The analysis of three people bouncing on a trampoline problem.

Team 471

Problem B: Trampoline Jumping

Abstract

In this paper we have found a solution to a problem regarding three people jumping on a trampoline. We have created a theoretical model using which we have found approximations of the parameters required for each person to reach their maximum height jump. Afterwards we have created a numerical simulation of the problem which we used to get an even better approximation of the parameters. To analyse the three body on the trampoline problem we have used these parameters and we have found the optimal strategy of jumping in order to launch one person as high as it is possible. Simulating three people jumping on the trampoline according to our strategy, we have found out that the person of mass 25 kg could jump as high as 3.2 m, around 2.7 meters higher than by themselves. For the person of mass 40 kg it was 2.74 m, around 2 meters higher than by themselves and accordingly for the heaviest person – 50 kg – it was 2.39 m, the lowest height of them all despite being the best jumper on their own.

1 Introduction

It is a well known fact, that when multiple people jump on a trampoline at the same time, the maximal heights they can obtain are much greater than when jumping alone. The aim of this paper is to analyse the problem of three people bouncing on a round trampoline, and find the maximal heights each of them can obtain, knowing that their masses are 25, 40 and 50 kg, and when jumping alone they can reach maximally 0.5, 0.8 and 1.2 m respectively.

2 Trampoline model of one player

It is necessary to build a working model of a trampoline. We will assume it is cylindrically symmetric. The optimal place for bouncing in order to achieve maximal height is the center of the trampoline. It is the case, because when jumping on a stiff surface (like ground), we don't bounce on it, but rather cushion the fall with our legs, and then jump. On an elastic and highly deformable surface however, we don't have to cushion the fall, and we can add energy to the already present in the system. So as the trampoline's center is the most deformable, and the further we get from it the stiffer the trampoline's surface gets on impact, it is then the center that has the smallest energy losses due to this cushioning. This effect won't be analysed in our models separately, but the fact that we model bouncing in the trampoline's center is the effect of it.

We will model the player as a material point at the middle of the trampoline. Forces acting on the player are due to gravity and loaded springs. The second of these forces is approximated by

$$F_t(z) = Nkz\left(1 - \frac{R}{\sqrt{R^2 + z^2}}\right), \tag{1}$$

where N is the number of strings, k is their spring constant (we assume that the springs are identical), R is the radius of the trampoline, and z is the coordinate of the jumping person (the z axis is directed upwards). We set that $z = 0$ at the center of an undeformed surface.
Due to air resistance, a drag force acts on the surface of the trampoline:

$$F_a(z, \dot{z}) = c_a \frac{\pi}{6} \frac{R^2}{\sqrt{1 + \frac{z^2}{R^2}}} \dot{z}^2 sgn(\dot{z}), \tag{2}$$

where c_a is the air resistance coefficient. There is also a damping force, originating from hysteresis and heating of springs. We will assume that it is linearly proportional to the velocity:

$$F_d(z, \dot{z}) = c_d \dot{z}. \tag{3}$$

This derivation of forces was inspired by [1]. Together those equations give the equation of motion for the jumping person:

$$(m + \frac{1}{3}M)\ddot{z} = -F_t(z) - F_a(z, \dot{z}) - F_d(z, \dot{z}),$$

what can be rewritten as:

$$(m + \frac{1}{3}M)\ddot{z} = -Nkz\left(1 - \frac{R}{\sqrt{R^2 + z^2}}\right) - c_a \frac{\pi}{6} \frac{R^2}{\sqrt{1 + \frac{z^2}{R^2}}} \dot{z}^2 sgn(\dot{z}) - c_d \dot{z}. \tag{4}$$

The coefficient before the mat's mass M is obtained by integration over the whole mat, as follows:

We know that for central stretching the velocity of a point in distance r from the mat's center is

$$v(r) = v_0\left(1 - \frac{r}{R}\right), \tag{5}$$

where v_0 is the velocity of the mat's center. Thus we can write the acceleration as $a(r) = a_0\left(1 - \frac{r}{R}\right)$ where a_0 is the acceleration of the center. Using that we can write the mat's equation of motion:

$$\int_S \rho a(r) = 2\pi\rho\ddot{z}\int_0^R r(1-\frac{r}{R})dr = \frac{1}{3}\pi\rho R^2\ddot{z} = \frac{1}{3}M\ddot{z}. \tag{6}$$

We neglect the fact, that density changes due to stretching, because this change is small, and the factor before M is only approximate.

2.1 Stationary solution

A ball falling from certain height will bounce lower and lower due to damping and air resistance. A player on the other hand can jump while on the trampoline, generating energy ΔE with his leg muscles. For the stationary solution, we know that this generated energy will be equal to the losses of energy in one jumping cycle. We want to approximate this energy. In order to do that we can observe that the trampoline movement of the player can be approximated as a harmonic oscillator. At the beginning of the cycle, when the player is directly above the trampoline, he has maximal velocity, thus his velocity can be approximated by $v(t) = v_0 cos(\omega t)$, where ω is its characteristic frequency. Jumping on the trampoline takes half of the period $\frac{\pi}{\omega}$ thus we can approximate the energy losses as:

$$-\Delta E_1 = \int_{0.5cycle}(F_a + F_d)(-\hat{\boldsymbol{e}}_{\boldsymbol{z}})\cdot\hat{\boldsymbol{e}}_{\boldsymbol{z}}dz = -\int_0^{\frac{\pi}{\omega}}\left(c_a\frac{\pi}{6}\frac{R^3}{\sqrt{R^2+z^2}}\dot{z}^3 sgn(\dot{z}) + c_d\dot{z}^2\right)dt \tag{7}$$

Now we can approximate that $\frac{R^3}{\sqrt{R^2+z^2}} \approx R^2$, because we want only order of magnitude approximate and deformation of typical trampoline for light players is smaller than the radius, thus we will ignore $\frac{z^2}{R^2}$ terms.
If we now input our anzatz $\dot{z} = v_0 cos(\omega t)$, we get:

$$\Delta E_1 = 2c_a\frac{\pi}{6}R^2\frac{v_0^3}{\omega}\int_0^{\frac{\pi}{2}} cos^3(x)dx + 2c_d\frac{v_0^2}{\omega}\int_0^{\frac{\pi}{2}} cos^2(x)dx = \left(\frac{4}{3}\frac{c_a}{6}R^2 v_0 + \frac{1}{2}c_d\right)\frac{\pi v_0^2}{\omega} \tag{8}$$

We also should include the kinetic energy of the mat which density velocity is equal to $v(r) = v_0\left(1-\frac{r}{R}\right)$. This energy is lost after launching the player due to dumping of trampoline:

$$\Delta E_2 = \frac{1}{2}\int_S \rho v(r)^2 = \frac{Mv_0^2}{12} \tag{9}$$

Only unknown for a given trampoline is v_0 - the escape velocity of a player. We ommit the drag during free-fall phase, thus it can be approximated with the energy conservation rule $v_0 = \sqrt{2gh_{max}}$. Where h_{max} is the maximal height above trampoline surface equilibrium the player reaches.
Finally, we get energy expenditure of the player during one jump ΔE as a function of his maximal height h_{max}:

$$\Delta E = gh_{max}\left(\frac{M}{6} + \left(\frac{4c_a}{9}R^2\sqrt{2gh_{max}} + c_d\right)\frac{\pi}{\omega}\right) \tag{10}$$

This approximation for energy that the player needs to generate during one cycle can be tested with a simulation solving differential equation. The stationary solution should be an attractor - thus not depend on the initial conditions. For it, the maximal height can be measured and compared with the one given at the beginning. ω was calculated numerically for simulation of the ball bouncing freely (without kick) and measuring time spent on the trampoline during the first bounce.

3 Simulation of a single player

In order to test the theoretical model developed in sec. 2, we developed a simulation solving an IVP (initial value problem) for ODE of evolution given by eq. 6. The implementation was created in python, using the solve_ivp function from the scipy package, with Runge-Kutta of 8th order method.

3.1 Non-resonant fall from certain height

The movement of the player was divided into stages. First, there was a part of free fall, upon contact with the trampoline the model with eq. 6 overtakes, to finally return to free fall phase upon leaving the trampoline. We treat the collision with the mat as conserving energy, because the vast majority of the energy losses has source in air resistance and spring heating.

The trampoline's parameters shown in table 1 were chosen by scaling up the parameters given in the reference [1], assuming that the spring elasticity constant and the mat's density won't change.

This simulation resulted in obtaining a damped oscillation we could expect, shown in figure 1.

Parameter	Value	Unit
R	2.5	m
M	6.25	kg
N	150	-
c_a	2.3	$\mathrm{Ns^2/m^4}$
c_d	16	Ns/m

$$\mathrm{k} = \begin{cases} 96\ \mathrm{kN/m}, & \Delta L < 1.5\ \mathrm{mm} \\ 2.7\ \mathrm{kN/m}, & \Delta L \geq 1.5\ \mathrm{mm} \end{cases}$$

Table 1: Parameters assumed for the trampoline, where ΔL is the elongation of the spring

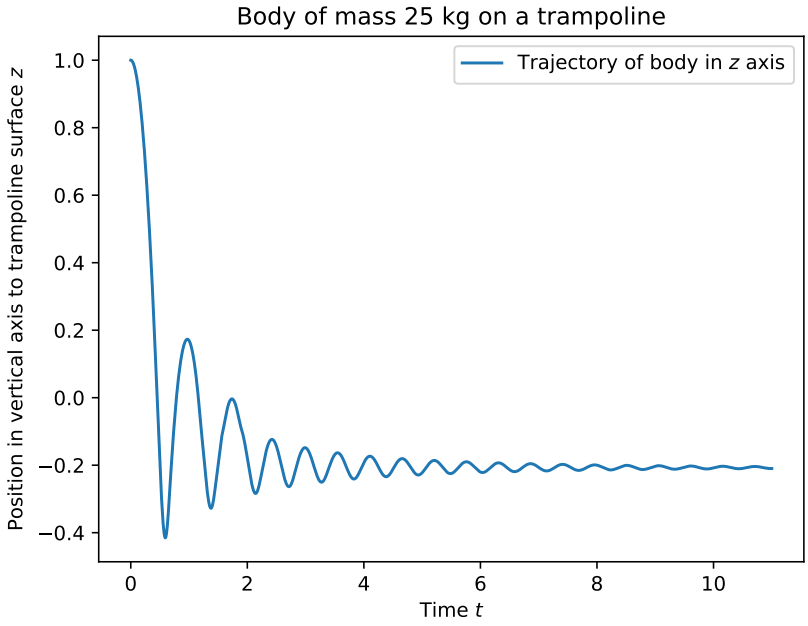

Figure 1: The non- resonant oscillation of the player with the mass of 25 kg

3.2 Resonant jumping

In reality, a player jumping on a trampoline doesn't act like a stiff material point, but during every contact with the mat he adds to the system some energy produced by his muscles. Knowing that, we can stop the solving of the equation of motion when the player stops in minimal z, change his kinetic energy by this ΔE with velocity directed upwards, and resume solving the equation.

We implemented this technique with a manually written event for solve_ivp. We have indeed obtained an attractor (as shown in figure 3.2), and in order to find the correct values of ΔE we first computed its values using the theoretical model 10, and by manual tweaks we optimised its value for every player separately in order for their maximal jumping height while bouncing alone h_{max} to be close to the given values.

Thus, we obtained ΔE for every player, as shown in table 2.

Player's mass, (kg)	ΔE_0, (J)	ΔE, (J)	Obtained h_{max}, (m)
25	123.25	126	0.496
40	215.04	220	0.799
50	347.03	343	1.202

Table 2: The theoretically obtained values of jump energy ΔE_0 and the values ΔE obtained from the resonant jumping simulation

It is worthy to emphasise the fact that the theoretically obtained values of ΔE are within 3% of the values obtained in the simulation. It shows the correctness of both the model and the simulation.

The comparison of the computed trajectories beginning in $z = 0$ with no initial velocity for different players is shown on figure 3.2.

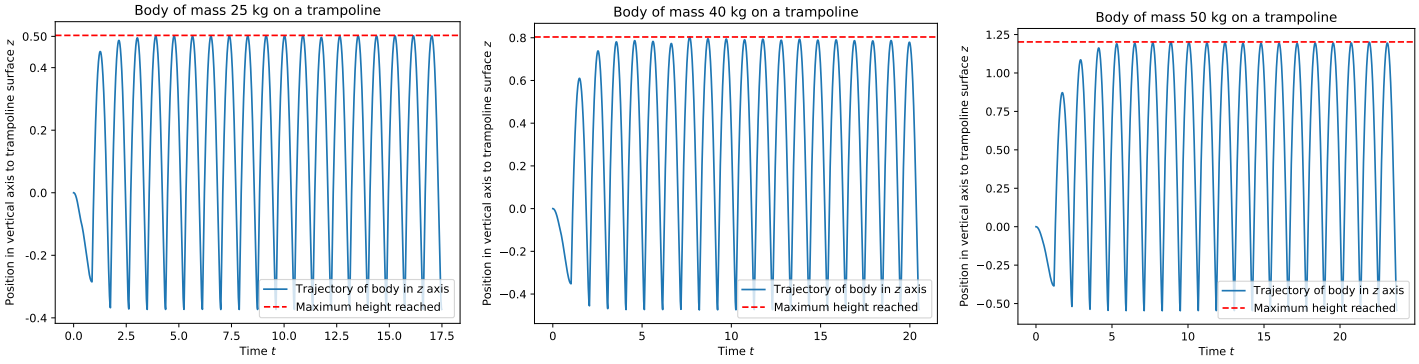

Figure 2: The comparison of the resonant trajectories with $z(t = 0) = 0$, $v(t = 0) = 0$

We have also analysed whether we will reach this value with different initial conditions, as such starting height $z = 1$ but for all initial conditions we have found that our trajectory will stabilise around the same h_{max} for a given jump energy.

4 Model of three players

The trampoline's radius $R = 2.5$ m is big in comparison with the human size, and the greatest height one can obtain is while jumping on the trampoline's center, so the optimal solution for the problem shouldn't be much different than the solution obtained with the assumption that all three players are jumping in the center of the mat, and cannot collide with each other.

The idea of the solution is based on [2]. When we consider a two-player static (no leg work) problem, we can observe that if contact with the trampoline times are far apart, every player is described by one player bouncing model 2. On the other hand, if times of contact are similar, they can transfer the energy between themselves using the trampoline. In a simple model if player A is making contact with the trampoline surface first he will be gradually slowed down. In the meantime player B who is near him is falling freely, thus accelerating. If the distance between them is small, there will be a moment when player B catches up to A, while having greater speed, thus he will be slowed down by the trampoline, while player A will start accelerating in free fall. They will switch positions, up to the point when both their velocities will be close to zero. Now the reverse process, where the trampoline accelerates them in turns will take place. During this process, at some point one of the players will leave the trampoline and never come back in this cycle, and due to the presence of the second player his escape velocity will be smaller than if the second player was not present. It will result in energy transfer to the second player. According to [2] if the players will keep bouncing, they will randomly transfer energy between each other.

It is the main difference between the single and multiple player problems. As we can see, 3 players (kids) can keep bouncing not interfering with each other, until all of them have reached their maximal heights. Thus they can pool together a lot of energy in order to launch one of them.

In the next step they need to focus on transferring as much energy as possible toward one of the players. We developed a simple model of optimizing such a transfer. First, one player lands on the trampoline and stretches it until he stops moving. Then, as he has no more energy that he could insert into the trampoline, he can jump, and on his place another player lands, exactly in the moment of jumping (fig. 3). This means that instead of accelerating the first player, the trampoline gets further deformed. The second player freely fell from his maximal height and has maximal kinetic energy, which is now transmitted into the trampoline. And when the second player stops, he jumps like the first one, and on his place in this exact moment lands the third player, preventing the trampoline from transferring any energy into the second player. Now the third player waits until he stops, then he jumps, and this time the trampoline accelerates him with all of the energy stored in it (except eventually some small part, transferred possibly to the second player if the third player with the trampoline accelerates enough to get ahead of him before reaching z=0, but it is already after the majority of the energy got transferred). It is easy to observe, that the acceleration is higher, when strings are more streched, thus majority of the energy stored in springs will be transferred to the intended target.

5 Simulation of three players

The algorithm mentioned in the previous paragraph has been implemented using the earlier written program simulating one player, and indeed, the height the last player jumping obtains much exceeds the heights of the other players, as shown in table 3.

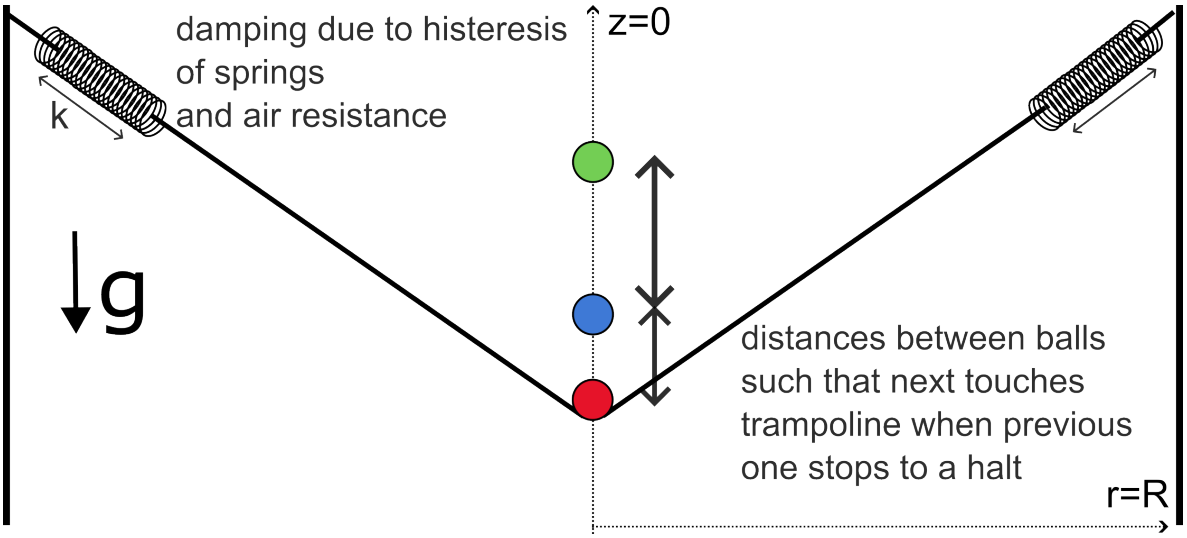

Figure 3: The schematic of a trampoline with balls separated in a way, allowing maximal transfer of energy towards one player. By engaging with the trampoline in the moment, when previous kid stops to a halt, players can store the greatest amount of energy in the trampoline, to eject one of them.

Player 1 (kg)	Player 2 (kg)	Player 3 (kg)	h_1 (m)	h_2, (m)	h_3, (m)
40	50	25	0.028	0.812	3.200
50	40	25	0.116	1.019	2.940
25	50	40	0.050	0.278	2.744
50	25	40	0.098	0.608	2.534
25	40	50	0.050	0.112	2.386
40	25	50	0.024	0.241	2.374

Table 3: The jump heights obtained while simulating the sequential trampoline stretching by players in given order

On the following figures 4, 5 and 6 the trajectories and velocities are shown in function of time for every player when he jumps last (the higher of the two jumps in each case was chosen).

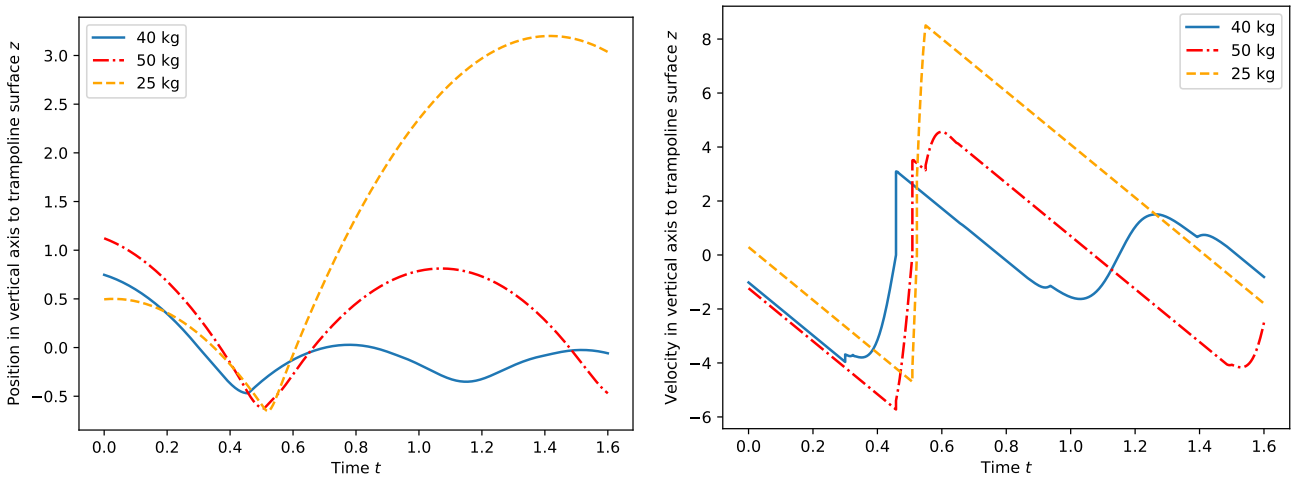

Figure 4: The optimal jump for the 25 kg player, locations and velocities in the function of time

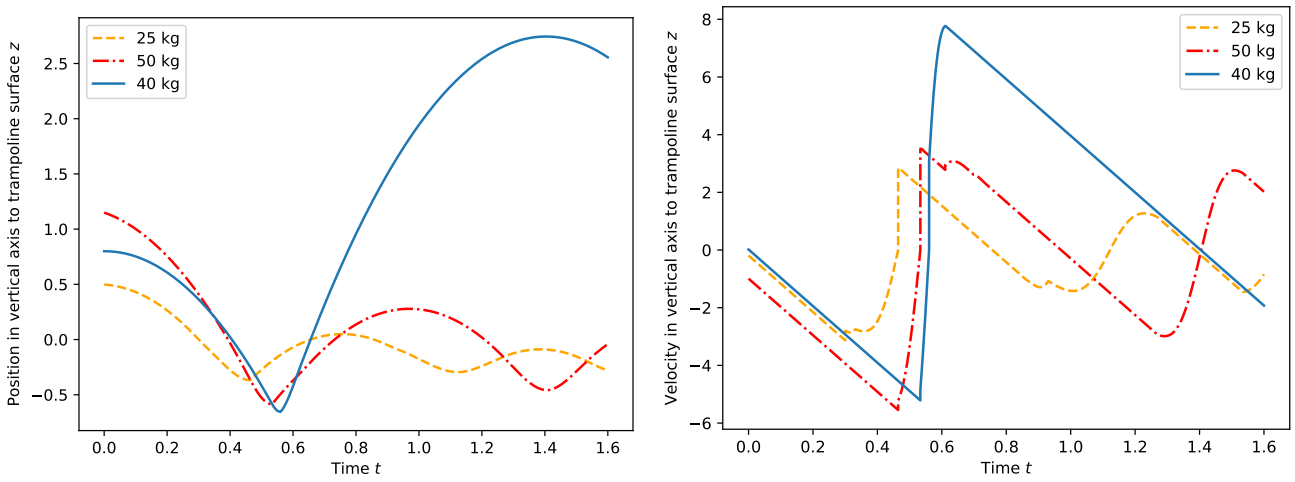

Figure 5: The optimal jump for the 40 kg player, locations and velocities in the function of time

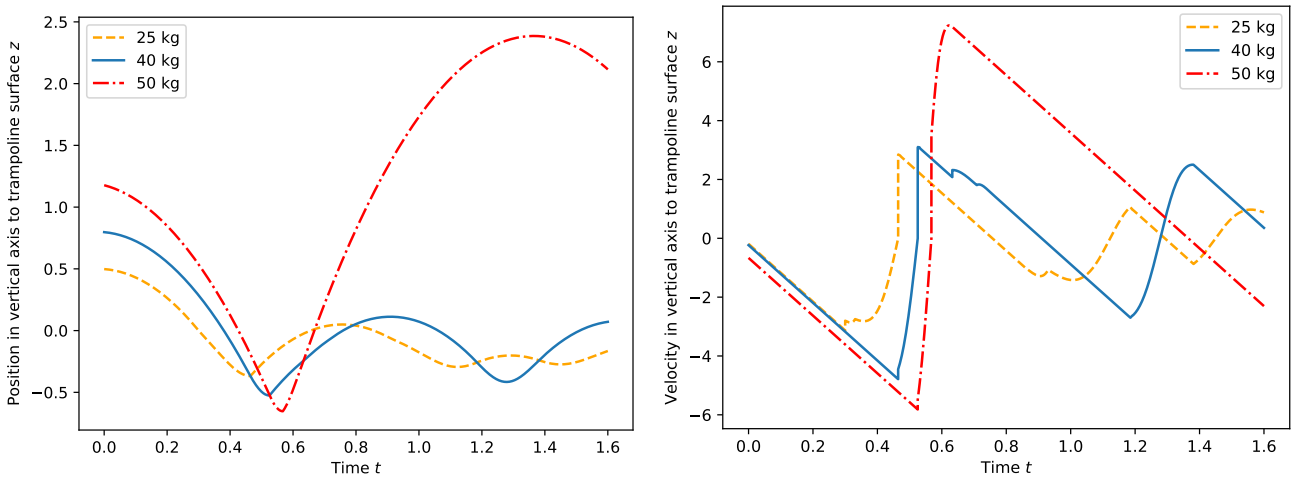

Figure 6: The optimal jump for the 50 kg player, locations and velocities in the function of time

6 Problems and discussion

6.1 Problem with energy

It is a well-known fact that many people jumping on a trampoline at the same time can reach much greater heights than when jumping alone, so the results seem to be of the correct form, increasing the height by the order of 3. It is unlikely that they are close to the exact values, because for the 50 kg player the system's total energy grows from 1025 J by 214 J, which is of the same order of magnitude as the sum of energies added by the muscles $\Sigma\Delta E = 689$ J. It is possible, that the model of instantaneous jump does not reflect reality well, and final velocity of the escaping player is too large, thus the maximal height is overestimated. The total energy of players after ejecting one of them is pretty much the same in every variation of order. It may be possible that in reality damping and drag forces do not balance the 3 players giving energy into the system. Most of the energy is in the ejected player, the smallest one will achieve the highest altitude, what is confirmed by our simulation.

6.2 Discussion and implications

The results although big are the correct order of magnitude and are in alignment with simple models of energy transfer omitting energy losses [2] and experimental results [1]. Some interesting conclusions may arise from our work.
First, what is intuitive energy generated by a player ΔE is an increasing function of mass of the player - greater muscle mass leads to higher generated power.
Second, single-player movement on the trampoline is given by the attractor, which does not depend on the initial conditions. No matter how certain player starts jumping after some time he jump with his maximal height.
Adding multiple players on one trampoline leads to chaotic behavior and energy transfers between them. Thus it is easy to see, that using the trampoline in a group can be way more interesting and fun, than jumping alone.
In the case of multiple players, it is possible for them to accidentally eject one of them way higher, than his maximal height. Maybe not by a factor of 3, as we calculated for an optimal situation, but it can still be dangerous for trampoline users. Falling from a height of more than 2 meters can cause twisted ankles and other injuries. Thus it is important to find ways to reduce possible maximum ejecting height while keeping the same maximum height for each player. From our model it can observed, that in the process of ejection, much higher maximal speeds are achieved (in some cases almost $8\frac{m}{s}$), thus the air drag on the trampoline starts to play a dominant role compared to dumping. It may be beneficial to design the trampoline in a way, that has a larger drag coefficient c_a, rather than damping one c_d.
Finally we can observe, that the lightest player was ejected the highest. This phenomenon can be explained intuitively, all the players store their energy in the trampoline, lightest player may have lowest power generating abilities ΔE, thus lowest single-bouncing maximal height h_{max}, but if all the players store their energy in the trampoline, the maximal height given by:

$$h = \frac{E_{total}}{mg} \tag{11}$$

Will be highest for smallest players if the total energy is the same. Of course, we cannot assume that all the energy is transferred into him, what can be observed in our simulation, but the intuition is right, thus while multiple people are playing on the trampoline it is really important to take safety of smaller kids seriously. In the same time it may be advantageous for breaking height records, to eject the smallest player.

7 Summary

In this paper a model of a single player bouncing on a trampoline was proposed. Both simulation and theoretical calculations gave a similar result for the energy produced by the muscles of each player during one jump (table 2), which is an important proof of the model's accuracy.
A model of three players bouncing simultaneously was proposed, and a strategy for obtaining the highest possible jump for a chosen player was constructed. It was then implemented in a simulation, and computed for every possible order in which the players can land on the trampoline (table 3). The highest jumps for the players weighing 25, 40 and 50 kg are respectively **3.200**, **2.744** and **2.386 m**. The trajectories and velocity plots for these highest jumps were drawn (figures 4, 5 and 6).

8 Appendix

To this paper there are added two scripts written in Python, the first one used to numerically model a ball dropped on a trampoline and to find the correct parameters for each person for them to be able to reach their maximum height. The second one was used to numerically simulate a three body jumping process on a trampoline and to find out how far can each person jump using a strategy described in the paper. More about scripts is written inside them using comments. In these scripts we are searching for the evolution of our system by simply solving the differential equations given by our theoretical model with forces acting on the bodies on the trampoline. To solve such differential equations we use command from the library scipy to python. There is more to these scripts besides simply solving differential equations, to find out which body is in contact with the trampoline we check the positions of each body during every part of solving the differential equations. Based on this which body is in contact with the trampoline, the equations for such body change, because when they are away from the trampoline they are simply free falling and when they are on the trampoline, the differential equations are the same as given by us in the theoretical model. When the body hits the trampoline we have to use the conservation of the energy law to find the velocity after the hit. To introduce bouncing from the trampoline we have to quit solving the differential equations, change the condition by adding some value to the velocity of a body bouncing and once again solve the differential equations right now with new initial conditions.

```
    import numpy as np
import matplotlib.pyplot as plt
import math as m
from scipy.integrate import solve_ivp
#parameters of the trampoline and air
R = 2.5 #radius of the trampoline
Mm = (2.5)**2 #trampoline's mat mass
Ns = 150 #number of springs
Ca = 2.3 #air resistance coefficient
Cd = 16 #damping coefficient
g = 9.81 #standard gravity constant
mass = [25,40,50] #mass of each person
energy = [126,220,343] #amount of energy each person uses while jumping
hmax = [0.5,0.8,1.2] #maximum height reached by a person while jumping alone
order = [1, 2, 0] #here we choose order of bodies dropping on the trampoline
for lista in mass, energy, hmax:
    help = np.copy(lista)
    for i in range(len(lista)):
        lista[i] = help[order[i]]
print(mass, hmax,energy)
def dL(H): #change of length of springs while stretching the trampoline
    return m.sqrt(R ** 2 + H ** 2) - R
def ks(H): #change of springs coefficient due to stretching
    if dL(H) < 1.5e-3:
        return 96000
    else:
        return 2700
def Ft(H): #vertical force of all the springs
    return Ns * ks(np.abs(H)) * (m.sqrt(R ** 2 + H ** 2)-R)*
    (H / (m.sqrt(R ** 2 + H ** 2)))
def Fa(H): #air resistance
    return (2 * np.pi * Ca) / R / m.sqrt(R ** 2 + H ** 2) * ((R ** 4) / 4 -
    2 * R * R ** 3 / 3 + R ** 2 * R ** 2 / 2)
```

```
def onebody(t, z,mass=mass[0]): #equations describing evolution
of a system with one body
    x, v = z
    dx = v
    if x > 0:
        dv = -g
    else:
        dv = -Ft(x) - mass * g - Fa(x) * v**2 * np.sign(v) - Cd*v
        dv = dv/(mass+(1/3)*Mm)
    return np.array([dx, dv])
def threebodies(t, z): #equations describing evolution of a system with three bodies
    positions = z[:3]
    velocities = z[3:]
    v1,v2,v3 = velocities
    positionequations = [0,0,0]
    velocitiesequations = [0,0,0]
    positionequations[0] = v1
    positionequations[1] = v2
    positionequations[2] = v3
    if min(positions) > 0:
        velocitiesequations[0] = -g
        velocitiesequations[1] = -g
        velocitiesequations[2] = -g
    else:
        velocitiesequations[0] = -g
        velocitiesequations[1] = -g
        velocitiesequations[2] = -g
        index = np.where(positions == min(positions))[0]
        for i in range(len(index)):
            velocitiesequations[index[i]] =
            -Ft(positions[index[i]]) - mass[index[i]] *
            g - Fa(positions[index[i]]) * velocities[index[i]]**2 * np.sign(velocities[i
            Cd*velocities[index[i]]
            velocitiesequations[index[i]] =
            velocitiesequations[index[i]]/(mass[index[i]]+(1/3)*Mm)
    return np.array([positionequations[0],positionequations[1],positionequations[2],velo
def twobodies(t, z): #equations describing evolution of a system with two bodies
    positions = z[:2]
    velocities = z[2:]
    v1,v2 = velocities
    positionequations = [0,0]
    velocitiesequations = [0,0]
    positionequations[0] = v1
    positionequations[1] = v2
    if min(positions) > 0:
        velocitiesequations[0] = -g
        velocitiesequations[1] = -g
    else:
        velocitiesequations[0] = -g
        velocitiesequations[1] = -g
        index = np.where(positions == min(positions))[0]
        for i in range(len(index)):
```

```
            velocitiesequations[index[i]] =
            -Ft(positions[index[i]]) - mass[index[i]]
            * g - Fa(positions[index[i]]) * velocities[index[i]]**2 * np.sign(velocities
            - Cd*velocities[index[i]]
            velocitiesequations[index[i]] = velocitiesequations[index[i]]/(mass[index[i]]
    return np.array([positionequations[0],positionequations[1],velocitiesequations[0],ve
'''
For two and more bodies our algorithm works in a way,
that when one of the bodies reaches the position under
0 (it means it reaches the trampoline
at the equilibrium with no mass on it), we are scanning
the positions to find the lowest one, the lowest one body is in a contact with the tramp
for this body we apply the equations for velocity as it
follows, and the other bodies are in a free fall; the trampoline moves with the velocity
of a body on it
'''
#functions used as events in solve_ivp functions to find times at which certain values a
def velocity1(t,z): #find time when velocity of the first body is equal to zero
    return z[3]
def velocity2(t,z): #find time when velocity of the second body is equal to zero
    return z[4]
def velocity3(t,z): #find time when velocity of the third body is equal to zero
    return z[5]
def velocity1twobodies(t,z): #find time when velocity of the first body is equal to zero
    return z[2]
def velocity2twobodies(t,z): #find time when velocity of the second body is equal to zer
    return z[3]
def velocityonebody(t,z): #find time when velocity of the only body is equal to zero (on
    return z[1]
def collision1(t,z): #find time when position of the third body is equal to position of
    return z[2] - z[0]
def collision2(t,z): #find time when position of the third body is equal to position of
    return z[2] - z[1]
def positionfunc(t,hmax,ti,hi): #function to find position of a free fallling body befor
    return hmax - g/2 * (t-(ti-m.sqrt(2*(hmax-hi)/g)))**2
def velocityfunc(t,hmax,ti,hi): #function to find velocity of a free fallling body befor
    return -g*(t-ti+m.sqrt(2*(hmax-hi)/g))
'''
We imagine that all the bodies start at a certain maximum height,
which is reached by each body when jumping alone and then from it
it falls on a trampoline during a free fall. According to our paper,
we will reach the highest jump, when every next body hits the trampoline
the moment the previous one reaches maximum depth for the trampoline
(its velocity reaches 0). That's why we start our simulation with only one body
at the level of trampoline and velocity gained from a free fall
from maximum height, multiplied by a coefficient connected
to conservation of momenta.
'''
z01 = [0.,-m.sqrt((mass[0]/(mass[0]+Mm))*(2*g*hmax[0]))]
res = solve_ivp(onebody,(0.3,15.),z01,method='DOP853',events=velocityonebody,dense_outpu
'''
Using events we find the time at which velocity of
```

```
the body reaches 0, it's the moment when we want our
second body to hit the trampoline. We find
TIME1 of this event and height of this body HMIN1
'''
TIME1 = res.t_events[0][0]
HMIN1 = res.y_events[0][0][0]
'''
Now we start the evolution of the two body system,
we suppose that the second body hits the trampoline
when the velocity of the first one is zero
that's why our initial conditions are as it follows,
we change the speed of the first body by a boost from
using its legs on the trampoline surface
to launch himself even further up, the velocity of the
second body comes from free fall from its maximum height
to the location of trampoline surface
'''
z02=[HMIN1,HMIN1,0+np.sqrt(2*energy[0]/(mass[0]+Mm))
,-m.sqrt((mass[1]/(mass[1]+Mm))*(2*g*(hmax[1]-HMIN1)))]
res1 = solve_ivp(twobodies,(TIME1,15.),z02,method='DOP853'
,dense_output=True,events=(velocity1twobodies,velocity2twobodies))
'''
We use events once again, this time to find the time
when velocity of the second body (the one on the trampoline)
reaches 0, so our third body
can hit the trampoline to gain maximum height
'''
A = res1.t_events
B = res1.y_events
TIME2 = res1.t_events[1][0]
HMIN2 = res1.y_events[1][0][1]
'''
Now we simulate the evolution of a three body system,
generating the third body at the position of the second one,
with velocity gained during
the free fall and multiplied by a coefficient connected
to the conservation of momenta
'''
z03 = [B[1][0][0],HMIN2,HMIN2,B[1][0][2],0+
np.sqrt(2*energy[1]/(mass[1]+Mm)),
-m.sqrt((mass[2]/(mass[2]+Mm))*(2*g*(hmax[2]-HMIN2)))]
res2 = solve_ivp(threebodies,
(TIME2,1.6),z03,method='DOP853',
events=(velocity3),dense_output=True)
'''
We evolve our system up until the moment the third body
(the one we want to launch) reaches velocity equal
to zero, to find the moment when we will
instantaneously push it from the trampoline with the energy dE
'''
A = res2.t_events
B = res2.y_events
```

```
TIME3 = A[0][0]
z04 = [B[0][0][0],B[0][0][1],B[0][0][2],B[0][0][3],B[0][0][4],B[0][0][5]+np.sqrt(2*energy
res3 = solve_ivp(threebodies, (TIME3,1.6), z04,method='DOP853',events=(collision1,collisi
'''
The last part of our evolution is equipped with
the events that check the positions of the body on
the trampoline and the previous two bodies, so
when the trampoline reaches one of the bodies, the
velocity of this system changes according to the rules of momentum conservation, we didn
it for the previous collisions, because the
velocities where similar and we could just
approximate that the velocity of the trampoline is exactly
the same as the velocity of a body hitting it
'''
A1 = res3.t_events[0]
A2 = res3.t_events[1]
TIMES = [A1[0],A2[0]]
TIME4 = min(TIMES)
TIMEINDEX = TIMES.index(min(TIMES))
B=res3.y_events[TIMEINDEX][0]
velocityaftercollision = B[TIMEINDEX+3]
* m.sqrt(1 + Mm*(B[5]**2 / B[TIMEINDEX+3]**2 - 1)
/(6*(mass[TIMEINDEX]+Mm)))
z05 =[B[0],B[1],B[2],B[3],B[4],B[5]]
z05[TIMEINDEX+3] = velocityaftercollision
#free evolution of a three body system
res4 = solve_ivp(threebodies, (TIME4,1.6), z05,
method='DOP853',dense_output=True)
#Lists where we will append positions and momenta of bodies
X1=[]
X2=[]
X3=[]
V1=[]
V2=[]
V3=[]
'''
Times at which we want to evaluate the
positions and momenta, the first three
t1dod, t2dod and t3dod are used for the
free fall evolution
before the simulation
'''
#There can be an error regarding the shapes
because of the m.floor function, then we
have to add one more element in the np.linspace()
t1dod = np.linspace(0,0.3,m.floor(2000*0.3/1.5))
t2dod = np.linspace(0,TIME1,m.floor(2000*TIME1/1.5))
t3dod = np.linspace(0,TIME2,m.floor(2000*TIME2/1.5))
t1 = np.linspace(0.3,TIME1,m.floor(2000*(TIME1-0.3)/1.5))
t2 = np.linspace(TIME1,TIME2,m.floor(2000*(TIME2-TIME1)/1.5))
t3 = np.linspace(TIME2,TIME3,m.floor(2000*(TIME3-TIME2)/1.5))
t4 = np.linspace(TIME3,TIME4,m.floor(2000*(TIME4-TIME3)/1.5))
```

```
t5 = np.linspace(TIME4,1.6,m.floor(2000*(1.6-TIME4)/1.5))
#Here we append the free fall evolution

for i in range(len(t1dod)):
    X1.append(positionfunc(t1dod[i],hmax[0],0.3,0))
    V1.append(velocityfunc(t1dod[i],hmax[0],0.3,0))
for i in range(len(t2dod)):
    X2.append(positionfunc(t2dod[i],hmax[1],TIME1,HMIN1))
    V2.append(velocityfunc(t2dod[i],hmax[1],TIME1,HMIN1))
for i in range(len(t3dod)):
    X3.append(positionfunc(t3dod[i],hmax[2],TIME2,HMIN2))
    V3.append(velocityfunc(t3dod[i],hmax[2],TIME2,HMIN2))

#Here we append elements coming from the evolution inside the differential equations
for i in range(len(t1)):
    X1.append(res.sol(t1)[0][i])
    V1.append(res.sol(t1)[1][i])
for i in range(len(t2)):
    X1.append(res1.sol(t2)[0][i])
    X2.append(res1.sol(t2)[1][i])
    V1.append(res1.sol(t2)[2][i])
    V2.append(res1.sol(t2)[3][i])
for i in range(len(t3)):
    X1.append(res2.sol(t3)[0][i])
    X2.append(res2.sol(t3)[1][i])
    X3.append(res2.sol(t3)[2][i])
    V1.append(res2.sol(t3)[3][i])
    V2.append(res2.sol(t3)[4][i])
    V3.append(res2.sol(t3)[5][i])
for i in range(len(t4)):
    X1.append(res3.sol(t4)[0][i])
    X2.append(res3.sol(t4)[1][i])
    X3.append(res3.sol(t4)[2][i])
    V1.append(res3.sol(t4)[3][i])
    V2.append(res3.sol(t4)[4][i])
    V3.append(res3.sol(t4)[5][i])
for i in range(len(t5)):
    X1.append(res4.sol(t5)[0][i])
    X2.append(res4.sol(t5)[1][i])
    X3.append(res4.sol(t5)[2][i])
    V1.append(res4.sol(t5)[3][i])
    V2.append(res4.sol(t5)[4][i])
    V3.append(res4.sol(t5)[5][i])
#Here we plot everything, before plotting we should change the labels, here they are set

plt.figure()
plt.plot(np.concatenate((t1dod,t1,t2,t3,t4,t5)),X1, label=f'{mass[0]}␣kg',linewidth=1.6)
plt.plot(np.concatenate((t2dod,t2,t3,t4,t5)),X2, label=f'{mass[1]}␣kg',c='red',linestyle=
plt.plot(np.concatenate((t3dod,t3,t4,t5)),X3, label=f'{mass[2]}␣kg',c='orange',linestyle=
plt.ylabel('Position␣in␣vertical␣axis␣to␣trampoline␣surface␣$z$')
plt.xlabel('Time␣$t$')
plt.legend()
```

```
plt.show()
plt.figure()
plt.plot(np.concatenate((t1dod,t1,t2,t3,t4,t5)),V1,
label=f'{mass[0]} kg',linewidth=1.6)
plt.plot(np.concatenate((t2dod,t2,t3,t4,t5)),V2,
label=f'{mass[1]} kg',c='red',linestyle='-.',linewidth=1.6)
plt.plot(np.concatenate((t3dod,t3,t4,t5)),V3, label=f'
{mass[2]} kg',c='orange',linestyle='—',linewidth=1.6)
plt.ylabel('Velocity in vertical axis to trampoline surface $z$')
plt.xlabel('Time $t$')
plt.legend()
plt.show()
#Here we can check the maximum values of the height of bodies during the whole evolution

print(max(X1))
print(max(X2))
print(max(X3))
```

[language=Python]

```
import numpy as np
import matplotlib.pyplot as plt
import math as m
from scipy.integrate import solve_ivp
'''
Code used to numerically simulate a ball fallling
on a trampoline without extra energy bounces to
reach a certain maximum energy and in the next loop
we add the possiblity of using the energy dE by
the body to bounce itself higher than previously
and reach its maximum energy. All variable names are
the same as in the other code
'''
Rm = 2.5
Rf = 2.7
Mm = (2.5)**2
Ns = 150
Ca = 2.3
Cd = 16
Fc = 2.5
g = 9.81
Mb = 25
dE = 126


def dL(H):
    return m.sqrt(Rf ** 2 + H ** 2) - Rf


def ks(H):
    if dL(H) < 1.5e-3:
        return 96000
    else:
```

```
        return 2700


def Ft(H):
    return Ns * ks(np.abs(H)) *
    (m.sqrt(Rf ** 2 + H ** 2)-Rf)
    *(H / (m.sqrt(Rf ** 2 + H ** 2)))


def Fa(H):
    return (2 * np.pi * Ca) / Rf / m.sqrt(Rf ** 2 + H ** 2)
    * (
                (Rm ** 4) / 4 - 2 *
                Rf * Rm ** 3 / 3 + Rf ** 2 * Rm ** 2 / 2)


def function(z, t):
    x, v = z
    dx = v
    if x > 0:
        dv = -g
    else:
        dv = -Ft(x) - Mb * g - Fa(x) * v**2 * np.sign(v) - Cd*v
        dv = dv/(Mb+Mm/3)
    return np.array([dx, dv])
def function1(t, z):
    x, v = z
    dx = v
    if x > 0:
        dv = -g
    else:
        dv = -Ft(x) - Mb * g - Fa(x) * v**2 * np.sign(v) - Cd*v
        dv = dv/(Mb+Mm/3)
    return np.array([dx, dv])
def velocity(t,z):
    return z[1]
"""
Part of code used to numerically simulate a ball
falling on a trampoline, not pushing itself from the surface
"""
t1=0
t3 = np.linspace(0, 17.48728557571967, 200000)
t = np.linspace(t1, 11, 10000)
z0 = [0, 0]
res = solve_ivp(function1, (0,11), z0,method='DOP853',
events=velocity,dense_output=True)
F = res.sol(t)[0]
plt.figure(figsize=(14, 8))
plt.title('Body␣of␣mass␣25␣kg␣on␣a␣trampoline')
plt.plot(t,res.sol(t)[0], label='Trajectory␣of
body␣in␣$z$␣axis')
plt.ylabel('Position␣in␣vertical␣axis␣to␣trampoline
```

```
surface $z$')
plt.xlabel('Time $t$')
plt.legend()
plt.show()
X=[]
V=[]
'''
Part of code used to numerically simulate a person
as a point, falling on a trampoline and boosting
itself (using legs) from its
surface by gaining some dE together with the trampoline.
'''
for i in range(20):
    values=[]
    times=[]
    A=0
    B=0
    res=0
    res = solve_ivp(function1, (t1+1e-5,300), z0,method='DOP853',events=velocity,dense_o
    A = res.t_events
    B = res.y_events
    for i in range(len(A[0])):
        if B[0][i][0] < 0:
            values.append(B[0][i])
            times.append(A[0][i])
    t2 = t1
    t1 = times[0]
    print(t2,t1)
    t4 = np.linspace(t2,t1,10000)
    X.append(res.sol(t4)[0])
    V.append(res.sol(t4)[1])
    z0=[values[0][0],values[0][1]+np.sqrt(2 * dE / (Mm+Mb))]
X1 =[]
V1 =[]
for i in range(len(X)):
    for j in range(10000):
        X1.append(X[i][j])
        V1.append(V[i][j])
granica=[]
xtiki=[-0.4,-0.2,0.0,0.2,0.4]
for i in range(len(X1)):
    granica.append(0.803639780522247)
plt.figure(figsize=(14, 8))
plt.title('Body of mass 25 kg on a trampoline')
plt.plot(t3,X1, label='Trajectory of body in $z$ axis')
plt.ylabel('Position in vertical axis to trampoline surface $z$')
plt.xlabel('Time $t$')
plt.yticks([0.50]+xtiki,["0.50"]+xtiki)
#plt.plot(t3,granica, '--', c='red', label='Maximum height reached ')
plt.axhline(0.5031372855361776, linestyle='—',
color='r',label='Maximum height reached ')
plt.legend(loc='lower right')
```

```
plt.show()
print(max(X1))
```

[language=Python]

References

[1] David Eager et al. “Investigation into the Trampoline Dynamic Characteristics and Analysis of Double Bounce Vibrations”. In: *Sensors* 22.8 (2022). ISSN: 1424-8220. DOI: 10.3390/s22082916. URL: https://www.mdpi.com/1424-8220/22/8/2916.

[2] Manoj Srinivasan, Yang Wang, and Alison Sheets. “People Bouncing on Trampolines: Dramatic Energy Transfer, a Table-Top Demonstration, Complex Dynamics and a Zero Sum Game”. In: *PLOS ONE* 8.11 (Nov. 2013), pp. 1–13. DOI: 10.1371/journal.pone.0078645. URL: https://doi.org/10.1371/journal.pone.0078645.