

ACCA5017_Richard Hankey

by Richard Hankey

FILE	6496_RICHARD_HANKEY_ACCA5017_RICHARD_HANKEY_234537_91727 281.DOCX (105.1K)		
TIME SUBMITTED	07-FEB-2020 10:53PM (UTC+0000)	WORD COUNT	2800
SUBMISSION ID	119779871	CHARACTER COUNT	15803

University of Wales Trinity Saint David

Richard Hankey

P151406

ACCA5017

Games Architecture and Mechanics

Assignment 1

Title:

Part 1a: Games Physics

Contents

Introduction	3
Task 1: Numerical Integrations	4
Merits and Disadvantages.....	6
Improving the Simulation.	6
Task 1: Implementation	7
Task 2: Vector Class.....	8
Task 2: Plane Class	10
Task3: Sphere To Sphere Collision	12
References	12
Appendix 1: Numerical Integration and Dragster Class	13
Appendix 2: Main.cpp	20

Introduction

This report will start off with a discussion and critical evaluation of the Euler's numerical integration method, examining the merits and drawbacks of the method. The report will then demonstrate an implementation of the method, showing the output of these calculations in the form of a line chart.

Moving on, this report will further the critique; the detailing of the Velocity Verlet method and its advantages over Euler's method, and again showing the output in the form of a graph.

Then there will be a short discussion of the dragster's variables and how they influence the velocity and position of the dragster. There will also be a discussion of how to tune the results to a more realistic result.

Task 1: Numerical Integrations

Euler's Numerical Integration Method is a 1st order numerical approximation that is solely dependant on the size of the interval between each calculation. In the program that has been developed for a dragster simulation within this report using Euler's Explicit Method:

$$V_{n+1} = V_n + (A_n * \Delta t)$$

$$X_{n+1} = X_n + (V_n * \Delta t)[1]$$

Equation 1: Explicit Euler's Method

Where V is the velocity, X is the position and A is the acceleration. If there was a large timestep (Δt) then the results would be unreliable.

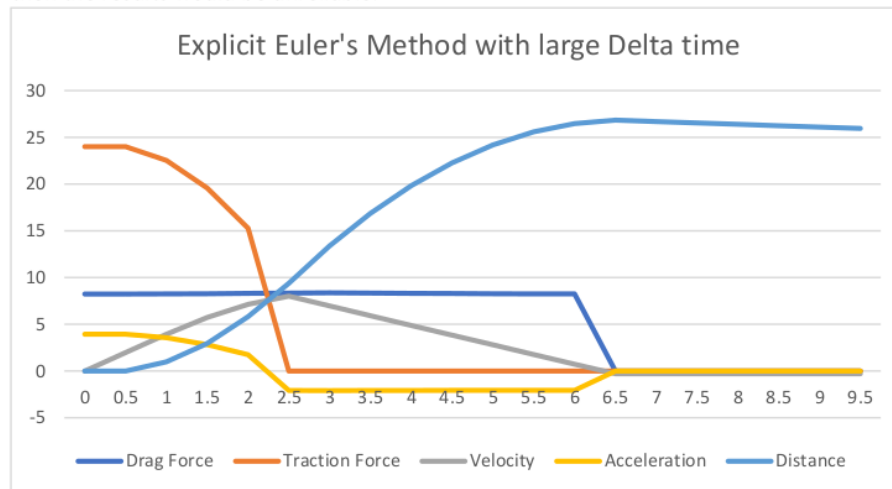


Figure 1: Explicit Euler's Method Dragster results with Large Delta Time

As shown above, the distance traveled and the velocity increase too fast and thus do not give an accurate representation of the dragster's movement. By comparison below, the Explicit Euler's Method with a smaller timestep shows that the distance and velocity is moving at a realistic rate.

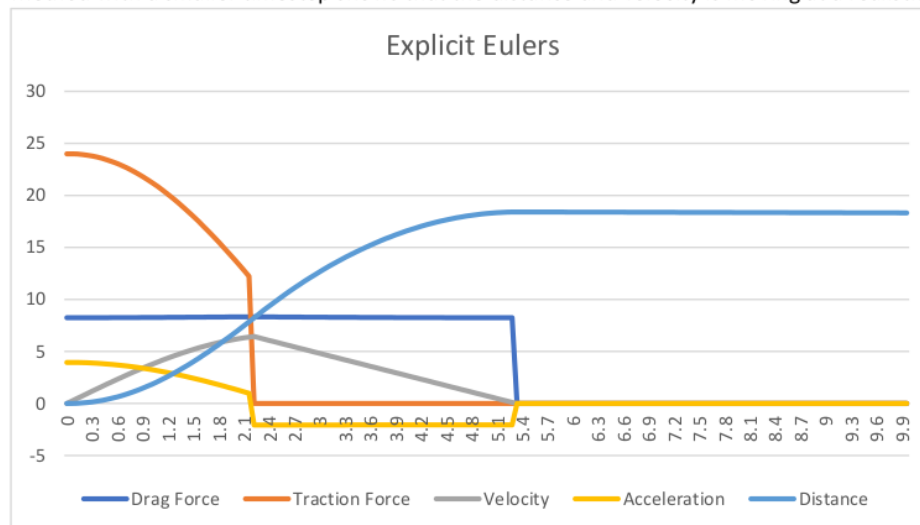


Figure 2 Explicit Euler's Method Dragster results With Smaller Delta Time

Contrary to this, there is a way to make it more accurate to realistic expectations and that would be higher-order numerical integrations, such as Velocity Verlet. This numerical integration method breaks down the algorithm into four parts and calculates the velocity twice.

Step 1)

$$Velocity_{n+1/2} = Velocity_n + Acceleration_n * \Delta/2$$

Step 2)

$$Position_{n+1} = Position_n + Velocity_{n+1/2} * \Delta$$

Step 3)

$$Acceleration_{n+1} = Force/Mass$$

Step 4)

$$Velocity_{n+1} = Velocity_{n+1/2} + Acceleration_{n+1} * \Delta/2[1]$$

Equation 2: Velocity Verlet Method

This gives a more accurate representation of the velocity and therefore a better result of the distance traveled:

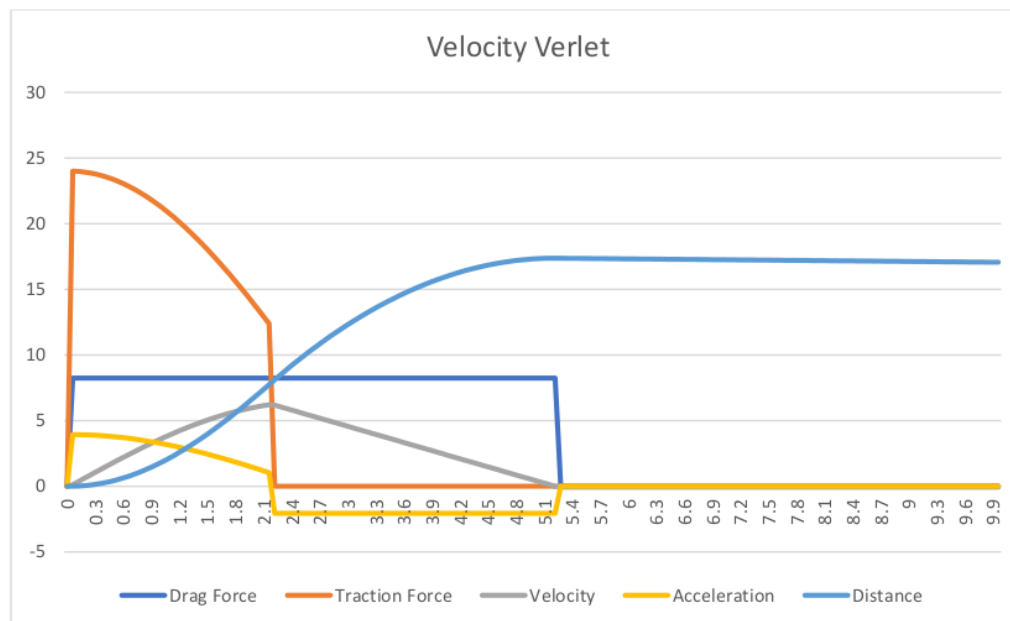


Figure 3: Velocity Verlet Dragster Results

Merits and Disadvantages

As previously mentioned; a disadvantage of the Euler's Numerical Integration Methods - Explicit, Symplectic or Implicit, is that they are dependant on the time step intervals for the accuracy. The other disadvantage to the Euler's methods is that they can have stability issues.[2][3][4][5]. These methods have their advantages; the numerical integration method is usually the first calculation in order to further the accuracy of more advanced methods, so it is the easiest to use.

Velocity Verlet is a second-order integration, implying that this could be used after the Euler's method to further approximate the results. This method though does lose precision the longer it runs.[6]

Improving the Simulation.

Regarding the dragster itself, many of the coefficients and forces were used to help get an accurate representation of the dragster in motion. But whilst viewing the simulation it has quite erratic results.

It is possible to tune this simulation using more calculations, the radii of the wheels and axle, spring tension and extension and inertia. Although it is important to note that inertia has not been calculated in this particular simulation, only implied.

With these calculations, the charts look more curvative and better represent the motion of speeding up and slowing down.

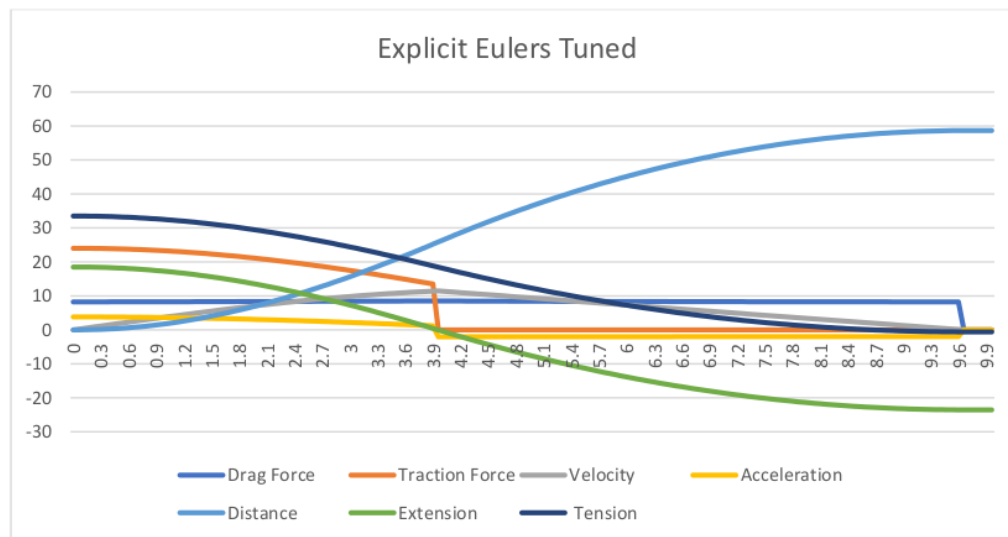


Figure 4:Explicit Euler's Method Dragster results with improved tuning

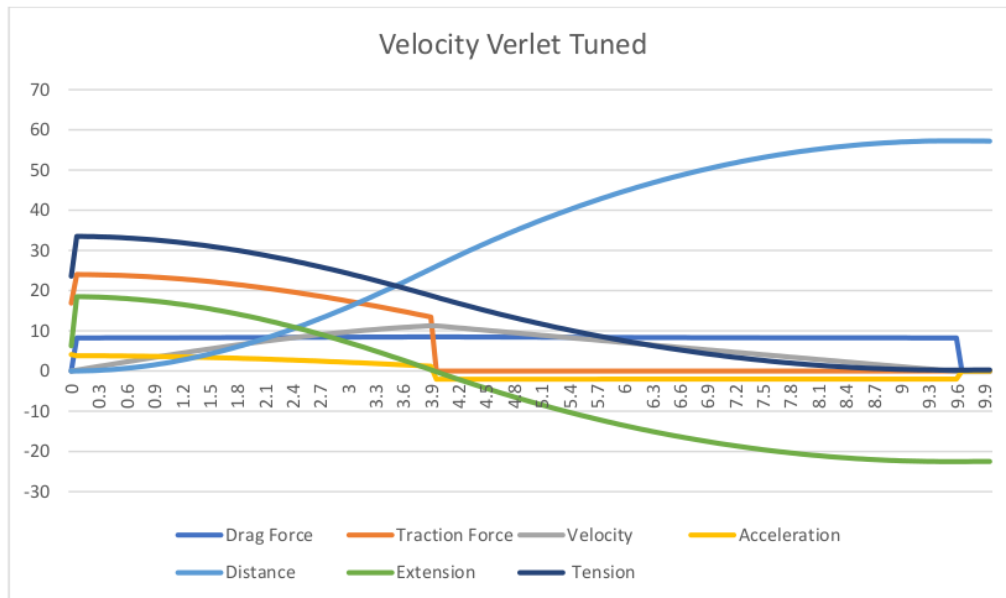


Figure 5: Velocity Verlet Method Dragster results with improved tuning

Task 1: Implementation

When the user runs the program it will run briefly then it will notify that the results have been placed in a file.

The location of the file is in the same folder as this report labeled "dragster.csv".

The simulation results will all be placed in this file.

Task 2: Vector Class

Vector Header File:

```
#ifndef VECTOR_H
#define VECTOR_H

class Vector

{
public:

    //construct
    Vector(float x, float y, float z);
    Vector(void);
    ~Vector(); //class did not have a destructor
    void Scale(float s);
    void Add(Vector & v1);

    // This calculates the distance between two points in 3d space using Pythagoras theory
    float DistanceBetweenPoints(Vector v3);

    void Add(Vector & v1, Vector & v2);
    Vector getVector(void);
    float getX(void);
    float getY(void);
    float getZ(void);
    void setVector(Vector a); //allows other classes to set vector values

private:

    float x;
    float y;
    float z;
    float diff;
};
#endif
```

Vector.cpp file:

```
#include "vector.h"
#include "stdafx.h"

//construct
Vector::Vector(void)
{
}

Vector::~Vector()
{
}

Vector::Vector(float myx, float myy, float myz)
{
    x = myx;
    y = myy;
    z = myz;
}

void Vector::Scale(float s)
{
}
```

```

        x = s * x;
        y = s * y;
        z = s * z;
    }

void Vector::Add(Vector & v1, Vector & v2)
{
    v1.x = v2.x + v1.x;
    v1.y = v2.y + v1.y;
    v1.z = v2.z + v1.z;
}

Vector Vector::getVector(void)
{
    return Vector(x, y, z);
}

float Vector::getX(void)
{
    return x;
}

float Vector::getY(void)
{
    return y;
}

float Vector::getZ(void)
{
    return z;
}

void Vector::setVector(Vector a) //allows other classes to set vector values
{
    a.x = x;
    a.y = y;
    a.z = z;
}

void Vector::Add(Vector & v1)
{
    x = x + v1.x;
    y = y + v1.y;
    z = z + v1.z;
}

void Vector::DistanceBetweenPoints(Vector v)
{
    float xDif = (x - v.getVector().x)*(x - v.getVector().x);
    float yDif = (y - v.getVector().y)*(y - v.getVector().y);
    float zDif = (z - v.getVector().z)*(z - v.getVector().z);
    diff = sqrt(xDif + yDif + zDif);

    cout << "The difference between" << x << ", " << y << ", " << z << " and " <<
v.x << ", " << v.y << ", " << v.z << " is : " << diff << endl;
}

```

In the .cpp file, the methods added to the class provided are:

```
void Vector::DistanceBetweenPoints(Vector v)
```

This method calculates the distance between two points in 3d using Pythagoras theorem and then outputting the results to console.

```
float getX(void);
float getY(void);
float getZ(void);
```

These methods allow other classes in the hierarchy to get the values of the vector that they have set.

```
void setVector(Vector a);
```

This method allows other classes in the hierarchy to set the values of the vector that they have inherited.

Task 2: Plane Class

Plane header file:

```
#pragma once
#include "Vector.h"

class Plane:public Vector
{
public:
    Plane(Vector *P, Vector *normal)
    {
        Vector point(2,4,5);
        P = new Vector(*P);
        normal = new Vector(*normal);
        P->setVector(*P);
        normal->setVector(*normal);
        //d = (ax0 + by0 + cz0)
        float d = normal->getX()*(P->getX() - point.getX())* normal->getY()*(P->getY() - point.getY())* normal->getZ()*(P->getZ() - point.getZ());
    };
    ~Plane();

    Vector PlaneToPlaneIntersect(Plane p, Plane p1);
    float DistanceOfPointOnPlane(Plane p, Vector a );

};
```

Plane.cpp file

```
#include "Plane.h"

Plane::~Plane()
{
}

Vector Plane::PlaneToPlaneIntersect(Plane p, Plane p1)
{
```

```
        return Vector();  
    }  
  
    float Plane::DistanceOfPointOnPlane(Plane p, Vector a)  
    {  
        return 0.0f;  
    }  
}
```

Plane(Vector *p, Vector *normal)

Constructs the class with the parameters and calculation using aggregation to ensure full construction of the plane per creation of the object [7]–[9]

Task3: Sphere To Sphere Collision

Sphere Header File:

```
#pragma once
#include "Vector.h"
class Sphere
{
public:
    Sphere(Vector *centre)
    {
        centre = new Vector(0, 0, 0);
        Vector position(1, 2, 3);

    };
    ~Sphere();
};
```

Sphere cpp file:

```
#include "Sphere.h"

Sphere::~Sphere()
{
}
```

References

- [1] J. Dickonson, "Numerical Integration in Games Development | Understanding Games Development." [Online]. Available: <https://jdickinsongames.wordpress.com/2015/01/22/numerical-integration-in-games-development-2/>. [Accessed: 03-Feb-2019].
- [2] B. Nikolic, "Computational Methods of Physics." [Online]. Available: http://www.physics.udel.edu/~bnikolic/teaching/phys660/numerical_ode/node5.html. [Accessed: 08-Feb-2019].
- [3] C. Barker, "Numerical Methods--Euler's Method." [Online]. Available: <http://calculuslab.deltacollege.edu/ODE/7-C-1/7-C-1-h-c.html>. [Accessed: 13-Feb-2019].
- [4] "Euler's Method , Taylor Series Method , Runge Kutta Methods , Multi-Step Methods and Stability .," no. 0, pp. 1–6.
- [5] L. Evans, "Limitations of euler's method for numerical integration," pp. 1–5, 2007.
- [6] "Idézet egy cikkből: The Verlet algorithm."
- [7] "Plane Equation." [Online]. Available: http://www.songho.ca/math/plane/plane.html#distance_origin. [Accessed: 21-Mar-2019].
- [8] S. Khan, "(26) Defining a plane in R3 with a point and normal vector (video) | Khan Academy." [Online]. Available: <https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/dot-cross-products/v/defining-a-plane-in-r3-with-a-point-and-normal-vector>. [Accessed: 18-Mar-2019].
- [9] P. Dawkins, "Calculus III - Equations of Planes." [Online]. Available: <http://tutorial.math.lamar.edu/Classes/CalcIII/EqnsOfPlanes.aspx>. [Accessed: 09-Feb-2019].

Appendix 1: Numerical Integration and Dragster Class

Dragster Header File:

```
#pragma once

class Dragster
{
public:
    Dragster();
    ~Dragster();

    void SetVehicleMass(float mass); // just to allow different size dragsters

protected:

    float CR = 0.21; // coefficient of rolling distance
    float m = 1.0; // mass of vehicle
    float g = 9.81; // gravity
    float cD = 0.34; // drag coefficient
    float rho = 1.2; // air resistance
    float A = 0.01; // frontal area of vehicle
    float v = 0; // velocity
    float ft = 0; // traction force
    float fd = 0; // drag force
    float x = 0; // distance travelled
    float Dt = 0.5; // Delta t (time-step)
    float acc = 0.00; // Acceleration initial

    // additional variables for fine-tuning the forces

    float RW; // radius of wheel
    float RA; // radius of wheel axle
    float d_init = 18.5; // force proportional to spring extension
    float D; // spring force
    float T; // spring tension
    float t_init = 18.5; // force proportional of spring tension

    float j = 5; // inertia of wheel (not calculated)

};
```

Dragster cpp File:

```
#include "Dragster.h"

Dragster::Dragster()
{
}

Dragster::~Dragster()
{
}

void Dragster::SetVehicleMass(float mass)
{
    m = mass;
}
```

Numerical Integrations Header File:

```
#pragma once
#include "Dragster.h"
#include "stdafx.h"

class Numerical_Integration: public Dragster
{
public:
    Numerical_Integration();
    ~Numerical_Integration();

    void ClearFile();// Clears file for new results
    void ShowFileName();
    void EulersExplicit();//uses Forces()
    void EulersExplicit_Tuned();//uses Forces_Modified()
    void Forces();//calculating the drag Coefficient and the traction forces
    void Forces_Modified();// Calculating forces() incorporating wheel and axle
    radius and calculating new coefficeints and traction force

    void VelocityVerlet();//uses Forces()
    void VelocityVerlet_Tuned();//uses Forces_Modified()

private:
    FILE *fp = NULL;
    float time = 0.00;
    float vn1;
    float xn1;

};
```

Numerical Integrations cpp File:

```
#include "Numerical_Integration.h"
#include "stdafx.h"

Numerical_Integration::Numerical_Integration()
{
}

Numerical_Integration::~Numerical_Integration()
{
}

void Numerical_Integration::ClearFile()
{
    fopen_s(&fp, "Dragster.csv", "w");
    if (fp == NULL)
    {
        printf("cant open file\n");
    }

    fclose(fp);
}

void Numerical_Integration::ShowFileName()
```

```

{
    cout << "Results in file: Dragster.csv" << endl;
}

void Numerical_Integration::EulersExplicit()
{
    fopen_s(&fp, "Dragster.csv", "a");

    if (fp == NULL)
    {
        printf("cant open file\n");
    }
    fprintf_s(fp, ",,,Eulers,Explicit\n\n");
    fprintf_s(fp, "Time Step,Drag Force,Traction
Force,Velocity,Acceleration,Distance,Stopped?\n");
    //cout << "Time Step " << "Drag Force" << " " << "Traction Force" << " "
    << "Velocity" << " " << "Acceleration" << " " << "Distance" << endl; //uncomment
    to print to console

    //fopen_s(&fp, "Dragster.csv", "a");    //just to test if append to the end of
    the file would work.
    //fprintf_s(fp, "9,7,8,6,5,6,6,7 \n");
    do
    {

        vn1 = v;
        v = v + (acc*Dt);//velocity equation
        x = x + (vn1*Dt);//distance equation

        if (time == 0)
        {
            x = 0;
            v = 0;
        }
        Forces();
        // uncomment to print to console
        //cout << setw(8)<<setprecision(3)<< time << setw(12) << setprecision(5)
        << fd << setw(16) << setprecision(5) << ft << setw(12) << setprecision(5) << v <<
        setw(18) << setprecision(5) << acc << setw(16) << setprecision(5) << x << endl;

        if (x > x + (v*Dt))// checks if the current distance is more than the
        next distance, to symbolize a stop.
        {
            fprintf_s(fp, "%.3f, %.3f, %.3f, %.3f, %.3f, %.3f, STOP\n", time,
            fd, ft, v, acc, x);
        }else
        fprintf_s(fp, "%.3f,%.3f,%.3f,%.3f,%.3f,%.3f\n", time,fd,ft,v,acc,x);

        time += Dt;
    } while (time < 10.00);

    fclose(fp);
    time = 0;
}

```

```

}

void Numerical_Integration::EulersExplicit_Tuned()
{
    fopen_s(&fp, "Dragster.csv", "a");

    if (fp == NULL)
    {
        printf("cant open file\n");
    }
    fprintf_s(fp, "\n\n,,Eulers,Explicit,Tuned\n\n");
    fprintf_s(fp, "Time Step,Drag Force,Traction
Force,Velocity,Acceleration,Distance,Stopped?,Extension, Tension \n");
    do
    {

        vn1 = v; //copy current v value;
        v = v + (acc*Dt); //velocity equation
        x = x + (vn1*Dt); //distance equation

        if (time == 0)
        {
            v = 0;
            x = 0;
            //ft = 0;

        }
        Forces_Modified();
        // uncomment to print to console
        //cout << setw(8)<<setprecision(3)<< time << setw(12) << setprecision(5)
        << fd << setw(16) << setprecision(5) << ft << setw(12) << setprecision(5) << v <<
        setw(18) << setprecision(5) << acc << setw(16) << setprecision(5) << x << endl;

        if (x > x + (v*Dt)) // checks if the current distance is more than the
        next distance, to symbolize a stop.
        {
            fprintf_s(fp, "%.3f, %.3f, %.3f, %.3f, %.3f,
            %.3f,STOP,%.3f,%.3f\n", time, fd, ft, v, acc, x, D, T);
        }
        else
            fprintf_s(fp, "%.3f, %.3f, %.3f, %.3f, %.3f, %.3f,%.3f,%.3f,\n",
            time, fd, ft, v, acc, x, D, T);

        time += Dt;
    } while (time < 10.00);
    fclose(fp);
    time = 0;
}

void Numerical_Integration::Forces()
{
    if (x > 8 )
    {
        ft = 0;
    }

```

```

    }
    else
    {
        ft = 1.5*(16 - x); //traction force equation
    }
    if (v < 0)
    {
        fd = 0;
    }
    else
    {
        fd = (CR * m*g) + (0.5*cD*rho*A*(vn1*vn1)); //drag force equation , F=ma
    }
    acc = (ft - fd) / m; //acceleration equation
}

void Numerical_Integration::Forces_Modified()
{
    RW = 3 * (0.5*m); //radius of wheel in proportion of the mass of the
vehicle
    RA = 2.15 * (0.5*m); // radius of wheel axle in proportion of the mass of the
vehicle

    D = d_init - x * (RA / RW);
    T = D * (t_init - 3.5) / d_init + d_init;

    if (D < 0)
    {
        ft = 0;
    }
    else
    {
        ft = T*(RA/RW); //traction force equation, using spring tension and the
wheel radius.
    }
    if (v < 0)
    {
        fd = 0;
    }
    else
    {
        fd = (CR * m*g) + (0.5*cD*rho*A*(v*v)); //drag force equation
    }
    acc = (ft - fd) / (m + j / (RW * RW)); //acceleration equation with inertia and
wheel radii
}

void Numerical_Integration::VelocityVerlet()
{
    fopen_s(&fp, "Dragster.csv", "a");

    if (fp == NULL)
    {
        printf("cant open file\n");
    }
}

```

```

    fprintf_s(fp, "\n\n,,Velocity,Verlet\n\n");
    fprintf_s(fp, "Time Step,Drag Force,Traction
Force,Velocity,Acceleration,Distance,Stopped?\n");

    //uncomment to print to console
    //cout << "Time Step " << "Drag Force" << " " << "Traction Force" << " "
<< "Velocity" << " " << "Acceleration" << " " << "Distance" << endl;

    do
    {
        float acc_c = acc; // store the current acceleration
        //float v_halfstep;
        Forces();

        //v_halfstep = v + acc_c * (Dt / 2); //velocity
        equation at half of the delta time
        x = x + v * Dt + (0.5* acc_c) * (Dt * Dt); //distance equation,
        velocity verlet solves for x first
        Forces();
        //to increase the acceleration variable

        v = v + 0.5*(acc + acc_c)*Dt; //velocity
        equation

        if (time == 0)
        {
            x = 0;
            v = 0;
        }

        // uncomment to print to console
        //cout << setw(8)<<setprecision(3)<< time << setw(12) << setprecision(5)
<< fd << setw(16) << setprecision(5) << ft << setw(12) << setprecision(5) << v <<
        setw(18) << setprecision(5) << acc << setw(16) << setprecision(5) << x << endl;

        if (x > x + v * Dt + (0.5* acc_c) * (Dt * Dt))// checks if the current
        distance is more than the next distance, to symbolize a stop.
        {
            fprintf_s(fp, "%.3f, %.3f, %.3f, %.3f, %.3f, %.3f, \n", time, fd,
            ft, v, acc, x);
        }
        else
            fprintf_s(fp, "%.3f,%.3f,%.3f,%.3f,%.3f,%.3f\n", time, fd, ft, v,
            acc, x);

        time += Dt;
    } while (time < 10.00);
    fclose(fp);
    time = 0;
}

void Numerical_Integration::VelocityVerlet_Tuned()
{
    fopen_s(&fp, "Dragster.csv", "a");

```

```

    if (fp == NULL)
    {
        printf("cant open file\n");
    }
    fprintf_s(fp, "\n\n,,Velocity,Verlet,Tuned\n\n");
    fprintf_s(fp, "Time Step,Drag Force,Traction
Force,Velocity,Acceleration,Distance,Stopped?,Extension, Tension \n");

    //uncomment to print to console
    //cout << "Time Step " << "Drag Force" << " " << "Traction Force" << " "
<< "Velocity" << " " << "Acceleration" << " " << "Distance" << endl;

    do
    {
        float acc_c = acc; // store the current acceleration
        //float v_halfstep;
        Forces_Modified();

        //velocity equation at half of the delta time, not used
        //v_halfstep = v + acc_c * (Dt / 2);

        //distance equation, velocity verlet solves for x first
        x = x + v * Dt + (0.5* acc_c) * (Dt * Dt);
        Forces_Modified();
        //to increase the acceleration variable

        v = v + 0.5*(acc + acc_c)*Dt; //velocity equation

        if (time == 0)
        {
            x = 0;
            v = 0;
        }

        // uncomment to print to console
        //cout << setw(8)<<setprecision(3)<< time << setw(12) << setprecision(5) << fd <<
        setw(16) << setprecision(5) << ft << setw(12) << setprecision(5) << v << setw(18) <<
        setprecision(5) << acc << setw(16) << setprecision(5) << x << endl;

        if (x > x + v * time + 0.5 + acc * (time * time))// checks if the current distance is
        more than the next distance, to symbolize a stop.
        {
            fprintf_s(fp, "%.3f, %.3f, %.3f, %.3f, %.3f, %.3f, STOP, %.3f,
            %.3f\n", time, fd, ft, v, acc, x, D, T);
        }
        else
            fprintf_s(fp, "%.3f, %.3f, %.3f, %.3f, %.3f, %.3f,%.3f,%.3f,\n",
            time, fd, ft, v, acc, x, D, T);

        time += Dt;
    } while (time < 10.00);
    fclose(fp);

```

```
    time = 0;
}
```

Appendix 2: Main.cpp

```
#include "stdafx.h"
#include "Dragster.h"
#include "Numerical_Integration.h"
#include "Vector.h"
#include "Plane.h"

int main(int argc, char* *argv[])
{
    Numerical_Integration ver1;
    ver1.ClearFile(); //clears csv file for a new program run.
    ver1.SetVehicleMass(4); //sets mass to reduce stopping time
    ver1.EulersExplicit();
    ver1.EulersExplicit_Tuned();
    ver1.VelocityVerlet();
    ver1.VelocityVerlet_Tuned();
    ver1.ShowFileName();

    Vector vec1(-1,2,1);
    Vector vec2(0, -3, 2);
    vec1.DistanceBetweenPoints(vec2);

    Plane planeA(&vec1, &vec2);

    return 0;
}
```

FINAL GRADE

GENERAL COMMENTS

Instructor

70/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

GRADING FORM: SMU GENERIC_GJ

RICHARD HANKEY**69.50****RESEARCH-INFORMED LITERATURE**

Refs used - try not to use too many on-line refs.
Essential topics identified and ref'd- additional ideas
investigated too.

6**KNOWLEDGE AND UNDERSTANDING**

Good understanding demonstrated here. Well done.
Some small confusion./ lack of clarity regarding plane
equation though. Good to see Verlet approach
included. Well done here.

15**ANALYSIS**

Relevant data gathered and graphs provided.
Perhaps the discussion could be more precise.

16**PRACTICAL APPLICATION AND DEP.**



Some good work here. Verlet attempted. Plane eqn class created. Sphere collision calc not addressed

29

SKILLS FOR PROFESSIONAL PRAC.



Report structured, refs included - avoid online. Could have used Bishop and van Verth.

3.5