MXB103 Project Group 64: BUNGEE!

Table of Contents

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 fundamental questions about the proposal.

In Section 2 of this report, the proposal and the key issues relating to it are discussed in more detail. In Section 3, the model is expanded on including the mathematical equation representing the acceleration of the jumper and how this equation was formed. This is followed by a discussion of the assumptions and limitations considered with the model and throughout the study as well as the parameters the study is basing its results on.

Section 4 introduces the formulated model and parameters for the formula to then show the model in graph form. Then section 5 comes to the analysis of the graphical predictions addressing the questions of the proposal with further analysis. Finally coming to a conclusion in section 6 discussing findings and future recommendations.

2 The proposal

The Brisbane City Council has proposed for a bungee jumping platform to be installed at the top of the Story Bridge.

We have been tasked with determining if the model will provide an adequate thrill to the customers, while still retaining safety. The model is also used to determine the perfect trigger-point for a camera to go off, to ensure the customer can purchase a photo with which to remember their experience.

Furthermore, a 'water touch' experience is wanted to be offered to allow the customers to experience touching the water. The model is to be used to determine how closely they will touch the water, and how it can be modified to optimise for this without sacrificing safety.

To solve these questions, the fourth order Runge-Kutta method has been used to approximate the velocity and distance fallen of the jumper using the model below. The following model will be used to answer these questions.

3 The model

The equation of motion for bungee jumping is

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

This equation is a compilation of three key forces acted on the bungee jumper over the course of the entire jump. These three forces are gravity (g), drag (-C|v|v), and tension (-max(0,K(y-L)). Note the drag and tension forces act against the fall of the bungee jumper, and are therefore negative.

(g) is gravity, as stated before, (C) is the drag coefficient (c) divided by the mass of the jumper (m). (v) is the velocity of the jumper, (K) is the spring constant of the bungee chord divided by (m). (y) is the height of the jumper relative to the initial position of the jumper, and (L) is the length of the bungee chord.

3.1 Assumptions and limitations

In the model, there will be a range of assumptions that will have to be taken into consideration. The model assumes there be a constant drag on the jumper and a vertical bounce from the bungee cord. This assumption leads to limitations on the model. The model does not accurately depict different types of bounces. Assuming that all bounces are vertical limits the ability to properly depict the amount of thrill and excitement that an individual would gain. Another limitation is derived from the issue of catasptophic cancellation. This limits the accuracy of the results of the model.

Roundoff errors and inaccuracies from methods used contribute to the inaccuracy of the model overall and can increase as multiple methods are strung together. Weather conditions and other minor forces are ignored in this model as well.

3.2 Parameters

```
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

The fourth order Runge-Kutta method was employed to find the velocity and displacement of the jumper with the method below:

$$k1 = h(g - C|v_i|v_i - \max(0, K(y_i - L)))$$

$$k2 = h(g - C|v_i + k1/2|(v_i + k1/2) - \max(0, K(y_i - L)))$$

$$k3 = h(g - C|v_i + k2/2|(v_i + k2/2) - \max(0, K(y_i - L)))$$

$$k4 = h(g - C|v_i + k3|(v_i + k3) - \max(0, K(y_i - L)))$$

$$v_{i+1} = v_i + 1/6(k1 + 2k2 + 2k3 + k4)$$

$$y_{i+1} = y_i + hv_i$$

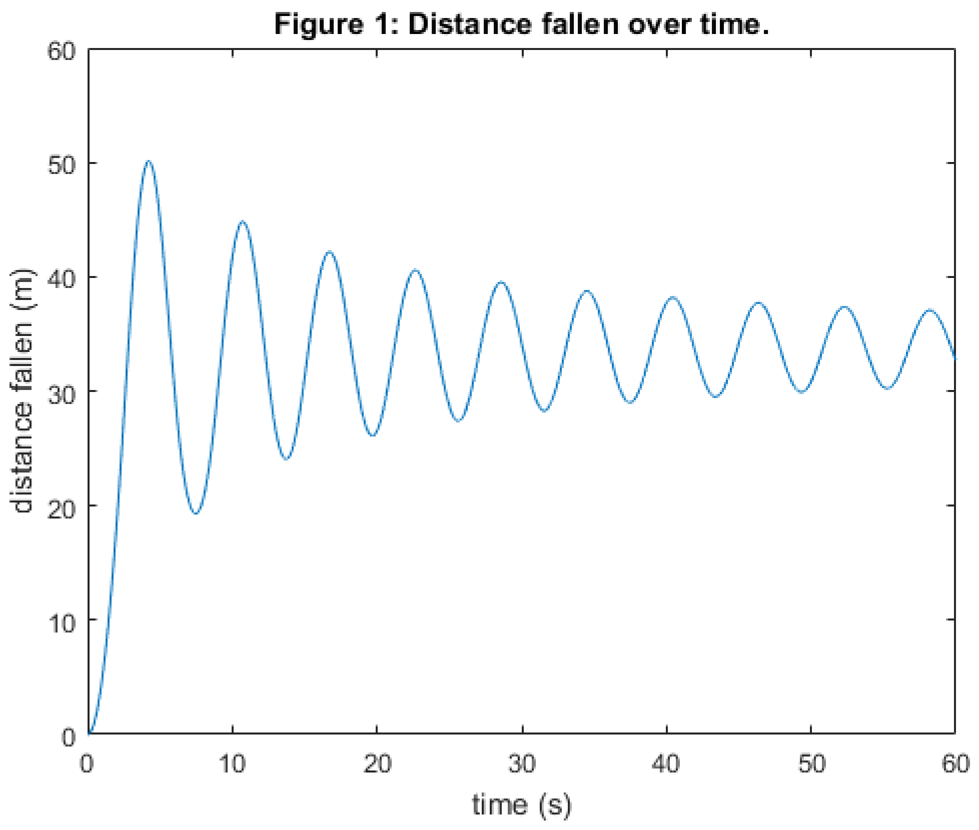
4.1 Parameters

```
T = 60;                 % Final time in simulation (s)
n = 10000;              % Number of subintervals
```

4.2 Solution

The ordinary differential equations are solved using fourth order Runge-Kutta method. The resulting distance y is plotted below.

```
[t, y, v, h] = RK4_bungee(T, n, g, C, K, L);

figure(1)
plot(t, y);
xlabel('time (s)');
ylabel('distance fallen (m)');
title('Figure 1: Distance fallen over time.');
```



5 Analysis

In this section, the model predictions are analysed with respect to the key questions being asked about the proposal.

5.1 Timing and bounces

The question describes the bungee company's suggested number of bounces that should take place within 60 seconds. The company suggested ten bounces be performed within the 60 seconds. Indexing through all y values, the peaks of each wave are counted towards a counter, which shows below having counted ten bounces as the company has suggested.

```
peaks = 0;
for i = 3:n
    if(y(i) < y(i-1) && y(i-1) > y(i-2))
        peaks = peaks + 1;
    end
end

fprintf('In %d seconds, %d bounces take place.\n', T, peaks);
```

```
In 60 seconds, 10 bounces take place.
```

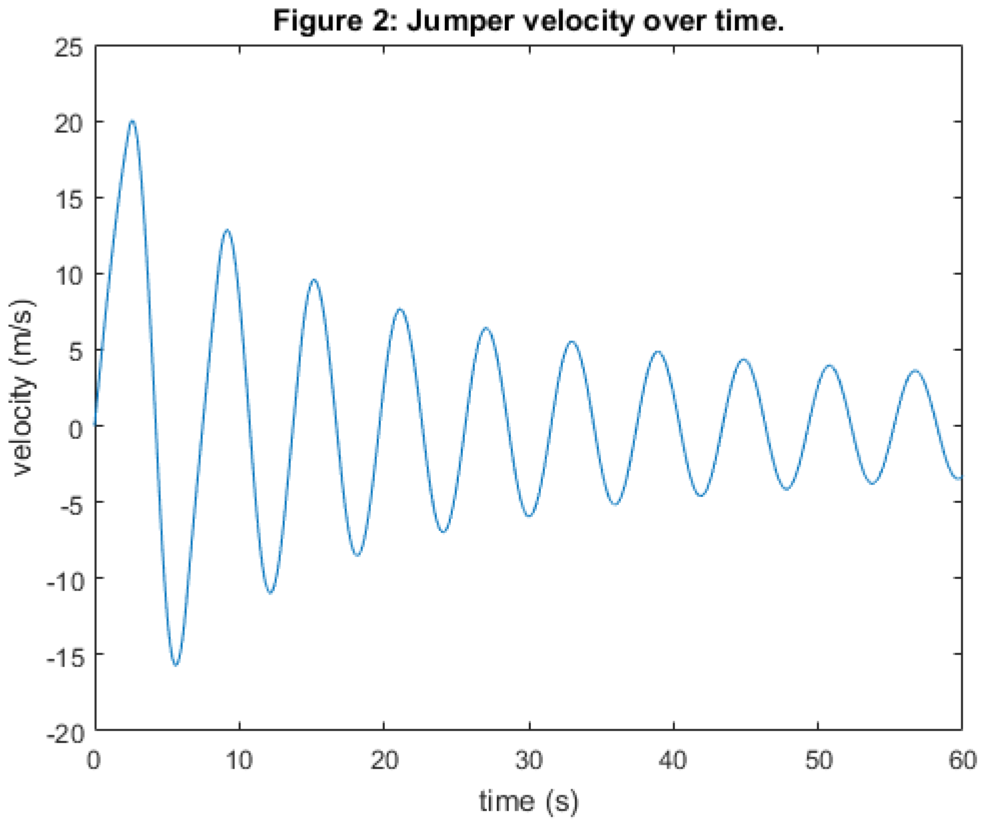
5.2 Maximum speed experienced by the jumper

The company wants to know the maximum speed the jumper experiences to know the velocity "thrill factor". The graph below describes the velocity over the course of 60 seconds beginning at the point the jumper falls.

The highest velocity is roughly 20m/s as indicated by the first peak in the graph, and is also displayed below for more precision.

```
figure(2)
plot(t, v);
xlabel('time (s)');
ylabel('velocity (m/s)');
title('Figure 2: Jumper velocity over time.');

[value, index] = max(v);
fprintf('Maximum speed of %.2fm/s occurs at %.2f seconds', value,
 t(index));
```

```
Maximum speed of 20.04m/s occurs at 2.61 seconds
```



5.3 Maximum acceleration experienced by the jumper

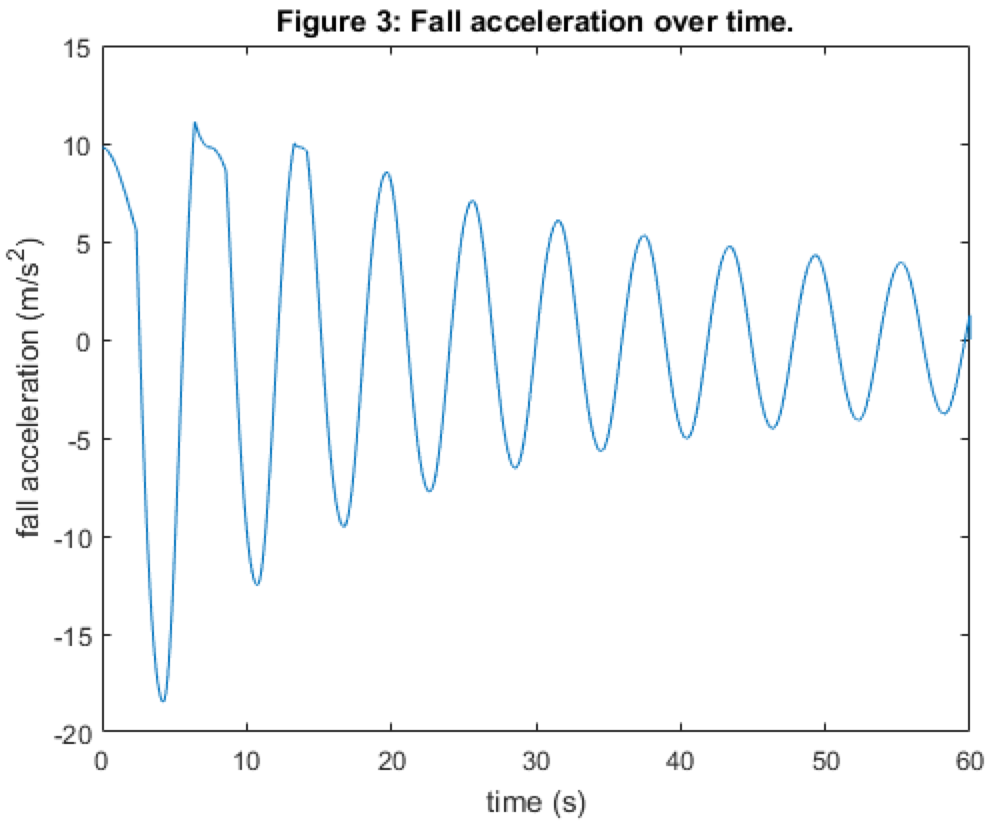
The client boasts of a maximum acceleration of up to 2g without exceeding it and requires verification of that claim. Firstly, the acceleration must be known, which is approximated by integrating the velocity v using the Second Order Central Method. The method has been modified to use indices to get values of v.

Knowing the acceleration at t = 0 is g due to lack of external forces at the exact point the jump begins we skip the first step for the arrays v and y to grab values from the previous and next index which allows Second Order Central differentiation to work. During that process, the maximum acceleration is recorded and displayed below, and as shown, sits comfortably (18.46m/s^2) below the 2g (19.6m/s^2) threshold while still close enough to uphold the company's claim.

```
f = @(t) v(t);
a = zeros(1,n+1);
a(1) = g;
max_a = g; % Finding the maximum acceleration the jumper experiences
index = 1;
for j = 2:n % Indexing through all values of v from 0 to 60 seconds
    a(j) = second_order_central(f, j, index, h);
    if max_a < abs(a(j))
       max_a = abs(a(j));
    end
end
```

```
figure(3)
plot(t, a);
xlabel('time (s)');
ylabel('fall acceleration (m/s^2)');
title('Figure 3: Fall acceleration over time.');

fprintf('The jumper experiences a maximum of %.3fm/s^2 acceleration
 during the 60 second jump', max_a);
```

```
The jumper experiences a maximum of 18.461m/s^2 acceleration during
 the 60 second jump
```



5.4 Distance travelled by the jumper

As a requirement for promotional material, the question asks to compute the total distance the jumper has travelled in 60 seconds. This is achieved by taking the absolute of the velocity v, whereby allowing the total distance to accumulate over the course of the entire jump rather than only when the velocity is positive.

The index of the first and last point of the jump is taken, then using Simpson's Rule the velocity is integrated from velocity at 0 seconds to 60 seconds. The total distance is printed below. Once again the Simpson's Rule function has been modified to take in indices of v.

```
% function for absolute value v
f_absv = @(t) abs(v(t));

% Index for velocity at 0 seconds and at 60 seconds
```

```
a = 1;
b = 10001;

% Simpson's Rule for calculating total distance of the jump.
distance = simprule(f_absv, a, b, n, h);

fprintf('The jumper has traveled %.2f metres in %d seconds', distance,
 T);
```

```
The jumper has traveled 292.10 metres in 60 seconds
```

5.5 Automated camera system

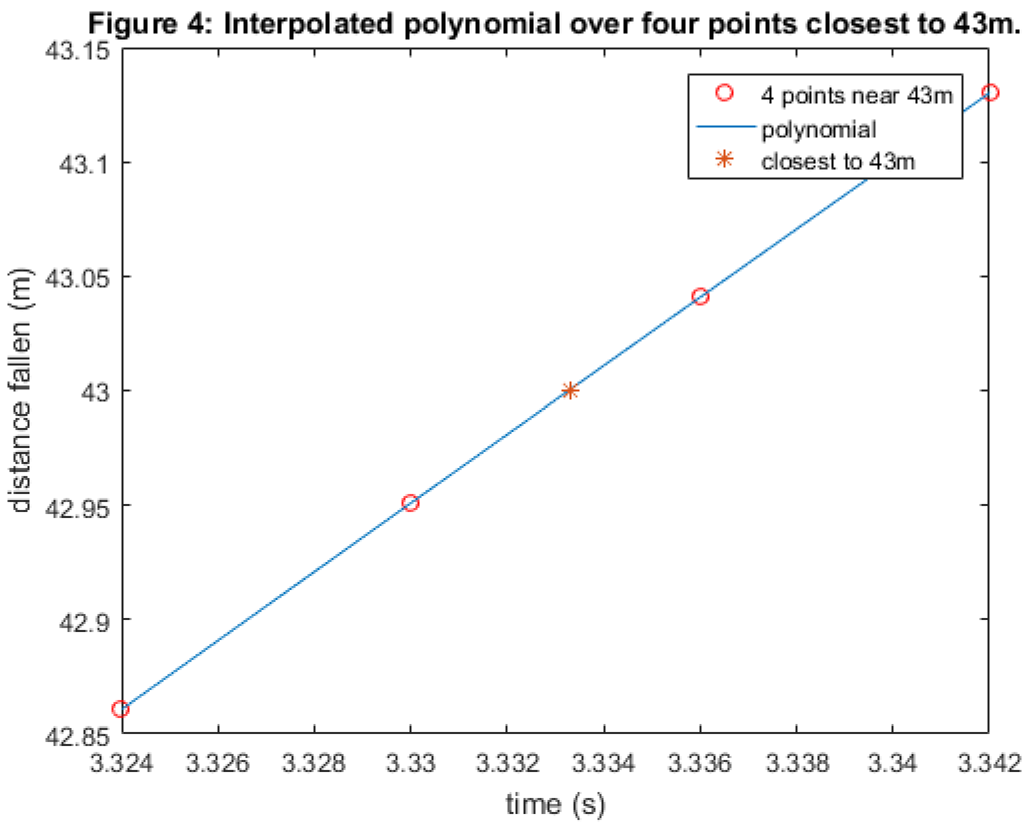
The client plans to implement an automated camera system to capture the jumper as they descend. To do this, the client wants to know the point at which the jumper will reach the camera on their first descent.

The solution below finds the closest four points to the position of the camera relative to the jumper, with two of those points being above the camera and two below. The points are then interpolated using Newton Forward Difference Form, then the closest point to 43m is found through the Bisection Method, modified to work with indices and looping until the closest value of y is found. Printed below is also the resulting height and time the camera should take a photo.

```
y_cam = H - D; % Distance from the jumper to the camera
y_index = 1; % Indexing through y to find the closest value to y_cam

% Once a y value that exceeds y_cam is found, the four y values used
 for
% interpolation can be extracted
while y(y_index) < y_cam
    y_index = y_index + 1;
end
Y_p = [y(y_index - 2) y(y_index - 1) y(y_index) y(y_index + 1)];

% Also grabbing the time for the same four points of Y
T_p = [t(y_index - 2) t(y_index - 1) t(y_index) t(y_index + 1)];

% Considering the y values are all equally spaced by interval h with
% regards to time, it is possible to use Newton Forward Difference
 Form to
% generate the interpolating polynomial for the four points gathered

M = forward_differences(Y_p);
t_interpol = T_p(1):h/100:T_p(length(T_p));
y_interpol = forward_eval(T_p, M, t_interpol);

% Using the bisection method to find the root of the interpolating
% polynomial at 43m
f_root = @(x) y_interpol(x);

% Indeces of the first and last values of y_interpol
root_a = 1;
root_b = length(y_interpol);
```

```
% Using the bisection method to find the index
p = bisection(f_root, root_a, root_b, y_cam);
figure(4)
plot (T_p, Y_p, 'ro');
hold on
plot (t_interpol, y_interpol);
plot (t_interpol(p), y_interpol(p), '*');
xlabel('time (s)');
ylabel('distance fallen (m)');
title('Figure 4: Interpolated polynomial over four points closest to
 43m.');
legend('4 points near 43m', 'polynomial', 'closest to 43m');

fprintf('The camera should trigger at the point %7.5fm after the
 jumper falls for %7.5f seconds', y_interpol(p), t_interpol(p));
```

```
The camera should trigger at the point 43.00041m after the jumper
 falls for 3.33330 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 using a series of values for L and 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 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.

```
% The _w denotes "watertouch"
L_w = 43.6;             % Length of bungee cord (m)
k_w = 76.2;             % Spring constant of bungee cord (N/m)
K_w = k_w/m;            % Scaled spring constant

% Recalculating using RK4 for new values
[t_w, y_w, v_w, h_w] = RK4_bungee(T, n, g, C, K_w, L_w);

% Finding max height fallen
max_y_w = max(y_w);

figure(5)
plot(t_w, y_w);
xlabel('time (s)');
ylabel('distance fallen (m)');
title('Figure 5: Distance fallen water touch.');

% Same method for counting peaks as before
peaks_w = 0;
for i = 3:n
    if(y(i) < y(i-1) && y(i-1) > y(i-2))
        peaks_w = peaks_w + 1;
    end
end


f_w = @(t) v_w(t);
a_w = zeros(1,n+1);
a_w(1) = g;
max_a_w = g; % Finding the maximum acceleration the jumper experiences
index = 1;
for j = 2:n % Indexing through all values of v from 0 to 60 seconds
    a_w(j) = second_order_central(f_w, j, index, h);
    if max_a_w < abs(a_w(j))
       max_a_w = abs(a_w(j));
    end
end

figure(6)
plot(t_w, a_w);
xlabel('time (s)');
ylabel('fall acceleration (m/s^2)');
title('Figure 6: Fall acceleration water touch.');

fprintf('The jumper falls %5.3fm on their first bounce, bounces
 %d times, and has a maximum acceleration of %6.4fm/s^2', max_y_w,
 peaks_w, max_a_w);

The jumper falls 74.030m on their first bounce, bounces 10 times, and
 has a maximum acceleration of 19.1846m/s^2
```

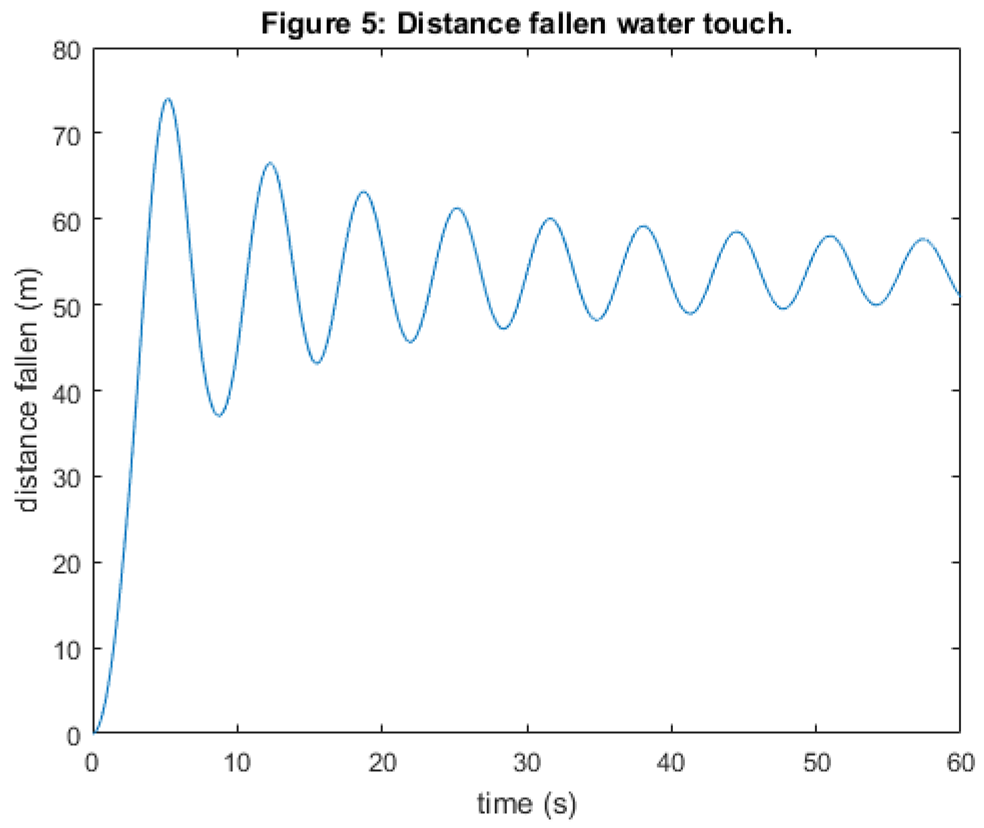


Figure 5: Distance fallen water touch.
distance fallen (m)
time (s)
0
10
20
30
40
50
60
70
80

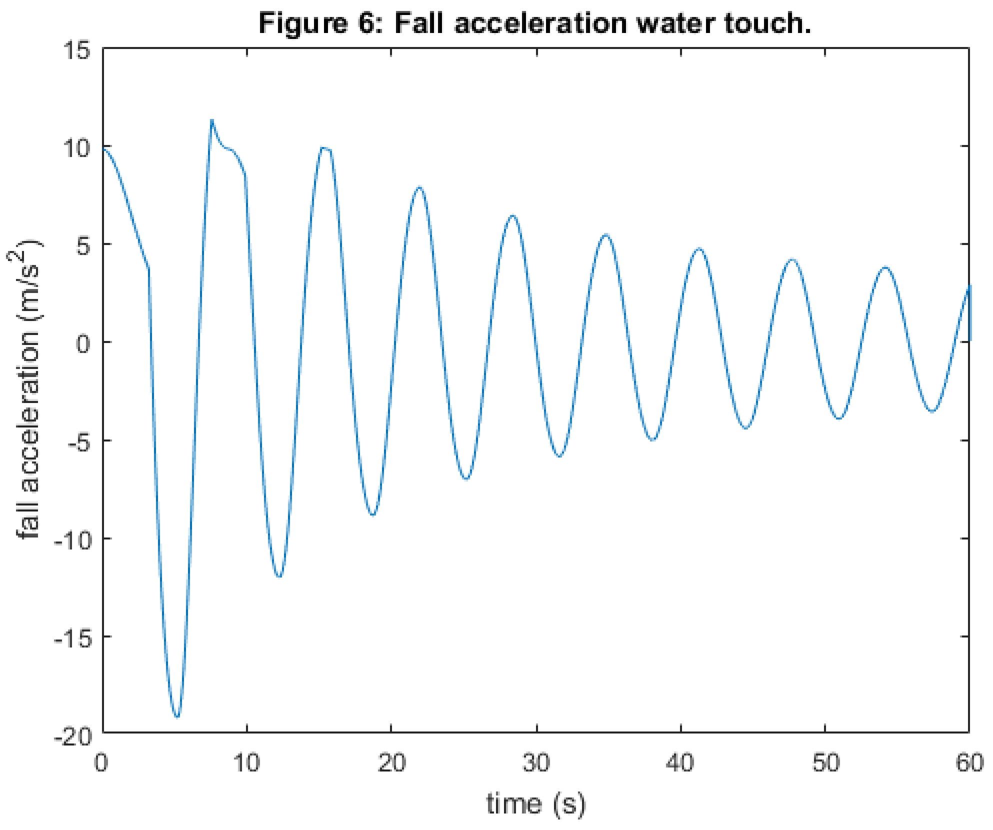


Figure 6: Fall acceleration water touch.
fall acceleration (m/s^2)
time (s)
-20
-15
-10
-5
0
5
10
15
0
10
20
30
40
50
60

6 Conclusion

Overall the proposal is feasible. The model accurately depicts the requirements of the client, allowing the jumper to experience ten jumps within the 60-second timeframe. The maximum acceleration experienced by the jumper is also similar enough to the 2g threshold to market it as such, without sacrificing safety.

Throughout the 60 seconds, the jumper will travel almost 300 metres through the air. For the automatic camera setup, the camera should take the photo approximately 3.3 seconds into the jump, when the jumper has fallen 43 metres.

For the marketed water touch option, a rope length of 43.6m was determined to work best, with a spring constant of 76.2. This allowed the jumper to submerge 30mm into the water, while still retaining the other requirements of the jump.

The bungee setup is both safe and provides an adequate thrill to the jumpers.

Published with MATLAB® R2016b