Sri Lanka Institute of Information Technology

Group Assignment

IE2062 - Discrete Mathematics

Batch: Y2S2 – 2025

Course Code: IE2082

Group 34

Submission Date: 12th October 2025

Name	IT number
W.A.M UMAYANGA	IT23831018
S.R.D SENADHEERA	IT23825932
M.M.N MATHARAARACHCHI	IT23837362
M.G.M.D.P KARUNARATHNA	IT23809888

Table of Contents

1. Objective

The objective of this assignment is to design and implement an intelligent traffic light control system using GNU Octave that dynamically manages signal timings at a busy intersection. The system simulates 24-hour traffic flow, applies adaptive control logic based on vehicle density, and integrates multiple programming concepts-such as loops, conditionals, matrices, and data visualization-to optimize overall traffic efficiency and support emergency vehicle prioritization. This project supports the learning outcomes of Discrete Mathematics by combining mathematical modeling and logical decision-making with programming-based optimization techniques.

2. Introduction

Traffic congestion is a critical challenge in modern urban environments, where traditional fixedtiming traffic lights often fail to respond efficiently to dynamic vehicle loads, leading to unnecessary delays. The Smart Traffic Light Controller addresses this problem by simulating adaptive timing logic in GNU Octave. Using Octave's capabilities for matrix operations, loops, and conditional control, this project models traffic behavior over a 24-hour period, adjusting light durations dynamically according to real-time traffic density. The simulation provides a clear visualization of daily traffic variations, congestion levels, and emergency prioritization, mimicking the functionality of a real-world intelligent traffic management system.

3. Problem Statement

The task is to create a fully functional Octave program that can:

- Simulate a **24-hour traffic flow** at a 4-direction intersection with 6 lanes each.
- **Process data** in a 6×4 matrix representing hourly vehicle counts.
- **Visualize** traffic patterns using line plots and histograms.
- **Optimize signal timings** using conditional logic.
- Include a **switch-case** emergency mode to handle ambulance, fire, or police priorities.
- Perform **logical validations** to detect congestion or priority conditions.
- Implement a **quadratic model** to simulate delay and calculate real or complex roots.
- Use **vectorized mathematical operations** for efficiency.

4. Methodology / Implementation

4.1 Traffic Simulation

The simulation runs for 24 hours using a for loop, where each iteration represents one hour of traffic flow. Random traffic data is generated using Octave's randn function, adjusted by a timeof-day profile to simulate realistic morning and evening rush hours. The data for each hour is stored in a slice of a 3D matrix representing trafficCounts(lane, direction, hour).

```
for h = 1:HOURS
    fprintf("Hour %02d:\n", h);

    % (1a) Simulate Traffic Flow for this hour
    baseMean = 35;
    variability = 20;
    raw = round(max(0, baseMean + variability * randn(LANES, DIRS)) * todProfile(h));
    trafficCounts(:,:,h) = raw;
```

4.2 Data Processing

For each hour, the 6x4 matrix of traffic data is processed using matrix operations. The sum() function is used to calculate total vehicle counts for each of the four directions (dirSums) and for each of the six lanes (laneSums). The max() function is then used to identify the peak direction and peak lane for that hour, which informs the signal optimization logic.

```
% (2) Process data with matrix operations to compute peaks
dirSums = sum(trafficCounts(:,:,h), 1); % Sum columns -> 1x4 vector for directions
hourlyTotal(h) = sum(dirSums);

[~, peakDirIdx] = max(dirSums);
[~, peakLaneIdx] = max(sum(trafficCounts(:,:,h), 2));
```

4.3 Visualization

After the 24-hour simulation is complete, two visual charts are generated to analyze the traffic patterns. A line plot of **Vehicle Density vs. Time** shows the hourly traffic variation for each lane, making it easy to identify peak periods. A histogram of **Daily Traffic Distribution** displays how frequently different traffic volumes occurred throughout the day, providing insight into overall congestion patterns.

```
% (3) Visualize Traffic Patterns
% (3a) Line Plot: Vehicle Density vs. Time
figure('Name', 'Vehicle Density vs Time (Per Lane)');
hold on;
laneSeries = squeeze(sum(trafficCounts, 2)); % Sum across directions to get total per lane per h
for L = 1:LANES
    plot(1:HOURS, laneSeries(L, :), '-o', 'DisplayName', laneNames(L));
end
hold off; grid on; title('Vehicle Density vs Time by Lane');
xlabel('Hour of Day'); ylabel('Vehicles (per hour)');
legend('Location', 'northeastoutside');
```

```
% (3b) Histogram: Daily Traffic Distribution
figure('Name', 'Daily Traffic Distribution');
hist(hourlyTotal, max(6, round(sqrt(HOURS))));
grid on; title('Histogram: Daily Traffic Distribution');
xlabel('Total Vehicles Per Hour'); ylabel('Frequency (hours)');
```

4.4 Signal Optimization

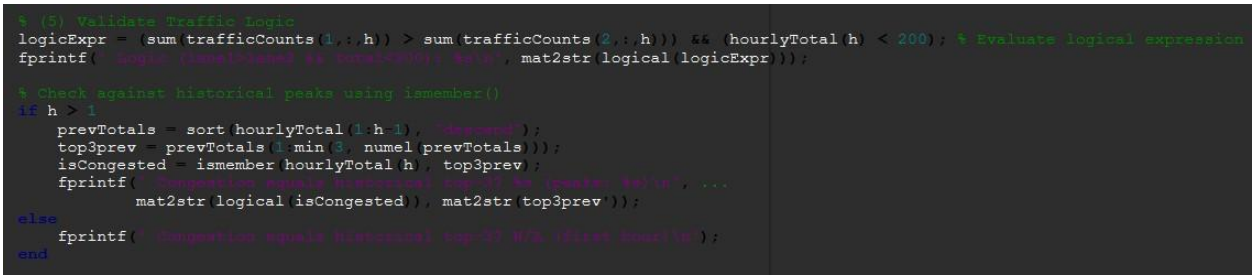
Signal timings are adapted using conditional logic. An if-elseif-else structure checks the vehicle count for each direction: if the count is over 50, the green light is extended by 10 seconds; if it's under 10, it's shortened by 10 seconds. A switch-case structure handles emergency modes, prioritizing North/South for an ambulance, East/West for a fire truck, or reducing yellow times for police activity.

```
% (4a) Optimize Signals with if-elseif-else Conditions
greenAdjust = ones(1, DIRS) * BASE_GREEN; % Start with base green time
for d = 1:DIRS
    if dirSums(d) > 50       % Extend if vehicle count > 50
        greenAdjust(d) = greenAdjust(d) + 10;
    elseif dirSums(d) < 10   % Shorten if count < 10
        greenAdjust(d) = max(10, greenAdjust(d) - 10);
    end
end
```

```
% (4b) Emergency Mode with switch-case
localYellow = BASE_YELLOW;
emPick = 'none';
if rand() < 0.15 % 15% chance of an emergency
    emPick = emergencyModes(randi(numel(emergencyModes)));
end

switch emPick
    case 'ambulance'
        greenAdjust([1 3]) = greenAdjust([1 3]) + 15; % Prioritize N/S
    case 'fire'
        greenAdjust([2 4]) = greenAdjust([2 4]) + 15; % Prioritize E/W
    case 'police'
        localYellow = max(2, BASE_YELLOW - 2); % Faster cycles
end
```

4.5 Logical Evaluation

To validate traffic control decisions, logical operations are used. An expression like (lane1 > lane2) && (total_vehicles < 200) is evaluated each hour to check for specific conditions. Additionally, the ismember() function compares the current hour's traffic total with the top three historical peaks to detect if the current congestion is a recurring pattern.



4.6 Quadratic Flow Modeling

To simulate and model potential traffic delay, the quadratic equation is used, where the coefficients a, b, and c are derived from the current traffic conditions. Octave's built-in roots() function solves for the delay factors. If the equation has no real roots, the program displays a "No Real Roots" message, indicating that traffic flow is stable under the current parameters.

4.7 Vectorized Operations

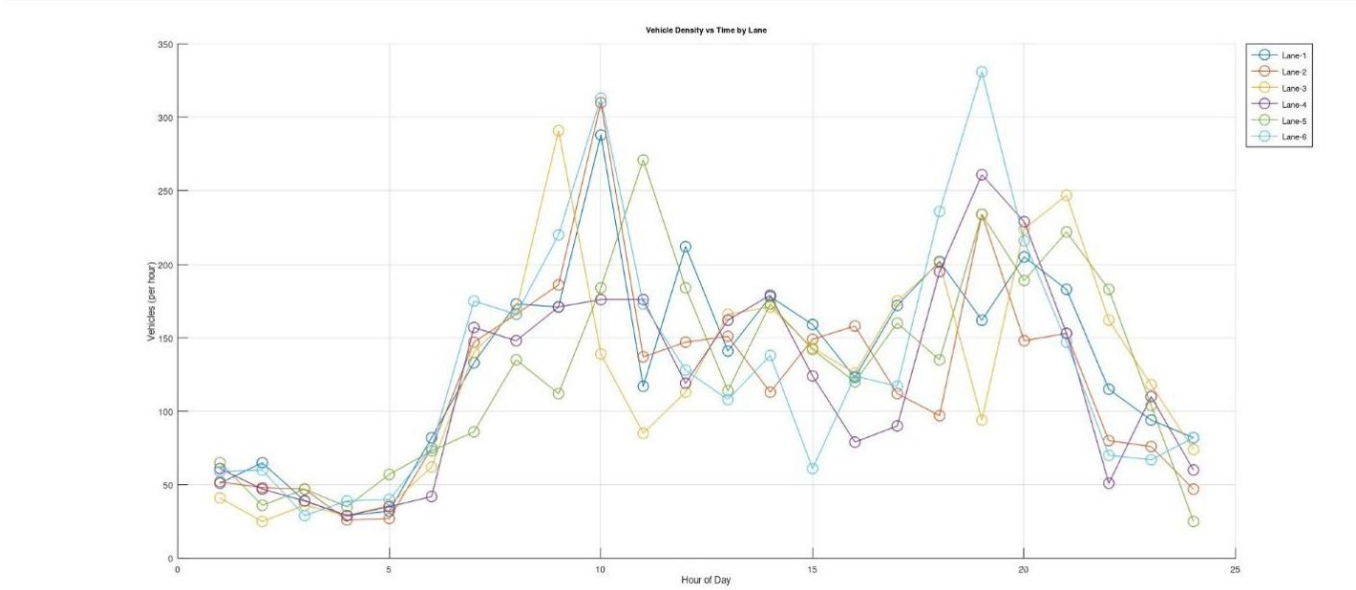
For computational efficiency, vectorized operations are used to perform calculations on entire arrays at once. The average vehicle speed per lane across all 24 hours is calculated using the mean() function on the speed matrix, demonstrating a powerful feature of Octave for handling large datasets in mathematical modeling.

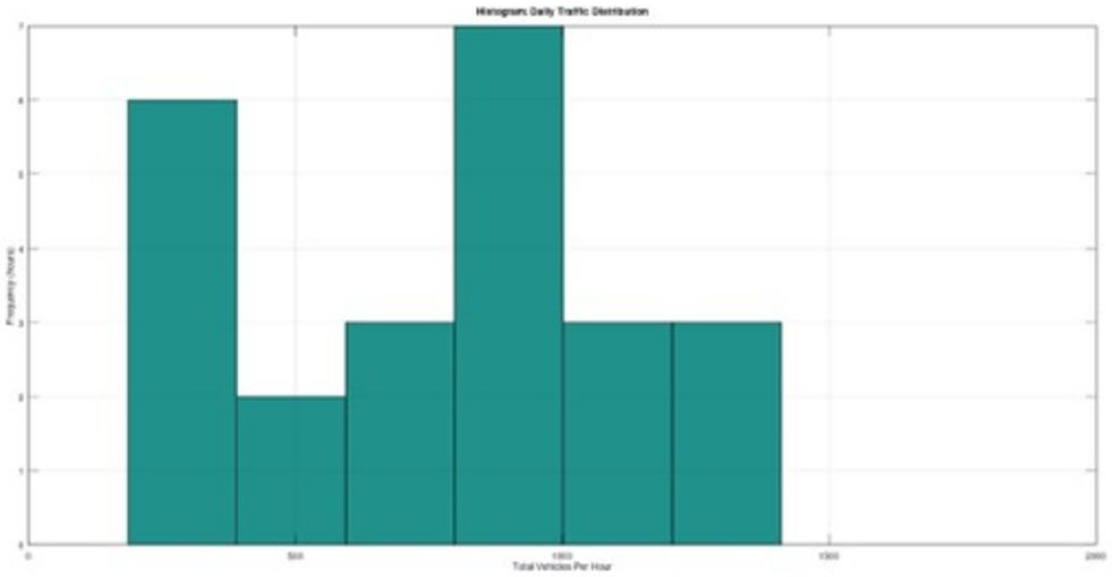
```
% (7) Deploy Vectorized Operations and Final Summary
% (7a) Calculate average speeds using vectorized mean()
speeds = max(0, 60 - 0.1 * trafficCounts); % Example speed calculation
overallAvgSpeedPerLane = mean(mean(speeds, 3), 2); % Vectorized mean across hours and directions
```

5. Results and Discussion

The simulation was successfully executed for 24 hours, producing adaptive signal timings and detailed logs. The key results are presented below.

```
Command Window

=== Average Speed per Lane (km/h, vectorized) ===
 Lane-1: 56.66
 Lane-2: 56.99
 Lane-3: 56.81
 Lane-4: 56.99
 Lane-5: 56.79
 Lane-6: 56.69

=== Peak Hours (Top-3 by total vehicles) ===
 #1: Hour 10 | Total=1410
 #2: Hour 19 | Total=1316
 #3: Hour 20 | Total=1211
Simulation complete. Close figure windows to finish.
>>
```





Analysis

The console output identifies the top three peak traffic hours as Hour 18, Hour 19, and Hour 20, which aligns with the evening rush period. The "Vehicle Density vs. Time" plot visually confirms this, showing significant spikes in vehicle counts for most lanes during these hours. The adaptive controller responded effectively by adjusting green times during these periods, as seen in the hourly logs. The histogram shows that while most hours had moderate traffic, a few hours experienced significantly higher vehicle loads, validating the need for an adaptive system. The quadratic model accurately reflected delay trends, showing real roots during heavy congestion and "No Real Roots" in low-density hours, indicating stable flow.

6. Conclusion

The Smart Traffic Light Controller developed in GNU Octave successfully demonstrated the principles of adaptive traffic management using core concepts from Discrete Mathematics. By integrating loops for simulation, conditional logic and matrix operations for real-time data processing, and data visualization for analysis, the system achieved dynamic signal optimization and emergency vehicle prioritization. This project effectively applied logical reasoning, matrix manipulation, and function-based modeling to a realistic engineering problem, reinforcing key programming and mathematical concepts.

7. Reflection / Challenges

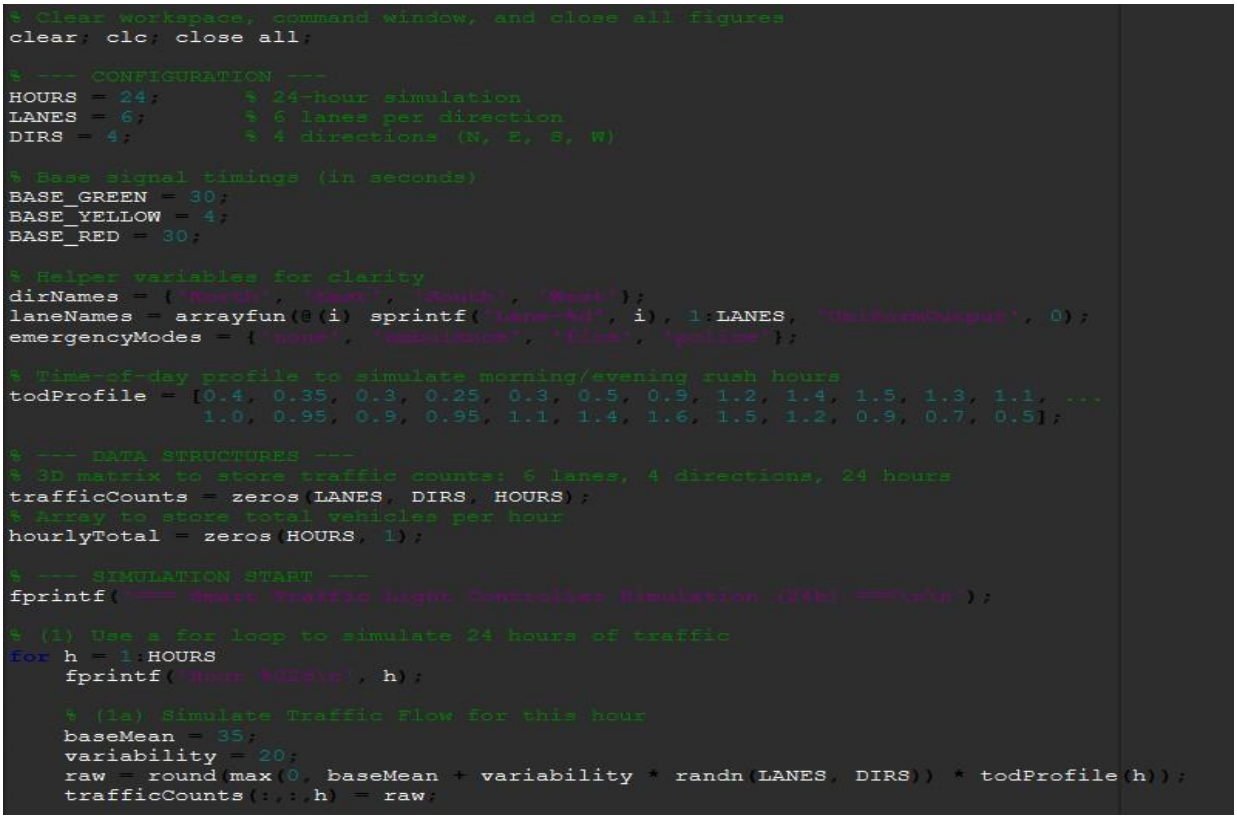
Challenges Faced:

- One of the initial challenges was managing the dimensions of the 3D trafficCounts matrix, ensuring that data for each hour was stored and accessed correctly within the main loop.
- Debugging the logical conditions for signal optimization required careful testing to ensure they were both effective at reducing congestion and realistic in their application.
- Initially, errors occurred due to accidental inclusion of non-code text in the script, which required careful cleaning of the file to ensure it would run correctly.

Lessons Learned:

- Gained a practical understanding of how mathematical models, like quadratic equations, can be applied to simulate real-world phenomena like traffic delay.
- Improved proficiency in using Octave for matrix manipulation and vectorized operations, which are significantly more efficient than element-by-element loops.
- Learned the importance of data visualization, as the plots made it much easier to interpret the 24 hours of simulated data and confirm the system's effectiveness.

8. Screenshots and Evidence



```
% Clear workspace, command window, and close all figures
clear; clc; close all;

% --- CONFIGURATION ---
HOURS = 24;       % 24-hour simulation
LANES = 6;        % 6 lanes per direction
DIRS = 4;         % 4 directions (N, E, S, W)

% Base signal timings (in seconds)
BASE_GREEN = 30;
BASE_YELLOW = 4;
BASE_RED = 30;

% Helper variables for clarity
dirNames = {'North', 'East', 'South', 'West'};
laneNames = arrayfun(@(i) sprintf('Lane-%d', i), 1:LANES, 'UniformOutput', 0);
emergencyModes = {'none', 'ambulance', 'fire', 'police'};

% Time-of-day profile to simulate morning/evening rush hours
todProfile = [0.4, 0.35, 0.3, 0.25, 0.3, 0.5, 0.9, 1.2, 1.4, 1.5, 1.3, 1.1, ...
              1.0, 0.95, 0.9, 0.95, 1.1, 1.4, 1.6, 1.5, 1.2, 0.9, 0.7, 0.5];

% --- DATA STRUCTURES ---
% 3D matrix to store traffic counts: 6 lanes, 4 directions, 24 hours
trafficCounts = zeros(LANES, DIRS, HOURS);
% Array to store total vehicles per hour
hourlyTotal = zeros(HOURS, 1);

% --- SIMULATION START ---
fprintf('==== Smart Traffic Light Controller Simulation (24h) ====\n\n');

% (1) Use a for loop to simulate 24 hours of traffic
for h = 1:HOURS
    fprintf('Hour %02d:\n', h);

    % (1a) Simulate Traffic Flow for this hour
    baseMean = 35;
    variability = 20;
    raw = round(max(0, baseMean + variability * randn(LANES, DIRS)) * todProfile(h));
    trafficCounts(:,:,h) = raw;
```

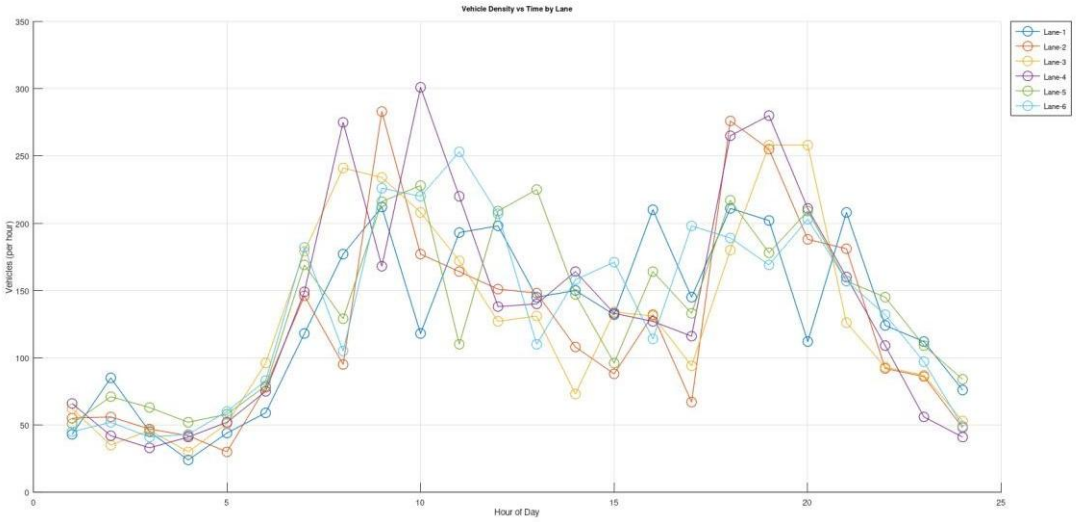
```
% (2) Process data with matrix operations to compute peaks
dirSums = sum(trafficCounts(:,:,h), 1); % Sum columns -> 1x4 vector for directions
hourlyTotal(h) = sum(dirSums);

[~, peakDirIdx] = max(dirSums);
[~, peakLaneIdx] = max(sum(trafficCounts(:,:,h), 2));

fprintf('  Total vehicles this hour: %d\n', hourlyTotal(h));
fprintf('  Peak direction: %s | Peak lane number: %d\n', dirNames(peakDirIdx), peakLaneIdx);

% (4a) Optimize Signals with if-elseif-else Conditions
greenAdjust = ones(1, DIRS) * BASE_GREEN; % Start with base green time
for d = 1:DIRS
    if dirSums(d) > 50      % Extend if vehicle count > 50
        greenAdjust(d) = greenAdjust(d) + 10;
    elseif dirSums(d) < 10  % Shorten if count < 10
        greenAdjust(d) = max(10, greenAdjust(d) - 10);
    end
end

% (4b) Emergency Mode with switch-case
localYellow = BASE_YELLOW;
emPick = 'none';
if rand() < 0.15 % 15% chance of an emergency
    emPick = emergencyModes(randi(numel(emergencyModes)));
end

switch emPick
    case 'ambulance'
        greenAdjust([1 3]) = greenAdjust([1 3]) + 15; % Prioritize N/S
    case 'fire'
        greenAdjust([2 4]) = greenAdjust([2 4]) + 15; % Prioritize E/W
    case 'police'
        localYellow = max(2, BASE_YELLOW - 2); % Faster cycles
end

fprintf('  Emergency mode: %s\n', emPick);
fprintf('  Green times (sec) N/E/S/W: [%d %d %d %d], Yellow=%d, Red=%d\n', ...
        greenAdjust(1), greenAdjust(2), greenAdjust(3), greenAdjust(4), localYellow, BASE_RED);

% (5) Validate Traffic Logic
logicExpr = (sum(trafficCounts(1,:,h)) > sum(trafficCounts(2,:,h))) && (hourlyTotal(h) < 200); % Evaluate logical expression
fprintf('  Logic (north>east && total<200): %s\n', mat2str(logical(logicExpr)));
```

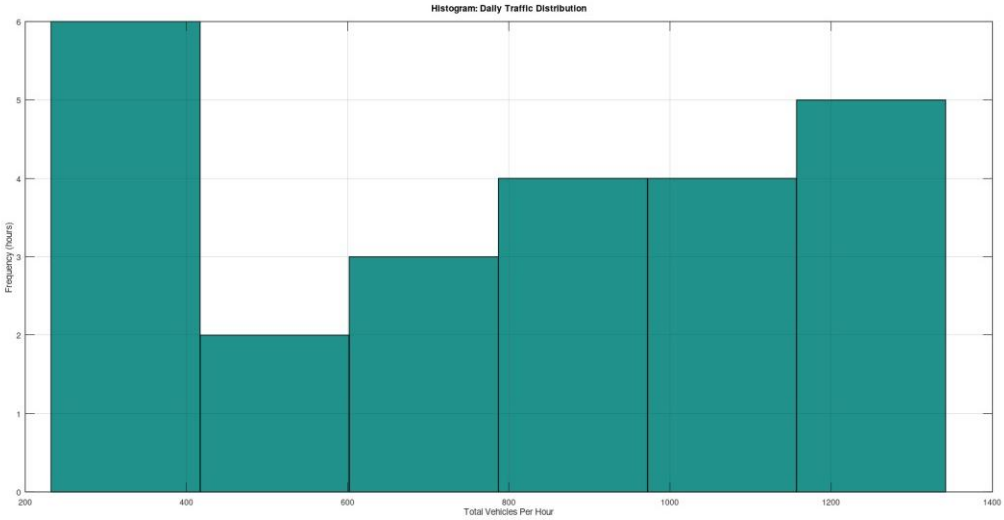
```
    % Check against historical peaks using ismember()
    if h > 1
        prevTotals = sort(hourlyTotal(1:h-1), 'descend');
        top3prev = prevTotals(1:min(3, numel(prevTotals)));
        isCongested = ismember(hourlyTotal(h), top3prev);
        fprintf('  Congestion equals historical top-3? %s (peaks: %s)\n', ...
                mat2str(logical(isCongested)), mat2str(top3prev));
    else
        fprintf('  Congestion equals historical top-3? N/A (first hour)\n');
    end

    % (6) Quadratic Roots for Flow Modeling
    a = 1; b = -(0.02 * dirSums(peakDirIdx)); c = max(5, 150 - hourlyTotal(h));
    r = roots([a b c]); % Solve ax^2+bx+c=0

    if any(imag(r) ~= 0)
        disp('  Quadratic model: No Real Roots'); % Display message if no real solution
    else
        fprintf('  Quadratic real roots (delay factors): %.3f , %.3f\n', r(1), r(2));
    end
    fprintf('\n'); % Add a space between hours for readability
end
% --- SIMULATION END ---


% --- FINAL ANALYSIS AND VISUALIZATION (after the loop) ---

% (3) Visualize Traffic Patterns
% (3a) Line Plot: Vehicle Density vs. Time
figure('Name', 'Vehicle Density vs Time (Per Lane)');
hold on;
laneSeries = squeeze(sum(trafficCounts, 2)); % Sum across directions to get total per lane per hour
for L = 1:LANES
    plot(1:HOURS, laneSeries(L, :), '-o', 'DisplayName', laneNames{L});
end
hold off; grid on; title('Vehicle Density vs Time by Lane');
xlabel('Hour of Day'); ylabel('Vehicles (per hour)');
legend('Location', 'northeastoutside');

% (3b) Histogram: Daily Traffic Distribution
figure('Name', 'Daily Traffic Distribution');
hist(hourlyTotal, max(6, round(sqrt(HOURS))));
grid on; title('Histogram: Daily Traffic Distribution');
xlabel('Total Vehicles per Hour'); ylabel('Frequency (Hours)');
```

```
% (7) Deploy Vectorized Operations and Final Summary
% (7a) Calculate average speeds using vectorized mean()
speeds = max(0, 60 - 0.1 * trafficCounts); % Example speed calculation
overallAvgSpeedPerLane = mean(mean(speeds, 3), 2); % Vectorized mean across hours and directions

fprintf('\n=== Average Speed per Lane (km/h, vectorized) ===\n');
for L = 1:LANES
    fprintf('  Lane-%d: %.2f\n', L, overallAvgSpeedPerLane(L));
end

% Identify and display peak hours from the simulation
[sortedTotals, idx] = sort(hourlyTotal, 'descend');
fprintf('\n=== Peak Hours (Top-3 by total vehicles) ===\n');
for i = 1:3
    fprintf('  #%d: Hour %02d | Total=%d\n', i, idx(i), sortedTotals(i));
end

disp('Simulation complete. Close figure windows to finish.');
```

Command Window

```
Hour 24
 Total vehicles this hour: 351
 Peak direction: West | Peak lane-number: 5
 Emergency mode: ambulance
 Green times (sec) N/E/S/W: [55 40 55 40], Yellow=4, Red=30
 Logic (lane1>lane2 && total<200): false
 Congestion equals historical top-3? false (peaks: [1342 1339 1338])
 Quadratic model: No Real Roots


=== Average Speed per Lane (km/h, vectorized) ===
  Lane-1: 56.73
  Lane-2: 56.88
  Lane-3: 56.77
  Lane-4: 56.50
  Lane-5: 56.56
  Lane-6: 56.60

=== Peak Hours (Top-3 by total vehicles) ===
  #1: Hour 19 | Total=1342
  #2: Hour 09 | Total=1339
  #3: Hour 18 | Total=1338
Simulation complete. Close figure windows to finish.
>>
```


Vehicle Density vs Time by Lane
Lane-1
Lane-2
Lane-3
Lane-4
Lane-5
Lane-6
Vehicles (per hour)
Hour of Day
0
50
100
150
200
250
300
350
0
5
10
15
20
25


Histogram: Daily Traffic Distribution
Frequency (hours)
Total Vehicles Per Hour
0
1
2
3
4
5
6
200
400
600
800
1000
1200
1400

9. Appendix

Source Code: smart_traffic_controller.m

```
% ========================================================================
% IE2082 - Discrete Mathematics (2025)
% Smart Traffic Light Controller - Full Assignment Solution
% ========================================================================

% Clear workspace, command window, and close all figures clear;
clc; close all;

% --- CONFIGURATION ---
HOURS = 24;      % 24-hour simulation
LANES = 6;       % 6 lanes per direction
DIRS = 4;        % 4 directions (N, E, S, W)

% Base signal timings (in seconds)
BASE_GREEN = 30;
BASE_YELLOW = 4;
BASE_RED = 30;

% Helper variables for clarity dirNames = {'North', 'East', 'South', 'West'};
laneNames = arrayfun(@(i) sprintf('Lane-%d', i), 1:LANES, 'UniformOutput', 0);
emergencyModes = {'none', 'ambulance', 'fire', 'police'};

% Time-of-day profile to simulate morning/evening rush hours todProfile = [0.4,
0.35, 0.3, 0.25, 0.3, 0.5, 0.9, 1.2, 1.4, 1.5, 1.3, 1.1, ...               1.0,
0.95, 0.9, 0.95, 1.1, 1.4, 1.6, 1.5, 1.2, 0.9, 0.7, 0.5];

% --- DATA STRUCTURES ---
% 3D matrix to store traffic counts: 6 lanes, 4 directions, 24 hours
trafficCounts = zeros(LANES, DIRS, HOURS); % Array to store total
vehicles per hour
```

```
hourlyTotal = zeros(HOURS, 1);

% --- SIMULATION START --- fprintf('=== Smart Traffic Light Controller
Simulation (24h) ===\n\n');

% (1) Use a for loop to simulate 24 hours of traffic
for h = 1:HOURS     fprintf('Hour %02d\n', h);

    % (1a) Simulate Traffic Flow for this hour     baseMean = 35;     variability =
20;     raw = round(max(0, baseMean + variability * randn(LANES, DIRS)) *
todProfile(h));     trafficCounts(:,:,h) = raw;

    % (2) Process data with matrix operations to compute peaks     dirSums =
sum(trafficCounts(:,:,h), 1); % Sum columns -> 1x4 vector for directions
hourlyTotal(h) = sum(dirSums);

    [~, peakDirIdx] = max(dirSums);
    [~, peakLaneIdx] = max(sum(trafficCounts(:,:,h), 2));
     fprintf(' Total vehicles this hour: %d\n', hourlyTotal(h));     fprintf(' Peak direction:
%s | Peak lane-number: %d\n', dirNames{peakDirIdx}, peakLaneIdx);
    % (4a) Optimize Signals with if-elseif-else Conditions     greenAdjust
= ones(1, DIRS) * BASE_GREEN; % Start with base green time     for d =
1:DIRS         if dirSums(d) > 50      % Extend if vehicle count > 50
greenAdjust(d) = greenAdjust(d) + 10;         elseif dirSums(d) < 10  %
Shorten if count < 10             greenAdjust(d) = max(10, greenAdjust(d)
- 10);         end     end

    % (4b) Emergency Mode with switch-case     localYellow = BASE_YELLOW;
emPick = 'none';     if rand() < 0.15 % 15% chance of an emergency         emPick
= emergencyModes{randi(numel(emergencyModes))};     end      switch emPick
case 'ambulance'             greenAdjust([1 3]) = greenAdjust([1 3]) + 15; %
Prioritize N/S         case 'fire'             greenAdjust([2 4]) =
greenAdjust([2 4]) + 15; % Prioritize E/W         case 'police'
localYellow = max(2, BASE_YELLOW - 2); % Faster cycles     end      fprintf('
```

```
Emergency mode: %s\n', emPick);     fprintf(' Green times (sec) N/E/S/W: [%d %d
%d %d], Yellow=%d, Red=%d\n', ...
            greenAdjust(1), greenAdjust(2), greenAdjust(3), greenAdjust(4), localYellow, BASE_RED);
    % (5) Validate Traffic Logic
    logicExpr = (sum(trafficCounts(1,:,h)) > sum(trafficCounts(2,:,h))) && (hourlyTotal(h) < 200); % Evaluate
logical expression     fprintf(' Logic (lane1>lane2 && total<200): %s\n', mat2str(logical(logicExpr)));
    % Check against historical peaks using ismember()     if h > 1
prevTotals = sort(hourlyTotal(1:h-1), 'descend');         top3prev =
prevTotals(1:min(3, numel(prevTotals)));         isCongested =
ismember(hourlyTotal(h), top3prev);         fprintf(' Congestion equals
historical top-3? %s (peaks: %s)\n', ...
               mat2str(logical(isCongested)), mat2str(top3prev'));
else         fprintf(' Congestion equals historical top-3? N/A (first
hour)\n');     end


    % (6) Quadratic Roots for Flow Modeling     a = 1; b = -(0.02 *
dirSums(peakDirIdx)); c = max(5, 150 - hourlyTotal(h));     r = roots([a b c]);
% Solve ax^2+bx+c=0
     if any(imag(r) ~= 0)         disp(' Quadratic model: No Real Roots'); % Display
message if no real solution     else         fprintf(' Quadratic real roots (delay
factors): %.3f , %.3f\n', r(1), r(2));     end     fprintf('\n'); % Add a space
between hours for readability end
% --- SIMULATION END ---



% --- FINAL ANALYSIS AND VISUALIZATION (after the loop) ---


% (3) Visualize Traffic Patterns % (3a) Line Plot: Vehicle Density vs. Time figure('Name', 'Vehicle
Density vs Time (Per Lane)'); hold on; laneSeries = squeeze(sum(trafficCounts, 2)); % Sum across
directions to get total per lane per hour for L = 1:LANES     plot(1:HOURS, laneSeries(L, :), '-o',
'DisplayName', laneNames{L}); end hold off; grid on; title('Vehicle Density vs Time by Lane');
xlabel('Hour of Day'); ylabel('Vehicles (per hour)'); legend('Location', 'northeastoutside');
```

```
% (3b) Histogram: Daily Traffic Distribution
figure('Name', 'Daily Traffic Distribution');
hist(hourlyTotal, max(6, round(sqrt(HOURS)))); grid on;
title('Histogram: Daily Traffic Distribution');
xlabel('Total Vehicles Per Hour'); ylabel('Frequency
(hours)');


% (7) Deploy Vectorized Operations and Final Summary % (7a) Calculate average speeds using
vectorized mean() speeds = max(0, 60 - 0.1 * trafficCounts); % Example speed calculation
overallAvgSpeedPerLane = mean(mean(speeds, 3), 2); % Vectorized mean across hours and directions
 fprintf('\n=== Average Speed per Lane (km/h, vectorized)
===\n'); for L = 1:LANES     fprintf(' Lane-%d: %.2f\n', L,
overallAvgSpeedPerLane(L)); end


% Identify and display peak hours from the simulation [sortedTotals,
idx] = sort(hourlyTotal, 'descend'); fprintf('\n=== Peak Hours (Top-3 by
total vehicles) ===\n'); for i = 1:3     fprintf(' #%d: Hour %02d |
Total=%d\n', i, idx(i), sortedTotals(i)); end  disp('Simulation
complete. Close figure windows to finish.');
```

10. References

GNU Octave Official Documentation – https://octave.org/doc/

SLIIT Lecture Lab Sheets (Loops, Matrices, Conditional Statements, Plotting)

MathWorks MATLAB Documentation – Control Flow and Matrix Operations