Ethan Fahnestock

# Marshmallow Madness Report

## Introduction

This experiment was performed to better understand the interactions between springs, friction, and marshmallows, while using this understanding to shoot marshmallows into cardboard boxes quickly and accurately. To reach this goal everything that interacted with the marshmallow during its journey from launch to landing had to be understood, which is what was attempted in this lab, with varying levels of success. The marshmallow begins its journey with springs (seen top-center in Figure 1.1), which is where it gets its kinetic energy. The amount of kinetic energy the marshmallow received is relative to Hooke's Law (Elasticity, n.d). Now airborne, the marshmallow's trajectory could be understood and predicted using Newton's Laws of Motion (Description of Motion in One Dimension, n.d.). Unfortunately this mortar exists in the real world, meaning everything is under friction's rein. Friction occurring in the barrel of the mortar could be accounted for by looking at the actual and expected muzzle velocities of the marshmallow. Friction in the air, or drag, on the other hand, is more complicated. The drag equation, which takes into account the fluid density, cross-sectional area of the projectile, and the velocity of the object, allows the calculation of the opposing frictional force on any object passing through a fluid. (The Drag Equation, n.d.) This equation was applied with limited success in the lab, as it requires a drag constant, which is found through experimentation usually performed in a wind tunnel. The mortar was built so that the effect of adjusting the angle of the barrel, as well as the length of pullback, on the projectile length could be tested.

Figure 1.1: Assembled Mortar

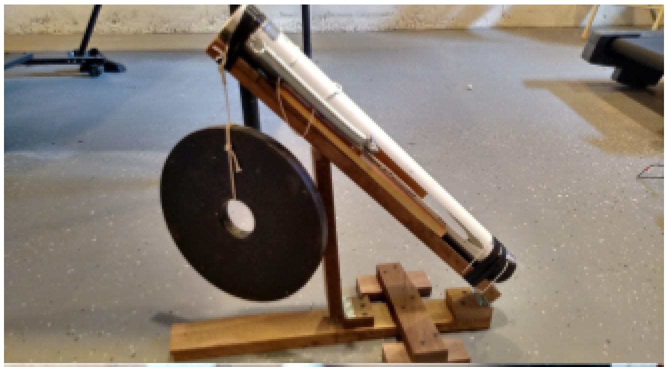

Figure 1.2: Finding the Spring Constant of Larger Springs

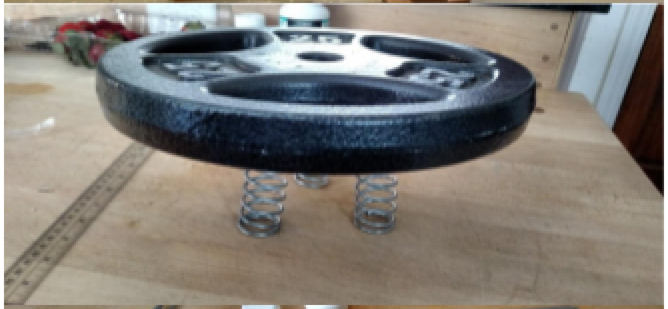

Figure 1.3: Finding the Average Marshmallow Mass

**Methods**

**Prototyping -** Initially a mortar was build using the compression springs in Figure 1.2 stacked on top of each other in the same PVC tube seen in Figure 1.1. This mortar evolved into the one seen in Figure 1.1 because the compression springs were far too powerful for the project. Smaller expansion springs with a spring constant 1/30th of the original springs were a better fit.

**Processing Data -** A program was written in Python (www.python.org) to run calculations as well as process and visualize data throughout this lab (All of the code written is well commented and understandable, and can be found using the link in the footnote.)

## Results & Calculations

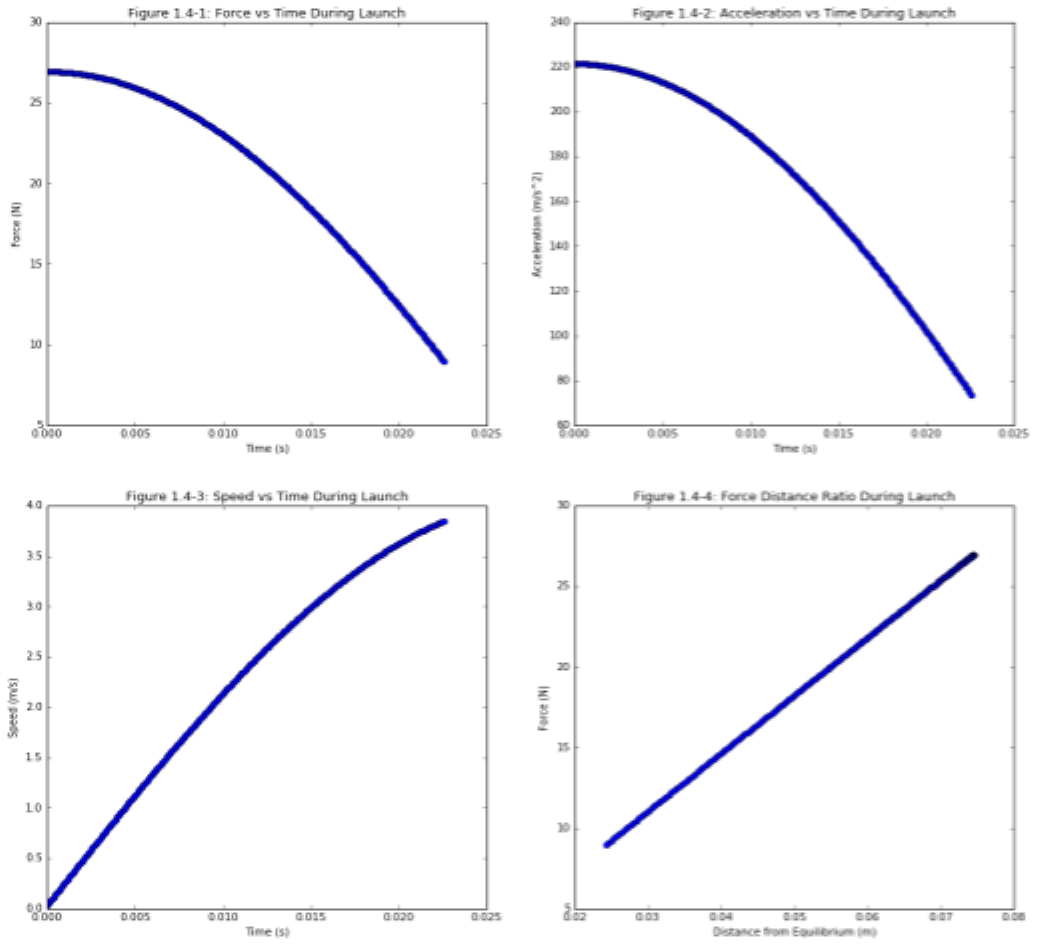

Figure 1.4: Predicted Launch Behaviour

**Table 2.1: Collected Data**

| **Trial** | **Angle (deg)** | **0.025 m** | **0.05 m** | **0.075 m** | **0.1 m** | **0.125 m** | **0.15 m** | **0.2 m** |
|---|---|---|---|---|---|---|---|---|
| **Trial 1.1** | 45 | 0.71 | 1.13 | 1.80 | 2.95 | 3.30 | N/A | N/A |
| **Trail 1.2** | 45 | 0.71 | 1.17 | 1.90 | 2.75 | 3.40 | N/A | N/A |
| **Trial 1.3** | 45 | 0.65 | 1.16 | 1.90 | 2.85 | 3.20 | N/A | N/A |
| **Average** | 45 | 0.69 | 1.15 | 1.86 | 2.85 | 3.30 | N/A | N/A |
| **Trial 2.1** | 60 | 0.43 | 0.90 | 1.40 | 2.13 | 2.77 | 2.8 | N/A |
| **Trial 2.2** | 60 | 0.40 | 0.99 | 1.60 | 2.26 | 3.10 | 3.33 | N/A |
| **Trial 2.3** | 60 | 0.38 | 1.00 | 1.40 | 2.34 | 3.00 | N/A | N/A |
| **Average** | 60 | 0.40 | 0.96 | 1.46 | 2.24 | 2.96 | 3.07 | N/A |

**Table 2.2 Average Predicted Distance with Percent Error for Distance Trials**

| Pullback (m) | 45º Average Distance (m) | 45º Predicted Distance (m) | 45º Percent Error (%) | 60º Average Distance (m) | 60º Predicted Distance (m) | 60º Percent Error (%) |
|---|---|---|---|---|---|---|
| **0.025** | 0.69 | 0.84 | 17.0 | 0.40 | 0.68 | 41.2 |
| **0.05** | 1.15 | 1.84 | 37.5 | 0.96 | 1.54 | 37.7 |
| **0.075** | 1.86 | 3.20 | 41.8 | 1.46 | 2.71 | 46.1 |
| **0.1** | 2.85 | 4.90 | 41.9 | 2.24 | 4.19 | 46.5 |
| **0.125** | 3.3 | 7.00 | 52.8 | 2.96 | 5.99 | 50.5 |
| **0.15** | N/A | N/A | N/A | 3.07 | 8.14 | 62.2 |

**Table 3.1: Comparison of Predicted Flight Times vs Times Pulled from Video.**

| Pullback | Frames (# of frames) | Flight Time (s) | Expected Time (s) | Percent Error (%) |
|---|---|---|---|---|
| **0.05** | 18 | 0.6157 | 0.6138 | 0.31 |
| **0.075** | 23 | 0.7867 | 0.8052 | 2.31 |
| **0.1** | 29 | 0.9919 | 0.9978 | 0.79 |

**Calculated Frames Per Second:** 29.24 fps

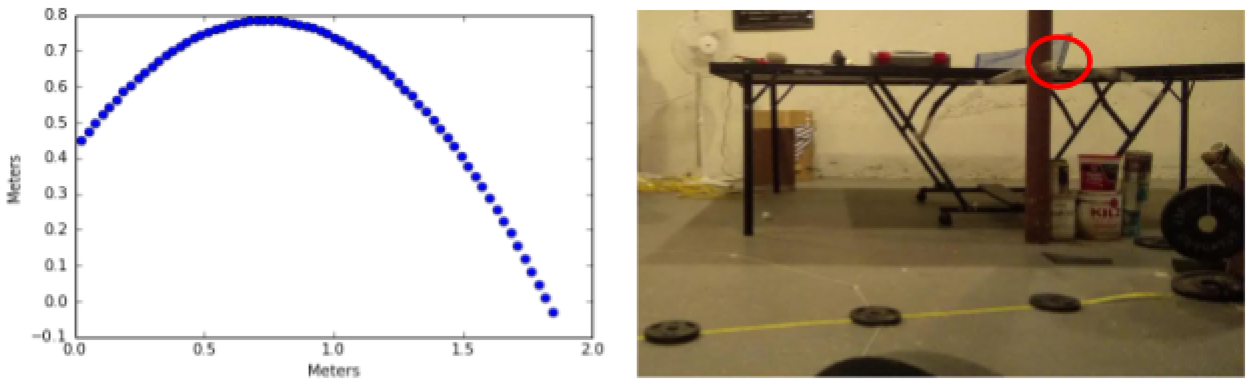

Figure 3.1: Comparison Between Predicted Projectile Motion and Actual Projectile Motion of a 0.05 m Pullback Launch

Figure 1.4 shows the calculated forces, accelerations, velocities, and displacements during the launching of a marshmallow that was pulled back 0.05 meters. All of this data was calculated, not observed, explaining the perfect curves/lines. Both Figure 1.4-1 and 1.4-2 drop towards the end while 1.4-3 curves upward. Figure 1.4-4 is the only linear graph

Table 2.1 shows the collected data from the launch trials. Any trail that would either launch the marshmallow too high or too far is marked as N/A. Trials occurred at different pullbacks and angles, with three trials of each configuration. The averages are also on the table. The 45 degree launches go farther than their 60 degree counterparts. Table 2.2 compares the average distance traveled by each scenario with the expected distance. The percent errors are very large, with an error of 41% appearing several times.

Table 3.1 shows the calculated and measured flight times for some of the 45º test scenarios. Other tests are not listed because the marshmallow traveled out of the frame during the recording, making them useless. Figure 3.1 compares the graphed predicted trajectory to an edited multi-exposure image stitched together from frames of the video. If looked at closely, the parabolic path of the marshmallow can be seen as a faint curved blur. The red circle points out the highest point in the arc. The weights on the floor are spaced every 0.5 meters and are parallel with the trajectory of the marshmallow. The launcher can be seen on the right side of the image.

**Calculations**

**Horizontal Distance Calculation:**

$$Dx = v_o cos\alpha \times (\frac{v_o}{g} + \sqrt{\frac{2(\Delta h + \frac{(v_o sin\alpha)^2}{2g})}{g}})$$

$Dx$ = horizontal distance
$v_o$ = muzzle velocity
$\alpha$ = angle of barrel
$\Delta$ h = change in height between top of box and launch height

**“Near-Instantaneous” Spring Calculation:**

$$d = S_{t-1} \times \beta + \frac{\left(\left(\frac{2k(do-d_{cont})}{mp+mm}\right)\times\beta^2\right)}{2}$$

d = distance the marshmallow has moved in the barrel (always less than $d_o$)
$d_o$ = pullback distance
$d_{cont}$ = cumulative distance traveled ($d_1$+$d_2$+$d_3$ etc)
$S_{t-1}$ = velocity prior to loop (changes each time due to acceleration from springs)
$\beta$ = time intervals ($1\mathrm{x}10^{-4}$ seconds for this lab - the smaller the more accurate)
k = spring constant
$m_p$ = mass of the springs/bolts/wooden plunger inside the barrel
$m_m$ = average marshmallow mass (from a sample of 20 marshmallows)
This equation is difficult to translate from the processing program it was written for, as many other things are going on. Check the code in the footnote for a better understanding.

**Discussion**

Every graph in Figure 1.4 presents a snapshot of the processing program during each iteration of its’ loop. The processing program is given several starting parameters - how far are the springs stretched, what the mass of the “plunger” inside the barrel is, what the spring constants are, etc. Using this information, it figures out what the system will be like after $10^{-4}$ seconds. During that time the system is accelerated by the springs, moving towards equilibrium. After the program updates everything from the distance stretched to the speed of the “plunger”, it runs through the loop again. (More detail under Calculations - “Near Instantaneous” Spring Calculation and in the program itself, link below) The program does this over and over again until the plunger, springs, and marshmallow travel to the end of the track, where the marshmallow is launched. Figure 1.4 shows the progression of the force, acceleration, and speed, throughout this process. This near instantaneous method of processing allows the calculations of variables that adapt to change without the use of calculus. The first two graphs show that over the interval of launch, the force and acceleration of the launcher decreases as the springs get closer to equilibrium. Figure 1.4-3 shows the effects of this depletion force/acceleration, because the system gains speed more rapidly at the beginning. The slope of Figure 1.4-4 could be used to calculate the spring constant, as the slope would be F/d, where F=kd. The “ds” cancel leaving the spring constant.

The effects of the changed variables on the range of the marshmallow can be seen in Tables 2.1 and 2.2. Every 60º trial landed short of the 45º trial with a corresponding pullback. This is because the distance depends on the sine and cosine of the angle as shown in the Horizontal Distance Calculation. It so happens that the maximum angle is 45º, where the sine and cosine are equal. These trig values could be bigger, but if you make one bigger, the other decreases and ends up costing more distance than gained. Both 2.x tables also make it obvious that the farther back the “plunger” in the barrel is pulled, the farther the marshmallow goes. As the springs are stretched back farther, they apply a larger force and therefore acceleration, so the marshmallow has a larger muzzle velocity.

The percent errors shown in Table 2.2 are very large, and strangely consistent. Seven out of the twelve percent errors are close to 43% This suggests that there was some kind of consistent error, something not accounted for in calculations. The program calculating the distance takes into account the starting height, the uneven stretching of the springs on the sides of the launcher, as well as the bit of stretch left in the springs while the mortar is at rest. To gain further insight into what might be causing the error, videos were taken of some of the trials. These videos were picked apart frame by frame using ffmpeg (FFMPEG, n.d.), then just the sequences containing the launch were filtered out. Using the number of frames and the length of the video, a frame rate was calculated (value in results). By counting the number of frames in which a marshmallow is in the air and dividing it by a frame rate (frames/second), the airtime of the marshmallows could be determined within 1/30th of a second. These are the times that are shown in Table 3.1 next to the estimated times (from dividing the horizontal distance calculation by the horizontal velocity). The miniscule percent error suggests that the time is calculated correctly, and if the time is calculated correctly then there isn't a problem with the horizontal velocity calculation because the two share the same angle (sine vs cosine) and the same initial velocity. With the frames from the video stitched together the marshmallow's path becomes visible, and can be compared to the predicted path, which is on the left. The highest point in the marshmallow's path is closer to the launch in the actual trial, occurring at about 0.5 meters, whereas in the predicted path the marshmallow peaks at about 0.75 meters. This is to be expected because all of the experimental values fall short of the calculated values, but why?

It is likely due to a combination of errors, and more experimentation is likely required. Isolating the horizontal and vertical, it can be deducted that the marshmallow must reach the predicted height in order for it to have the right air time, but because it is falling short the horizontal velocity must not be as high as expected. A possible explanation for this would be if the springs weren't as powerful as they were thought to be, and the angle of the barrel was higher than measured. This would result in a slower muzzle velocity, but the higher angle would account for the air time, giving us results where the flight time is accurate but the horizontal distance was off.

This data could also be a result of drag on the marshmallow or friction while launching. Normally at speeds this low, drag wouldn't have much effect, but the mass of the marshmallow is so low it may have a large effect. In an attempt to understand if drag would have any effect, the drag equation (The Drag Equation, n.d.) was used to calculate the frictional force the air would apply to the marshmellow. As mentioned in the introduction this formula requires a drag coefficient, which is discovered through testing. Without the ability to test for one, an average drag coefficient for a cylinder was used in its place, but the formula returned that friction would be decelerating the marshmallow at 3m/s, which didn't seem plausible. Both drag and friction in the tube would affect the vertical and horizontal velocities, so unless coupled with some other sort of error, these wouldn't cause the trends found in the collected data. Instead of making assumptions more testing should be done to better understand this situation. Another video could be taken of the launching of a marshmallow, but with a meterstick along it's path so the muzzle velocity could be calculated and checked for sources of error.

*Footnote:*

*Code and image launch sequences can be found at:*
*https://github.com/aknh9189/code/blob/master/physicsScripts/catapault/launcherPhysics.py*
*And https://github.com/aknh9189/code/tree/master/physicsScripts/catapault*

## References

Air Friction. (n.d.). Retrieved March 24, 2016, from http://hyperphysics.phy-astr.gsu.edu/hbase/airfri.html

Description of Motion in One Dimension. (n.d.). Retrieved March 24, 2016, from http://hyperphysics.phy-astr.gsu.edu/hbase/mot.html#motcon

The Drag Equation. (n.d.). Retrieved March 24, 2016, from https://www.grc.nasa.gov/www/k-12/airplane/drageq.html

Elasticity. (n.d.). Retrieved March 24, 2016, from http://hyperphysics.phy-astr.gsu.edu/hbase/permot2.html#c3

FFmpeg – the swiss army knife of Internet Streaming – part II. (2011, August 07). Retrieved March 24, 2016, from https://sonnati.wordpress.com/2011/08/08/ffmpeg-—-the-swiss-army-knife-of-internet-streaming-—-part-ii/