MXB103 Project Team 65: Bungee!

Daniel Edwards (n5538815), Mingjun Liu (n10122885), Christabel Vanessa (n10595112), Louis Yanagisawa (n10221221)

1 Introduction

Bungee jumping is an activity where a jumper has to jump off a height with their feet being tied to an elastic rubber cord. In a report written by Li and Lu (2013), they stated that the first modern bungee jump was done by the Oxford University Sports Club on April Fool's Day of 1979 when they jumped from the Clifton Suspension Bridge in Bristol, England. Bungee jumping is still a popular sport up to this day, but this sport can also result in accidents. One of the accidents that can happen is when the jumper is detached from the rope, this led to the use of body harness by commercial operators (Wikipedia, n.d.). Since the Brisbane City Council is proposing to allow bungee jumping off Story Bridge, mathematical models can be used to investigate some aspects of the proposal to assure its safety.

This report answers a few key aspects of allowing bungee jumping off the Story Bridge. The proposal calls for a platform to be installed at the very top of the bridge, from which the bungee jumps will take place. The platform will be at height H from the water level and y would be the distance the jumper has fallen. Other aspects which were proposed includes an automated camera system which will be installed at the bridge deck D and also a "water touch" where the jumper touches the water at the first bounce. In this report we will answer questions regarding the number of bounces in 60 seconds, the maximum velocity and acceleration which will be experienced by the jumper, the distance travelled by the jumper, at what time should an automated camera system capture an image of the jumper, and how to implement the "water touch" option.

Our approach to answering all the questions for thrill-seekers is to analyze all the factors in play like Gravity, Drag, and Tension during the Jump. In Section 2 of this report, a mathematical model is described along with what needs to be considered and how. In Section 3 of this report, a numerical solution for the given problems is formulated. In Section 4 of this report, the actual analysis is done using MATLAB. In section 5 of this report, a conclusion of our findings and suggestions are provided.

2 Mathematical Model

The motion that goes on in bungee jumping is one-dimensional where the rubber cord is considered as a massless elastic, the jumper is the point mass, and the stress vs strain curve is assumed to be linear (Heck et al., 2010). This motion can be divided into three forces, which are 1) Gravity – the free fall of the jumper (mg), 2) Drag/Air Resistance – the stretching of the rubber cord ($-cv^2$), and 3) Tension – the recoiling of the rubber cord ($-k(y-L)$), this is the stage where Hooke's Law is applied and it only occurs when the displacement (y) is larger than the length of the bungee cord (L). According to Newton's Second Law, $F = ma$, the sum of all the forces is equal to the jumper's mass multiplied by their acceleration. The acceleration a is equal to the derivation of velocity over time, $a = \frac{dv}{dt}$. We can replace $F = ma$ with the sum of the forces for F and $\frac{dv}{dt}$ for a to get:

$$m\frac{dv}{dt} = mg - cv^2 - k(y - L)$$

$$\frac{dv}{dt} = g - \frac{c}{m}v^2 - \frac{k}{m}(y - L)$$

m = mass of the jumper (kg)
$a = \frac{dv}{dt}$ = acceleration (m/s^2)
g = gravitational acceleration (m/s^2)
c = drag coefficient (kg/m)
v = velocity (m/s)
k = spring constant of cord (N/m)
y = position/displacement (m)
L = length of bungee cord (m)

One of the limitations this model has is that there might be inaccuracies due to the roundoff errors of the numbers that were used. Another limitation is that this model ignores the different temperatures which can cause a change in air density which also causes a change in the air resistance/drag (Lincoln, 2020).

3 Numerical Solution

```
function [t, v, y, h] = bungee_project_RK4(a, b, n)
% bungee_project_RK4 Bungee Project Model Using Fourth Order Runge-Kutta
% Method
% [t, v, y, h] = bungee_project_RK4(f, a, b, alpha, n) computes the bungee
% jumping using RK4 taking n steps from t = a to t = b with v(a) = alpha.
% let y(1) = 0 and v(1) = 0 which are the distance and velocity of the
% jumper at the platform (y0 & v0).

% declare the constants
c = 0.9; % drag coefficient in kg/m
m = 80; % mass of jumper in kg
k = 90; % spring constant of bungee rope in N/m
L = 25; % length of bungee rope in m
g = 9.8; % gravitational acceleration in m/s^2
C = c/m;
K = k/m;
f1 = @(t,v) v;
f2 = @(t,y,v) g - C*abs(v)*v - max(0, K*(y - L));

% calculate h
h = (b-a)/n;

% create t array
t = a:h:b;
% initialise v & y array
v = zeros(size(t));
v(1) = 0;
y = zeros(size(t));
y(1) = 0;

% perform the iterations
for i = 1:n
   k1_v = h * f2(t(i), y(i), v(i));
   k1_y = h * f1(t(i), v(i));

   k2_v = h * f2(t(i) + (h/2), y(i) + (k1_y/2), v(i) + (k1_v/2));
   k2_y = h * f1(t(i) + (h/2), v(i) + (k1_v/2));

   k3_v = h * f2(t(i) + (h/2), y(i) + (k2_y/2), v(i) + (k2_v/2));
```

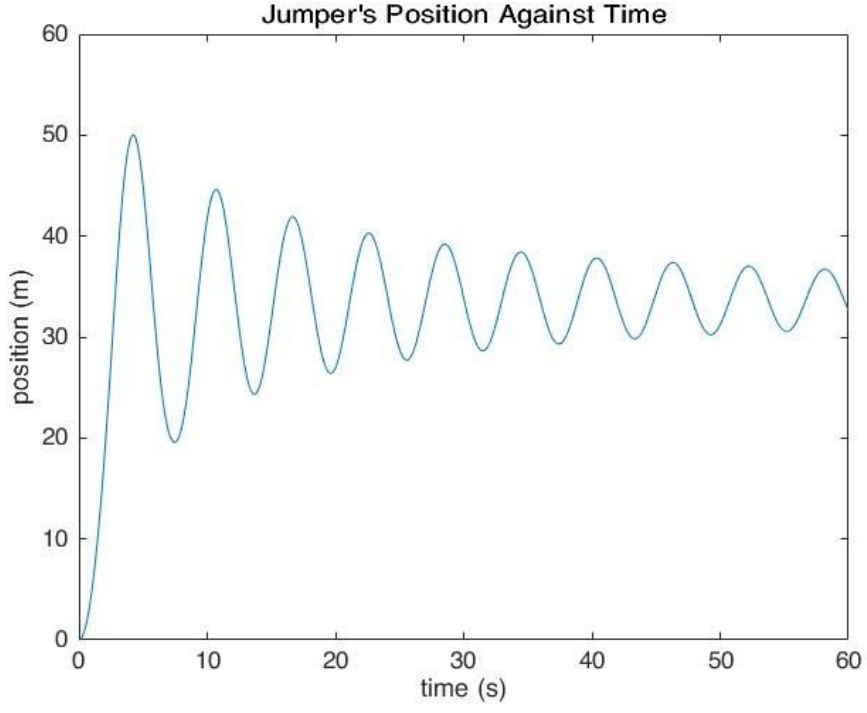
```
    k3_y = h * f1(t(i) + (h/2), v(i) + (k2_v/2));

    k4_v = h * f2(t(i) + h, y(i) + k3_y, v(i) + k3_v);
    k4_y = h * f1(t(i) + h, v(i) + k3_v);

    v(i+1) = v(i) + (1/6 * (k1_v + (2*k2_v) + (2*k3_v) + k4_v));
    y(i+1) = y(i) + (1/6 * (k1_y + (2*k2_y) + (2*k3_y) + k4_y));
end

end
```

The Runge Kutta 4th order method was chosen as the numerical method to solve this ODE because of its accuracy and ease of use. Out of the three numerical methods provided to us, the Runge Kutta 4th order method was the most accurate compared to the Modified Euler Method and it does not require any differentiation, unlike the Second-Order Taylor Method, while still being more accurate than it.

The model requires 3 parameters a, b and n while the outputs are t, v, y and h. b and a are the range of time required with b being the upper limit and a being the lower limit. n is the number of steps the model takes. For the bungee jump we used $a = 0$, $b = 60$ and $n = 1000$ as the bungee jump goes from 0 to 60 seconds and we used 1000 steps for accuracy. The output t is the time vector which is the time for each step, output v is the velocity vector which is the velocity of the jumper for each step, output y is the position of the jumper for each step, and finally output h is the size of steps from the model. Other parameters used in this model are:

- Height of Jump Point $(H) = 74$ m
- Deck Height $(D) = 31$ m
- Drag Coefficient $(c) = 0.9$ kg/m
- Mass of Jumper $(m) = 80$ kg
- Length of Bungee Rope $(L) = 25$ m
- Spring Constant of Rope $(k) = 90$ N/m
- Gravitational Acceleration $(g) = 9.8$ m/s^2

Figure 1. Jumper's Position vs Time Graph



4 Analysis

4.1 Timing and Bounces

Our model agrees with what the company suggests regarding 10 “bounces” in 60 seconds and this can also be seen in Figure 1 in the Numerical Solution section. If we take a look at the figure, we can see that in the graph that there are 10 wave peaks which are the “bounces”. We can calculate this by iterating through all of the y values and storing the peaks as the bounces:

```
bounces = 0; % create new variable to store number of bounces

% start for loop to count bounces which starts from i = 3
for i = 3:n
    % compare y(i) with y(i-1) and y(i-2) to check for bounces
    if(y(i - 2) < y(i - 1) && y(i - 1) > y(i))
        bounces = bounces + 1; % add 1 to bounces if a new bounce is found
    end
end

% print the results
fprintf('There are %d "bounces" in %d seconds\n', bounces, b);
```

Output: `There are 10 "bounces" in 60 seconds`

In the code above, we can see that for the *for* loop, the iterations start from 3, this is because we are comparing 3 values of y and since we do not know y_{1-1} and y_{1-2}, we start with $i = 3$. To check if there is a “bounce” on the graph, we compare y_{i-2} with y_{i-1} and also y_{i-1} with y_i. If y_{i-1} is larger than both y_{i-2} and y_i, it means that y_{i-1} is one of the “bounces”.

4.2 Maximum Speed Experienced by the Jumper

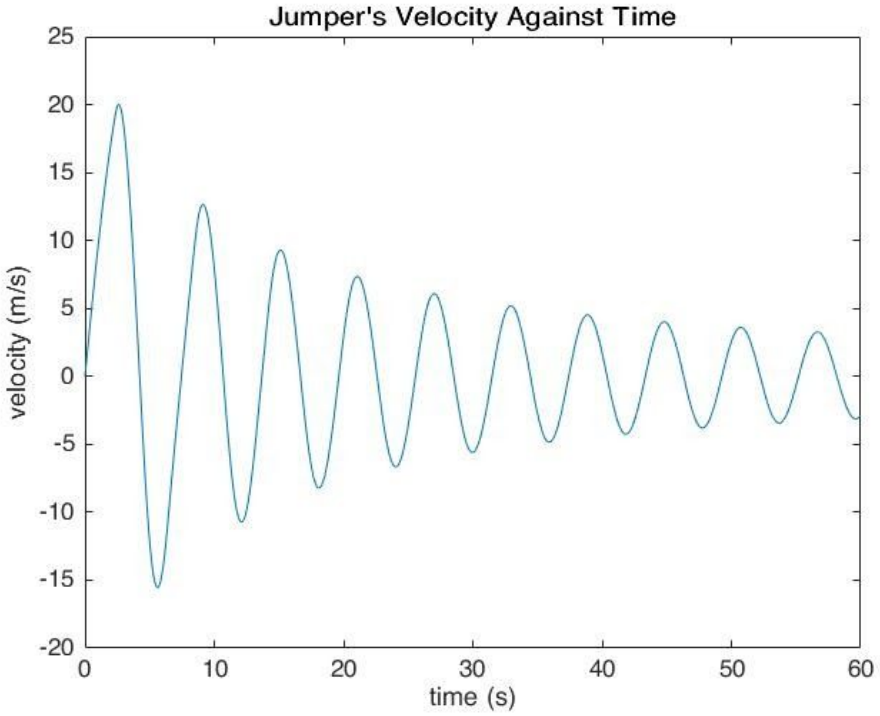
In order to plot the Velocity vs Time graph and find the maximum speed, including at what time it occurs, we can use the following code:

```
% plot velocity vs time graph
figure(2);
plot(t, v);
title('Jumper''s Velocity Against Time');
xlabel('time (s)'); ylabel('velocity (m/s)');

max_speed = 0; % create new variable to store the maximum speed

% start for loop to find what the maximum speed is
for i = 1:n
    if(abs(v(i)) > max_speed)
          max_speed = abs(v(i)); % set current velocity as max_speed if its
larger
    end
end

max_speed_index = find(v == max_speed); % find the index of the max speed
max_speed_time = t(max_speed_index); % find the time the max speed occurs

fprintf('The maximum speed experienced by the jumper is %.4f m/s and it
occurs at %.4f seconds\n', max_speed, max_speed_time); % print the results
```

Output: `The maximum speed experienced by the jumper is 20.0096 m/s and it occurs at 2.5800 seconds`

In the code above, we can see that we do iteration through v in order to find the maximum speed. We then find the index of the maximum speed and it is used to find the time when the maximum speed occurs. The Velocity vs Time graph itself can be viewed in Figure 2 below.

Figure 2. Jumper's Velocity vs Time Graph



4.3 Maximum Acceleration Experienced by the Jumper

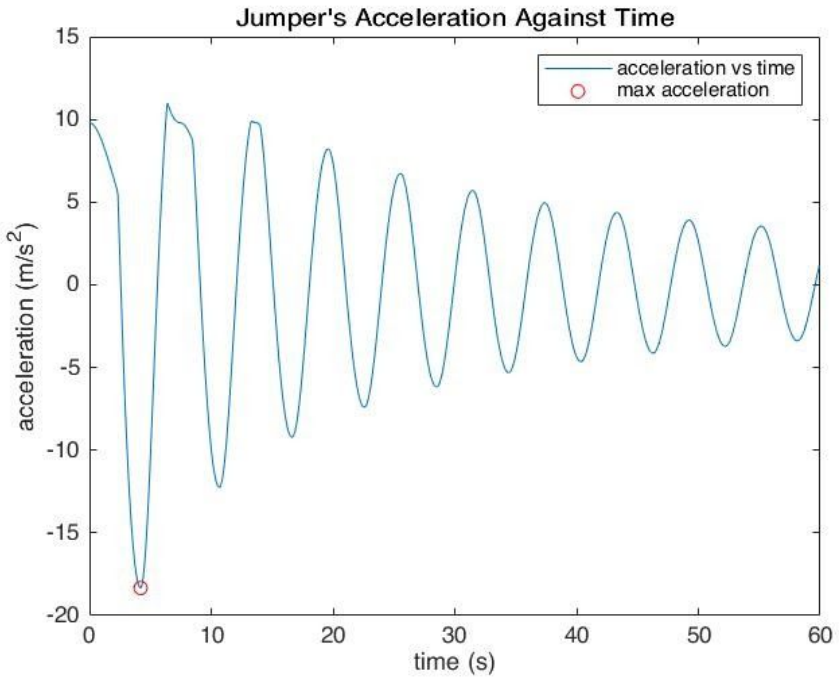
```
h = (b-a)/n; % declare h: size between points

% calculate the acceleration for each point using forward difference method
for i = 1:n
    acc1(i) = (v(i+1)-v(i))./ h; % finding acceleration for acc(i)
end

% fill the last point because we do not have n+1 points
acc1(n+1) = acc1(n);

% plot acceleration vs time graph
plot(t,acc1);
xlabel('Time (s)'); ylabel('Acceleration (m/s)');
title('Jumper''s Acceleration Against Time');

% find coordinates of the largest point
[max_acceleration position_y] = max(abs(acc1))
% finding time using the y position
time = t(position_y)

% plot the maximum acceleration
hold on; plot(time, acc1(position_y), 'ro');
legend('acceleration vs time', 'max acceleration');
```

```
fprintf('The maximum acceleration is %.4f meters per second square and occurs at %.2f seconds',max_acceleration,time); % print the results
```

Output: `The maximum acceleration is 18.3347 meters per second square and occurs at 4.20 seconds`

We obtained a maximum acceleration of 18.3347 m/s^2 and it occurs at 4.20 seconds. $2g$ acceleration is equal to $2 \times 9.8 = 19.6$ meters per second which is quite close to the maximum acceleration we obtained, to be exact it is 1.2653 m/s^2 more than the maximum acceleration obtained. Therefore, the claim of "up to $2g$" acceleration is supported by the model.

Figure 3. Jumper's Acceleration Against Time



4.4 Distance Travelled by the Jumper

The distance travelled by the jumper was found by using the trapezoidal rule.

```
h = (b-a) / n; % declare h: the size between points

S = 0; % initialising S

% Trapezoid rule, summing the values from 2 to n
for j = 2:n
    S = S + abs(v(j)); % add velocity at j (v(j)) to the sum
end

% calculate the integration
I = h/2 * (abs(v(1)) + 2*S + abs(v(n+1)))

% print the results
fprintf('The distance travelled by the jumper is %.4f meters\n', I);
```

Output: `The distance travelled by the jumper is 281.0541 meters`

Therefore, the distance travelled by the bungee jumper is 281.0541 meters

4.5 Automated Camera System

In order to find the time t where $y = H - D$, we can use the following code:

```
H = 74; D = 31; % declare the values of H and D
H_to_D = H - D; % calculate H-D

% set yi and yi+1 to be the min y and y1+2 and y1+3 to be the max y
y_1 = min(y); y_2 = min(y); y_3 = max(y); y_4 = max(y);

% use for loop to find the four closest value to H-D
for i = 1:n
    if round(y(i)) < H_to_D % to find 2 values which are less than H-D
        if y(i) > y_2
            y_1 = y_2;
            y_2 = y(i);
        end
    else
        if round(y(i)) > H_to_D % to find 2 values which are more than H-D
            if y(i) < y_4
                y_4 = y_3;
                y_3 = y(i);
            end
        end
    end
end

% find the index of y_1 to y_4
index_1 = find(y == y_1); index_2 = find(y == y_2); index_3 = find(y == y_3);
index_4 = find(y == y_4);

% find the time when y_1, y_2, y_3 and y_4 occurs
t_1 = t(index_1); t_2 = t(index_2); t_3 = t(index_3); t_4 = t(index_4);

Y = [y_1 y_2 y_3 y_4]; % create Y array to store the four y values
T = [t_1 t_2 t_3 t_4]; % create T array to store the corresponding t for Y

% use the bisection method to find an approximation for t
for i = 1:n
    p = (a+b)/2; % compute the midpoint value
    if sign(divided_difference_q5(T, Y, a) - H_to_D) ==
    sign(divided_difference_q5(T, Y, p) - H_to_D)
        a = p;
    else
        b = p;
    end
end

fprintf('The camera should trigger to capture an image of the jumper at %.4f
seconds\n', p); % to print the results
```

Output: `The camera should trigger to capture an image of the jumper at 3.3330 seconds`

From these calculations, we obtain a result that an automated camera system should capture an image of the jumper at **3.3330 seconds**. We first find 4 y values which are closest to $H - D$ by iterating through y. We then use the Bisection Method to approximate the time t. We can see in the code that a function called `divided_difference_q5` was used. This is a

function used to construct the interpolating polynomial by using Newton Divided Difference. The code is as follows:

```
function interpol = divided_difference_q5(T, Y, t_1)

% construct divided difference table
j = length(T);

table = zeros(j, j); % create a table of 0s with size jxj
table(:,1) = Y; % fill the first column of the table with the values of Y

% use for loop to iterate through the table
for z = 2:j
    for i = z:j
        table(i,z) = (table(i,z-1) - table(i-1,z-1)) / (T(i) - T(i+1-z));
    end
end

% evaluate divided difference
m = size(table);

interpol = zeros(size(t_1)); % initialise sum of interpolating polynomial
for k = 1:m % for loop to iterate through table(k, k)
    P = ones(size(t_1)); % initialise product for interpol
    for i = 1:k-1 % for loop to iterate through T
        P = P .* (t_1-T(i)); % multiply the next factor
    end
    interpol = interpol + table(k,k) * P; % add the next term
end
end
```

4.6 Water Touch Option

Figure 4. Height Displacement Over Time (Original k and L)

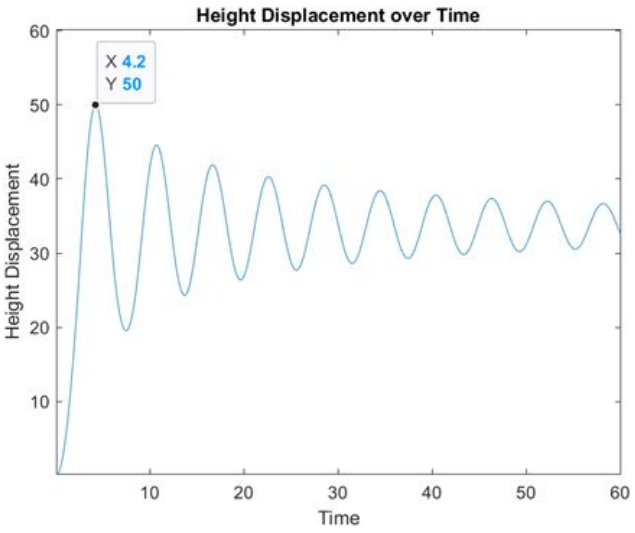
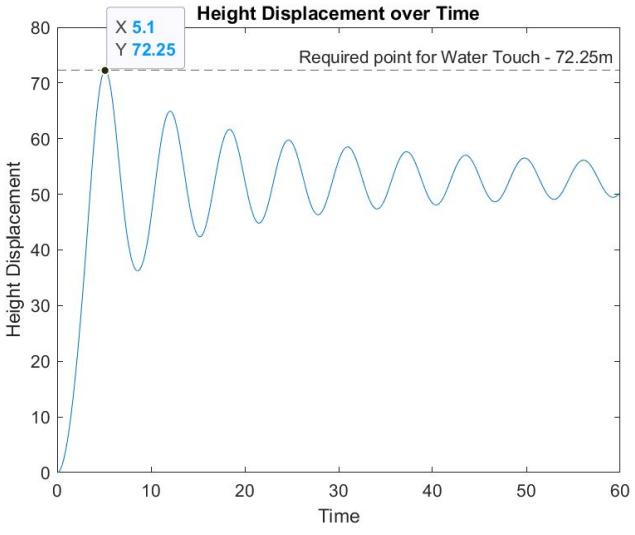


Figure 5. Height Displacement Over Time (Modified k and L)



Given the initial parameters, the jumper travels a total of 50m from the jumping platform, as shown by the graph in Figure 4.

The jumping platform is stated as being at a height of 74 m, and given a jumper of height 1.75 m, would leave the jumpers head 22.25 m above the water.

It is worth pointing out that for the purposes of this calculation the height of the jumper is determined to be the distance between the jumper's head and the attachment point of the rope. In reality the jumper would likely be extending their arms to touch the water and the rope would be tied around their ankles. However, as arm length and the rope attachment point are not specifically given and would not be considered constant, they have not been taken into account.

Whilst altering the variables of rope length and spring constant, it was discovered that decreasing the spring constant too low invariably resulted in too few bounces in 60 seconds, whilst increasing the rope length too far, resulted in the acceleration exceeding the set limit of 2g's of force. For these reasons, the model only takes into account spring constants of 80 and above. The attached bungee script calculated that the ideal parameters for the abovementioned 'water touch' option are as follows:

Spring Constant of Rope (k) = 80.00 N/m
Rope Length (L) = 42.9 m

As can be seen in Figure 5 above, this scenario does indeed result in a water touch whilst achieving almost 10 complete bounces. Given that the accuracy of the model correlates to the value of n, to ensure the model can find a solution, a tolerance of 0.01m has been introduced.

Using the method from 4.3 to calculate acceleration, the maximum acceleration in this scenario was determined to be **19.5534** m/s and occurred **5.10 seconds** after jumping. This falls below the stated threshold of $2g$'s or $2 \times 9.8 =$ **19.6** m/s.

Whilst there may be additional scenarios which also result in a water touch option, the above mentioned parameters meet the requirements as set out in the specification, and is the one arrived at by our model. As such, it is the one we would recommend.

5 Conclusion

The results and outcomes of the model play significant roles for the proposal of bungee jumping off the Story Bridge. These results can suggest the safest ways that the bungee jump could be done and how to implement extra features like an automated camera system and a "water touch" feature. During our analysis, we found that if we followed all safety measures, the jumper can bounce 10 times in 60 seconds. The jumper experiences a maximum speed of 20.0096 m/s at 2.5800 seconds. Moreover, the jumper experiences a maximum acceleration of 18.3347 m/s^2 which occurs at 4.2 seconds. According to our calculation, the bungee jumper's total flight distance is equal to 281.0541 meters. In order to find out at what time an automated camera system should be triggered to capture an image of the jumper, we used the Newton Divided Difference to construct the interpolating polynomial and the Bisection Method to find the time t.

Some recommendations we can put forward are to increase the number of steps taken, which currently is 1000 in the model, in order to increase the accuracy. Furthermore, if the bungee jump organisers decided to include an automated camera system, the camera has to be triggered to capture an image at a particular time, which we found out to be 3.3330 seconds. If the organisers wanted to add a "water touch" feature, there are some modifications which need to be made to the bungee rope spring constant and the length of the bungee rope itself. According to our calculations, the spring constant of the rope reduced to 80 N/m and the length of the rope should be extended to 42.9 m.

6 References

Heck, A., Uylings, P., Kedzierska, E. (2010). *Understanding the physics of bungee jumping*. Senior Physics. http://seniorphysics.com/physics/bungee_physics.pdf

Lincoln, D. (2020). *Air resistance, drag force, and velocity: how falling works.* The Great Course Daily. https://www.thegreatcoursesdaily.com/air-resistance-drag-force-and-velocity-how-falling-works/

Li, Y., & Lu, Z. (2013). *Dynamic analysis of bungee jumping.* Worcester Polytechnic Institute. https://web.wpi.edu/Pubs/E-project/Available/E-project-042313-150806/unrestricted/MQP_final_report.pdf

Wikipedia. (n.d.). *Bungee jumping*. Wikipedia. https://en.wikipedia.org/wiki/Bungee_jumping