HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

---------***----------

MAJOR ASSIGNMENT REPORT

MODULE: LINEAR ALGEBRA (EXERCISE)

CODE: MT1008

Student: TRẦN NGUYỄN GIA PHÁT

Student ID: 2153681

Class: CC21KHM2

Faculty: Computer Science and Engineering

Cohort: CC03 – Group 13

Semester: 213

Academic year: 2021-2022

Lecturer: HOÀNG HẢI HÀ

Submission date: 10th September 2022

FOR EXAMINERS ONLY

Grade (in number):

……..

Grade (in words):

……..

Examiner 1

(Signature & Full name)

……..

Examiner 2

(Signature & Full name)

……..

Ho Chi Minh City, September 2022

STATEMENT OF AUTHORSHIP

Except where reference is made in the text of the report, this assignment contains no material published elsewhere or extracted in a whole or in part from an assignment which we have submitted or qualified for or been awarded another degree or diploma.

Any work, if not owned by the author himself, resembles to any part of this report without written permission from the author is stricly prohibited and considered plagiarism.

No other person's work has been used without the acknowledgments in the report.

This report has not been submitted for the evaluation of any other models or the award of any degree or diploma in other tertiary institutions.

Ho Chi Minh City, September 2022

ACKNOWLEDGEMENT

I would like to express my gratitude towards Ms Hoang Hai Ha, for helping me to coordinate in writing this report. I have explored meaningful findings during this work.

I am also very grateful to article writers on the internet for providing priceless knowledge about the field I am doing. Without their contributions, this report may not be done completely.

DISCLAIMER

My goal is to create user-friendly, multi-purpose and interactive programs for convenience in providing versatile inputs and returning desired results. Therefore, robust codes and advanced commands/syntaxes used in the process which are out of this subject's scope are not explained in the central part of the report.

For concise explanation, only the snippets that contain necessary calculations are shown and may not be similar to the final code. Comments are minimized for readability.

Please only run the codes provided in the *Appendices* for demostration. Readers may look up essential installation, required libraries and guidelines in the same section.

ABOUT THE AUTHOR

Trần Nguyễn Gia Phát **2153681**

If you need more information, contact me at:

- Phone number: (+84) 049 737 622
- Email: phat.trancsdev@hcmut.edu.vn

INTRODUCTION

Linear algebra is the broad area of mathematics dealing with such topics as systems of linear equations, vector spaces, matr-ix theory, and optimization. Underlying all of these topics are linear algebra concepts, such as linear transformation, determinants, vector bases, eigenvalues and eigenvectors. Linear algebra has two primary applications: numerical linear algebra and statistics linear algebra.

In the report, the writers will give explanations and solutions for problems involving (a) matrix decomposition, (b) Leslie's model for population growth, (c) Markov model for probabilities and (d) traffic flow network.

TABLE OF CONTENT

I. Problem 1 5

II. Problem 2 8

III. Problem 3 10

IV. Problem 4 15

V. Problem 5 20

VI. Conclusion 24

VII. References 25

VIII. Appendices 27

This report also provides coded calculators for solving the system of linear equations using matrix analysis. Matrix analysis is a branch of mathematics that studies matrices and their algebraic properties. The properties of these basic concepts form the basis of many of the mathematical tools used in other fields such as applied mathematics, physics, economics and engineering.

I. PROBLEM 1

Given $\boldsymbol{A}$ be a square matrix size $\boldsymbol{n}$. Factorizing $\boldsymbol{LU}$ of $\boldsymbol{A}$ is the process of finding a lower triangular matrix $\boldsymbol{L}$ and a upper triangular matrix $\boldsymbol{U}$ such that $\boldsymbol{A = LU}$.

Write a code to find matrices $\boldsymbol{L, U}$ for a given $\boldsymbol{A}$. Knowing that the main diagonal of $\boldsymbol{L}$ contains only number 1.

1.1. Theories

Certain types of matrices are more efficiently to work with than others. Therefore, one can write an arbitrary square matrix $\boldsymbol{M}$ as the product of two triangular matrices for easier computation. Let $\boldsymbol{M = LU}$, where:

$\boldsymbol{L}$ is lower triangular. This indicates that all entries above the main diagonal are zeros and the main diagonal consists of only ones.

$$\boldsymbol{L} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \cdots \\ \boldsymbol{l_j^{i-1}} & \mathbf{1} & \mathbf{0} & \cdots \\ \boldsymbol{l_j^i} & \boldsymbol{l_{j+1}^i} & \mathbf{1} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

$\boldsymbol{U}$ is upper triangular. This indicates that all entries below the main diagonal are zeros.

$$\boldsymbol{U} = \begin{pmatrix} \boldsymbol{u_1^1} & \boldsymbol{u_{j-1}^i} & \boldsymbol{u_j^i} & \cdots \\ \mathbf{0} & \boldsymbol{u_2^2} & \boldsymbol{u_j^{i+1}} & \cdots \\ \mathbf{0} & \mathbf{0} & \boldsymbol{u_3^3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

However, not every matrix has $\boldsymbol{LU}$ factorization. Let:

$$\Delta_1 = \boldsymbol{a_{11}}, \Delta_2 = \begin{vmatrix} \boldsymbol{a_{11}} & \boldsymbol{a_{12}} \\ \boldsymbol{a_{21}} & \boldsymbol{a_{22}} \end{vmatrix}, \dots, \Delta_n = \begin{vmatrix} \boldsymbol{a_{11}} & \cdots & \boldsymbol{a_{1n}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{n1}} & \cdots & \boldsymbol{a_{nn}} \end{vmatrix}$$

be leading principal minors of the matrix.

$\boldsymbol{LU}$ factorization of a nonsingular (invertible) matrix exists if and only if all leading principal submatrices are nonsingular. ($\Delta_1 \neq \boldsymbol{0}, \boldsymbol{i} \in \{\boldsymbol{1}, \dots, \boldsymbol{n}\}$)

1.2. Implementation

In Python, Sympy is the library used for symbolic mathematics computation, and it already has a built-in method for calculating LU decomposition. Therefore, one does not need to build a complex algorithm from scratch.

Code snippet:

```
from sympy import Matrix
n = int(input("Size n of matrix: "))
A = Matrix([input().strip().split() for _ in range(n)])
print("A")
A
L,U,_ = A.LUdecomposition()
print("L")
L
print("U")
U
```

1.3. Result

The coding result matches precisely with the manual double check and is the final implementation for Problem 1. See the below figure:

```
print("A")
A
L,U,_ = A.LUdecomposition()
print("L")
L
print("U")
U
```

A

$$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 2 & 3 \\ 0 & 2 & 1 \end{bmatrix}$$

L

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

U

$$\begin{bmatrix} -1 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 1.1: Sample result for Problem 1

Source codes for all problems are available at:

https://github.com/Zaphat/Linear-Algebra-Project

II. PROBLEM 2

Given a square matrix $\boldsymbol{A_n}$ and a vector $\boldsymbol{B_{n\,x\,1}}$. Using the code in previous question to find $\boldsymbol{L}, \boldsymbol{U}$. Then solving the following systems: Find $\boldsymbol{y}$ such that $\boldsymbol{Ly = b}$ and find $\boldsymbol{x}$ such that $\boldsymbol{Ux = y}$.

2.1. Theories

Compared to the Gaussian elimination, $\boldsymbol{LU}$ decomposition has a particular advantage when solving the equation system. The Gaussian method does not determine $\boldsymbol{L}$ explicitly but forms $\boldsymbol{L^{-1}b}$ so that the right-hand sides must be known before solving the equation. On the other hand, the $\boldsymbol{LU}$ method does not need the right-hand sides to be known in advance, since $\boldsymbol{L}$ and $\boldsymbol{U}$ are previously obtained and can be used for any right-hand sides without recalculation.

Since $\boldsymbol{A = LU}$, then $\boldsymbol{Ax = b}$ becomes

$$\boldsymbol{LUx = b}$$

where $\boldsymbol{b}$ is not restricted to a single column. Let $\boldsymbol{y = Ux}$ leads to

$$\boldsymbol{Ly = b}$$

As $\boldsymbol{L}$ is lower triangular matrix, $\boldsymbol{y}$ can be solved by forward substitution. To find $\boldsymbol{x}$ we solve

$$\boldsymbol{Ux = b}$$

As $\boldsymbol{U}$ is upper triangular matrix, $\boldsymbol{x}$ can be solved by backward substitution.

Therefore, we derive the solution set $\boldsymbol{x}$ for the equation system.

2.2 Implementation

Code snippet:

```
n = int(input("Size n of matrix A: "))
A = Matrix([input().strip().split() for _ in range(n)])
```

```
B = Matrix([input().strip().split() for _ in range(n)])
L,U,_ = A.LUdecomposition()
Y = L.inv()@B
X = U.inv()@Y
print("A")
A
print("B")
B
print("X")
X
```

2.3. Result

The coding result matches precisely with the manual computation and is the final implementation for Problem 2. See the below figure:

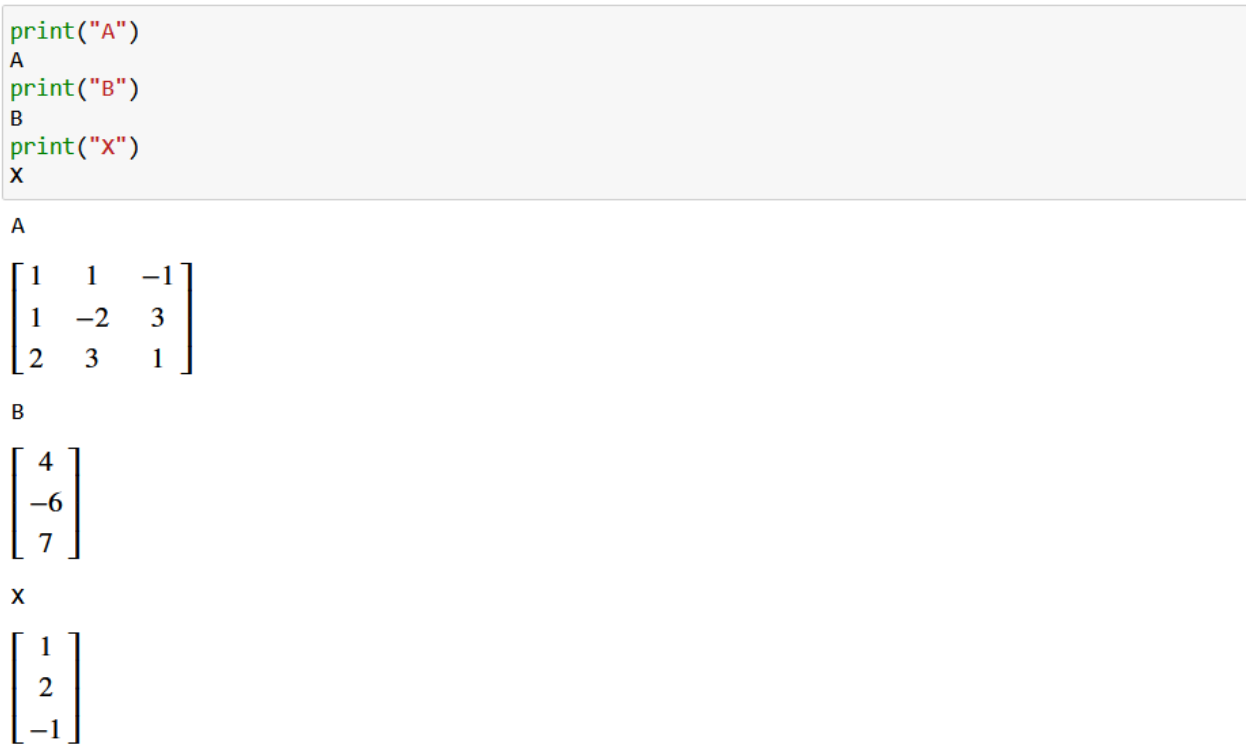


Figure 2.1: Sample result for Problem 2

III. PROBLEM 3

The following tables indicate the birth rate and survival rate of a population of woodland caribou and the number of individuals in each group of age class in 1990, respectively

Age (years)	Birth Rate	Survival Rate
0–2	0.0	0.3
2–4	0.4	0.7
4–6	1.8	0.9
6–8	1.8	0.9
8–10	1.8	0.9
10–12	1.6	0.6
12–14	0.6	0.0

Age (years)	Number
0–2	10
2–4	2
4–6	8
6–8	5
8–10	12
10–12	0
12–14	1

- Write a code to find the number of individuals in each group of class age in T, where T is some year after 1990 (for instance, T = 2022).
- In a long run, can you give some conclusions about the number of members in each group?

3.1. Theories

Leslie Model is an age-structured model of population growth in population ecology. It is a discrete model in which the population of one sex (usually females) is classified into age classes. The population is isolated and only examines the births and deaths amongst the age classes.

The Leslie model is writen as:

$$\boldsymbol{X_{n+1} = LX_n}$$

$\boldsymbol{L}$ is the Leslie matrix and takes the form:

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{F_0} & \boldsymbol{F_1} & \cdots & \boldsymbol{F_n} \\ \boldsymbol{S_0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{S_{n-1}} & \boldsymbol{S_n} \end{bmatrix}$$

- $\boldsymbol{F_x}$: age-specific fecundity (Birth rate)
- $\boldsymbol{S_x}$: age-specific survival rate (values ranging from 0 to 1)

And $\boldsymbol{X_n}$ is the population matrix of each age class at state $\boldsymbol{n}$ and takes the form:

$$\boldsymbol{X_n} = \begin{bmatrix} \boldsymbol{N_0} \\ \boldsymbol{N_1} \\ \vdots \\ \boldsymbol{N_n} \end{bmatrix}$$

- $\boldsymbol{N_x}$: age-specific population

To find the population proportion at the next period step, we multiply $\boldsymbol{L}$ to the left of $\boldsymbol{X}$. Hence, we derive the formula for estimating the population at any given state $\boldsymbol{n}$, given the initial population $\boldsymbol{X_0}$:

$$\boldsymbol{X_n = L^n X_0}$$

3.2. Implementation

Code snippet:

```
#Setting up Leslie Matrix and initial values
print("Enter 'Birth rate - Survival Rate - Initial population' of
each group\nSeperated by white space")
A = Matrix([input().strip().split() for _ in range(n)])
L = Matrix((Matrix(1,n,A.col(0)),Matrix(np.diag([i for i in
A.col(1)]))))
L = L[:n,:]
```

```
N = Matrix(A.col(2))
#Calculate population at given year
year = int(input("Enter the year to review: "))
if year<1991:
    year = 1991
print(f"Number of individuals in each group in {year}: ")
power(L,year-1990)@N
#Graphing population change in a long period of time
span = int(input("Enter number of years (eg 100) for examinating
population proportions: "))
x = [int(i*(span/10)) for i in range(11)]
y = [list(power(L,i)@N) for i in x]
fig,ax = plt.subplots(constrained_layout=False, figsize=(9.6,7));
fig.canvas.toolbar_position = 'bottom';
fig.canvas.header_visible = False;
ax.tick_params(axis='x',which='minor',direction='out',bottom=True
,length=5)
ax.set_xlabel('Time steps (years)');
ax.set_ylabel('Population (individuals)');
ax.set_title(f'Future Population Per Age Class in {span} Years');
ax.minorticks_on();
ax.grid(visible=True, which='major', axis='both', alpha=0.8);
ax.grid(visible=True, which='minor', axis='both',
alpha=0.5,linestyle='dashed');
plt.xticks(np.arange(len(x)),x);
ax.autoscale(enable=True, axis='both', tight=None);
for j in range(len(y[0])):
    ax.plot(list(y[i][j] for i in
range(len(y))),label=f"{ordinalGen(j+1)}
group",color=colorsys.hsv_to_rgb(j/n+1,1,0.75));
ax.set_xlim(0,auto=True);
ax.legend(loc=2, prop={'size': 10});
```

3.3. Result

The coding result matches precisely with manual computation and is the final implementation for Problem 3. See the below figures:

```
Enter 'Birth rate - Survival Rate - Initial population' of each group
Seperated by white space
0 0.3 10
0.4 0.7 2
1.8 0.9 8
 1.8 0.9 5
1.8 0.9 12
1.6 0.6 0
0.6 0.0 1
```

```
def power(A,n):
    return A**n
print("Leslie matrix: ")
L
print("Initial population: ")
N
```

```
Leslie matrix:
```

$$\begin{bmatrix} 0 & 0.4 & 1.8 & 1.8 & 1.8 & 1.6 & 0.6 \\ 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.6 & 0 \end{bmatrix}$$

```
Initial population:
```

$$\begin{bmatrix} 10 \\ 2 \\ 8 \\ 5 \\ 12 \\ 0 \\ 1 \end{bmatrix}$$

Figure 3.1: Set up matrices from user input

```
Enter the year to review: 2022
Number of individuals in each group in 2022:
```

$$\begin{bmatrix} 484.789389613458 \\ 133.132803180503 \\ 85.3084458192699 \\ 70.2824597962908 \\ 57.9037077472845 \\ 47.7047074205198 \\ 26.2010952309779 \end{bmatrix}$$

```
Enter number of years (e.g. 100) for examinating population proportions: 150
```

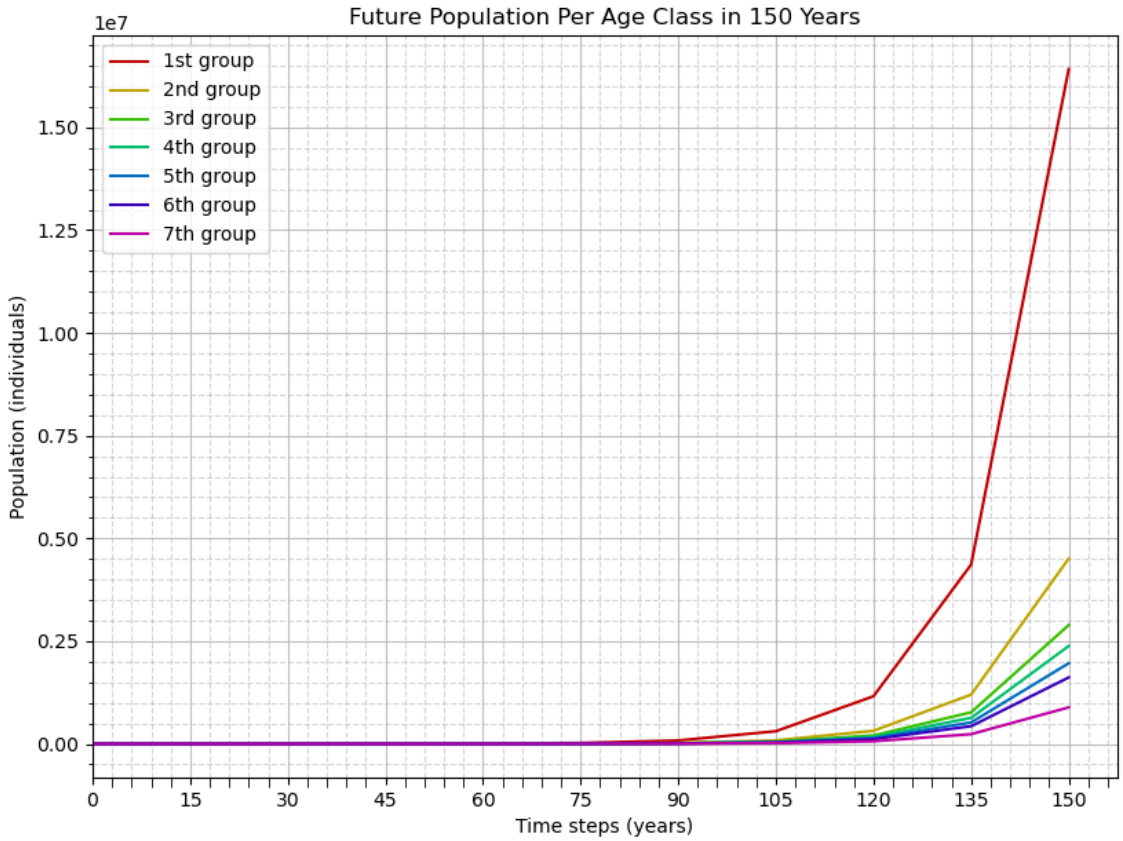
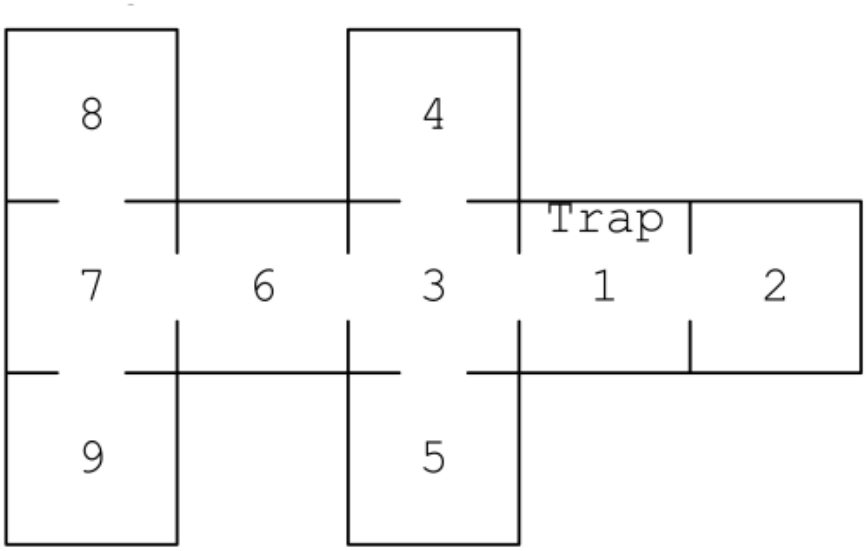


Figure 3.2: Final result for Problem 3

IV. PROBLEM 4

A mouse trap is placed in room 1 of the house with the pictured floor plan. Each time the mouse comes into room 1, he is trapped with probability p = 0.1. If he is not trapped, he leaves each room by one of its exits, chosen at random.



- Find the matrix transition describing the path of the mouse through the house.
- A vector q is called steady vector of Markov model if P q = q where P is matrixtransition, find q.
- Suppose that the mouse starts in Room 4, what is the probability that the mouse in Room 1 after 3 steps?

4.1. Theories

A mathematical called a Markov chain transitions from one state to another following certain probabilistic principles. Markov chains are a widespread and straightforward method for statistically modeling random processes which depend only on the result of the previous event. Their applications cover a range of areas, from text generation to business analysis.

Let $\boldsymbol{M}$ be the transition matrix from each state to another state based on the probability that the event will occur. $\boldsymbol{M}$ then takes the form:

$$M = \begin{matrix} m_{1,1} & \cdots & m_{1,1} & \cdots & m_{1,n} \\ m_{2,1} & \ddots & \vdots & \ddots & \vdots \\ \vdots & \cdots & m_{1,1} & \cdots & m_{i,n} \\ m_{i,1} & \cdots & \cdots & \ddots & \vdots \\ \vdots & \cdots & m_{i,j} & \cdots & m_{n,n} \end{matrix}$$

where $\boldsymbol{m_{i,j}}$ represents the transition probability an entity from state $\mathbf{i^{th}}$ in current time moves to state $\mathbf{j^{th}}$ at the next step.

Suppose we want to know the probability an entity at state $\mathbf{i^{th}}$ will be at state $\mathbf{j^{th}}$ after $\boldsymbol{k}$ steps, then we take $\boldsymbol{M}$ to the power of $\boldsymbol{k}$. The value of $\boldsymbol{m_{i,j}}$ after calculation is the desired answer. In the notation form:

$$\boldsymbol{P(i,j) = m_{i,j} \leftrightarrow m_{i,j} \in M^k}$$

- $\boldsymbol{P(i,j)}$: the probability an entity in state i moves to state j after k steps.
- $\boldsymbol{M}$: the transition matrix.

In most Markov chains, however, the probability for a particular one of the states will approach a limiting value as the number of steps goes to infinity. In other words, in the long run, the probabilities will not change much from one transition to the next. These limiting values are called stable probabilities. Let $\boldsymbol{q}$ be the vector consisting of the stable probability of each state. We have:

$$\boldsymbol{Pq = q}$$

4.2. Implementation

Code snippet:

```
#Get user input
n = int(input("Number of rooms: "))
print("Enter adjacent room(s) relative to the current room,
seperated by white space\nAdjacent rooms to")
room_map = [input(f"Room {i+1}:   ").strip().split() for i in
range(n)]
```

```
trap = input("Enter room(s) that contain mouse
trap:    ").strip().split()
trap_prob = float(input("Enter successfully trapped probability:
"))
#Construct transition matrix
def get_Markov(room_map,trap,trap_prob):
    m = []
    for i in range(len(room_map)):
        if str(i+1) in trap:
            tmp = [nsimplify((1-trap_prob)/len(room_map[i])) if
str(room+1) in room_map[i] else 0 for room in range(n)]
            tmp[i]=trap_prob
        else:
            tmp = [nsimplify(1/len(room_map[i])) if str(room+1)
in room_map[i] else 0 for room in range(n)]
        m.append(tmp)
    return Matrix(m)
print("Total Transition Matrix")
markov = get_Markov(room_map,trap,trap_prob)
markov
#Compute probability
details =
[widgets.IntSlider(value=4,min=1,max=9,step=1,description='From
Room',orientation='horizontal',readout=True,readout_format='d'),
 widgets.IntSlider(value=1,min=1,max=9,step=1,description='To
Room',orientation='horizontal',readout=True,readout_format='d'),
 widgets.Text(value='3',description='After',layout=Layout(display
="flex", justify_content="center", width="200px")),
 widgets.Label(value='Steps')]
result = widgets.Label(value="")
button_compute =
widgets.Button(description='Compute',layout={'width':
'auto'},tooltip = "Cook math",style = dict(button_color =
'lightgreen'))
print("Transtition Probability")
display(widgets.HBox([i for i in details]))
display(button_compute)
```

```
display(result)
def compute(event):
    path = markov**int(details[2].value)
    result.value = f'P({details[0].value},{details[1].value}) =
{round(path[details[0].value-1,details[1].value-1],3)}'
button_compute.on_click(compute)
```

4.3. Result

The coding result matches precisely with manual computation and is the final implementation for Problem 4. See the below figures:

```
Enter adjacent room(s) relative to the current room, seperated by white space
Adjacent rooms to
Room 1:   2 3
Room 2:   1
Room 3:   1 4 5 6
Room 4:   3
Room 5:   3
Room 6:   3 7
Room 7:   6 8 9
Room 8:   7
Room 9:   7
Enter room(s) that contain mouse trap:    1
Enter successfully trapped probability: 0.1
```

Total Transition Matrix

$$\begin{bmatrix} 0.1 & \frac{9}{20} & \frac{9}{20} & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Steady-State Vector of Markov Chain

Out[7]:

$$\begin{bmatrix} \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \\ \frac{1}{9} \end{bmatrix}$$

Figure 4.1: User input and the transition matrix/steady vector

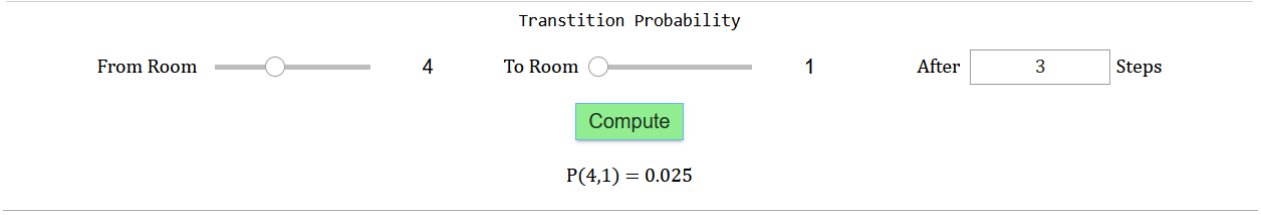


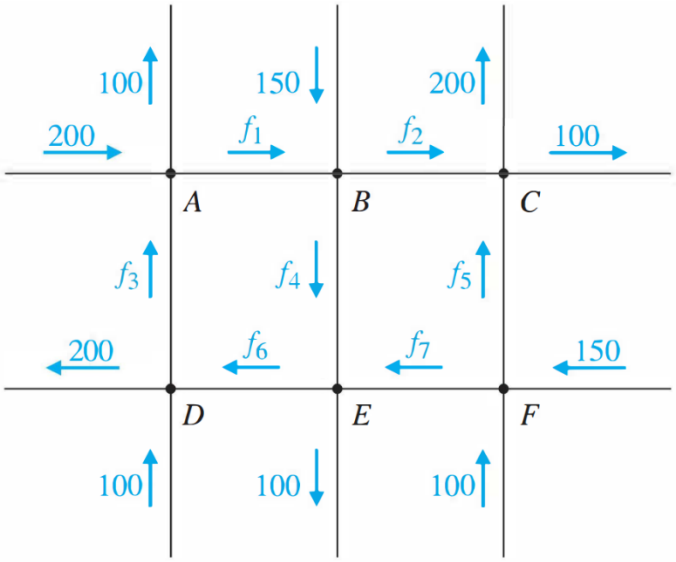
Figure 4.2: Probability result for Problem 4

Source codes for all problems are available at:

https://github.com/Zaphat/Linear-Algebra-Project

V. PROBLEM 5

The next figure indicates a traffic network, the number of vehicles being in and out at each node. Suppose that all streets are one way.



- Set up a system to find the unknown flows.
- Solve the system when $f_1 = 100$, $f_7 = 150$.
- Solve the system when $f_4 = 0$, then what will the range of flow be on each of the other branches?

5.1. Theories

In a traffic network, consider road intersections as nodes and the volume of vehicles on roads as flows. In such a network, the rate of entering flow is equal to the rate of leaving flow at each node. That is:

$$\sum \boldsymbol{flow\ in} = \sum \boldsymbol{flow\ out}$$

Hence, we obtain $\boldsymbol{n}$ equalities from the corresponding number of nodes and determine hidden flows based on specific known flows by labeling unknown flows as variables. The problem eventually leads to solving a system of linear equations.

5.2. Implementation

Code snippet:

```
# Set up the widgets for input
nodes =
[widgets.Text(value='f3+200=f1+100',description='A',layout=Layout
(display="flex", justify_content="center", width="500px")),
        widgets.Text(value='f1+150=f2+f4',description='B',layout=
Layout(display="flex", justify_content="center", width="500px")),
         widgets.Text(value='f2+f5=200+100',description='C',layou
t=Layout(display="flex", justify_content="center",
width="500px")),
         widgets.Text(value='f6+100=200+f3',description='D',layou
t=Layout(display="flex", justify_content="center",
width="500px")),
         widgets.Text(value='f4+f7=f6+100',description='E',layout
=Layout(display="flex", justify_content="center",
width="500px")),
         widgets.Text(value='100+150=f5+f7',description='F',layou
t=Layout(display="flex", justify_content="center",
width="500px"))]
subs = [widgets.Text(value='',description=f" f{i+1} =
",layout=Layout(width='180px',height='30px'))
       for i in range(7)]
result = [widgets.Text(value='',description=f" f{i+1} =
",style={'description_width':
'initial'},layout=Layout(width='250px',height='30px'))
       for i in range(7)]
button_compute =
widgets.Button(description='Compute',layout={'width':
'auto'},tooltip = "Make magic",style = dict(button_color =
'lightgreen'))
display(widgets.HBox([widgets.VBox([widgets.Label(value="Enter
equality of in and out flows at each node")]+[_ for _ in nodes]),
                      widgets.VBox([widgets.Label(value="Substitu
tion")]+[_ for _ in subs])]))
display(button_compute)
```

```
#Solving
def get_expr(s):
    return f"({s.split('=')[0]})-({s.split('=')[1]})"
def compute(event):
    try:
        clear_output(wait=True)
        display(widgets.HBox([widgets.VBox([widgets.Label(value="
Enter equality of in and out flows at each node")]+[_ for _ in
nodes]),
                widgets.VBox([widgets.Label(value="Substitution")
]+[_ for _ in subs])]))
        display(button_compute)
        values = [i.value for i in subs]
        sys_eq = [get_expr(i.value) for i in nodes]
        if any(values):
            for i in range(len(values)):
                if values[i]:
                    sys_eq[:] = [s.replace(f'f{i+1}',values[i])
for s in sys_eq]
        solutions = next(iter(linsolve(sys_eq,symbols('f1 f2 f3
f4 f5 f6 f7'))))
        for r,v,s in zip(result,values,solutions):
            if v:
                r.value = str(v)
            else:
                r.value = str(s)
        print('Result')
        display(widgets.VBox([_ for _ in result]))
    except:
        clear_output(wait=True)
        display(widgets.HBox([widgets.VBox([widgets.Label(value="
Enter equality of in and out flows at each node")]+[_ for _ in
nodes]),
                widgets.VBox([widgets.Label(value="Substitution")
]+[_ for _ in subs])]))
        display(button_compute)
        display('CANNOT COMPUTE')
```

5.3. Result

Manual solution

Obtained equation system:

$$\begin{cases} -f_1 + f_3 = -100 \\ f_1 - f_2 - f_4 = -150 \\ f_2 + f_5 = 300 \\ -f_3 + f_6 = 100 \\ f_4 - f_6 + f_7 = 100 \\ f_5 + f_7 = 250 \end{cases} \Rightarrow \begin{cases} f_1 = f_6 \\ f_2 = f_7 + 50 \\ f_3 = f_6 - 100 \\ f_4 = f_6 - f_7 + 100 \\ f_5 = 250 - f_7 \\ f_6 = f_6 \\ f_7 = f_7 \end{cases}$$

where f_6, f_7 are free variables.

When $f_1 = 100, f_7 = 150$:

$$\begin{cases} f_1 = 100 \\ f_2 = 200 \\ f_3 = 0 \\ f_4 = 50 \\ f_5 = 100 \\ f_6 = 100 \\ f_7 = 150 \end{cases}$$

When $f_4 = 0$:

$$\begin{cases} f_1 = f_4 - 100 \\ f_2 = f_7 + 50 \\ f_3 = f_7 - 200 \\ f_4 = 0 \\ f_5 = 250 - f_7 \\ f_6 = f_7 - 100 \\ f_7 = f_7 \end{cases}$$

The coding result matches precisely with manual calculation and is the final result for Problem 5. See the below figure:

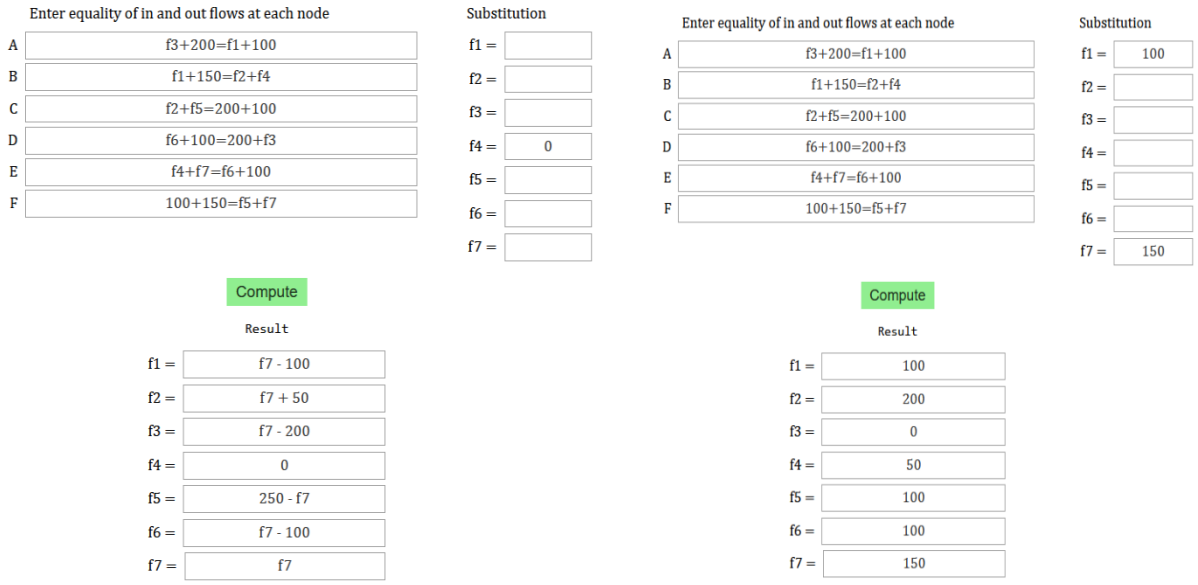


Figure 5.1: Final results for Problem 5

VI. CONCLUSION

In the real world, various problems can be addressed using matrix analysis and systems of linear equations. Nonetheless, the process of calculating by hand can be time-consuming and error-prone. Therefore, computer programs offer the ability to solve many problems quickly and conveniently and are becoming increasingly popular among scientists and engineers in various fields of study.

While the calculator programs written for this report might help find solutions on most occasions, some issues still need improvement for better quality and performance.

VII. REFERENCES

[1] Age Structured Leslie Matrix. Amrita Vishwa Vidyapeetham University. (2011). Retrieved September 7, 2022, from https://vlab.amrita.edu/?sub=3&brch=65&sim=183&cnt=1

[2] Age-Structured Population Models. San Diego State University. (2007). Retrieved September 7, 2022, from https://jmahaffy.sdsu.edu/courses/f09/math636/lectures/les_mat/les_mat.pdf

[3] Anton, H. (n.d.). Elementary Linear Algebra. University of Pennsylvania. Retrieved September 7, 2022, from https://www2.math.upenn.edu/~kazdan/312F12/JJ/MarkovChains/markov_chains_(Anton).pdf

[4] Brownlee, J. (2019, August 9). A gentle introduction to linear algebra. Machine Learning Mastery. Retrieved September 7, 2022, from https://machinelearningmastery.com/gentle-introduction-linear-algebra/

[5] Joseph, R. (2018, July 16). The Markov Mouse. Towards Data Science. Retrieved September 7, 2022, from https://towardsdatascience.com/in-5-mins-the-markov-mouse-a4f7a38289fb

[6] Leslie Growth Models. Duke University. (n.d.). Retrieved September 7, 2022, from https://services.math.duke.edu/education/ccp/materials/linalg/leslie/lesl1.html

[7] Lu decomposition. ScienceDirect . (n.d.). Retrieved September 7, 2022, from https://www.sciencedirect.com/topics/mathematics/lu-decomposition

[8] LU Decomposition. University of California, Davis . (n.d.). Retrieved September 7, 2022, from https://www.math.ucdavis.edu/~linear/old/notes11.pdf

[9] LU factorization of a nonsingular matrix exists if and only if all leading principal submatrices are nonsingular. Mathematics Stack Exchange. (n.d.). Retrieved September 7, 2022, from https://math.stackexchange.com/questions/2951461/lu-factorization-of-a-nonsingular-matrix-exists-if-and-only-if-all-leading-princ Markov chains.

[10] Brilliant Math & Science Wiki. (n.d.). Retrieved September 7, 2022, from https://brilliant.org/wiki/markov-chains/

[11] math et al. (2015, December 25). Steady-state vectors for Markov chains | discrete mathematics. YouTube. Retrieved September 7, 2022, from https://www.youtube.com/watch?v=8noldJCb86Y

[12] Population Projection Matrix. Montana State University. (n.d.). Retrieved September 7, 2022, from https://www.montana.edu/screel/teaching/bioe-440r-521/documents/leslie.pdf

[13] Solving Systems of Linear Equations. Michigan Technological University. (n.d.). Retrieved September 7, 2022, from https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/INT-APP/CURVE-linear-system.html

[14] What does the steady state represent to a Markov chain? Mathematics Stack Exchange. (n.d.). Retrieved September 7, 2022, from https://math.stackexchange.com/questions/133214/what-does-the-steady-state-represent-to-a-markov-chain

[15] Whitt. (n.d.). Introduction to Markov Chains. Columbia University NYC. Retrieved September 7, 2022, from http://www.columbia.edu/~ww2040/4701Sum07/MarkovMouse.pdf

VIII. APPENDICES

Download the source codes for Problem 1-5

1. Go to: https://github.com/Zaphat/Linear-Algebra-Project
2. Select Code, click on Download ZIP and unzip the downloaded file.

Install Jupyter Notebook using Anaconda

1. Go to: https://www.anaconda.com/ and download the latest version
2. Install and open Anaconda, then launch Jupiter Notebook.

Install Python

1. Go to https://www.python.org/downloads/ and download the latest version.
2. Install and restart the computer.
3. Add Python to system PATH: https://www.javatpoint.com/how-to-set-python-path

Install Python libraries

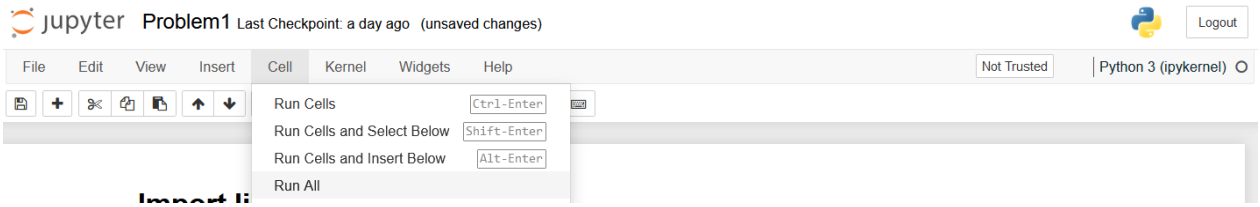
1. Run Command Prompt as administrator.
2. Type each line below to the Command Prompt and hit Enter:

```
pip install ipywidgets
pip install sympy
pip install ipython
pip install numpy
pip install matplotlib
```

Launch the code

1. From Start Menu browse for Anaconda folder and click on Jupyter Notebook, a web page will be opened in the default browser.
2. Navigate to where the code files are located, click on the file name to open.

3. From Cells select Run All, after that the program is ready to use.



Video tutorial: https://youtu.be/_G9RGoFD6QA

Further documentation for Sympy: https://docs.sympy.org/latest/index.html

Further documentation for ipywidgets: https://ipywidgets.readthedocs.io/en/stable/

Further documentation for matplotlib: https://matplotlib.org/stable/index.html