0 & 2 & 0 & 1 & 0 \\ 1 & 3 & 4 & 0 & 0 & 1 \end{array}\right] \tag{1b}$$

Then we can take $4r_1 - r_2$ to zero out the a_{21} term giving

$$\left[\begin{array}{ccc|ccc} 1 & 3 & 1 & 1 & 0 & 0 \\ 0 & 12 & 2 & 4 & -1 & 0 \\ 1 & 3 & 4 & 0 & 0 & 1 \end{array}\right] \tag{1c}$$

Next, the a_{31} is zeroed out by performing $r_1 - r_3$ giving

$$\left[\begin{array}{ccc|ccc} 1 & 3 & 1 & 1 & 0 & 0 \\ 0 & 12 & 2 & 4 & -1 & 0 \\ 0 & 0 & -3 & 1 & 0 & -1 \end{array}\right] \tag{1d}$$

Now we continue zeroing out entries of $\mathbf{A}$ moving up the matrix starting with a_{23} by performing $-\frac{2}{3}r_3 - r_2$ giving

$$\left[\begin{array}{ccc|ccc} 1 & 3 & 1 & 1 & 0 & 0 \\ 0 & -12 & 0 & -14/3 & 1 & 2/3 \\ 0 & 0 & -3 & 1 & 0 & -1 \end{array}\right] \tag{1e}$$

Moving up the third column we zero out a_{13} using $-\frac{1}{3}r_3 - r_1$ giving

$$\left[\begin{array}{ccc|ccc} -1 & -3 & 0 & -1/3 & 0 & -1/3 \\ 0 & -12 & 0 & -14/3 & 1 & 2/3 \\ 0 & 0 & -3 & 1 & 0 & -1 \end{array}\right] \tag{1f}$$

Next we can zero a_{12} with $\frac{1}{4}r_2 - r_1$ giving

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/6 & 1/4 & -1/6 \\ 0 & -12 & 0 & -14/3 & 1 & 2/3 \\ 0 & 0 & -3 & 1 & 0 & -1 \end{array}\right] \tag{1g}$$

Lastly we can multiply each pivot by its reciprocal, i.e., $r_2 = -\frac{r_2}{12}$ and $r_3 = -\frac{r_3}{3}$, to finally get the inverse of matrix $\mathbf{A}$,

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/6 & 1/4 & -1/6 \\ 0 & 1 & 0 & 7/18 & -1/12 & -1/18 \\ 0 & 0 & 1 & -1/3 & 0 & 1/3 \end{array}\right] \tag{1h}$$

Thus

$$\boxed{\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} 1/6 & 1/4 & -1/6 \\ 7/18 & -1/12 & -1/18 \\ -1/3 & 0 & 1/3 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 11/6 \\ 23/18 \\ -2/3 \end{bmatrix}} \tag{1i}$$

(b) From the system of equations

$$\begin{bmatrix} 1 & 4 & 2 \\ 2 & 8 & 4 \\ 1 & 5 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \tag{2}$$

it is obvious that the second row is a linear combination of the first row, i.e. $r_2 = 2r_1$. Therefore the second equation in the system adds no new information, that we do not already have from the first row. Thus, we have a system with three unknowns, x, y, z, and only two equations, therefore the system does not have an unique solution.

(c) The simplest way to solve the following system

$$\begin{bmatrix} 1 & 3 & 1 \\ 4 & 1 & 2 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \tag{3a}$$

is to use to use the first row to cancel out the first entry of the second row, i.e. $4r_1 - r_2$, giving

$$\begin{bmatrix} 1 & 3 & 1 \\ 0 & 11 & 2 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \tag{3b}$$

and them backwards substituting. Thus,

$$\boxed{\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 15/44 \\ 3/22 \\ 1/4 \end{bmatrix}} \tag{3c}$$

(d) The easiest way to solve this system of equations

$$\begin{bmatrix} 0 & 1 & 2 & 0 & 0 \\ 5 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \tag{4a}$$

is to swap the rows so that all the pivots are non-zero giving

$$\begin{bmatrix} 5 & 0 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 5 \\ 3 \\ 4 \end{bmatrix} \tag{4b}$$

Now the system of equations can be solved quite easily using backwards substitution. Thus,

$$\boxed{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 16/5 \\ 15 \\ -7 \\ -1 \\ 4 \end{bmatrix}} \tag{5}$$

3 Exercise 3

The following system of equations

$$2x + 3y = 4 \tag{6a}$$

$$x + 2y = 5 \tag{6b}$$

can easily be solve by hand. First solve for x in terms of y in Eq. (6b),

$$x = 5 - 2y \tag{6c}$$

Next substitute x into Eq. (6a) to obtain a value for y, namely

$$2(-5 - 2y) + 3y = 4 \rightarrow y = 6 \tag{6d}$$

Thus,

$$\boxed{x = -7} \tag{6e}$$

$$\boxed{y = 6} \tag{6f}$$

Furthermore, the solution can be verified by solving for y in both equations and plotting both curves and observing their intersection point, as shown in Fig (1).

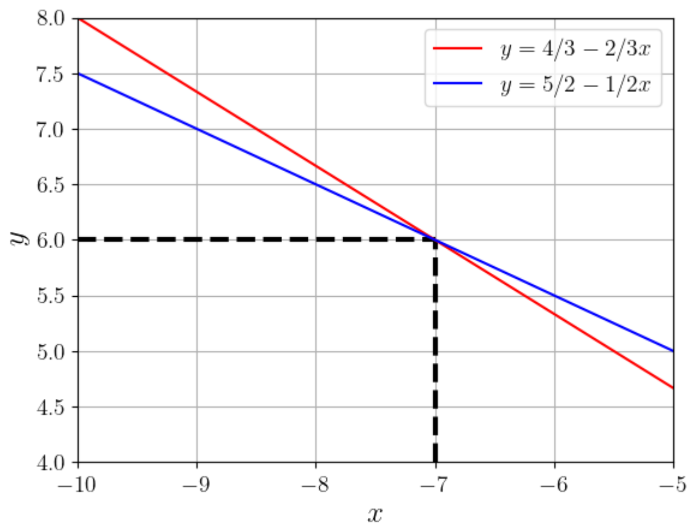

Figure 1: Verification plot for Exercise 3

Please see the Appendix for the full version of the verification code.

4 Exercise 4

Prove the following matrix properties:

(a) Prove the following:

$$(AB)^T = B^T A^T \tag{7a}$$

Start be defining two $N \times N$ matrices **A** and **B**,

$$\mathbf{A} = A_{ij} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n_2} & \cdots & a_{n,n} \end{bmatrix} \tag{7b}$$

$$\mathbf{B} = B_{ij} = \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix} \tag{7c}$$

which gives the following for the LHS of Eq. (7a)

$$AB = \underbrace{\sum_{k=1}^{N} A_{ik} B_{kj}}_{\equiv C_{ij}} = \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{1,1}b_{1,2} + a_{1,2}b_{2,2} & \cdots \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} & \cdots \\ \vdots & \ddots & \vdots \end{bmatrix} \tag{7d}$$

Therefore,

$$C^T = \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & \cdots \\ a_{1,1}b_{1,2} + a_{1,2}b_{2,2} & a_{2,1}b_{1,2} + a_{2,2}b_{2,2} & \cdots \\ \vdots & \ddots & \vdots \end{bmatrix} \tag{7e}$$

Next we can look at the RHS of Eq. (7a) where,

$$A^T = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{n,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{n,n} \end{bmatrix} \tag{7f}$$

$$B^T = \begin{bmatrix} b_{1,1} & b_{2,1} & \cdots & b_{n,1} \\ b_{1,2} & b_{2,2} & \cdots & b_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,n} & b_{2,n} & \cdots & b_{n,n} \end{bmatrix} \tag{7g}$$

Therefore,

$$B^T A^T = \begin{bmatrix} b_{1,1}a_{1,1} + b_{2,1}a_{1,2} & b_{1,1}a_{2,1} + b_{2,1}a_{2,2} & \cdots \\ b_{1,2}a_{1,1} + b_{2,2}a_{1,2} & b_{1,2}a_{2,1} + b_{2,2}a_{2,2} & \cdots \\ \vdots & \ddots & \vdots \end{bmatrix} \tag{7h}$$

which is equivalent to Eq. (7e).

(b) To prove the following

$$\left(A^{-1}\right)^T = \left(A^T\right)^{-1} \tag{8a}$$

start with

$$A^{-1}A = I \tag{8b}$$

where from part(a) we know that

$$\left(A^{-1}A\right)^T = A^T\left(A^{-1}\right)^T = I^T \tag{8c}$$

and since

$$I^T = I \tag{8d}$$

then

$$A^{-1}A = A^T\left(A^{-1}\right)^T = I \tag{8e}$$

Thus,

$$\boxed{\left(A^{-1}\right)^T = \left(A^T\right)^{-1}} \tag{8f}$$

(c) Lets consider the following symmetric matrix

$$C = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & \cdots & c_{1,n} \\ c_{1,2} & c_{2,2} & c_{2,3} & \cdots & c_{2,n} \\ c_{1,3} & c_{2,3} & c_{3,3} & \cdots & c_{3,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{1,n} & c_{2,n} & c_{3,n} & \cdots & c_{n,n} \end{bmatrix} \tag{9a}$$

where for in index notation $C_{ij} = C_{ji}$. Therefore,

$$\boxed{C^T = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & \cdots & c_{1,n} \\ c_{1,2} & c_{2,2} & c_{2,3} & \cdots & c_{2,n} \\ c_{1,3} & c_{2,3} & c_{3,3} & \cdots & c_{3,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{1,n} & c_{2,n} & c_{3,n} & \cdots & c_{n,n} \end{bmatrix} = C} \tag{9b}$$

5 Exercise 5

(a)

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 5 & 4 \\ 2 & 4 & 6 \end{bmatrix} \tag{10a}$$

As covered in lecture we know that A can be factored into a product of lower and upper matrices, namely

$$A = LU = \begin{bmatrix} 1 & 0 & 0 \\ l_{2,1} & 1 & 0 \\ l_{3,1} & l_{3,2} & 1 \end{bmatrix} \begin{bmatrix} u_{1,2} & u_{1,2} & u_{1,2} \\ 0 & u_{2,2} & u_{2,2} \\ 0 & 0 & u_{3,2} \end{bmatrix} \tag{10b}$$

where the non-zero off diagonal entries of L are the multipliers used in forward elimination, and the entries of U are the values after all the row operations are performed. Therefore, the easiest way to perform LU decomposition is to set $U = A$ and perform forward elimination. Starting with the first row, since $r_2 - 3r_1$ zeros out $a_{2,1}$, we know that $l_{2,1} = 3$, and that the updated U matrix is

$$U = \begin{bmatrix} 1 & 3 & 2 \\ 0 & -4 & -2 \\ 2 & 4 & 6 \end{bmatrix} \tag{10c}$$

To cancel out the third row we need $r_3 - 2r_1$, therefore $l_{31} = 2$ and the updated U is

$$U = \begin{bmatrix} 1 & 3 & 2 \\ 0 & -4 & -2 \\ 0 & -2 & 2 \end{bmatrix} \tag{10d}$$

Lastly, $r_3 - \frac{1}{2}r_2$ would zero out the last entry, therefore $l_{32} = 1/2$, and

$$U = \begin{bmatrix} 1 & 3 & 2 \\ 0 & -4 & -2 \\ 0 & 0 & 3 \end{bmatrix} \tag{10e}$$

Thus

$$\boxed{A = LU = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 3 & 2 \\ 0 & -4 & -2 \\ 0 & 0 & 3 \end{bmatrix}} \tag{10f}$$

Furthermore, we can obtain the $A = LDU$ factorization, by dividing U by a diagonal matrix D that contains the pivots, namely

$$\boxed{A = LDU = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 2 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 3 & 2 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}} \tag{10g}$$

(b)

$$B = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 2 & 2 \\ 3 & 4 & 5 \end{bmatrix} \tag{11a}$$

Using the same methodology as part(a), we use $r_2 - 2r_1$ which gives $l_{2,1} = 2$ and

$$U = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 3 & 4 & 5 \end{bmatrix} \tag{11b}$$

Next, we need $r_3 - 3r_1$, therefore $l_{3,1} = 3$ and

$$U = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 4 & 2 \end{bmatrix} \tag{11c}$$

Lastly, we need $r_3 - 2r_2$ giving $l_{3,2} = 2$ and

$$U = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{11d}$$

Thus,

$$\boxed{B = LU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}} \tag{11e}$$

and

$$\boxed{B = LDU = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}} \tag{11f}$$

(c)

$$\begin{bmatrix} 2 & 1 & 3 \\ 3 & 2 & 3 \\ 3 & 1 & 0 \end{bmatrix} \tag{12a}$$

First, since there is a zero pivot in the third row we know were going to need a permutation matrix. A good chose for this problem would be to make the following row switches,

$$r_1 \to r_3 \tag{12b}$$

$$r_2 \to r_1 \tag{12c}$$

$$r_1 \to r_3 \tag{12d}$$

Therefore we get

$$PA = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 3 \\ 3 & 2 & 3 \\ 3 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 3 \\ 3 & 1 & 0 \\ 2 & 1 & 3 \end{bmatrix} \tag{12e}$$

Now we can apply the same methodology as above, starting with $r_2 - r_1$, therefore $l_{2,1} = 1$ and

$$U = \begin{bmatrix} 3 & 2 & 3 \\ 0 & -1 & -3 \\ 2 & 1 & 3 \end{bmatrix} \tag{12f}$$

Next, $r_3 - \frac{2}{3}r_1$, therefore $l_{3,1} = \frac{2}{3}$ and

$$U = \begin{bmatrix} 3 & 2 & 3 \\ 0 & -1 & -3 \\ 0 & -1/3 & 1 \end{bmatrix} \tag{12g}$$

Lastly, $r_3 - \frac{1}{3}r_2$, therefore $l_{3,2} = \frac{1}{3}$ and

$$U = \begin{bmatrix} 3 & 2 & 3 \\ 0 & -1 & -3 \\ 0 & 0 & 2 \end{bmatrix} \tag{12h}$$

Thus,

$$\boxed{PA = LU = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2/3 & 1/3 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 & 3 \\ 0 & -1 & -3 \\ 0 & 0 & 2 \end{bmatrix}} \tag{12i}$$

and

$$\boxed{PA = LDU = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2/3 & 1/3 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2/3 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}} \tag{12j}$$

6 Exercise 6

The following algorithm was used to perform the LU factorization

Algorithm 1: LU Factorization

Result: Factoring matrix A into L and U
Input : A
Output: L, U

```
L = I(3)
U = A
for k in range(0, M-1) do
    for j in range(k+1, M) do
        L[J,K] = U[J,K]/U[K,K]
        U[J,K:] = U[J,K:]-L[J,K]*U[K,K:]
        U[J,K] = 0.0
    end
end
```

Secondly, we can plot the number of points versus computational time, as seen in Fig .(2), where we see how important it is to use built in Python packages.

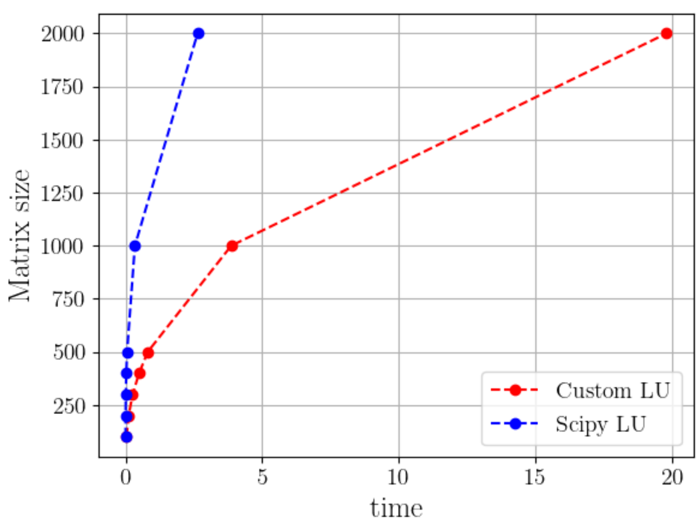

Figure 2: Performance study of user defined LU factorization tool

For an example of the complete code please refer to the Appendix.

7 Exercise 7

The following inverses were calculated using the augmented matrix approach.

(a)

$$A = \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 2 & 1 & 3 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{13a}$$

First we can use $2r_1 - r_2$ to cancel out the entries in the first column, giving

$$A = \left[\begin{array}{ccc|ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & -1 & 2 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{13b}$$

Next we can zero out the entries in the third column using $r_3 + r_2$ and $r_3 - r_1$ giving

$$A = \left[\begin{array}{ccc|ccc} -1 & 0 & 0 & -1 & 0 & 1 \\ 0 & -1 & 0 & 2 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{13c}$$

Thus,

$$\boxed{A^{-1} = \begin{bmatrix} 1 & 0 & -1 \\ -2 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}} \tag{13d}$$

(b)

$$B = \left[\begin{array}{ccc|ccc} 2 & 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 & 1 & 0 \\ 1 & 1 & 2 & 0 & 0 & 1 \end{array}\right] \tag{14a}$$

First we can use $r_2 - \frac{1}{2}r_1$ and $r_3 - \frac{1}{2}r_1$ to cancel out the first in the first column, giving

$$B = \left[\begin{array}{ccc|ccc} 2 & 1 & 1 & 1 & 0 & 0 \\ 0 & 3/2 & 1/2 & -1/2 & 1 & 0 \\ 0 & 1/2 & 3/2 & -1/2 & 0 & 1 \end{array}\right] \tag{14b}$$

Next, we can use $r_3 - \frac{1}{3}r_2$ to cancel out the entries of the second column beneath the pivot, giving

$$B = \left[\begin{array}{ccc|ccc} 2 & 1 & 1 & 1 & 0 & 0 \\ 0 & 3/2 & 1/2 & -1/2 & 1 & 0 \\ 0 & 0 & 4/3 & -1/3 & -1/3 & 1 \end{array}\right] \tag{14c}$$

Next, we can use $r_2 - \frac{3}{8}r_3$ and $r_1 - \frac{3}{4}r_3$ to cancel out the entries of the third column above the pivot, giving

$$B = \left[\begin{array}{ccc|ccc} 2 & 1 & 0 & 5/4 & 1/4 & -3/4 \\ 0 & 3/2 & 0 & -3/8 & 9/8 & -3/8 \\ 0 & 0 & 4/3 & -1/3 & -1/3 & 1 \end{array}\right] \tag{14d}$$

Lastly, we use $r_1 - \frac{2}{3}r_2$ to zero out the entries above the pivot in the second column, giving

$$B = \left[\begin{array}{ccc|ccc} 2 & 0 & 0 & 3/2 & -1/2 & -1/2 \\ 0 & 3/2 & 0 & -3/8 & 9/8 & -3/8 \\ 0 & 0 & 4/3 & -1/3 & -1/3 & 1 \end{array}\right] \tag{14e}$$

Thus,

$$\boxed{B^{-1} = \begin{bmatrix} 3/4 & -1/4 & -1/4 \\ -1/4 & 3/4 & -1/4 \\ -1/4 & -1/4 & 3/4 \end{bmatrix}} \tag{14f}$$

(c)

$$C = \left[\begin{array}{ccc|ccc} 2 & -1 & -1 & 1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 1 & 0 \\ -1 & -1 & 2 & 0 & 0 & 1 \end{array}\right] \tag{15a}$$

Starting with $r_2 + \frac{1}{2}r_1$ and $r_3 + \frac{1}{2}r_1$ to cancel the entries beneath the pivot in the first column, giving

$$C = \left[\begin{array}{ccc|ccc} 2 & -1 & -1 & 1 & 0 & 0 \\ 0 & 3/2 & -3/2 & 1/2 & 1 & 0 \\ 0 & -3/2 & 3/2 & 1/2 & 0 & 1 \end{array}\right] \tag{15b}$$

Next, we see that $r_2 + r_3$ results in

$$C = \begin{bmatrix} 2 & -1 & -1 \\ 0 & 3/2 & -3/2 \\ 0 & 0 & 0 \end{bmatrix} \tag{15c}$$

Thus, C is a singular matrix and has no inverse. Furthermore, there are many different ways to show a matrix is singular, therefore any solutions that rigorously proves that C is not invertible would be considered correct.

8 Exercise 8

The inverse for each of the grids were calculated using the LU factorials of matrix, namely

$$A^{-1} = U^{-1}L^{-1} \tag{16}$$

where the factorials were found using the subroutine discussed in Exercise 6. The following logic was used to find the inverse of matrix A:

Algorithm 2: A^{-1} Subroutine

Result: Calculating A^{-1}
Input : A
Output: A^{-1}

$[L, U] = \text{LU\_facrorization}(A)$
$Linv = lower\_inverse(L)$
$Uinv = upper\_inverse(L)$
$Ainv = Uinv \times Linv$

Where the inverse subroutines use the following logic:

Algorithm 3: L^{-1} Subroutine

Result: Lower triangular inverse
Input : L
Output: L^{-1}

```
out = identity(M)
for j in range(0, M-1) do
    for i in range(j,M-1) do
        c = mat[i+1,j]
        out[i+1,:] = out[i+1,:]-c*out[j,:]
    end
end
```

Algorithm 4: U^{-1} Subroutine

Result: Upper triangular inverse
Input : U
Output: U^{-1}

```
out = identity(M)
for j in range(M-1, -1, -1) do
    for i in range(j, 0, -1) do
        c = mat[i-1,j]/mat[j,j]
        out[i-1,:] = out[i-1,:]-c*out[j,:]
    end
    out[j,:] = out[j,:]/mat[j,j]
end
```

Using the above we get the following result shown in Fig. (3), where we can see once again the benefit in using the pre-compiled optimized code. Lastly, the full inverse code is provided in the Appendix.

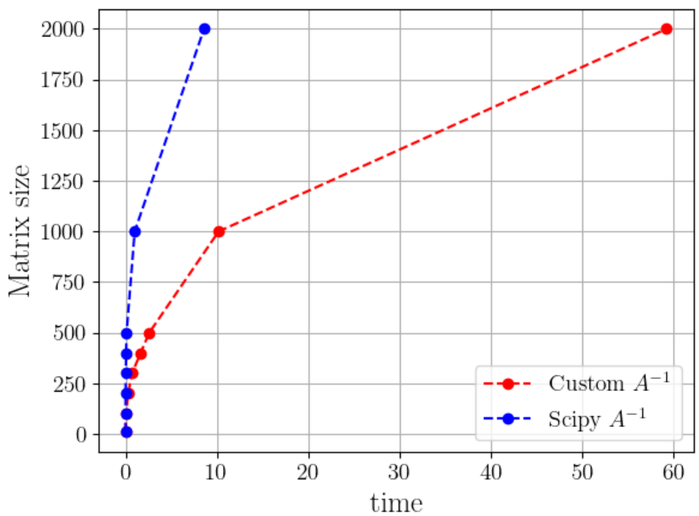

Figure 3: Performance study of user defined A^{-1} tool

9 Appendix

9.1 Exercise 3 verification code

```python
#!/usr/bin/env python3
"""========================================================================
Purpose:
    Verification of exercise 3

Author:
    Emilio Torres
========================================================================"""
#=========================================================================#
# Preamble                                                                #
#=========================================================================#
#-------------------------------------------------------------------------#
# Python packages                                                         #
#-------------------------------------------------------------------------#
import sys
import os
from subprocess import call
from numpy import *
import matplotlib.pyplot as plt
#-------------------------------------------------------------------------#
# User packages                                                           #
#-------------------------------------------------------------------------#
from ales_post.plot_settings import plot_setting
#=========================================================================#
# Main preamble                                                           #
#=========================================================================#
if __name__ == '__main__':
    #---------------------------------------------------------------------#
    # Main preamble                                                       #
    #---------------------------------------------------------------------#
    call(['clear'])
    sep          = os.sep
    pwd          = os.getcwd()
    media_path   = pwd + '%c..%cmedia%c'         %(sep, sep, sep)
    #---------------------------------------------------------------------#
    # Domain variables                                                    #
    #---------------------------------------------------------------------#
    x   = linspace(-10, -5, 100)
    y1  = 4./3. - 2./3*x
    y2  = 5./2.-x/2.
    #---------------------------------------------------------------------#
    # Plotting solution                                                   #
    #---------------------------------------------------------------------#
    plot_setting()
    plt.plot([-10,-7], [6,6], 'k--', lw = 3.0)
    plt.plot([-7,-7], [4,6], 'k--', lw = 3.0)
    plt.plot(x,y1,'r', lw=1.5, label = '$y = 4/3 - 2/3 x$')
    plt.plot(x,y2,'b', lw=1.5, label = '$y = 5/2 - 1/2 x$')
    #---------------------------------------------------------------------#
```

```
    # Plot settings                                                      #
    #--------------------------------------------------------------------#
    plt.legend(loc=0)
    plt.ylabel('$y$')
    plt.xlabel('$x$')
    plt.grid()
    plt.xlim([-10,-5])
    plt.ylim([4,8])
    plt.savefig(media_path + 'exercise-3.png')
    plt.close()

    print('**** Successful run ****')
    sys.exit(0)
```

9.2 Exercise 6 *LU* factorization

```
#!/usr/bin/env python3
"""==========================================================================
Purpose:
    The purpose of this script is to build subroutines to perform the
    following:
        1. LU factorization

    **** Note:
            This is pseudo code to help with Assignment 1

Author:
    Emilio Torres
=========================================================================="""
#==========================================================================#
# Preamble                                                                 #
#==========================================================================#
#--------------------------------------------------------------------------#
# Python packages                                                          #
#--------------------------------------------------------------------------#
import os
import sys
from subprocess import call
import time
from numpy import copy, identity, random, zeros, array
import scipy.linalg as la
import matplotlib.pyplot as plt
#==========================================================================#
# User defined functions                                                   #
#==========================================================================#
#--------------------------------------------------------------------------#
# Pretty print matrix                                                      #
#--------------------------------------------------------------------------#
def print_matrix(
        mat,                    # input matrix
        var_str):

    """ Pretty printing a matrix """
    #----------------------------------------------------------------------#
```

```
    # Looping over columns and rows                                        #
    #----------------------------------------------------------------------#
    out = ''                    # initialize string
    for I in range(0, mat.shape[0]):
        for J in range(0, mat.shape[1]):
            out += '%12.5f'          %(mat[I,J])
        out += '\n'
    print(var_str)
    print(out)
#----------------------------------------------------------------------#
# Plotting settings                                                    #
#----------------------------------------------------------------------#
def plot_setting():

    """ Useful plotting settings """
    #----------------------------------------------------------------------#
    # Plotting settings                                                    #
    #----------------------------------------------------------------------#
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    SMALL_SIZE = 14
    MEDIUM_SIZE = 18
    BIGGER_SIZE = 20
    plt.rc('font',      size=SMALL_SIZE)
    plt.rc('axes',      titlesize=SMALL_SIZE)
    plt.rc('axes',      labelsize=MEDIUM_SIZE)
    plt.rc('xtick',     labelsize=SMALL_SIZE)
    plt.rc('ytick',     labelsize=SMALL_SIZE)
    plt.rc('legend',    fontsize=SMALL_SIZE)
    plt.rc('figure',    titlesize=BIGGER_SIZE)
#----------------------------------------------------------------------#
# LU factorization                                                     #
#----------------------------------------------------------------------#
def LU_factorization(
        mat):                   # input matrix

    """ Calculating the LU factorization of the input vector """
    #----------------------------------------------------------------------#
    # Preallocating matrices                                               #
    #----------------------------------------------------------------------#
    M   = mat.shape[0]          # matrix size
    U   = copy(mat)             # make sure you copy matrix  (No!!! U = mat )
    L   = identity(M)           # Initialize L with the identity matrix
    #----------------------------------------------------------------------#
    # Cheking it is error matrix                                           #
    #----------------------------------------------------------------------#
    if not mat.shape[0] ==  mat.shape[1]:
        print('Input matrix must be square')
        print('Input --> [%i, %i]'           %(mat.shape[0], mat.shape[1]))
        sys.exit(8)
    #----------------------------------------------------------------------#
    # Calculating both L and U                                             #
    #----------------------------------------------------------------------#
    for K in range(0, M-1):
        for J in range(K+1, M):
```

```
            L[J,K]      = U[J,K]/U[K,K]
            U[J,K:]     = U[J,K:]-L[J,K]*U[K,K:]
            U[J,K]      = 0.0

    return (L, U)
#==========================================================================#
# Main                                                                     #
#==========================================================================#
if __name__ == '__main__':
    #----------------------------------------------------------------------#
    # Main preamble                                                        #
    #----------------------------------------------------------------------#
    call(['clear'])
    sep         = os.sep
    pwd         = os.getcwd()
    media_path  = pwd + '%c..%cmedia%c'             %(sep, sep, sep)
    #----------------------------------------------------------------------#
    # Testing LU                                                           #
    #----------------------------------------------------------------------#
    A               = array([[3, 2, 3], [3, 1, 0], [2, 1, 3]])
    (Lower,Upper)   = LU_factorization(A)
    print_matrix(Lower, 'L')
    print_matrix(Upper, 'U')
    print_matrix(A - (Lower@Upper), 'A-LU')
    sys.exit(8)
    #----------------------------------------------------------------------#
    # Time study                                                           #
    #----------------------------------------------------------------------#
    N       = [100, 200, 300, 400, 500, 1000, 2000]
    times   = zeros(len(N))
    times2  = zeros(len(N))
    for k, i in enumerate(N):
        A               = random.rand(i,i)          # random matrix
        tic             = time.time()               # start time
        (Lower,Upper)   = LU_factorization(A)       # LU
        toc             = time.time()               # end time
        times[k]        = toc-tic                   # time elapsed
        tic             = time.time()
        (p, l, u)       = la.lu(A)
        toc             = time.time()
        times2[k]       = toc-tic
    #----------------------------------------------------------------------#
    # Plotting the solutions                                               #
    #----------------------------------------------------------------------#
    plot_setting()
    plt.plot(times, N, 'ro--', lw=1.5, label='Custom LU')
    plt.plot(times2, N, 'bo--', lw=1.5, label='Scipy LU')
    #----------------------------------------------------------------------#
    # Plot settings                                                        #
    #----------------------------------------------------------------------#
    plt.ylabel('Matrix size')
    plt.xlabel('time')
    plt.grid(True)
    plt.legend(loc=0)
    plt.savefig(media_path + 'exercise-6.png')
```

```
    plt.show()
    plt.close()

    print('**** Successful run ****')
    sys.exit(0)
```

9.3 Exercise 8 Inverse

```
#!/usr/bin/env python3
"""===========================================================================
Purpose:
    The purpose pf this script is to build subroutines to perform the
    following:
        1. Matrix inverse
Author:
    Emilio Torres
==========================================================================="""
#=========================================================================#
# Preamble                                                                #
#=========================================================================#
#-------------------------------------------------------------------------#
# Python packages                                                         #
#-------------------------------------------------------------------------#
import os
import sys
from subprocess import call
import time
from numpy import copy, identity, random, zeros, array
import scipy.linalg as la
import matplotlib.pyplot as plt
#=========================================================================#
# User defined functions                                                  #
#=========================================================================#
#-------------------------------------------------------------------------#
# Pretty print matrix                                                     #
#-------------------------------------------------------------------------#
def print_matrix(
        mat,                    # input matrix
        var_str):

    """ Pretty printing a matrix """
    #---------------------------------------------------------------------#
    # Looping over columns and rows                                       #
    #---------------------------------------------------------------------#
    out = ''                    # initialize string
    for I in range(0, mat.shape[0]):
        for J in range(0, mat.shape[1]):
            out += '%25.5f'          %(mat[I,J])
        out += '\n'
    print(var_str)
    print(out)
#-------------------------------------------------------------------------#
# Plotting settings                                                       #
#-------------------------------------------------------------------------#
```

```python
def plot_setting():

    """ Useful plotting settings """
    #-------------------------------------------------------------------------#
    # Plotting settings                                                       #
    #-------------------------------------------------------------------------#
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    SMALL_SIZE = 14
    MEDIUM_SIZE = 18
    BIGGER_SIZE = 20
    plt.rc('font',      size=SMALL_SIZE)
    plt.rc('axes',      titlesize=SMALL_SIZE)
    plt.rc('axes',      labelsize=MEDIUM_SIZE)
    plt.rc('xtick',     labelsize=SMALL_SIZE)
    plt.rc('ytick',     labelsize=SMALL_SIZE)
    plt.rc('legend',    fontsize=SMALL_SIZE)
    plt.rc('figure',    titlesize=BIGGER_SIZE)
#-----------------------------------------------------------------------------#
# LU factorization                                                            #
#-----------------------------------------------------------------------------#
def LU_factorization(
        mat):                   # inpuit matrix

    """ Calculating the LU factorization of the input vector """
    #-------------------------------------------------------------------------#
    # Preallocating matrices                                                  #
    #-------------------------------------------------------------------------#
    M   = mat.shape[0]          # matrix size
    U   = copy(mat)             # make sure you copy matrix  (No!!! U = mat )
    L   = identity(M)           # Initialize L with the identity matrix
    #-------------------------------------------------------------------------#
    # Cheking it is error matrix                                              #
    #-------------------------------------------------------------------------#
    if not mat.shape[0] ==  mat.shape[1]:
        print('Input matrix must be square')
        print('Input --> [%i, %i]'          %(mat.shape[0], mat.shape[1]))
        sys.exit(8)
    #-------------------------------------------------------------------------#
    # Calculating both L and U                                                #
    #-------------------------------------------------------------------------#
    for K in range(0, M-1):
        for J in range(K+1, M):
            L[J,K]      = U[J,K]/U[K,K]
            U[J,K:]     = U[J,K:]-L[J,K]*U[K,K:]
            U[J,K]      = 0.0

    return L, U
#-----------------------------------------------------------------------------#
# Diagonal inverse tool                                                       #
#-----------------------------------------------------------------------------#
def diagonal_inverse(
        mat):

    """ Subroutine to calculate the inverse of a diagonal matrix """
```

```
    #------------------------------------------------------------------------#
    # Domain variables                                                       #
    #------------------------------------------------------------------------#
    M   = mat.shape[0]
    out = idnetity(M)
    #------------------------------------------------------------------------#
    # Diagonal inverse                                                      #
    #------------------------------------------------------------------------#
    for i in range(0,M):
        out[i,i]    = 1/mat[i,i]

    return out
#----------------------------------------------------------------------------#
# Upper matrix inverse                                                       #
#----------------------------------------------------------------------------#
def upper_inverse(
        mat):

    """ Subroutine to calculate the inverse of an upper diagonal matrix """
    #------------------------------------------------------------------------#
    # Domain variables                                                       #
    #------------------------------------------------------------------------#
    M   = mat.shape[0]
    out = identity(M)
    #------------------------------------------------------------------------#
    # Upper inverse                                                          #
    #------------------------------------------------------------------------#
    for j in range(M-1, -1, -1):
        for i in range(j-1, -1, -1):
            c           = mat[i,j]/mat[j,j]
            out[i,:]    =  out[i,:]-c*out[j,:]
        out[j,:] = out[j,:]/mat[j,j]
    return out
#----------------------------------------------------------------------------#
# Upper matrix inverse                                                       #
#----------------------------------------------------------------------------#
def lower_inverse(
        mat):

    """ Subroutine to calculate the inverse of a lower diagonal matrix """
    #------------------------------------------------------------------------#
    # Domain variables                                                       #
    #------------------------------------------------------------------------#
    M   = mat.shape[0]
    out = identity(M)
    #------------------------------------------------------------------------#
    # Lower inverse                                                          #
    #------------------------------------------------------------------------#
    for j in range(0, M-1):
        for i in range(j,M-1):
            c           = mat[i+1,j]
            out[i+1,:]  = out[i+1,:]-c*out[j,:]

    return out
#----------------------------------------------------------------------------#
```

```
# Inverse tool                                                              #
#---------------------------------------------------------------------------#
def mat_inverse(
        mat):
    """ Subroutine to determine a matrix inverse """
    #-----------------------------------------------------------------------#
    # Calculating the inverse using LU factors                              #
    #-----------------------------------------------------------------------#
    N       = mat.shape[0]
    [L,U]   = LU_factorization(mat)
    #-----------------------------------------------------------------------#
    # Calculating the inverse                                               #
    #-----------------------------------------------------------------------#
    Linv    = lower_inverse(L)
    Uinv    = upper_inverse(U)
    inv     = Uinv@Linv

    return inv
#===========================================================================#
# Main                                                                      #
#===========================================================================#
if __name__ == '__main__':
    #-----------------------------------------------------------------------#
    # Main preamble                                                         #
    #-----------------------------------------------------------------------#
    call(['clear'])
    sep         = os.sep
    pwd         = os.getcwd()
    media_path  = pwd  + '%c..%cmedia%c'            %(sep, sep, sep)
    #-----------------------------------------------------------------------#
    # Testing LU                                                            #
    #-----------------------------------------------------------------------#
    N                   = 5
    A                   = random.rand(N,N)
    Ainv                = mat_inverse(A)
    I                   = identity(N)
    print_matrix(A, 'A')
    print_matrix(Ainv, 'A^{-1}')
    print_matrix(I-A@Ainv, 'I-AxA^{-1}')
    #-----------------------------------------------------------------------#
    # Time study                                                            #
    #-----------------------------------------------------------------------#
    N       = [10, 100, 200, 300, 400, 500, 1000, 2000]
    times   = zeros(len(N))
    times2  = zeros(len(N))
    for k, i in enumerate(N):
        print(i)
        A               = random.rand(i,i)          # random matrix
        tic             = time.time()               # start time
        Ainv            = mat_inverse(A)            # inv(A)
        toc             = time.time()               # end time
        times[k]        = toc-tic                   # time elapsed
        tic             = time.time()
        ainv            = la.inv(A)
        toc             = time.time()
```

```
        times2[k]       = toc-tic
    #-----------------------------------------------------------------------#
    # Plotting the solutions                                                 #
    #-----------------------------------------------------------------------#
    plot_setting()
    plt.plot(times, N, 'ro--', lw=1.5, label='Custom $A^{-1}$')
    plt.plot(times2, N, 'bo--', lw=1.5, label='Scipy $A^{-1}$')
    #-----------------------------------------------------------------------#
    # Plot settings                                                          #
    #-----------------------------------------------------------------------#
    plt.ylabel('Matrix size')
    plt.xlabel('time')
    plt.grid(True)
    plt.legend(loc=0)
    plt.savefig(media_path + 'exercise-8.png')
    plt.close()

    print('**** Successful run ****')
    sys.exit(0)
```