The Maximum Height a Three-person Trampoline Can Reach for Each One

Team 499, Li Siyuan, Cao Shujun, and Zhang Yuhao, Problem B

November 5, 2023

Abstract

This paper explores the dynamics of a trampoline system when multiple individuals simultaneously jump on it. The analysis begins by modeling the trampoline's behavior, accounting for various forms of resistance, including air resistance, damping, and friction. The study initially focuses on a single-person trampoline model, highlighting the energy conversion and release process at each bounce, ultimately reaching a stable maximum height. The three-person trampoline model is then investigated, revealing that under the influence of resistance, all individuals can achieve higher maximum heights compared to the single-person scenario. We consider both realistic conditions, where energy is not continuously added, and ideal conditions, where individuals have maximum initial energy. The results indicate that individuals with smaller mass tend to reach greater maximum heights. Despite the various resistances considered, the three individuals approximately conserve mechanical energy in the steady state. The paper provides detailed models, extensive time simulations, and a comparison between ideal and realistic scenarios, offering insights into energy transmission in multiple-spring systems. However, limitations related to time increment precision are also discussed.

Key Words: multiple trampoline system, air resistance, energy transfer.

Contents

1 Introduction

1.1 Background

The system of that single object bouncing on the trampoline is simple to research and illustrate its statement period precisely and practically. However, in reality, there is more than one single person jumping on the same trampoline. We call this a multiple trampoline system, which is complex and chaotic. Any tiny change of initial velocity, mass, or time between each person can cause a huge difference in the specific movement. Furthermore, in this system, it's complex to demonstrate all the air resistance and divide adding energy to each jump and down.

1.2 Problem Restatement

We know the highest height of each three single persons jumping on the trampoline, and we want to know the highest height of each person when they jump together.

2 Assumptions and Notations

2.1 Assumptions

- each person is a rigid body
- junction of trampoline and bracket is rigid
- all the person who jump together drop at the same point of trampoline
- Neglecting the mass of the spring and the net surface
- The length of the spring can be neglected

2.2 Notations

Symbols	Description	Value	Unit
g	Gravitational acceleration	9.8	m/s^2
k	Stiffness coefficient of a single spring	96000	N/m
R	Trampoline radius	5	m
H_m	Trampoline height	1	m
N	Number of springs	60	-
S_h	The area of contact between man and space in free fall	0.08	m^2
Ca	Air resistance coefficient	2.3	N·s^2/m^4
Cd	Damping coefficient	16	N·s/m
Fc	Friction force	2.5	N

Here the main notations are defined while their specific values will be discussed and given later.

3 Physical Analysis of the Model

3.1 Modeling of the Trampoline

In this research, a round trampoline is used for the bounce analysis, as shown in Figure 1. For this trampoline, the mat is hooked and connected to the frame by springs which are evenly distributed around the frame and have initial extensions which stretch the mat. Springs hold the trampoline mat in place and absorb the displacement of the mat when a jumper impacts and pushes down the mat. The spring recoil then releases that force back into the upward movement of the mat, which sends the jumper back up with a large portion of their landing energy. The main trampoline dimensions and parameters are shown in Table 1.

A simplified model is used to illustrate the trampoline deflection under an external force F , as shown in Figure 2. Since the mat is significantly less elastic than the spring, the external force mainly causes spring extension. When the external force acts at the centre of the mat, the springs are stretched and the mat centre moves downward with a height of H . At the static state, the external force is balanced by the spring force. From

Figure 1: Trampoline used in our model

Parameter	Description	Value	Unit
k	Stiffness coefficient of a single spring	96000	N/m
R	Trampoline radius	5	m
H_m	Trampoline height	1	m
N	Number of springs	60	-

Table 1: Trampoline dimensions and parameters

the initial free state to the new static state under F, the spring deformation is:

$$\Delta L = L - y = \sqrt{R^2 + y^2} - y \tag{1}$$

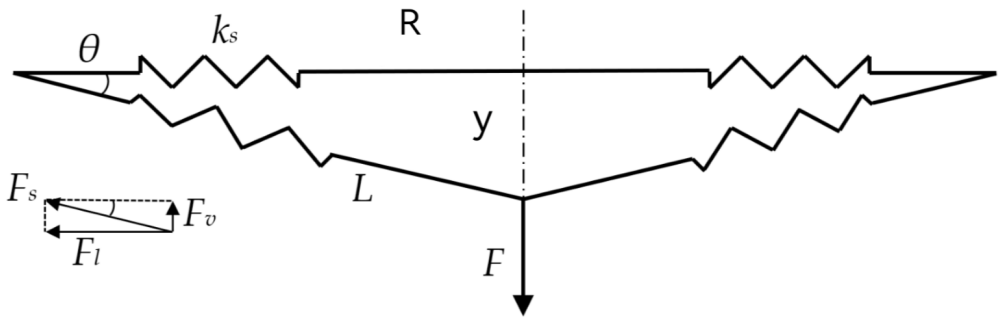


Figure 2: Sketch model of the trampoline under external force

The spring force generated by its deformation is:

$$F_s = k\Delta L = k(\sqrt{R^2 + y^2} - y) \tag{2}$$

The spring forces can be split into vertical force and lateral force. As all the springs are identical and evenly arranged around the circular trampoline frame, the lateral forces are considered to be balanced with each other. As such, only the vertical force acts on the mat. The single spring vertical force F_v is:

$$F_v = F_s \sin\theta = F_s \frac{y}{\sqrt{R^2 + y^2}} \tag{3}$$

At the static state, the combined vertical force of all the springs should be equal to the external force but with the opposite direction, and the following equation is obtained:

$$F = \sum_{i=1}^{N} F_v = Nk(\sqrt{R^2 + y^2} - y)\frac{y}{\sqrt{R^2 + y^2}} \tag{4}$$

Since $\frac{y}{R}$ is small, Expand $\frac{y}{R}$ in a Taylor series up to the quadratic term:

$$F \approx \frac{Nky^3}{2R^2} \tag{5}$$

To facilitate energy calculations later on, we calculate the trampoline's elastic potential

energy through integration:

$$\begin{aligned} E_p &= \int_0^y F dy \\ &= \int_0^y Nk(\sqrt{R^2+y^2}-y)\frac{y}{\sqrt{R^2+y^2}} \\ &= Nk(\frac{y^2}{2} - R^2(\sqrt{R^2+y^2}-R)) \\ &\approx \frac{Nky^4}{8R^2} \end{aligned} \tag{6}$$

3.2 Factors of Resistance

Parameter	Description	Value	Unit
Ca	Air resistance coefficient	2.3	N·s$^2/m^4$
Cd	Damping coefficient	16	N·s/m
Fc	Friction force	2.5	N

Table 2: Resistance parameters

During the bouncing process, individuals are subjected not only to gravity and the spring force but also to resistance. Below, we will analyze resistance based on its type. When performing free fall or vertical upward motion in the air, one experiences the influence of air resistance:

$$F_m = C_a S_h v^2 \tag{7}$$

During contact with the trampoline, the entire trampoline surface is subjected to air resistance.Assuming the trampoline centre velocity is v , the mat is divided into multiple rings,as shown in Figure 3, where the velocity of each ring is:

$$v_x = \frac{R-x}{R}v \tag{8}$$

where x is the distance from the ring to the trampoline centre.

Then, for each ring, the air resistance in the vertical direction is:

$$dF_{ax} = \frac{R}{\sqrt{R^2+y^2}} C_a (\frac{R-x}{R} v)^2 \pi x dx \tag{9}$$

where C a is the air resistance coefficient.
The total air resistance of the whole mat would be the integration of the air resistance of all rings:

$$F_a = \int_0^R dF_{ax} = \int_0^R \frac{R}{\sqrt{R^2+y^2}} C_a (\frac{R-x}{R} v)^2 \pi x dx = \frac{\pi C_a R^3 v^2}{6\sqrt{R^2+y^2}} \tag{10}$$

There is also damping resistance working on the trampoline which could be described

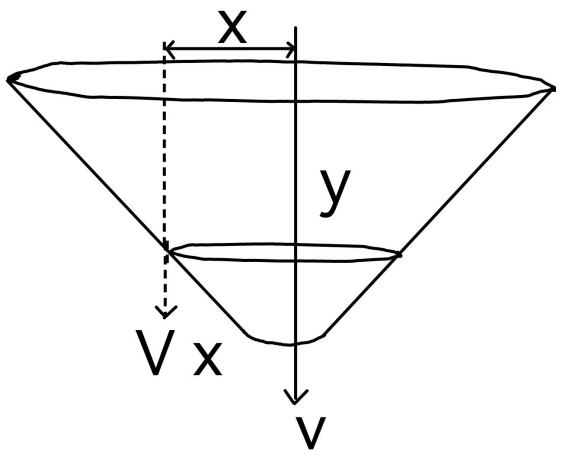


Figure 3: The mat is divided into multiple rings

by an equivalent damping coefficient multiplying the trampoline centre velocity:

$$F_d = C_d v \tag{11}$$

where C_d is the equivalent damping coefficient.
There is also resistance caused by friction between the mat, springs, and frame, which is insignificant compared to the air and damping resistances. It can be described by a constant resistance force:

$$F_c \tag{12}$$

The above constitutes our modeling of the resistance system.

3.3 Dynamical Analysis of the Single Trampoline Model

Bounce and Energy Release Model

During contact with the trampoline, individuals release energy by exerting force with their legs, causing the bounce height to increase continuously. We can simplify the energy release process as follows: after each contact with the spring reaches its lowest point, individuals adjust their posture to lower their body's center of gravity, thereby increasing the total energy of the system. (Subsequent simulations will demonstrate the validity of this assumption.) Once individuals reach a certain height, the energy released is offset by resistance, leading to a stabilization of the total system energy. We assume that the work done by individuals during each bounce is denoted as E.

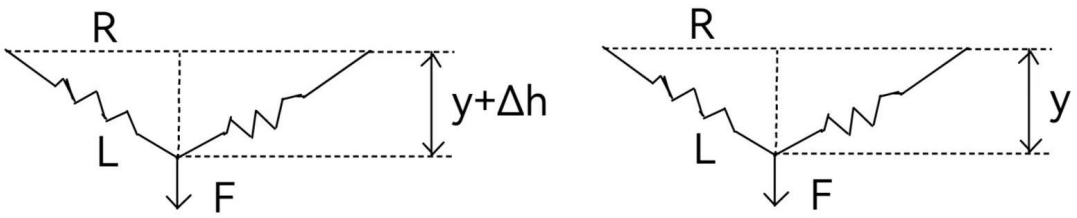


Figure 4: The way to release energy

We assume that a constant amount of energy is released by the jumper during each bounce. By combining this assumption with the elastic potential energy formula provided earlier, we can determine the distance Δh by which the center of mass should descend at the lowest point.

$$\Delta h = \sqrt[4]{\frac{8E}{Nk} + y^4} - y \tag{13}$$

Analysis of the Motion Process

The human bouncing process can be divided into two parts: free fall in the air and the upward motion combined with the action of the trampoline's springs. As the

relative horizontal height between the individual and the springs changes, these two processes interchange, resulting in a periodic motion trajectory. Free fall and upward motion($y \geq 0$):

$$Ma = -Mg - F_m sgn(v) \tag{14}$$

The process of springs acting on the trampoline($y < 0$):

$$Ma = F - Mg - (F_a + F_d + F_c)sgn(v) \tag{15}$$

Next, let's take the first child as an example and solve the displacement-time relationship of the motion using numerical values. We have $y_1(0) = 0$, $v_1(0) = 0$, and $E_1 = 0$. By simulating in Python, create a displacement-time motion graph:

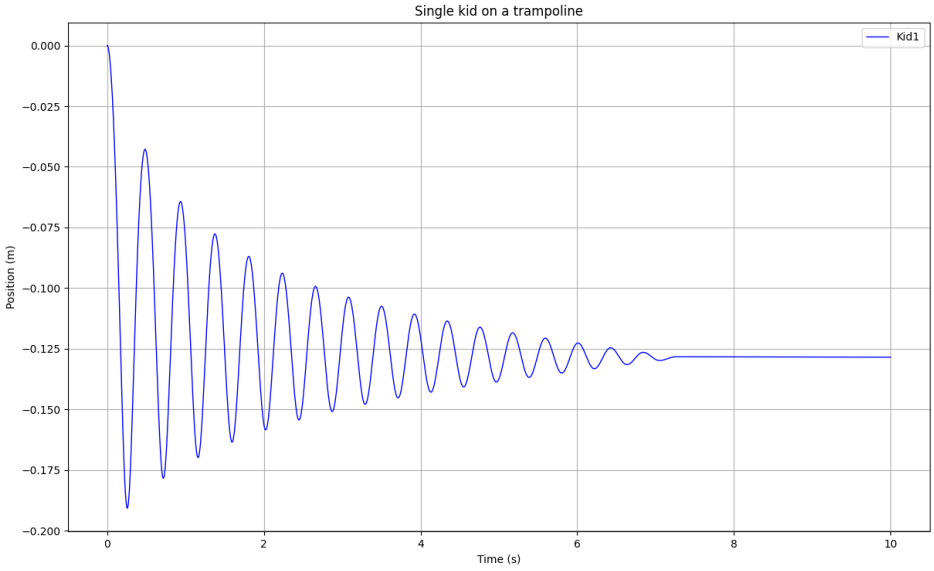
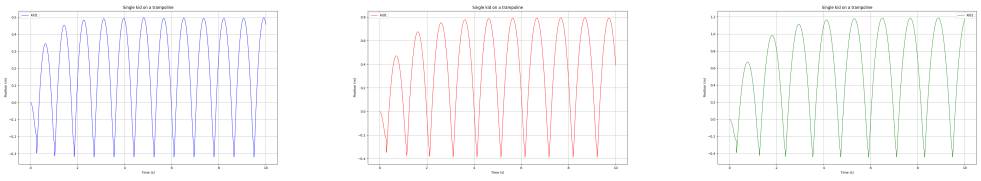


Figure 5: The displacement-time motion graph of kid1 ($E_1 = 0$)

It is evident that if the child does not do work, they will come to a stop around 6 seconds, which aligns with reality. This demonstrates the reasonableness of our resistance modeling. Next, we will set E_1 to 50J and 100J, respectively, to observe the differences

in the motion graphs.

We observed that by allowing the child to do work, the child's motion becomes periodic, and with an increase in the work done during each bounce, the maximum height attainable in the steady state becomes higher. By adjusting the value of the work done, E_1, we were able to achieve a final stable maximum height as given in the problem, which is 0.5 meters. It was found that $E_1 = 180J$. Using the same method, we measured the work done by the second and third children during each bounce, which is $E_2 = 328J$ and $E_3 = 538J$, respectively.



(a) $E_1 = 180J$ (b) $E_2 = 328J$ (c) $E_3 = 538J$

Figure 6: The displacement-time motion graph of the three kids

By observing the graphs, we find that in the single trampoline model, the total mechanical energy of the individual and the trampoline is approximately conserved in the steady state. This can be viewed as an approximation where we neglect all resistance effects and give the individual a certain initial energy, after which no energy is released.

In the subsequent analysis of the three-person system, we will compare these two models.

3.4 Dynamical Analysis of the Three-people Trampoline Model

According to the modeling in Section 3.1, we assume that the landing points of the three individuals on the trampoline are all at the center of the trampoline. This is because the area formed by the triangle they create is very small relative to the large trampoline surface and can be treated as a single point. According to Section 3.1, the

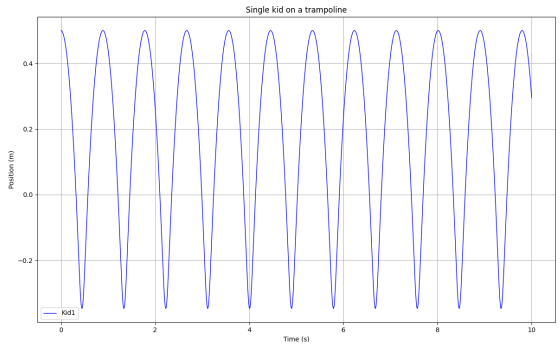


Figure 7: The ideal displacement-time motion graph of kid1

relationship between the force at this point and displacement is approximated as a cubic function.

In reality, individuals cannot collide with each other, so this model can be imagined as a spring with a lightweight board on top, and the landing points of the three individuals are different. Due to the properties of the lightweight spring and board, when two individuals simultaneously move downward and make contact with the board, the board moves at the same speed as the one with greater velocity. In this scenario, there are only two possible motions for the three individuals. One is that all three individuals are in free fall. The other is that one individual performs simple harmonic motion on the board while the other two individuals are in free fall.

To describe this process more conveniently, let's first consider the scenario with two individuals on the trampoline.

Two-person Model

Suppose one person is performing simple harmonic motion downward on the spring, and at this point, their mechanical energy is being converted into the spring's elastic potential energy. Another person, moving at a faster speed, catches up with the first person, causing the spring to move at the faster velocity. The second person begins free fall in the air. In this scenario, their roles exchange, and the person who caught up begins to convert their mechanical energy into the spring's elastic potential energy. They continue to exchange roles, converting their mechanical energy into elastic potential energy. Until

a certain moment when one person's velocity reaches zero, and they start being pushed upward by the spring, at this point, the spring converts its elastic potential energy into their mechanical energy. They will meet the other person falling from above, and the two individuals switch roles. The spring first compresses and then extends. Although we don't know if this individual's mechanical energy will increase or decrease, it is certain that the change in mechanical energy will gradually transition from negative to positive. During the ascent, the spring splits its elastic potential energy into two parts, giving each person a portion.

We use Python simulation to visually observe this process, divide time into very small segments of dt, during which the acceleration and velocity of the objects are approximately constant:

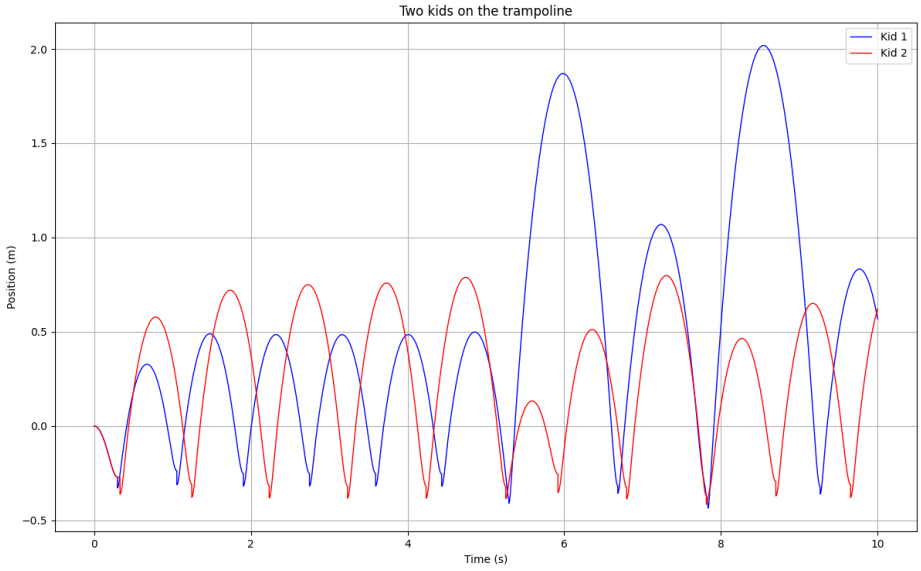


Figure 8: The displacement-time motion graph of kid1 and kid2 (time = 20s)

From this, we can see that the motion of the two kids is very complex and unpredictable. By extending the simulation time and observing the changes in their trajectories, we can roughly estimate the maximum height reached during the motion.

Three-person Model

We assume that in the initial state, all three individuals are on the trampoline surface with zero velocity, the situation with three people is essentially the same as that with two people, where the person at the bottom always interacts with the spring, while the other two people perform free fall or are in an upward motion. Consistent with the assumption in section 3.3, when a person in contact with the spring reaches zero velocity, they release their energy.We conduct MATLAB simulation method to study the situation with three people on the trampoline:

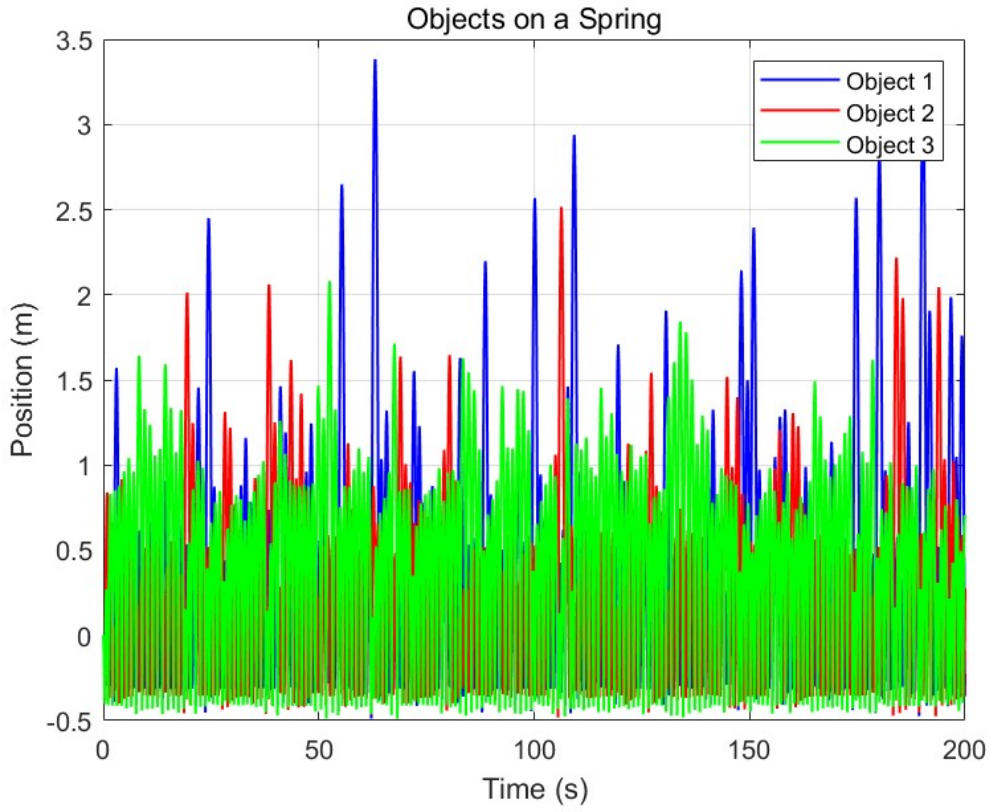


Figure 9: The real displacement-time motion graph of the three kids (time = 500s)

Now we consider a simplified case, we do not consider the effect of resistance. Instead, we assume that all three individuals have a certain amount of initial energy, and they do not add more energy thereafter. We conduct MATLAB simulation in this case:

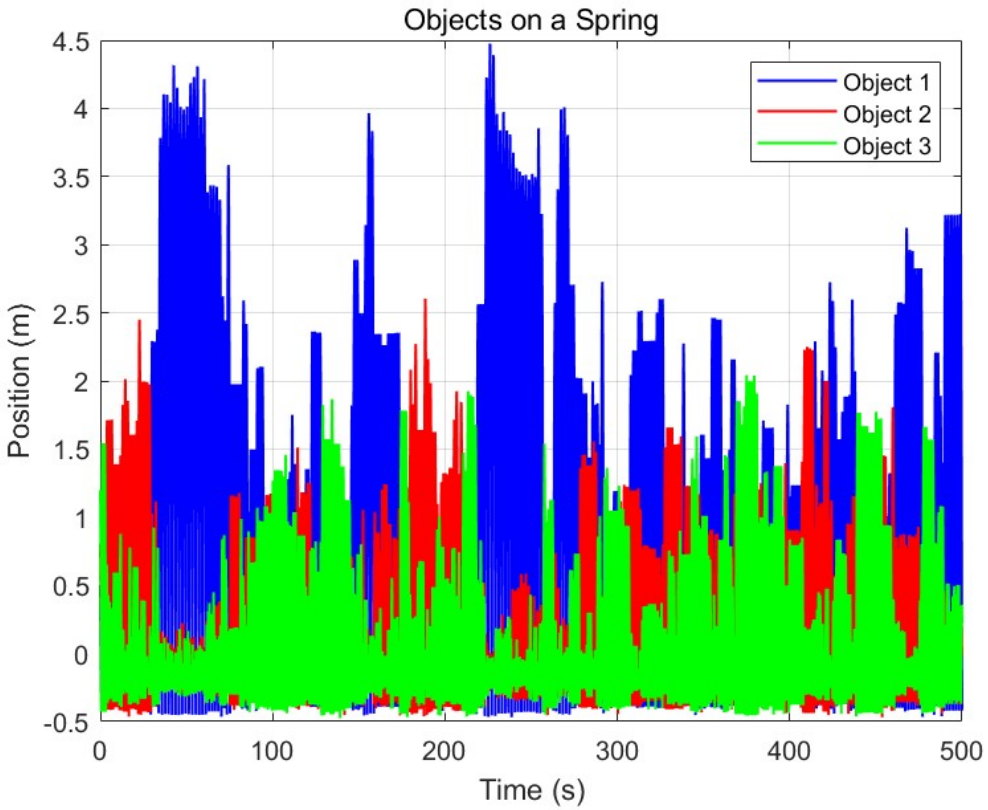


Figure 10: The ideal displacement-time motion graph of the three kids(time = 500s)

It should be noted that the principle behind our simulation is to take a small time increment dt during which the objects undergo approximately uniform acceleration in a straight-line motion. We calculate subsequent velocities and displacements based on the initial conditions, and this process is iterated. Due to limitations in computer mem-

ory, the smallest feasible dt achievable in our Python program is on the order of 10^{-6}. Even when using a more computationally capable tool like MATLAB, the best we can achieve is on the order of 10^{-7}. As a result, the simulation may not precisely match real-world scenarios. This method allows us to estimate the approximate range of maximum heights.

However, to more accurately address the initial question of determining the maximum heights of the three individuals, we rewrote a program using C++. This eliminated the image processing step, allowing us to achieve a precision of 10^{-8} for each dt. As a result, we were able to obtain the respective maximum heights for all three individuals:

$$h_{1max} = 3.388m$$
$$h_{2max} = 2.545m$$
$$h_{3max} = 2.286m$$

We also used this method to calculate the maximum height under ideal conditions (i.e., without considering various resistances and assuming that the bouncing individuals start with maximum mechanical energy):

$$h_{1max} = 4.403m$$
$$h_{2max} = 2.601m$$
$$h_{3max} = 2.092m$$

The very small discrepancy between the simulation results obtained from MATLAB and C++ confirms the validity of our model.

4 Results

4.1 Dynamical Analysis of the Single Trampoline Model

This result indicates that when a person extends their legs at the lowest point on the springboard to lower their center of gravity, they are able to convert some of their biological energy into elastic potential energy. In this way, the energy added to the system in each cycle matches the energy consumed by air resistance, resulting in stability at a maximum height. It's important to note that each person can add a different amount of energy in each cycle. The lightest individual can add 180 joules, the middle-weight individual can add 328 joules, and the heaviest individual can add 538 joules.

4.2 Dynamical Analysis of the Three-people Trampoline Model

Two-person Model

From the displacement-time graph of the first 20 seconds, we can observe that if there is no intersection between the curve of one person and the curve of another person during the segment where one person's displacement is negative, their maximum ascending height will remain constant. This is reasonable because in such a scenario, there is no opportunity for energy transfer between the two individuals through the spring, and as a result, the mechanical energy of each individual and the spring remains conserved separately.

However, if there is an approximate overlap in the curves of the two individuals near the zero potential surface, it indicates that they are continuously exchanging roles, alternating between simple harmonic and free-fall motions. In the absence of role exchange, the difference in the displacements of the two individuals should exhibit significant variations, rather than approximate overlap.

After both individuals reascend to the zero potential surface, their mechanical energy is redistributed. This is consistent with the observation of one person achieving a greater height while the other person's height decreases in the graph. These results support the validity of our explanation for the two-person trampoline motion.

Three-person ideal Model

This model simulates the most natural process, without considering air resistance, and initially injects the maximum energy for each of the three individuals separately. The maximum heights achieved by them in this model are also quite close to our expectations. In the system composed of three people and springs, mechanical energy is conserved. When one person reaches maximum mechanical energy, the mechanical energy of the other two people is zero. This allows us to conclude that:

$$M_i h_{imax} = M_1 h_1 + M_2 h_2 + M_3 h_3 \tag{16}$$

The calculated results are:

$$h_{1max} = 4.18m$$
$$h_{2max} = 2.61m$$
$$h_{3max} = 2.09m$$

which are very close to the data we obtained.

Three-person real Model

In real-world data, when compared to the data obtained in the ideal scenario, the differences in the maximum heights of the three individuals are smaller and more tightly packed. This could be attributed to the different ways in which energy is transferred, and these differences in energy transfer are primarily caused by the presence of resistance. We speculate that the presence of resistance makes the interaction between the three individuals more tightly connected.

Similarly to the ideal model, in the real-world data, we observe that individuals with smaller mass achieve greater maximum heights. This is in line with the concept that, for a given energy, objects with smaller mass reach higher heights when they have larger initial velocities.

5 Model Evaluation

5.1 Strengths

1. Simulate as realistic: consider the air resistance of the elastic surface and human body.

2. As for this multiple chaotic trampoline system, we stimulate it over a long time period to guarantee all the motion can be considered.

3. We module the trampoline itself precisely by theory.

4. We start from one person to two persons and three persons, so that the predictive equations and conclusion could assist further research for complex multiple spring systems, which as a way to energy transmit.

5. Use kinds of modules from ideal to realistic for stimulating the problem.

5.2 Weaknesses

1. Because the module software such as Matlab can't integral, we just decrease the dt adequately. however, different level values of dt influence the result violently, so there exists some error.

2. We simplify the adding energy process much more. Humans can't add the same energy each time which we research like.

3. We regard the human body as a rigid body which is not accurate. Humans in reality should be considered as a more complex mechanic system.

6 Conclusion

We began by modeling the trampoline as two springs and considered various types of air resistance. Starting with the single-person trampoline model, we found that under the influence of various resistances, individuals can achieve and maintain a maximum height by releasing a certain amount of energy at each low point. Furthermore, in

the three-person spring model, all three individuals can attain higher maximum heights compared to single-person bouncing, and they approximately adhere to the law of energy conservation. In scenarios where we considered various resistances and individuals not adding extra energy and scenarios where we neglected air resistance and set initial energy, the three individuals achieved higher maximum heights. However, the differences in the maximum heights between the three individuals were smaller than in the ideal scenario, and we suspect that this is due to the influence of resistance on energy transfer methods.

In any case, we obtained the maximum heights achievable by three individuals in real-world conditions and observed that individuals with smaller mass reach higher maximum heights. Finally, our model has several advantages, such as detailed consideration of resistance and a wide time span. However, there are also some limitations, including the significant impact of the time increment dt on experimental results.

References

[1] Eager, D.; Zhou, S.; Ishac, K.; Hossain, I.; Richards, A.; Sharwood, L.N. Investigation into the Trampoline Dynamic Characteristics and Analysis of Double Bounce Vibrations. Sensors 2022, 22, 2916. https://doi.org/10.3390/s22082916

[2] Brown, O.H., Mullineaux, D.R. Mulloy, F. Dynamic testing to determine and predict trampoline function. Sports Eng 24, 13 (2021). https://doi.org/10.1007/s12283-021-00348-z

A single_person_real.py

```
import numpy as np
import matplotlib.pyplot as plt

# Define constants and parameters
g = 9.8  # Acceleration due to gravity
Pi = 3.14
k = 96000  # Spring constant
R = 5  # Radius of the trampoline
N = 60  # Number of springs
Sh = 0.08  # Kid's cross-sectional area
Ca = 2.3  # Air resistance coefficient
Cd = 16  # Drag coefficient
Fc = 2.5  # Coefficient of friction
E1 = 0  # Initial energy
m1 = 25  # Mass of the kid
t0 = 0.0  # Initial time
tf = 10.0  # Final time
dt = 0.001  # Time step

# Initialize arrays
t = np.arange(t0, tf + dt, dt)
n = len(t)
y1 = np.zeros(n)  # Position of the kid
v1 = np.zeros(n)  # Velocity of the kid

# Initial conditions
y1[0] = 0  # Initial position of the kid
v1[0] = 0  # Initial velocity of the kid

# Numerical solution
for i in range(n - 1):
    if y1[i] >= 0:
        # Free fall of the kid
        Fm = Ca * Sh * (v1[i] ** 2) * np.sign(v1[i])
        a1 = -g - Fm / m1
    elif y1[i] <= 0:
        # Kid contacts the trampoline
```

```
            F = -N * k * (np.sqrt(R ** 2 + y1[i] ** 2) - R) * y1[i] / np.sqrt(R ** 2 + y1[i] ** 2)  # Spring force
            Fa = Pi * Ca * R ** 3 * v1[i] * np.abs(v1[i]) / (6 * np.sqrt(R ** 2 + y1[i] ** 2))
            Fd = -Cd * v1[i] * np.sign(v1[i])
            Fc = 2.5 * np.sign(v1[i])
            a1 = -g + (F - Fa - Fd - Fc) / m1

        v1[i + 1] = v1[i] + a1 * dt  # Update the kid's velocity
        if v1[i] <= 0 and v1[i + 1] >= 0:
            y1[i + 1] = y1[i] + v1[i + 1] * dt - y1[i] - ((E1 + 28800 * y1[i] ** 4) ** 0.25) / (2 * np.sqrt(2 * np.sqrt(2)) * np.sqrt(15)) # Convert added energy E into downward displacement h
        else:
            y1[i + 1] = y1[i] + v1[i + 1] * dt

    # Plot the results
    plt.figure()
    plt.plot(t, y1, 'b-', linewidth=1)
    plt.xlabel('Time (s)')
    plt.ylabel('Position (m)')
    plt.title('Single kid on a trampoline')
    plt.legend(['Kid1'])
    plt.grid(True)
    plt.show()

```

B three_people_real.m

```
% Define constants and parameters
g = 9.8;          % Gravitational acceleration
k = 96000;        % Spring constant
m1 = 25;          % Mass of Object 1
m2 = 40;          % Mass of Object 2
m3 = 50;          % Mass of Object 3
t0 = 0.0;         % Initial time
tf = 200.0;       % Final time
dt = 0.0000005;   % Time step
```

```
    R = 5;              % Trampoline radius
    N = 60;             % Number of springs
    Sh = 0.08;          % Cross-sectional area of contact with the
trampoline
    Ca = 2.3;           % Air resistance coefficient
    Cd = 16;            % Damping coefficient
    Fc = 2.5;           % Frictional force
    Pi = 3.14;
    E1 = 180;           % Energy released by Object 1 each time
    E2 = 328;           % Energy released by Object 2 each time
    E3 = 538;           % Energy released by Object 3 each time

    % Initialize arrays
    t = t0:dt:tf;
    n = length(t);
    y1 = zeros(1, n);  % Position of Object 1
    y2 = zeros(1, n);  % Position of Object 2
    y3 = zeros(1, n);  % Position of Object 3
    v1 = zeros(1, n);  % Velocity of Object 1
    v2 = zeros(1, n);  % Velocity of Object 2
    v3 = zeros(1, n);  % Velocity of Object 3

    % Initial conditions
    y1(1) = 0;  % Initial position of Object 1
    y2(1) = 0;  % Initial position of Object 2
    y3(1) = 0;  % Initial position of Object 3
    v1(1) = 0;  % Initial velocity of Object 1
    v2(1) = 0;  % Initial velocity of Object 2
    v3(1) = 0;  % Initial velocity of Object 3

    % Numerical solution
    for i = 1:n-1
        if y1(i) >= y2(i) && y3(i) >= y2(i) && y2(i) <= 0
            % Object 2 contacts the springs, and Objects 1 and 3 are
in free fall
            Ft = -N*k*(sqrt(R*R+y2(i)*y2(i))-R)*y2(i)/sqrt(R*R+y2(i)*
y2(i));     % Spring force
            Fa = Pi*Ca*R*R*R*v2(i)*v2(i)*sign(v2(i))/(sqrt(R*R+y2(i)*
y2(i))*6);  % Air resistance to the trampoline
```

```
45            Fd = -Cd*v2(i)*sign(v2(i));  % Damping force
46            Fc = 2.5*sign(v2(i));        % Trampoline friction
47            Fm1 = - Ca*Sh*(v1(i)*v1(i))*sign(v1(i)); % Air resistance
     to Object 1
48            Fm3 = - Ca*Sh*(v3(i)*v3(i))*sign(v3(i)); % Air resistance
     to Object 3
49            a2 = (Ft-Fa-Fd-Fc - m2 * g) / m2;       % Acceleration of
     Object 2
50            a1 = -g + Fm1/m1;                       % Acceleration of
    Object 1
51            a3 = -g + Fm3/m3;                       % Acceleration of
    Object 3
52        elseif y2(i) >= y1(i) && y3(i) >= y1(i) && y1(i) <= 0
53            % Object 1 contacts the springs, and Objects 2 and 3 are
    in free fall
54            Ft = -N*k*(sqrt(R*R+y1(i)*y1(i))-R)*y1(i)/sqrt(R*R+y1(i)*
    y1(i));
55            Fa = Pi*Ca*R*R*R*v1(i)*v1(i)*sign(v1(i))/(sqrt(R*R+y1(i)*
    y1(i))*6);
56            Fd = -Cd*v1(i)*sign(v1(i));
57            Fc = 2.5*sign(v1(i));
58            Fm2 = - Ca*Sh*(v2(i)*v2(i))*sign(v2(i));
59            Fm3 = - Ca*Sh*(v3(i)*v3(i))*sign(v3(i));
60            a1 = (Ft-Fa-Fd-Fc - m1 * g) / m1;
61            a3 = -g + Fm3/m3;
62            a2 = -g + Fm2/m2;
63        elseif y2(i) >= y3(i) && y1(i) >= y3(i) && y3(i) <= 0
64            % Object 3 contacts the springs, and Objects 1 and 2 are
    in free fall
65            Ft = -N*k*(sqrt(R*R+y3(i)*y3(i))-R)*y3(i)/sqrt(R*R+y3(i)*
    y3(i));
66            Fa = Pi*Ca*R*R*R*v3(i)*v3(i)*sign(v3(i))/(sqrt(R*R+y3(i)*
    y3(i))*6);
67            Fd = -Cd*v3(i)*sign(v3(i));
68            Fc = 2.5*sign(v3(i));
69            Fm2 = - Ca*Sh*(v2(i)*v2(i))*sign(v2(i));
70            Fm1 = - Ca*Sh*(v1(i)*v1(i))*sign(v1(i));
71            a2 = -g + Fm2/m2;
72            a1 = -g + Fm1/m1;
```

```
            a3 = (Ft-Fa-Fd-Fc - m3 * g) / m3;
        else
            a1 = -g;
            a2 = -g;
            a3 = -g;
        end

        v1(i + 1) = v1(i) + a1 * dt;  % Update velocity of Object 1
        if v1(i) <= 0 && v1(i+1) >= 0 && y2(i) >= y1(i) && y3(i) >=
    y1(i)
            y1(i + 1) = y1(i) + (v1(i)+v1(i+1))/2 * dt - y1(i) - ((E1
    +28800*y1(i)^4)^0.25)/(2*sqrt(2*sqrt(2))*sqrt(15)); % Update
    position of Object 1
        else
            y1(i + 1) = y1(i) + (v1(i)+v1(i+1))/2 * dt; % Update
    position of Object 1
        end

        v2(i + 1) = v2(i) + a2 * dt;  % Update velocity of Object 2
        if v2(i) <= 0 && v2(i+1) >= 0 && y1(i) >= y2(i) && y3(i) >=
    y2(i)
            y2(i + 1) = y2(i) + (v2(i)+v2(i+1))/2 * dt - y2(i) - ((E2
    +28800*y2(i)^4)^0.25)/(2*sqrt(2*sqrt(2))*sqrt(15)); % Update
    position of Object 2
        else
            y2(i + 1) = y2(i) + (v2(i)+v2(i+1))/2 * dt; % Update
    position of Object 2
        end

        v3(i + 1) = v3(i) + a3 * dt;  % Update velocity of Object 3
        if v3(i) <= 0 && v3(i+1) >= 0 && y2(i) >= y3(i) && y1(i) >=
    y3(i)
            y3(i + 1) = y3(i) + (v3(i)+v3(i+1))/2 * dt - y3(i) - ((E3
    +28800*y3(i)^4)^0.25)/(2*sqrt(2*sqrt(2))*sqrt(15)); % Update
    position of Object 3
        else
            y3(i + 1) = y3(i) + (v3(i)+v3(i+1))/2 * dt; % Update
    position of Object 3
        end
```

```
    end

    % Plot the results
    figure;
    plot(t, y1, 'b-', 'LineWidth', 1);
    hold on;
    plot(t, y2, 'r-', 'LineWidth', 1);
    hold on;
    plot(t, y3, 'g-', 'LineWidth', 1);
    xlabel('Time (s)');
    ylabel('Position (m)');
    title('Objects on a Spring');
    legend('Object 1', 'Object 2', 'Object 3');
    grid on;


```

C three_people_ideal.m

```
% Define constants and parameters
g = 9.8;          % Gravitational acceleration
m1 = 25;          % Mass of Object 1
m2 = 40;          % Mass of Object 2
m3 = 50;          % Mass of Object 3
t0 = 0.0;         % Initial time
tf = 800.0;       % End time
dt = 0.0000015;   % Time step
N = 60;           % Number of springs
k = 96000;        % Spring constant for each spring
R = 5;            % Radius of the trampoline

% Initialize arrays
t = t0:dt:tf;
n = length(t);
y1 = zeros(1, n);  % Position of Object 1
y2 = zeros(1, n);  % Position of Object 2
y3 = zeros(1, n);  % Position of Object 3
v1 = zeros(1, n);  % Velocity of Object 1
```

```
    v2 = zeros(1, n);  % Velocity of Object 2
    v3 = zeros(1, n);  % Velocity of Object 3

    % Initial conditions
    y1(1) = 0.5;  % Initial position of Object 1
    y2(1) = 0.8;  % Initial position of Object 2
    y3(1) = 1.2;  % Initial position of Object 3
    v1(1) = 0;    % Initial velocity of Object 1
    v2(1) = 0;    % Initial velocity of Object 2
    v3(1) = 0;    % Initial velocity of Object 3

    % Numerical solution
    for i = 1:n-1
        if y1(i) >= y2(i) && y3(i) >= y2(i) && y2(i) <= 0
            % Object 2 contacts the springs, and Objects 1 and 3 are
    in free fall
            F2 = -N*k*(sqrt(R*R+y2(i)*y2(i))-R)*y2(i)/sqrt(R*R+y2(i)*
    y2(i));    % Spring force
            a2 = (F2  - m2 * g) / m2;       % Acceleration of Object
    2
            a1 = -g;                        % Acceleration of Object 1
            a3 = -g;                        % Acceleration of Object 3
        elseif y2(i) >= y1(i) && y3(i) >= y1(i) && y1(i) <= 0
            % Object 1 contacts the springs, and Objects 2 and 3 are
    in free fall
            F1 = -N*k*(sqrt(R*R+y1(i)*y1(i))-R)*y1(i)/sqrt(R*R+y1(i)*
    y1(i));     % Spring force
            a2 = -g;                        % Acceleration of Object 2
            a1 = (F1 - m1 * g) / m1;        % Acceleration of Object 1
            a3 = -g;                        % Acceleration of Object 3
        elseif y2(i) >= y3(i) && y1(i) >= y3(i) && y3(i) <= 0
            % Object 3 contacts the springs, and Objects 1 and 2 are
    in free fall
            F3 = -N*k*(sqrt(R*R+y3(i)*y3(i))-R)*y3(i)/sqrt(R*R+y3(i)*
    y3(i));    % Spring force
            a2 = -g;                        % Acceleration of Object 2
            a1 = -g;                        % Acceleration of Object 1
            a3 = (F3 - m3 * g) / m3;        % Acceleration of Object 3
        else
```

```
            a1 = -g;  % Acceleration of Object 1
            a2 = -g;  % Acceleration of Object 2
            a3 = -g;  % Acceleration of Object 3
        end

        v1(i + 1) = v1(i) + a1 * dt;        % Update velocity of
    Object 1
        v2(i + 1) = v2(i) + a2 * dt;        % Update velocity of
    Object 2
        v3(i + 1) = v3(i) + a3 * dt;        % Update velocity of
    Object 3
        y1(i + 1) = y1(i) + (v1(i + 1)+v1(i))/2 * dt; % Update
    position of Object 1
        y2(i + 1) = y2(i) + (v2(i + 1)+v2(i))/2 * dt; % Update
    position of Object 2
        y3(i + 1) = y3(i) + (v3(i + 1)+v3(i))/2 * dt; % Update
    position of Object 3
    end

    % Plot the results
    figure;
    plot(t, y1, 'b-', 'LineWidth', 1);
    hold on;
    plot(t, y2, 'r-', 'LineWidth', 1);
    hold on;
    plot(t, y3, 'g-', 'LineWidth', 1);
    xlabel('Time (s)');
    ylabel('Position (m)');
    title('Objects on a Spring');
    legend('Object 1', 'Object 2','Object 3');
    grid on;
```

D three_people_real.cpp

```
#include <iostream>
```

```
    #include <cmath>

using namespace std;

int main() {
    // Define constants and parameters
    double g = 9.8;       // Gravitational acceleration
    double k = 96000;     // Spring stiffness coefficient
    double m1 = 25;       // Mass of object 1
    double m2 = 40;       // Mass of object 2
    double m3 = 50;       // Mass of object 3
    double t0 = 0.0;      // Initial time
    double tf = 500.0;    // End time
    double dt = 0.000000005; // Time step
    double R = 5;         // Trampoline radius
    double N = 60;        // Number of springs
    double Sh = 0.08;     // Area of contact between man and space in
    free fall
    double Ca = 2.3;      // Air resistance coefficient
    double Cd = 16;       // Damping coefficient
    double Fc = 2.5;      // Friction force
    double Pi = 3.14;     // Pi
    double E1 = 180;      // Energy released by object 1 each time
    double E2 = 328;      // Energy released by object 2 each time
    double E3 = 538;      // Energy released by object 3 each time
    double v1_new, v2_new, v3_new;
    double a1, a2, a3;
    double Ft, Fa, Fd, Fm1, Fm2, Fm3;
    double y1max = -1, y2max = -1, y3max = -1;

    // Initialize variables
    double t = t0;
    double y1 = 0, y2 = 0, y3 = 0;  // Initial Position
    double v1 = 0, v2 = 0, v3 = 0;  // Initial velocity

    // Numerical solution
    while (t <= tf) {
        // Object 2 contacts the spring, object 1 and 3 in free fall
        if (y1 >= y2 && y3 >= y2 && y2 <= 0) {
```

```
40            Ft = -N * k * (sqrt(R * R + y2 * y2) - R) * y2 / sqrt(R *
    R + y2 * y2);      // Spring force
41            Fa = Pi * Ca * R * R * R * v2 * v2 * copysign(1.0, v2) /
   (sqrt(R * R + y2 * y2) * 6);  // Air resistance to Trampoline
42            Fd = -Cd * v2 * copysign(1.0, v2);  // Damping Force
43            Fc = 2.5 * copysign(1.0, v2);   // Friction of Trampoline
44            Fm1 = -Ca * Sh * (v1 * v1) * copysign(1.0, v1);   // Air
   resistance to m1
45            Fm3 = -Ca * Sh * (v3 * v3) * copysign(1.0, v3);    // Air
    resistance to m3
46            a2 = (Ft - Fa - Fd - Fc - m2 * g) / m2;  // Acceleration
   of object 2
47            a1 = -g + Fm1 / m1;  // Acceleration of object 1
48            a3 = -g + Fm3 / m3;  // Acceleration of object 3
49        }
50        // Object 1 contacts the spring, object 2 and 3 in free fall
51        else if (y2 >= y1 && y3 >= y1 && y1 <= 0) {
52            Ft = -N * k * (sqrt(R * R + y1 * y1) - R) * y1 / sqrt(R *
    R + y1 * y1);
53            Fa = Pi * Ca * R * R * R * v1 * v1 * copysign(1.0, v1) /
   (sqrt(R * R + y1 * y1) * 6);
54            Fd = -Cd * v1 * copysign(1.0, v1);
55            Fc = 2.5 * copysign(1.0, v1);
56            Fm2 = -Ca * Sh * (v2 * v2) * copysign(1.0, v2);
57            Fm3 = -Ca * Sh * (v3 * v3) * copysign(1.0, v3);
58            a1 = (Ft - Fa - Fd - Fc - m1 * g) / m1;
59            a3 = -g + Fm3 / m3;
60            a2 = -g + Fm2 / m2;
61        }
62        // Object 3 contacts the spring, object 1 and 2 in free fall
63        else if (y2 >= y3 && y1 >= y3 && y3 <= 0) {
64            Ft = -N * k * (sqrt(R * R + y3 * y3) - R) * y3 / sqrt(R *
    R + y3 * y3);
65            Fa = Pi * Ca * R * R * R * v3 * v3 * copysign(1.0, v3) /
   (sqrt(R * R + y3 * y3) * 6);
66            Fd = -Cd * v3 * copysign(1.0, v3);
67            Fc = 2.5 * copysign(1.0, v3);
68            Fm2 = -Ca * Sh * (v2 * v2) * copysign(1.0, v2);
69            Fm1 = -Ca * Sh * (v1 * v1) * copysign(1.0, v1);
```

```
            a2 = -g + Fm2 / m2;
            a1 = -g + Fm1 / m1;
            a3 = (Ft - Fa - Fd - Fc - m3 * g) / m3;
        }
        // All three objects in free fall
        else {
            a1 = -g;
            a2 = -g;
            a3 = -g;
        }
        v1_new = v1 + a1 * dt;
        v2_new = v2 + a2 * dt;
        v3_new = v3 + a3 * dt;

        // Update position
        if (v1 <= 0 && v1 + v1_new >= 0 && y2 >= y1 && y3 >= y1) {
            y1 = y1 + ((v1 + v1_new) / 2) * dt - y1 - pow((E1 + 28800
 * pow(y1, 4)), 0.25) / (2 * sqrt(2 * sqrt(2)) * sqrt(15)); //
When reaching the lowest point, the center of the person will
actively descend
        } else {
            y1 = y1 + ((v1 + v1_new) / 2) * dt;
        }

        if (v2 <= 0 && v2 + v2_new >= 0 && y1 >= y2 && y3 >= y2) {
            y2 = y2 + ((v2 + v2_new) / 2) * dt - y2 - pow((E2 + 28800
 * pow(y2, 4)), 0.25) / (2 * sqrt(2 * sqrt(2)) * sqrt(15)); //
When reaching the lowest point, the center of the person will
actively descend
        } else {
            y2 = y2 + ((v2 + v2_new) / 2) * dt;
        }

        if (v3 <= 0 && v3 + v3_new >= 0 && y2 >= y3 && y1 >= y3) {
            y3 = y3 + ((v3 + v3_new) / 2) * dt - y3 - pow((E3 + 28800
 * pow(y3, 4)), 0.25) / (2 * sqrt(2 * sqrt(2)) * sqrt(15)); //
When reaching the lowest point, the center of the person will
actively descend
        } else {
```

```
            y3 = y3 + ((v3 + v3_new) / 2) * dt;
        }
        // Update velocity
        v1 = v1_new;
        v2 = v2_new;
        v3 = v3_new;
        if (y1 > y1max)
            y1max = y1;
        if (y2 > y2max)
            y2max = y2;
        if (y3 > y3max)
            y3max = y3;
        // Update time
        t += dt;
    }

    // Output results
    cout << "Maximum height of object 1: " << y1max << "m" << endl;
    cout << "Maximum height of object 2: " << y2max << "m" << endl;
    cout << "Maximum height of object 3: " << y3max << "m" << endl;

    return 0;
}
```

E three_people_ideal.cpp

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {
    // Define constants and parameters
    double g = 9.8;      // Gravitational acceleration
    double k = 96000;     // Spring stiffness coefficient
    double m1 = 25;       // Mass of object 1
```

```
        double m2 = 40;        // Mass of object 2
        double m3 = 50;        // Mass of object 3
        double t0 = 0.0;       // Initial time
        double tf = 500.0;     // End time
        double dt = 0.00000001;    // Time step
        double R = 5;          // Trampoline radius
        double N = 60;         // Number of springs
        double v1_new, v2_new, v3_new;
        double a1, a2, a3;
        double Ft;
        double y1max = -1, y2max = -1, y3max = -1;

        // Initialize variables
        double t = t0;
        double y1 = 0.5, y2 = 0.8, y3 = 1.2;  // Initial positions
        double v1 = 0, v2 = 0, v3 = 0;        // Initial velocities

        // Numerical solution
        while (t <= tf) {
            // Object 2 contacts the spring, objects 1 and 3 are in free fall
            if (y1 >= y2 && y3 >= y2 && y2 <= 0) {
                Ft = -N * k * (sqrt(R * R + y2 * y2) - R) * y2 / sqrt(R * R + y2 * y2);     // Spring force

                a2 = (Ft - m2 * g) / m2;  // Acceleration of object 2
                a1 = -g;                  // Acceleration of object 1
                a3 = -g;                  // Acceleration of object 3

            }
            // Object 1 contacts the spring, objects 2 and 3 are in free fall
            else if (y2 >= y1 && y3 >= y1 && y1 <= 0) {
                Ft = -N * k * (sqrt(R * R + y1 * y1) - R) * y1 / sqrt(R * R + y1 * y1);     // Spring force

                a1 = (Ft - m1 * g) / m1;  // Acceleration of object 1
                a3 = -g;                  // Acceleration of object 3
                a2 = -g;                  // Acceleration of object 2
```

```
46
47              }
48              // Object 3 contacts the spring, objects 1 and 2 are in
    free fall
49              else if (y2 >= y3 and y1 >= y3 and y3 <= 0) {
50                  Ft = -N * k * (sqrt(R * R + y3 * y3) - R) * y3 / sqrt
    (R * R + y3 * y3);      // Spring force
51
52                  a2 = -g;                    // Acceleration of object 2
53                  a1 = -g;                    // Acceleration of object 1
54                  a3 = (Ft - m3 * g) / m3;    // Acceleration of object 3
55
56              }
57              // All three objects are in free fall
58              else {
59                  a1 = -g;  // Acceleration of object 1
60                  a2 = -g;  // Acceleration of object 2
61                  a3 = -g;  // Acceleration of object 3
62              }
63              v1_new = v1 + a1 * dt;
64              v2_new = v2 + a2 * dt;
65              v3_new = v3 + a3 * dt;
66
67              // Update positions
68              y1 = y1 + ((v1 + v1_new) / 2) * dt;
69              y2 = y2 + ((v2 + v2_new) / 2) * dt;
70              y3 = y3 + ((v3 + v3_new) / 2) * dt;
71
72              // Update velocities
73              v1 = v1_new;
74              v2 = v2_new;
75              v3 = v3_new;
76
77              // Update maximum heights
78              if (y1 > y1max)
79                  y1max = y1;
80              if (y2 > y2max)
81                  y2max = y2;
82              if (y3 > y3max)
```

```
83                  y3max = y3;
84
85              // Update time
86              t += dt;
87          }
88
89          // Output results
90          cout << "Maximum height of object 1: " << y1max << "m" <<
    endl;
91          cout << "Maximum height of object 2: " << y2max << "m" <<
    endl;
92          cout << "Maximum height of object 3: " << y3max << "m" <<
    endl;
93
94          return 0;
95      }
96
97
98
```