The Derivation of Initial Velocities and Predicted Displacement through the Application of Kinematic and Drag Formulae.

By: Justin Domagala-Tang, Austin Chow, Jared Fallowfield

The 5 Kinematic equations will be of massive importance when determining different factors within projectile motion, and are listed as such:

$$\vec{d} = \overrightarrow{v_a}t$$

$$\overrightarrow{v_f} = \overrightarrow{v_0} + \vec{a}t$$

$$\vec{d} = d_0 + \overrightarrow{v_0}t + \frac{1}{2}\vec{a}t^2$$

$$\vec{d} = d_0 + \overrightarrow{v_f}t - \frac{1}{2}\vec{a}t^2$$

$$v_f^2 = v_0^2 + 2\vec{a}\vec{d}$$

To determine the velocity prior to projectile motion, the airborne time must first be derived:

$$\vec{d} = d_0 + \overrightarrow{v_f}t - \frac{1}{2}\vec{a}t^2$$

Since v_f and d_0 equate to zero, their included terms can be eliminated:

$$:= \vec{d} = -\frac{1}{2}\vec{a}t^2$$

Rearrange to solve for time:

$$\frac{\vec{d}}{-\frac{1}{2}\vec{a}} = t^2 \rightarrow t = \sqrt{\frac{\vec{d}}{0.5\vec{a}}}$$

Lastly, decompose the equation to the y-axis, so that displacement is the height of release, and acceleration is equal to gravity.

$$t = \sqrt{\frac{\overrightarrow{d_y}}{0.5\vec{g}}}$$

Drag Force Formula:

$$F_{drag} = C \cdot p \cdot v^2 \cdot \frac{A}{2}$$

For simplicity, as values *C, p,* and *A* are all constants, let them equate to *n*.

$$F_{drag} = v^2 n$$

In order to properly utilize this equation in the following program, it needs to be set equivalent to acceleration(velocity).

$$ma = v^2 n \rightarrow a = \frac{v^2 n}{m}$$

$$\therefore \ a(v) \ = \frac{v^2 n}{m}$$

The acceleration value is later used in the kinematic equation:

$$\overrightarrow{v_f} \ = \ \overrightarrow{v_0} + \vec{a}t$$

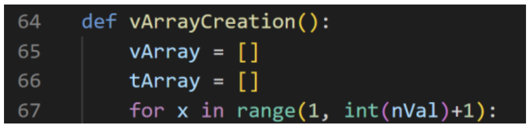
Where the program splits the path of the projectile into many indices, of which the above kinematic equation is employed in order to approach the level of accuracy of:

$$\lim_{t \to 0} \overrightarrow{v_f} = \overrightarrow{v_0} + \vec{a}t$$

Instead of employing the use of Calculus, this equation was utilized through the use of a Python program, the explanation of which will be best attempted below.

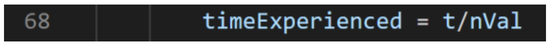
____

The first step within the program is to split the path produced by the object whilst in projectile motion into several equal indices dictated by the *nVal* variable.



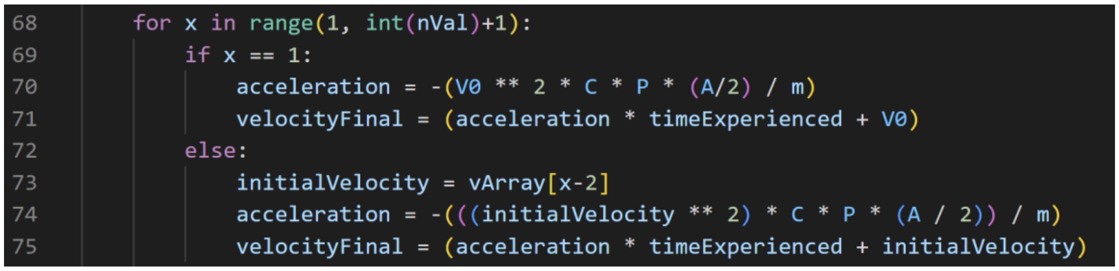
```
def vArrayCreation():
    vArray = []
    tArray = []
    for x in range(1, int(nVal)+1):
```

The program then creates the variable *timeExperienced*, this equation represents the infinitesimal time experienced in between the indices.



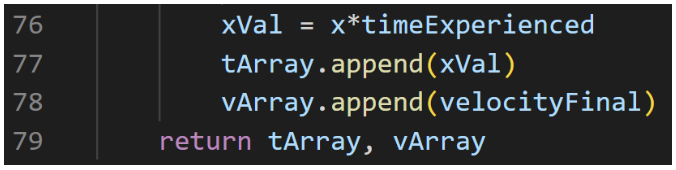
```
        timeExperienced = t/nVal
```

The application of the formulae is split into two different if statements. Since the program refers to the previous velocity index to calculate the next, the first index is sent into the first if statement to utilize the initial velocity rather than a list index, as none has been yet created.



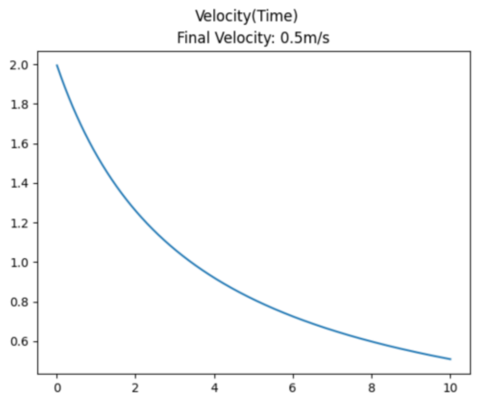
```
    for x in range(1, int(nVal)+1):
        if x == 1:
            acceleration = -(V0 ** 2 * C * P * (A/2) / m)
            velocityFinal = (acceleration * timeExperienced + V0)
        else:
            initialVelocity = vArray[x-2]
            acceleration = -(((initialVelocity ** 2) * C * P * (A / 2)) / m)
            velocityFinal = (acceleration * timeExperienced + initialVelocity)
```

Finally the program appends the time and velocity values of each index to be graphed using Matplotlib Pyplot in another function.

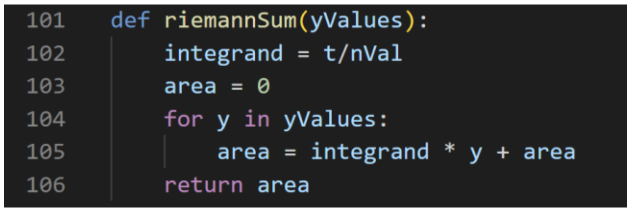


```
            xVal = x*timeExperienced
            tArray.append(xVal)
            vArray.append(velocityFinal)
        return tArray, vArray
```

When run, the programs returns the following graph displaying the Velocity(Time) graph for the given initial velocity.

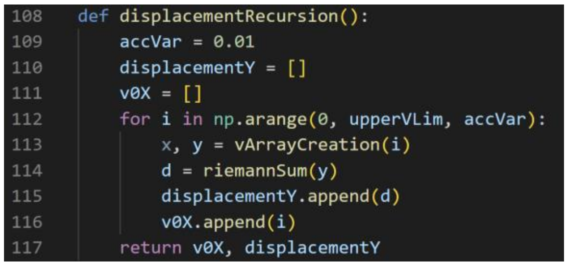


Since $\vec{d} = \vec{v}t$, by integrating this graph using the Riemann sum below the displacement can be derived from the initial velocity.



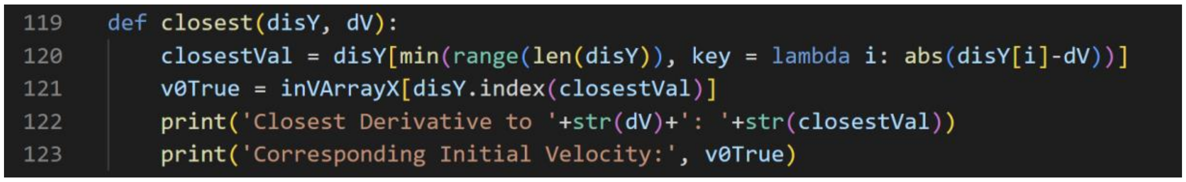
```
def riemannSum(yValues):
    integrand = t/nVal
    area = 0
    for y in yValues:
        area = integrand * y + area
    return area
```

Using this function, the program iterates from zero, to an upper limit of possible initial velocity at a defined accuracy, so that a graph can be formed using displacement(initial velocity).



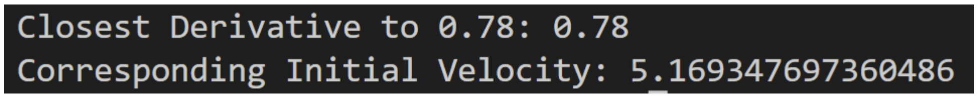
```python
def displacementRecursion():
    accVar = 0.01
    displacementY = []
    v0X = []
    for i in np.arange(0, upperVLim, accVar):
        x, y = vArrayCreation(i)
        d = riemannSum(y)
        displacementY.append(d)
        v0X.append(i)
    return v0X, displacementY
```

The program then takes the displacement measured from the experiment, and takes the inverse derivative of the displacement(initial velocity) graph. This is done by using a function which relates the given derivative to the closest [x,y] tuple graphed.



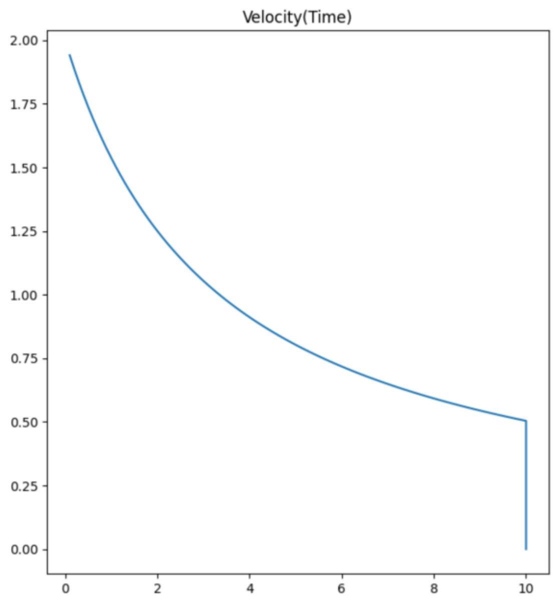
```python
def closest(disY, dV):
    closestVal = disY[min(range(len(disY)), key = lambda i: abs(disY[i]-dV))]
    v0True = inVArrayX[disY.index(closestVal)]
    print('Closest Derivative to '+str(dV)+': '+str(closestVal))
    print('Corresponding Initial Velocity:', v0True)
```

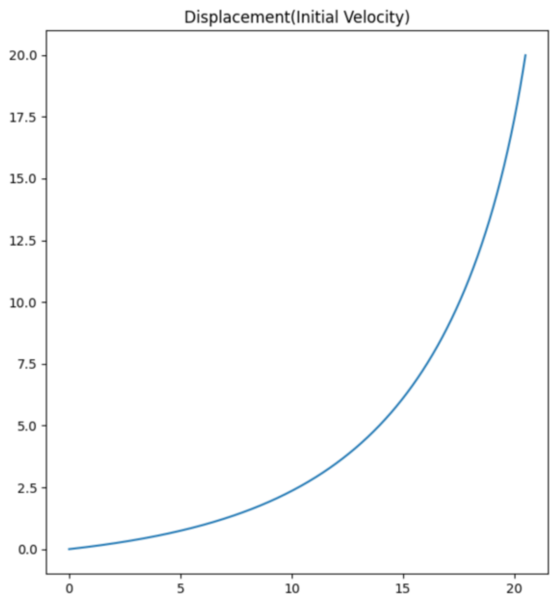
The program then finally returns the initial velocity related to the measured displacement value, along with the true derivative pulled from.



```
Closest Derivative to 0.78: 0.78
Corresponding Initial Velocity: 5.169347697360486
```

Through the employment of this Python program, and the math done beforehand it is possible to derive the initial velocity of a projectile using its displacement and known values with the relative avoidance of complex calculus.


Velocity(Time)


Displacement(Initial Velocity)