

MaxJ Assignment1 ACCB5017

by Max Jones

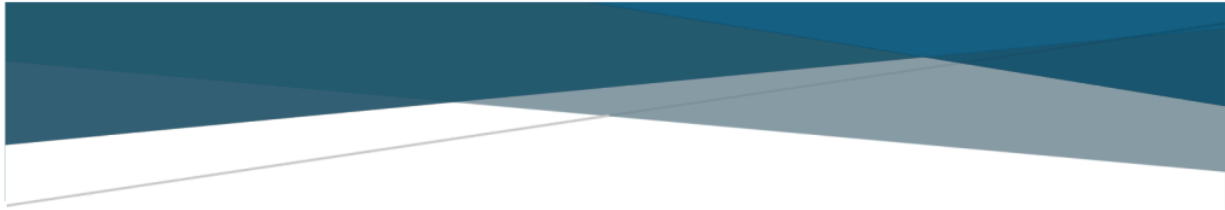
Submission date: 30-Nov-2024 08:46AM (UTC+0000)

Submission ID: 244183443

File name: 50579_Max_Jones_MaxJ_Assignment1_ACCB5017_628791_1018032080.docx (223.02K)

Word count: 1397

Character count: 7805



Games Architecture and Mechanics Assignment 1

ACCB5017

Max Jones (2305556)

Table of Contents

Games Architecture and Mechanics Assignment 1 0

Part 1: Describe two numerical integration methods. 2

 Euler's Method.....2

 Verlet integration.....2

Part 2 : Simulate the movement of a spring powered car..... 3

 Euler's method results 3

 Verlet Results.....5

Part 1: Describe two numerical integration methods.

Euler's Method

Euler's method is a form of numerical integration which was first described by Leonhard Euler. Euler's method is used in a variety of fields to calculate quick and simple simulations. This method of numerical integration works on a step-by-step basis updating both the velocity and position in each iteration. The basic general formula for Euler's method given by freeCodeCamp is.

$$y_{i+1} = y_i + h \int (x_i, y_i)$$

where-

y_{i+1} is the next step to be calculated,

y_i is the previously calculated step,

h is the time interval between the steps.[1]

This formula takes the current or initial value into account to calculate the next value. A common use of Euler's method in the game development industry is to simulate the velocity of an object. To apply the general equation to this we would use $v_{t+\Delta t} = v_t + (\frac{F}{m})\Delta t$ to find the velocity which is denoted by v . [2] As shown in the previous equation Euler's method only takes one input, in this case v , this makes it a first order integration method.

Euler's method is used for these simulations as it is one of the simpler integration methods to implement. Euler's method also has very low computational costs which makes it very effective and efficient for simulations that need to be run frequently [3].

Although Euler's method is easy to implement and does not use up much computational power it is never completely accurate. The error value produced is the difference between the actual solution and the estimate that this method gives us. This error margin can also get less accurate the longer the simulation is run, this is due to the fact that the previous result of the simulation is used in the calculation. This gives us a compounding error value [2]. Because of the fact that Euler's method is not completely accurate we need something to simulate objects and their physics more accurately. To do this we use higher order methods like Verlet integration.

Verlet integration

As previously stated, Verlet integration is a second order integration method. This means that this method uses both the current value and previous value into account to make the simulation more stable and have a lower error margin. Verlet has many use cases in game development to simulate different positions, such uses are cloth simulations, ragdoll physics and soft body dynamics [4]. Verlet allows for more accurate results than Euler's while still being fairly easy to implement. The formula to get the position of a object using Verlet is given below:

$$\text{Position}_{n+1} = 2 * \text{Position}_n - \text{Position}_{n-1} + \text{Acceleration} * dt^2 \text{ [5]}$$

Since Verlet doesn't explicitly calculate the velocity of the object during the integration it uses an estimate. This estimate of the velocity can be derived from the method using the equation below:

$$\text{Velocity}_{n+1} = (\text{Position}_n - \text{Position}_{n-1}) / dt [5]$$

This formula uses the current and previous position divided by the time step it gives the velocity of the object at that point in time. Since the Verlet method doesn't calculate the velocity only estimates it there has been another version of Verlet produced that does give the velocity. This method is called Velocity Verlet, it is more commonly used within game development due to the fact that velocity isn't estimated but calculated, this makes the Verlet even more accurate. Below is the Velocity Verlet equation.

Due to the fact that Verlet is to a higher order than Euler's this method will always be more accurate, however you can still get a more accurate simulation using an integration method with an even higher order such as the Runge-Kutta method.

Another downside to Verlet is that without deriving or using Velocity Verlet it only produces the position of the object being simulated. Which can come as a downside in some games where the speed of the car may be displayed to the player, or the velocity needs to be altered in some way.

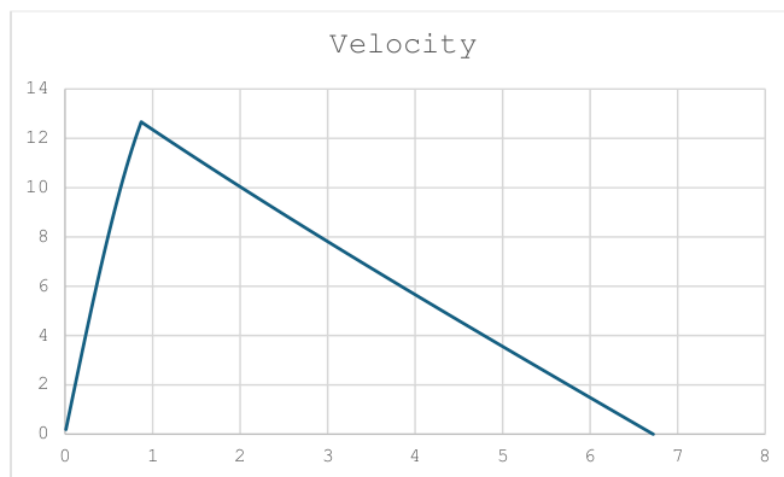
Velocity Verlet is a form of the Verlet method that explicitly calculates the velocity of the object not just estimates. You can produce results for both position and velocity as well as acceleration. The formula for these are below.

$$\begin{aligned} x(t+\Delta t) &= x(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \\ a(t+\Delta t) &= f(x(t+\Delta t)) \\ v(t+\Delta t) &= v(t) + \frac{1}{2}(a(t) + a(t+\Delta t))\Delta t \end{aligned} [6]$$

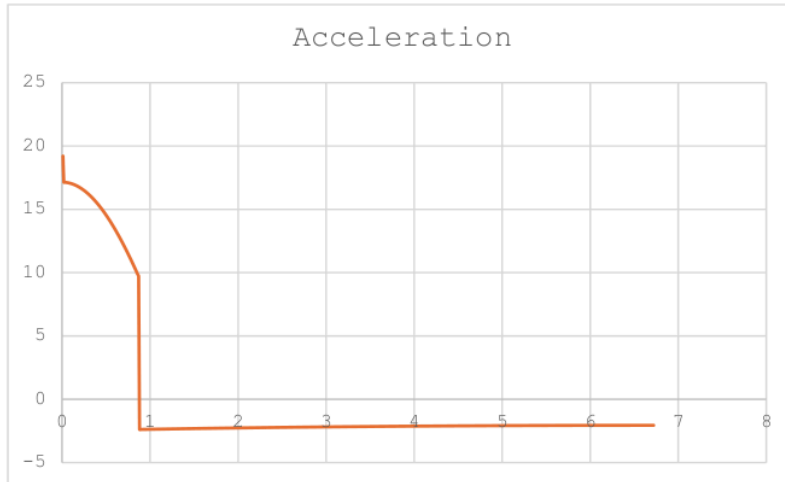
Part 2 : Simulate the movement of a spring powered car.

Euler's method results

Below are the results of my simulation using Euler's Method. All the graphs are plotted with Time in seconds on the X axis.



This graph depicts the acceleration of the spring powered car during this simulation.

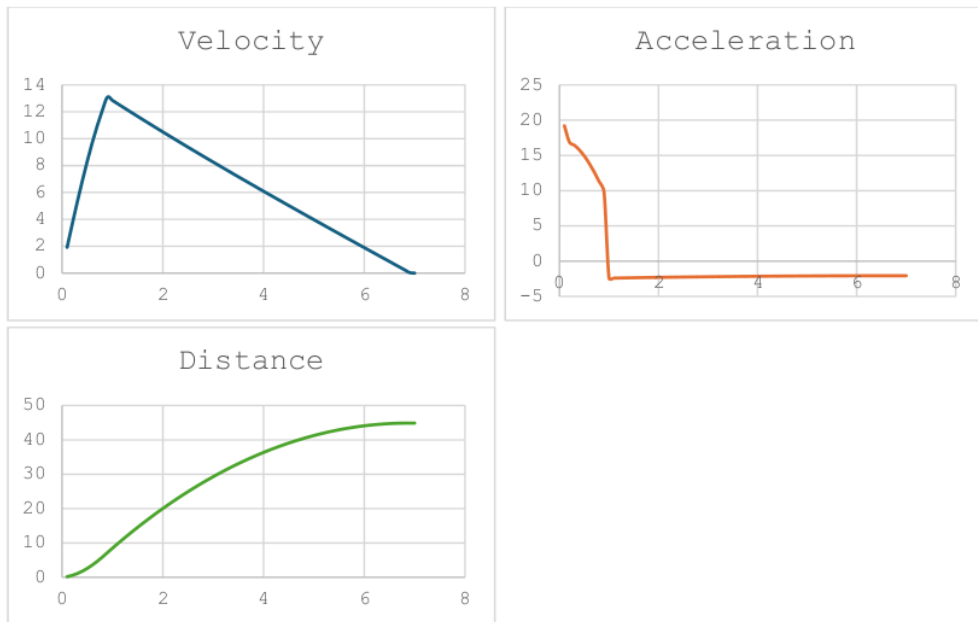


This graph depicts the acceleration of the spring powered car during the simulation.



This graph shows the distance the spring powered car travelled.

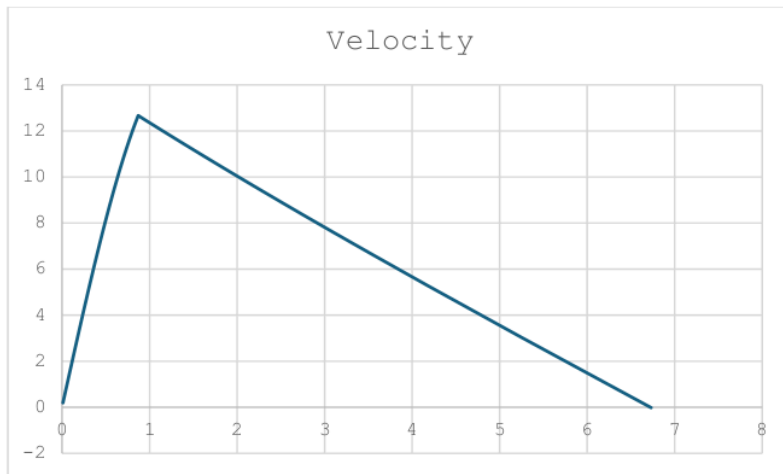
Since Euler's method becomes less accurate with bigger timesteps, if we change the timestep from 0.01s to 0.1s the simulation should theoretically get less accurate.

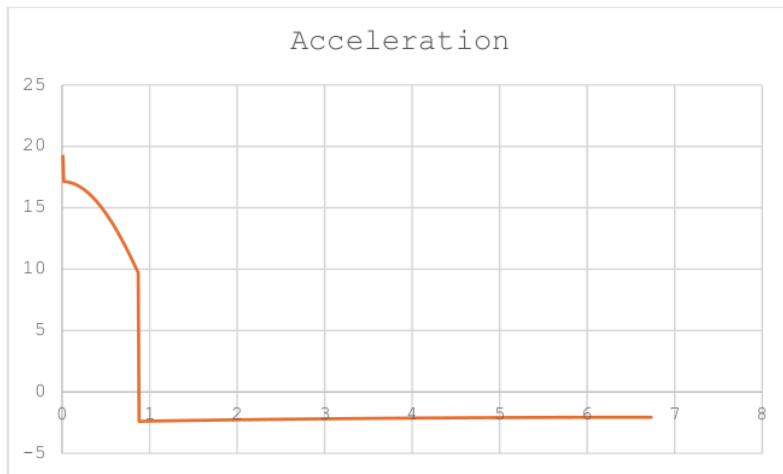


As you can see by these subsequent graphs the simulation still outputs similar results however it is to a less accurate degree which is how it should behave.

Verlet Results

Below are the graphs of the results of the Verlet method. All of the graphs are plotted against time in seconds on the X axis.





As you can see by the graphs, the two methods produced similar results. In fact, when looking at the raw data the time it took the cars to stop only varied by 0.01 second, with Verlet taking longer for the car to stop.

```
#include <iostream>
#include <fstream>

//Preset Values
float rollingResistanceCoefficient = 0.21f;
float dragCoefficient = 0.34f;

float vehicleMass = 1.0f;
float frontalVehicleArea = 0.01f;

float g = 9.81f;
float airDensity = 1.2f;
```

```

float deltaTime = 0.01f;

//Variables
float vehicleVelocity = 0.0f;
float distance = 0.0f;
float acceleration = 0.0f;
float dragForce = 0.0f;
float tractiveForce = 0.0f;
float overallTime = 0.0f;

//Variables used in Verlet
float dtSqr = deltaTime * deltaTime;
float currentPos = 0.0f;
float prevPos = 0.0f;

//Calculation Methods
float dragForceCalculation()
{
    //  $F_d = CR_{mg} + \frac{1}{2} CD \rho A V^2$ 

    if (vehicleVelocity == 0)
    {
        dragForce = 0;
    }
    else
    {
        float CRmgTemp = rollingResistanceCoefficient * vehicleMass * g;
        float temp = vehicleVelocity * vehicleVelocity;
        float dragTemp = dragCoefficient / 2;
        float CDpAVTemp = dragTemp * airDensity * frontalVehicleArea *
temp;
        dragForce = CRmgTemp + CDpAVTemp;
    }
    return dragForce;
}

float tractiveForceCalculation()
{
    //  $F_t = 1.2(16 - x)$ 

    float temp = 16 - distance;
    tractiveForce = 1.2 * temp;
    if (distance >= 6)
    {
        tractiveForce = 0;
    }
    return tractiveForce;
}

float accelerationCalculation()
{
    //  $a = (F_t - F_d) / m$ 
    float temp = tractiveForce - dragForce;
    acceleration = temp / vehicleMass;
    return acceleration;
}

float updateVelocityEulers()
{
    // velocity = velocity + (acceleration * time) unless 0

```

```

    vehicleVelocity += acceleration * deltaTime;
    if (vehicleVelocity <= 0 )
    {
        vehicleVelocity = 0;
    }
    return vehicleVelocity;
}

float updateDistanceEulers()
{
    float temp2 = vehicleVelocity * deltaTime;
    float newDistance = distance + temp2;
    distance = newDistance;
    if (distance <= 0)
    {
        distance = 0;
    }
    return distance;
}

int main()
{
    std::ofstream results;
    results.open("Eulers.csv");
    results << "Time,Velocity,Distance,Acceleration" << std::endl;

    while(vehicleVelocity > 0 || distance == 0)
    {
        //Calls for Calculations
        dragForce = dragForceCalculation();
        tractiveForce = tractiveForceCalculation();
        acceleration = accelerationCalculation();
        vehicleVelocity = updateVelocityEulers();
        distance = updateDistanceEulers();
        //Verlet Method
        /*
        float verletTemp = 2*currentPos - prevPos;
        float verletTemp2 = acceleration * dtSqr;
        distance = verletTemp + verletTemp2;
        prevPos = currentPos;
        currentPos = distance;
        float velTemp = currentPos - prevPos;
        vehicleVelocity = velTemp / deltaTime;
        */
        overallTime = overallTime + deltaTime;
        results << overallTime << "," << vehicleVelocity << "," << distance
        << "," << acceleration << std::endl;
        std::cout << "Overall time: " << overallTime << "s Distance: " <<
        distance << " Velocity: " << vehicleVelocity << "\n";
    }
}

```

See the working code in the video below.

<https://youtu.be/TcJd1ZiV5DU>

References

- [1] freeCodeCamp, "Euler's Method Explained with Examples," Jan, 26 2020. Available: <https://www.freecodecamp.org/news/eulers-method-explained-with-examples/>
- [2] D. M. Bourg, *Physics for game developers*, Beijing :: O'Reilly, 2002.
- [3] D. M. Bourg, *Physics for game developers*, Beijing :: O'Reilly, 2002.
- [4] "Verlet Integration and Cloth Physics Simulation." <https://pikuma.com/blog/verlet-integration-2d-cloth-physics-simulation> (accessed Nov 30).
- [5] "Numerical Integration in Games Development", *Understanding Games Development*, <https://jacksonsongames.wordpress.com/2015/01/22/numerical-integration-in-games-development-2/> (accessed Nov 30, 2024).
- [6] "Verlet Integration · Arcane Algorithm Archive," Accessed: Nov 30, 2024, [Online]. Available: https://www.algorithm-archive.org/contents/verlet_integration/verlet_integration.html

MaxJ Assignment1 ACCB5017

GRADEMARK REPORT

FINAL GRADE

82/100

GENERAL COMMENTS

Engagement with Literature: 8/15

Knowledge and Understanding: 12/15

Cognitive and Intellectual Skills: 18/20

Practical Skills: 40/45

Transferable Skills: 4/5

Overall 82/100

Comment:

Overall a really good job, with great attention to detail.

In the future be sure to label your screenshots as figures and provide a description/caption for each.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

