UNIVERSITY of SAN CARLOS SCIENTIA • VIRTUS • DEVOTIO	Author Paul Emmanuel G. Corsino Klinnsonveins Yee	Project Title Traffic Flow Simulation using the Jacobi Method	Date 12/18/2023
		Instructor Engr. Alvin Macapagal	Class Schedule MW 10:30 AM – 12:00 PM

CPE 3108: Programming Project

Traffic Flow Simulation using the Jacobi Method

Table of Contents

List of Figures

1 Introduction

1.1 Project Description

The Traffic Flow Simulation using the Jacobi Method is a Java Swing application designed to model and simulate traffic flow through a network of junctions. The Jacobi method is employed to iteratively compute traffic flow distribution across the junctions based on user-provided connectivity matrices, external forces, and initial guesses. The simulation helps analyze and understand the dynamic behavior of traffic under varying conditions.

1.2 Features

1. ***User-friendly GUI:*** A graphical user interface allows users to input the number of traffic junctions, connectivity matrix (A), external forces (b), and initial guess vector (x).
2. ***Iterative Simulation:*** Utilizes the Jacobi Method for solving the traffic flow equations iteratively, providing insights into the evolution of traffic conditions over multiple iterations.
3. ***Input Guidance:*** Offers a guide tab explaining the input variables and their significance to help users input accurate data.
4. ***Tabbed Interface:*** Organizes information into tabs, including input guidance, input variables, and iteration results, for a clear and structured user experience.
5. ***Convergence Check:*** Verifies the convergence of the Jacobi method and notifies users if the simulation reaches a solution or exceeds the maximum iterations.

1.3 Limitations

1. ***Simplicity of Model:*** The simulation assumes a simplified traffic flow model and may not capture all real-world complexities.
2. ***Numeric Stability:*** The Jacobi Method may not converge for certain input configurations, and the application provides a maximum of 100 iteration limit to prevent infinite loops.
3. ***Limited Visualization:*** The application focuses on textual output, and graphical visualization of traffic flow patterns is not included.

1.4 Input

1. ***Number of Junctions (n):*** User input specifying the number of traffic junctions in the network.
2. ***Connectivity Matrix (A):*** User-provided matrix representing the traffic flow relationships between junctions.
3. ***External Forces (b):*** Input vector representing external traffic forces at each junction.
4. ***Initial Guess (x):*** Initial guess vector representing the initial traffic conditions at each junction.

1.5 Output

1. ***Traffic Flow Results:*** Textual output displaying the traffic flow at each junction after the simulation process.
2. ***Iteration Results:*** Tabulated results showcasing the evolution of traffic flow values over each iteration.
3. ***Convergence Status:*** Indicates whether the Jacobi method converged within the specified maximum iterations or not.

2 Programming Algorithm

2.1 TrafficFlowSimulation (Main)

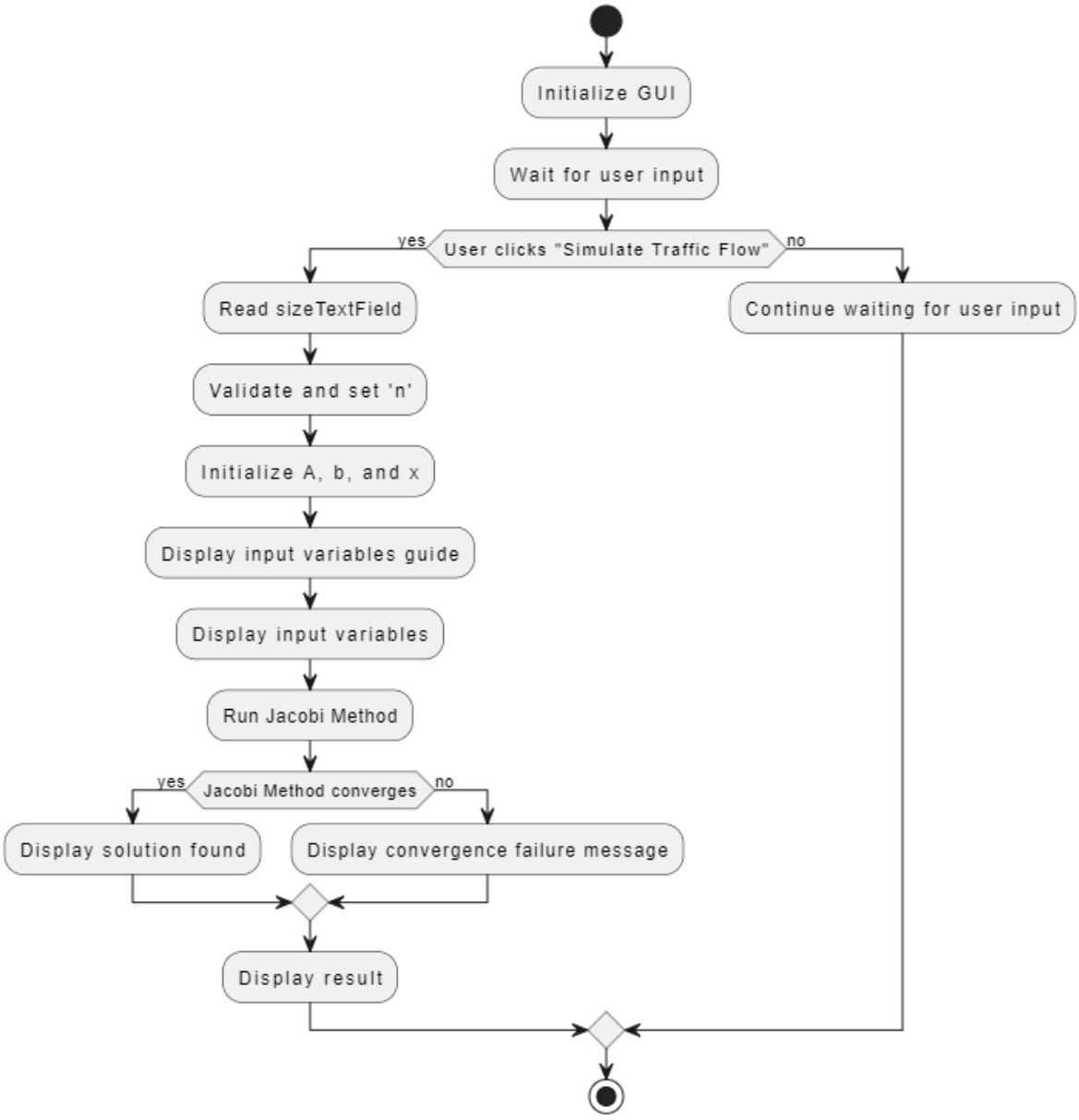


Figure 2.1a.TrafficFlowSimulation (Main) Flowchart

2.2 Simulate Traffic Flow

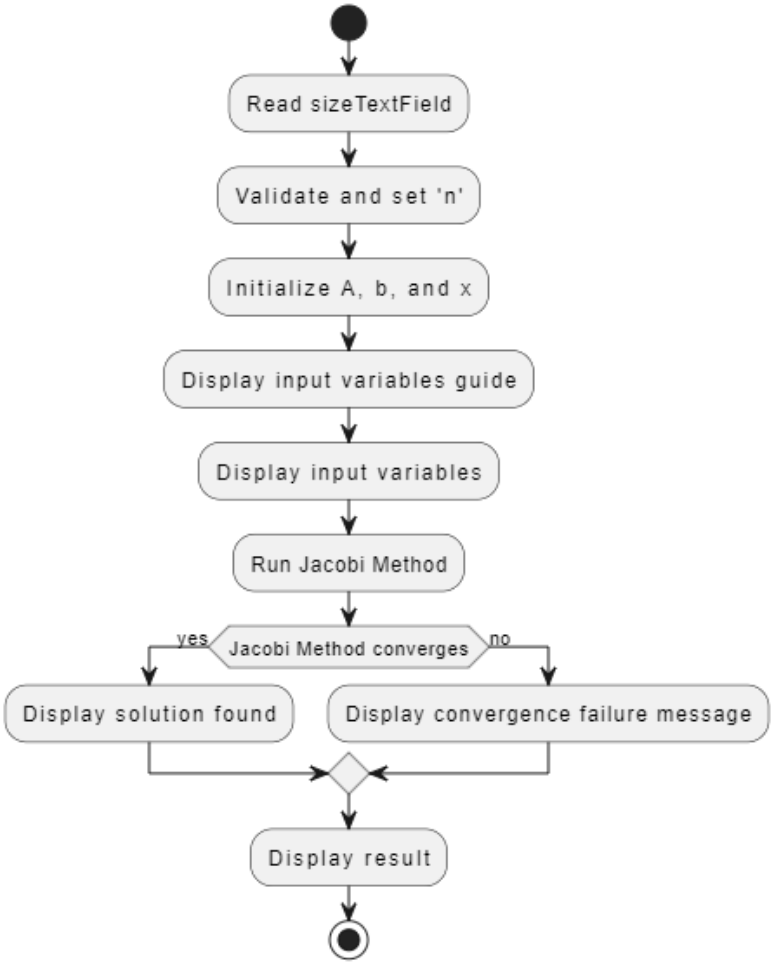


Figure 2.1b.Simulate Traffic Flow Flowchart

2.3 Initialize GUI and User Input Handling

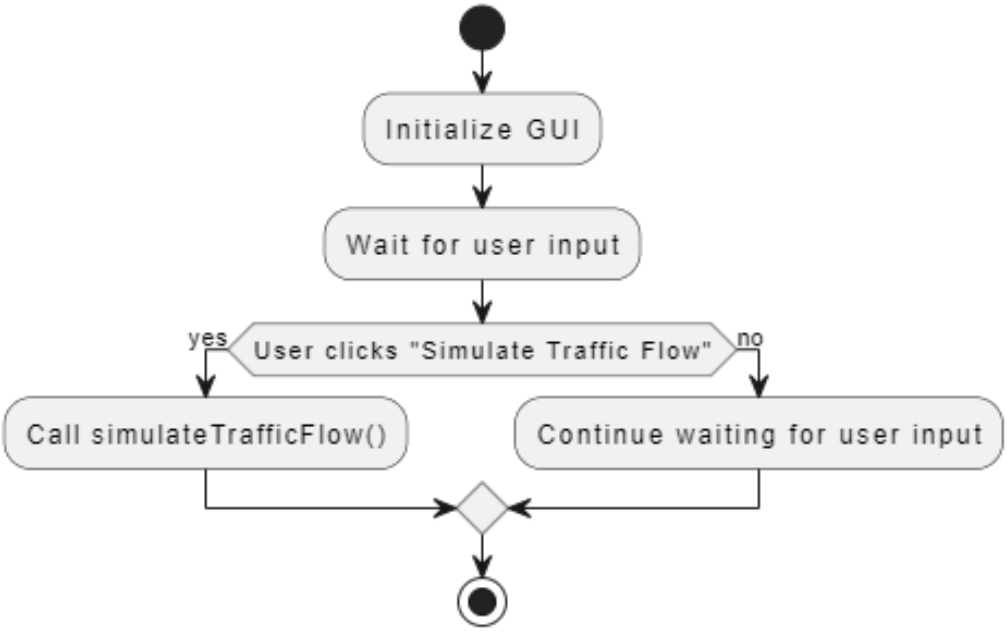
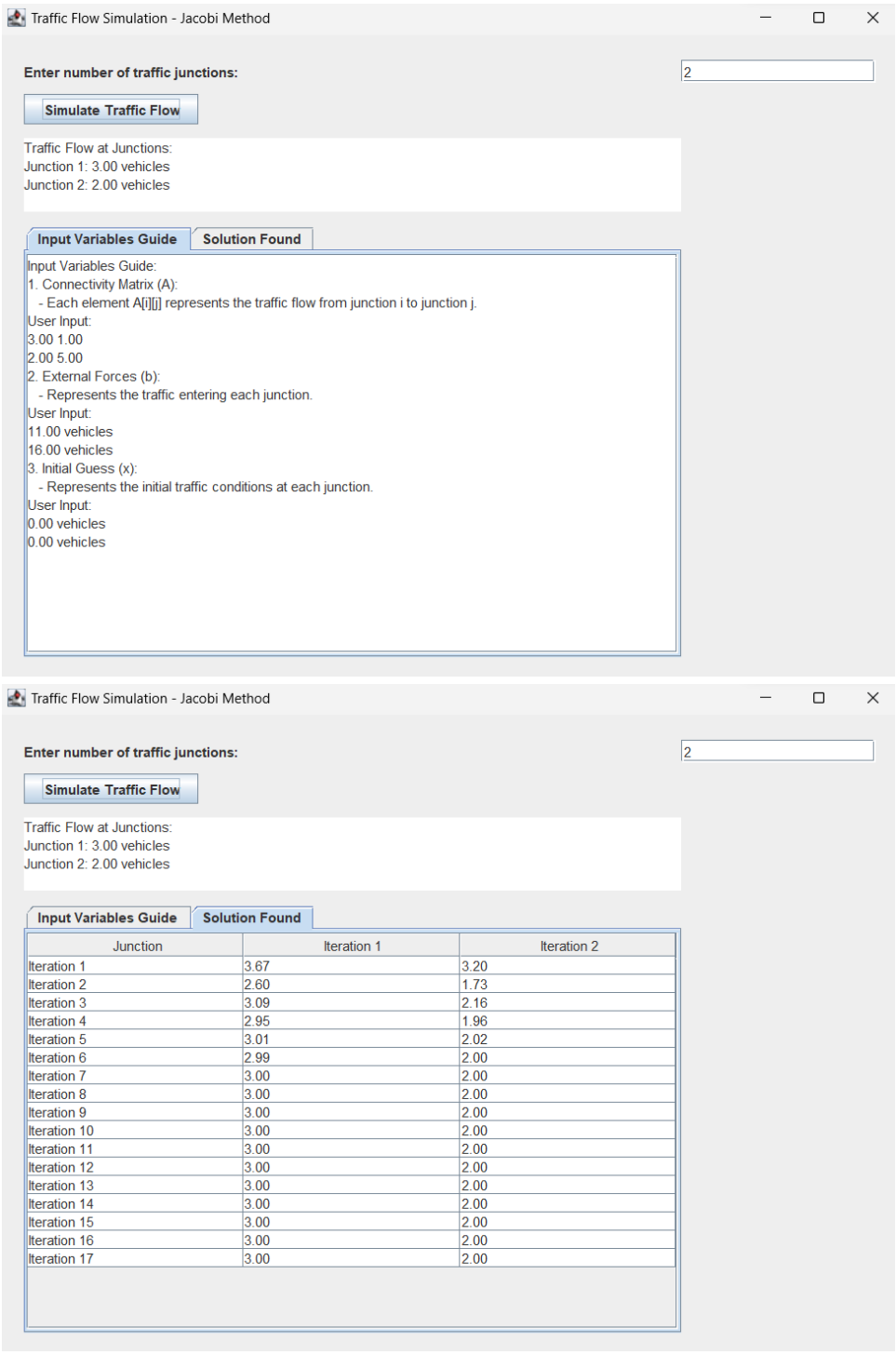


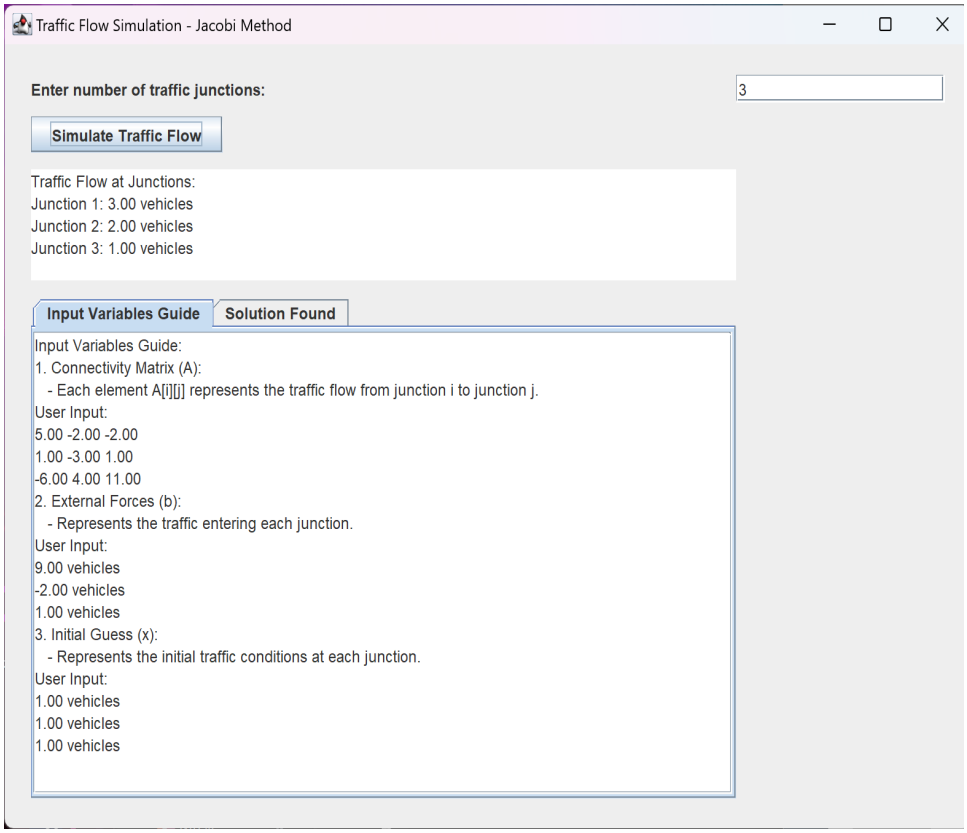
Figure 2.1c.Initialize GUI and User Input Handling Flowchart

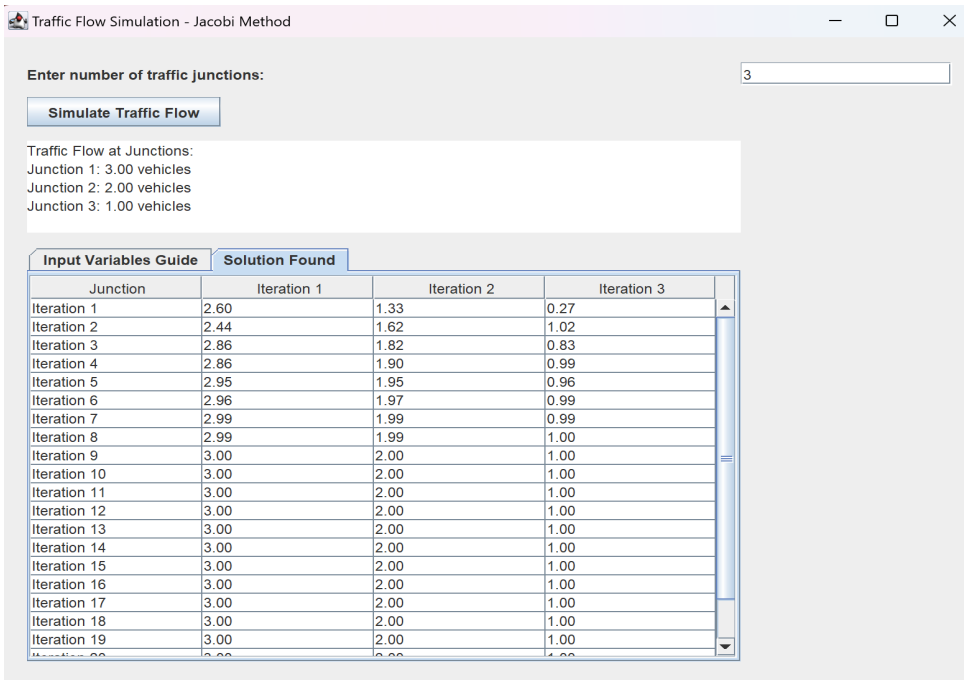
3 Sample Input-Output



Junction	Iteration 1	Iteration 2
Iteration 1	3.67	3.20
Iteration 2	2.60	1.73
Iteration 3	3.09	2.16
Iteration 4	2.95	1.96
Iteration 5	3.01	2.02
Iteration 6	2.99	2.00
Iteration 7	3.00	2.00
Iteration 8	3.00	2.00
Iteration 9	3.00	2.00
Iteration 10	3.00	2.00
Iteration 11	3.00	2.00
Iteration 12	3.00	2.00
Iteration 13	3.00	2.00
Iteration 14	3.00	2.00
Iteration 15	3.00	2.00
Iteration 16	3.00	2.00
Iteration 17	3.00	2.00

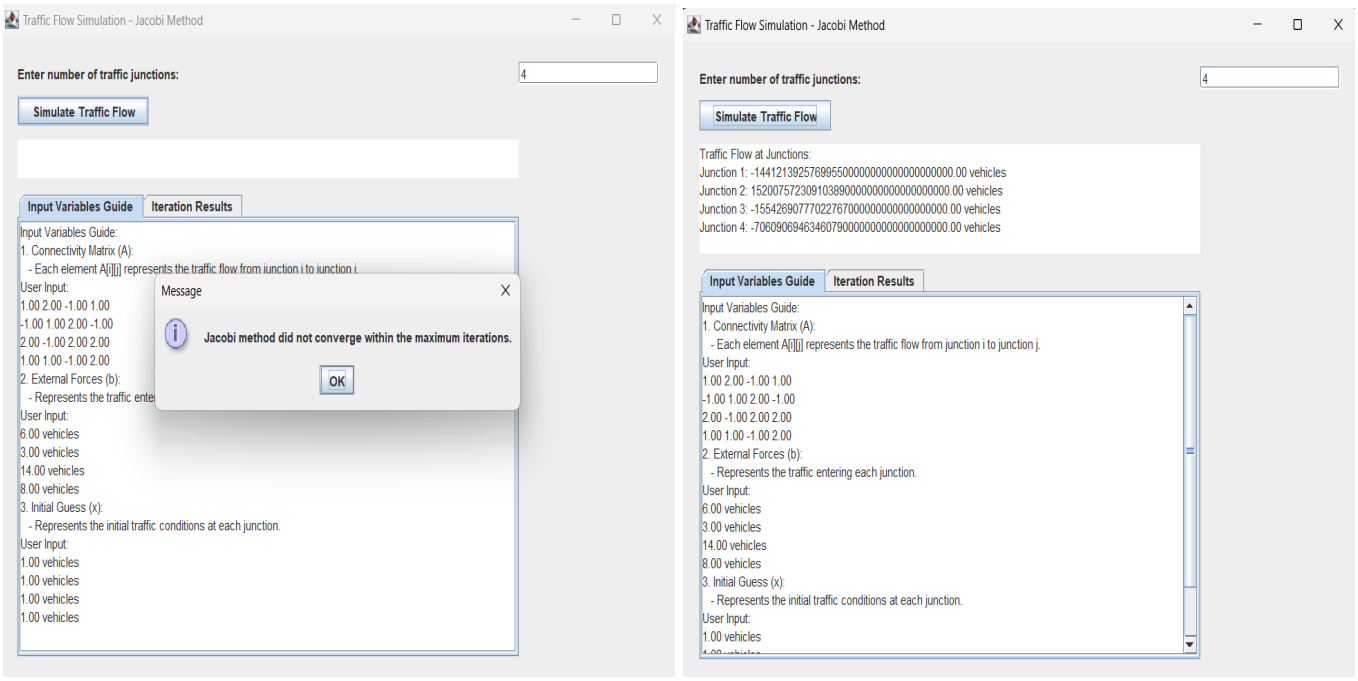
Figure 3a. 2 Unknowns

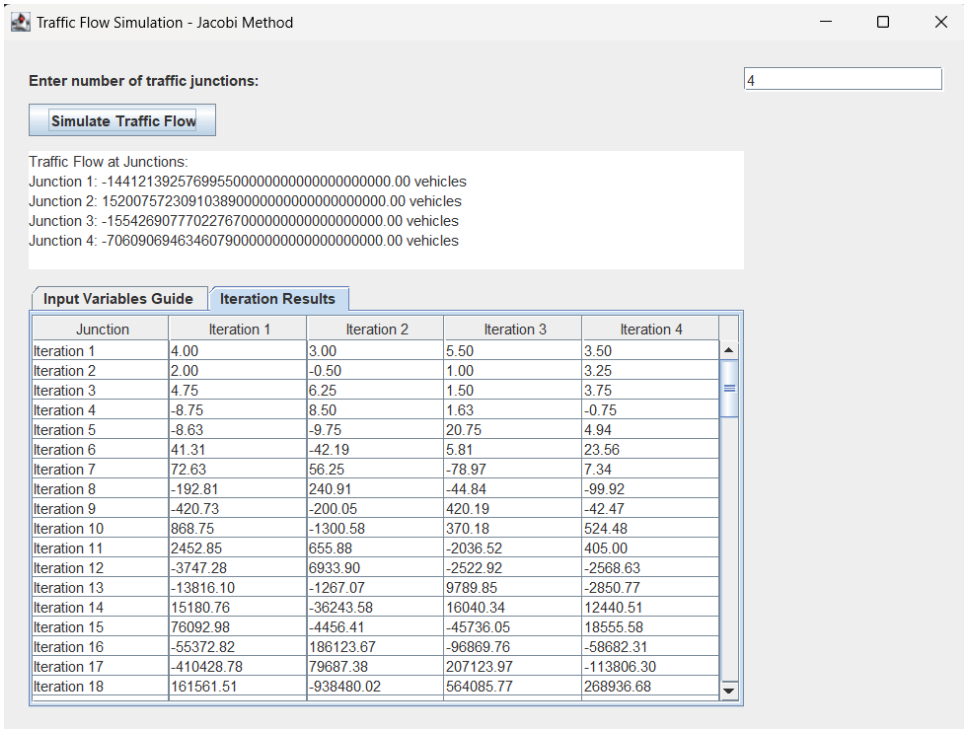




Junction	Iteration 1	Iteration 2	Iteration 3
Iteration 1	2.60	1.33	0.27
Iteration 2	2.44	1.62	1.02
Iteration 3	2.86	1.82	0.83
Iteration 4	2.86	1.90	0.99
Iteration 5	2.95	1.95	0.96
Iteration 6	2.96	1.97	0.99
Iteration 7	2.99	1.99	0.99
Iteration 8	2.99	1.99	1.00
Iteration 9	3.00	2.00	1.00
Iteration 10	3.00	2.00	1.00
Iteration 11	3.00	2.00	1.00
Iteration 12	3.00	2.00	1.00
Iteration 13	3.00	2.00	1.00
Iteration 14	3.00	2.00	1.00
Iteration 15	3.00	2.00	1.00
Iteration 16	3.00	2.00	1.00
Iteration 17	3.00	2.00	1.00
Iteration 18	3.00	2.00	1.00
Iteration 19	3.00	2.00	1.00

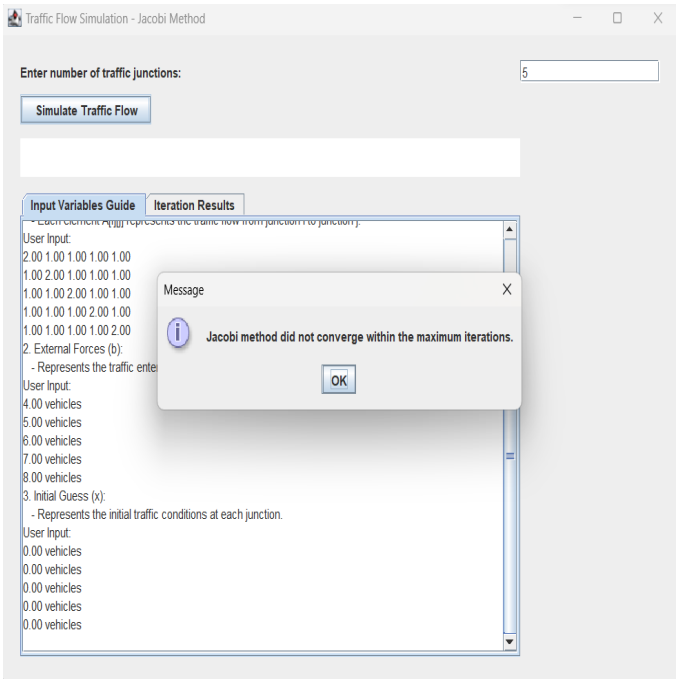
Figure 3b. 3 Unknowns

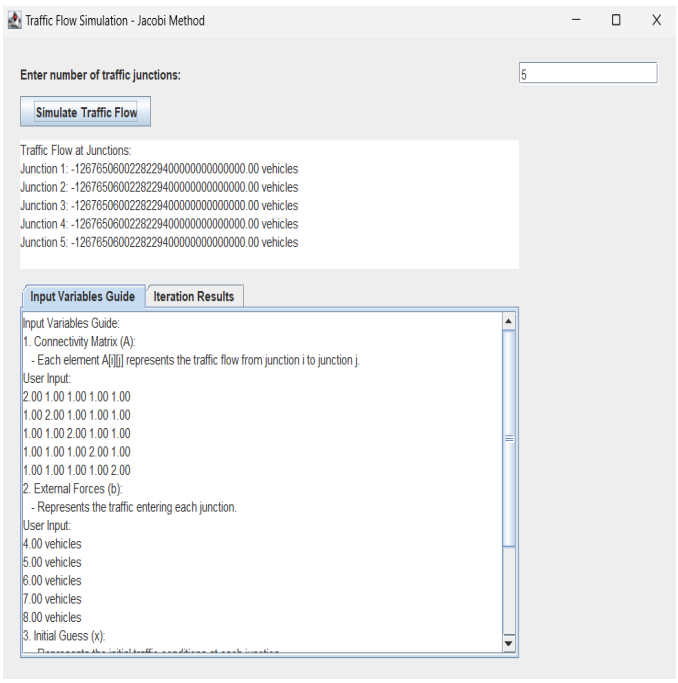


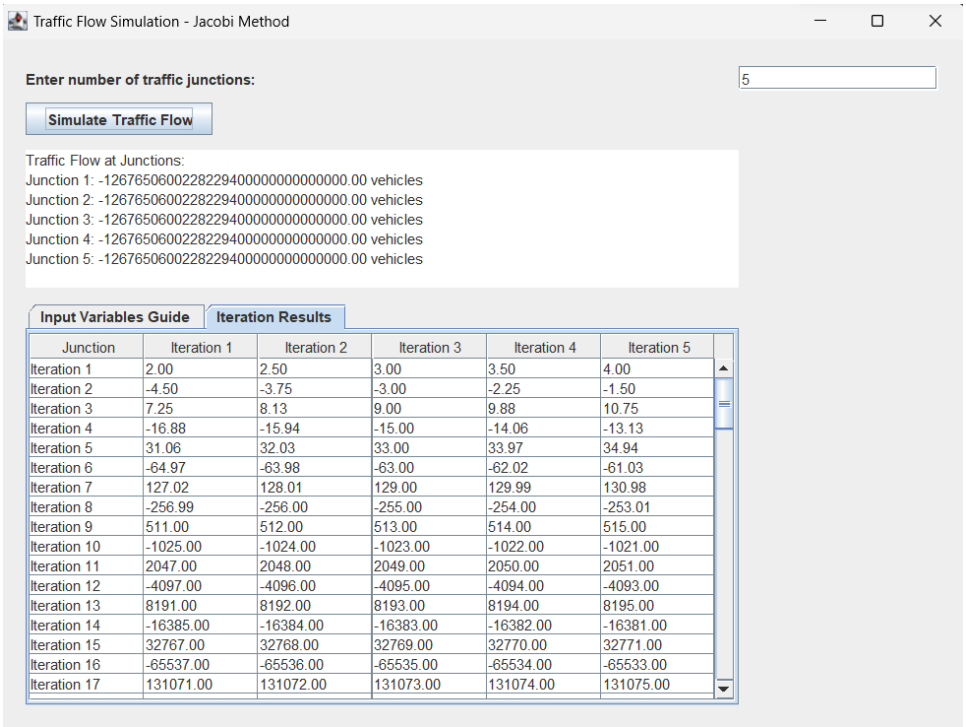


Junction	Iteration 1	Iteration 2	Iteration 3	Iteration 4
Iteration 1	4.00	3.00	5.50	3.50
Iteration 2	2.00	-0.50	1.00	3.25
Iteration 3	4.75	6.25	1.50	3.75
Iteration 4	-8.75	8.50	1.63	-0.75
Iteration 5	-8.63	-9.75	20.75	4.94
Iteration 6	41.31	-42.19	5.81	23.56
Iteration 7	72.63	56.25	-78.97	7.34
Iteration 8	-192.81	240.91	-44.84	-99.92
Iteration 9	-420.73	-200.05	420.19	-42.47
Iteration 10	868.75	-1300.58	370.18	524.48
Iteration 11	2452.85	655.88	-2036.52	405.00
Iteration 12	-3747.28	6933.90	-2522.92	-2568.63
Iteration 13	-13816.10	-1267.07	9789.85	-2850.77
Iteration 14	15180.76	-36243.58	16040.34	12440.51
Iteration 15	76092.98	-4456.41	-45736.05	18555.58
Iteration 16	-55372.82	186123.67	-96869.76	-58682.31
Iteration 17	-410428.78	79687.38	207123.97	-113806.30
Iteration 18	161561.51	-938480.02	564085.77	268936.68

Figure 3c. 5 Unknowns







Junction	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5
Iteration 1	2.00	2.50	3.00	3.50	4.00
Iteration 2	-4.50	-3.75	-3.00	-2.25	-1.50
Iteration 3	7.25	8.13	9.00	9.88	10.75
Iteration 4	-16.88	-15.94	-15.00	-14.06	-13.13
Iteration 5	31.06	32.03	33.00	33.97	34.94
Iteration 6	-64.97	-63.98	-63.00	-62.02	-61.03
Iteration 7	127.02	128.01	129.00	129.99	130.98
Iteration 8	-256.99	-256.00	-255.00	-254.00	-253.01
Iteration 9	511.00	512.00	513.00	514.00	515.00
Iteration 10	-1025.00	-1024.00	-1023.00	-1022.00	-1021.00
Iteration 11	2047.00	2048.00	2049.00	2050.00	2051.00
Iteration 12	-4097.00	-4096.00	-4095.00	-4094.00	-4093.00
Iteration 13	8191.00	8192.00	8193.00	8194.00	8195.00
Iteration 14	-16385.00	-16384.00	-16383.00	-16382.00	-16381.00
Iteration 15	32767.00	32768.00	32769.00	32770.00	32771.00
Iteration 16	-65537.00	-65536.00	-65535.00	-65534.00	-65533.00
Iteration 17	131071.00	131072.00	131073.00	131074.00	131075.00

Figure 3d. 5 Unknowns

4 Source Code

```
package JacobiMethod;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

public class TrafficFlowSimulation extends JFrame {

    private JTextField sizeTextField;
    private JTextArea resultTextArea, inputVariablesTextArea;
    private JTabbedPane tabbedPane;

    public TrafficFlowSimulation() {
        setTitle("Traffic Flow Simulation - Jacobi Method");
        setSize(800, 600);                          // SIZE
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // TAAS
        JLabel sizeLabel = new JLabel("Enter number of traffic junctions:");
        sizeTextField = new JTextField(5);
        JButton simulateButton = new JButton("Simulate Traffic Flow");
        resultTextArea = new JTextArea(5, 20);
        resultTextArea.setEditable(false);
        tabbedPane = new JTabbedPane();
        inputVariablesTextArea = new JTextArea(10, 10);
        inputVariablesTextArea.setEditable(false);

        JPanel panel = new JPanel();
        GroupLayout layout = new GroupLayout(panel);
        panel.setLayout(layout);
        // Default color ra sir para di sakit sa mata

        layout.setAutoCreateGaps(true);
        layout.setAutoCreateContainerGaps(true);

        layout.setHorizontalGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addComponent(sizeLabel)
                        .addComponent(simulateButton)
                        .addComponent(resultTextArea)
                        .addComponent(tabbedPane))
                .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
                        .addComponent(sizeTextField))
        );

        layout.setVerticalGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                        .addComponent(sizeLabel)
                        .addComponent(sizeTextField))
                .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                        .addComponent(simulateButton))
                .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                        .addComponent(resultTextArea))
                .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
```

```
            .addComponent(tabbedPane)
        );

        // BUTTON
        simulateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                simulateTrafficFlow();
            }
        });

        panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        add(panel);
        setVisible(true);
    }

    private void simulateTrafficFlow() {
        try {
            int n = Integer.parseInt(sizeTextField.getText());

            // MINIMUM IS SET TO 2
            if (n < 2) {
                JOptionPane.showMessageDialog(this, "Minimum number of junctions is 2. Setting size to 2.");
                n = 2;
                sizeTextField.setText("2");
            }

            // PROMPT USER TO INPUT THE CONNECTIVITY MATRIX (A)
            ArrayList<ArrayList<Double>> A = new ArrayList<>();
            for (int i = 0; i < n; i++) {
                ArrayList<Double> row = new ArrayList<>();
                for (int j = 0; j < n; j++) {
                    double coefficient = Double.parseDouble(
                            JOptionPane.showInputDialog("Enter A[" + (i + 1) + "][" + (j + 1) + "]:"));
                    row.add(coefficient);
                }
                A.add(row);
            }

            // PROMPT USER TO INPUT THE EXTERNAL FORCES VECTOR (b)
            ArrayList<Double> b = new ArrayList<>();
            for (int i = 0; i < n; i++) {
                double force = Double.parseDouble(
                        JOptionPane.showInputDialog("Enter b[" + (i + 1) + "]:"));
                b.add(force);
            }

            // PROMPT USER TO INPUT THE INITIAL GUESS VECTOR (x)
            ArrayList<Double> x = new ArrayList<>();
            for (int i = 0; i < n; i++) {
                double guess = Double.parseDouble(
                        JOptionPane.showInputDialog("Enter x[" + (i + 1) + "]:"));
                x.add(guess);
            }

            // DISPLAYING THE GUIDE
            displayGuide(A, b, x);

            // DISPLAYING THE INPUTTED VARIABLES
            displayInputVariables(A, b, x);
```

```
            // ASSIGNING TO JACOBI
            jacobiMethod(n, A, b, x, 2);

            // DISPLAYING THE RESULT
            displayResult(x, n);

            // IF WAY INPUT
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(this, "Please enter valid numeric values.");
        }
    }

    private void displayGuide(ArrayList<ArrayList<Double>> A, ArrayList<Double> b, ArrayList<Double> x) {
            // SHOW GUIDE
        JTextArea inputGuideTextArea = new JTextArea(10, 10);
        inputGuideTextArea.setEditable(false);
        inputGuideTextArea.setText("Input Variables Guide:\n");

        inputGuideTextArea.append("1. Connectivity Matrix (A):\n");
        inputGuideTextArea.append("   - Each element A[i][j] represents the traffic flow from junction i to junction j.\n");
        inputGuideTextArea.append("User Input: \n");
        for (ArrayList<Double> row : A) {
            for (Double coefficient : row) {
                inputGuideTextArea.append(String.format("%.2f", coefficient) + " ");
            }
            inputGuideTextArea.append("\n");
        }

        inputGuideTextArea.append("2. External Forces (b):\n");
        inputGuideTextArea.append("   - Represents the traffic entering each junction.\n");
        inputGuideTextArea.append("User Input: \n");
        for (Double force : b) {
            inputGuideTextArea.append(String.format("%.2f", force) + " vehicles\n");
        }

        inputGuideTextArea.append("3. Initial Guess (x):\n");
        inputGuideTextArea.append("   - Represents the initial traffic conditions at each junction.\n");
        inputGuideTextArea.append("User Input: \n");
        for (Double guess : x) {
            inputGuideTextArea.append(String.format("%.2f", guess) + " vehicles\n");
        }

        JScrollPane guideScrollPane = new JScrollPane(inputGuideTextArea);
        tabbedPane.addTab("Input Variables Guide", guideScrollPane);
        tabbedPane.setSelectedComponent(guideScrollPane);
    }

    private void jacobiMethod(int n, ArrayList<ArrayList<Double>> A, ArrayList<Double> b,
                              ArrayList<Double> x, int decimalPlaces) {
        int maxIterations = 100; // MAX ITERATIONS IS SET TO 100
        double tolerance = 1e-6; // DEFAULT TOLERANCE

        DefaultTableModel tableModel = new DefaultTableModel();
        tableModel.addColumn("Junction");
        for (int i = 0; i < n; i++) {
            tableModel.addColumn("Iteration " + (i + 1));
        }

        JTable iterationsTable = new JTable(tableModel);
        JScrollPane iterationsScrollPane = new JScrollPane(iterationsTable);
        tabbedPane.addTab("Iteration Results", iterationsScrollPane);
```

```
        boolean converged = false;

        for (int iteration = 0; iteration < maxIterations; iteration++) {
            ArrayList<Double> tempX = new ArrayList<>(x);

            Object[] rowData = new Object[n + 1];
            rowData[0] = "Iteration " + (iteration + 1);

            for (int i = 0; i < n; i++) {
                double sum = b.get(i);
                for (int j = 0; j < n; j++) {
                    if (j != i) {
                        sum -= A.get(i).get(j) * tempX.get(j);
                    }
                }
                x.set(i, sum / A.get(i).get(i));
                rowData[i + 1] = String.format("%.2f", x.get(i));
            }

            tableModel.addRow(rowData);

            // CHECK IF NICONVERGE NA
            converged = hasConverged(x, tempX, tolerance);

            // IF CONVERGED KAY EXIT
            if (converged) {
                break;
            }
        }

        if (converged) {
            tabbedPane.setTitleAt(tabbedPane.indexOfComponent(iterationsScrollPane), "Solution Found");
        } else {
            JOptionPane.showMessageDialog(this, "Jacobi method did not converge within the maximum iterations.");
        }
    }

    private boolean hasConverged(ArrayList<Double> x, ArrayList<Double> tempX, double tolerance) {
        // CHECK IF NICONVERGE
        for (int i = 0; i < x.size(); i++) {
            if (Math.abs(x.get(i) - tempX.get(i)) >= tolerance) {
                return false;
            }
        }
        return true;
    }

    private void displayResult(ArrayList<Double> x, int n) {
        // DISPLAYING RESULT SA JTEXTAREA
        resultTextArea.setText("Traffic Flow at Junctions:\n");
        for (int i = 0; i < n; i++) {
            resultTextArea.append("Junction " + (i + 1) + ": " + String.format("%.2f", x.get(i)) + " vehicles\n");
        }
    }

    private void displayInputVariables(ArrayList<ArrayList<Double>> A, ArrayList<Double> b, ArrayList<Double> x) {
        // DISPLAYING INPUTTED VARIABLES SA TEXTAREA
        inputVariablesTextArea.setText("Input Variables:\n");
        inputVariablesTextArea.append("1. Connectivity Matrix (A):\n");
        for (ArrayList<Double> row : A) {
            for (Double coefficient : row) {
                inputVariablesTextArea.append(String.format("%.2f", coefficient) + " ");
```

```
            }
            inputVariablesTextArea.append("\n");
        }
        inputVariablesTextArea.append("2. External Forces (b):\n");
        for (Double force : b) {
            inputVariablesTextArea.append(String.format("%.2f", force) + " vehicles\n");
        }
        inputVariablesTextArea.append("3. Initial Guess (x):\n");
        for (Double guess : x) {
            inputVariablesTextArea.append(String.format("%.2f", guess) + " vehicles\n");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new TrafficFlowSimulation();
            }
        });
    }
}
```

5 Profile of Members

Paul Emmanuel G. Corsino

- Leader
- Main Programmer
- Sub Documentation

Klinnsonveins Yee

- Main Documentation
- Sub Programmer