Mathematical Modelling of Bungee Jump Dynamics with Numerical Methods and ODEs

Anna Lloyd-Jones	N12013595	*Maxwell Hanks*	N12405710
Elliot Phelps	N11273968	*Scott Byrne*	N11239140

1 – Introduction

As part of Brisbane's "New World City" transformation, the Brisbane City Council is proposing to allow bungee jumping off the Story Bridge. Preliminary information, such as model parameters for both the bridge, and a jumper have been provided by a commercial bungee jump operator. In this project, as part of a consultancy contract, we are required to develop a mathematical model of the bungee jumping process, as well as a provide a numerical solution of the jumper's position and velocity using *Fourth Order Runge-Kutta (RK4)* method with the aid of MATLAB.

As well as a mathematical model solution, numerical analysis will be implemented to obtain subsequent performance measures, such as timing and number of bounces, maximum speed and acceleration, distance travelled, optimal timing for automated photography, and feasibility of the "water touch" option.

In Section 2 of this report, the mathematical model describing the forces acting on the jumper is outlined, and the key equations of motions are derived.

2 – Mathematical Model

The mathematical model of bungee jumping can be derived by considering three forces acting on a jumper at all times; gravity (g), drag ($-cv|v|$), and tension in the rope ($-\max\big(0, k(y-L)\big)$). There are four phases Newton's second law of motion states that the sum of the forces must be equal to the product of the jumper's mass and acceleration, hence the following differential equation is formed:

$$m\dot{v} = mg - cv|v| - \max\big(0, k(y-L)\big)$$

Where $\dot{v}$ is the jumper's acceleration. This ODE can be simplified by diving through by m, to obtain

$$\dot{v} = g - Cv|v| - \max\big(0, K(y-L)\big)$$

Where $C = c/m$ and $K = k/m$. During Phase 1, where the cord is slack has no tension ($y \leq L$), hence $-\max\big(0, K_m(y-L)\big) = 0$, and:

$$\dot{v}(t) = \ddot{y}(t) = g - Cv^2$$

With initial condition $v(0) = 0$:

$$\int \mathrm{d}v = \int g - Cv^2 \,\mathrm{d}t$$

$$v(t) = \dot{y}(t) = \sqrt{\frac{g}{C}}\tanh(\sqrt{gC}\,t)$$

Integrating $\dot{y}(t) = v(t)$ gives

$$\int \mathrm{d}y = \int \sqrt{\frac{g}{C}}\tanh(\sqrt{gC}\,t)\,\mathrm{d}t$$

$$\therefore y(t) = \frac{1}{C} \ln \cosh(\sqrt{gC}\, t)$$

During Phase 2, the spring contributes tension force, and the system is as previously mentioned:

$$\dot{v}(t) = g - Cv|v| - K(y - L)$$

Representing this in terms of y:

$$\ddot{y}(t) = g - C\dot{y}|\dot{y}| - K(y - L)$$

The damping force (drag) is nonlinear in $\dot{y}(t)$ and instead quadratic due to $\dot{y}|\dot{y}|$. With linear damping i.e., $\ddot{y}(t) + C\dot{y} + K(y - L) = g$ is solvable in terms of decaying exponentials as seen in logistic growth, etc. With quadratic damping, the ODE is now a nonlinear second-order ODE, which evaluates to:

$$\int \frac{\mathrm{d}v}{g - Cv^2 - K(y - L)}$$

Which is not integrable in closed form by elementary functions.

2.1 – Assumptions and limitations

In this model, a range of assumptions must be taken into consideration for the model to function accurately and to the standards of the company. Firstly, since the differential equation is not integrable by elementary functions, a numerical method will be used for the solution. To account for higher accuracy, RK4 method will be used as opposed to Euler method. However, while RK4 is more accurate than Euler method, local and global error still exists in order of $\mathcal{O}(h^5)$ and $\mathcal{O}(h^4)$ respectively. There is still some error present, limiting the accuracy of the method.

For this model, external factors such as the weather like wind and rain, and initial jumping velocity cannot be accounted for using only ODEs. Furthermore, these differential equations are modelled to simulate a one-dimensional space, where the bungee jumper only travels vertically, hence this model cannot account for e.g., horizontal displacement.

Other limitations with this model can be identified, such as the drag coefficient and spring constant. A jump in the real world would result in different tension and drag in the cord and damper respectively. For this model, only fixed spring constants and drag coefficients can be utilised.

2.2 – Parameters

To model and simulate this jump, various parameters have to be considered, including drag coefficients, jumping height, an average mass of a jumper, bungee length, spring constant, gravity, etc. We have been provided with model parameters as such:

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

3 – Numerical Solution

Thus, the function for RK4 method can be defined as:

```
% Initial conditions
y0 = 0;
v0 = 0;
T = 60;
n = 10000;
% Anonymous function
a = @(t,y,v) g - C*v*abs(v) - K*max(0, y - L);
% Subintervals
h = T/n;        % time step
t = 0:h:T;     % time vector
% State arrays
y = zeros(1,n+1);  v = zeros(1,n+1);
% RK4
for j = 1:n
    k1 = h*(g - C*abs(v(j))*v(j) - max(0, K*(y(j) - L)));
    k2 = h*(g - C*abs(v(j) + k1/2)*(v(j) + k1/2) - max(0, K*(y(j) - L)));
    k3 = h*(g - C*abs(v(j) + k2/2)*(v(j) + k2/2) - max(0, K*(y(j) - L)));
    k4 = h*(g - C*abs(v(j) + k3)*(v(j) + k3) - max(0, K*(y(j) - L)));
    v(j+1) = v(j) + 1/6 * (k1 + 2*k2 + 2*k3 + k4);
    y(j+1) = y(j) + h*v(j);
end
% plot
figure,
plot(t, Y,'b', 'LineWidth',1)
xlabel('time (seconds)');
ylabel('Distance travelled (m)');
title('Distance travelled using RK4 method');
```

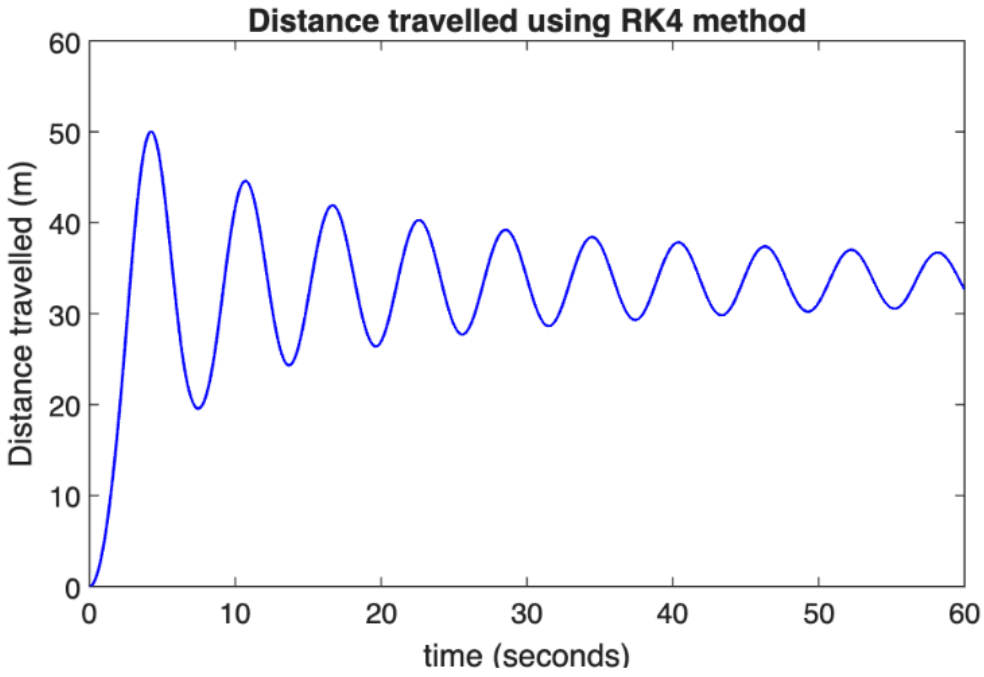

Figure 1: the distance travelled by the bungee jumper, using the RK4 method

4 – Analysis

The consultancy contract requires several key performance factors to be assessed. These include the thrill factor (maximum speed & acceleration), distance travelled, automatic camera settings and whether it is viable to include a "water touch" option. Detailed MATLAB analysis will then be performed to assess these factors and provide advice to the client.

4.1 – Timing and bounces

The timing and bounce analysis was performed to determine if the bungee jump company's claim that a standard jump will consist of 10 "bounces" in approximately 60 seconds is correct. To perform this analysis, the team used the previously created displacement curve, noting that the "peaks" of this curve represent a change in direction, or a "bounce". A simple for-loop was written which counts the peaks, confirming that visual analysis that there are indeed 10 bounces in the 60 second interval.

```
% BOUNCES
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, there are 10 bounces.

4.2 – Maximum speed experienced by the jumper

The maximum speed experienced by the jumper determines the 'thrill factor'. First the velocity profile was obtained from the RK4 method and plotted in figure 2, then the maximum speed was determined from the data using the function fprintf. It was determined that the maximum velocity reached is 20.04 m/s (72.14 km/h), at 2.61 seconds. This occurs during the first free-fall phase of the fall, just before the bungee cord begins to retract/change to an upward direction. The maximum speed is slower than an average bungee jump of 80-110km/h (BUNGEE.IT, 2022), but is within a realistic range for the height of this specific jump, confirming the safety and thrill factor is expected.

```
% VELOCITY
plot(t,(V),'r', 'LineWidth',1)
xlabel('time (seconds)');
ylabel('Velocity (m/s)');
title('Velocity using RK4 method');
```

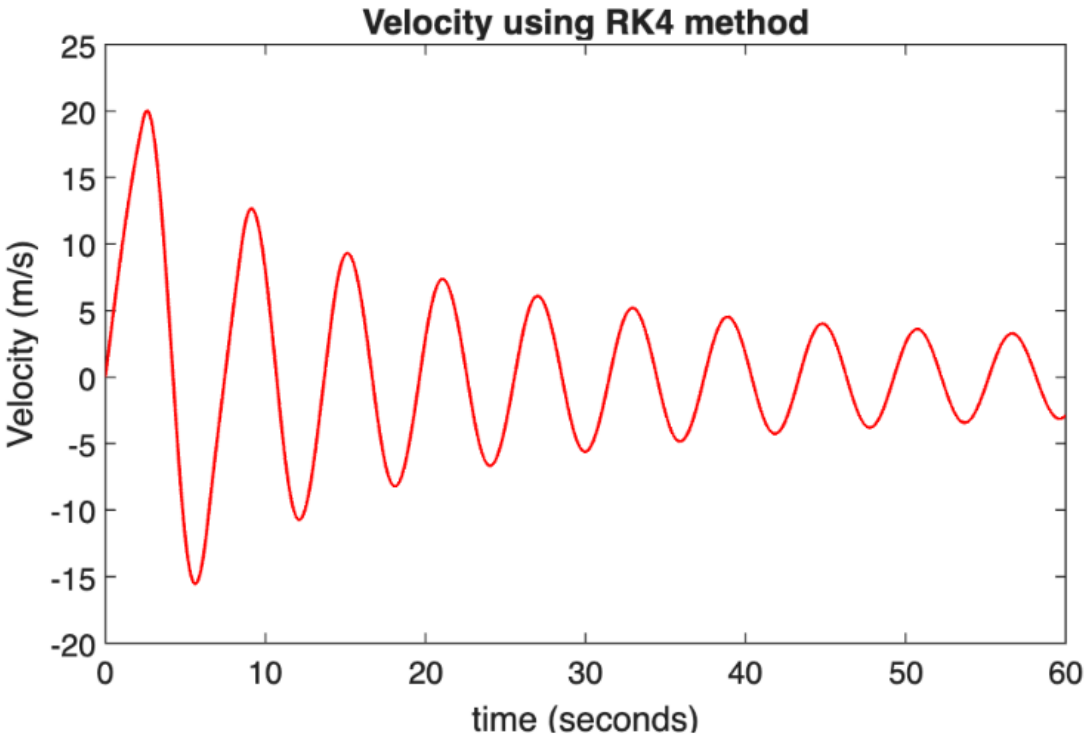

Figure 2: The velocity profile of the jumper, plotted against time using the RK4 method

```
% MAXIMUM SPEED
[value, index] = max(v)
value = 20.0443
index = 436
fprintf('Maximum speed of %.2fm/s occurs at %.2f seconds.\n', value,
t(index));
Maximum speed of 20.04m/s occurs at 2.61 seconds
```

4.3 – Maximum acceleration experienced by jumper

Maximum acceleration was then assessed to verify the claim that the jumper will experience acceleration "up to 2g" (19.62m/s^2). The chosen solution method was to use a second order central differences approximation to find the derivative of the velocity of the jumper over the 60 second interval. The resulting data was then plotted as shown in figure 3, with peak acceleration occurring in the first few seconds reaching close to 20m/s^2. The absolute maximum value was then extracted from the data, confirming an exact value of 18.46m/s^2 occurring 4.23 seconds into the jump. The resulting maximum acceleration of 18.46m/s^2 falls short of the "up to 2g" (19.62m/s^2) claim, however the simulation is based on an 80kg jumper, and max acceleration may reach slightly higher with a higher mass jumper. Although the claim is not 100% verified, the consulting team suspect the claim may be true with higher mass jumpers. The consulting team recommends further analysis to verify the claim, but this is considered out-of-scope of the initial investigation.

```
% ACCELERATION PLOT
a = (v(3:end) - v(1:end-2))/(2*h);
figure(4);
plot(t(1:n-1), a, 'g', 'LineWidth',1)
xlabel('ime (seconds)'); ylabel('Acceleration (m/s^2)');
title(' Jumper acceleration vs time determined using RK4 method');
```

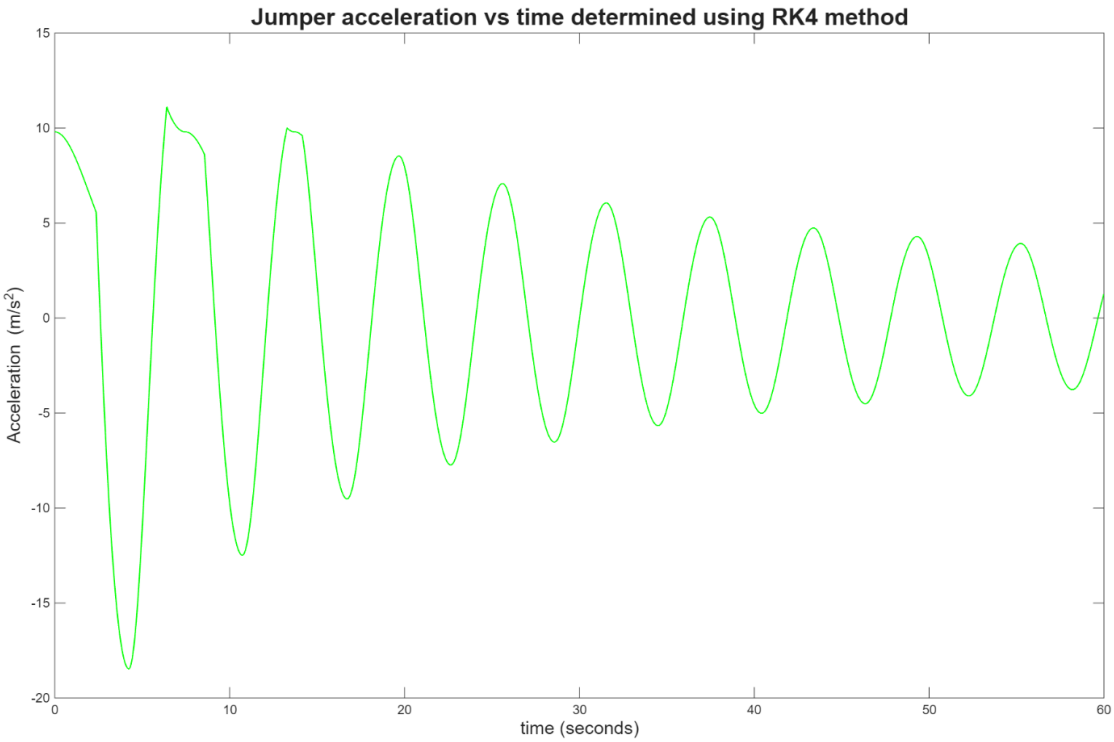

Figure 3: Jumper's acceleration plotted against time using the RK4 method

```
% MAXIMUM ACCELERATION
[value, index] = max(abs(a))
value = 18.4611
index = 706
fprintf('Maximum acceleration of %.3fm/s^2 after %.2f seconds during the 60 second jump.\n', value, t(index))
Maximum acceleration of 18.461m/s^2 after 4.23 seconds during the 60 second jump
```

4.4 – Distance travelled by the jumper

To determine the total distance the jumper travels during the 60 second jump, the magnitude of the velocity is numerically integrated with respect to time. This includes both the upward and downward movements of the bungee jumper. Using the velocity found in the numerical solution and section 4.2, the trapezoidal rule is used to approximate the integral:

$$\int_0^{60} |v|\, dt$$

In MATLAB, $\text{trapz}(t, |v|)$ was used to calculate the integral and approximate the total distance. This determined that the total distance that the jumper will travel is 292.10m over a 60 second time interval.

```
% TOTAL DISTANCE
distance_total = trapz(t, abs(v));  % numerical integral
fprintf('Total distance travelled by the jumper over %.0f seconds: %.2f m\n', T, distance_total);
Total distance travelled by the jumper over 60 seconds: 292.10 m
```

4.5 – Automated camera system

The activation time for an automated camera located on the bridge deck can be determined using an interpolating polynomial $\boldsymbol{p}(t)$ for a slice of the RK4 jumper position dataset described in 4.1 – Timing and Bounces. The camera should activate at t when t satisfies $\boldsymbol{p}(t) = H - D$.

The slice of the RK4 jumper position dataset is the two closest points either side of the camera's position when the jumper first passes the bridge deck.

$$y_i \ , y_{i+1} < H - D < y_{i+2} \ , y_{i+3}$$

Since the desired data occurs at the first set of points that satisfy the above condition, it can be found by testing the above condition against the dataset in order. The following code snippet finds the index of y_{i+2} and returns the set of points $[(x_i, y_i), (x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2}), (x_{i+3}, y_{i+3})]$.

```
function points = findClosestPoints(target, data)

for i = 2:length(data)
    if data(i-1) < target && data(i) > target
        firstClosestPoint = i;
        break;
    end
end

points = [firstClosestPoint - 2, data(firstClosestPoint - 2);
        firstClosestPoint - 1, data(firstClosestPoint - 1);
        firstClosestPoint, data(firstClosestPoint);
        firstClosestPoint + 1, data(firstClosestPoint + 1)];
```

Using these points, the interpolating polynomial can be found by solving the associated Vandermonde Matrix. Since $\boldsymbol{p}(t)$ is found using four points, the Vandermonde Matrix will be a 4×4 matrix and result in a cubic polynomial.

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$
$$p(x_0) = y_0, \ldots, p(x_m) = y_m$$
$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} p(x_0) \\ p(x_1) \\ p(x_2) \\ \vdots \\ p(x_m) \end{bmatrix}$$

The following code snippet finds the polynomial $\boldsymbol{p}(t)$ using $[(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_{i+3}, y_{i+3})]$.

```
function polynomial = interpolation(points)

A = [points(:, 1).^3, points(:, 1).^2, points(:, 1), ones(4,1)];
polynomial = A \ points(:,2);
```

The roots of $\boldsymbol{p}(t) - (H - D) = 0$ are possible solutions to the activation time of the camera. The eigenvalues of an $n \times n$ matrix A are the roots of the associated characteristic polynomial $p_A(t)$. The following code snippet finds the roots of a given polynomial by creating the matrix A and finding its eigenvalues. Importantly the constant value a_0 in $\boldsymbol{p}(t)$ must converted to $a_0 - (H - D)$ before using the code snippet to find the roots associated with the camera's activation time.

```
function roots = rootFinder(polynomial)
```

```
A = diag(ones(2,1),-1);
A(1,:) = -polynomial(2:4)./polynomial(1);
roots = eig(A);
```

Since $\boldsymbol{p}(t)$ is cubic, it has 3 roots. The root, r_c, associated with the camera's activation time is the root whose value satisfies $y_{i+1} < \boldsymbol{p}(r_c) < y_{i+2}$. The following code snippet is the supporting code used call the functions described earlier in this section and convert the found root into a result in seconds.

```
% Automated Camera System
camera = H - D;

closest_points = findClosestPoints(camera, y);
p = interpolation(closest_points);
r = rootFinder([p(1), p(2), p(3), p(4) - camera]);

result = r(isbetween(r,closest_points(1,1),closest_points(4,1)));
result_seconds = result * h;

fprintf('The camera should take a photograph at %.2f seconds after the jump.',
result_seconds)
```

For the model parameters provided, the camera should trigger at **3.34 seconds** in order to capture the image of the jumper.

4.6 – Water touch option

In this section, the feasibility of a water touch option will be examined. Defined as "... the jumper just touches the water at the bottom of the first bounce" by the proposal, a successful water touch can be defined mathematically as when the distance from the water D_w is within 1cm of the water and the maximum acceleration of the jumper is less than $2g$:

$$D_w \in [-0.01, 0.01] \cap \max(|a|) < 2g$$

Where the distance from the water D_w is defined as:

$$D_w = H - (\max(y) + \text{Jumper Height})$$

With the parameters provided, $D_w \approx 22.1$ meters, which does not satisfy the conditions for a successful water touch.

Values of L and k that allow for a successful water touch can be found using an optimisation approach. This involves simulating the jumper repeatedly and making small adjustments to the parameters until there is a successful water touch. The following code snippet adjusts the parameters L and k in each iteration. Acceleration is reduced by allowing the tension of the rope to act over a longer period.

D_w is reduced by increasing the length of the rope and reducing the decelerating force on the jumper. If D_w is negative, the length of the rope is reduced.

Magnitude is a scale factor is the degree of change and roughly corresponds with the significant figures in the final parameters found.

```
magnitude = 0.01;

    if max(abs(a)) > 2*g
        k = k - 0.1 * magnitude;
        L = L - 1 * magnitude;
```

```
elseif distance_from_water > 0
    k = k + 0.1 * magnitude;
    L = L + distance_from_water;
elseif distance_from_water < 0
    L = L + distance_from_water;
end
K = k/m;
```

The remaining supporting code for the optimisation approach can be found in the included MATLAB files. The optimisation approach found that a feasible water touch option with 9 bounces in 60 seconds uses:

$$L \approx 42.7\ m, k \approx 79.6\text{N/m}$$

5 – Conclusion

Through use of the RK4 numerical method, the team has created an idealised mathematical model of the bungee jump experience that is planned to be offered at the Story Bridge. Further analysis was performed to back the claims of the bungee jump company, simulate key performance metrics, and investigate whether key features such as an automated camera and "water touch" were viable.

The claims of ten bounces within the 60 seconds jump period were confirmed via the initial numerical solution, with 10 bounces easily counted on the displacement vs time plot. The "thrill factor" claim of a maximum acceleration "up to 2gs" was then simulated with a second order central differences approximation, with the resulting maximum value of 18.46m/s^2 falling short of the 19.82m/s^2 value to support the claim. However, the simulation was performed with an 80kg mass jumper, and the team expects that higher mass jumpers may reach a higher value of acceleration. Recommendations will be made to the client to perform additional simulations with higher mass jumpers; however, this was considered out of scope for the initial investigation.

Maximum speed or "thrill factor" was then determined by plotting the solved velocity vs time curve from the solved numerical solution, then using the MATLAB max() function to obtain a value of 20.04m/s (72.14 km/h), at 2.61 seconds. Distance travelled was determined with the key fact that displacement is the integral of velocity, with a trapezoidal integral approximation returning 292.10m over a 60 second time interval.

The automated camera system was then investigated, with the goal of finding the precise time for the camera to activate to capture the jumper passing the deck. Solutions were found by using an interpolating polynomial, with a trigger time of 3.34seconds being the ideal setting for a successful photograph

The viability of the "water touch" option was last to be investigated, whilst keeping the number of bounces close to 10 in 60 seconds and not exceeding the dangerous acceleration value of 2g (19.82m/s^2). A mathematical solution was by altering both the length "L" and spring constant "K" of the jumper rope, with an optimisation solution strategy returning an optimal value of L= 42.7m and K= 79.6N/m. These values met the criteria of not exceeding an acceleration of 2g and still maintained an acceptable 9 bounces over the 60 second period. Due to the precise nature of these adjustments, the team suggests this may not be feasible to perform, as minor miscalculations could result in unsafe conditions.

The resulting analysis has shown the effectiveness of RK4 method in providing an excellent approximation of what the real-world bungee jumper would experience. Several key performance metrics have been obtained, most of the bungee jump companies claims verified, and solution strategies have determined the viability of the key additional features. Final recommendations from the consultancy team are to perform more simulations to verify the performance metrics of jumpers with different masses, and to abandon the water touch feature. If there was to be a water touch feature, additional safety factors should be included, and analysis should be performed with each jumper's height and weight obtained prior to modifying the length and spring constant of the rope.

6 – References

BUNGEE.IT. (2022). *Bungee Jumping from The Colossus Bridge (152m) near Turin.* Retrieved from Manawa: https://www.manawa.com/en-GB/activity/italy/turin/bungee-jumping/bungee-jumping-from-the-colossus-bridge-152m-near-turin/5770?