PROJECT Report
Calculus and Analytical Geometry

BS(CS)-3B

AIR UNIVERSITY ISLAMABAD

Submitted to: Sir Usman Ghani

Traffic Monitoring System Using Definite Integrals

Introduction:

Traffic monitoring and analysis are crucial in modern transportation systems for ensuring smooth vehicular flow and reducing congestion. This project aims to demonstrate how traffic density over a specific time interval can be calculated and visualized using calculus. By applying definite integrals, we estimate the total number of vehicles passing through a road segment during a given period.

The system also includes features to plot a graph of traffic density, calculate the integral for total traffic flow, and animate vehicle movement to represent real-time traffic visually. The project leverages Python libraries such as scipy, matplotlib, and tkinter.

Scenario Representation Using Definite Integrals:

The traffic density at a given time t is modeled using the function:

f(t)=10+5sin⁡(t)f(t) = 10 + 5 \sin(t)

This equation represents periodic traffic density, where 10 is the average density (vehicles per minute) and 5sin⁡(t)5 \sin(t) introduces fluctuations due to peak and off-peak hours.

To compute the **total traffic flow** over a time interval [a,b][a, b], we use the definite integral:

Total Traffic Flow=∫abf(t) dt\text{Total Traffic Flow} = \int_{a}^{b} f(t) \, dt

This integral calculates the area under the curve of f(t), representing the cumulative number of vehicles passing through.

The equation is suitable because it reflects realistic variations in traffic density and allows us to estimate traffic over any period.

Code Analysis:

Graph Representation:

The traffic density function is visualized using the matplotlib library.

- The graph plots f(t) over a time interval [a, b], with t on the x-axis and f(t) (vehicles/minute) on the y-axis.

- The fill_between method shades the area under the curve to visually represent the integral.

Key steps:

1. Generate a sequence of t values using np.linspace.
2. Compute f(t) for each t.
3. Plot t vs f(t) and highlight the area under the curve.

Definite Integral Logic:

The integral of f(t) over [a, b] is calculated using the quad function from the scipy.integrate module.

- The function returns the integral value and an estimate of the error.
- User input for a (start time) and b (end time) defines the interval.
- The result is displayed as the total traffic flow in vehicles.

Animation:

To animate traffic flow:

1. A separate Tkinter window displays a road with moving car icons.
2. Cars are added to the canvas using the PhotoImage class from the PIL library.
3. The function f(t) determines traffic density, affecting the speed of the cars (higher density → slower speed).
4. Threading ensures smooth animation alongside the main GUI.

GUI Design:

The graphical user interface (GUI) is implemented using tkinter:

- Input fields for start and end times.
- A button to calculate traffic flow and generate visualizations.
- Dynamic display of results and a polished layout with a custom background.

Interconnection:

The system integrates all components seamlessly:

1. User inputs are validated and used for integral calculation.
2. The results trigger both graph plotting and animation.
3. Real-time feedback makes the system interactive and informative.

Complete Code:

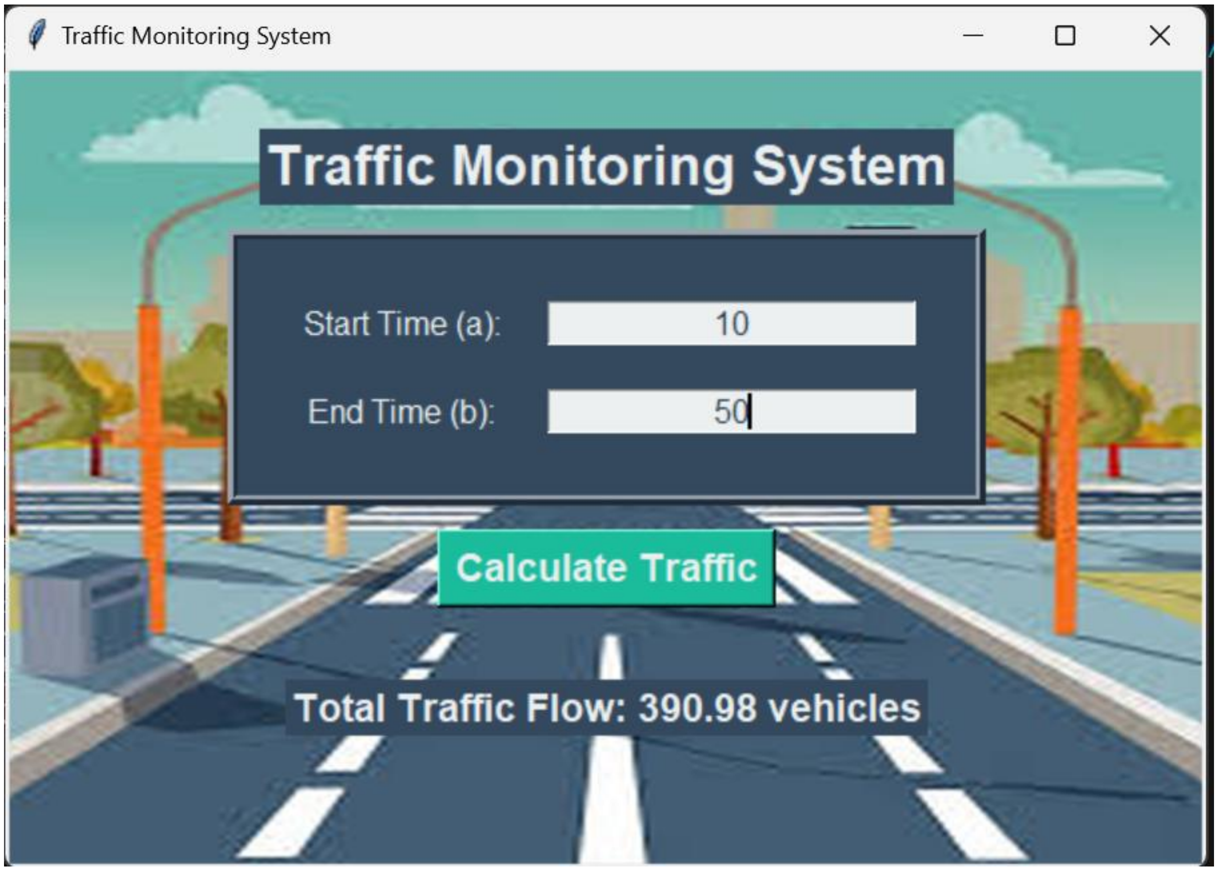
```python
import tkinter as tk
from tkinter import messagebox
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.integrate import quad
import threading
from PIL import Image, ImageTk  # For car icons and background image

# Define the traffic density function
def f(t):
    return 10 + 5 * math.sin(t)

# Function to calculate the total traffic flow
def calculate_traffic():
    try:
        a = float(entry_start.get())
        b = float(entry_end.get())
        # Compute definite integral
        total_traffic, _ = quad(f, a, b)
        result_label.config(text=f"Total Traffic Flow: {total_traffic:.2f}
vehicles")
        # Plot the graph in the main thread
        plot_graph(a, b)
        # Start animation for traffic flow in a separate thread
        threading.Thread(target=animate_traffic, args=(a, b)).start()
    except ValueError:
        messagebox.showerror("Input Error", "Please enter valid numerical values
for the time interval.")


# Function to plot the traffic density graph
def plot_graph(a, b):
    t = np.linspace(a, b, 1000)
    y = [f(ti) for ti in t]
    plt.figure(figsize=(8, 5))
    plt.plot(t, y, label="Traffic Density f(t)")
    plt.fill_between(t, y, color="skyblue", alpha=0.4, label="Area under curve")
    plt.title("Traffic Density Graph")
    plt.xlabel("Time (minutes)")
    plt.ylabel("Density (vehicles/minute)")
    plt.legend()
    plt.grid(True)
    plt.show()


# Function to animate traffic flow
def animate_traffic(a, b):
    # Create a window for the animation
    animation_window = tk.Toplevel(root)
    animation_window.title("Traffic Animation")
    canvas = tk.Canvas(animation_window, width=800, height=300, bg="white")
    canvas.pack()

    # Draw road
    road = canvas.create_rectangle(50, 100, 750, 200, fill="black")
    lane_line = canvas.create_line(50, 150, 750, 150, dash=(10, 10),
fill="white")

    # Load car image
    car_image = Image.open("car.png")
    car_image = car_image.resize((50, 30), Image.Resampling.LANCZOS)
    car_icon = ImageTk.PhotoImage(car_image)

    cars = []
    # Add cars to the canvas
    for i in range(5):
        x = 60 + i * 100
        car = canvas.create_image(x, 135, image=car_icon, anchor=tk.NW)
        cars.append(car)


    t = a
    while t <= b:
        density = f(t)  # Get the current traffic density
        speed = 200 / density  # Adjust speed based on density
        for car in cars:
            canvas.move(car, speed, 0)
            pos = canvas.coords(car)
            if pos[0] > 750:  # Reset car position when it exits the road
                canvas.move(car, -700, 0)
        t += 0.1
        animation_window.update()
        canvas.after(100)
```

```python
# GUI Setup
root = tk.Tk()
root.title("Traffic Monitoring System")
root.geometry("600x400")
root.configure(bg="#2c3e50")

# Load and set the background image
bg_image = Image.open("background.jpg")
bg_image = bg_image.resize((600, 400), Image.Resampling.LANCZOS)
bg_photo = ImageTk.PhotoImage(bg_image)

canvas = tk.Canvas(root, width=600, height=400)
canvas.pack(fill="both", expand=True)
canvas.create_image(0, 0, image=bg_photo, anchor="nw")

# Title label
title_label = tk.Label(root, text="Traffic Monitoring System", font=("Arial", 20,
"bold"), bg="#34495e", fg="#ecf0f1")
canvas.create_window(300, 50, window=title_label)

# Input fields for time interval
frame = tk.Frame(root, bg="#34495e", padx=20, pady=20, bd=5, relief="ridge")
canvas.create_window(300, 150, window=frame)

tk.Label(frame, text="Start Time (a):", font=("Arial", 12), bg="#34495e",
fg="#ecf0f1").grid(row=0, column=0, padx=10, pady=10)
entry_start = tk.Entry(frame, font=("Arial", 12), bg="#ecf0f1", fg="#2c3e50",
justify="center")
entry_start.grid(row=0, column=1, padx=10, pady=10)

tk.Label(frame, text="End Time (b):", font=("Arial", 12), bg="#34495e",
fg="#ecf0f1").grid(row=1, column=0, padx=10, pady=10)
entry_end = tk.Entry(frame, font=("Arial", 12), bg="#ecf0f1", fg="#2c3e50",
justify="center")
entry_end.grid(row=1, column=1, padx=10, pady=10)

# Button to calculate traffic and generate visualizations
calculate_button = tk.Button(root, text="Calculate Traffic", font=("Arial", 14,
"bold"), bg="#1abc9c", fg="#ecf0f1", relief="raised", command=calculate_traffic,
cursor="hand2")
canvas.create_window(300, 250, window=calculate_button)

# Label to display the result
```

```
result_label = tk.Label(root, text="Total Traffic Flow: ", font=("Arial", 14,
"bold"), bg="#34495e", fg="#ecf0f1")
canvas.create_window(300, 320, window=result_label)

root.mainloop()
```

Output:

Traffic Monitoring System:



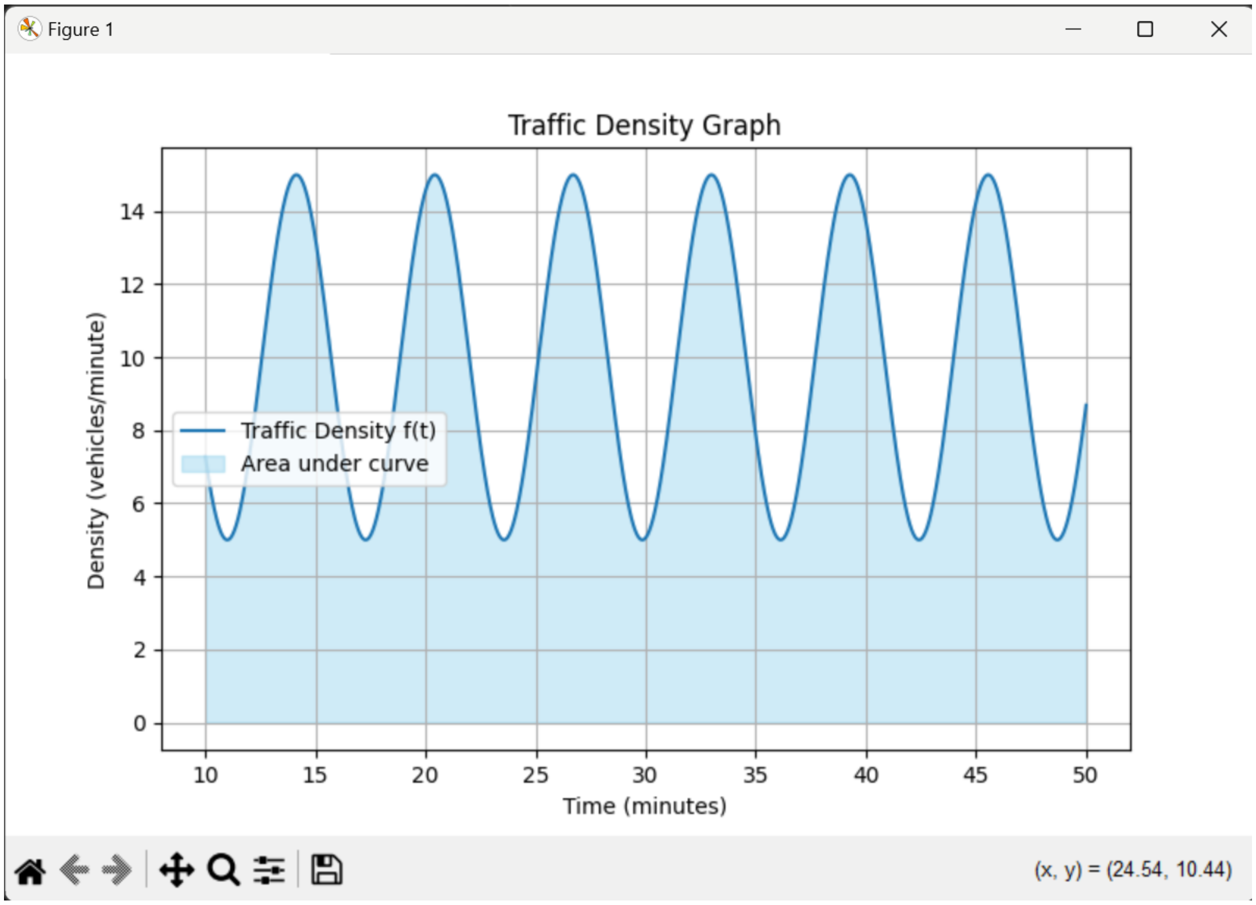
Traffic Monitoring System

Start Time (a): 10

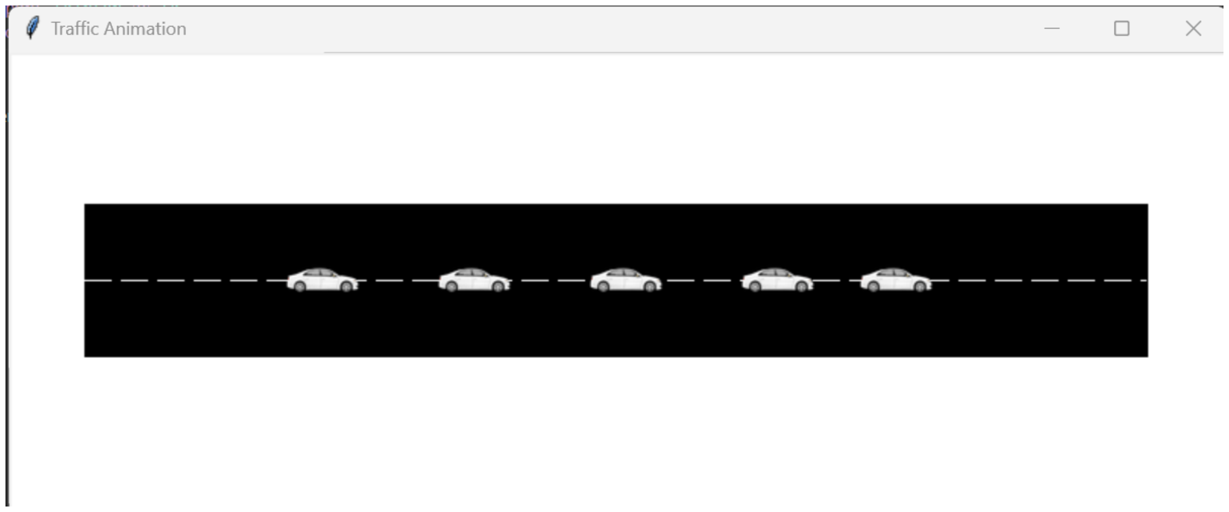
End Time (b): 50

Total Traffic Flow: 390.98 vehicles

Traffic Density Graph:



Traffic Animation:



Conclusion:

- This project demonstrates how calculus, specifically definite integrals, can model and solve real-world problems like traffic flow estimation. By integrating mathematical logic with Python programming, the system provides an interactive solution for visualizing and analyzing traffic data.
- Through this project, we gain insight into the power of calculus in practical applications and the importance of data visualization in understanding complex scenarios.