MXB103 Project Group 57: BUNGEE!

1 Introduction

As part of Brisbane's "New World City" transformation, the Brisbane City Council is investigating a proposal to allow bungee jumping off the Story Bridge. This report addresses several key questions about the proposal.

In Section 2 of this report, the proposal and the key questions relating to it are discussed in more detail. In Section 3, a mathematical equation representing acceleration of the jumper throughout the jump is derived, followed by the a section describing limitations and assumptions taken into account in this model and the respective parameters used to find solutions.

Section 4 lists the model paramaters used to answer the analysis questions in Section 5, where the model is represented graphically to make predictions about jumper's acceleration, distance and velocity. Leading to the last section which concludes the report and suggests future recommendations.

2 The proposal

It has been proposed that a platform is to be installed at the very top of the Story Bridge for bungee jumping purposes. Our group has been hired as consultants to verify the facts and figures provided by the commerical bungee company. The facts provided will be assessed in Section 5 of this report using a numerical solution designed in Section 4.

The claims made by the bungee company being assessed are: number of bounces, "thrill factor" (maximum velocity and maximum acceleration), distance travelled, camera installation on the bridge deck and a water touch option. These claims will be anaylised in Section 4, chronologically as listed.

The above claims will be estimated using numerical estimations, such as: numerical differentiation, numerical integration, interpolating polynomials. All of which are derived from the initial fourth order method of our model's choice - the Runga Kutta Fourth Order Method.

The next section described the model itself in more detail, including the definitions of motion and interaction of forces in the model.

3 The model

This section describes the mathematical basis for describing the bungee jump, using the given parameters of the scenario.

The equation of motion for bungee jumping is

$$\frac{dv}{dt} = g - C|v|v - \max(0, K(y - L))$$

We have considered all the forces experienced by the jumper during the jump and used Newton's second law ($F = m * a$) to create the equation $m * \frac{dv}{dt} = mg - c|v|v - max(0, k(y - L))$. This shows the interaction of upward and downward forces, which result in kinetic change in the jumper's motion. This particular function shows that the derivative of velocity with respect to time, or in other words acceleration, consists of an interaction between gravitational, drag, and spring tension (from the rope) characteristics.

Here, RHS of the equation is the sum of all the forces - gravity ($m * g$), drag($-c * abs(v) * v$) and tension ($-max(0, k(y - L))$).

m is mass of the jumper, g is gravitational acceleration, c is drag coefficient, k is spring constant of the bungee rope, y is the distance covered by the jumper and L is length of the bungee rope.

Dividing both sides by m gives us $\frac{dv}{dt} = g - C|v|v - max(0, K(y - L))$ where $C = c/m$ and $K = k/m$.

3.1 Assumptions and limitations

Several assumptions and limitations are to be accounted for in this model. We ignore the rounding errors and inaccuracies as we have finite memory in MATLAB as well as other factors like the other forces acting on jumper, uncontrollable weather parameters.

It is also important to note that the model assumes that positive values are downward with respect to the real world, and that conversely negative values represent upward movements and forces.

We are assuming stability in the chosen rope's tension, and do not consider any wear, temperature differences or other factors which could possible change the tension properties of the rope.

Aside from the extent that the jumper affects the values of constants K and C, we are also assuming the drag coefficient of each jumper is constant, because it is impractical to assign unique values to each jumper by way of actually calculating drag force ahead of each jump.

For finding the conditions under which the jumper just touches the water, we have assumed the value of H includes the height of the jumper and hence it does not need to be accounted for. We are also assuming a constant average water height of the value supplied, and do not consider the effect of tides on the water touch option.

3.2 Parameters

The following intial parameter values are used in this model.

```
H = 74;             % Height of jump point (m)
D = 31;             % Deck height (m)
c = 0.9;            % Drag coefficient (kg/m)
m = 80;             % Mass of the jumper (kg)
L = 25;             % Length of bungee cord (m)
k = 90;             % Spring constant of bungee cord (N/m)
g = 9.8;            % Gravitational acceleration (m/s^2)
C = c/m;            % Scaled drag coefficient
K = k/m;            % Scaled spring constant
```

4 The numerical method

We implemented the Runge-Kutta Fourth Order (RK4) method to find the velocity and jumper position in the given model. which requires the generation of four partial approximations (k1 to k4), which are then combined together to create the final approximations for v (velocity) and y (position) from the formula previously described. We used the equations derived from the project specification as below:

$$k1_v = h * f1(t(j), y(j), v(j))$$

$$k1_y = h * f2(t(j), y(j), v(j))$$

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

$$k3_v = h * f1(t(j) + h/2, y(j) + k2_y/2, v(j) + k2_v/2)$$
$$k3_y = h * f2(t(j) + h/2, y(j) + k2_y/2, v(j) + k2_v/2)$$

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

$$v_{j+1} = v_j + 1/6 * (k1_v + 2 * k2_v + 2 * k3_v + k4_v)$$
$$y_{j+1} = y_j + 1/6 * (k1_y + 2 * k2_y + 2 * k3_y + k4_y)$$

4.1 Parameters

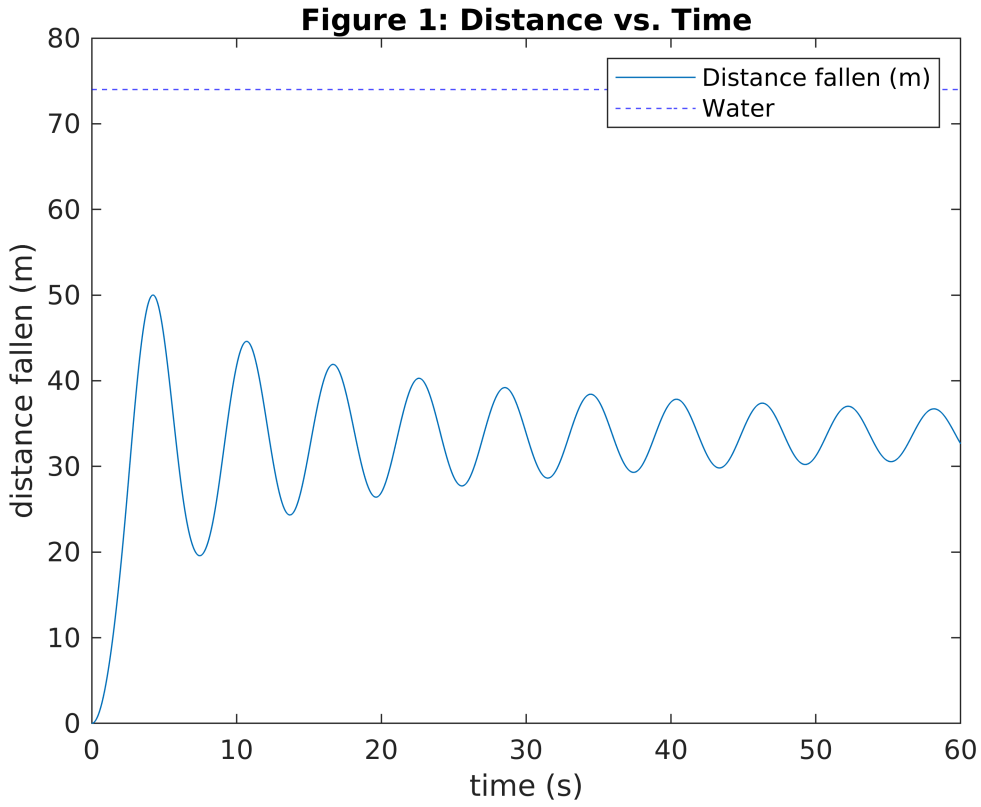
The following were the numerical approximation parameters used in the model. A total of 60 seconds is all that is required, and 10000 subintervals for approximation calculations is sufficient.

```
T = 60;              % Final time in simulation (s)
n = 10000;           % Number of subintervals (equivalent to 0.006s / step)
```

4.2 Solution

The ordinary differential equations are solved using the Runge-Kutta Fourth Order Method.

```
[t, y, v, h] = rk4_bungee(T, n, g, C, K, L);

% Create new figure.
figure

% Plot jumper's velocity against time.
plot(t, y);

% Label figure.
xlabel('time (s)');
ylabel('distance fallen (m)');
title('Figure 1: Distance vs. Time');
hold on;

% Plot the ground to aid understanding
yline(H, 'b--');
hold off;

legend('Distance fallen (m)', 'Water');
```



5 Analysis

This section will verify the experience which is being sought by the client, by proving certain facts about the scenario using the given parameters.

5.1 Timing and bounces

It is important for the bungee jump company to define a consistent definition for a standard jump. This is so that the timing can be consistent but still maintain a fun experience. They have determined this value to be **10 "bounces" within 60 seconds**.

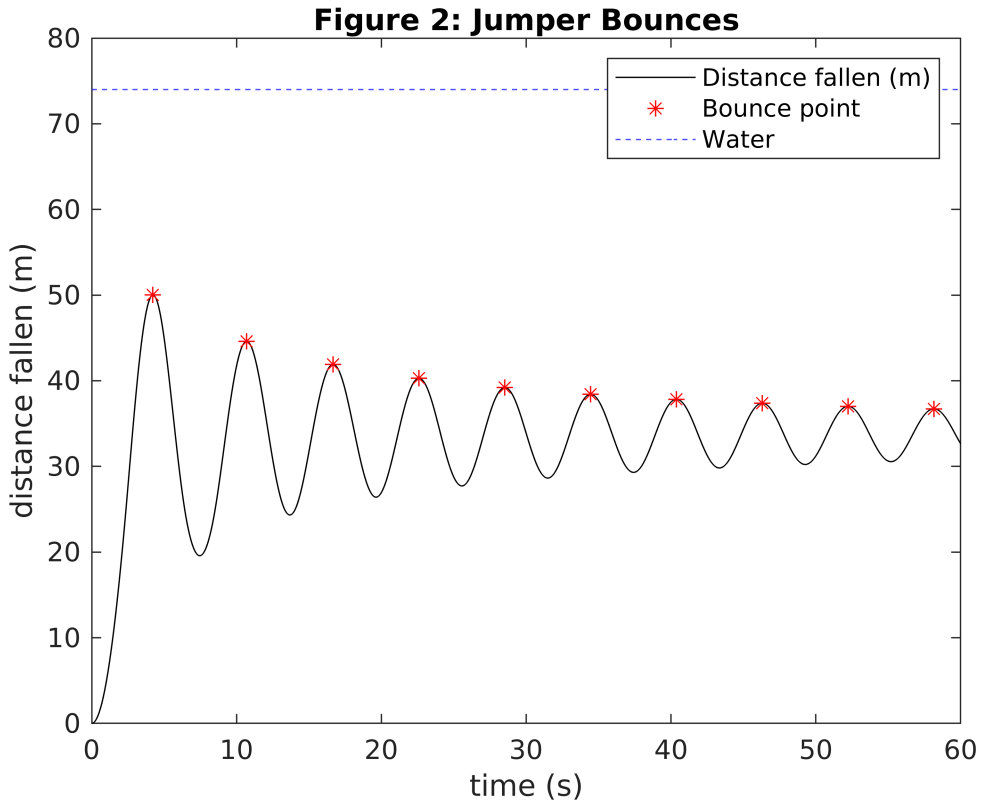
Firstly the assumption of what defines a "bounce" should be considered: which has been determined to be a change of direction from travelling down from the platform towards the water, to travelling up towards the platform again.

This involves the identification of a bounce, by finding a point where the two points either side of that point are lower in value (and hence according to the model, higher off the ground).

Figure 2 indicates the locations of the bounces, and **confirms that the model predicts 10 bounces to occur within 60 seconds**.

```
% Create a new figure
figure

% Plot the jumper position in black
plot(t, y, 'k-');
hold on;
```

```
% Calculate bounce points by checking groups of three points to find a
% combination such that:
%     (point) < (point+1) > (point+2)
bounce_x = [];
bounce_y = [];
for i = 1:n-1
   if y(i) < y(i+1) && y(i+1) > y(i+2)
       % Add a new detected bounce (cannot preallocate because the output
       % size (number of bounces) is not known ahead of time.
       bounce_x = [bounce_x, t(i)];
       bounce_y = [bounce_y, y(i)];
   end
end

% Plot the bounce positions
plot(bounce_x, bounce_y, 'r*');

% Plot the ground to aid understanding
yline(H, 'b--');

% Label figure.
xlabel('time (s)');
ylabel('distance fallen (m)');
title('Figure 2: Jumper Bounces');
legend("Distance fallen (m)", "Bounce point", "Water");
```



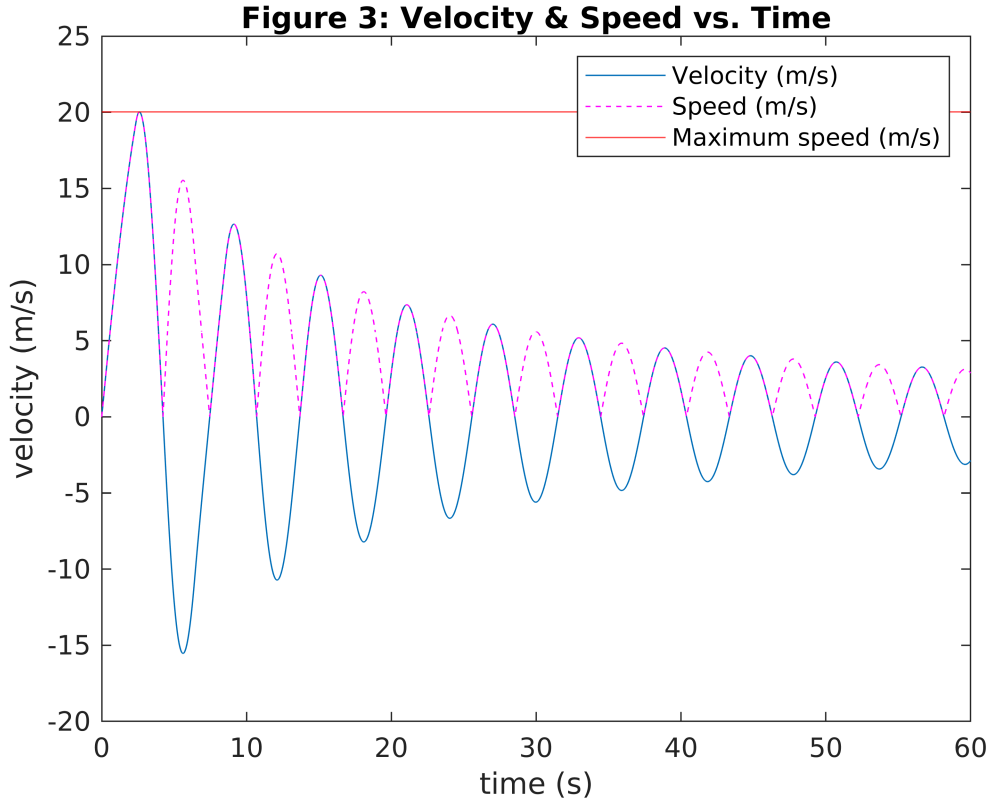
5.2 Maximum speed experienced by the jumper

To assist in determining "thrill factor", the maximum speed is a key component. Speed is defined in this section as the magnitude of velocity in any direction, as measured in metres per second.

Figure 3 shows that the maximum speed is reached on the first fall, before the bounce at the bottom of the rope length. This is because the rope's forces act upon the jumper when the rope becomes taught, causing deceleration and dampening the ability of the jumper to reach any higher speeds.

The calculated maximum speed is **20.0144 m/s** and occurs at **2.6 seconds** into the jump. At this point, the jumper is **29.71 metres** into the **first drop** of the jump, or about **59% of the total maximum height** they will fall.

```
% Convert velocity to speed
speed = abs(v);

% Plot both velocity and speed to make clear the relationship
figure
plot(t, v);
hold on;
plot(t, speed, 'm--');

% Find the max_speed and plot a line indicating it on the figure
[max_speed, max_speed_index] = max(speed);
yline(max_speed, 'r-');

% Calculate extra contextual information about the max speed
max_speed_time = t(max_speed_index);
max_speed_position = y(max_speed_index);
max_speed_relative_position = max_speed_position / max(y);

% Apply labelling to the figure
xlabel('time (s)');
ylabel('velocity (m/s)');
title('Figure 3: Velocity & Speed vs. Time');
legend('Velocity (m/s)', 'Speed (m/s)', 'Maximum speed (m/s)');
```



5.3 Maximum acceleration experienced by the jumper

Another "Thrill factor" for thrill-seekers is the maximum acceleration which will be calculated in this section of the report. Since the acceleration fluctuates between negative and positive values, the maximum acceleration will be referred to by the accelerations magnitude - ie. $|a|$ (absolute value of a).

However, as mentioned, too much acceleration can hinder the experience and possibly even cause harm. According to Aerotime.aero (**Source 1**) the maximum acceleration an average person can handle is ~5g, which can be translated to ~49ms^-2. This will be taken into account if the maximum acceleration exceeds the current claim of 'up to 2g,' as too much g-force is harmful.

The claim for this bungee company is that the jumper will reach an acceleration of 'up to 2g' which is approximately $2 * g$ = 19.6ms^-2. The below will demonstrate an approximate acceleration using the velocity aquired from the rk4_bungee function and the central difference formula stated below:

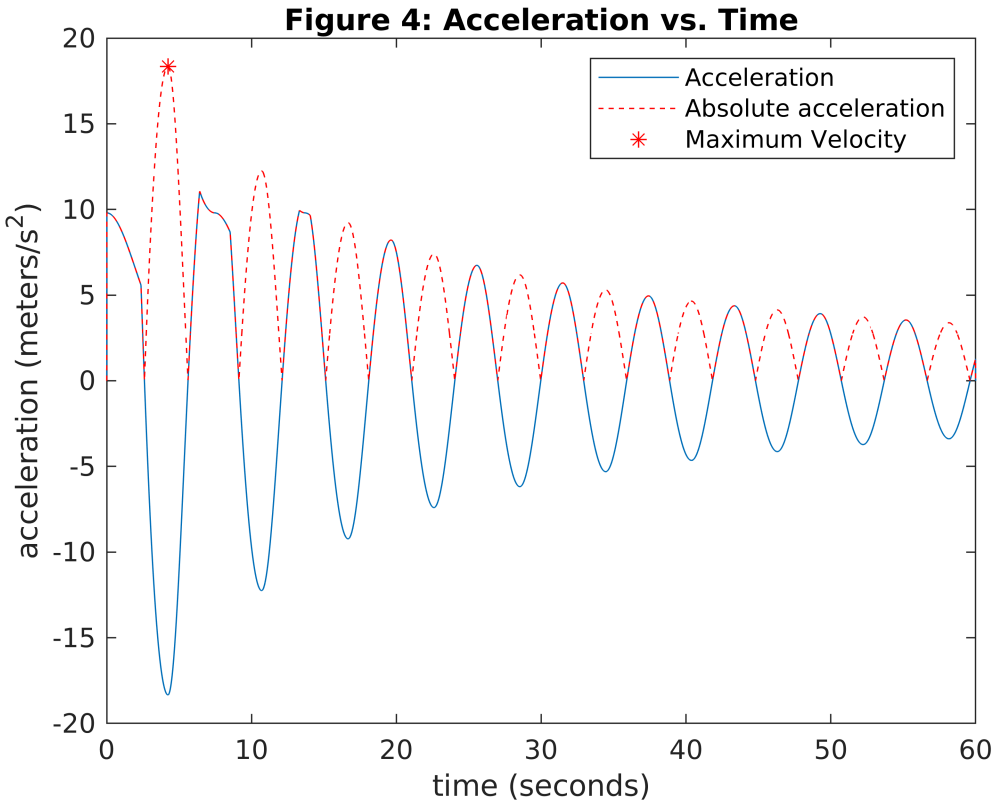
$$f'(x0) = (f(x0 + h) - f(x0 - h))/2 * h$$

Figure 4 Consists of an acceleration vs. time graph, where the maximum acceleration can be indicated via the '*' at the top of the first curve. The maximum acceleration, according to our model is **18.3365 ms^-2** which is below **19.6ms^-2**, the claim made by the bungee company. Therefore, our model **supports** the claim of "up to 2g" made by the bungee jump company.

Source 1 : https://www.aerotime.aero/aerotime.extra/22913-2721

```
% Initialise array to hold acceleration values.
a = zeros(length(y),1);
```

```
% Set the first acceleration value to '0', since when $t$ (time) is 0 the
% bungee jumper is stationary with no acceleration and velocity.
a(1) = 0;

% Iterate over calculated velocity values using the second_order_centralv2
% function to approximate the acceleration.
for i = 2:length(a)-1
    a(i) = second_order_centralv2(v(i-1),v(i+1),h);
end

% Create new figure and plot the approximated acceleration values against
% time.
figure;
plot(t,a);
hold on;

% Plot the absolute value of the acceleration to visualise the maximum
% acceleration and find the maximum acceleration.
[max_Accel,MaxIndex] = max(abs(a));
plot(t,abs(a),'r--');
plot(t(MaxIndex),max_Accel, 'r*');

% Appropriately label figure.
title('Figure 4: Acceleration vs. Time');
xlabel('time (seconds)');
ylabel('acceleration (meters/s^2)');
legend('Acceleration','Absolute acceleration', 'Maximum Velocity');

hold off;
```




```
% Print the maximum acceleration to the Command Window.
disp("Maximum Acceleration: " + max_Accel);
```

```
Maximum Acceleration: 18.3365
```

5.4 Distance travelled by the jumper

For the writing of promotional material purposes, the bungee jump company wants to know how far the jumper actually travels in the 60 second jump. The integral of v (velocity) will give an estimation of the total distance travelled. To estimate the integral of 'v' numerical integration will be used, specifically the trapezoidal rule.

The most accurate numerical integration method to use would be Simpson's Rule however, since our 'rk4_bungee' function produces an odd number of results (10,001 due to being inclusive of $t = 0$) Simpson's Rule cannot be used. This is due to Simpson's Rule alternating between two constant values (2 and 4) meaning an even number of results is needed for the formula to work as intended. This is why the Trapezoidal Rule will be used instead.

Trapezoidal Rule Formula: $S = S + 2 * f(x_i)$, Where a < x_i < b

```
$$I = h/2 * (f(a) + S + f(b));$
```

Using our model and the Trapezoidal Rule, the calculated distance travelled is approximately **281.0451m (meters)** in 60 seconds.

Velocity has already converted to speed (v (absolute value of v)), therefore use the 'speed' variable.

```
% Call the modified trapezoidal rule to estimate the distance travelled
% with in 60 seconds (0 <= t <= 60).
% Since the rk4 method already holds results from 0-60, the trapezoidal
% rule has been modified to take in those already calculated 'v' array and
% 'h' value.
Distance_travelled = traprule_v2(speed,h);

% Display the distance travelled to the command line.
disp("Distance Travelled: " + Distance_travelled + "m");
```

```
Distance Travelled: 281.0451m
```

5.5 Automated camera system

The proposal includes having a camera installed at height D from the water, which makes the equivalent distance fallen value $H - D$. This camera will take a photo of the jumper as they pass the observation deck and allow them to purchase the photo afterwards.

To find the time into the jump to take this photograph, a function will be interpolated to fit four known points, such that two points fit either side of the sought value.

The interpolated function is then shifted on the y axis by the value of $H - D$ to allow a root-finding method to seek the camera timing.

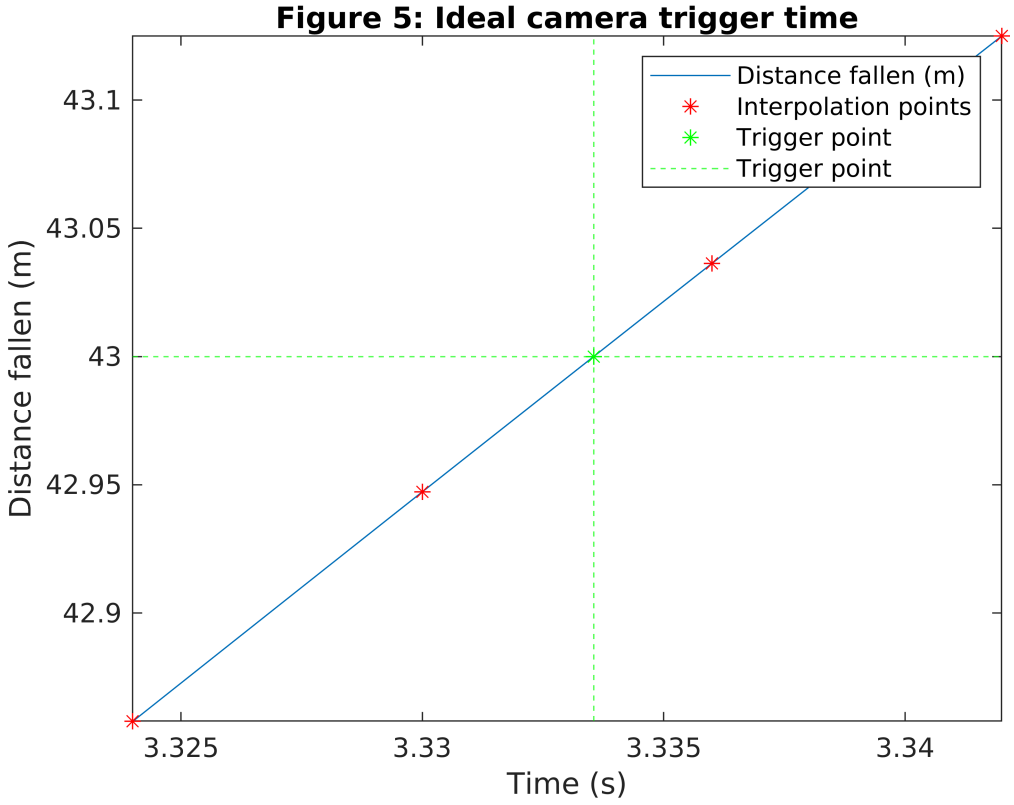
Then the secant root-finding method is used to find the root, which represents the time into the jump when the jumper is at the required height $H - D$. Given the nature of the photography a high shutter speed is likely to be used, around 1/1000 seconds. This means at least this level of accuracy is needed to time the shot correctly, and ideally much more than that.

The secant method is configured to seek the correct value with accuracy of 1e-12 seconds, which is tremendously more than is needed.

The results of this part of the proposal study indicate that the camera should be triggered so that a photo is taken at **3.3335 seconds**. This timing is when the photo should be taken, so does not consider the delay in actually triggering or taking the photo. So independently of camera equipment used, that is the final value to actually take the photo at.

```
% Part (a) - Find four indices of the set of y values surrounding a point
% where $y(t) = H-D$
camera_i = zeros(4);
for i = 3:(n-2)
    if y(i-2) < (H-D) && y(i-1) < (H-D) && y(i) > (H-D) && y(i+1) > (H-D)
        camera_i = [(i-2), (i-1), (i), (i+1)];
        break
    end
end

% Part (b) - Fit an interpolating polynomial using Newton's Forward
```

```
% Differences. It is fine to use this instead of divided differences
% because it will not be appended to or revisited.
camera_t = t(camera_i);
camera_y = y(camera_i);
camera_T = forward_differences(camera_y);
interpolated_pos = @(x) forward_eval(camera_t, camera_T, x);

% Part (c) - Root find using secant method. This method seeks to the
% position of $x$ where $y=0$, so by shifting the interpolated position
% function by the actual sought value ($H - D$) then the secant method will
% be seeking the camera position.
interpolated_pos_shifted_root = @(x) interpolated_pos(x) - (H-D);
cam_pos_x = secant(interpolated_pos_shifted_root, camera_t(1), camera_t(4), 1e-12, 10);

% Part (d) - Time at which to trigger camera
figure;
fplot(interpolated_pos, [camera_t(1) camera_t(4)]);
hold on;
plot(camera_t, camera_y, 'r*');
plot(cam_pos_x, interpolated_pos(cam_pos_x), 'g*');
yline(interpolated_pos(cam_pos_x), 'g--');
xline(cam_pos_x, 'g--');
title('Figure 5: Ideal camera trigger time');
ylabel('Distance fallen (m)');
xlabel('Time (s)');
legend('Distance fallen (m)', 'Interpolation points', 'Trigger point', 'Trigger point')
```



```
disp("Photo time: " + cam_pos_x + " seconds.");
```

```
Photo time: 3.3335 seconds.
```

5.6 Water touch option

The company is interested in a new option where the jumper will drop far enough that they will touch the water at their first bounce while maintaining as close to ten bounce and maximum 2g limit.

A solution was found by varying a series of values for L (Length of bungee rope) and spring constant k, and using the previous methods for counting bounces, maximum acceleration, and finding the maximum drop height as close to 74m as possible.

The final solution below contains the best result conceived where the jumper dips ***9.30mm*** into the water while still maintaining ten bounces, and has a maximum acceleration of ***below 2g***. Detail of exact values can be seen before the following graphs.

```
% Length of bungee rope and spring constant k for water touch option.
L_w = 43.6;             % Length of bungee cord (m)
k = 75.8;              % Spring constant of bungee cord (N/m)
K_w = k/m;             % Scaled spring constant


[t_w, y_w, v_w, h_w] = rk4_bungee(T, n, g, C, K_w, L_w);

figure()
plot(t_w, y_w);
xlabel('time (s)');
ylabel('distance fallen (m)');
title('Figure 6: Distance fallen Water Touch.');
disp("Maximum height when jumper touches the water during his first bounce: "+ max(y_w
```

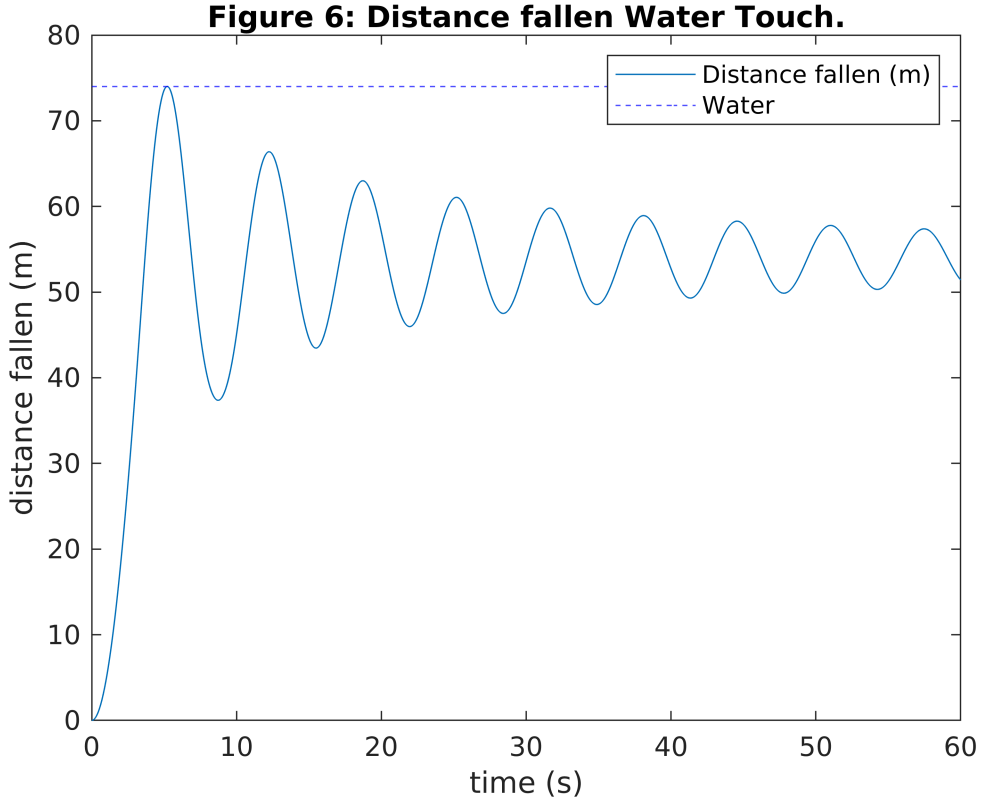
```
Maximum height when jumper touches the water during his first bounce: 74.0093 meters
```

```
disp("Jumper goes " + (max(y_w)- H)*1000+  "mm into the water.");
```

```
Jumper goes 9.3361mm into the water.
```

```


% Plot the ground to aid understanding
yline(H, 'b--');
hold off;

legend('Distance fallen (m)', 'Water');
```

Figure 6: Distance fallen Water Touch.




```
% Initialise array to hold acceleration values.
a_w = zeros(length(y_w),1);

% Set the first acceleration value to '0', since when $t$ (time) is 0 the
% bungee jumper is stationary with no acceleration and velocity.
a_w(1) = 0;

% Iterate over calculated velocity values using the second_order_centralv2
% function to approximate the acceleration.
for i = 2:length(a_w)-1
    a_w(i) = second_order_centralv2(v_w(i-1),v_w(i+1),h_w);
end

% Create new figure and plot the approximated acceleration values against
% time.

% Plot the absolute value of the acceleration to visualise the maximum
% acceleration and find the maximum acceleration.
[max_Accel_w,MaxIndex_w] = max(abs(a_w));


% Print the maximum acceleration to the Command Window.
disp("Maximum Acceleration is " + max_Accel_w+ "ms^-2, which satifies the safety measu
```

```
Maximum Acceleration is 19.0127ms^-2, which satifies the safety measure of maximum acceleration being sma
```

6 Conclusion

In conclusion, the above report addresses several key questions about the given proposal via numerical methods produced in Section 4. Serveral claims hade by the Bungee Company have been assessed in Sections 5.1 and 5.3, both of which hold true according to our numerical models. Section 5.2 covers the estimated maximum acceleration experienced by the bungee jumper, to analyse the "thrill factor" of this particular bungee jump. Section 5.5 estimated the time at which a photo would be taken if a camera was installed at height D from the water. It was estimated that the camera would trigger at: **3.335**, when the jumper passes the observation deck. Lastly, Section 5.6 suggested possible alterations to the bungee jump experience have been considered to implement the 'water touch' option suggested via the bungee company. The lengeth of the bungee cable and its spring constant have been altered to allow for this 'water touch' option, where a jumper of ~80kg would dip approximately ***9.30mm**** into the water.

Recommendations:

It is recommended that if this report is satisfactory, further work is done to confirm and challenge the assumptions held in this report. The focus of further activity or work on this project should be on confirming the safety of the attraction - as safety is currently unverifiable within the scope of this report.

For example, if the "water touch option" is considered, this could have safety implications, were the effects of tidal changes in water level not considered. For this reason the inclusion of tidal factors by using a function to effect a change in variables H and D, would provide a much more satisfactory answer on safety in that regard.

Additionally, by including current temperature/humidity or other factors which could affect the bungee cord material (if it is determined to be an significantly important factor) then safety is increased.

Given a bungee cord is only constrained at the top of the rope and is uncontrolled apart from that anchor, more thought should also be given to the modelling of other axes - particularly as the fall will pass a mezzanine pedestrian deck, which has the potential for jumper collision during wind gusts.

Finally, the overall recommendation for bettering the current model would be the expanding the domain of model inputs, to ensure the range of model outputs fall within acceptable safety factors.