NUMERICAL METHODS ON TRAFFIC FLOW

A Dissertation
Submitted in partial fulfillment of requirements for the award of the degree of
Master of Science
In
Mathematics

Submitted By:
ADITI GUPTA (2K19/MSCMAT/07)
RIYA SANKHLA (2K19/MSCMAT/26)
Under the guidance of:
Dr. Vivek Kumar Aggarwal

DEPARTMENT OF APPLIED MATHEMATICS
DELHI TECHNOLOGICAL UNIVERSITY
Shahbad Daulatpur, Main Bawana Road, Delhi-110042, India
May,2021

DEPARTMENT OF APPLIED MATHEMATICS
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road,New Delhi-110042

DECLARATION

We hereby declare that the work presented in this report entitled "NUMERICAL METHODS FOR TRAFFIC FLOW", in fulfillment of the requirement for the award of the degree Master of Science in Mathematics, submitted in the Applied Mathematics Department of Delhi Technological University, New Delhi, is an authentic record of our own work carried out during our degree under the guidance of Dr. Vivek Kumar Aggarwal.
The work reported in this has not been submitted by us for award of any other degree or diploma**.**

Date: 24th May,2021

Riya.

RIYA SANKHLA
(2K19/MSCMAT/26)

Aditi Gupta

ADITI GUPTA
(2K19/MSCMAT/07)

DEPARTMENT OF APPLIED MATHEMATICS
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, New Delhi-110042

CERTIFICATE

This is to certify that the Project work entitled "NUMERICAL METHODS FOR TRAFFIC FLOW" submitted by Aditi Gupta and Riya Sankhla in fulfillment for the requirements of the award of Degree Master of Science in Mathematics is an authentic work carried out by them under my supervision and guidance. To the best of my knowledge, the matter embodied in the Dissertation has not been submitted to any other University / Institute for the award of any Degree.

[illegible]

Date: 24th May, 2021

Dr. Vivek Kumar Aggarwal
Department of Applied
Mathematics

DEPARTMENT OF APPLIED MATHEMATICS
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, New Delhi-110042

ACKNOWLEDGEMENT

We express our sincere gratitude to Dr. Vivek Kumar Aggarwal (Professor, Department of Applied Mathematics), for his valuable guidance and timely suggestions during the entire duration of our dissertation work, without which this work would not have been possible. We would also like to convey our deep regards to all other faculty members of the Department of Applied Mathematics, who have bestowed their great effort and guidance at appropriate times without which it would have been very difficult on our part to finish this work. Finally, we would also like to thank our friends for their advice and pointing out our mistakes.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

To address initial value issues, we use Euler's approach and the Runge-Kutta method in this thesis. The project's purpose is to compare the two methodologies on preliminary problems, highlighting their limitations and benefits. We also use the Runge-Kutta method as a way for creating a mathematical traffic flow model. The purpose of this thesis is to shed light on how **The Optimal Velocity Model** is solved using the fourth-order Runge-Kutta method. To execute simulations of the model, we select beginning conditions and base cases. To validate the usage of the two - fourth-order Runge-Kutta method , Optimal Velocity Model, we consider a few examples with one-car ,two-car systems.

CHAPTER-1

INTRODUCTION

The field of applied mathematics includes numerical analysis. Exact methods can be used to solve some equations, but others, which are more complex numerical estimating approaches must be used to solve them. Many studies on traffic modelling employ numerical analysis for solving the models, but they don't explain why they picked that strategy. They also don't go into great detail on the start conditions or the necessary base case scenarios. The models could be solved in a variety of ways. Some researchers, for example, employ Euler's approach. Others employ the Runge-Kutta method. It is critical that the viewer knows the starting conditions and how this system works for either way.

Both methods are used to approximate points on the graph of the solution in a given interval to solve a differential equation given initial values. One of these numerical analytic approaches' limitations is that they are distant from the beginning value point, the further the approximation. The estimation is based on the actual values. Euler's approach has a higher error compared to Runge-Kutta, although Runge-Kutta computations are more complicated. Euler's method has a higher inaccuracy than Runge Kutta's, but it's a good starting point for more advanced techniques.

CHAPTER-2

Ordinary Differential Equations (Numerical methods)

Numerical methods such as Euler's and Runge-Kutta approximate the solutions of the differential equations to initial value issues on a given interval, as previously stated:

$$\frac{dy}{dt} = f(t, y)\ ,\ \ y(a) = \alpha\ , a \leq t \leq b \qquad (2.1)$$

At N discrete places, both approaches estimate the answer. Throughout the interval $[a, b]$, the t values are evenly distributed. We employ a step size.

$$h = \frac{(b-a)}{N} \qquad (2.2)$$

Both methods, which are have been based on the Taylor Series expansion, use $f(t, y)$ to approximate the solution. The approaches employ a difference equation that estimates the solution w_{i+1}at the next time step t_{i+1} based on the estimated solution w_i at the present time (step) t_i

The difference equations yields N values $w_0\ , w_1, \dots., w_i\ , \dots.., w_N$ that are close to $y_0, y_1, \dots., y_i, \dots, y_N$ at times $t_0, t_1, \dots., t_i, \dots., t_N$ respectively. It is a difference equation because it depends on the quotient(difference) $\frac{\text{wi+1 −wi}}{h}$ $h = t_{i+1} + t_i$ (as per the approximation of $f(t, y)$).

We'll use the following example problem to demonstrate how these strategies function.

$$y' = 1 + \frac{y}{t}\ , \qquad 1 \leq t \leq 2, y(1) = 2, \text{ with } h = 0.25. \qquad (2.3)$$

Let's have a look at two of the specific techniques now.

2.1 EULER'S METHOD

At mesh point i , the approximate solution of Initial Value Problem 2.1. will be defined as W_i, and the t value at the ith mesh point will be defined as t_i . Using the following procedure, Euler's approach yields the approximate solution point (t_i, w_i):

$$t_0 = a\, w_0 = \alpha,$$
$$w_{i+1} = w_i + hf(t_i, w_i),\ i = 0,1,\ldots\ldots,N-1, \tag{2.4}$$
$$t_{i+1} = t_i + h,$$

where y is the exact solution's beginning value as provided in Equation 2.1. The difference equation of Euler's approach is Equation 2.4. N points (t_i, w_i) are obtained using this procedure.

-Example 2.3

We can see that $f(t, w) = 1 + \frac{w}{t}$

As per the known algorithm,

$$t_0 = 1, w_0 = 2$$

computing the difference equations

$w_1 = w_0 + h*f(t_0, w_0) = 2 + .25*(1 + 2/1) = 2.750000$, $t_1 = t_0 + h = 1.25$,
$w_2 = w_1 + h*f(t_1, w_1) = 2.75 + .25*(1 + 2.75/1.25) = 3.550000$, $t_2 = t_0 + 2h = 1.5$,
$w_3 = w_2 + h*f(t_2, w_2) = 3.55 + .25*(1 + 3.55/1.5) = 4.391667$, $t_3 = t_0 + 3h = 1.75$,
$w_4 = w_3+h*f(t_3, w_3) = 4.391667+.25*(1+4.391667/1.75) = 5.269047$, $t_4 = t_0+4h = 2$.

As a result, the approximation indicated in Table 2.1 is obtained using Euler's approach on Problem 2.3.

Table 2.1 Euler's Method Solution for the equation

i	t_i	w_i
0	1	2
1	1.25	2.760000
2	1.5	3.55
3	1.75	4.3891667
4	2	5.2789048

Initial Value Problem 2.3 is a straightforward problem that comes from a textbook. It does, in fact, have an exact solution. The exact solution is

$$y(t) = t(\ln|t| + 2\,),$$

which may be found using the linear equation technique of differential equations. We may now evaluate how well Euler's technique approximates the solution curve points. Table 2.2 may now display the error

$$|y(t_i) - w_i|$$

When we add a column with the actual values at the (t) stages.

Table 2.2 : Euler's method solutions and exact solutions

i	t_i	w_i	$y(t_i)$	error
0	1	2	2	0
1	1.25	2.7500	2.7789	0.0289
2	1.5	3.55	3.6082	0.0582
3	1.75	4.3916	4.4793	0.0877
4	2	5.2690	5.3863	0.1172

We can notice that the error grows as we move away from the starting point. When analysing a numerical approach like Euler's Method, error analysis is crucial.

2.1.1 Euler's Method Error Analysis

Because numerical approaches such as Euler's Approach do not discover accurate solutions, error plays a significant role in determining which method to adopt. The expansion of $y(t)$ around the point $t = t_0$ by Taylor Series is used in many of the methods. We know that $t_i = t_0 + ih$ in Euler's approach.
The Taylor polynomial of nth degree takes the form

$$Pn(t_i) = y(t_0) + y'(t_0)ih + \frac{y''(t_0)}{2!} i^2h^2 + \cdots + \frac{y^{(n)}(t_0)}{n!} i^nh^n \qquad (2.5)$$

The step size h in the formula is the same as in Euler's approach. The Taylor series has a n that goes to infinity. The series is truncated at the $(n + 1)th$ term by the nth Taylor polynomial. There is some kind of error in the Taylor polynomial and y when the series is truncated (t). The remaining term or truncation error is defined as follows:

$$R_n(t_i) = \frac{y^{(n+1)}(\xi(t))}{(n+1)!} i^{n+1}h^{n+1} \qquad (2.6)$$

This word aids us in determining the numerical method's inaccuracy. Because the Taylor polynomial of order 1 is used in Euler's approach, the remaining term is

$$R_n(t_i) = y'\big(\xi(t)\big)ih \tag{2.7}$$

So as $y'(t)$ is constrained by M on $[a, b]$, and hence the error reduces as the step size h decreases the remainder term . This is one way to reduce mistakes.
When the step size is smaller, the more calculations a computer has to perform.

2.2 Runge-Kutta

Runge-Kutta performs better than Euler's approach in terms of estimation. The second-order Midpoint technique is the simplest of the various Runge-Kutta methods. The function is used to estimate the second-order Taylor Polynomial.

$$a_1 f(t + \alpha_1, y + \beta_1). \tag{2.8}$$

When enlarged Taylor polynomials are used to solve the problem, we get

$$a_1 = 1,\ \alpha_1 = \tfrac{h}{2},\ \beta_1 = \tfrac{h}{2}.$$

The method of the midpoint is

$$w_0 = \alpha \tag{2.9}$$

$$w_{i+1} = w_i + hf\left(t_i + \tfrac{h}{2}, w_i + \tfrac{h}{2} f(t_i, w_i)\right), for\ i = 0,1, \ldots\ldots, N-1 \tag{2.10}$$

The difference equation is Equation 2.10. The step size in Equation 2.2 is used in the Midpoint technique. We cycle through the process finding w_i until we reach the end of the interval, just as Euler's approach. Consider the following scenario.

We'll utilize the example from Burden and Faires (2003), Equation 2.3, that we used to show Euler's Method. We'll use the Midpoint approach this time. We may derive $h =$

0.25 using Equation 2.3, which shows that w_0= 2. Using Equation 2.10, we find that the difference equation for the above problem using the Midpoint approach is

$$w_{i+1=} \frac{(4i2 + 38i + 90)wi + i2 \ + \ 9i \ + 20}{(4i2 + 34i + 72)}$$

We can now calculate the respective values at mesh points:

$$w_1 = \frac{(4(0)2 + 38i + 90)2 + (0)2 + 9(0) + 20}{(4(0)2 + 34(0) + 72)\ )} = 2.777778,$$

$$w_2 = \frac{(4(1)2 + 38i + 90)2.7777778 + (1)2 + 9(1) + 20}{(4(1)2 + 34(1) + 72)} = 3.606060.$$

As shown in Table 2.3, the process is repeated for all N points.
$y(t) = t(\ln|t| + 2)$ is the exact answer to this issue. We may use it to observe the way the error changes when we go closer to the interval's end and further away from the starting point.

Table 2.3 : Midpoint method solutions and exact solutions of the equation

i	t_i	*wi*	*y(ti)*	*error*
0	1	2	2	0
1	1.25	2.77778	2.7789	0.00115
2	1.5	3.60606	3.6082	0.00214
3	1.75	4.47630	4.4793	0.00303
4	2	5.38243	5.3863	0.00385

The fourth-order Runge-Kutta method is the most often used method out of the rest of the variants. It is more precise as compared to the Midpoint approach, but it is also more difficult. The exactly same step size as in Equation 2.2 is used by Runge-Kutta, and is the beginning value of y.
The procedure is as follows:

$$t_0 = a\,, w_0 = \alpha\,,$$
$$k_1 = hf(t_i, w_i),$$
$$k_2 = hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}, k_1\right),$$
$$k_3 = hf\left(t_i + \frac{h}{2}, w_i + \frac{1}{2}, k_2\right),$$
$$k_4 = hf(t_i + h, w_i + k_3),$$

$$w_{i+1} = w_i + 16(k_1 + 2k_2 + 2k_3 + k_4) \qquad i = 1,2, \ldots N - 1 \qquad (2.11)$$
$$t_i = a + ih$$

Applying Runge kutte fourth order method to the above stated problem.
The following is the initial iteration:

$$w0 = 2,$$

$$k1 = 0.25 * f(1, 2) = 0.25 * (1 + 2)/4 = 3/4 = 0.750000,$$
$$k2 = 0.25 * f\left(1 + 0.125, 2 + \frac{1}{2(0.75)}\right) = 0.25 * f(1.125, 2.375) = 0.777778,$$
$$k3 = 0.25 * f\left(1.125, 2 + \frac{1}{2(0.766778)}\right) = 0.25 * f(1.125, 2.387789)$$
$$= 0.25 * (1 + 2.387789/1.125) = 0.780642,$$
$$k4 = 0.25 * f(1.25, 2 + 0.78064) = 0.25 * (1 + 2.78064/1.25) = 0.806373,$$
$$w1 = 2 + \frac{1}{6}, (0.75 + 0.777778 + 0.780642 + 0.806373) = 2.778909.$$

2.2.1 The Runge-Kutta Method Error Analysis

Runge-Kutta methods, like Euler's approach, are based on Taylor series polynomials.The second-order Taylor series polynomial is used in second-order Runge-Kutta procedures like the Midpoint technique.

$$R_n(t_i) = \frac{y^2(\xi(t))}{2!} i^2 h^2 \qquad (2.12)$$

As far as the derivative $y^{(2)}$ is constrained by M on $[a, b]$ the Midpoint approach has an error of $O(h^2)$. The error in the Midpoint technique is less when compared to the error of Euler's technique because of $h^2 < h$ for step sizes smaller than 1.
The residual term of the fourth-order Runge-Kutta technique is the fourth-order Taylor polynomial. i.e.

$$R_n(t_i) = \frac{y^4(\xi(t))}{4!} i^4 h^4 \qquad (2.13)$$

.

As far as the derivative $y^{(4)}$ is bounded by M on $[a, b]$, then the error is $O(h^4)$ obviously better than any other approaches given thus far. With a significantly bigger step size, the fourth-order Runge-Kutta technique has a lower error than the Euler's technique or the Midpoint technique. To demonstrate, consider Problem 2.3, which was solved using Euler's technique with $h = 0.025$, the Midpoint technique with $h = 0.05$, and fourth-order Runge-Kutta with $h = 0.1$. Notice how, despite having the biggest step size, the fourth-order Runge-Kutta algorithm has the smallest error. Since Euler's technique has an error of $O(h^2)$ and the fourth-order Runge-Kutta has an error of $O(h^5)$ the Runge-Kutta error must be less, as indicated in the table.

It should be observed that fourth-order Runge-Kutta will require less calculations to achieve similar inaccuracy than the other approaches. Because when the step size is larger, the computer will be able to repeat the process several times to obtain precise estimates.

CHAPTER- 3
Numerical Methods For High Order Systems

3.1 Runge-Kutta (first order systems)

Runge-Kutta can also be used to solve initial value problems in systems. The form of an initial value problem with m equations is

$$\frac{du_1}{dt} = f_1(t, u_1, u_2, \ldots., u_m)$$
$$\frac{du_2}{dt} = f_2(t, u_1, u_2, \ldots., u_m)$$
$$\vdots$$
$$\frac{du_m}{dt} = f_m(t, u_1, u_2, \ldots., u_m) \tag{3.1}$$
$$u_1(a) = \alpha_1, u_2(a) = \alpha_2, \ldots\ldots, u_m(a) = \alpha_m$$

You specify a N specifying the number of mesh points to solve with the Runge-Kutta system. The step size is now given by Equation 2.2 and $t_j = a + jh$ for each $j = 0, 1,..., N$, just as it was before with Euler's and Runge-Kutta. The approximated solution of u_i in the system of Equations will be denoted w_{ij}
So the initial conditions give us

$$w_{1,0} = \alpha_1, w_{2,0} = \alpha_2, \ldots\ldots, w_{m,0} = \alpha_m$$

To get w_{j+1} using the already known w_j values, we must compute $k_1, k_2, k_3,$ and k_4 values, just as we did with the fourth-order Runge-Kutta method. Before moving on to the next k value , each k value for each equation in the system must be computed. As a result, for each $i = 1, 2,..., m$, we calculate;

$$k_{1,i} = hf_i(t_j, w_{1,j}, w_{2,j}, \ldots\ldots, w_{m,j}),$$

then for each $i = 1, 2, \ldots, m$ we calculate;

$$k_{2,i} = hf_i\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{1,1}, w_{2,j} + \frac{1}{2}k_{1,2}, \ldots\ldots, w_{m,j} + \frac{1}{2}k_{1,m}\right),$$

then for each $i = 1, 2, \ldots, m$ we calculate;

$$k_{3,i} = hf_i\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{2,1}, w_{2,j} + \frac{1}{2}k_{2,2}, \ldots\ldots, w_{m,j} + \frac{1}{2}k_{2,m}\right),$$

then for each $i = 1, 2, \ldots, m$ we calculate,

$$k_{4,i} = hf_i\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{3,1}, w_{2,j} + \frac{1}{2}k_{3,2}, \ldots\ldots, w_{m,j} + \frac{1}{2}k_{3,m}\right),$$

and then for each $i = 1, 2, \ldots, m$

$$w_{i,j+1} = w_{i,j} + \frac{1}{6}(k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i})$$

It can assist to see a simple example. We calculate a total of twelve k values and three difference equations to depict the calculations for just one step in a system of three equations. Implementing the procedure on a computer and analysing the outcomes is the best approach to demonstrate it.

$$\begin{aligned}
p'_1 &= p_2 - p_3 + t, \; p_1(0) = 1; \\
p'_2 &= 3t^2, \; p_2(0) = 1; \\
p'_3 &= p_2 + e^{-t}, \; p_3(0) = -1
\end{aligned} \tag{3.2}$$

for $t \in [0, 1]$. We used a step size of $h = 0.1$ and were given the exact solutions of

$$p_1(t) = -0.05*t^5 + 0.25*t^4 + t + 2 - e^{-t},$$
$$p_2(t) = t^3 + 1, \text{ and } p_3(t) = 0.25*t^4 + t - e^{-t}.$$

The fourth -order Runge-Kutta for the above system of equations produced the values for $w_{i,j}$

Table 2.4 : Runge kutta method values and exact solutions for the equation

j	*t_i*	*$w_{1,j}$*	*$p_{1,j}$*	*$w_{2,j}$*	*$p_{2,j}$*	*$w_{3,j}$*	*$p_{3,j}$*
0	0	1	1	1	1	-1	-1
4	.3	1.5610	1.5610	1.027	1.027	-0.4388	-0.43879323
7	.6	2.0796	2.0798	1.216	1.216	0.0836	0.08358836
9	.8	2.4367	2.436	1.512	1.512	0.453	0.45307106
11	1	2.8321	2.8321	2	2	0.882	0.8821205

3.2 Runge-Kutta Method for Higher-Order Systems

So far, all of the methods have been used to solve a variety of ordinary differential equations. Regrettably, as we'll see in Chapter 4, the Optimal Velocity Model we're going to assess is a system of second-order differential equations. Now we'll proceed by conversion : we will convert all second order differential equations to two first order differential equations.
Consider the following equation of second order:

$$\frac{d^2y}{dt^2} = f(t, y, \frac{dy}{dt}) , y(a) = \alpha_1, \frac{dy}{dt} (a) = \alpha_2. \qquad (3.3)$$

The system of equations is then constructed as follows:

$$\frac{dy}{dt} = v,$$
$$\frac{dv}{dt} = f(t, y, v)$$

This is now just a two first-order differential equations system. For systems, we may now employ the fourth-order Runge-Kutta approach.

CHAPTER-4

A Traffic Model's Numerical Solution

The Optimal Velocity Model, referred as OVM, for traffic flow is discussed in this chapter. Analytically, the initial-value problem cannot be solved. The Runge-Kutta method will be used to approximate the solution. We demonstrate the procedure and describe how and why the Runge-Kutta method can be used to a traffic flow model.

4.1 THE MODEL

The following equation is used to describe the movement of automobile i.

$$\frac{d^2x_i(t)}{dt^2} = a(V(\Delta x_i(t)) - \frac{dx_i(t)}{dt} \qquad (4.1)$$

At time t, the position of automobile I is $x_i(t)$. The distance between the two cars (car i and the car in front of it) at time t is defined as $\Delta x_i(t)$.

$$\Delta x_i(t) = x_{i+1}(t) - x_i(t). \qquad (4.2)$$

The headway of car i at time t is a common term for this. The sensitivity parameter, a, is a constant. It describes the average driver's sensitivity to the motion of the vehicle in front of him. The time lag when a front car changes speeds and the automobile behind it reactes and then adapting to the front car is the inverse of the sensitivity parameter. $\frac{dx_i(t)}{dt}$ is the velocity of the car i.
The optimal velocity of the car is denoted by :

$$V\,(\Delta x_i(t)) = \frac{Vmax}{2}\,[tanh\,(\Delta x_i(t) - x_c) + tanh(x_c)]. \qquad (4.3)$$

If $x_c = 4$, the optimal velocity $V\,(x_i(t))$ approaches a value in close bracket of 0.0004 percent of Vmax, as the headway $x_i(t)$ approaches infinity. As a result, when the headway is sufficiently larger than the safety distance, cars will approach their

maximum speed. The value of $V\ (\Delta x_i(t))$ will approach 0 if the safety distance gets arbitrarily long. The acceleration is positive when the ideal velocity in Equation 4.3 is larger than the car's current velocity $\frac{dx_i(t)}{dt}$. In this instance, the car's speed will increase. The acceleration is 0 when the ideal velocity is same as the present velocity. As a result, the car's speed remains constant. When the present speed exceeds the optimal speed, the acceleration becomes negative, and thus the car slows down.
A proportionality parameter is present in the acceleration. A high sensitivity parameter means a quick response time because the sensitivity parameter is equal to the inverse of the time lag it takes for a driver to react to a car in front of it. As a result of the high sensitivity parameter, the acceleration is increased.
We'll use the following parameters to take the calculations forward.

$$a = 1,$$
$$Vmax = 4,$$
$$x_c = 4.$$

Over the t interval $[0, 6]$, we ran each Runge-Kutta technique with a step size of 0.001.

4.2 Runge-Kutta Implementation on the Model

A series of second-order differential equations governs the Optimal Velocity Model mentioned above. As mentioned in Subsection 3.2, each second-order differential equation must be transformed into two subsequent first-order differential equations.

$$\frac{dx_i(t)}{dt} = y_{i(t)}\ and\ \frac{dy_i(t)}{dt} = a\left(V\left(x_{i+1}(t) - x_i(t)\right) - y_i(t)\right). \tag{4.4}$$

There are two differential equations associated with each automobile i. As a result, suppose there are M cars in our system, then there are 2M equations.
To implement a fourth-order Runge-Kutta technique, we must first define a starting condition for each equation. Each car in the system must have a starting position and velocity in this model.There is no way to duplicate their precise simulation without knowing the start conditions. We establish our own starting conditions.

The Optimal Velocity Equation 4.3 also has an automobile in front of the ith automobile, which is a problem. The system's final equation is

$$\frac{dy_M(t)}{dt} = a\left(V\left(x_{M+1}(t) - x_M(t)\right) - y_M(t)\right).$$

This is predicated on the existence of a $(M + 1)$ automobile, which does not exist. So, what to with the Mth car's motion as part of the system?
The entire system is based on how to programme the lead car's behaviour, which influences the behaviour of the automobiles behind it. We set our own values for x_{M+1} **.** To evaluate the approach and model's application, we chose to simplify the model to only one automobile, then two automobiles.

4.3 One Car and a Stopped Object Simulation

This simulation depicts a car's reaction to a halted item. We go close enough to the thing that the car then has to slow down.
For this simulation, we'll use Equation 4.4's first-order system for a single car's position $x_1(t)$ and velocity $y_1(t)$. We discussed $x_2(t)$ as the position of the second automobile in the system in Section 4.1, but now we consider it to be an item halted at position 20.

Through the entire process, we will make use of : $x_2(t) = 20$.
And then the system becomes

$$\frac{dx_1(t)}{dt} = y_1(t),\ \frac{dy_1(t)}{dt} = a(V\left(20 - x_1(t)\right) - y_1(t))$$

The car begins at a position $x_1(0) = 0$.
The car's start velocity is set to be the ideal velocity as follows:

$$\frac{dx_1}{dt}\Big|_{t=0} = V\ (\Delta x_1(0))$$

where $\Delta x_1(0) = x_2(0) - x_1(0) = 20 - 0 = 20$
is headway at time 0

We may now solve for the car's position $x1(t)$ and velocity $y1(t)$ using the fourth-order Runge-Kutta method.

The figure below shows the result for the experiment. The left one shows the position of car versus time , the right one shows velocity versus time.

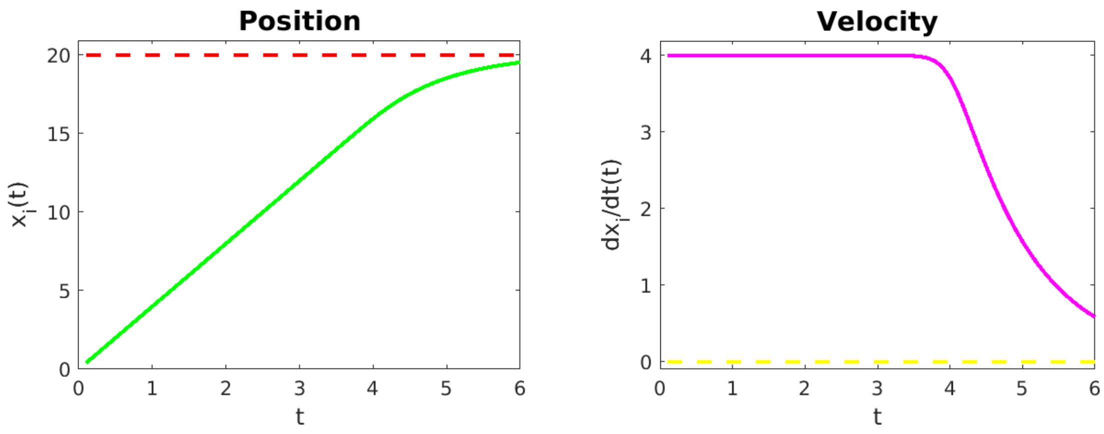

Figure 4.1 : Position versus time and velocity versus time when approaches to the stopped object (dashed line) .

4.4 Simulation With a clear road in front of the car

This simulation depicts a car's reaction to a clean road ahead of it. Then we place the thing far enough away from the car that it has no effect on it.
The first-order system for the position x_1 is used in this simulation (t) and velocity $y1(t)$ of a single car.

We think of $x2(t)$ as an item halted at the far distant point of 100, much as we did with the stopped car simulation. $x2(t) = 100$ is used throughout the simulation. As a result, the system is

$$\frac{dx_1(t)}{dt} = y_1(t), \ \frac{dy_1(t)}{dt} = a(V(100 - x_1(t)) - y_1(t))$$

The car begins at position $x1(0) = 0$.
The car's velocity is initially set to a sluggish speed of 1, recalling that the Vmax is four in Section 4.1.

As a result

$$\frac{dx_1}{dt}\Big|_{t=0} = 1$$

Starting the car slowly will allow it to pick up speed as the scenario progresses.We may now solve the car's position $x_1(t)$ and velocity $y_1(t)$ using the fourth-order Runge-Kutta method.

The simulation's findings are shown in Figure 4.2. The position of the car as a function of time is depicted in the left picture. The velocity is depicted in the right figure. Figure 4.2's right graph depicts the car moving up to attain its top speed of four miles per hour. The right graph's concavity indicates that the car is speeding because it is concave up.

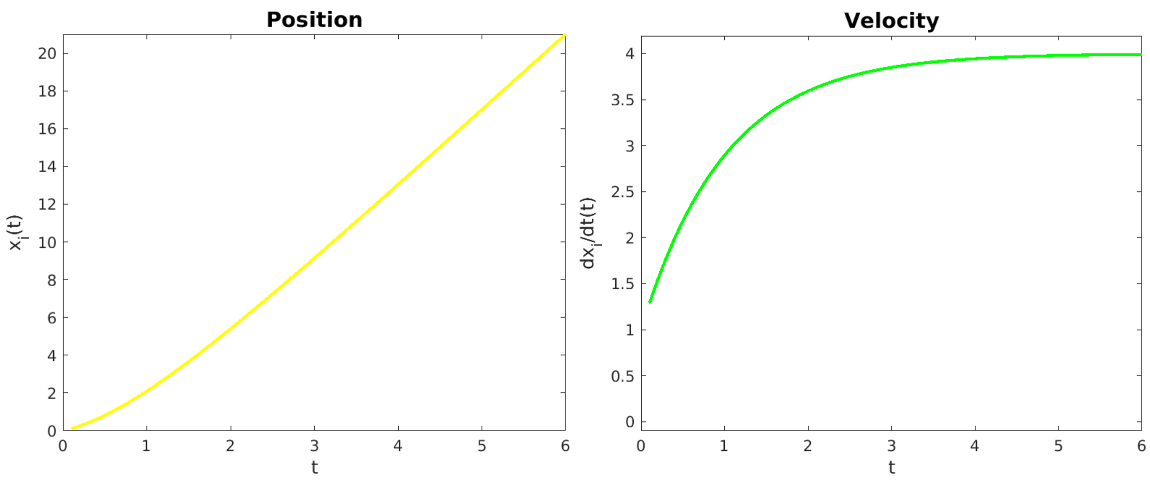

Figure 4.2: Position versus time (left graph) and velocity versus time (right graph) for a car starting slowly and accelerating due to the clear path in front..

4.5 Simulating an Object Traveling at a Constant Slow Speed

This simulation depicts a car's reaction to any type of slow-moving vehicle in front of it. We'll start with the two cars with minimum distance between them so that the back car will have to slow down.
For this simulation, we'll use Equation 4.4's first-order system for a single car's position $x1(t)$ and velocity $y_1(t)$. $x_2(t)$ does describe a car's motion in this scenario. Throughout the experiment, we give it a predetermined velocity of two and a starting position of ten. As a result.

$$x_2(t) = 10 + 2t.$$

Thus, the system is

$$\frac{dx_1(t)}{dt} = y_1(t), \ \frac{dy_1(t)}{dt} = a(V\big((10+2t) - x_1(t)\big) - y_1(t))$$

Car 1 begins at a position $x_1(0) = 5$.

The car's start velocity is set to be the ideal velocity as follows:

$$\frac{dx_1}{dt}\Big|_{t=0} = V\ (\Delta x_1(0))$$

where $\Delta x_1(0) = x_2(0) - x_1(0) = 10 - 5 = 5$
is the headway at time zero.
We may now solve for the car's position $x1(t)$ and velocity $y1(t)$ using the fourth-order Runge-Kutta method.

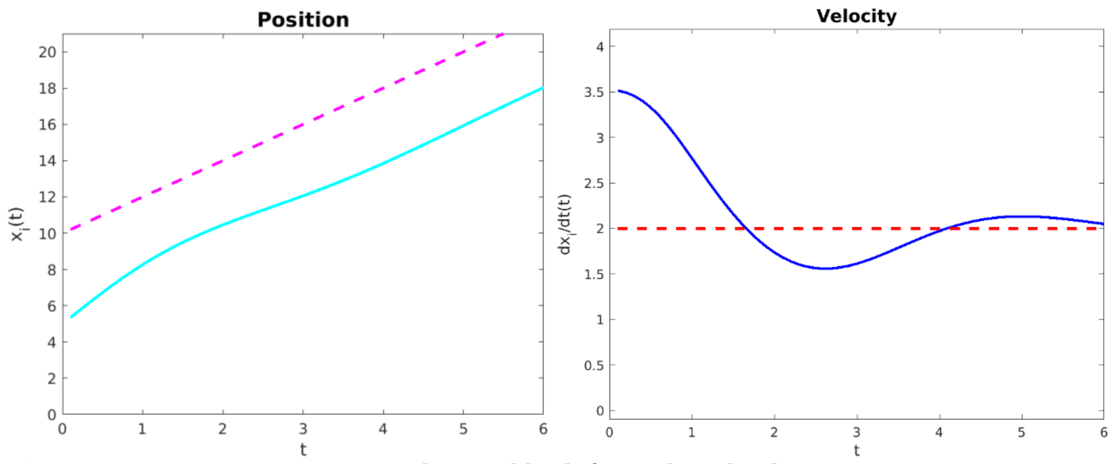

Figure 4.3: Position versus time depicted by left graph and velocity versus time depicted by right graph of a car depicted by solid line with a slow moving constant speed object depicted by dashed line.

The simulation's findings are shown in Figure 4.3. The position of the two cars are shown as functions of time in the left figure. The velocities are depicted in the right figure. In the graphs the car slows down to match the slower moving car's speed.

4.6 Two Cars and a Stopped Object Simulation

The reaction of two cars to a halted item is depicted in this simulation. This simulation is similar to the one in Section 4.3, except that we add another automobile to study how the dynamics change.
The first-order system in Equation 4.4 is used in this simulation for the coordinates *xi (t)* and velocity *$y_i(t)$. $x_3(t)$* is the position of a halted object at 20, similar to x_2(t) in Section 4.3. x_3(t) = 20 is used throughout the simulation.

Thus, the system is

$$\frac{dx_i(t)}{dt} = y_i(t),\ \frac{dy_i(t)}{dt} = a(V\left(x_{i+1}(t) - x_i(t)\right) - y_i(t))\ ,\ where\ i\ =\ 1, 2.$$

The headways of the cars are

$$\Delta x_1(t)\ =\ x_2(t) -\ x_1(t)$$

And

$$\Delta x_2(0)\ =\ 20 -\ x_2(t)$$

The cars begin at positions *$x_1(0) = 0$ and $x_2(0) = 5$.*

The cars' initial velocity is set to be the ideal velocity as follows:

$$\frac{dx_1}{dt}\Big|_{t=0} = V\,(\Delta x_i(0))\ where\ i\ =\ 1, 2.$$

Now we can solve for the coordinates *xi(t)* and velocities *yi(t)* using the fourth-order Runge-Kutta technique, $where\ i\ =\ 1, 2.$

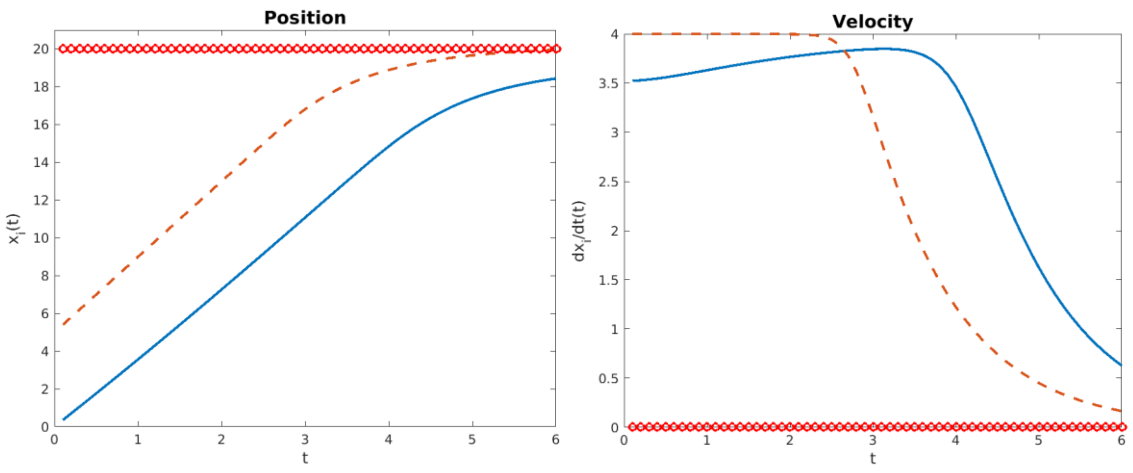

Figure 4.4: Position versus time (left graph) and velocity versus time (right graph) as a stopped object is being approached by two cars(solid and dashed lines).

The simulation's findings are shown in Figure 4.4. The position of the cars is depicted on the left diagram. The velocities are depicted in the right figure. Car2 slows down as it approached the halted item in both graphs. Car 1 (solid line) accelerates somewhat faster than car 2, reaches top speed, and then slows down.

4.8 Two Cars and a Simulated Object Traveling at a Slow Constant Speed

This simulation shows how two cars react to a slow, continuously moving object.
This simulation is similar to the one in depicted in 4.5, except that we now added another automobile to study how the dynamics of the process changes.
For this simulation, we'll use Equation 4.4 to get the coordinates $x_i(t)$ and velocities $y_i(t)$ when i is one and subsequently two. We assume $x_3(t)$ to represent the position of an automobile driving at a constant velocity of three and starting at position ten, as we did in Section 4.5. As a result

$$x_3(t)\ =\ 10\ +\ 3t$$

all during the simulation As a result, the system is

$$\frac{dx_i(t)}{dt} = y_i(t),\ \frac{dy_i(t)}{dt} = a(V(x_{i+1}(t) - x_1(t)) - y_i(t))\ ,\ where\ i\ =\ 1, 2.$$

where

$$\Delta x_1(t)\ =\ x_2(t) -\ x_1(t)$$

And

$$\Delta x_2(0) = (10 + 3t) -\ x_2(t)$$

The cars begin at positions *$x_1(0) = 0$, $x_2(0) = 5$, and $x_3(0) = 10$.v 5rfgv v 5rfgv 4r*
The cars' initial velocity is set to be the ideal velocity as follows:

$$\frac{dx_1}{dt}|_{t=0} = V\ (\Delta x_i(0))$$

Now we can solve for the coordinates *$x_i(t)$* and velocities *$y_i(t)$* using the fourth-order Runge-Kutta technique, *where i = 1, 2.*
The simulation's findings are shown in Figure 4.5. The position of the autos as a function of time is depicted in the left picture. The velocities are depicted in the right figure. Automobile 2 depicted by dashed line slows down to adjust to the slow speed of the item ahead depicted by circle line in Figure 4.5, and car 1 (solid line) follows suit.

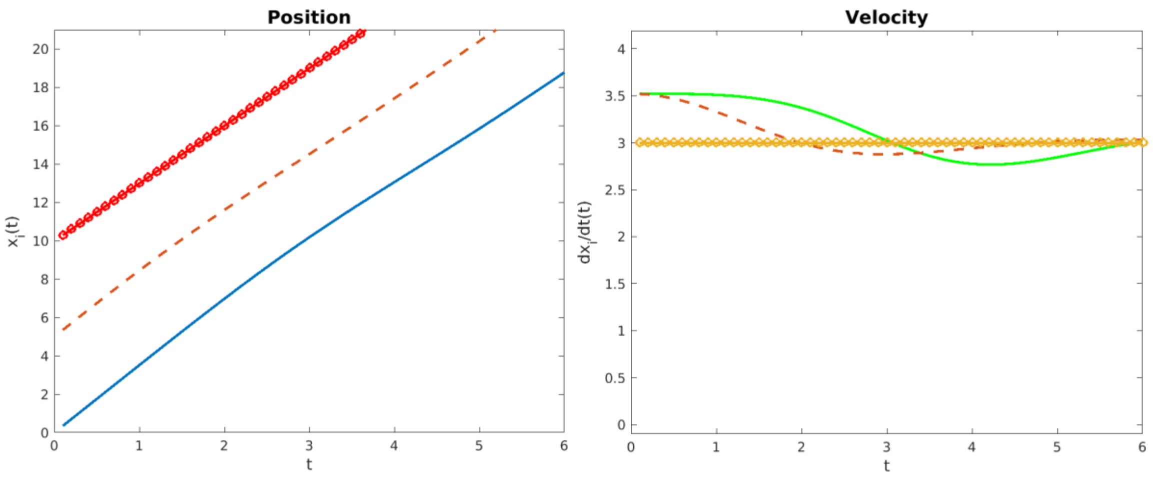

Figure 4.5: Position versus time depicted by left graph and velocity versus time depicted by right graph as two cars depicted by solid line and dashed line slows down for a slow moving object depicted by circle line.

CHAPTER 5
Conclusion

In this paper, we show how to use the Optimal Velocity Model to implement the fourth-order Runge-Kutta algorithm. The lead car's start conditions and base cases were specified and addressed. We further extended Kurata and Nagatani (2003)'s dynamics by imposing a small range of the leading automobile behaviours as well as a number of start and halt conditions.
The performance of the fourth-order Runge-Kutta approach yielded digestory results for the Optimal Velocity Model since the traffic simulations accurately matched real life scenarios. This study gave a different approach the Range Kutta Method.

REFERENCES

1. Burden, R. L., & Faires, J. D. (2003). Numerical analysis. Boston, MA: Brooks/Cole.
2. Chen, J., Peng, Z., & Fang, Y. (2014). Effects of car accidents on three-lane traffic flow.
3. Mathematical Problems in Engineering, 2014 (413852).
4. Kurata, S., & Nagatani, T. (2003). Spatio-temporal dynamics of jams in two-lane traffic
5. Physica A: Statistical Mechanics and Its Applications, 318 ,537–550.
6. MATLAB. (2014). Version 8.3 (r2014a).
7. https://en.m.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods
8. https://en.m.wikipedia.org/wiki/Euler_method
9. https://en.m.wikipedia.org/wiki/Traffic_flow
10. https://www.math24.net/higher-order-linear-homogeneous-differential-equations-constant-coefficients#:~:text=Higher%20Order%20Linear%20Homogeneous%20Differential%20Equations%20with%20Constant%20Coefficients,-The%20linear%20homogeneous&text=y(n)(x),y(x)%3D0%2C&text=is%20called%20the%20characteristic%20equation,exactly%20n%20roots%2C%20counting%20multiplicity

Appendix A

Code Implementation

Code for traffic model

A.1 Code for the Traffic Model Equation

The below 3 functions gives the optimal velocity model

vdot : a * (Optimal Velocity Model)

```
function x = vdot( u1,u2, y)
x=V(u1, u2)-y;
end
```

V : Equation 4.3- Optimal Velocity Equation

```
function p= V(u1,u2)
p=4/2*(tanh((u2-u1)-4)+tanh(4));
%this is the velocity function
end
```

xdot : gives the value for the velocity of the car.

```
function u = xdot( y )
u=y;
end
```

A.2 Code for applying Runge Kutta method to one car and stopped object

This program plots a graph of a car moving from zero to an object stopped at 20

```
function c = RungeKuttaStopped( )
function p= V(u1,u2)
p=4/2*(tanh((u2-u1)-4)+tanh(4));
%this is the velocity function
end
function x = vdot( u1,u2, y)
x=V(u1, u2)-y;
end
function u = xdot( y )
u=y;
end
lane1=[0 20];
%lane1 is the initial positions of cars
v1=V(lane1(1),lane1(2));
%v are the initial velocity of the cars
h=.001;
disp(v1);
count=1;
fileID=fopen('RungeKuttaStopped.dat','w');
for t=1:6000
kx2(1)=h*xdot(v1);
kv2(1)=h*vdot(lane1(1),lane1(2), v1);
kx2(2)=h*xdot(v1+1/2*kv2(1));
kv2(2)=h*vdot(lane1(1)+1/2*kx2(1),lane1(2),v1+1/2*kv2(1));
kx2(3)=h*xdot(v1+1/2*kv2(2));
kv2(3)=h*vdot(lane1(1)+1/2*kx2(2),lane1(2),v1+1/2*kv2(2));
kx2(4)=h*xdot(v1+kv2(3));
kv2(4)=h*vdot(lane1(1)+kx2(3),lane1(2),v1+kv2(3));
lane1(1)=lane1(1)+(kx2(1)+2*kx2(2)+2*kx2(3)+kx2(4))/6;
v1=v1+(kv2(1)+2*kv2(2)+2*kv2(3)+kv2(4))/6;
if (mod(t,100)==0)
rkplot(count,:)=[t*h lane1(1) lane1(2) v1 0];
count=1+count;
end


if (mod(t,100)==0)
fprintf(fileID, '%4.7f %4.7f \r\n', lane1);
end
```

```
end

fclose(fileID);

figure

%to plot the required graphs

plot(rkplot(:,1),rkplot(:,2),'-g',...

rkplot(:,1),rkplot(:,3),'--r','LineWidth',2);

title('Position','FontSize',14)

xlabel('t','FontSize',12)

ylabel('x_i(t)','FontSize',12)

axis([0 6 0 21])

figure

plot(rkplot(:,1),rkplot(:,4),'-m',...

rkplot(:,1),rkplot(:,5),'--y','LineWidth',2);

title('Velocity','FontSize',14)

xlabel('t','FontSize',12)

ylabel('dx_i/dt(t)','FontSize',12)

axis([0 6 -.1 4.2])

c=[lane1(1) lane1(2)];

end
```

A.3 Code for applying Runge Kutta method to clear road in front of the car

This program plots a graph of a car starting from zero and approaching a far away object at 100.

```
function c = RungeKuttaClearPath( )
function c= V(u1,u2)
c=4/2*(tanh((u2-u1)-4)+tanh(4));
end
function x = vdot( u1,u2, y)
x=V(u1, u2)-y;
end
function u = xdot( y )
u=y;
end
lane1=[0 100];
%lane1 is the initial positions of cars
v1=1;
%v are the initial velocity of the cars
h=.001;
disp(v1);
count=1;
fileID=fopen('RungeKuttaStopped.dat','w');
for t=1:6000
kx2(1)=h*xdot(v1);
kv2(1)=h*vdot(lane1(1),lane1(2), v1);
kx2(2)=h*xdot(v1+1/2*kv2(1));
kv2(2)=h*vdot(lane1(1)+1/2*kx2(1),lane1(2),v1+1/2*kv2(1));
kx2(3)=h*xdot(v1+1/2*kv2(2));
kv2(3)=h*vdot(lane1(1)+1/2*kx2(2),lane1(2),v1+1/2*kv2(2));
```

```
kx2(4)=h*xdot(v1+kv2(3));
kv2(4)=h*vdot(lane1(1)+kx2(3),lane1(2),v1+kv2(3));
%final values
lane1(1)=lane1(1)+(kx2(1)+2*kx2(2)+2*kx2(3)+kx2(4))/6;
v1=v1+(kv2(1)+2*kv2(2)+2*kv2(3)+kv2(4))/6;
if (mod(t,100)==0)
rkplot(count,:)=[t*h lane1(1) lane1(2) v1 0];
count=1+count;
end
if (mod(t,100)==0)
fprintf(fileID, '%4.7f %4.7f \r\n', lane1);
end
end
fclose(fileID);
%to plot the respective graphs
figure
plot(rkplot(:,1),rkplot(:,2),'-y',...
'LineWidth',2);
title('Position','FontSize',14)
xlabel('t','FontSize',12)
ylabel('x_i(t)','FontSize',12)
axis([0 6 0 21])
figure
plot(rkplot(:,1),rkplot(:,4),'-g',...
'LineWidth',2);
title('Velocity','FontSize',14)
xlabel('t','FontSize',12)
ylabel('dx_i/dt(t)','FontSize',12)
axis([0 6 -.1 4.2])
c=[lane1(1) lane1(2)];
end
```

A.4 Code for applying Runge Kutta method to an object moving at slow constangt speed is being followed by one car

This program plots a graph of a car which starts at five approaching an object which starts at ten moves at a speed of two.

```
function c = RungeKuttaConstant( )
function c= V(u1,u2)
c=4/2*(tanh((u2-u1)-4)+tanh(4));
end
function x = vdot( u1,u2, y)
x=V(u1, u2)-y;
end
function u = xdot( y )
u=y;
end
lane1=[5 10];
%lane1 is the initial positions of cars
v1=V(lane1(1),lane1(2));
v2=2;
%v are the initial velocity of the cars
h=.001;
disp(v1);
count=1;
fileID=fopen('RungeKuttaConstant.dat','w');
for t=1:6000
kx1(1)=h*xdot(v1);
kv1(1)=h*vdot(lane1(1),lane1(2), v1);
kx1(2)=h*xdot(v1+1/2*kv1(1));
kv1(2)=h*vdot(lane1(1)+1/2*kx1(1),lane1(2),v1+1/2*kv1(1));
kx1(3)=h*xdot(v1+1/2*kv1(2));
```

```
kv1(3)=h*vdot(lane1(1)+1/2*kx1(2),lane1(2),v1+1/2*kv1(2));
kx1(4)=h*xdot(v1+kv1(3));
kv1(4)=h*vdot(lane1(1)+kx1(3),lane1(2),v1+kv1(3));
%final values
lane1(1)=lane1(1)+(kx1(1)+2*kx1(2)+2*kx1(3)+kx1(4))/6;
lane1(2)=10+v2*t*h;
v1=v1+(kv1(1)+2*kv1(2)+2*kv1(3)+kv1(4))/6;
if (mod(t,100)==0)
rkplot(count,:)=[t*h lane1(1) lane1(2) v1 v2];
count=1+count;
disp(v2);
disp(t*h);
disp(lane1(2));
end
if (mod(t,100)==0)
fprintf(fileID, '%4.7f %4.7f \r\n', lane1);
end
end
fclose(fileID);
%the graphs to be plot
figure
plot(rkplot(:,1),rkplot(:,2),'-c',...
rkplot(:,1),rkplot(:,3),'--m','LineWidth',2);
title('Position', 'FontSize', 14)
xlabel('t','FontSize',12)
```

```
ylabel('x_i(t)','FontSize', 12)
axis([0 6 0 21])
figure
plot(rkplot(:,1),rkplot(:,4),'-b',...
rkplot(:,1),rkplot(:,5),'--r','LineWidth',2);
title('Velocity', 'FontSize', 14)
xlabel('t','FontSize',12)
ylabel('dx_i/dt(t)','FontSize',12)
axis([0 6 -.1 4.2])
c=[lane1(1) lane1(2)];
end
```

A.5 Code for applying Runge Kutta method to two cars and a stopped object

This program plots a graph of two cars which starts at zero and five approaching an object which is stopped at 20.

```
function c = RungeKuttaStopped2( )
function c= V(u1,u2)
c=4/2*(tanh((u2-u1)-4)+tanh(4));
end
function x = vdot( u1,u2, y)
x=V(u1, u2)-y;
end
function u = xdot( y )
u=y;
end
lane1=[0 5 20];
%lane1 is the initial positions of cars
v1=V(lane1(1),lane1(2));
v2=V(5,20);
%v are the initial velocity of the cars
h=.001;
disp(v1);
count=1;
fileID=fopen('RungeKuttaDecel.dat','w');
for t=1:6000
kx1(1)=h*xdot(v1);
kv1(1)=h*vdot(lane1(1),lane1(2), v1);
kx2(1)=h*xdot(v2);
kv2(1)=h*vdot(lane1(2),lane1(3), v2);
kx1(2)=h*xdot(v1+1/2*kv1(1));
```

```
kv1(2)=h*vdot(lane1(1)+1/2*kx1(1),lane1(2)+1/2*kx2(1),v1+1/2*kv1(1));
kx2(2)=h*xdot(v2+1/2*kv2(1));
kv2(2)=h*vdot(lane1(2)+1/2*kx2(1),lane1(3),v2+1/2*kv2(1));
kx1(3)=h*xdot(v1+1/2*kv1(2));
kv1(3)=h*vdot(lane1(1)+1/2*kx1(2),lane1(2)+1/2*kx2(2),v1+1/2*kv1(2));
kx2(3)=h*xdot(v2+1/2*kv2(2));
kv2(3)=h*vdot(lane1(2)+1/2*kx2(2),lane1(3),v2+1/2*kv2(2));
kx1(4)=h*xdot(v1+kv1(3));
kv1(4)=h*vdot(lane1(1)+kx1(3),lane1(2)+1/2*kx2(3),v1+kv1(3));
kx2(4)=h*xdot(v2+kv2(3));
kv2(4)=h*vdot(lane1(2)+kx2(3),lane1(3),v2+kv2(3));
lane1(1)=lane1(1)+(kx1(1)+2*kx1(2)+2*kx1(3)+kx1(4))/6;
lane1(2)=lane1(2)+(kx2(1)+2*kx2(2)+2*kx2(3)+kx2(4))/6;
v1=v1+(kv1(1)+2*kv1(2)+2*kv1(3)+kv1(4))/6;
v2=v2+(kv2(1)+2*kv2(2)+2*kv2(3)+kv2(4))/6;
if (mod(t,100)==0)
rkplot(count,:)=[t*h lane1(1) lane1(2) v1 v2];
count=1+count;
end
if (mod(t,100)==0)
fprintf(fileID, '%4.7f %4.7f \r\n', lane1);
end
end
fclose(fileID);
figure
plot(rkplot(:,1),rkplot(:,2),'-',...
rkplot(:,1),rkplot(:,3),'--',rkplot(:,1),20,'ro-','LineWidth',2);
title('Position','FontSize',14)
xlabel('t','FontSize',12)
ylabel('x_i(t)','FontSize',12)
axis([0 6 0 21])
figure
plot(rkplot(:,1),rkplot(:,4),'-',...
rkplot(:,1),rkplot(:,5),'--',rkplot(:,1),0,'ro-','LineWidth',2);
title('Velocity','FontSize',14)
xlabel('t','FontSize',12)
ylabel('dx_i/dt(t)','FontSize',12)
c=[lane1(1) lane1(2)];
end
```

A.6 Code for applying Runge Kutta method to two cars follows an object with slow constant speed

This program plots a graph of cars which starts at zero and five approaches an object which starts at ten moving at a speed of three.

```
function c = RungeKuttaConstant2( )
function c= V(u1,u2)
c=4/2*(tanh((u2-u1)-4)+tanh(4));
end
function x = vdot( u1,u2, y)
x=V(u1, u2)-y;
end
function u = xdot( y )
u=y;
end
lane1=[0 5 10];
%lane1 is the initial positions of cars
v1=V(lane1(1),lane1(2));
v2=V(lane1(2),lane1(3));
v3=3;
%v are the initial velocity of the cars
h=.001;
disp(v1);
count=1;
fileID=fopen('RungeKuttaConstant.dat','w');
for t=1:6000
kx1(1)=h*xdot(v1);
kv1(1)=h*vdot(lane1(1),lane1(2), v1);
kx2(1)=h*xdot(v2);
kv2(1)=h*vdot(lane1(2),lane1(3), v2);
```

```
kx1(2)=h*xdot(v1+1/2*kv1(1));
kv1(2)=h*vdot(lane1(1)+1/2*kx1(1),lane1(2)+1/2*kx2(1),v1+1/2*kv1(1));
kx2(2)=h*xdot(v2+1/2*kv2(1));
kv2(2)=h*vdot(lane1(2)+1/2*kx2(1),lane1(3),v2+1/2*kv2(1));
kx1(3)=h*xdot(v1+1/2*kv1(2));
kv1(3)=h*vdot(lane1(1)+1/2*kx1(2),lane1(2)+1/2*kx2(2),v1+1/2*kv1(2));
kx2(3)=h*xdot(v2+1/2*kv2(2));
kv2(3)=h*vdot(lane1(2)+1/2*kx2(2),lane1(3),v2+1/2*kv2(2));
kx1(4)=h*xdot(v1+kv1(3));
kv1(4)=h*vdot(lane1(1)+kx1(3),lane1(2)+1/2*kx2(3),v1+kv1(3));
kx2(4)=h*xdot(v2+kv2(3));
kv2(4)=h*vdot(lane1(2)+kx2(3),lane1(3),v2+kv2(3));
lane1(1)=lane1(1)+(kx1(1)+2*kx1(2)+2*kx1(3)+kx1(4))/6;
lane1(2)=lane1(2)+(kx2(1)+2*kx2(2)+2*kx2(3)+kx2(4))/6;
v1=v1+(kv1(1)+2*kv1(2)+2*kv1(3)+kv1(4))/6;
v2=v2+(kv2(1)+2*kv2(2)+2*kv2(3)+kv2(4))/6;
lane1(3)=10+v3*t*h;
if (mod(t,100)==0)
rkplot(count,:)=[t*h lane1(1) lane1(2) v1 v2 lane1(3)];
count=1+count;
disp(v2);
disp(t*h);
disp(lane1(2));
end
if (mod(t,100)==0)
fprintf(fileID, '%4.7f %4.7f \r\n', lane1);
end
end
fclose(fileID);
figure
plot(rkplot(:,1),rkplot(:,2),'-',...
rkplot(:,1),rkplot(:,3),'--',rkplot(:,1),rkplot(:,6),'ro-','LineWidth',2);
title('Position','FontSize',14)
xlabel('t','FontSize',12)
ylabel('x_i(t)','FontSize',12)
axis([0 6 0 21])
figure
plot(rkplot(:,1),rkplot(:,4),'-',...
rkplot(:,1),rkplot(:,5),'--',rkplot(:,1),3,'ro-','LineWidth',2);
title('Velocity','FontSize',14)
xlabel('t','FontSize',12)
ylabel('dx_i/dt(t)','FontSize',12)
axis([0 6 -.1 4.2])
c=[lane1(1) lane1(2)];
end
```