Assignment 1
MATH205: Linear Algebra

October 3, 2019

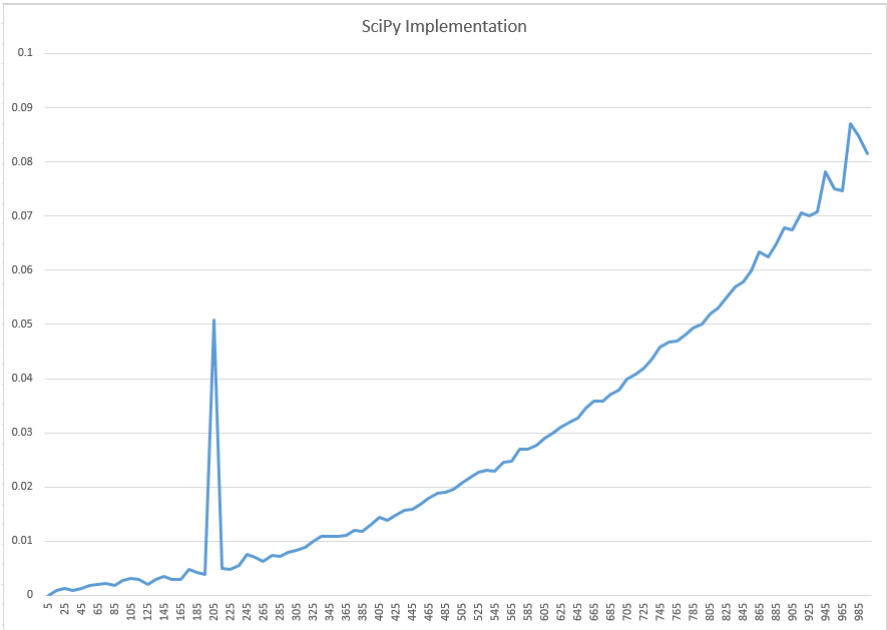
1 Group Members

1. Muhammad Shahrom Ali
2. Fizza Abbas
3. Muhammad Shahzain

2 Questions

1. Question 1

Runtime of SelfImplemented Gaussian Elimination.



Runtime of SciPy Gaussian Elimination.

(Runtime on the vertical axes, matrix dimemsions on the horizontal axes)

As evident, the library implementation is much faster than our implementation of Gaussian Elimination. It takes less than 1 second

for decomposition of a 985-matrix, compared to 58 seconds for our implementation.

The curves are increasing exponentially though one thing that can be seen is that our curve increases much more smoothly as compared to the curve of scipy.

For Scipy, 205 seems to be an anomalous point in the running of the algorithm (perhaps due to the nature of the matrix?). We can also observe a few peaks near the end, and some more jerks throughout the curve which may be due to the heavy optimization in the library; there are some jerks and anomalies but the overall runtime is low.

For our implementaion, we can see that the runtime is sufficiently low for as large as 335-matrices (2.075451612 seconds) but, as observed earlier, runtime appears to be increasing exponentially and later the runtime increases by as much as 2 seconds (which is the total time for solving a 335-matrix) when increasing the dimensions of a matrix by 10.

Another important thing we can deduce is that the basic implementation of Gaussian elimination is good enough for small data sets of size 250 - 350. But it would be wiser to switch to the library implementation for any larger matrices. If the trends were drawn on the same graph, scipy would appear as a straight line.

(Python Code in Appendix **A**).

2. Question 2

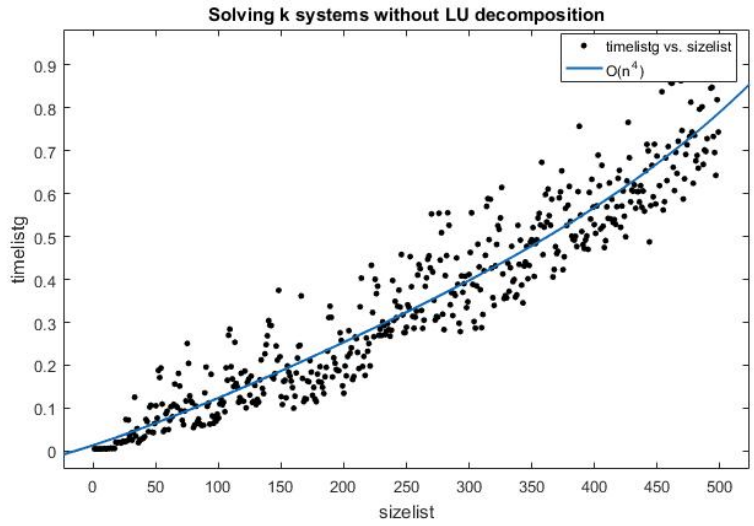
   The advantage given by LU factorization is that it decreases the overhead of calculations when Gaussian elimination is used to solve k different systems of equations. For the below analysis, we have taken different sets of equations for each order (say 1x1, 2x2, ..., 500x500). The code submitted (See Appendix **B**) will also show the differences in time complexities between the two methods of solving k systems of equations.

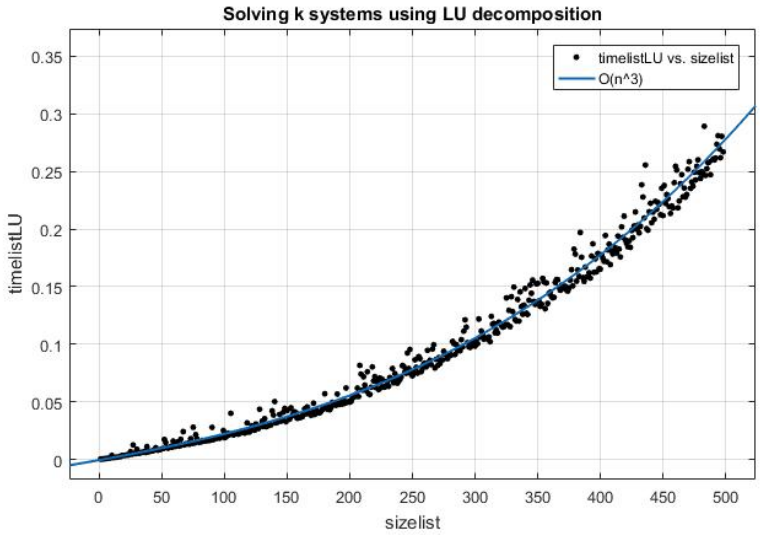
   The dataset (size, time) has been imported from the code, and has been used to plot respective scatter plots with polynomial fits, using MATLAB [5]. The LU decomposition method shows a trend of $\mathcal{O}(n^3)$ while the Gaussian elimination with k systems shows a higher trend ranging close to $\mathcal{O}(n^4)$.

   We have set $k = \frac{n}{8}$ as we did not want k to be too large. We know from the given question that the cost of solving systems of equations with

LU decomposition is $\mathcal{O}(n^3 + kn^2)$ and on the other hand if we apply gaussian elimination, the runtime complexity of solving the system is $\mathcal{O}(kn^3)$. But since we have $k = \frac{n}{8}$, our equations eventually transform $\mathcal{O}(n^3 + n^{\frac{3}{8}}) = \mathcal{O}(n^3)$ and $\mathcal{O}(n^{\frac{48}{)}} = \mathcal{O}(n^4)$, respectively.

The respective graphs are below:





3. Question 3

Linear combination is the sum of scaled vectors.

If we have n vectors $\vec{v}_1, \vec{v}_2, \vec{v}_3, ..., \vec{v}_n$ and n scalars $a_1, a_2, a_3, ..., a_n$ then

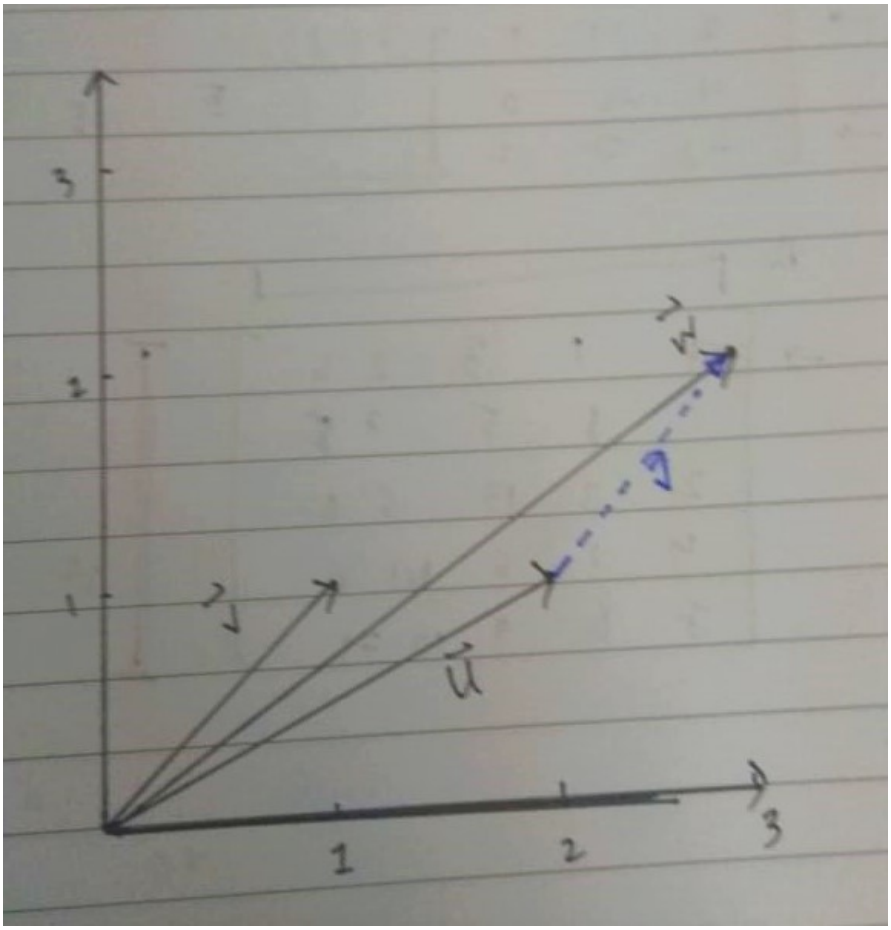
their linear combination can be written as:

$$a_1\vec{v}_1 + a_2\vec{v}_2 + a_3\vec{v}_3 + ... + a_n\vec{v}_n$$

where $\vec{v}_i$ is any arbitary vector of scalars, functions, polynomials, etc. and a_i is a scalar which could be any non-zero number.

We call it a linear combination because if we put together two or more vectors in such a way that we fix the first vector and move the other vector(s) freely in a way that they stay connected to one another then the resultant will always be a straight line (hence a *linear* combination). Scaling any of the vectors does not cause disconnection between the vectors hence we are allowed to scale the vectors and the summing of vectors is by the head-to-tale method; This is why we can scale the vectors and sum the vectors, resulting in the form mentioned above.

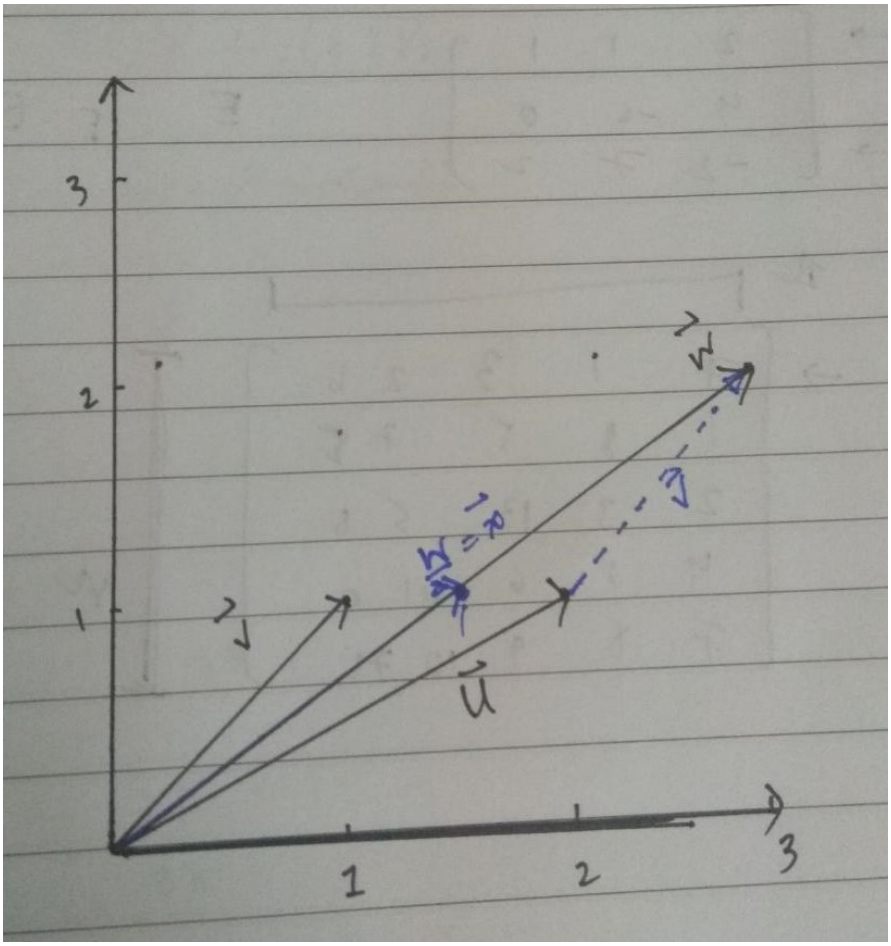
For example $\vec{u} = (2, 1)$ and $\vec{v} = (1, 1)$, and $\vec{w} = \vec{u} + \vec{v}$. This can be graphically represented as:



$\vec{w}$ is the resultant vector and can be reached if we displace $\vec{v}$ to the head of $\vec{u}$ (blue dashed line).

We can say $\vec{w}$ is a linear combination of $\vec{u}$ and $\vec{v}$ with their scalars being 1.

Another example: $\vec{x} = \frac{1}{2}\vec{u} + \frac{1}{2}\vec{v}$



$\vec{x} = \frac{1}{2}(\vec{u} + \vec{v})$

$\vec{x} = \frac{1}{2}\vec{w}$

And as represented in the graph above, the expression holds graphically aswell, with $\vec{x}$ being right up to the middle of $\vec{w}$.

For a systems of linear equations,

$$\vec{a}_1 + 2\vec{a}_2 = 4$$

$$\vec{a}_1 + 4\vec{a}_2 = 10$$

$\vec{x} = (1, 2)$

$\vec{y} = (2, 4)$

$\vec{r} = (4, 10)$

The above equation shows how many of $\vec{a}_1$ and $\vec{a}_2$ is required to make the resultant $\vec{r}$.

This can be shown as

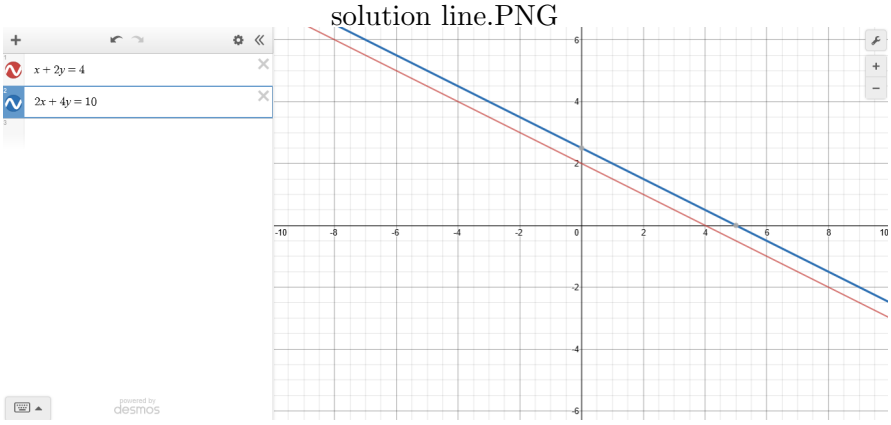
$$a_1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + a_2 \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 10 \end{bmatrix}$$

The equations are solved simultaneously to find the values of a_1 and a_2. Upon solving:

$$0 = -2$$

This represents that there is no linear combination of these vectors possible. Graphically, it means that the lines do not intersect at any point i.e. they are parallel to each other as shown in figure below:

solution line.PNG



Another system of linear equations,

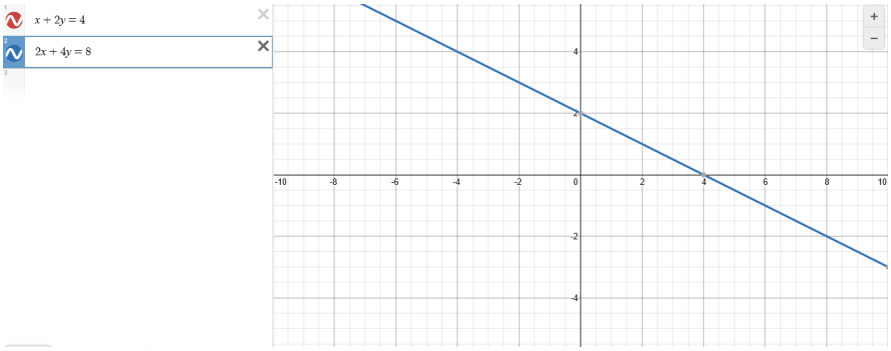
$$\vec{a}_1 + 2\vec{a}_2 = 4$$

$$\vec{a}_1 + 4\vec{a}_2 = 8$$

Upon solving:

$$0 = 0$$

This represents that there are infinite linear combinations possible for the above equations. Graphically, It means that the two lines intersect at every point i.e. they are collinear as shown in figure below:



Consider another system of linear equations.

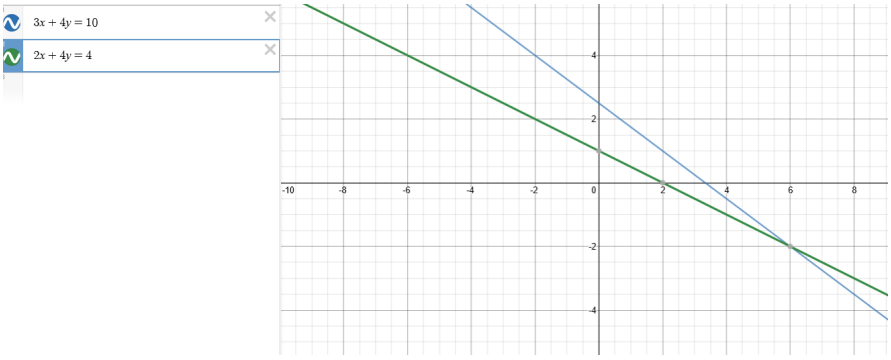
$$3\vec{a}_1 + 4\vec{a}_2 = 10$$

$$2\vec{a}_1 + 4\vec{a}_2 = 4$$

Upon solving

$$a_1 = 6; a_2 = -2$$

This represents that a unique linear combination of vectors $\begin{bmatrix}3\\4\end{bmatrix}$ and vector $\begin{bmatrix}4\\4\end{bmatrix}$ to give a resultant vector $\begin{bmatrix}10\\4\end{bmatrix}$. Graphically, the two lines intersect at only one point, as shown in the figure below:



Just as we discussed lines in $\mathcal{R}^2$ we discuss planes in $\mathcal{R}^3$.

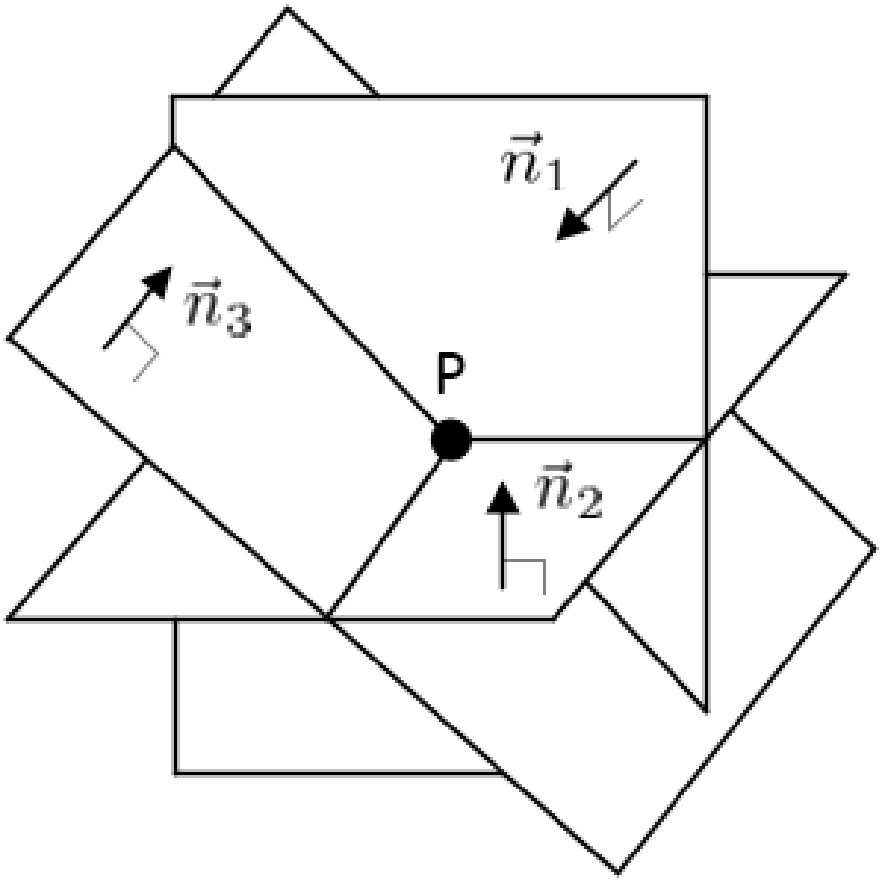
If we have vectors in the form,

$$a_1 \begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix} + a_2 \begin{bmatrix}y_1\\y_2\\y_3\end{bmatrix} + a_3 \begin{bmatrix}z_1\\z_2\\z_3\end{bmatrix}$$

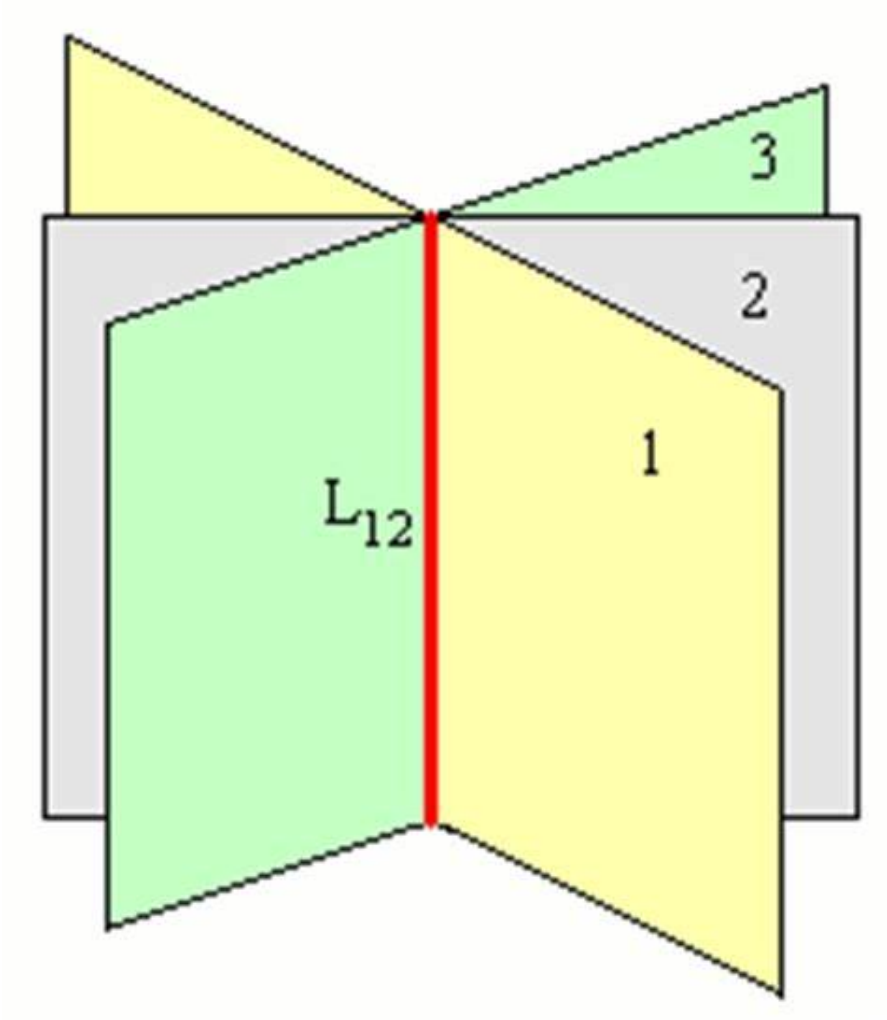
then their linear combination will only be possible if all these planes

intersect.

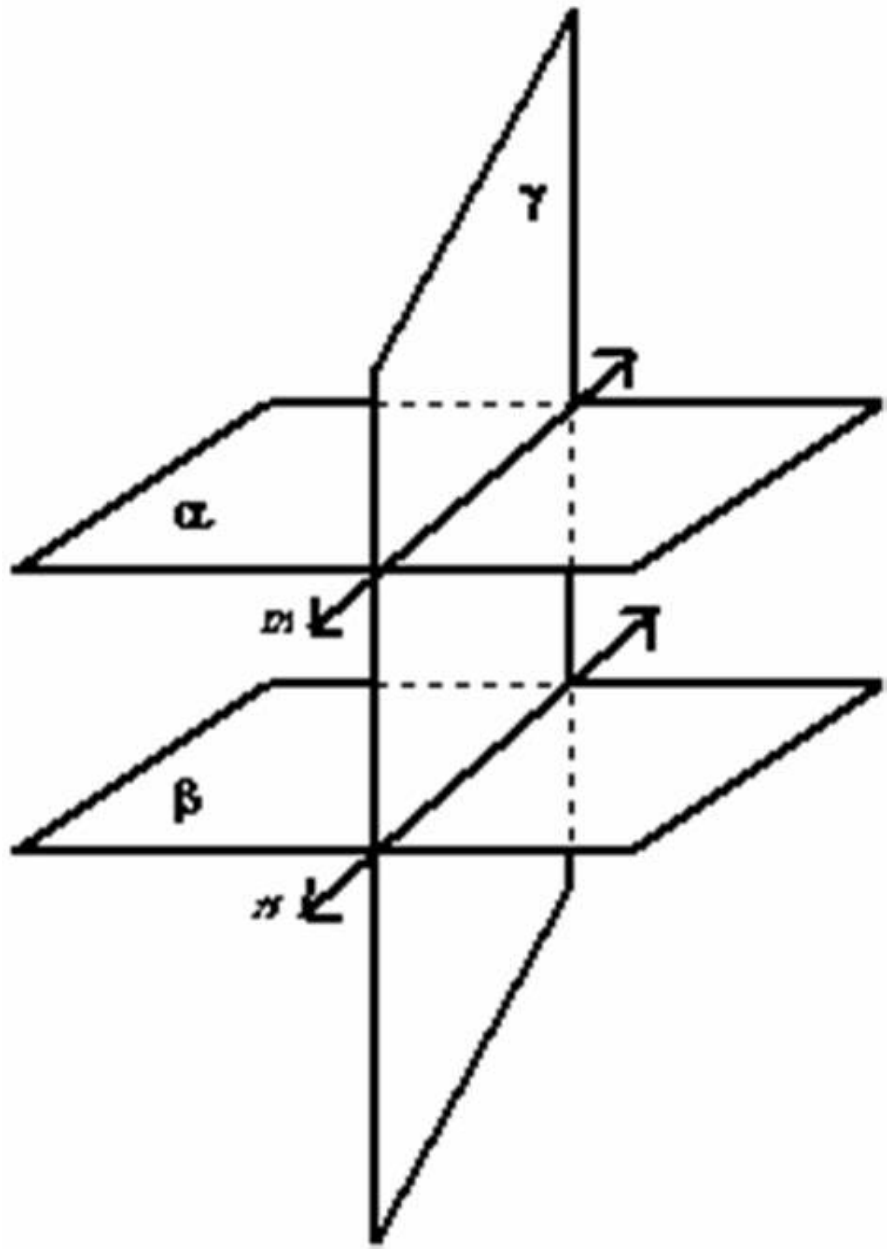
If the planes intersect at just one common point then they have a unique solution. This is shown in figure below. [3]

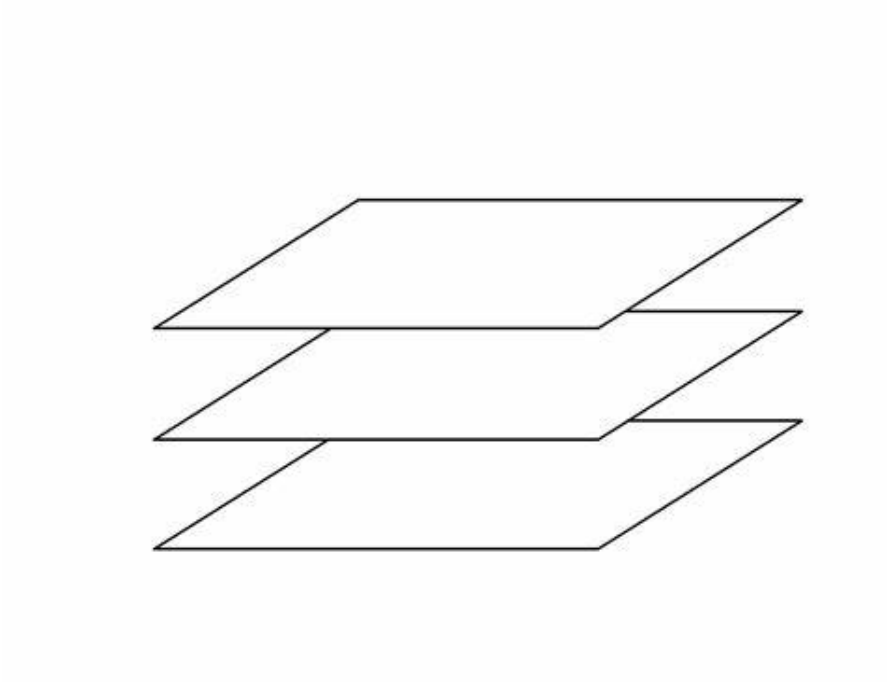


If the planes intersect such that the intersection is not a point but a line then there are infinitely many linear combinations possible. This is shown in figure below. [6]



If all the planes do not meet at some point together (For example, if out of 4 planes two intersect at one point and the other two intersect at some other point) then the linear combination of these planes is not possible. This is shown in figure below. [7] [8]


γ
α
β



-

4. Question 4

   (a) Gaussian elimination can be done as a series of matrix multiplications with $\mathbf{A}$ resulting in an upper-triangular matrix $\mathbf{U}$.

   To convert this into an upper triangular matrix $\mathbf{U}$, we need to convert entries $\mathbf{A}_{21}$, $\mathbf{A}_{31}$, and $\mathbf{A}_{41}$ to 0.

   $-\frac{v_2}{v_1}\mathrm{R}1 + \mathrm{R}2$

   $$\mathbf{E}_{21} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{v_2}{v_1} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   $-\frac{v_3}{v_1}\mathrm{R}1 + \mathrm{R}3$

   $$\mathbf{E}_{31} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{v_3}{v_1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   $-\frac{v_4}{v_1}\mathrm{R}1 + \mathrm{R}4$

$$\mathbf{E}_{41} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{v_4}{v_1} & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{E}_{41}\mathbf{E}_{31}\mathbf{E}_{21}\mathbf{A} = \mathbf{U}$$

We can write this as $\mathbf{EA} = \mathbf{U}$

$$\mathbf{E} = \mathbf{E}_{41}\mathbf{E}_{31}\mathbf{E}_{21} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{v_2}{v_1} & 1 & 0 & 0 \\ -\frac{v_3}{v_1} & 0 & 1 & 0 \\ -\frac{v_4}{v_1} & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{EA} = \mathbf{U} = \begin{bmatrix} v_1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can also say $\mathbf{A} = \mathbf{E}^{-1}\mathbf{U}$ where, $\mathbf{E}^{-1}$ is a lower triangular matrix so it can be called $\mathbf{E}^{-1} = \mathbf{L}$ so

$$\mathbf{A} = \mathbf{LU}$$

$$\mathbf{L} = \mathbf{E}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{v_2}{v_1} & 1 & 0 & 0 \\ \frac{v_3}{v_1} & 0 & 1 & 0 \\ \frac{v_4}{v_1} & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A} = \mathbf{LU} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{v_2}{v_1} & 1 & 0 & 0 \\ \frac{v_3}{v_1} & 0 & 1 & 0 \\ \frac{v_4}{v_1} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) $\mathbf{A} = \mathbf{LU}$

$$\mathbf{A^{-1}} = (\mathbf{LU})^{-1}$$

$$\mathbf{A^{-1}} = \mathbf{U^{-1}L^{-1}}$$

From the previous part we know, $\mathbf{L^{-1}} = \mathbf{E}$ hence

$$\mathbf{A^{-1}} = \mathbf{U^{-1}E}$$

Since $\mathbf{U}$ is a diagonal matrix, $\mathbf{U}^{-1} = \begin{bmatrix} \frac{1}{v_1} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

and $\mathbf{E} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{v_2}{v_1} & 1 & 0 & 0 \\ -\frac{v_3}{v_1} & 0 & 1 & 0 \\ -\frac{v_4}{v_1} & 0 & 0 & 1 \end{bmatrix}$

$$\mathbf{A}^{-1} = \begin{bmatrix} \frac{1}{v_1} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{v_2}{v_1} & 1 & 0 & 0 \\ -\frac{v_3}{v_1} & 0 & 1 & 0 \\ -\frac{v_4}{v_1} & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}^{-1} = \begin{bmatrix} \frac{1}{v_1} & 0 & 0 & 0 \\ -\frac{v_2}{v_1} & 1 & 0 & 0 \\ -\frac{v_3}{v_1} & 0 & 1 & 0 \\ -\frac{v_4}{v_1} & 0 & 0 & 1 \end{bmatrix}$$

(c) For any non-singular matrix $\mathbf{A}$ there exists one and only one matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$. [1]

If $\mathbf{A} = \mathbf{L}_1\mathbf{U}_1$ and $\mathbf{A} = \mathbf{L}_2\mathbf{U}_2$

then $\mathbf{A}^{-1} = \mathbf{U}_1^{-1}\mathbf{L}_1^{-1}$ and $\mathbf{A}^{-1} = \mathbf{U}_2^{-1}\mathbf{L}_2^{-1}$

$\mathbf{A}\mathbf{A}^{-1} = \mathbf{L}_1\mathbf{U}_1\mathbf{A}^{-1}$

$\mathbf{A}\mathbf{A}^{-1} = \mathbf{L}_1\mathbf{U}_1\mathbf{U}_2^{-1}\mathbf{L}_2^{-1}$

$\mathbf{I} = \mathbf{L}_1\mathbf{U}_1\mathbf{U}_2^{-1}\mathbf{L}_2^{-1}$

and the only way for $\mathbf{L}_1\mathbf{U}_1\mathbf{U}_2^{-1}\mathbf{L}_2^{-1} = \mathbf{I}$ is if $\mathbf{U}_2^{-1} = \mathbf{U}_1^{-1}$ and $\mathbf{L}_2^{-1} = \mathbf{L}_1^{-1}$ which is only possible if $\mathbf{U}_1 = \mathbf{U}_2$ and $\mathbf{L}_1 = \mathbf{L}_2$.

Furthermore if $\mathbf{A} = \mathbf{L}_1\mathbf{U}_1$ and $\mathbf{A} = \mathbf{L}_2\mathbf{U}_2$ then since matrix multiplication is not commutative, if $\mathbf{L}_1\mathbf{U}_1 = \mathbf{L}_2\mathbf{U}_2$ then $\mathbf{L}_1$ must $= \mathbf{L}_2$ and $\mathbf{U}_1$ must $= \mathbf{U}_2$.

(d) LU decomposition of **any** matrix $\mathbf{A}$ can also be written as $\mathbf{A} = \mathbf{LDU}$ where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices, respectively with 1s as their pivots, and $\mathbf{D}$ is a diagonal matrix.

For any arbitrary non-singular matrix $\mathbf{A}$:

$\mathbf{A} = \mathbf{LDU}$ and $\mathbf{A}^T = (\mathbf{LDU})^T$

When $\mathbf{A}$ is a symmetric matrix, we know that $\mathbf{A} = \mathbf{A}^T$

$\mathbf{A} = \mathbf{LDU} = \mathbf{A}^T = (\mathbf{LDU})^T$

$\mathbf{LDU} = \mathbf{U}^T\mathbf{D}^T\mathbf{L}^T$

$\mathbf{D}$ is a diagonal matrix hence the transpose of it is the same.

$\mathbf{LDU} = \mathbf{U}^T\mathbf{DL}^T$

Since matrix multiplication is not commutative, for the equality to hold, $\mathbf{L} = \mathbf{U}^T$ and $\mathbf{U} = \mathbf{L}^T$

$\mathbf{A}^T = \mathbf{U}^T\mathbf{DL}^T = \mathbf{A}$

$\mathbf{A} = \mathbf{U}^T\mathbf{DL}^T$ and since $\mathbf{L} = \mathbf{U}^T$,

$\mathbf{A} = \mathbf{LDL}^T$

3 Appendix A

Python Code for Question 1. [2]

```
import random, time
from scipy.linalg import lu

#implement a basic GE algorithm and compare it's runtime
#with that of a library implementation

#elementary row operations
def row_mul(s, lst):
    return [i*s for i in lst]

def row_subt(r2, r1):
    #r2 - r1
    #r1 and r2 are a row of the matrix (in a list)
    assert len(r1) == len(r2)
    return [r2[i] - r1[i] for i in range(len(r1))]

#Gaussian Elimination implementation:
def GE(mat):
    for i in range(1, len(mat)):
        for j in range(i):
            if mat[j][j] == 0:
                return False
            mat[i] = row_subt(mat[i], row_mul(mat[i][j]/mat[j][j], mat[j]))
    return mat

def generateMatrix(dim):
    return [[random.randint(1, 10) for i in range(dim)] for j in range(dim)]

def main():

    record = open("SelfImplementedGE.csv", "w")
    scipyRecord = open("SciPyGE.csv", "w")

    for i in range(5, 500, 5):

        A = generateMatrix(i) #Matrix for Self Implemented GE
        B = [i.copy() for i in A]

        start_time = time.time()
```

```
        while GE(A) == False: #if A is singular
            A = generateMatrix(i) #Regenerate a matrix
            B = [i.copy() for i in A]
            start_time = time.time() #restart timer

        stop_time = time.time() - start_time

        record.write(str(i) + ", " + str(stop_time) + "\n")

        start_time = time.time()

        pl, u = lu(B, permute_l=True)

        stop_time = time.time() - start_time

        scipyRecord.write(str(i) + ", " + str(stop_time) + "\n")


    record.close()
    scipyRecord.close()

main()
```

4 Appendix B

Python Code for Question 2

```
import numpy as np
import scipy
import scipy.linalg
import time
import matplotlib.pyplot as plt

def create_square_matrix(_type,order):
    if _type == 1:
        return scipy.random.randint(1,200,(order,order))
    if _type == 2:
        return numpy.random.randint(1,200,(order,order))

timeListLU=[]
sizeListLU=[]
timeListG=[]
def makeGraph(n):

    for i in range(8,n,1):
        A = create_square_matrix(1,i)
        a=time.clock()
        LU = scipy.linalg.lu_factor(A)
        for j in range (i):
            b=scipy.random.randint(1,20,(i,1))
            result=scipy.linalg.lu_solve(LU,b)
        b=time.clock()

        sizeListLU.append(i)
        print(i)
        timeListLU.append(b-a)
        a=time.clock()
        for j in range (i//8):
            b=scipy.random.randint(1,20,(i,1))
            result=np.linalg.solve(A,b)
        b=time.clock()
        timeListG.append(b-a)
    plt.plot(sizeListLU,timeListG)
    plt.plot(sizeListLU,timeListLU)

    ## plt.plot(sizeListLU,[(a*i)**3 for i in sizeListLU])
    plt.show()
```

```
makeGraph(200)
```

References

[1] Dr. Majeed's lecture notes.
Lecture 4 slide 5/23: Existence of an inverse

[2] Using Scipy for LU Decomposition

`https://stackoverflow.com/questions/15638650/is-there-a-standard-solution-for-gauss-elimination-in-python`

[3] Planes intersecting at one point

`https://www.bing.com/th?id=OIP.9YOKRzWPzMUt5CjadfiBDAAAAA&pid=Api&rs=1`

[4] Desmos for plotting lines (Question 1)

[5] Mathworks: MATLAB for Question 2

[6] Planes intersecting on a line.

`https://www.bing.com/th?id=OIP._tToqHdNhT-VvimtTnFngQHaIk&pid=Api&rs=1`

[7] Planes not intersecting 1

`https://www.bing.com/th?id=OIP.hMuQlz6JpsSQKbXlhdVwoAHaKe&pid=Api&rs=1`

[8] Planes not intersecting 2

`https://www.bing.com/th?id=OIP.0UYCquCiPyS3dLcd8KWVOQHaFu&pid=Api&rs=1`