**Traffic Flow: Density-Dependent Model**

Malena Fassnacht

**Abstract**

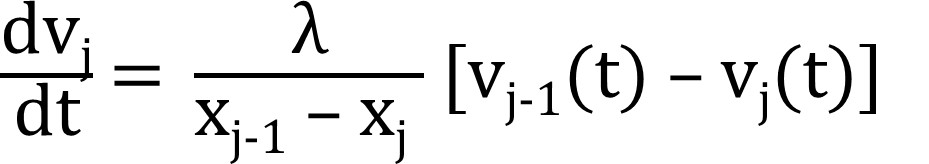
This project models the flow of traffic on a single-lane road under various conditions. The model is based on the idea that a driver will be more alert if the density of traffic is greater. The reaction time of the driver decreases as the distance between cars decreases, which corresponds to a larger acceleration/deceleration. When there is a large distance between cars, the driver will be less alert and slower to accelerate/decelerate. In my code, I implement the fourth-order Runge-Kutta method to integrate the acceleration equation. At each point for a given timestep, the code will display the velocity and position of each vehicle in a line of traffic. If a car ‘surpasses’ the car ahead of it, the code will display a crash statement. I consider four different test cases based on realistic driving scenarios:

1. The traffic accelerates from rest, as it would from a stoplight.
2. The leading car slows to a sudden stop.
3. The leading car travels at a lower initial velocity than the following cars.
4. The velocity of leading car changes randomly (within specified bounds) in regular time intervals.

With a few modifications to optimize output, a model like this could be utilized to determine guidelines for safe driving practices (such as the minimum following distance between cars to prevent a crash at a given speed).

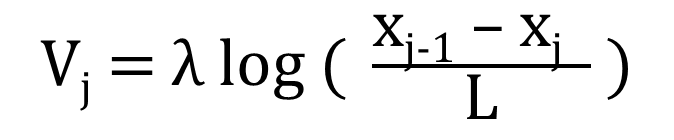
**Mathematical Statement of the Problem**

The acceleration of the vehicles is given by the equation:



Lambda is a constant related to the reaction time of the driver, x[j-1] – x[j] is the distance between two cars, and v[j-1] – v[j] is the difference in velocity between the two vehicles. According to this equation, if the velocity of car 1 is greater than the velocity of car 2, car 2’s acceleration will be positive. If the velocity of car 1 is less than the velocity of car 2, car 2’s acceleration will be negative (it will decelerate). When the distance between cars is small, acceleration/deceleration is large, and when the distance between cars is large, acceleration/deceleration is small.

The equation can be integrated as:



This is the exact solution for the velocity of the cars. When L = x[j-1] – x[j], the velocity becomes zero. Therefore, L corresponds to the jamming distance for traffic.

**Numerical Method**

I used the fourth-order Runge-Kutta method to integrate the acceleration equation for both velocity and position of the vehicles. The position of the cars for a given timestep is approximated using the velocity of the cars approximated at the previous timestep. This is a general pseudo-code for the fourth-order Runge-Kutta method:

u1 = un + f(tn , un )

u2 = un + f(tn + , u1)

u3 = un + Δt f(tn + , u2)

un+1 = un + [f(tn , un) + 2f(tn + , u1) + 2f(tn + , u2) + f(tn , u3)]

In my code, I used arrays to store both my velocity and position values. Each array index corresponds to one car in a line of traffic. My code:

for(i = 1; i < N; i++)

{

x1[i] = x[i] + ((dt/2)\*(f1(v,i)));

v1[i] = v[i] + ((dt/2)\*(f2(lambda,x,v,i)));

x2[i] = x[i] + ((dt/2)\*(f1(v1,i)));

v2[i] = v[i] + ((dt/2)\*(f2(lambda,x1,v1,i)));

x3[i] = x[i] + (dt\*(f1(v2,i)));

v3[i] = v[i] + (dt\*(f2(lambda,x2,v2,i)));

xNew[i] = x[i] + (dt/6)\*(f1(v,i)+2.0\*f1(v1,i)+2.0\*f1(v2,i)+f1(v3,i));

vNew[i] = v[i] + (dt/6)\*(f2(lambda,x,v,i) + 2.0\*f2(lambda,x1,v1,i) +

2.0\*f2(lambda,x2,v2,i) + f2(lambda,x3,v3,i));

x[i] = xNew[i];

v[i] = vNew[i];

}

**Technical Specifications of Computer**

Processor: Intel Core i5-6200U CPU @2.30 GHz

2.40GHz

Installed RAM: 4.00 GB (3.89 GB usable)

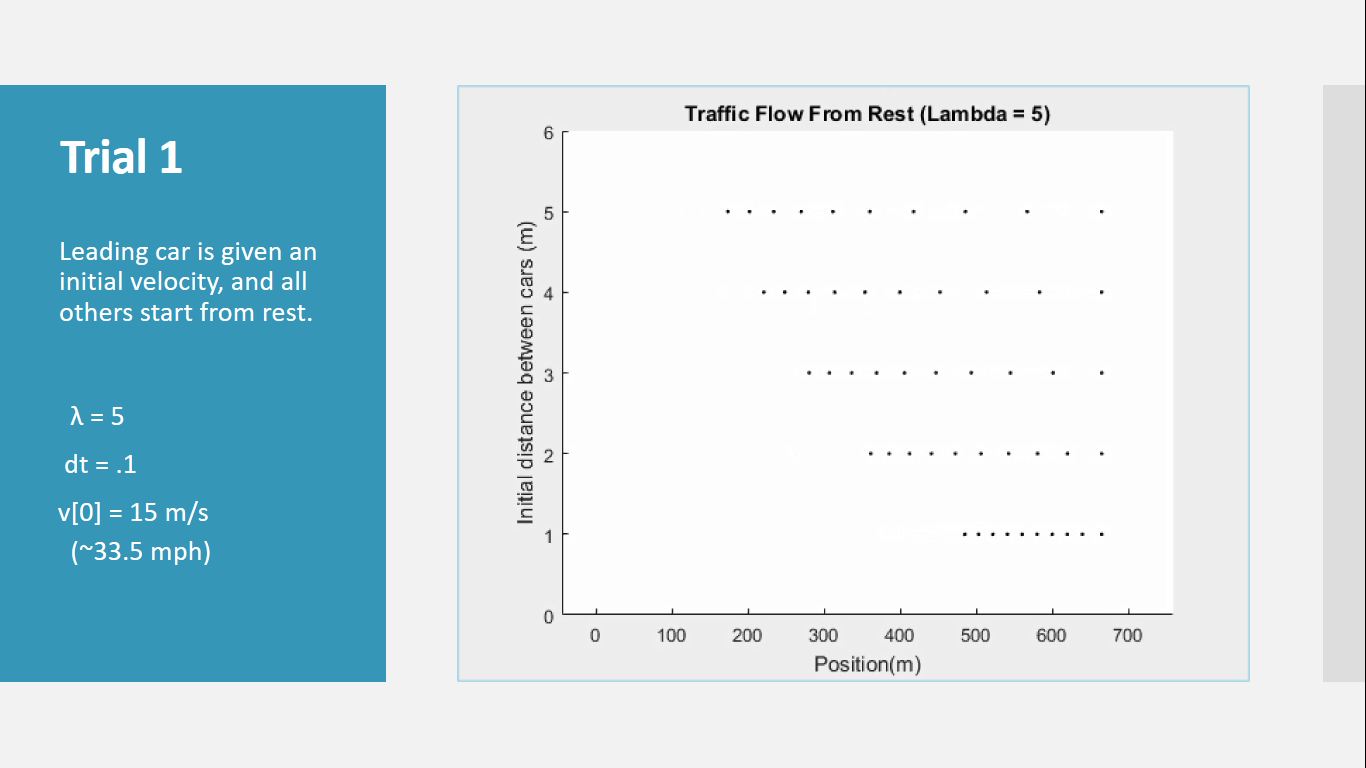
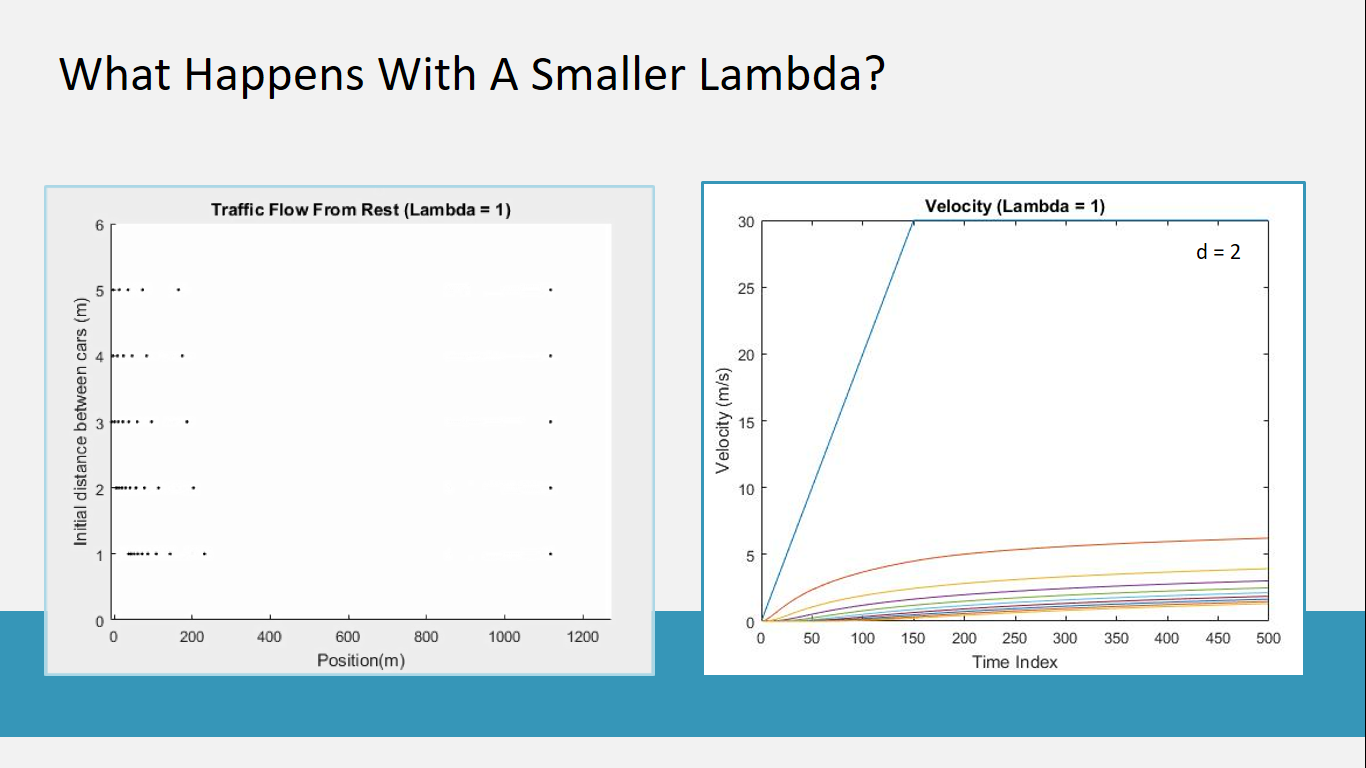
System type: 64-bit operating system, x64-based processor

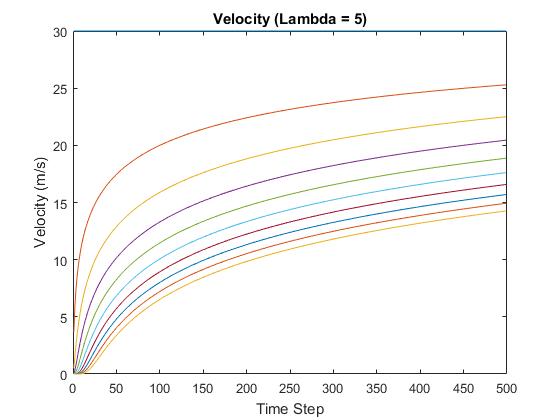
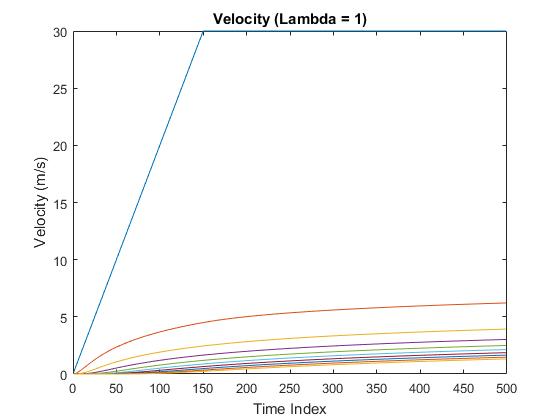
**Results**

**Case 1: Cars accelerating from rest**

The leading car is given an initial velocity, and all the following cars start from rest.

The greater the initial distance is between cars, the longer they take to react to the lead car’s velocity change. When lambda is small, drivers are extremely slow to react to the lead car.

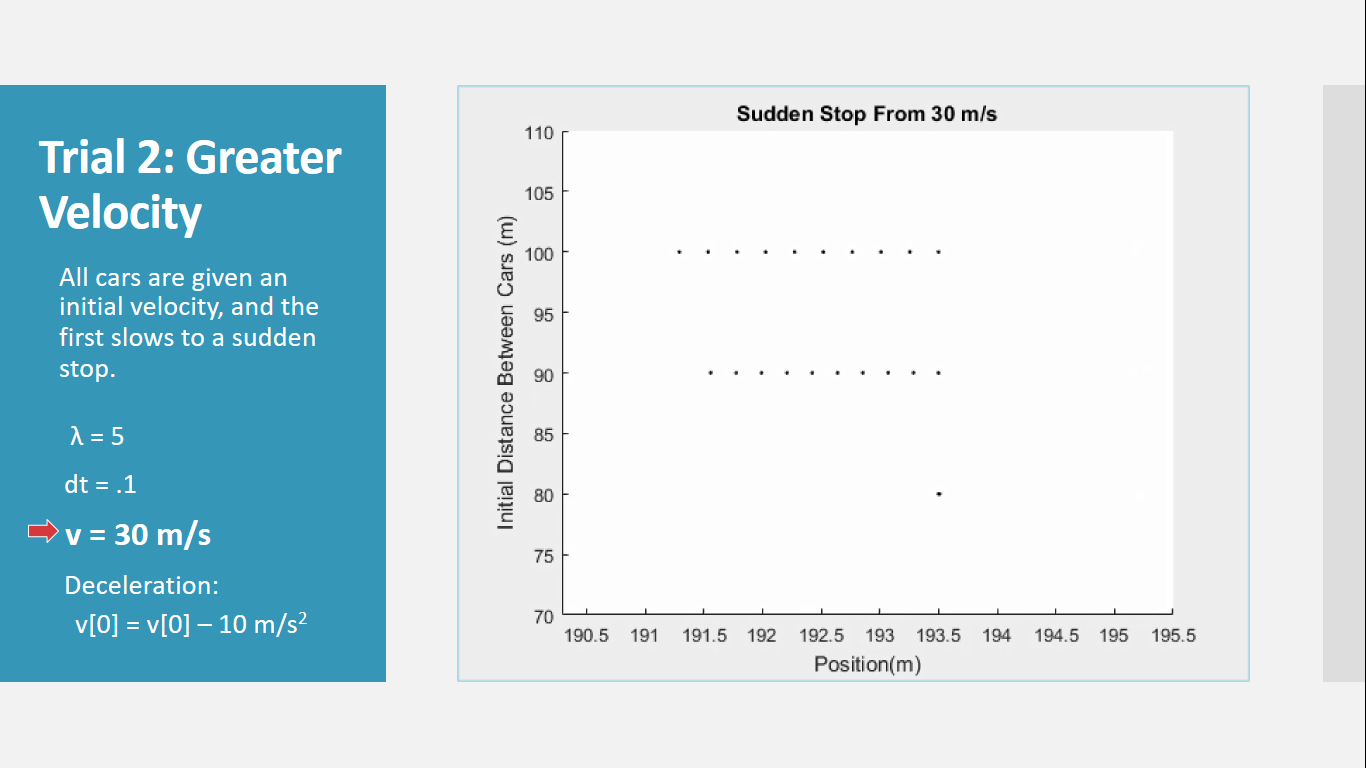
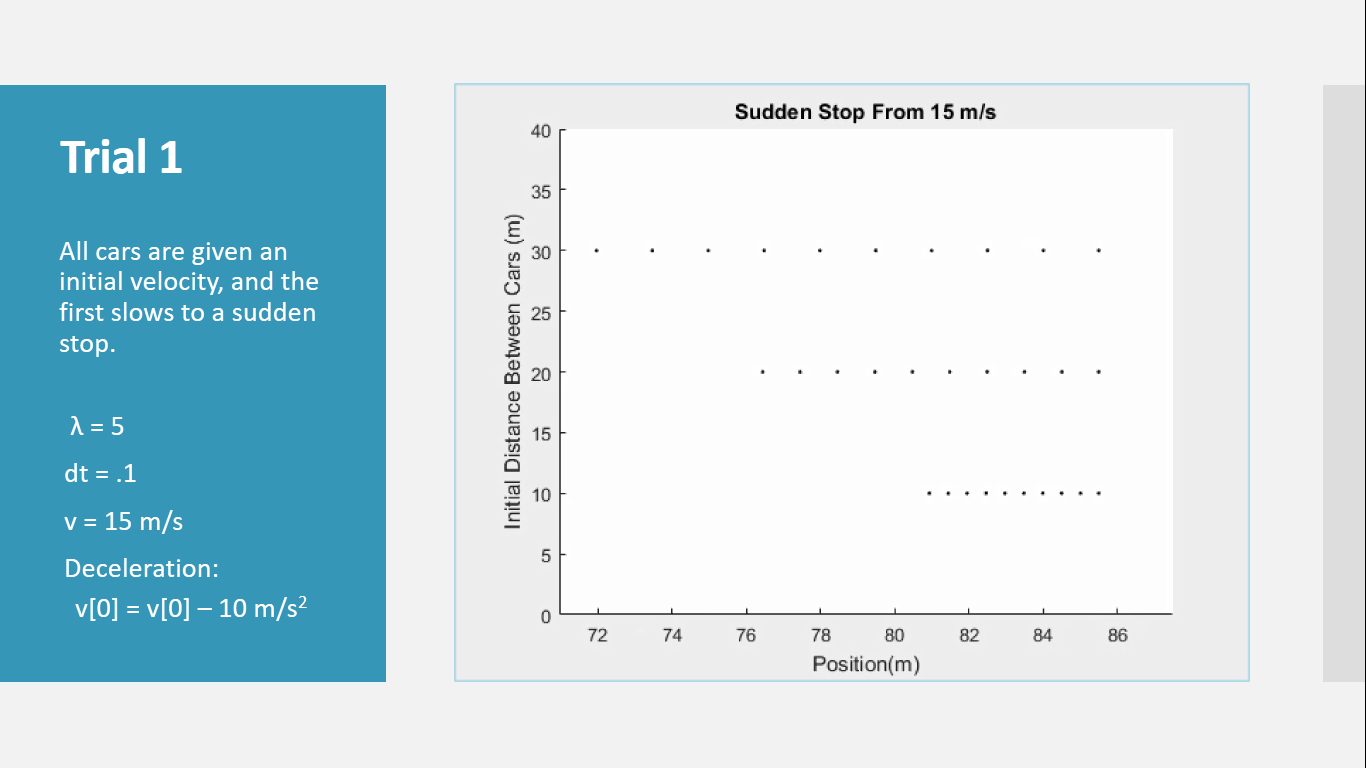
Lambda = 5; dt = .1; v[0] = 15 m/s. Lambda = 1; dt = .1; v[0] = 15 m/s



**Case 2: Lead car comes to a sudden stop**

All of the cars are given an initial velocity. The lead car slows to a sudden stop (acceleration = -10m/s2), and the following cars must react to prevent a crash. Because of the density dependence of the drivers’ reaction times, the cars begin slowing down only when they are very close to the vehicle ahead. With a small timestep, however, the cars will not crash even if they have a large initial velocity. This is because as a car nears the one ahead of it, the magnitude of acceleration can become infinitely large. A more realistic model may include a limit for the absolute value of acceleration or a maximum reaction time of the driver. A larger timestep will result in a more sensitive crash case; however, the values for velocity and position will become less accurate.

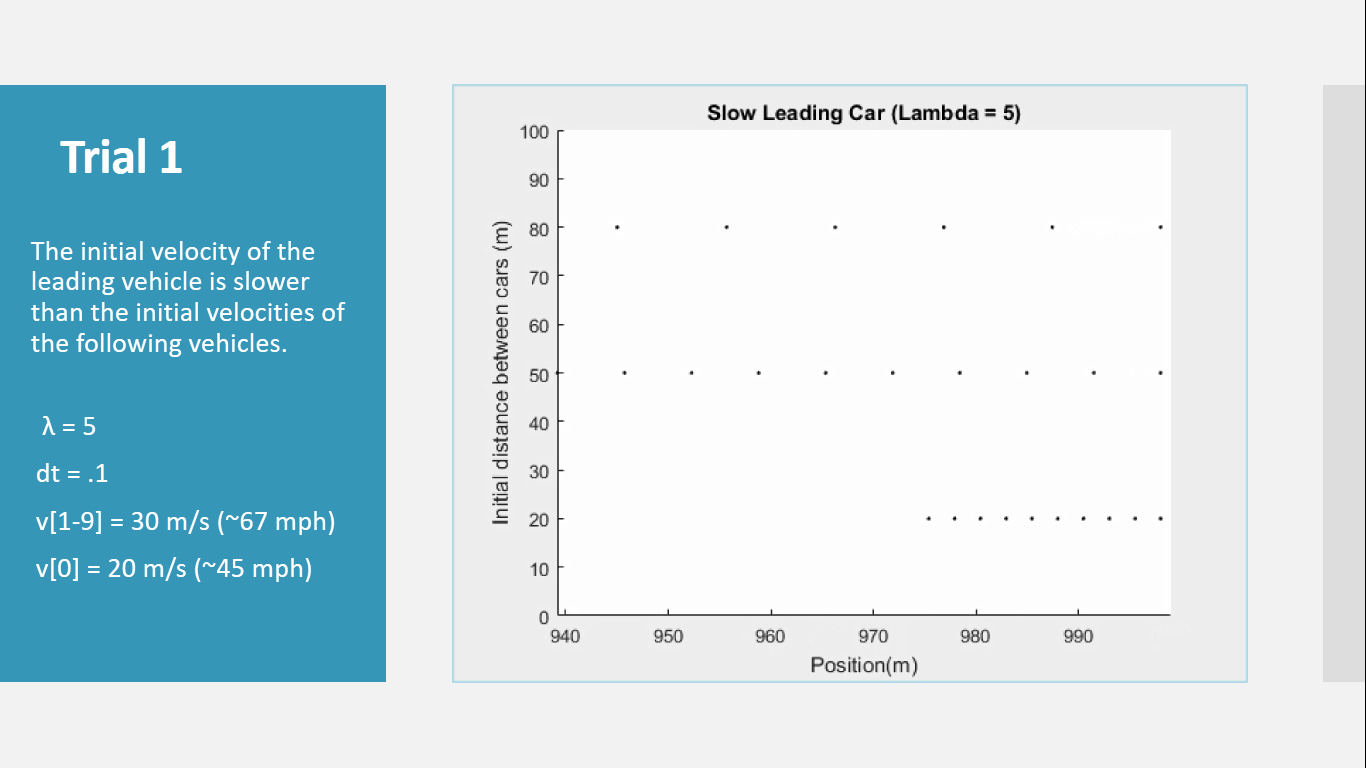
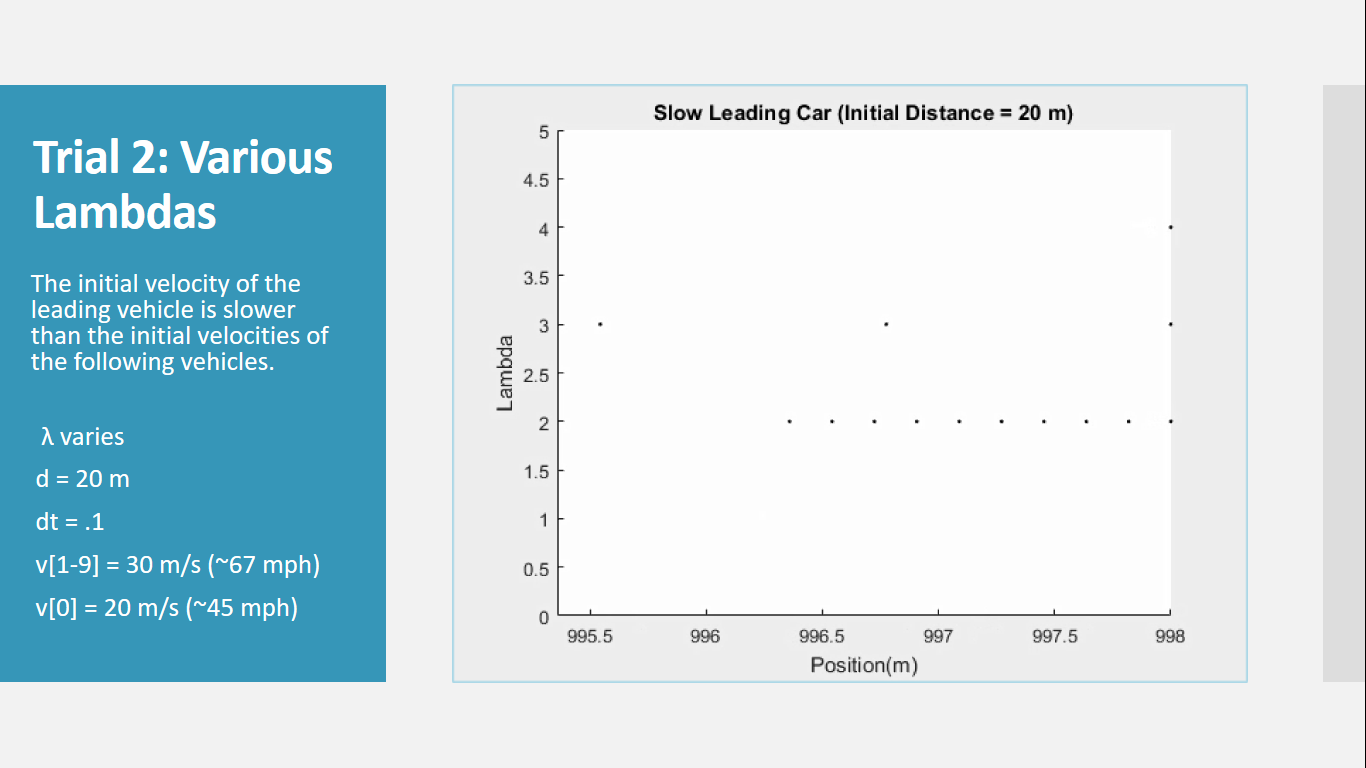
Final frame of stopped vehicles:

Lambda = 5; dt = .1; v = 15 m/s Lambda = 5; dt = .1; v = 30 m/s

**Case 3: Slow lead car**

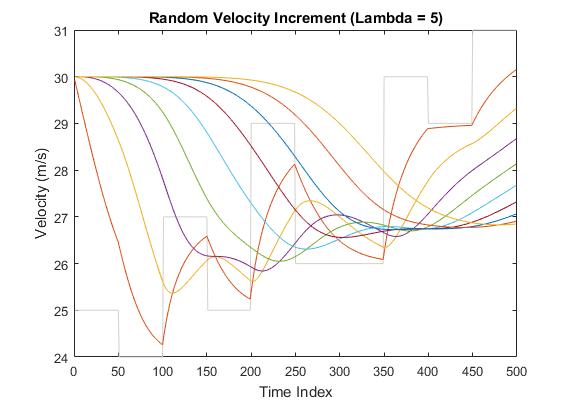
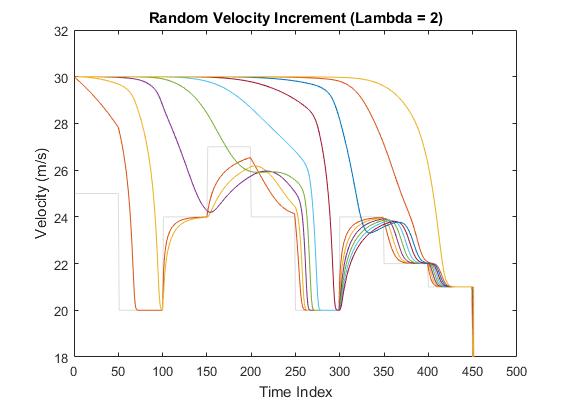
The initial velocity of the first car is less than the initial velocities of the following cars. When the initial distance between cars is greater, it takes more time for every car to achieve the speed of the first vehicle than when starting from a smaller initial distance. With a lambda of 1, my program detected a crash between vehicles. With a lambda of 2, the cars did not crash, but the final following distance was <.2 m. Final following distance increases as lambda increases.

Final frame of vehicles:

v[0] = 20 m/s; v[1-9] = 30 m/s; dt = .1

**Case 4: Random Changes to Vehicle Velocity**

All cars begin with the same initial velocity, but the velocity of the first car changes by a random amount (limited to +/- 5 m/s) after every 5 seconds. With a lambda of 5, the cars were consistently able to react quickly enough to avoid crashes. With a lambda of 2, my code detected a crash for most of the trial runs. In the case that it didn’t, the cars still came extremely close to one another at times (within < .5m).

dt = .1; d = 30m; v = 25m/s