A Three Dimensional Spring-Mass Model for Bipedal Locomotion and Prosthetic Leg Design

Bachelor Thesis

Author	J.J. Sleijfer[*]
Supervisors	dr. V. Vaniushkina
	dr. P.M. Visser
	prof. dr. J.M. Thijssen
Faculties	EEMCS and AS
	Delft University of Technology

[*] J.J.Sleijfer@student.tudelft.nl

July 20, 2023

Laymen Summary

Walking is a very complex motion. This report aims to approximate this complex motion with a very simple system consisting of one mass and two springs connected to two massless feet. The model could be used to design springs in simple prosthetic legs. The model can create data that is periodic and also find the best springs to emulate a person walking or running. The periodic data generation shows there are two distinct ways to walk, but only one way to run. The model also shows that the maxima of the vertical and lateral trajectories are very close to where the feet are placed. The fit describes the general characteristics of the motion well, but fine details are lost. Future research should investigate how problematic these deviations of the data are perceived by people using a simple prosthetic.

Summary

Humans are efficient at moving due to their exceptional mastery of bipedal locomotion. Several models have been made that attempt to model the motion of the centre of mass with a spring-mass system with various degree of success. For example, a two dimensional model tracks the height of the centre of mass well and a three dimensional model explains the normal force exerted on the ground.

In this report a three dimensional model is constructed to describe the motion of the centre of mass, with the purpose to determine the spring constant and the rest length in a simple prosthetic leg. The research question is: can a three dimensional spring-mass model accurately track the centre of mass and can this be used to determine the optimal properties of the spring of a simple prosthetic leg?

The spring mass model consists of two springs connected to the ground and the centre of mass. The springs do not exert a force on the centre of mass if they are extended. The model incorporates steps by moving the ground connection points instantaneously over fixed points on a rail. The model can be used in two distinct ways. The first generates a periodic trajectory by finding the optimal initial positions y_0 and z_0 (PPFA) and the second finds the optimal spring parameters k and u to fit a data set (DFA). The optimal conditions are found by discretizing the parameter space and using an iterative process. The space around the best parameters becomes the parameter space for the next iteration.

The PPFA shows that the initial lateral displacement z_0 is nearly constant with respect to the spring constant when the rest length is chosen such that the centre of mass is at constant height if the model is stationary. The PPFA also shows that there are two distinct y-trajectories possible for walking. Running, however, has only one trajectory. Several versions of the model are fit to the data: only discretised step widths, discretised step widths and a x_0-offset, and distinct legs. There is not much difference between these models. The x_0-offset is only $\Delta s = -0.02\text{m}$ confirming that the position of the feet does align with the extreme values of the y- and z-position. The model fits the general characteristics well but fine details in the motion are lost. Future research should investigate how cumbersome these deviations of the data are perceived by people using a simple prosthetic.

Table of Quantities

Quantity	Unit	Explanation
$_1$ and $_2$	[-]	$_1$ refers to the left leg and $_2$ to the right leg.
k	[Nm^{-1}]	Spring constant of a spring.
k_v	[Nm^{-1}]	Virtual spring constant that is a function of position of the centre of mass of the feet.
u	[m]	Rest length of a spring.
M	[kg]	Total mass.
$\mathcal{U}$	[J]	Potential of the system.
$\mathcal{H}$	[J]	Hamiltonian of the system.
$\mathbf{R}$	[m]	Position of the centre of mass with components x, y, z.
$\mathbf{R}_0$	[m]	Initial position with components x_0, y_0, z_0.
$\mathbf{R}_d$	[m]	Position points of the data.
n	[-]	Number of data points.
$\mathbf{p}$	[kgms^{-1}]	Momentum of the centre of mass with components p_x, p_y, p_z.
$\mathbf{p}_0$	[kgms^{-1}]	Initial momentum with components $p_{x_0}, p_{y_0}, p_{z_0}$.
$\mathbf{p}_d$	[kgms^{-1}]	Momentum points of the data
$\mathbf{r}$	[m]	Position of a foot.
t	[s]	Time.
t_t	[s]	Termination time, the duration that a simulation runs before it terminates.
t_c	[s]	Computation time. The time that it takes to compute a simulation.
T	[s]	Period of a trajectory, provided that it has one.
e	[-]	Error of either the PPFA or the DFA.
g	[ms^{-2}]	Gravitational constant
s_w	[m]	Step width. The distance between the foot is $2s_w$.
s_s	[m]	Step size. The distance in the forward (x)-direction between $\mathbf{r}_1$ and $\mathbf{r}_2$.
Δs	[m]	Displacement of the initial position with respect to the feet.
$\mathbf{s}_s$	[m]	The step size vector that has value $(s_s, 0, 0)^\top$.
N	[-]	The maximum number of steps that a simulation may consist of. If this number is reached the simulation terminates.
x	[m]	Coordinate along the direction of motion.
y	[m]	Coordinate of the height.
z	[m]	Coordinate of the lateral position.

Contents

CHAPTER
ONE

Introduction

Walking is the most fundamental form of movement for humans. Moving on two legs, more generally called bipedal locomotion, is a special skill. Humans are not unique in this, some other animals walk, hop, or run bipedally. The difference between running and walking is that running has a period each stride where both feet are off the ground, while walking does not. For example, birds walk on two feet, just as some apes do occasionally. On the other hand, many lizards and cockroaches run bipedally at their highest speed (figure 1.1). There are also animals that move solely on two feet other than humans, such as kangaroos. However, the level of mastery that humans have achieved regarding bipedal locomotion is one of a kind in the animal kingdom [1].

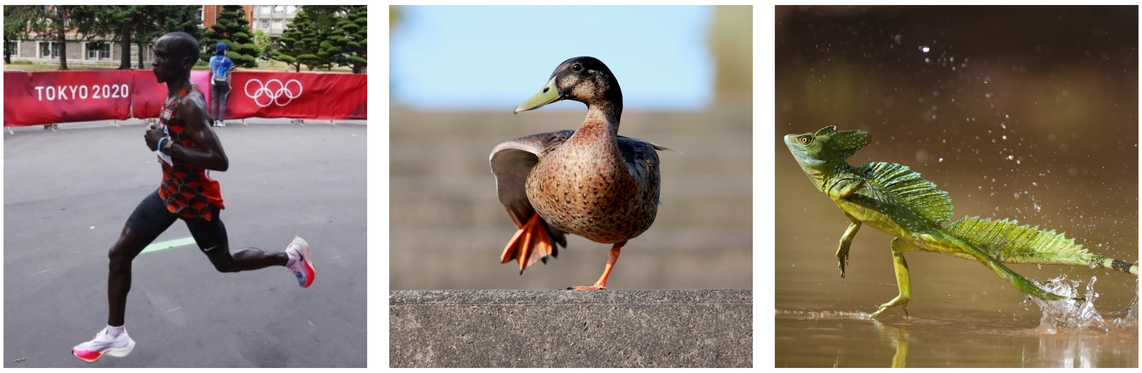

Figure 1.1: Several animals moving bipedally. Left: Eliud Kipchoge running. He is considered to be the best marathon runner of all time [2]. Centre: front view of a walking mallard [3]. Right: a double-crested basilisk running over a water surface [4]

Learning to walk is a step by step process. From birth we spend years trying to mimic other humans in their way of motion. This undertaking is quite literally a process of falling and rising back up. On average, children aged 12 to 19 months fell 17 times per hour walked. A benefit of walking in comparison to crawling is the ease of movement. A novice walker can travel farther and faster than expert crawlers, while having comparable fall rates. So the risk is the same, with higher rewards making walking a very attractive alternative, motivating to improve on the skill even further [5].

Among the animals, humans are not the fastest, do not have the sharpest claws, are not the strongest, and are not the best swimmers. However, there is something that humans are truly marvelous at: bipedal locomotion. In an experiment, chimpanzees were monitored while they were walking on two legs. The investigators found that this is on average more energy intensive than walking on all fours, the so called knuckle walk. And the knuckle walk already uses 75% more energy than when humans walk, requiring a mere $0.050 \pm 0.004 \mathrm{mlO_2(kgm)^{-1}}$ versus $0.210 \pm 0.014 \mathrm{mlO_2(kgm)^{-1}}$ [6]. This is true in general: humans are exceptionally efficient at moving [7].

Suppose there would be a running competition between animals, say the cheetah, a horse and a human. If the distance is a hundred metres, the cheetah will win by reaching a tremendous speed. If the distance is longer, for instance ten kilometres, then the horse will win. If the competition is a marathon, then the human will win, provided that the climate is hot. Otherwise either the human or the horse could be victorious, neither one having a clear advantage over the other. The competitiveness of humans at a large distance can be extrapolated even further. Humans can win at marathon distance from all other mammals. The reason being that, at this distance, elite marathon runners run at speeds over $5\ \mathrm{ms}^{-1}$, which is above the trot-gallop transition speed of all quadrupedal mammals. This is relevant because the trotting speed of a mammal is the endurance running speed: the speed that mammals can run at for sustained periods of time [8].

The ability to run for long distances is a significant physiological trait separating humans from the other mammals. Walking is a complex process where all sorts of organs in the human body collaborate. For complex processes it is generally a good idea to reduce the complexity of the system in order to be able to model it. A simplified model will improve upon the understanding of primary walking characteristics, but comes at the cost of nuance. This is a fine balance that some recent papers tried to address.

In 2006, Geyer, Seyfarth and Blickhan reduced the complete walking motion to a two dimensional dual spring, spring-mass model [9]. Although this model yields accurate height tracking relative to the forward direction, the model falls short in explaining the lateral motion, and therefore also in explaining the ground reaction forces. In 2019, the model has been extended to a three dimensional model (figure 1.2) which has been validated against the ground reaction forces. In the same year, Schreiber and Moissenet published a data set [10] that tracks the movement of 52 sensors on human participants walking on a treadmill at different speeds.

Even more recent, in January of 2023 to be precise, a cost-effective prosthetic leg designed by Hoque et al. showed an elegant simplistic design (figure 1.3). This prosthetic leg costs only 277 dollars which is over an order of magnitude less than other simple prosthetic legs [12]. The drastically lower price makes this prosthetic leg, in stark contrast with other prosthetics, a suitable option for people missing a leg in developing nations.

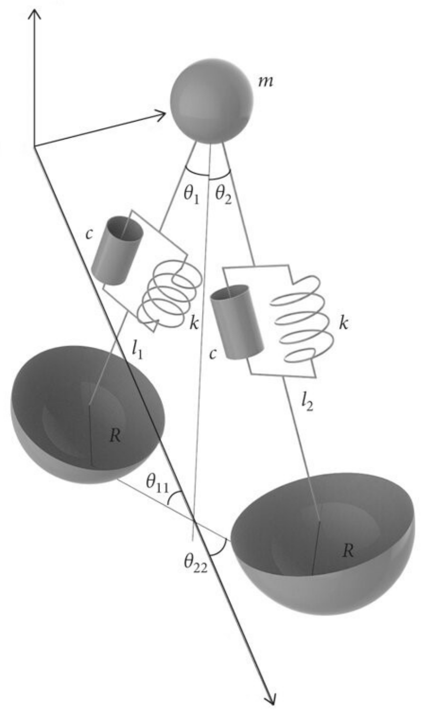

Figure 1.2: The three dimensional spring-mass model as proposed by Liang et al. This model includes a damper and sphere like feet [11].

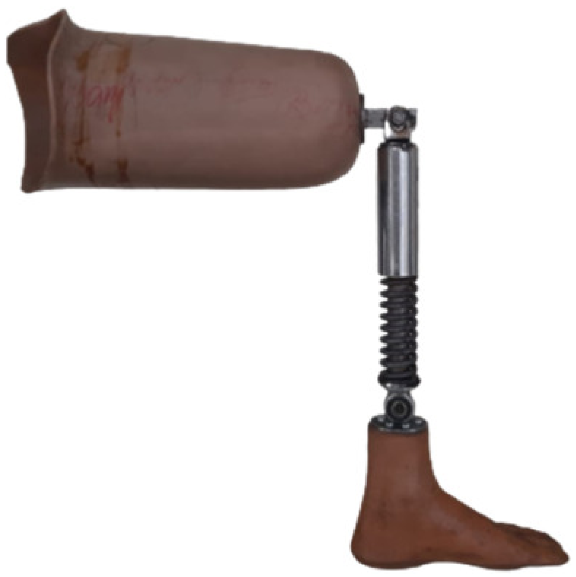

Figure 1.3: The prosthetic leg designed by Hoque consists of a single spring [12] .

This report aims to unify the three dimensional spring-mass model with the data set with two main goals in mind. The first is the verification of the model in the first place: does the three dimensional mass spring model accurately describe the motion of a human bipedal gait cycle? If this is the case, then the second goal may be addressed. The second goal is to investigate the properties of the simple prosthetic with the verified model in order to find the optimal spring given a persons key characteristics. These goals are captured in the research question: can the three dimensional spring-mass model accurately track the centre of mass and can this be used to determine the optimal properties of the spring of a simple prosthetic leg?

First the chapter Theory will set up the model and derive the equations of motion. Then the chapter Implementation will discuss the numerical implementation of the equations of motion, the construction of two fitting algorithms, and the estimation of parameters. In the Data chapter the processed data is introduced. Finally there is a chapter Results and Discussion which is followed by the Conclusion.

CHAPTER

TWO

Theory

In this chapter, we will build up the model for two legs and discuss the forces in the system. The goal of the model is to track the centre of mass, just as the three dimensional spring-mass model proposed by Liang et al. [11]. That model is not satisfactory for the goal of this report because it cannot simulate steps without performing coordinate transformations. This limits the possibilities for acquiring insight regarding the behaviour of the model over several steps. Hence a new foundation must be made. First the setup will be explained. Then the equations of motion are derived with Hamiltonian mechanics. The last sections discuss the position dependent spring constant, taking steps, and periodic properties of the solution.

2.1 Setup of a Three Dimensional Spring-Mass Model

Consider three dimensional space with a single point mass connected to two ideal massless springs. The other ends of the springs are fixed in place. A spring represents an entire leg. This is the most significant assumption of the model. Let $\mathbf{R}$ denote the position of the point mass. Let $\mathbf{r}_1$ and $\mathbf{r}_2$ denote the position of the fixed ends of the springs. Let the springs have spring constant k_1 and k_2 and rest length u_1 and u_2 respectively. Figure 2.1 illustrates the setup. In this system, there are several forces that are acting on the system. The gravitational force is defined as: $\mathbf{F}_g = -Mg\hat{\mathbf{y}}$, where M is the mass of the object, g the gravitational constant, and $\hat{\mathbf{y}}$ the unit vector in the vertical direction. [13]

Hooke's law states that the force of a spring is linearly dependent with the compression (or extension). This empirical law forms the basis of the ideal spring. The total length of the spring is given by $||\mathbf{R} - \mathbf{r}||$. By applying Hooke's law, the spring force $\mathbf{F}_s$ for a spring with rest length u and fixed to a point mass at $\mathbf{R}$ and at $\mathbf{r}$ is

$$\mathbf{F}_s = -k(||\mathbf{R} - \mathbf{r}|| - u)\frac{\mathbf{R} - \mathbf{r}}{||\mathbf{R} - \mathbf{r}||}. \tag{2.1}$$

In this equation, the spring constant k is the proportionality coefficient. The next section derives the Hamiltonian from the forces [14, 15].

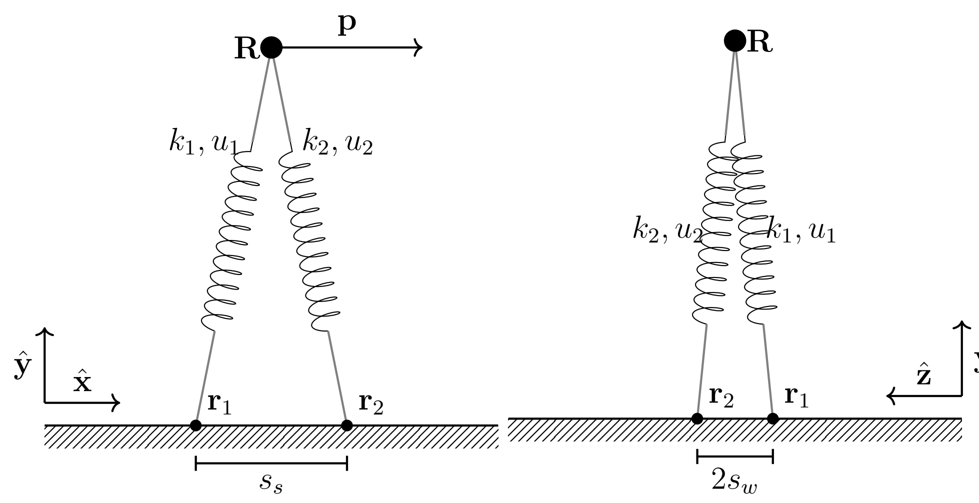

Figure 2.1: The setup of the three dimensional spring-mass model. The left figure shows a side view and the right figure a front view. The centre of mass is positioned at $\mathbf{R}$. The two springs are connected to the centre of mass and the ground at $\mathbf{r}_1$ and $\mathbf{r}_2$. The momentum is denoted by the vector $\mathbf{p}$. The ground contact points $\mathbf{r}_1$ and $\mathbf{r_2}$ differ in the x-direction one step size s_s and in the z-direction two step widths s_w.

2.2 Equations of Motion of a Spring-Mass System

Both the gravitational force and the spring force are conservative forces. Therefore there exists a potential $\mathcal{U}(\mathbf{R}) : \mathbb{R}^3 \to \mathbb{R}$ such that $\mathbf{F} = -\nabla\mathcal{U}$ [16]. It is clear that $\mathcal{U} = \mathcal{U}_s + \mathcal{U}_g$ due to the linearity of the gradient operator. Near the surface of the earth, the potential of gravity is: $\mathcal{U}_g(\mathbf{R}) = Mgy$, where $\mathbf{R} = (x, y, z)^\top$. The potential of a one dimensional spring is given by: $\frac{k}{2}(x - u)^2$ [15]. It is easily verified that this also works for the three-dimensional case:

$$\mathcal{U}_s(\mathbf{R}) = \frac{k}{2}(\|\mathbf{R} - \mathbf{r}\| - u)^2. \tag{2.2}$$

The next step is to find the potential of our complete system. This potential is given by:

$$\mathcal{U}(\mathbf{R}) = \mathcal{U}\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right) = \frac{k_1}{2}(\|\mathbf{R} - \mathbf{r}_1\| - u_1)^2 + \frac{k_2}{2}(\|\mathbf{R} - \mathbf{r}_2\| - u_2)^2 + Mgy. \tag{2.3}$$

Note that this expression is independent of time and velocity. This is as expected from a potential consisting solely of conservative forces. Moreover, taking the derivative and adding a minus sign returns the spring forces and gravitational force.

With the potential derived, the Hamiltonian of the system can be stated. The Hamiltonian is in this case just the total energy of the system. Also, the Hamiltonian is independent of time, implying that the total energy is conserved [15]. Finally, the kinetic energy of the point mass can be written as $\frac{\mathbf{p}\cdot\mathbf{p}}{2M}$ [16], where $\mathbf{p}$ is the momentum of the centre of mass. The Hamiltonian is:

$$\mathcal{H}(\mathbf{R},\mathbf{p}) = \mathcal{H}\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}\right) = \frac{\mathbf{p}\cdot\mathbf{p}}{2M} + \frac{k_1}{2}(\|\mathbf{R}-\mathbf{r}_1\|-u_1)^2 + \frac{k_2}{2}(\|\mathbf{R}-\mathbf{r}_2\|-u_2)^2 + Mgy \tag{2.4}$$

The equations of motions follow from Hamilton's equations (equations (2.5)) for the generalised coordinates and momenta [15]. Recall that the generalised coordinates are simply x, y, and z with corresponding momenta p_x, p_y, and p_z.

$$\dot{q}_i = \frac{\partial \mathcal{H}}{\partial p_i} \text{ and } \dot{p}_i = -\frac{\partial \mathcal{H}}{\partial q_i} \tag{2.5}$$

Therefore, the two vector equations of motion follow and completely describe the motion of the point mass. The equation expressing the derivative in momentum is:

$$\dot{\mathbf{R}} = \nabla_{\mathbf{p}}\mathcal{H} = \frac{\mathbf{p}}{M}. \tag{2.6}$$

This states that momentum is the product of velocity and mass. The remaining equation is:

$$\begin{aligned}\dot{\mathbf{p}} &= -\nabla_{\mathbf{R}}\mathcal{H} \\ &= -k_1\left(\|\mathbf{R}-\mathbf{r}_1\| - u_1\right)\frac{\mathbf{R}-\mathbf{r}_1}{\|\mathbf{R}-\mathbf{r}_1\|} - k_2\left(\|\mathbf{R}-\mathbf{r}_2\| - u_2\right)\frac{\mathbf{R}-\mathbf{r}_2}{\|\mathbf{R}-\mathbf{r}_2\|} - Mg\hat{\mathbf{y}}.\end{aligned} \tag{2.7}$$

The equations of motion now have been derived, but the model is not yet complete. For example, if the spring is extended, then the springs will pull the centre of mass towards the fixed end. However, this is of course not realistic. Generally when walking, feet are fixed in the (x, z)-plane and the feet can not move below ground. There is neither a friction force nor a normal force holding back an upwards movement.

2.3 Virtual Spring Constant

The springs should not apply a force to the centre of mass when they are extended. This has not been taken into account in the equation of motion, neither in the corresponding numerical derivations. This problem is solved by introducing virtual spring constants, one for the left spring and one for the right spring. The virtual spring constant has value zero if the distance between the centre of mass and the base vector of the spring is larger than the rest length:

$$k_v = \begin{cases} 0 & \text{if } \|\mathbf{R}(t) - \mathbf{r}(t)\| > u \\ k & \text{otherwise.} \end{cases} \tag{2.8}$$

On the trajectory where the position of the feet is fixed, the spring constant is a function of position. The spring constant is constant on the extended and contracted domains. There the derivative is zero. So if $\|\mathbf{R} - \mathbf{r}\| \neq u$, then the potential and the equations of motion will remain the same. $\|\mathbf{R} - \mathbf{r}\| = u$ is only a single point and will not impact a discretised solution.

2.4 Incorporating Movement by Taking Steps

The mass of the system is concentrated in the centre of mass. This means that the springs are massless. The fixed end $\mathbf{r}$ will therefore only move if the coordinates are changed, not as a result of the equations of motion. The model will assume that the steps are of equal size and on a rail in the forward x-direction. The rail consists of two lines, one for the left foot $\mathbf{r}_1$ and one for the right foot $\mathbf{r}_2$. Each line is spaced one step width s_w from the centre plane $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$. The step width is constant. On each line there are periodically spaced points that act as possible coordinates for each foot. These points are spaced two step sizes s_w apart. The actual location of the foot may only be on one coordinate at the time. The x-distance between the left and right foot is one step length s_s. The step size is predetermined.

A step is a translation of a foot from a location $\mathbf{r}$ to a new location two step lengths forward. The forward motion of all trajectories makes backwards steps unnecessary. A step should be performed only if the upcoming possible location is closer. Consider $\mathbf{r}(t)$ to be a function of time. Then that function must satisfy the following inequality:

$$||\mathbf{R} - \mathbf{r}|| < ||\mathbf{R} - (\mathbf{r} + 2\mathbf{s}_s)|| \quad \forall t > 0, \tag{2.9}$$

where $\mathbf{s}_s$ represents the step size vector $(s_s, 0, 0)^\top$. Steps are instantaneous and therefore not modelled. Equation (2.9) states that a step is taken as soon as the next possible position is closer to the position of the centre of mass than the current end of the spring. This condition is satisfied just after the distances are equal. That is, $||\mathbf{R}-\mathbf{r}|| = ||\mathbf{R}-(\mathbf{r}+2\mathbf{s}_s)||$. Hence the energy in the system is conserved. Figure 2.2 shows the layout of the rails and the connection with the possible ground connections points.

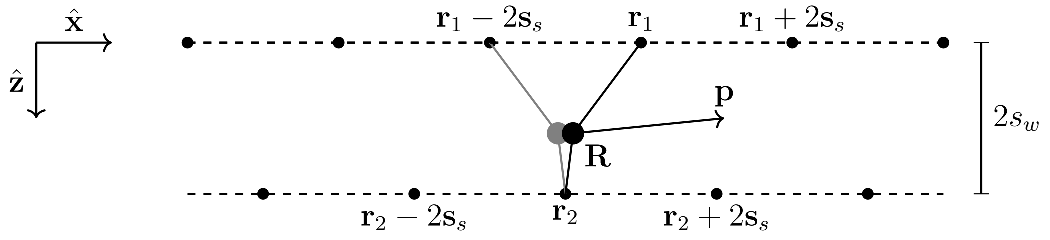
$\hat{\mathbf{x}}$
$\hat{\mathbf{z}}$
$\mathbf{r}_1 - 2\mathbf{s}_s$
$\mathbf{r}_1$
$\mathbf{r}_1 + 2\mathbf{s}_s$
$\mathbf{p}$
$\mathbf{R}$
$2s_w$
$\mathbf{r}_2 - 2\mathbf{s}_s$
$\mathbf{r}_2$
$\mathbf{r}_2 + 2\mathbf{s}_s$

Figure 2.2: Top view of the model with guide rails (dashed lines) for the left and right foot . The dots on these lines are the possible positions of the feet. The centre of mass is connected with a spring to the closest point on the left and right rail. The gray circle is the centre of mass a fraction of a second earlier. Then the left foot is still connected to the previous ground contact point. The vector $\mathbf{p}$ is the momentum of the centre of mass.

2.5 Periodic Properties of the Solution

There are also boundary conditions that have to be specified. We will use the expected periodic behaviour of the solution. Let T be the period time of a solution $\mathbf{R}, \mathbf{p}$ in seconds. That is, the time for a solution to reach the following criteria:

$$\begin{aligned} \mathbf{R}(t) &= \mathbf{R}(t+T) + 2\mathbf{s}_s, && \forall t \geq 0, \\ \mathbf{p}(t) &= \mathbf{p}(t+T), && \forall t \geq 0. \end{aligned} \tag{2.10}$$

These conditions state that the x-position is shifted by two step sizes s_s. Equation (2.9) states that the closest possible ground contact points are the locations for the feet. The possible locations are spaced $2\mathbf{s}_s$ apart. Hence the functions describing the location of the feet are also periodic:

$$\begin{aligned} \mathbf{r}_1(t) &= \mathbf{r}_1(t+T) + 2\mathbf{s}_s, && \forall t > 0, \\ \mathbf{r}_2(t) &= \mathbf{r}_2(t+T) + 2\mathbf{s}_s, && \forall t > 0. \end{aligned} \tag{2.11}$$

Let $\mathbf{R}_0 = (x_0, y_0, z_0)^\top$ and $\mathbf{p}_0 = (p_{x_0}, p_{y_0}, p_{z_0})^\top$ denote the initial conditions. Suppose that the solution corresponding to these initial conditions is determined. We assume that the trajectory has a monotone increasing x-coordinate. This is reasonable for a walking person. In order to verify periodicity only one point of that trajectory has to be compared with the initial conditions. This trajectory is periodic if for some $t > 0$ the values $\mathbf{R}_0 + 2\mathbf{s}_s$ and $\mathbf{p}_0$ are attained.

The valid trajectories have periodic properties that make a determination of periodicity of a trajectory in a shorter time period possible. A time span of $\frac{T}{2}$ is sufficient. The first half of the gait cycle should deviate by a reflection in the (x, y) plane and a translation by $\mathbf{s}_s$. So the following conditions determine periodicity:

$$\begin{aligned} x\left(\frac{T}{2}\right) &= x_0 + s_s, & p_x\left(\frac{T}{2}\right) &= p_{x_0}, \\ y\left(\frac{T}{2}\right) &= y_0, & p_y\left(\frac{T}{2}\right) &= p_{y_0}, \\ z\left(\frac{T}{2}\right) &= -z_0, & p_z\left(\frac{T}{2}\right) &= -p_{z_0}. \end{aligned} \tag{2.12}$$

These conditions will be used in the Implementation chapter to determine a scoring method for trajectories.

CHAPTER

THREE

Implementation

This chapter consists of three main parts. The first is about the numerical implementation of the derived equations of motion (2.6) and (2.7). The second part will set up two algorithms for finding optimal parameters. The first algorithm will find the optimal initial conditions y_0 and z_0 for periodicity of the trajectory and the second will find the optimal spring constant and rest length to fit a data set. The last part discusses the parameter estimation of the algorithms.

3.1 Numerical Methods

The numerical implementation is done with two well-known methods: central differences and Runge-Kutta integration. The benefit of the former is that the computation of the trajectory is less computationally intensive than the latter for equal time step sizes. The benefit of Runge-Kutta is that it is a fourth order method, whereas central differences is a second order method. The higher order implies that if the time step is sufficiently small and the machine precision sufficiently high, then Runge-Kutta integration will always have a smaller numerical error.

First the numerical approximation of equation (2.6) will be treated. Let Δt denote the step size in time and suppose the initial position $\mathbf{R}_0 = (x_0, y_0, z_0)^\top$ and the initial momentum $\mathbf{p}_0 = (p_{x_0}, p_{y_0}, p_{z_0})^\top$ are known. The initial position and the initial momentum are together the initial conditions. As an example, the x-coordinate will be worked out completely. The other coordinates y and z are similar. For x, the first order differential equation reads $\dot{R}_x^i = \frac{1}{M} p_x^i$ at time $t = i\Delta t$, with $i \in \mathbb{N}$. The central difference approximation [17] has been substituted for $\dot{R}_x^i$:

$$\frac{R_x^{i+1} - R_x^{i-1}}{2\Delta t} + \mathcal{O}((\Delta t)^2) = \frac{p_x^i}{M}, \quad \forall i \in \mathbb{N}. \tag{3.1}$$

Rewriting this gives the equation for R_x^{i+1}. The resulting vector equation is:

$$\mathbf{R}^{i+1} = \mathbf{R}^{i-1} + \frac{2\Delta t}{M}\mathbf{p}^i + \mathcal{O}((\Delta t)^2), \quad \forall i \in \mathbb{N}. \tag{3.2}$$

This equation is ill-defined for $i = 0$. This problem can be solved by applying Backwards Euler [17] for $i = 0$ to the differential equation (2.6). The resulting equation where the virtual point $\mathbf{R}^{-1}$ has been extracted is:

$$\mathbf{R}^{-1} = \mathbf{R}^0 - \frac{\Delta t}{M}\mathbf{p}^0 + \mathcal{O}(\Delta t). \tag{3.3}$$

Substituting this equation in equation 3.2 gives:

$$\mathbf{R}^1 = \mathbf{R}^0 - \frac{\Delta t}{M}\mathbf{p}^0 + \mathcal{O}(\Delta t) + \frac{2\Delta t}{M}\mathbf{p}^0 + \mathcal{O}((\Delta t)^2) = \mathbf{R}^0 + \frac{\Delta t}{M}\mathbf{p}^0 + \mathcal{O}(\Delta t). \tag{3.4}$$

Next is the derivation of the numerical expression of equation (2.7). In order to make the notation more manageable, the force function $\mathbf{F}(\mathbf{R}) : \mathbb{R}^3 \to \mathbb{R}^3$ is defined as:

$$\mathbf{F}(\mathbf{R}) = -k_1 \left(\|\mathbf{R} - \mathbf{r}_1\| - u_1\right) \frac{\mathbf{R} - \mathbf{r}_1}{\|\mathbf{R} - \mathbf{r}_1\|} - k_2 \left(\|\mathbf{R} - \mathbf{r}_2\| - u_2\right) \frac{\mathbf{R} - \mathbf{r}_2}{\|\mathbf{R} - \mathbf{r}_2\|} - Mg\hat{\mathbf{y}} \tag{3.5}$$

Equation (2.7) reduces with this simplified notation to $\dot{\mathbf{p}} = \mathbf{F}(\mathbf{R})$. The exact same method as for equation (2.6) can then be applied. So $\dot{\mathbf{p}}^i = \frac{1}{2\Delta t}(\mathbf{p}^{i+1} - \mathbf{p}^{i-1}) + \mathcal{O}((\Delta t)^2) = \mathbf{F}(\mathbf{R}^i)$. This gives the recursive expression for finding all $\mathbf{p}^i$:

$$\mathbf{p}^{i+1} = \mathbf{p}^{i-1} + 2\Delta t \mathbf{F}(\mathbf{R}^i) + \mathcal{O}((\Delta t)^2). \tag{3.6}$$

This equation is also not defined for $i = 0$. Backwards Euler is used to approximate the momentum at the virtual point $t = -\Delta t$. This first order approximation gives: $\mathbf{p}^{-1} = \mathbf{p}^0 - \Delta t \mathbf{f}(\mathbf{R}^0)$. By substitution we can find the relation between $\mathbf{p}^0$ and $\mathbf{p}^1$:

$$\mathbf{p}^1 = \mathbf{p}^0 - \Delta t \mathbf{F}(\mathbf{R^0}) + 2\Delta t \mathbf{F}(\mathbf{R}^0) + \mathcal{O}(\Delta t) = \mathbf{p}^0 + \Delta t \mathbf{F}(\mathbf{R}^0) + \mathcal{O}(\Delta t). \tag{3.7}$$

Apart from the method above, another option is to use the Runge-Kutta equations. To be more precise, the fourth order, classic, Runge-Kutta equations that, as the name suggest, have been contrived by Carl Runge and Wilhelm Kutta. As this model consists of three pairwise coupled differential equations, a modified version must be used. Equation 3.8 gives the iterative result which has been adapted from [18].

$$\begin{aligned}
\mathbf{R}^{n+1} &= \mathbf{R}^n + \frac{1}{6}(\mathbf{b}_1 + 2\mathbf{b}_2 + 2\mathbf{b}_3 + \mathbf{b}_4) + \mathcal{O}((\Delta t)^4) \\
\mathbf{p}^{n+1} &= \mathbf{p}^n + \frac{1}{6}(\mathbf{a}_1 + 2\mathbf{a}_2 + 2\mathbf{a}_3 + \mathbf{a}_4) + \mathcal{O}((\Delta t)^4)
\end{aligned} \tag{3.8}$$

In these equations, the eight dummy variables $\mathbf{a}_1, \ldots \mathbf{a}_4, \mathbf{b}_1 \ldots \mathbf{b}_4$ are determined each time step as follows:

$$
\begin{aligned}
\mathbf{a}_1 &= \Delta t \mathbf{F}(\mathbf{R}^n, \mathbf{p}^n), \\
\mathbf{b}_1 &= \frac{\Delta t}{M}\mathbf{p}^n, \\
\mathbf{a}_2 &= \Delta t \mathbf{F}(\mathbf{R}^n + \frac{\mathbf{b}_1}{2}, \mathbf{p}^n + \frac{\mathbf{a}_1}{2}), \\
\mathbf{b}_2 &= \frac{\Delta t}{M}(\mathbf{p}^n + \frac{\mathbf{a}_1}{2}), \\
\mathbf{a}_3 &= \Delta t \mathbf{F}(\mathbf{R}^n + \frac{\mathbf{b}_2}{2}, \mathbf{p}^n + \frac{\mathbf{a}_2}{2}), \\
\mathbf{b}_3 &= \frac{\Delta t}{M}(\mathbf{p}^n + \frac{\mathbf{a}_2}{2}), \\
\mathbf{a}_4 &= \Delta t \mathbf{F}(\mathbf{R}^n + \mathbf{b}_3, \mathbf{p}^n + \mathbf{a}_3), \\
\mathbf{b}_4 &= \frac{\Delta t}{M}(\mathbf{p}^n + \mathbf{a}_3).
\end{aligned}
\tag{3.9}
$$

3.2 Setup of Periodic Path Finding

The program is split into two parts: the periodic path finding algorithm (PPFA) which will be introduced this section and the data fitting algorithm (DFA). The PPFA has as goal the generation of synthetic data according to the spring mass model. The DFA should be able to fit the data generated by the PPFA, enabling the observation of numerical properties of the solution trajectory. Moreover, several relations between parameters can be determined with the PPFA such as between the spring constant k and rest length u and the initial conditions y_0 and z_0.

The PPFA will, given the spring constant k and the rest length u, find the optimal initial conditions y_0 and z_0. In this algorithm, the parameters for the step size s_s, step width s_w, and the initial momentum in the forward direction p_{x_0} are assumed to be known. A given search range is divided into linearly spaced points. Another option could be logarithmically spaced points. This alternative will barely bring any benefit, as only the first few iterations have a range spanning multiple orders of magnitude. Moreover, the first few iterations already require a fine mesh grid in order to locate the optimal parameter values (or optimal initial conditions). There has to be at least one point in the region around the optimal points such that it is not only within the region where the score map is monotone decreasing, but also the best scoring point. These demands require a fine mesh grid, negating the benefit a coarsely logarithmically spaced grid might have brought.

The simulation is run for all discretised points. The best initial condition is chosen and the space around that initial condition is discretised again (figure 3.1). One parameter defines the fineness of the mesh grid. A second parameter is the number of discretised steps around the optimal solution that becomes the range in the following iteration. These parameters together determine the rate of convergence of the solution, provided that the algorithm locates the optimal parameters. Setting the rate of convergence too high could result in the optimal solution becoming unobtainable. The iterative process keeps going until one of the stopping criteria is met.

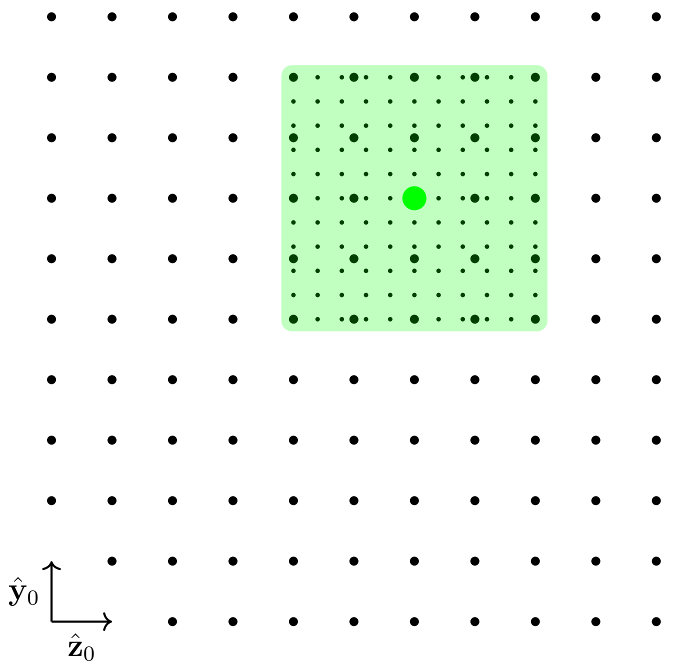

Figure 3.1: The discretisation of the parameter space (y_0, z_0) for the PPFA. The mesh discretisation equals 11 and the convergence rate is 2. The green square is the parameter space for the next iteration, which is 2 steps sizes in each direction from the best found solution (big green dot). The small black dots are the discretisation of the next iteration. Some of these coincide with the points of the previous discretisation.

3.3 Scoring Method of PPFA

The best initial condition should satisfy the earlier discussed periodic properties (equation (2.12)). As for each guess, the initial conditions are known, just like the phase space values at half a period T. The position at half a period is exactly one step length from the initial condition. Consequently there is a point in phase space ($\mathbf{R}_{\frac{T}{2}}$, $\mathbf{p}_{\frac{T}{2}}$) to compare the simulation against. The distance between this benchmark point and the closest point in the simulation is used to define a scoring algorithm. Closest is defined as the closest point in time. Because the x-trajectory is very linear with time, this is also the point with the lowest x-distance. This gives the following scoring function:

$$e_{\frac{T}{2}} = \left(\frac{||\mathbf{R}^{i_{\min}} - \mathbf{R}_{\frac{T}{2}}||^2}{||\hat{\mathbf{R}}_{\frac{T}{2}}||^2} + \frac{||\mathbf{v}^{i_{\min}} - \mathbf{v}_{\frac{T}{2}}||^2}{||\hat{\mathbf{v}}_{\frac{T}{2}}||^2} \right). \tag{3.10}$$

where $i_{\min}$ the index of the point closest in time to $\frac{T}{2}$. The velocity $\mathbf{v} = \frac{\mathbf{p}}{M}$ is used instead of the momentum to keep all coordinates roughly the same order. Otherwise there is a bias favouring the momentum. For zero distance the scoring method will result in a perfectly periodic solution to the differential equations, provided it exists. Furthermore, the discretisation of the initial condition space should consist of a sufficient number of points such that the best found point is in the monotone decreasing domain of the scoring function around the optimal point. There is a way to improve the method such that some of these imperfections are included.

A problem with the scoring method is the discretisation of the path. There will never be a point in the trajectory exactly at one step size. A solution could be to interpolate between the points around it. Then the numerical error in the x direction can be eliminated. However, this makes the simulation less stable. The reason is that this eliminates the effect of the numerical discretisation. This might seem preferred but the opposite holds true. The scoring method does not force the division of half a period in an integer amount of steps by not taking into account the numerical deviation in the forward direction. This results in an ever growing offset that causes instability. Note that the numerical deviation from the periodic trajectory is amplified because the trajectory is inherently unstable. The numerical approximations are either based on central differences or on Runge-Kutta integration. The central difference equations depend on the two previous points in time to calculate the upcoming point. Therefore only using a single point is not sufficient and two points must be taken into consideration. The second point is based on the first calculated point $\mathbf{R}^1$, $\mathbf{p}^1$. If these two points coincide perfectly with two points of the trajectory, then the trajectory must be periodic. In practise this is of course prohibited by the numerical error due to the finite time step. The second point $\mathbf{R}_{\frac{T}{2}+\Delta t}$, $\mathbf{p}_{\frac{T}{2}+\Delta t}$ is determined by using the same mirroring principles as with the initial conditions, but for the first calculated time step $\mathbf{R}^1$, $\mathbf{p}^1$. The scoring method then becomes:

$$e_{\text{PPFA}} = e_{\frac{T}{2}} + \frac{\|\mathbf{R}^{i_{\min}} - \mathbf{R}_{\frac{T}{2}+\Delta t}\|^2}{\|\hat{\mathbf{R}}_{\frac{T}{2}+\Delta t}\|^2} + \frac{\|\mathbf{v}^{i_{\min}} - \mathbf{v}_{\frac{T}{2}+\Delta t}\|^2}{\|\hat{\mathbf{v}}_{\frac{T}{2}+\Delta t}\|^2}, \tag{3.11}$$

where $i_{\min}$ is the point closest in time to the second benchmark point.

3.4 Setup of Data Fitting

The DFA has as goal to fit data to the model by finding the best spring constant k and rest length u. The implementation has many parts in common with PPFA. For example, the iterative mesh grid is set up the same but in (k, u) space. Also, the simulation is run the same way, as well as the stopping criteria. The DFA differs substantially from PPFA in the scoring algorithm. This is logical as the goal is different, namely fitting the model to data, instead of generating periodic data with the model.

Before discussing the scoring algorithm, the setup must be clarified a bit more. It is assumed that the data is not error free. In that case, it is not valid to assume that the initial conditions are perfect. Of course it would be impossible to perform a proper fit with an incorrect premise. A solution is to run PPFA (with modified scoring) to find the optimal initial conditions. Using PPFA in tandem with DGA has the additional benefit of forcing a periodic fit. However, this is computationally expensive. The doubling in fitting variables to also include y_0 and z_0 approximately squares the computational time as both use a similar algorithm. A more rational approach is to observe the data. If the data is smooth and periodic, then the fit will be already quite periodic. If the data is periodic but very noisy, then an approach would be to take the average of the first few values. This model is not applicable without a sufficient degree of periodicity of the data.

The score of a parameter guess will be determined by calculating the distance in phase space between every data point and the closest simulated point:

$$e_{\text{DFA}} = \sum_{j=0}^{n-1} \frac{||\mathbf{R}^{i_{\min,j}} - \mathbf{R}_d^j||^2}{||\hat{\mathbf{R}}_d^j||^2} + \frac{||\mathbf{v}^{i_{\min,j}} - \mathbf{v}_d^j||^2}{||\hat{\mathbf{v}}_d^j||^2}, \tag{3.12}$$

where n is the number of data points and $i_{\min,j}$ is the index of the simulation point that is closest in time to the j^{th} data point. Here it is essential that the density (in time) of simulated points is much greater than the density of data points. Only then is the expected distance between those points is negligible in comparison to the distances between the trajectories.

3.5 Termination Criteria

Once the fitting has commenced, it is required to specify the stopping criteria. Otherwise the program would keep trying to find more optimal points. These conditions are applicable for both PPFA and DFA. To begin, it is not necessary iterate beyond machine precision, which is approximately 14 decimal digits. As the rest length is in the order of metres and the spring constant in the order of kilo Newtons per metre, a value of 10^{-12}m and 10^{-9}Nm^{-1} respectively are sufficient to stay above machine precision. Similarly, the step size for DFA should be above 10^{-12}m for both y_0 and z_0. This is quite a theoretical limit. A fit to experimental data should never require such precision. Therefore, as there is only fitting on half a gait cycle, it should not even be necessary to make such fine adjustments to the model parameters or initial conditions. The next termination criterion addresses this task.

If better scoring parameters are found, then these parameters should provide a significant improvement over the previous best parameters. An improvement is considered significant if the new score is at least 0.1% lower than the previous score. This will reduce the number of iterations but comes at the cost of not finding the absolute best parameters.

Until now we have only considered termination criteria of the iterative phase of the model. In order to boost the efficiency of the program, individual simulations should also terminate eventually, preferably as soon as possible. To put an absolute upper bound on the length of simulations, a parameter N is included to define the maximum number of steps of a simulation. A value of $N = 5(\Delta t)^{-1}$ proves to be more than sufficient. This means that the maximum simulation duration is 5 seconds. This criterion is hardly ever used, as it should. Instead the following criteria defining the allowable range of the simulation will address simulations that are known to fail.

If an individual simulation is destined to fail, then there is no reason to waste computing power for it. Because the springs can only contract due to the virtual spring constant, once the centre of mass has moved too much laterally the program should terminate. This is when it has either moved one step width to the left or one step width to the right. The step width is the parameter that determines the lateral position z of the feet. Even more, if the y coordinate of the centre of mass is below zero, then there is also no more use in continuing as the model has fallen over. Too much lateral deviation is the primary termination criterion.

Finally, as there is only half a step to be simulated in the fitting phase, the program should terminate as soon as the forward x-coordinate is higher than one step length. However, if a complete simulation is run, then this condition should be inactive.

3.6 Parameter Estimation

The algorithms introduced in the previous sections come with many parameters. Some assumptions have to be made in order to reduce the the number of parameters. A search range of these parameters also has to be established. Otherwise the program would be computationally too intensive. For both the PPFA and DFA, the assumption of equal legs is made. There is only one spring constant and one rest length. This assumption comes especially in hand with PPFA, halving the amount of parameters that have to be estimated. The next section will discuss the bounds on the spring constant and rest length for the DFA.

There are four degrees of freedom regarding the springs. Moreover, there are six degrees of freedom in the initial condition. It is crucial to determine sufficient bounds on the parameters to enable completion in an acceptable amount of time. For example each parameter is discretised in (only) ten equally spaced values, 10^{10} simulations would have to be run. We will begin by stating the bounds on all parameters, after which we will assign numerical values to these bounds.

First note that the springs are equivalent in all but the fixed coordinate. We will assert that u is in $[u_{\min}, u_{\max}]$. Clearly the spring constant must be at least $\frac{Mg}{2u}$, otherwise the centre of mass will never be able to overcome gravity and fall down immediately. This gives a lower bound for the spring constant of $\frac{Mg}{2u_{\max}}$. This bound can be improved a bit by taking the angle in between the spring into consideration. However, the marginal gains are negligible and therefore deemed superfluous. The range for the spring constant is bounded from above by a value $k_{\max}$.

Table 3.1 gives an overview of the parameters for DFA. Note that these ranges are still quite broad but manageable as only two parameters have to be determined. PPFA, on the other hand, consists of six parameters. Hence the ranges should be very small, or the number of parameters should be reduced significantly. The latter turns out to be the case, as will be discussed in the next paragraph.

Now the parameters of the PPFA will be discussed. For continued periodic locomotion, the center of mass will traverse every value of x, hence we choose $x_0 = 0$. Moreover, we fix the step-length at s_s, with lateral width s_w. Then $\mathbf{r}_1 = (0, 0, -s_w)^\top$ and $\mathbf{r}_2 = (s_s, 0, s_w)^\top$. It is an assumption that the initial coordinates have the same x-coordinate as the left foot $\mathbf{r}_1$.

Table 3.1: The parameters of the Data Fitting Algorithm of the spring mass model with the given search ranges.

	Parameter	Reduced	Minimum value	Maximum value
Spring constant [Nm^{-1}]	k_1	k	$\frac{Mg}{2u_{\max}}$	$k_{\max}$
	k_2	k	$\frac{Mg}{2u_{\max}}$	$k_{\max}$
Rest length [m]	u_1	u	0	$u_{\max}$
	u_2	u	0	$u_{\max}$

Continuing, we will assume that the centre of mass is initially at the left most point of the lateral cycle. That is, $z_0 = z_{\min}$. For stability, it must be that z is always in between the fixed coordinates of the legs. That is, $z \in [-s_w, s_w]$. Therefore, $z_{\min} \in [-s_w, 0]$. As the left most point is an extreme value, it must be that $p_{z_0} = 0$ due to equation (2.6). We will first assume that p_{y_0} is 0 here as well: the y-coordinate attains an extreme value here. However, this is not necessarily the case and therefore has to be verified with the data. In any case the extreme values should coincide closely, keeping the assumption of no offset valid. For p_{x_0}, if the velocity is high enough, the average momentum $\overline{p}_x$ is an adequate estimator of the initial momentum. After fitting, the simulation can be run again with a different velocity if the modelled average velocity deviates too much from the data.

To summarize, the initial conditions are $(0, y_0, z_0)^\top$ and $(\bar{p}_x, 0, 0)^\top$ with y_0 and z_0 in a bounded range. It would make sense if p_{x_0} would be an extreme value as well. Table 3.2 gives an overview of the ranges corresponding to each initial condition.

Table 3.2: The parameters of the Periodic Path Finding Algorithm of the spring mass model with the given ranges or values.

	Parameters	Minimum value	Maximum value
Position [m]	x_0	0	0
	y_0	$y_{\min}$	$y_{\max}$
	z_0	$-s_w$	0
Momentum [kgms^{-1}]	p_{x_0}	$\overline{p}_x$	$\overline{p}_x$
	p_{y_0}	0	0
	p_{z_0}	0	0

The four parameters have been halved for the DFA and reduced by a factor of three for the PPFA. All remaining parameters have been assigned a range in which the optimal parameter has to be located. These ranges have yet to be determined variables, namely $k_{\max}$ and $u_{\max}$ for the DFA and $y_{\min}$ and $y_{\max}$ for the PPFA. However, the exact choices of these variables are not that important. If the optimal parameters are excluded from the range, then the model will only find fits with a high error. Moreover, it could be that the optimal found parameters are outside of the initial search ranges. This is an indication that the model is not able to locate a proper fit.

CHAPTER

FOUR

Data

This chapter treats the acquisition and processing of the data of the 2019 paper [10] by Schreiber and Moissenet.

4.1 A Data set of 100 Self-set Gait Cycles

The paper measured the gait of 50 participants. These participants completed a series of walking trials on a treadmill. The velocity of the treadmill was always fixed, but in some trials it was prescribed while in others it was self-set. As this research investigates natural motion, only the self-set motion is taken into account.

The self-set motion measurements sets are split into two velocities. The first is set as normal by a healthy asymptomatic participant and the other is defined as fast. The expectation is that the model is better fitted to the faster prescribed motion. The reason is that the higher velocity results in shorter, more intense contact with the ground. That allows for less active mid step corrections.

The provided data set is primarily a collection of c3d files which can be imported into Python by the ezc3d module [19]. Each c3d file in the data set is of a specific participant at a specified velocity and only contains the gait-cycle of 52 markers scattered all over the body. Figure 4.1 shows the position of all markers placed. Each individual data set is the average of multiple gait-cycles. A pressure sensor is used to define the start and end of a gait cycle by determining events. The initial touchdown of a foot can, for example, be considered such an event. All gait-cycles of a measurement set can with the event data be transposed onto itself, yielding the single gait cycle. Finally, a fourth order Butterworth filter [20] with a 6Hz cutoff frequency has been applied to the result. The Butterworth has been constructed to eliminate higher frequencies.

Apart from the c3d files containing the trajectories, there is a file that includes relevant properties of the participants such as the height and mass of the participant. The mass is essential, as it is a variable in the derived equations of motion (2.6) and (2.7). The height is important because it allows the estimation of the vertical position of the centre of mass of that participant. When standing erect, the position of the centre of mass is approximately 56% of ones height [21]. Moreover, the length of the legs have been measured.

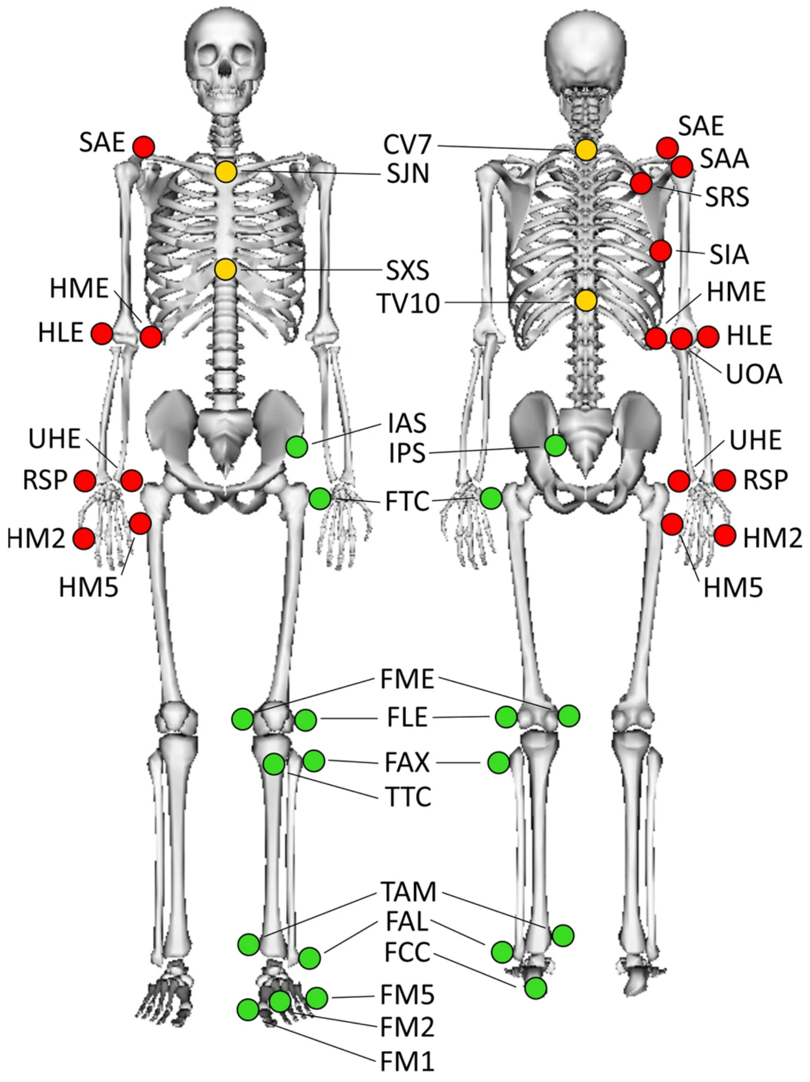

Figure 4.1: The markers placed over the participants bodies with corresponding names. This research will use the IPS and IAS markers [10].

The difference between the two legs range from no difference at all to over two centimeters. It seems likely that a large difference will lead to an asymmetric gait cycle.

4.2 Processing of the Data

The centre of mass has to be determined from the available markers. It will be calculated by taking an average of the left- and right-, anterior- and posterior-superior iliac spine. The anterior- and posterior-superior iliac spine are represented by IAS and IPS in figure 4.1. The markers are quite rigid with respect to the centre of mass. Taking the average will definitely result in a correct z-trajectory. The trajectory of the y-coordinate could deviate by a constant, but the shape of the trajectory and the momentum will be correct. Hence the position of the centre of mass is given by:

$$\mathbf{R}_d^i = \frac{1}{4}(\text{LIPS}^i + \text{RIPS}^i + \text{LIAS}^i + \text{RIAS}^i). \tag{4.1}$$

The height can be verified by the aforementioned 56%.

With the single gait cycle data sets the next step is to process it such that the model can be fitted to it. The model works best on about half a gait cycle beginning at the left most lateral (z-axis) displacement. This can be achieved by a translation of the data, where the minimum and maximum points define half a gait-cycle. Because the z-trajectory can be quite aperiodic, the extreme values of the y-trajectory will also be taken into account. Figures 4.2, 4.3, and 4.4 display the processed data for several people. The data has been processed by inverting the x-axis and subtracting the mean value of a half gait cycle of the z-position from the z-values. Apart from that the x-data is translated such that $x_0 = 0$m. The unprocessed data spans between one and two gait cycles. The method only requires a half gait cycle. The DFA can use more than a half gait cycle but due to deviations in feet placement this is not possible. Then the assumption of constant step size and step width is no longer accurate, which is necessary for the model. Moreover, then it is not longer possible to subtract the mean of the z-trajectory. Using less than half a gait cycle will limit the fitting capabilities. For each participant the most periodic half gait cycle has been selected. Sometimes this is from a right foot to a left foot as support, in which case the z-axis is inverted such that the initial z-coordinate is negative. The first and last data points are set to be as close to the extreme y- and z- values as possible. This way the assumption of aligning feet and extreme values can be verified or disproved.

The momentum is calculated with the position data and the Forward Euler method:

$$\mathbf{p}_d^i = M\frac{\mathbf{R}_d^{i+1} - \mathbf{R}_d^i}{\Delta t}, \tag{4.2}$$

where $\Delta t = 0.01$s is the time step of the data. It is not necessary to account for the case $i = n$ (recall that n is the size of the data set) because only the best half gait cycle is used, which in the cases of table 4.1 does not include the endpoints.

Table 4.1: Several key characteristics of participants of the gait cycle analysis study [10].

General				Lengths		
ID	M [kg]	Sex	age [years]	Height [m]	Left leg [m]	Right leg [m]
1	74.0	man	21	1.78	0.840	0.833
2	98.0	man	67	1.83	0.855	0.874
3	60.5	woman	41	1.67	0.805	0.806
4	61.9	woman	28	1.69	0.750	0.758

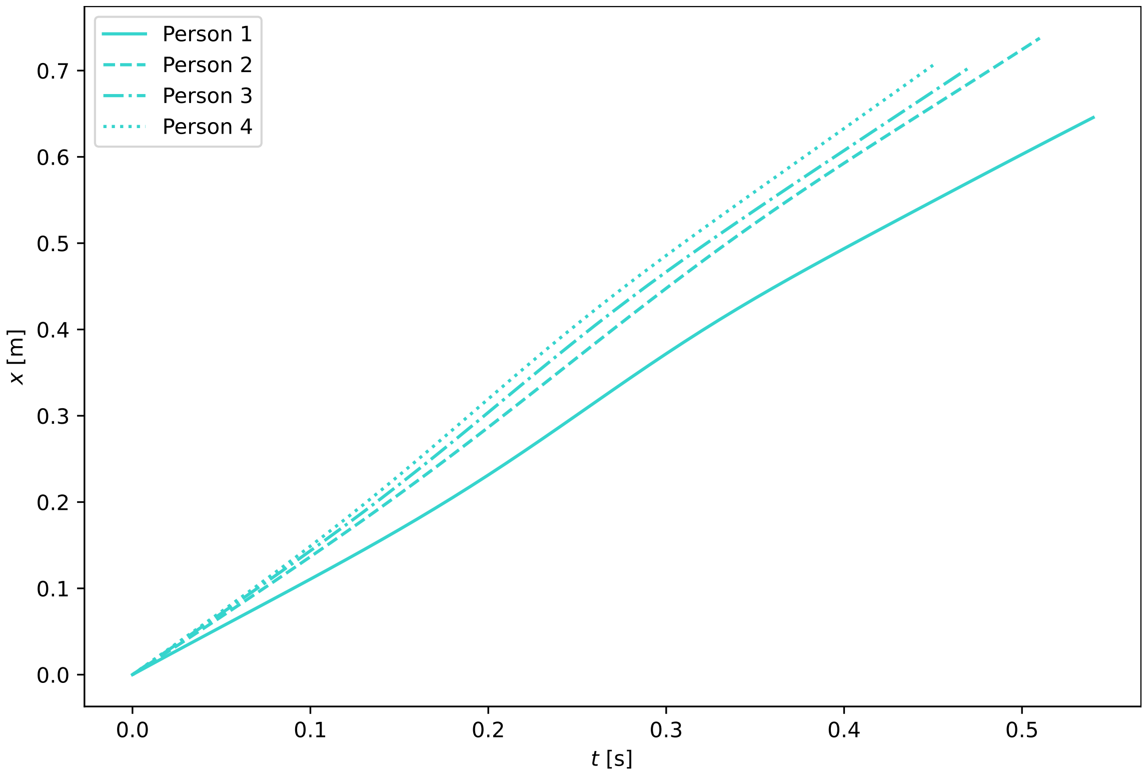

Figure 4.2: The x-coordinate of the processed data of the centre of mass of several people walking at speeds self defined as fast for a single step plotted against time t. Data obtained from [10].

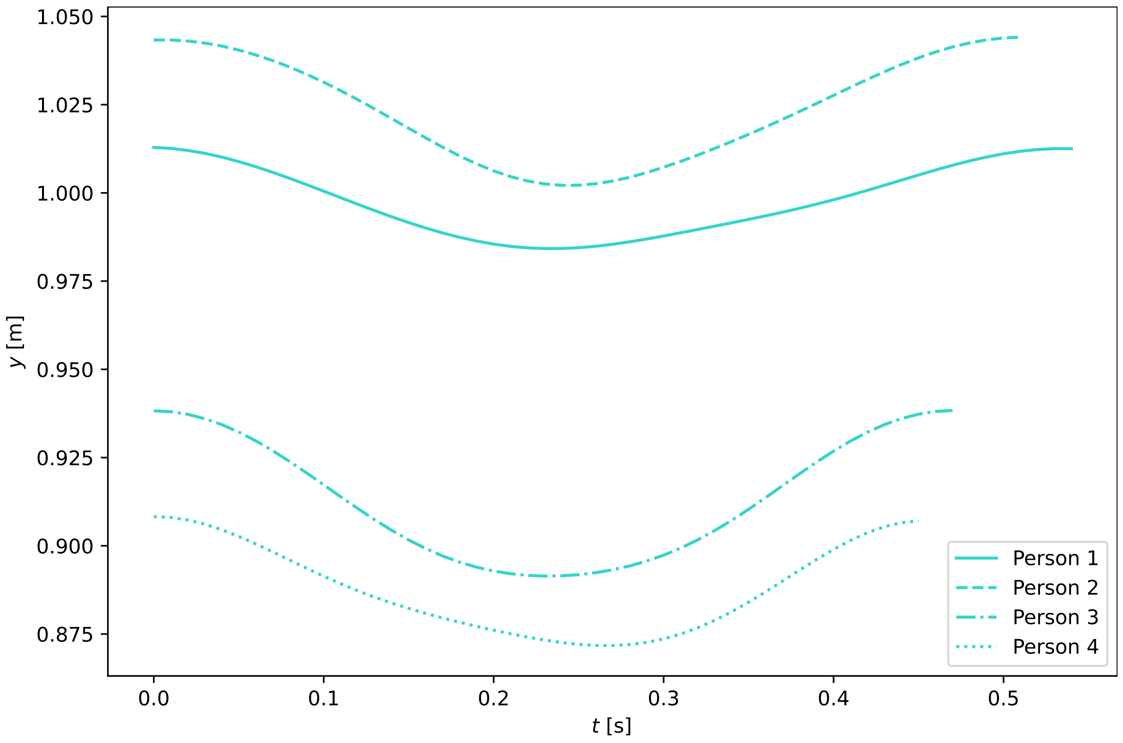

Figure 4.3: The y-coordinate of the processed data of the centre of mass of several people walking at self-set speeds for one step which is half the period $\frac{T}{2}$ plotted against time t. Data obtained from [10].

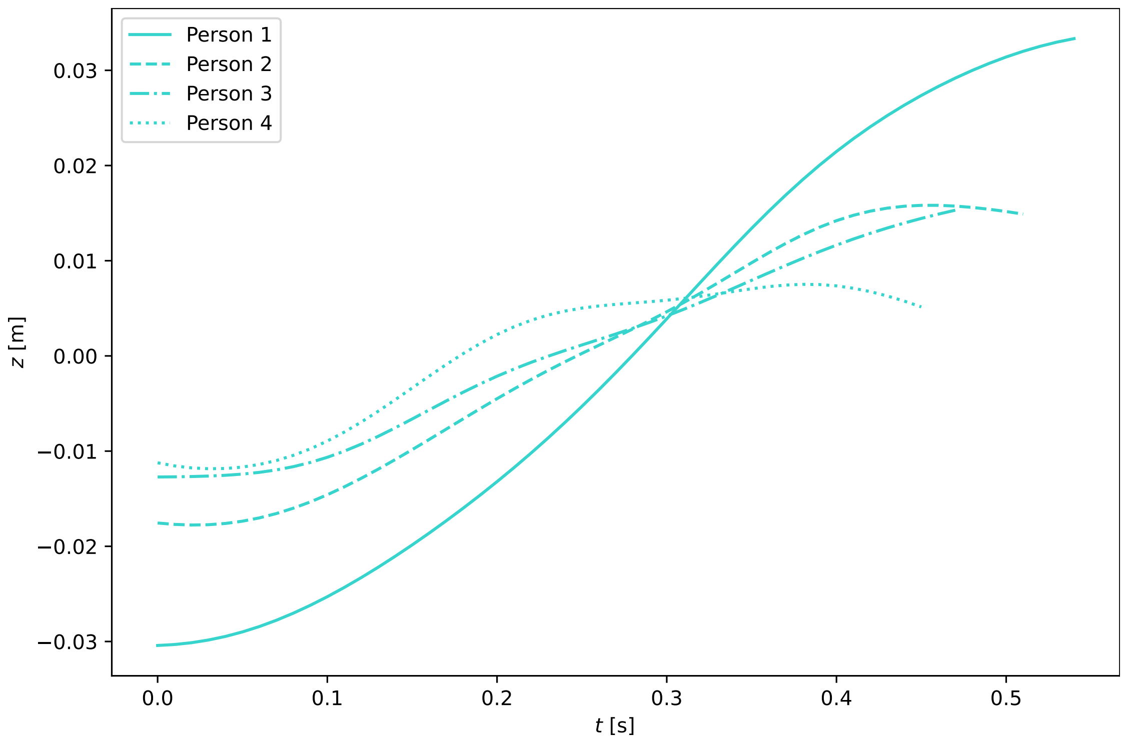

Figure 4.4: The z-coordinate of the processed data of the centre of mass of several people walking at self-set speeds plotted against time t. The data has been processed such that the initial z-position is negative. Data obtained from [10].

Although the half gait cycles are similar, there are some differences. The lateral position of person 1 is the most periodic and of person 4 the least periodic. On the other hand, the x-position of person 1 shows the greatest deviation. This implies that the momentum of person 1 is the least constant. Finally, the y-trajectories of persons 2 and 3 are the most periodic. All y-trajectories attain the value of 56% of the height of a person at some point in time. Hence we can conclude that the data tracks the centre of mass sufficiently. For the fitting of the model the data of person 1 will be used. Persons 2 and 3 are also viable options. The model can not fit to the data set of person 4 because the periodicity of the z-trajectory is too low. This is a significant limitation of the model.

CHAPTER

FIVE

Results and Discussion

This chapter consists of two sections. The first section discusses the PPFA and how it improves the understanding of the behaviour of the model. The second section discusses the verification of the model with synthetic data of the PPFA and the fitting to real data of the individual introduced in chapter 4, to test the feasibility of the model in practise.

5.1 Characteristics of the Model

In this section we first model the situation of a professional marathon runner. Key parameters include an initial forward speed of $v_{x_0} = 5.83\text{ms}^{-1}$. This comes from the average pace of the world record marathon run. The mass M is 52kg. The spring constant is chosen as $k_1 = k_2 = 15\text{kNm}^{-1}$ and the rest length of the spring is set to $u_1 = u_2 = 0.90\text{m}$. The full set of parameters is stated in table 5.1. The trajectory of the x-component is an elementary straight line due to the high velocity of the centre of mass (figure A.1). The trajectories for the y- and z- position are more interesting.

The y- and z- position of the optimal periodic path are shown in figures 5.1 and 5.2 respectively. The figures show optimised paths for several time steps, all executed with central differences and Runge-Kutta integration. These figures show the entire simulations until they are unstable. Only the first half step is modelled in the iterative fitting process. This shows that all time steps eventually become unstable. A smaller time step does result in a more stable trajectory. The time step $\Delta t = 0.01\text{s}$ shows a major deviation in the z-direction, even in the fitting range of half a step. On the other hand, the time step $\Delta t = 0.001\text{s}$ is much more stable in the y-direction. However, it only manages to be stable for one gait cycle in the z-direction. Even then it shows significant deviation from a periodic trajectory. The smallest time step computed is $\Delta t = 0.0001\text{s}$. This time step performs by far the best, managing five steps with high periodicity. The gains of reducing the time step by an order of magnitude do not yield even close to an order of magnitude improvement in stability. This confirms the expected chaotic behaviour of the system. The results for the time steps have been computed using both central differences and Runge-Kutta integration. Only $\Delta t = 0.01\text{s}$ shows a major difference between the two in figure 5.2. These trajectories differ significantly. Another difference is the numerical stability of the solution. Central difference shows a larger variation than Runge-Kutta integration. These oscillations are especially present at higher values for t. The optimal trajectory is physically unstable. Hence small numerical deviations amplify, resulting in oscillations around the unstable solution.

Table 5.1: The parameters of the PPFA for the marathon runner case. The time step Δt varies per fit.

	Parameter	Value or range	Unit
Springs	k	15000	[Nm^{-1}]
	u	0.90	[m]
System	M	52	[kg]
	s_s	1.91	[m]
	s_w	0.10	[m]
	Δs	0.00	[m]
	$\mathbf{p}_0$	$M \cdot (5.83, 0, 0)^\top$	[kgms^{-1}]
Numerical	Δt	$[0.1, 0.0001]$	[s]
	N	$5(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences and Runge-Kutta integration	
	minimum improvement	0.001	[-]
Search range	y_0	$[0.6, 1.0]$	[m]
	z_0	$[-s_w, 0]$	[m]

Figure 5.1 and 5.2 show that there is a difference in the initial conditions. This difference is visible in the (y,t)-plot, but only when $\Delta t = 0.01$s is compared with the other time step sizes. The deviations are more pronounced in the (z,t)-plot, where it is visible for all time step sizes.

The added precision of a smaller time step requires additional computational power. Runge-Kutta integration requires consistently more computing power than Central differences (figure A.2). As other analysis of the model demands hundreds of simulations to be run, there has to be made a trade off. It is most important that the first half step, the fitting range, shows no numerical oscillations. Therefore $\Delta t = 0.01$s is rejected. The Runge-Kutta method does not provide an additional benefit over central differences in the first half gait cycle for smaller time step sizes. At last, time steps smaller than 0.0005s require too much computational power, while providing no significant benefit in the first half of the gait cycle.

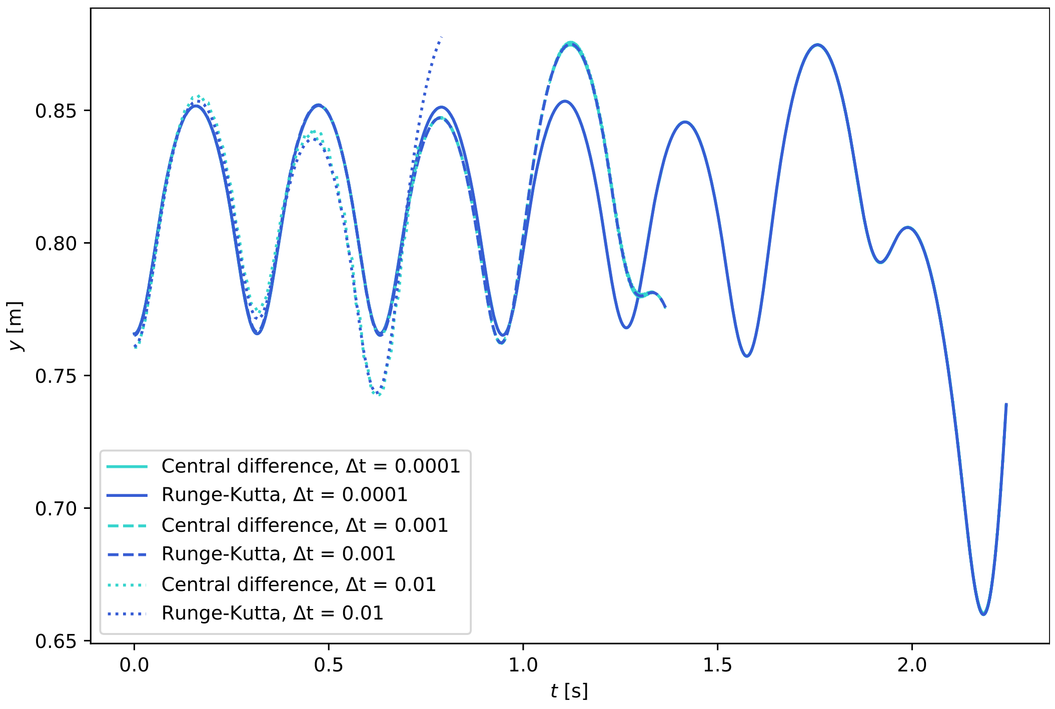

Figure 5.1: An hefty decrease in time step size results in a small increase in stable simulation length. The y-values of the optimal trajectories for the marathon running case. the vertical axis is the height y and the horizontal axis is time t. The trajectories have been plotted using both central differences as well as Runge-Kutta integration for different time steps. The spring constant has value 15kNm^{-1} and the rest length has value 0.90m.

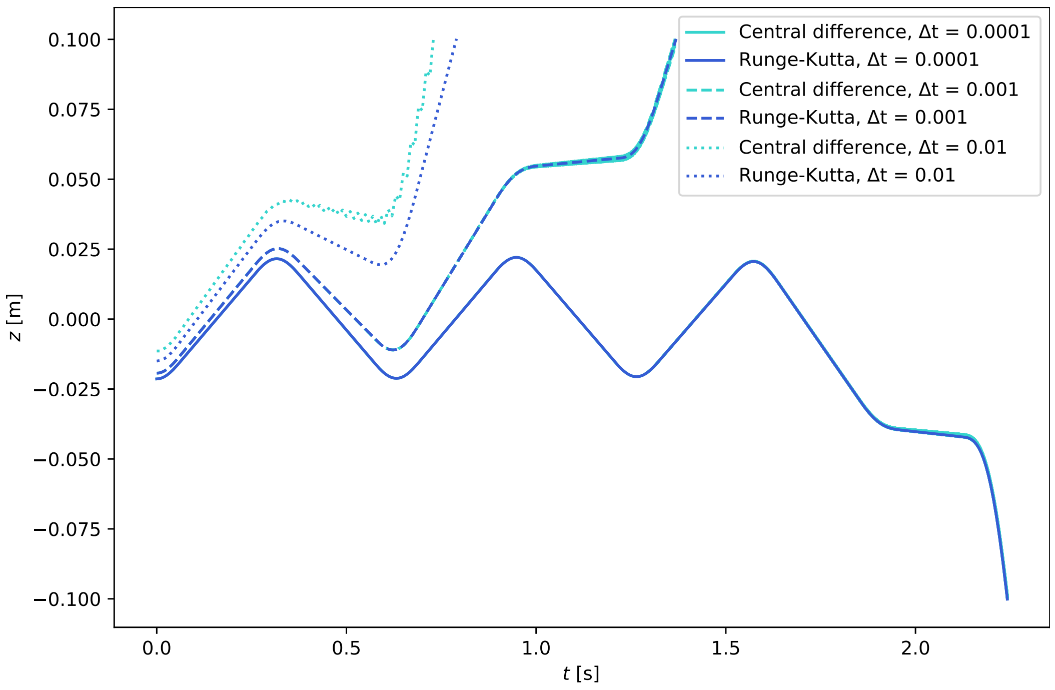

Figure 5.2: The z-values of the optimal trajectories for the marathon running case plotted against the z-position and the time axis t. The trajectories have been plotted for both central differences as well as Runge-Kutta integration and for different time steps.

Initially the spring constant k was fixed to a value of 15kNm^{-1}. Now that a proper time step and method have been chosen, this assumption can be relaxed. Then it becomes possible to study the behaviour of the initial conditions as a function of the spring constant. By adhering to the condition that the y-coordinate of the centre of mass of a person at rest is 56% of a persons height, the rest length of the spring is calculated:

$$u(k) = 0.56l + \frac{Mg}{2k}, \tag{5.1}$$

where l is the length of the person, which is set to 1.67m. The spring constant k spans a search range from 2kNm^{-1} to 20kNm^{-1}. Table B.1 provides all parameters. Figure 5.3 shows the relation between the fitted initial conditions and the spring constant. The initial height of the centre of mass appears to converge as k goes to infinity. This is a consequence of two elements. The first is the convergence of the equation of the rest length (equation 5.1) as k tends towards infinity. The second is the behaviour of the spring in this limit case. As the spring constant increases, the required deviation for a certain force decreases. Therefore the initial height must converge towards rest length, otherwise the spring force would be infinite. The initial height y_0 will not exactly converge towards u as z_0 also has to be taken into account. Hence $\lim_{k\to\infty} \|(0, y_0, z_0)\| = u$.

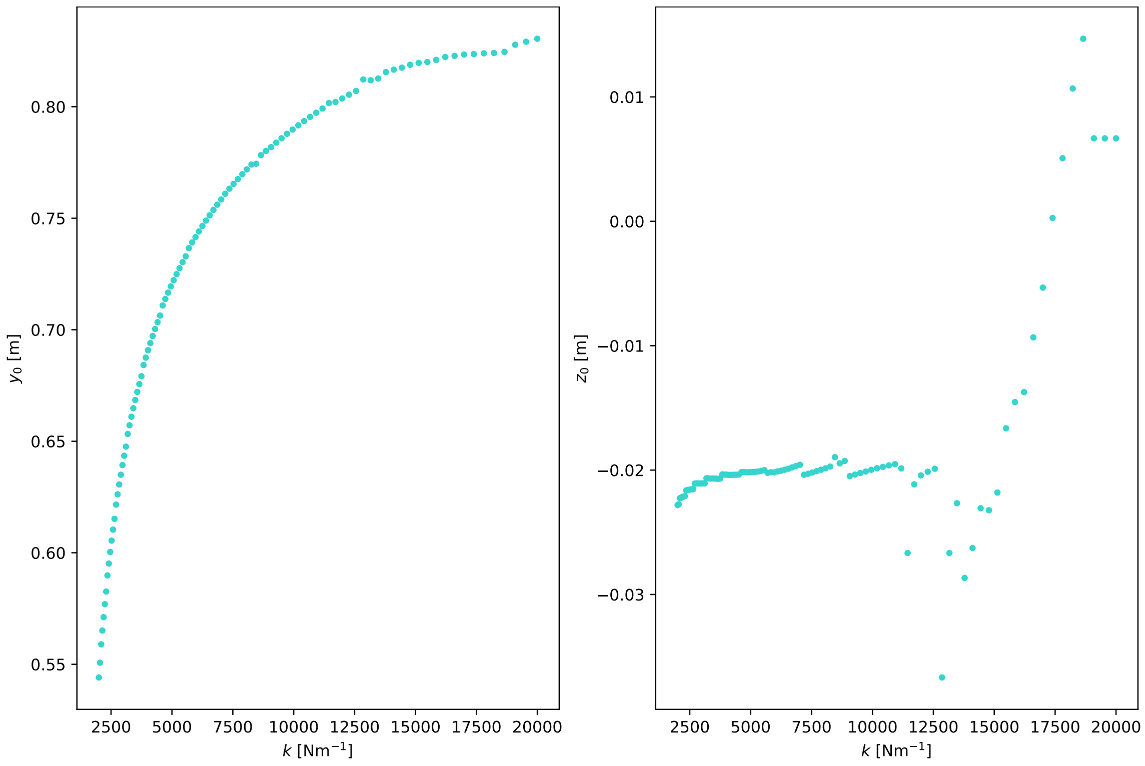

Figure 5.3: The relation between the spring constant k (plotted on the horizontal axis) and the initial conditions. The left figure plots the initial height y_0 and the right figure plots the initial lateral displacement z_0 on the vertical axis. The rest length u is determined by: $0.56l + \frac{Mg}{2k}$, where $l = 1.67\text{m}$.

The relation between k and z_0 is displayed in the right plot of figure 5.3. The plot shows a relatively constant trend until k equals 12500Nm^{-1}. Then the optimal value for the initial height z_0 changes rapidly, eventually even becoming larger than 0. This shows that this range of k values is non-physical because it is impossible for the system to be stable if $z_0 > 0$m. This is confirmed by the error plot in figure 5.4 (and figure A.3) which shows a severe increase in the error for values of k above 12500Nm^{-1} Hence the regime of feasible solutions is bounded. Not every combination of k and u has an initial condition that results in a periodic trajectory. Moreover, there are oscillations present in region $k < 12500$Nm^{-1}. Changing the spring constant changes the time until the benchmark points are reached. Which discretised simulation point is closest changes then, resulting in oscillations.

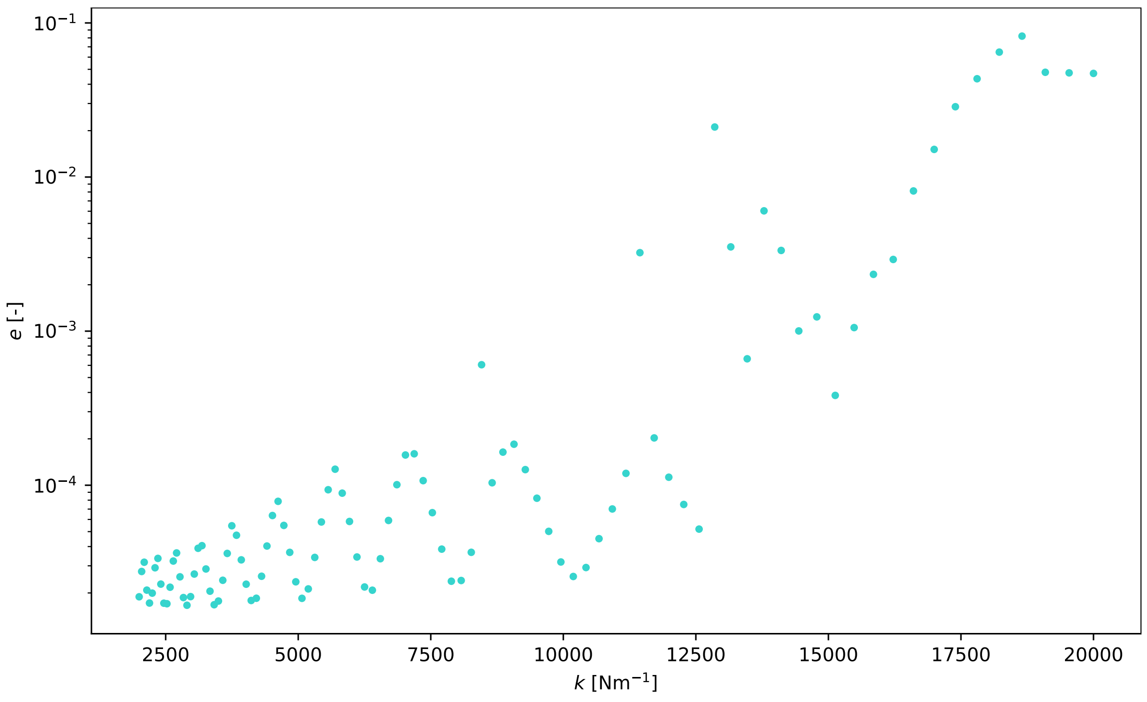

Figure 5.4: The error e of the simulation of the marathon runner. The error is plotted on a logarithmic scale against the spring constant k. The error shows a major increase when $k \geq 12500$Nm^{-1} and oscillations for lower values.

The trajectory of the y-position in figure 5.1 does not match the data in figure 4.3. Where the data set has a maximum at $z = z_{\text{min}}$ and $z = z_{\text{max}}$, the simulation has a minimum. On the other hand, the data has a minimum at $z = 0$, while the simulation has a crest. This could pose to be major issue that will be addressed in the next simulation. A y-trajectory where the maximum occurs at $z = 0$ is referred to as a positive trajectory. A y-trajectory where the maximum occurs at $z = z_{\text{max}}$ and $z = z_{\text{min}}$ is called a negative trajectory.

If the spring constant can be varied, so can the rest length. Equation 5.1 will not be used this time. Otherwise the generated data would be nearly identical, at most consisting of a different range. Moreover, this simulation will simulate a person walking. The spring constant is set to 4kNm^{-1} and the rest length u ranges from 0.7m to 1.2m. Other parameters have been changed to simulate the modelling of a person walking. The main differences are a much shorter step size $s_s = 0.66$m, a wider lateral step width $s_w = 0.2$m, and a lower initial speed of 1.2ms^{-1}. The complete set of parameters is in table B.2.

The initial conditions y_0 and z_0 are given as a function of the rest length u in figure 5.5. These figures have a much more consistent trend line than when varying the spring constant. There are five points that deviate from the trend line. Four are cluttered together around $u = 0.75$m. Figure 5.6 shows that these points result in a much shorter stable simulation. Therefore these deviations can be the result from the mesh grid being too coarse. The last one is a sole point at approximately $u = 1.16$m. Even though the sole point deviates clearly from the pattern, it still runs for quite some time (but the error does deviate, figure A.4). Rerunning the simulation with a finer mesh grid shows that the iterative method has failed when the mesh discretisation is 11 and the convergence rate is 2. This figure does not provide an explanation for the transition of y_0 being either a maximum or a minimum as there is no sudden change.

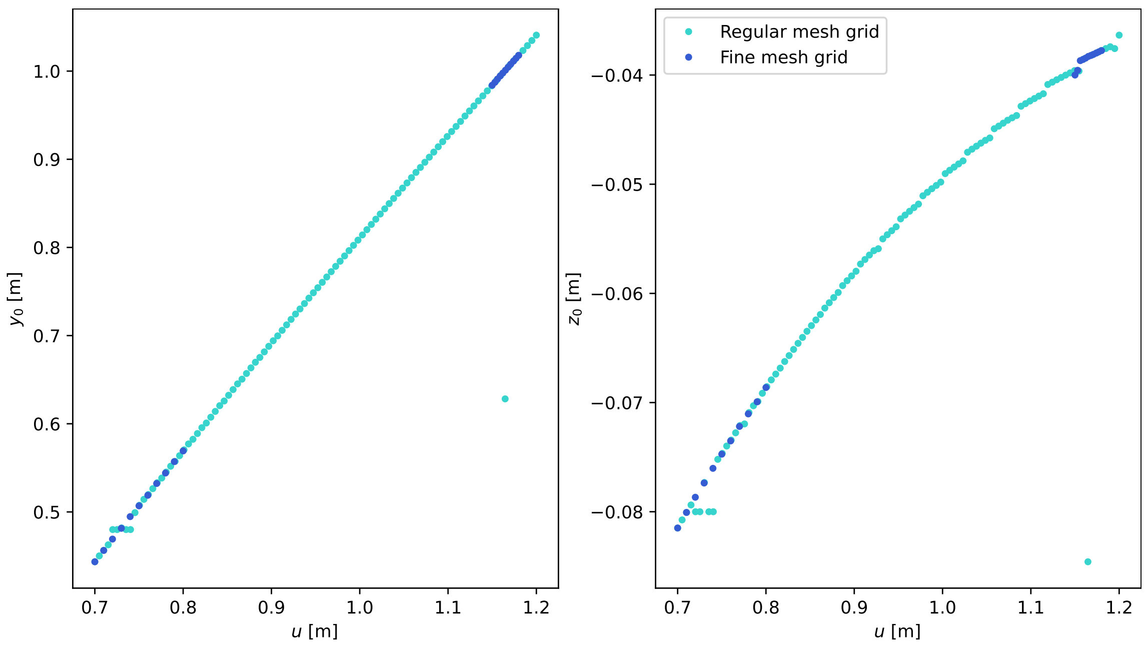

Figure 5.5: The initial conditions y_0 and z_0 plotted on the y-axis against the rest length. The fine mesh grid has double the mesh discretisation (22) and convergence rate (4) of the regular mesh grid.

Figure 5.7 does provide some insight. This figure shows the difference between the y-values at $t = 0.1$s and $t = 0$s. If this value is positive then y_0 is a minimum. If the value is negative, then y_0 is a maximum. The y-trajectory is either consistently higher or consistently lower than y_0 in the first half of the gait cycle. Hence the indicator $y^{33} - y^0$ is valid up to a period of $T = 0.2$s. For higher stride frequencies the difference used in the indicator has to be smaller. This figure shows that the transition from a maximum to a minimum is a gradual process. The deviating points visible once again.

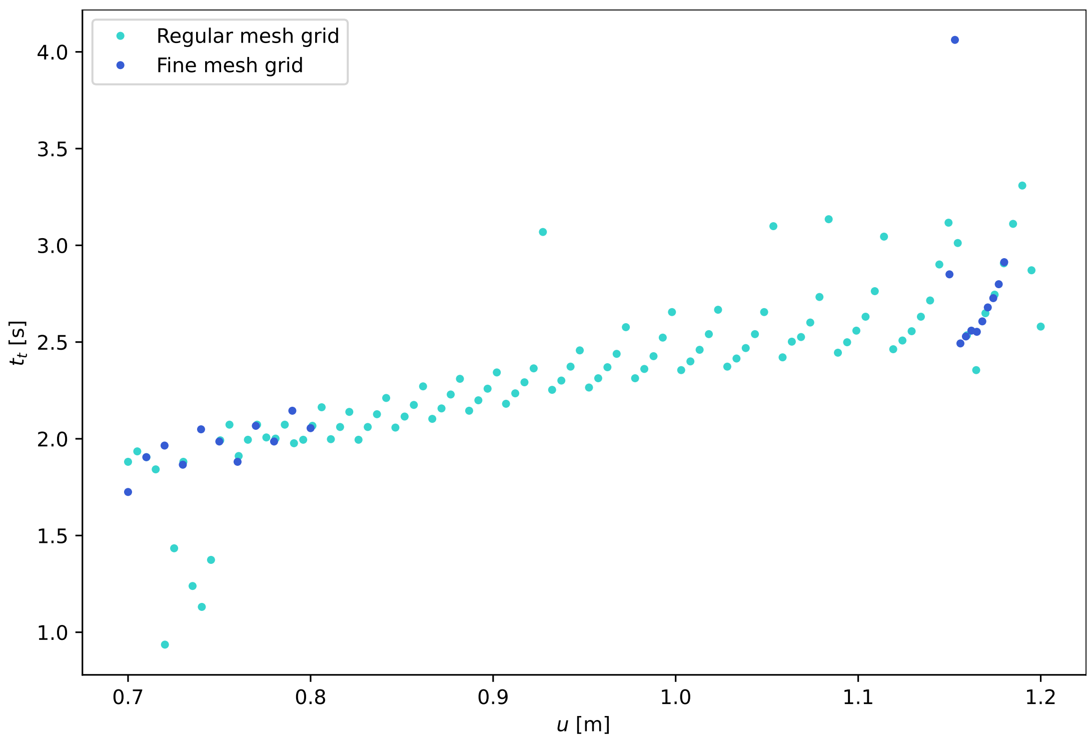

Figure 5.6: The termination time t_t of each simulation as a function of the rest length u. The fine mesh grid has double the mesh discretisation (22) and convergence rate (4) of the regular mesh grid (11 and 2).

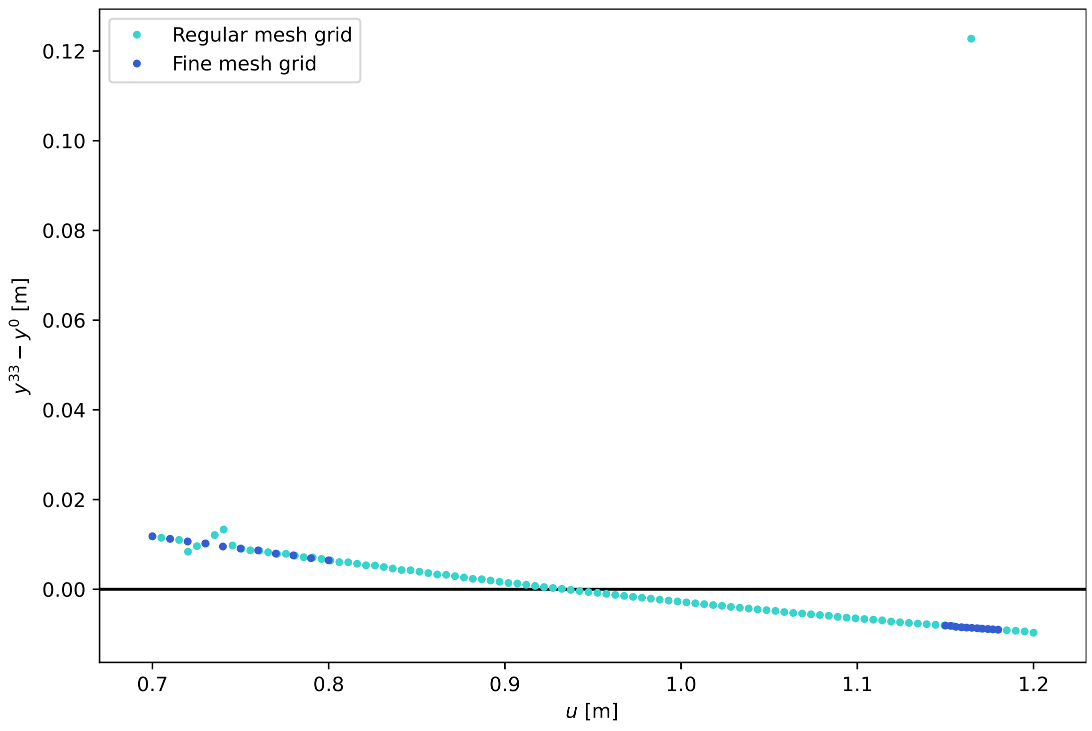

Figure 5.7: The difference between the y-position of the centre of mass of the walking person simulation at $t = 0.1$s and $t = 0$s. The spring constant is fixed at 4kNm^{-1}. The horizontal axis is the rest length u, which is given by the solid black line. The fine mesh grid has double the mesh discretisation (22) and convergence rate (4) of the regular mesh grid (11 and 2).

The optimal y-trajectory for several values of the rest length u is shown in figure 5.8. y_0 is at $t = 0$s initially a minimum, but as the rest length increases, the shape of the trajectory changes. Between $u = 0.92$m and $u = 0.98$m the extreme value at $t = 0$s changes from a minimum to a maximum. Finally, as the rest length u increases further, the trajectory again becomes a cosine alike function as it was for lower values of the rest length. The difference is apparent in a (y, z)-plot for $u = 0.7$m and $u = 1.2$m (figure 5.9). The parameters are stated in table B.2.

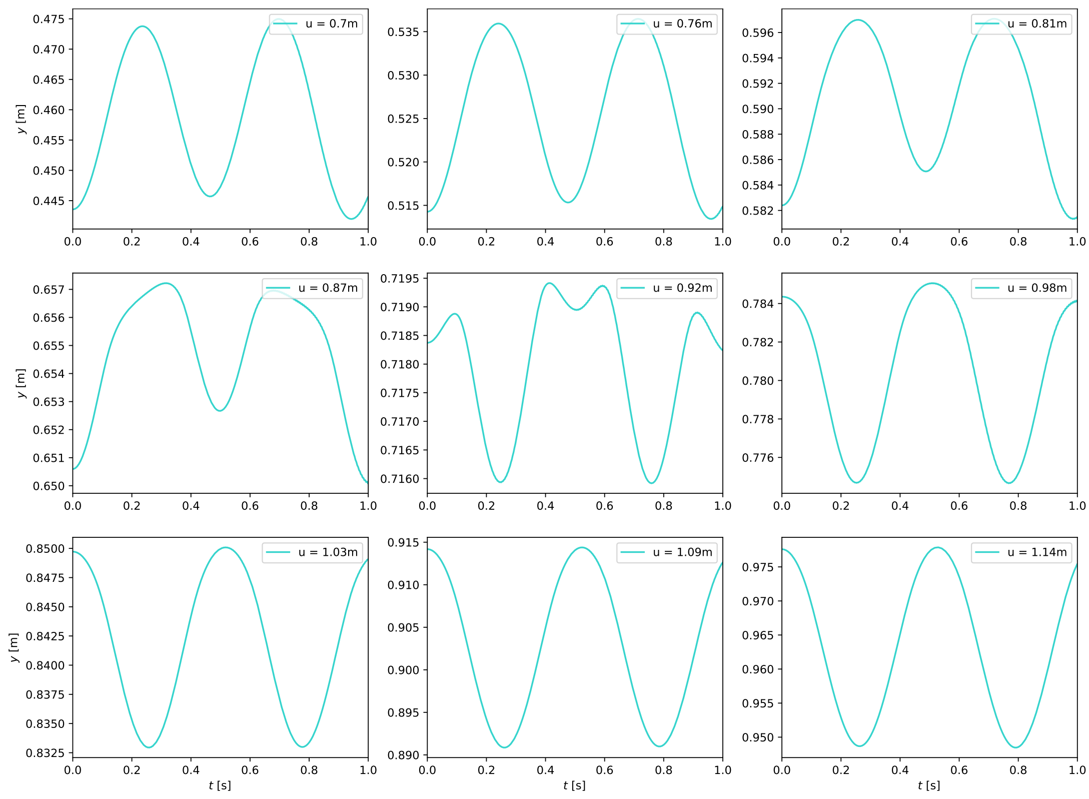

Figure 5.8: The optimal y-trajectories for certain values (0.7m to 1.14m) of the rest length u as a function of time t. The graph transitions from an initial minimum to an initial maximum. The spring constant k equals 4kNm^{-1}.

There are two distinct walking trajectories according to the model. These distinct trajectories are actually walking and not running. There is a normal force exerted on the ground at all times. This is shown in figure 5.10. The figure also shows that a longer rest length results in a more consistent force by the springs. A person walking is capable of performing these different trajectories. The negative trajectory ($u = 1.20$m) is the normal method of walking. A constant load on the spring could imply a higher efficiency. The positive trajectory ($u = 0.7$m) is a more exited method of walking. This method of walking can be achieved by applying a higher force to the ground.

Notice that the height of positive trajectory in figure 5.9 is much lower than the height of the negative trajectory. So much so that the positive trajectory, given the mass $M = 74$kg, does not model a real person anymore. The length of such a person is approximately 0.80m, corresponding to a bmi of 116. Modifying the parameters can change this height to be more realistic. Solely changing the rest length is insufficient because then the y-trajectory transitions into the negative trajectory. The parameters $k = 7\text{kNm}^{-1}$ and $u = 1.1$m have been used in figure 5.11. It shows a positive trajectory that is at a reasonable height. The other parameters are in table B.4.

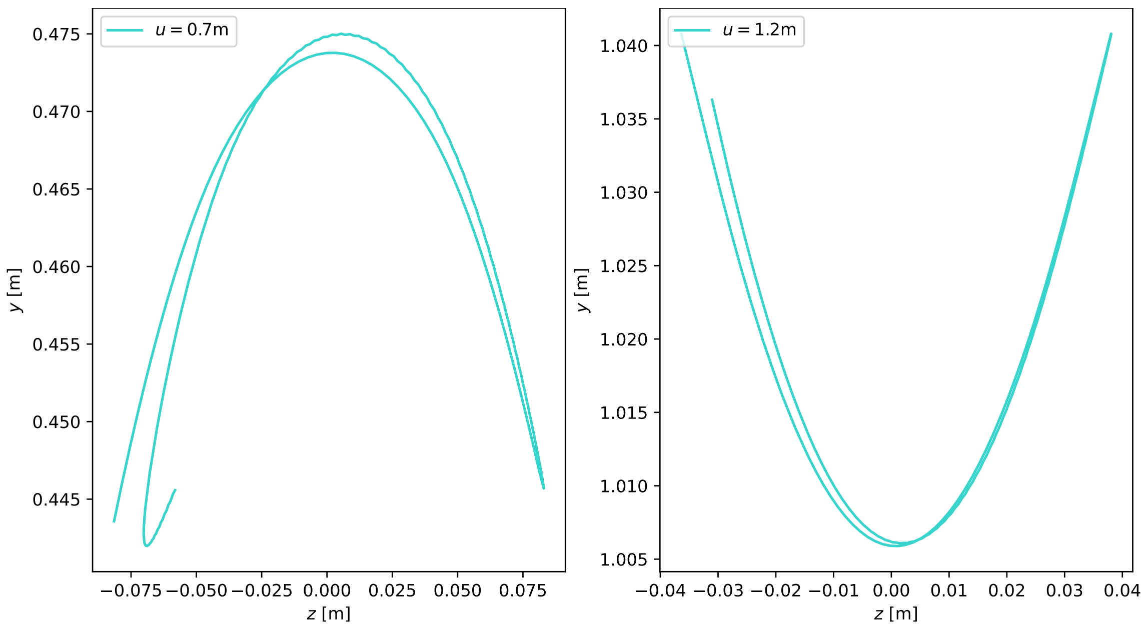

Figure 5.9: The two distinct trajectories as a result of a different rest length. The trajectories are plotted until $t = 1$s. The spring constant k equals 4kNm^{-1}. The height y is plotted on the vertical axis and the lateral z-position on the horizontal axis.

The model also shows that there is only one feasible way of running in the marathon case (figure 5.12). The exact parameters used are in table B.3. There is, however, still a downwards linear trend. Hence for very large values of u there is a second running possibility. Such a high rest length is not realistic. Moreover, for very large values of u the model would not be running but walking instead.

There is a reason why there are two ways to walk but only one exists for running. When running, there is a period where there is no spring force acting on the centre of mass. This period includes the part of the trajectory around $z = 0$m. Gravity, however, does still act upon the centre of mass. If there would be a minimum at $z = 0$m, then the trajectory would have to go down even further. But $z = 0$m was a minimum. Therefore $z = 0$m can not be a minimum. Hence the negative trajectory can not exists when running.

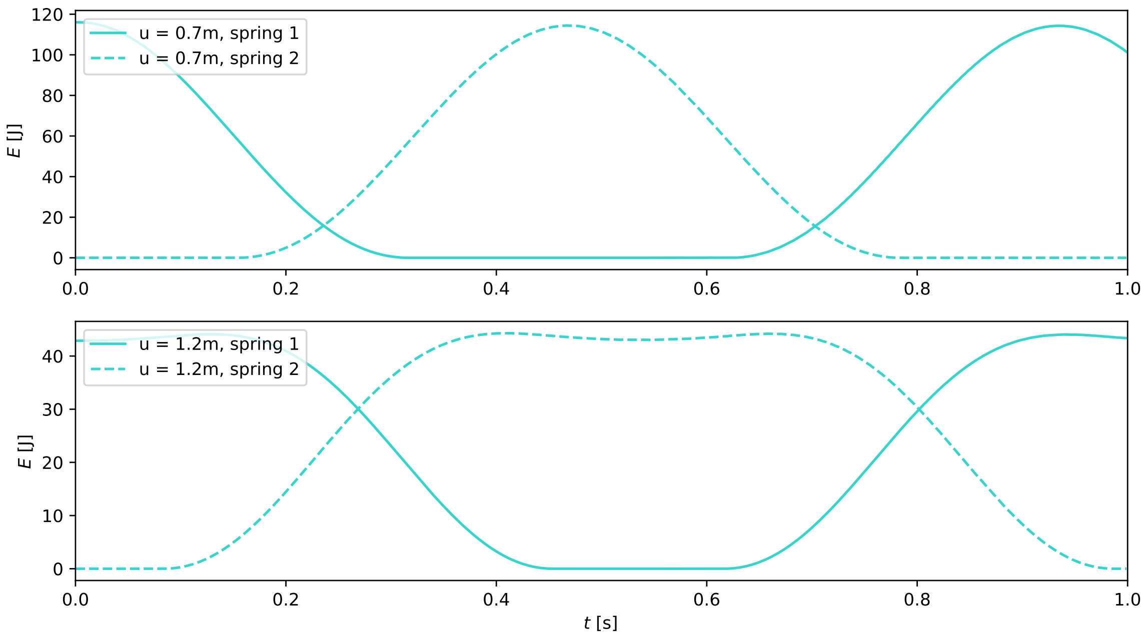

Figure 5.10: The spring energies of two distinct y-trajectories with different rest lengths. the vertical axis is the energy E and the horizontal axis is time t. The total spring energy is always larger than zero.

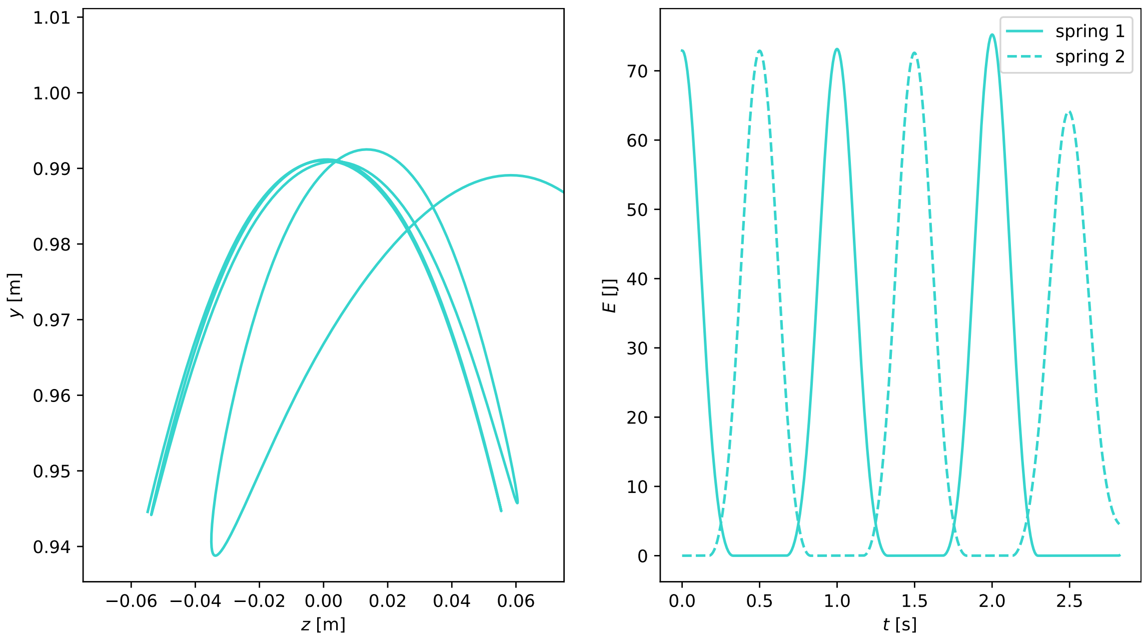

Figure 5.11: Left: a (y, z) projection of a positive trajectory of a person walking. The right plot shows the spring energies corresponding to this trajectory. The energy E is plotted against the time t axis. The total spring energy is always larger than zero. The unstable, aperiodic part of this trajectory continues until $z = 0.2$m.

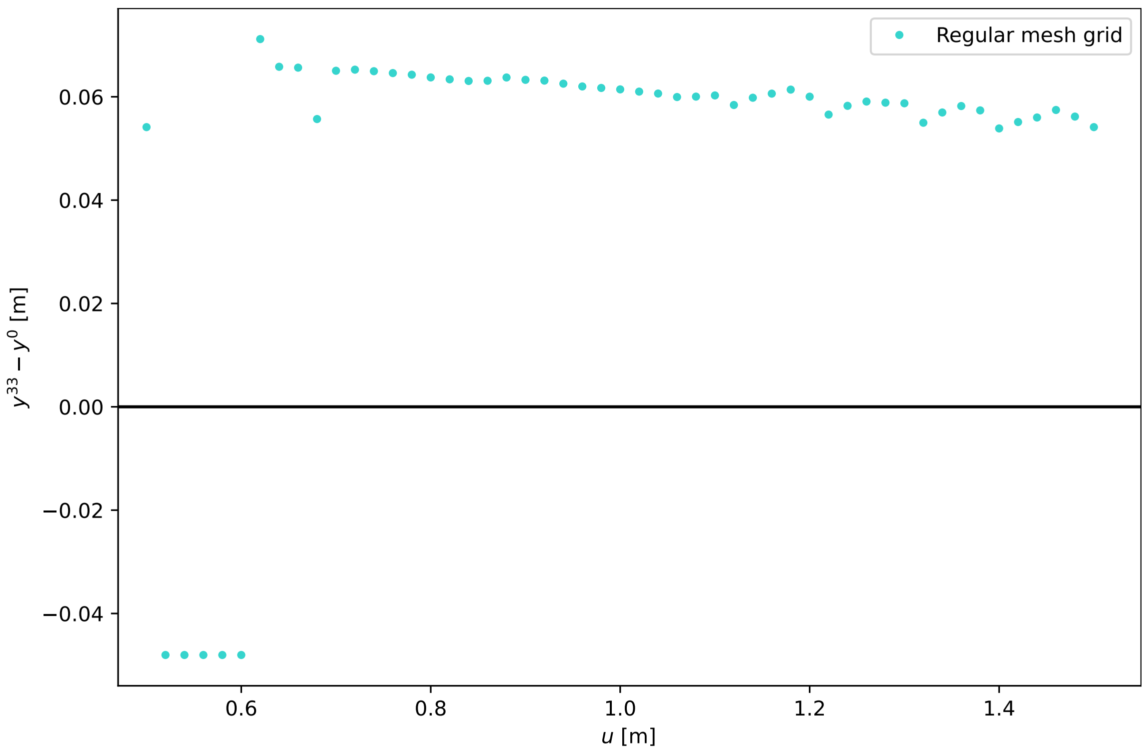

Figure 5.12: The difference between the y-values at $t = 0.1$s and $t = 0$s of the optimal trajectory of the marathon runner case. The vertical axis is the difference $y^{33} - y^0$ and the horizontal axis is the rest length u. The black line is the horizontal axis. The model is unable to fit the values of u lower than 0.6m.

5.2 Fitting the Model to the Data

This section consists of two parts. The first part will fit the model to generated data. The second part will fit the model to experimental data. For the first part, synthetic data has to be generated. The walking case will be used for the generation with $k = 4\text{kNm}^{-1}$ and $u = 1.1\text{m}$. The time step is set to $\Delta t = 0.01\text{s}$ to match the frequency of the experimental data. Then the integration method is Runge-Kutta integration and not central differences to keep the fit from oscillating. Table B.5 provides all other parameters for the data generation.

The DFA will fit the model to the data set using a time step of $\Delta t = 0.001\text{s}$ and central differences. The step size s_s is obtained from the periodicity of the data. As the step width is unknown, several values must be attempted to monitor the effect on the fitting capabilities. Table B.6 states the other parameters. The results are summarised in table 5.2. Several observations can be made from the table. The step width is very relevant for obtaining a proper fit, especially regarding the error. A deviation of 5cm already results in an difference of 13 per cent with the true spring constant. The rest length changes only 2 per cent when the step lengths changes 5cm. Note that the best found fit is not with the true step width. If $s_w = 0.19\text{m}$, then the error is 3 times lower than when $s_w = 0.20\text{m}$. $s_w = 0.19\text{m}$ has a larger deviation in the parameters with respect to the true parameters. This phenomenon is due to the different time steps and numerical methods, resulting in a different optimal step width.

Table 5.2: Fitting capabilities of the DFA to synthetic data for several step widths s_w. All values have been rounded to four significant digits. The actual values are $k = 4\text{kNm}^{-1}$ and $u = 1.100\text{m}$ with a step width of 0.2000m.

s_w [m]	k [Nm^{-1}]	u [m]	10^{-2} e [-]
0.1000	4808	1.066	42.93
0.1500	4473	1.079	7.200
0.1900	4071	1.096	0.4113
0.2000	3960	1.101	1.165
0.2500	3418	1.130	15.73
0.4000	2464	1.229	111.8

The y- and z-trajectories of fits with several step widths are shown in figures 5.13 and 5.14. A change in the step with has a greater effect on the z-trajectory than on the y-trajectory. The reason is that a change in step width changes the angle between the trajectory and the connection points of the feet. This results in a rotation of the yz-projection of the spring force.

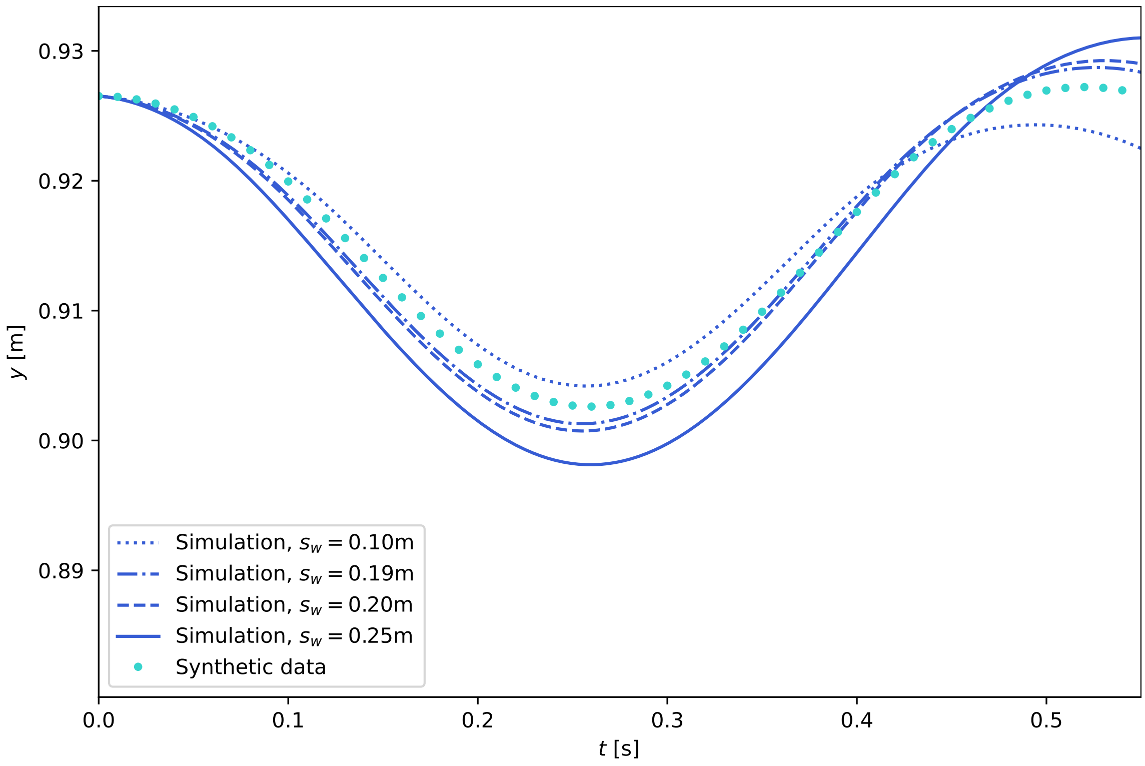

Figure 5.13: Various step widths s_w fitted on a synthetic data set ($s_w = 0.20$m). The vertical axis plots the height y of the centre of mass and the horizontal axis the time t.

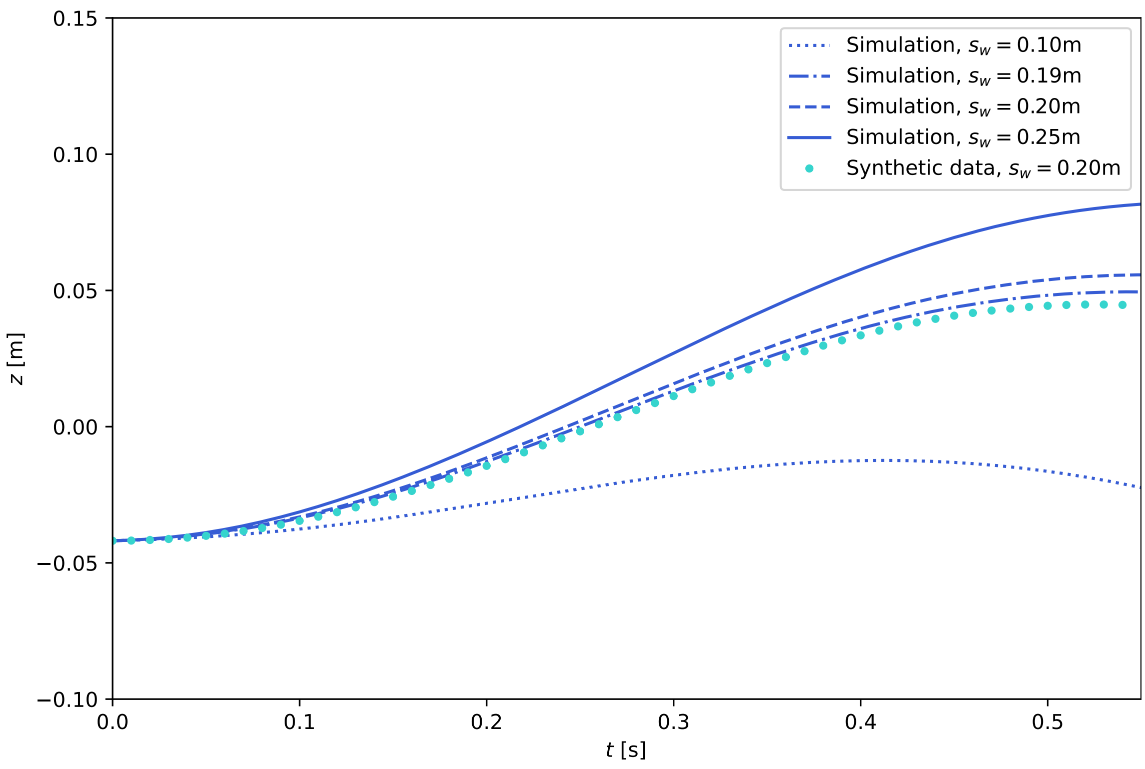

Figure 5.14: Various step widths s_w fitted on a synthetic data set ($s_w = 0.20$m). The vertical axis plots the lateral position z of the centre of mass and the horizontal axis the time t.

The next step is to implement the model to the data provided in figures 4.2, 4.3, and 4.4. The step size is determined with the periodicity of the data as 0.66m. The other parameters can be found in table B.7. As the step width is unknown, it has been varied from 0m to 0.5m. Figure 5.15 shows the parameters for the range of step widths. The low values for the step length show a deviation in the pattern. This occurs because the data has a larger z-component than the maximal values that the z-trajectory of the simulation can attain. That makes a fit with a reasonable error impossible for values for s_w.

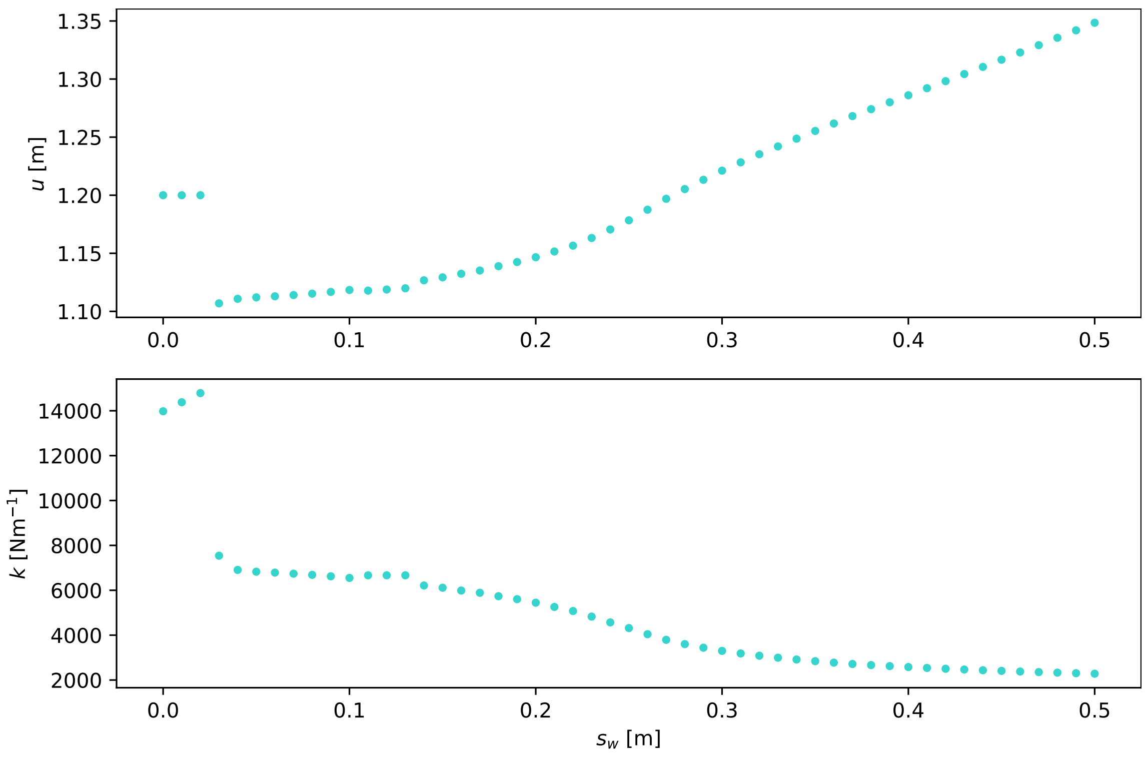

Figure 5.15: The spring constant k and rest length u plotted as a function of the step width s_w. The spring constant and the rest length are the fitted parameters for the DGA. When the step width is lower than the lateral position of the data no proper fit can exist.

The error and termination time for different step widths are shown in figure 5.16. There is a clear minimum for the error located at $s_w = 0.13$m. Note that in this case the termination time does not correspond with the error. A low error in the DGA algorithm does not correlate to the periodicity of the trajectory. The error is big for large values ($s_w > 0.18$m) of the step width. Therefore no relation between the step width and the fitting parameters k and u can be stated.

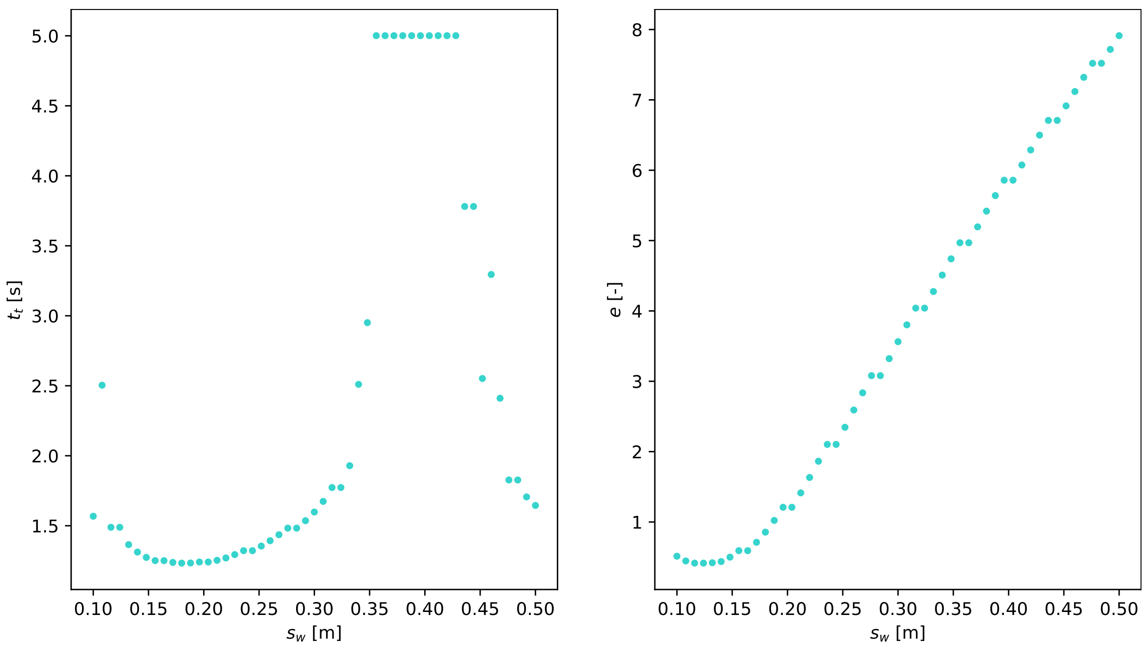

Figure 5.16: The termination time t_t and error e as a function of the step width s_w. There is no correlation between termination time and error. The error is minimal at $s_w = 0.13$m.

There are several methods that will be used to fit the model to the data. First there will be a fit where a range of step widths are considered (table B.8). A logical extension is to translate the initial conditions relative to the feet. This will be implemented by adding a vector $(\Delta s, 0, 0)^\top$ to the initial coordinates of the feet ($\mathbf{r}_1$ and $\mathbf{r}_2$). This is a verification for the assumption that the extreme values of the y- and z-position coincide with the coordinates of the feet. The parameter can be found in table B.9. A third option is to forgo the assumption of equal legs. This can address the non-periodicity of the data (table B.10).

The DFA does not force the fit to be periodic. This could be a problem as the data is not periodic. Then this can result in a non-periodic, unstable fit. Using the PPFA in combination with DFA will force the solution to be periodic (table B.11). By disregarding the y_0- and z_0- coordinates a periodic solution can be found for (reasonable) combinations of k and u. This has the highest error of all methods ($e_{\text{DFA}} = 0.4884$ and $e_{\text{PPFA}} = 2.482{\cdot}10^{-5}$). Only accounting for the step width results in an error of 0.4209. The error is a bit lower when the feet are shifted relative to the initial conditions ($e = 0.4023$). The shift is -0.02m, showing that the assumption of alignment of extreme values and the feet is reasonable. The lowest error occurs when both legs are considered distinct ($e = 0.3535$). The found parameters are stated in table 5.3.

The fits match the x-trajectory excellent as expected (figure A.5). On the contrary, the x-momentum deviates the most from the data (figure 5.17), but the trend of maxima and minima aligns. The y-position (figure 5.18) is a decent fit, as is the corresponding momentum. Regarding the y-momentum (figure 5.19), initially there is a delay and the intricacies at later times are not well modelled. The z-position (figure 5.20) and momentum (figure A.6) align with the trajectory, showing only minor deviations.

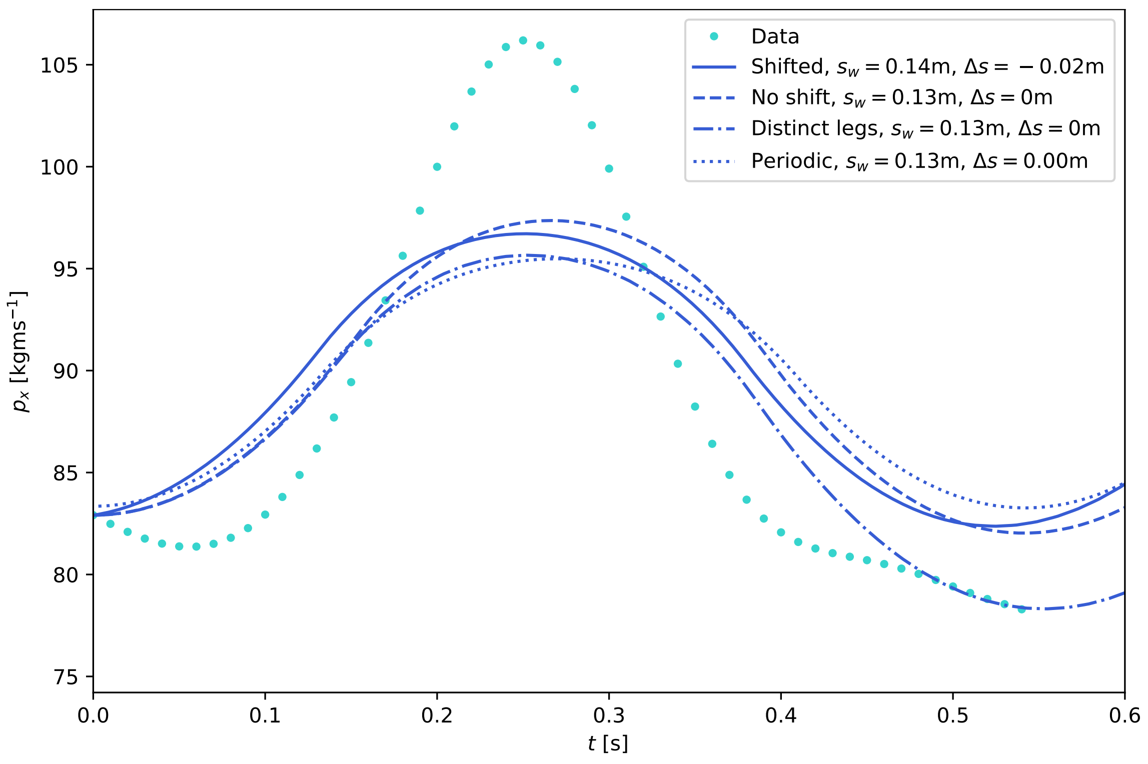

Figure 5.17: Four different methods fitted to the data. The vertical axis is the x-momentum p_x and the horizontal axis is time t.

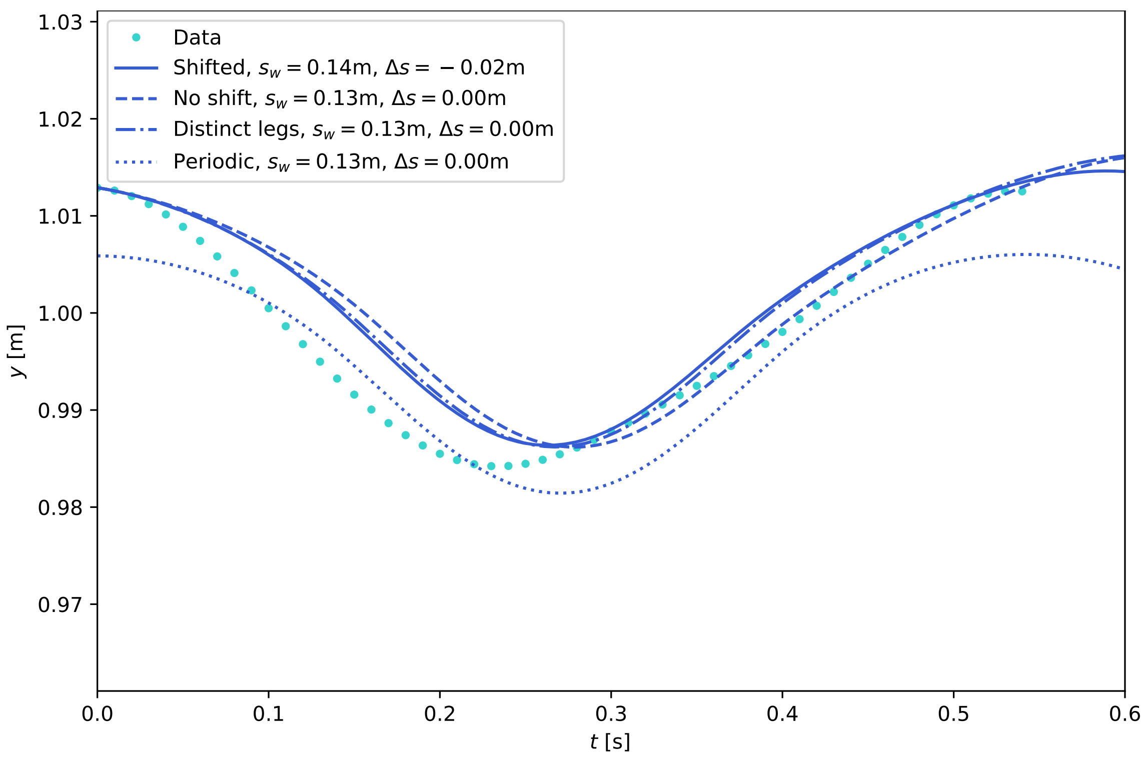

Figure 5.18: Four different methods fitted to the data. The vertical axis is the y-position and the horizontal axis is time t.

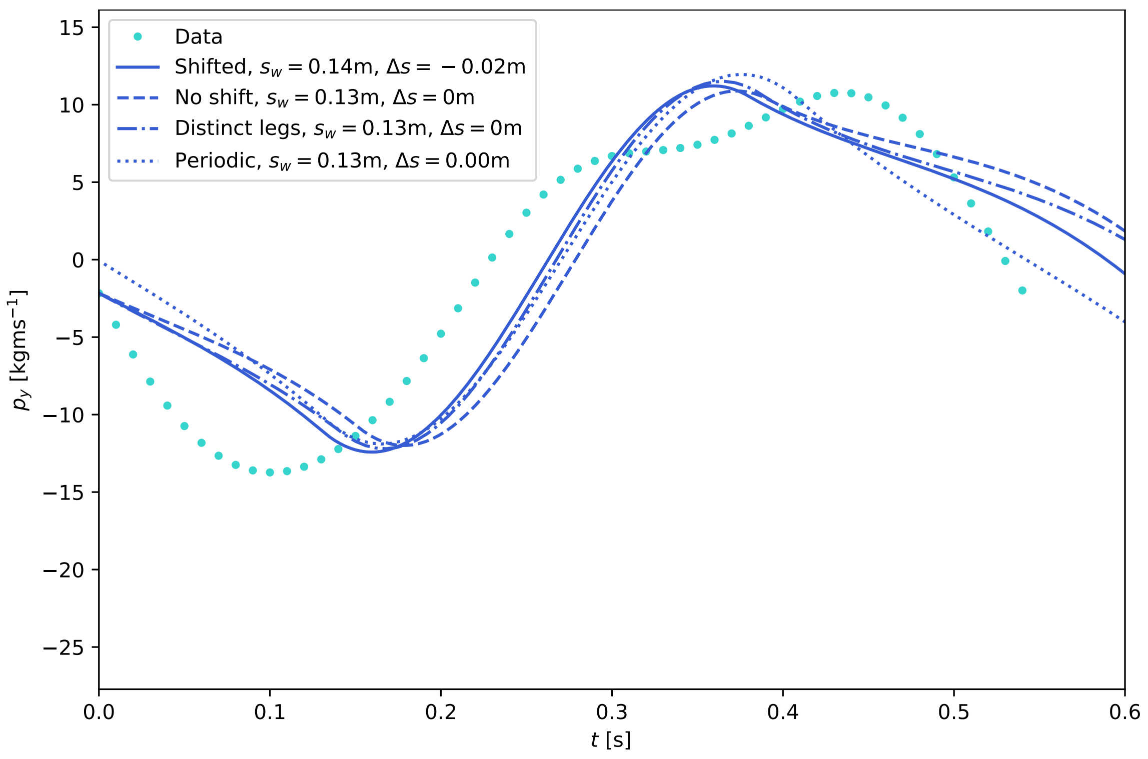

Figure 5.19: Four different methods fitted to the data. The vertical axis is the y-momentum p_y and the horizontal axis is time t.

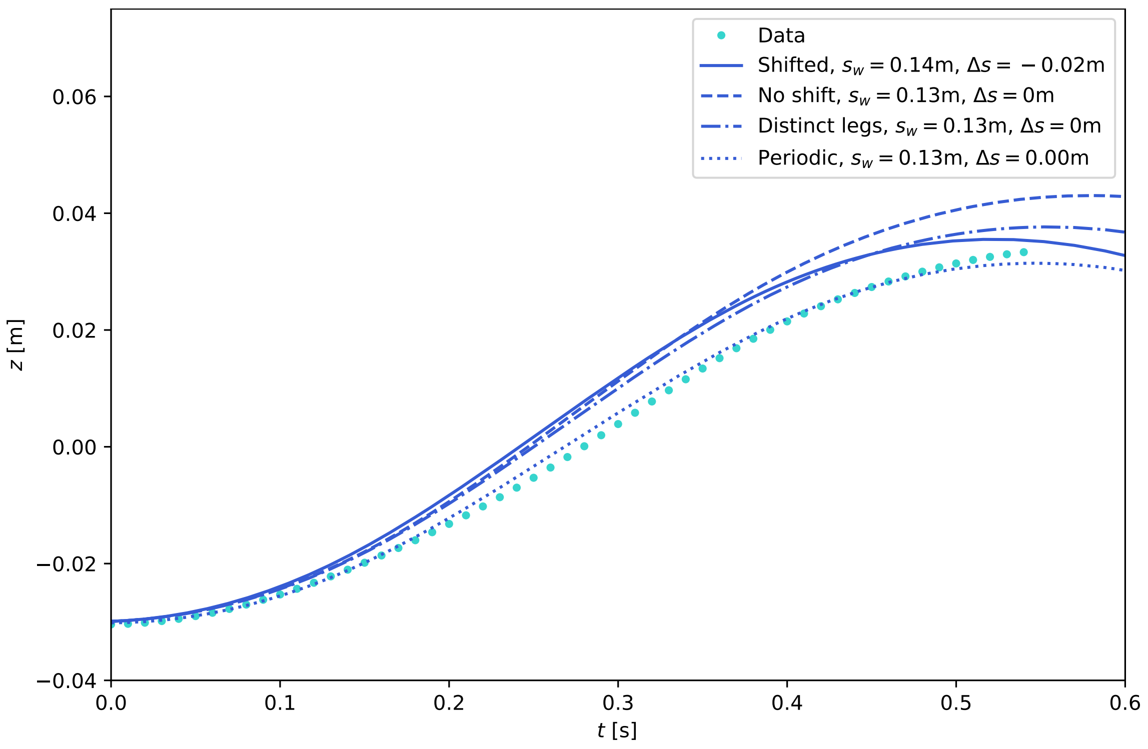

Figure 5.20: Four different methods fitted to the data. The vertical axis is the z-position and the horizontal axis is time t.

Table 5.3: The fitted parameters for several methods. Each situation is fitted to the data set of a walking person. The distinct legs method and the periodic method have $s_w = 0.13$m and $\Delta s = 0$m.

Type of fit	Parameters	Value	Unit
No shift	k	6667	[Nm^{-1}]
	u	1.119	[m]
	s_w	0.13	[m]
Shifted	k	6375	[Nm^{-1}]
	u	1.125	[m]
	s_w	0.14	[m]
	Δs	-0.02	[m]
Distinct legs	k_1	6802	[Nm^{-1}]
	k_2	6421	[Nm^{-1}]
	u_1	1.116	[m]
	u_2	1.126	[m]
Periodic	k	5670	[Nm^{-1}]
	u	1.127	[m]
	y_0	1.006	[m]
	z_0	-0.03015	[m]

All in all can the fit of the data be considered adequate. The trend of the fit aligns with the data but some details are lost or less present. There are not a lot of differences between the methods and the difference in error is small. This means that the assumption of equal legs is also reasonable, even though the participant has a longer left leg, with a difference of 0.007m.

The periodic fit is the most relevant fit for the prosthetic leg. It ensure periodic continued locomotion while also following the natural movement as much as possible. The deviations from the data could be significant, but not necessarily so. The deviations in the x-momentum, for example, smoothen the trajectory. That could be experienced as pleasant by a person using a prosthetic leg with a spring constant designed. It is not trivial to design a prosthetic leg with a spring constant as prescribed by the model. Because the centre of mass is inside the body, the spring-like behaviour of a lower torso also has to be taken into account. Future research could create a simple prosthetic and fit it with the predicted spring by the model. Then a test person can judge whether or not the found parameters result in a pleasant experience. The trajectory can then be compared with that of an able-bodied person.

There is a distinction that should be made regarding the applicability of the model. Namely that of people missing one or two legs. A person missing one leg will require a prosthetic design that results in as much as possible a natural trajectory. For them the deviation in the trajectory is more likely to be unpleasant. The requirement of natural motion is less important for people missing both legs. For them a smoother trajectory and especially momentum could be desirable. Future research should investigate these claims.

The main limitation of the model is the processing requirement of the data. Because the data has to be somewhat periodic, person 4 was excluded. This limits the general applicability of the model. Moreover, the steps taken by the model have constant step width and step size. This is not the case with the data, even though it was averaged and smoothed. As shown by the fit that varies the step width, small changes in the step width greatly affects the fit. The solution of only using half a gait cycle does not do justice to the details of bipedal locomotion. The next suggestions for expansions of the model will address these two issues, or improve the model in another way.

There are several ways this model can be expanded. The first decouples the extreme values of the y- and z-trajectories in the PPFA. There is a small shift between these extreme values visible in the data (figures 4.3 and 4.4). The DFA already accounts for such a shift, but the PPFA does not. Hence the quality of the periodic fit can be increased by adding another parameter accounting for this difference.

A second option no longer considers the springs as linear. Then the effect of different springs can be studied. It could be that a higher power relation between the extension and the force provides a better fit. The downside of this method is that it is no longer applicable to simple prosthetic leg design. Non-linear springs are much more costly, defeating the purpose of a simple prosthetic.

Another option is to increase the complexity of the model. This can be done in several ways. The first makes the feet more realistic by increasing the number of contact points with the ground. If the ground connections are connected to the centre of mass by springs with different properties, then the model would become too computationally intensive. However, even when all the springs are equivalent this could be a useful addition. Then the effect of the shape of a foot can be studied. A second extension no longer considers the legs merely as springs. Instead the knees are included too. For example, the knees could be modelled as an angle dependent spring.

A fourth option is to extend the model by adding more forces to the system. Certain friction forces can be included. Then the feet could slide if the sliding friction force is too low. Air resistance can be added too, but for a runner this only accounts for 2 to 5 per cent of the total forces present [22]. The internal forces in the muscles that damp the motion are by far the most present. In order to counteract these forces a driving force has to be added. This driving force, which should be a periodic function, can be defined in different ways to find the optimal fit. Attempting different functions could result in even more methods of walking or running.

Finally the method of taking steps could be changed, making the fitting of unsmoothed data possible. Currently the steps are not modelled. Instead of instantaneous steps, the feet could move like a simple pendulum. Then the next ground contact point can be determined by the place where the foot strikes. This results in a variable step length and step width. This will not improve the fit to the smoothed data by much, but it could result in a stable trajectory. The feet could also have mass, allowing for proper dynamics.

CHAPTER

SIX

Conclusion

The purpose of this report was twofold. The first was to implement a three dimensional spring-mass model to model the gait cycle of humans. The second was to find the optimal parameters for a spring in a simple leg prosthetic. To do so, two algorithms were designed. The Periodic Path Finding Algorithm (PPFA) finds the optimal initial height and lateral displacement such that the trajectory is periodic. The Data Fitting Algorithm (DFA) finds the optimal spring constant and rest length to best fit a data set.

The PPFA showed that the initial z-trajectory is independent of the spring constant if the rest length is set such that the centre of mass is at 56% of the height of a person. The algorithm also showed that there are two distinct trajectories for walking but there is only type of trajectory for running. The additional trajectory present for walking has a more consistent spring force profile.

The synthetic data fit of DFA showed that the step width is important for finding a proper fit. The real data set has been fitted using several methods. The first method has a variable step width and another method also has a variable initial x-coordinate. These methods yielded very similar results ($e = 0.4209$ and $e = 0.4023$). The displacement in the x-direction is only $\Delta s = 0.02$m. This shows that the assumption of aligning the extreme value of the z-trajectories with the position of the feet is valid. Considering the legs as distinct gives a small improvement ($e = 0.3535$). The last fit to the real data combined PPFA and DFA. This forces the fit to be periodic by relaxing the initial position coordinates y_0 and z_0. Although this fit has the highest error ($e = 0.4884$), it is the most applicable to the prosthetic leg design. It shows that periodic motion is possible with a single spring that can mimic the general characteristics of the trajectory of a person walking. It is unknown how the deviations will impact the experience of a person missing a leg.

Summarizing, the three dimensional spring-mass model describes the general traits of the centre of mass well, but the details are missing. There are two distinct ways to walk, but only one to run. The extreme y- and z-values align with the placement of feet. Further research should investigate how deviations with the data are experienced by people missing a leg.

Bibliography

[1] Robert McNeill Alexander. “Bipedal animals, and their differences from humans”. In: *Journal of Anatomy* 204.5 (May 2004), pp. 321–330. DOI: 10.1111/j.0021-8782.2004.00289.x.

[2] Sophie Penney. “Kipchoge wants athletics to embrace tech”. In: *The West Australian* (Aug. 2021). URL: https://thewest.com.au/sport/athletics/kipchoge-wants-athletics-to-embrace-tech-c-3783537.

[3] Freepik. *Front view of a walking wild duck.* URL: https://www.freepik.com/free-photo/dancing-duck_10759315.htm.

[4] Bence Mate. *Double-crested basilisk (Basiliscus plumifrons) running across water surface.* Nov. 2008. URL: https://powerspeedendurance.com/n1-dont-run-scissors-go-run-lizards/.

[5] Karen E. Adolph et al. “How do you learn to walk? Thousands of steps and dozens of falls per day”. In: *Psychological Science* 23.11 (Oct. 2012), pp. 1387–1394. DOI: 10.1177/0956797612446346.

[6] Michael D. Sockol, David A. Raichlen, and Herman Pontzer. “Chimpanzee locomotor energetics and the origin of human bipedalism”. In: *Proceedings of the National Academy of Sciences* 104.30 (July 2007), pp. 12265–12269. DOI: 10.1073/pnas.0703267104.

[7] Louis Buckley. “This chimp is made for walking”. In: *Nature* (July 2007). DOI: 10.1038/news070716-2.

[8] Daniel E Lieberman and Dennis M Bramble. “The Evolution of Marathon Running, capabilities in Humans”. In: *Sports Medicine* 4–5 (Apr. 2007), pp. 288–290. DOI: 10.2165/00007256-200737040-00004.

[9] Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. “Compliant leg behaviour explains basic dynamics of walking and running”. In: *Proceedings of the Royal Society B: Biological Sciences* 273.1603 (Aug. 2006), pp. 2861–2867. DOI: 10.1098/rspb.2006.3637.

[10] Céline Schreiber and Florent Moissenet. “A multimodal dataset of human gait at different walking speeds established on injury-free adult participants”. In: *Scientific Data* 6.1 (July 2019). DOI: 10.1038/s41597-019-0124-4.

[11] Huiqi Liang et al. "A three-dimensional mass-spring walking model could describe the ground reaction forces". In: *Mathematical Problems in Engineering* 2021 (July 2021), pp. 1–20. DOI: 10.1155/2021/6651715.

[12] Enamul Hoque, Shifat Al Riham, and Abdul Shuvo. "A cost-effective prosthetic leg: Design and development". In: *Hybrid Advances* 2 (Apr. 2023). DOI: 10.1016/j.hybadv.2022.100017.

[13] Richard Wolfson. *Essential University Physics*. 3rd ed. Vol. 1. Pearson Education Limited, 2015.

[14] Editors of Encyclopaedia Brittanica. *Hooke's law*. Accessed on 05 04 2023. Dec. 2022. URL: https://www.britannica.com/science/Hookes-law.

[15] John Robert Taylor. *Classical Mechanics*. 1st ed. University Science Books, 2005.

[16] William David McComb. *Dynamics and Relativity*. 1st ed. Oxford University Press, 1999.

[17] C. Vuik et al. *Numerical methods for Ordinary Differential Equations*. 2nd ed. Delft Academic Press, 2018.

[18] Bo Einarsson and Yurij Shokin. *Fortran 90 for the Fortran 77 programmer*. Siberian Division of the Russian Academy of Science, 1995.

[19] Benjamin Michaud and Mickaël Begon. "Ezc3d: An easy C3D file I/o cross-platform solution for C++, python and MATLAB". In: *Journal of Open Source Software* 6.58 (Feb. 2021), p. 2911. DOI: 10.21105/joss.02911.

[20] Stephen Butterworth. "On the theory of filter amplifiers". In: *Experimental Wireless and the Wireless Engineer* 7 (Oct. 1930), pp. 536–541.

[21] Paul Davidovits. *Physics in Biology and Medicine*. 5th ed. Academic Press, an imprint of Elsevier, 2019.

[22] A.V. Hill. "The air-resistance to a runner". In: *Proceedings of the Royal Society of London. Series B, Containing Papers of a Biological Character* 102.718 (Dec. 1927), pp. 380–385. DOI: 10.1098/rspb.1928.0012.

Appendix

A Additional Figures

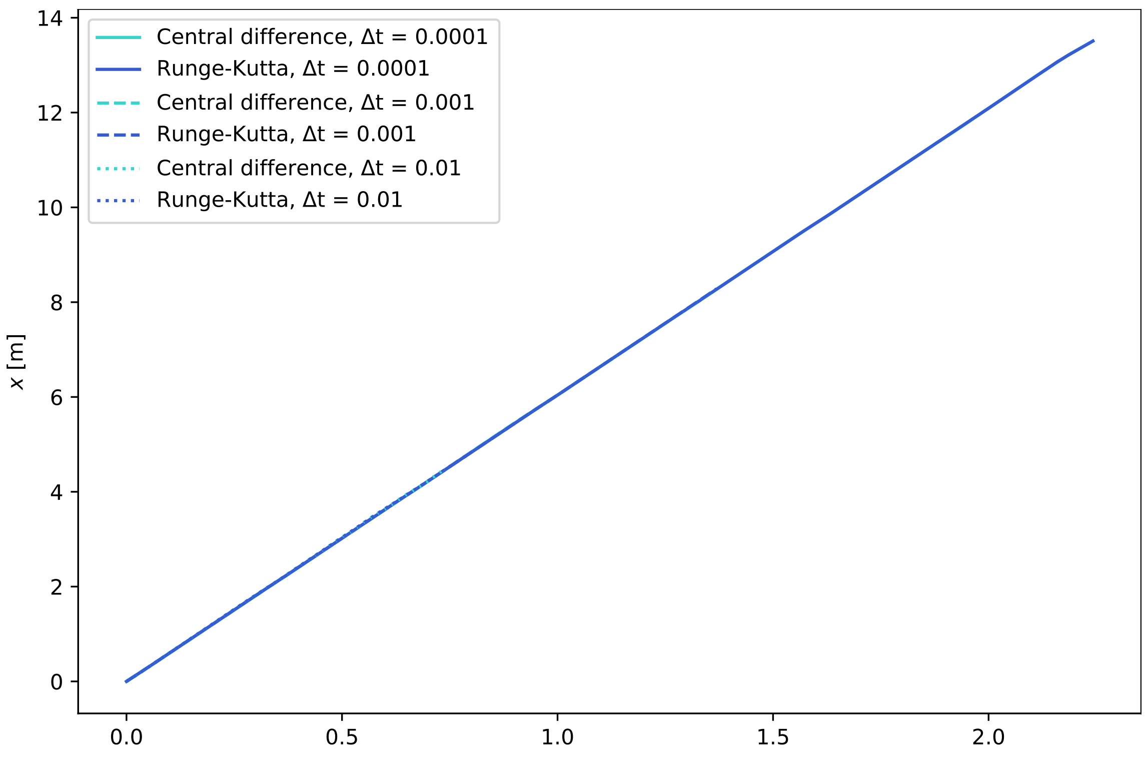

Figure A.1: The x-values of the optimal trajectories for the marathon running case plotted against the x-axis and the time axis t. The trajectories have been plotted using both central differences as well as Runge-Kutta integration for different time steps.

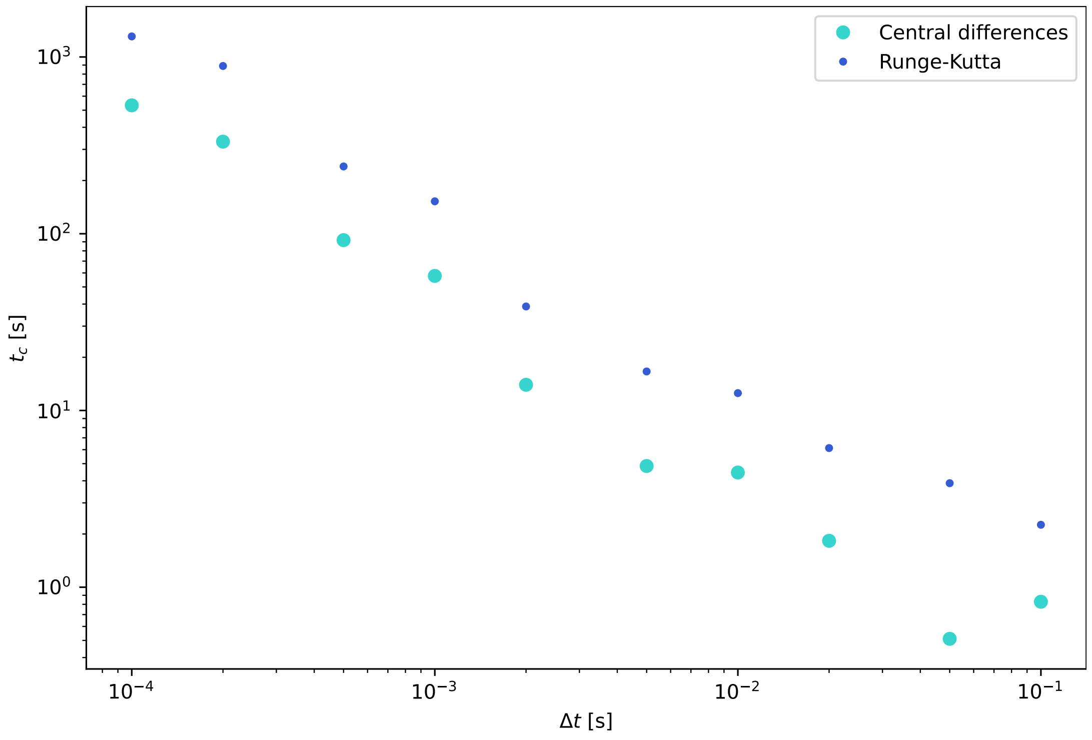

Figure A.2: Computation time of the simulation on a i7-9750H. The vertical axis displays the computation time t_c and the horizontal axis displays the time step Δt. Both scales are logarithmic.

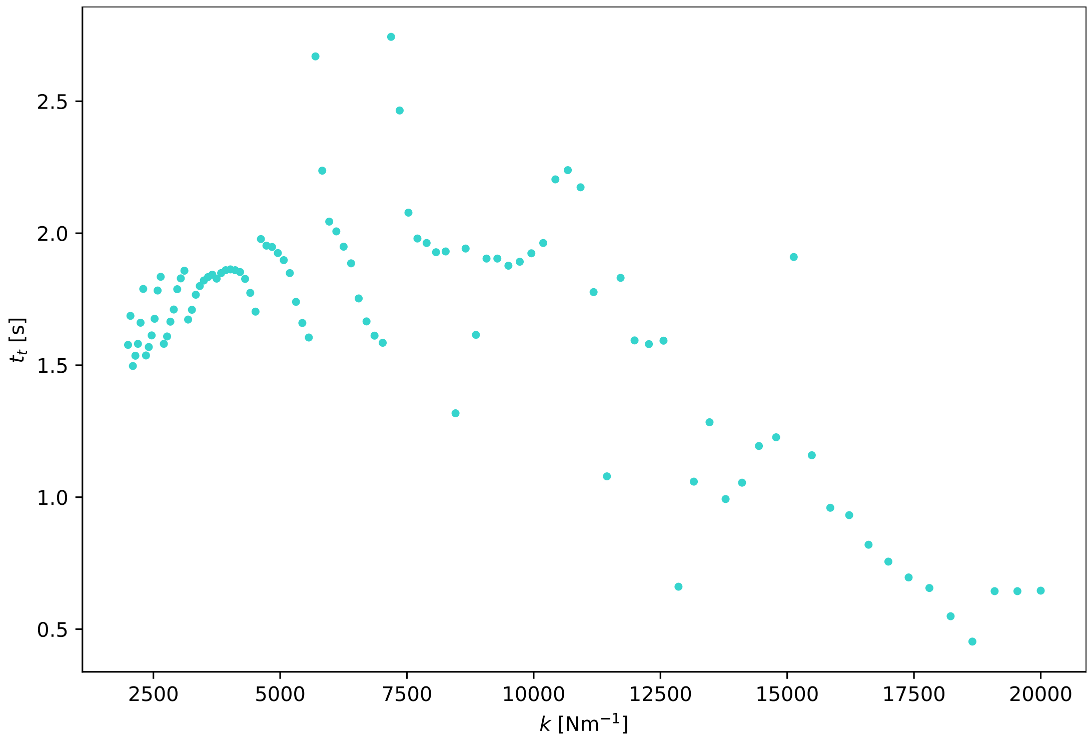

Figure A.3: The termination time of the marathon simulation t_t plotted against the spring constant k. At higher values of k the model can not find a proper fit.

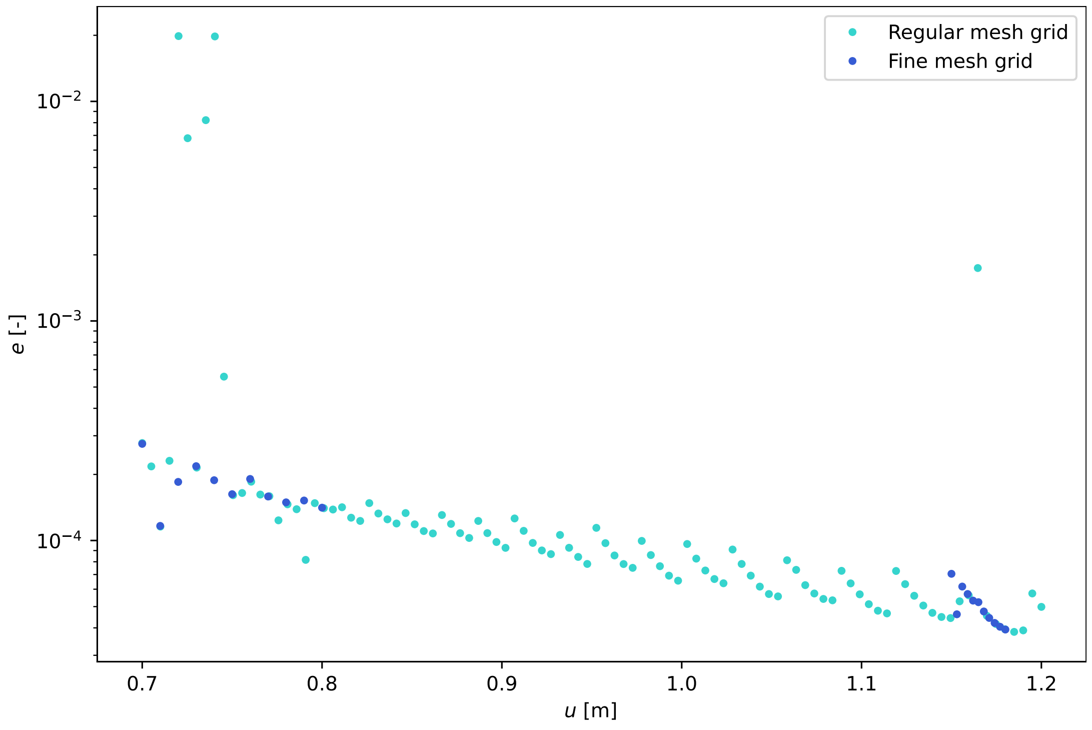

Figure A.4: The error e of the walking person simulation for various values of u. The fine mesh grid has double the mesh discretisation (22) and convergence rate (4) than the regular mesh grid $(11$ and $2)$.

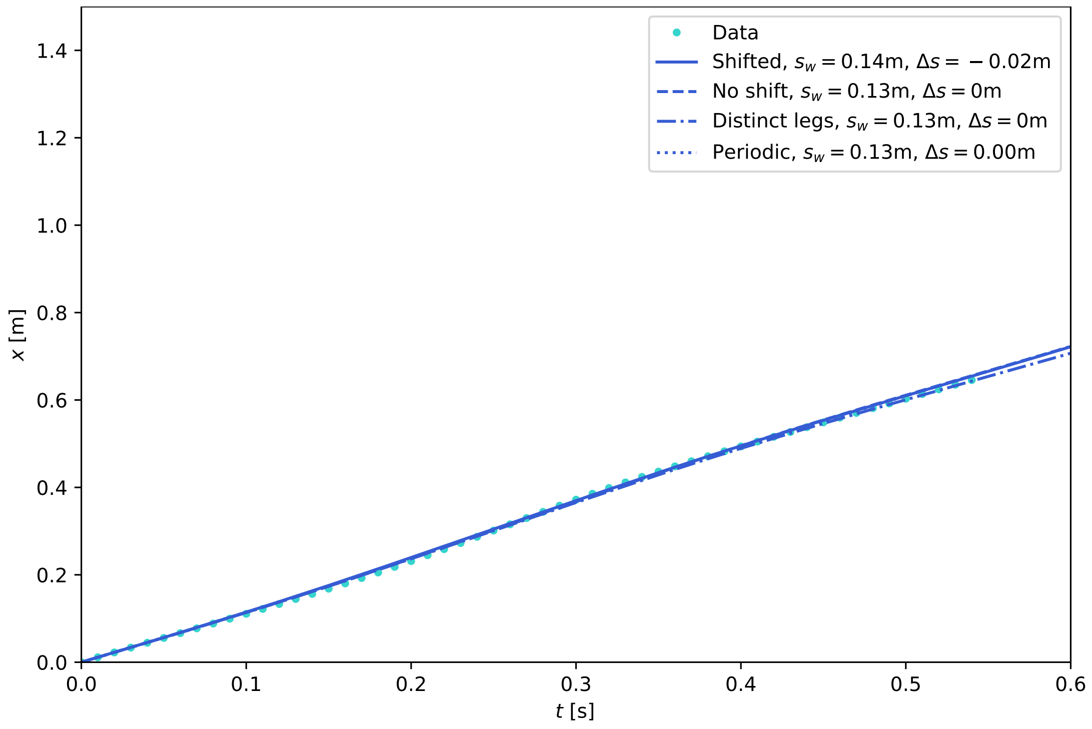

Figure A.5: Four different methods fitted to the data. The vertical axis is the x-position and the horizontal axis is time t.

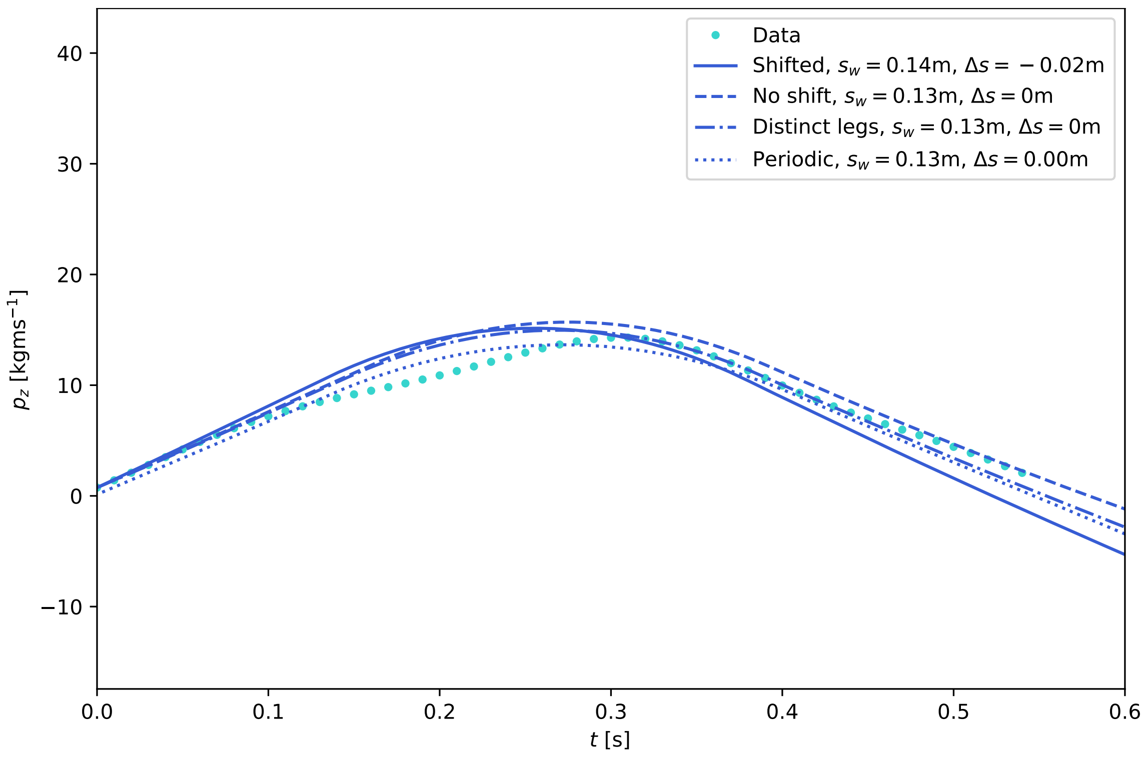

Figure A.6: Four different methods fitted to the data. The vertical axis is the z-momentum p_z and the horizontal axis is time t.

B Parameter Tables

The gravitational constant g is set to 9.81ms^{-2} in all simulations.

Table B.1: The parameters of the PPFA for the marathon runner case with varying spring constant and corresponding rest length.

	Parameter	Value	Unit
Springs	k	$[2000, 20000]$	[Nm^{-1}]
	u	$0.56l + \frac{Mg}{2k}$	[m]
System	l	1.67	[m]
	M	52	[kg]
	s_s	1.91	[m]
	s_w	0.1	[m]
	Δs	0	[m]
	$\mathbf{p}_0$	$M(5.83, 0, 0)^\top$	[kgms^{-1}]
Numerical	Δt	0.001	[s]
	N	$5(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	y_0	$[0.6, 1.0]$	[m]
	z_0	$[-s_w, 0]$	[m]

Table B.2: The parameters of the PPFA for the walking person for varying rest length u and fixed spring constant k.

	Parameter	Value or range	Unit
Springs	k	4000	[Nm^{-1}]
	u	[0.7, 1.2]	[m]
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	0.2	[m]
	Δs	0	[m]
	$\mathbf{p}_0$	$M(1.2, 0, 0)^\top$	[kgms^{-1}]
Numerical	Δt	0.003	[s]
	N	$5(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	y_0	$[0.4, 1.2]$	[m]
	z_0	$[-s_w, 0]$	[m]

Table B.3: The parameters of the PPFA for the marathon runner for varying rest length u and fixed spring constant k.

	Parameter	Value or range	Unit
Springs	k	10000	[Nm^{-1}]
	u	$[0.5, 1.5]$	[m]
System	M	52	[kg]
	s_s	1.91	[m]
	s_w	0.1	[m]
	$\mathbf{p}_0$	$M(5.83, 0, 0)^\top$	[kgms^{-1}]
Numerical	Δt	0.003	[s]
	N	$5(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	y_0	$[0.4, 1.2]$	[m]
	z_0	$[-s_w, 0]$	[m]

Table B.4: The parameters of the PPFA used to provide a positive trajectory on a reasonable height..

	Parameter	Value or range	Unit
Springs	k	7000	[Nm^{-1}]
	u	1.1	[m]
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	0.2	[m]
	$\mathbf{p}_0$	$M(1.2, 0, 0)^\top$	[kgms^{-1}]
Numerical	Δt	0.001	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	Runge-Kutta integration	[-]
	minimum improvement	0.001	[-]
Search range	y_0	$[0.4, 1.0]$	[m]
	z_0	$[-s_w, 0]$	[m]

Table B.5: The parameters of the PPFA for the person walking used for synthetic data generation.

	Parameter	Value or range	Unit
Springs	k	4000	[Nm^{-1}]
	u	1.1	[m]
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	0.2	[m]
	$\mathbf{p}_0$	$M(1.2, 0, 0)^\top$	[kgms^{-1}]
Numerical	Δt	0.01	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	Runge-Kutta integration	[-]
	minimum improvement	0.001	[-]
Search range	y_0	$[0.4, 1.0]$	[m]
	z_0	$[-s_w, 0]$	[m]

Table B.6: The parameters of the DFA for the person walking used for fitting synthetic data. The initial conditions are obtained from the data.

	Parameter	Value or range	Unit
System	M	74	[kg]
	s_s	0.657	[m]
	s_w	$[0.1, 0.4]$	[m]
Numerical	Δt	0.001	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	k	$[2000, 12000]$	[Nm^{-1}]
	u	$[0.5, 1.4]$	[m]

Table B.7: The parameters of the DFA for the person walking used with real processed data. The initial conditions are obtained from the data.

	Parameter	Value or range	Unit
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	$[0.1, 0.5]$	[m]
Numerical	Δt	0.001	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	k	$[2000, 12000]$	[Nm^{-1}]
	u	$[0.5, 1.5]$	[m]

Table B.8: The parameters of the DFA for the person walking used with real processed data. Both the step width as well as the coordinates of the feet are discretised by a shift in the x direction of value s.

	Parameter	Value or range	Unit
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	$[0.1, 0.3]$	[m]
	Δs	[-0.1, 0.1]	[m]
Numerical	Δt	0.003	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	k	$[2000, 12000]$	[Nm^{-1}]
	u	$[0.5, 1.5]$	[m]

Table B.9: The parameters of the DFA for the person walking used with real processed data. Both the step width as well as the coordinates of the feet are discretised by a shift in the x direction of value s.

	Parameter	Value or range	Unit
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	$[0.1, 0.3]$	[m]
	Δs	[-0.1, 0.1]	[m]
Numerical	Δt	0.003	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	k	$[2000, 12000]$	[Nm^{-1}]
	u	$[0.5, 1.5]$	[m]

Table B.10: The parameters of the DFA for the person walking used with real processed data. This simulation models the springs in the legs as distinct.

	Parameter	Value or range	Unit
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	$[0.1, 0.3]$	[m]
	Δs	0	[m]
Numerical	Δt	0.003	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	11	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	k_1	$[2000, 12000]$	[Nm^{-1}]
	k_2	$[2000, 12000]$	[Nm^{-1}]
	u_1	$[0.5, 1.5]$	[m]
	u_2	$[0.5, 1.5]$	[m]

Table B.11: The parameters of the DFA used on PPFA for the person walking used with real processed data. This simulation models the springs in the legs as distinct.

	Parameter	Value or range	Unit
System	M	74	[kg]
	s_s	0.66	[m]
	s_w	0.13	[m]
	Δs	0	[m]
Numerical	Δt	0.003	[s]
	N	5 $(\Delta t)^{-1}$	[-]
	mesh discretisation	7	[-]
	convergence rate	2	[-]
	integration method	central differences	[-]
	minimum improvement	0.001	[-]
Search range	k	$[4000, 8000]$	[Nm^{-1}]
	u	$[0.8, 1.2]$	[m]
	y_0	$[0.8, 1.2]$	[m]
	z_0	$[-s_w, 0]$	[m]

C Code

The following modules are the periodic path finding algorithm and the data fitting algorithm.

C.1 Periodic Path Finding Algorithm

```python
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
from itertools import product
from mpl_toolkits.mplot3d import axes3d
import copy
from matplotlib.animation import FuncAnimation
import pickle
import time
import ezc3d
def f(R, k1, k2, u1, u2, M, gc, r1, r2): # -dH/dR, helper function to keep everything manageable.
    part_spring1 = -k1 * (norm(R - r1) - u1) * (R - r1) / norm(R-r1)
    part_spring2 = -k2 * (norm(R - r2) - u2) * (R - r2) / norm(R-r2)
    gravity = np.array([0, -M*gc, 0])
    return part_spring1 + part_spring2 + gravity


def energy(R, p, k1, k2, u1, u2, M, gc, r1, r2):
    return [M*gc*R[1],\
            sum(p**2) / (2*M),\
            k1/2 * (norm(R - r1) - u1)**2,\
            k2/2 * (norm(R - r2) - u2)**2] # all the energies


def simulate(R0, p0, parameters, make_steps = False, stop_conditions = True):
    spring_parameters, system_parameters, numerical_parameters = parameters
    k1, k2, u1, u2 = spring_parameters
    gc, M, r1, r2, step_size, step_width, _ = system_parameters
    dt, N, mesh_discr, integration_method, _ , _= numerical_parameters

    # Data arrays for position, momentum and energy, the latter being 2d
    step_array = np.array([2*step_size, 0, 0])

    if norm(R0 - r1) >= u1: # only spring force if contracting
```

```
         virtual_k1 = 0
   else:
      virtual_k1 = k1

   if norm(R0 - r2) >= u2:
      virtual_k2 = 0
   else:
      virtual_k2 = k2

   # First step for both position and momentum using Backwards
      + Central
   if integration_method == "central":
      R1 = R0 + dt/M * p0

      p1 = p0 + dt * f(R0, virtual_k1, virtual_k2, u1, u2, M,
         gc, r1, r2)


   elif integration_method == "RK4":

      a1 = dt * f(R0, virtual_k1, virtual_k2, u1, u2, M, gc, r1
         ,r2)
      b1 = dt / M * p0

      a2 = dt * f(R0 + b1/2, virtual_k1, virtual_k2, u1, u2, M,
         gc, r1 ,r2)
      b2 = dt / M * (p0 + a1/2)

      a3 = dt * f(R0 + b2/2, virtual_k1, virtual_k2, u1, u2, M,
         gc, r1 ,r2)
      b3 = dt / M * (p0 + a2/2)

      a4 = dt * f(R0 + b3, virtual_k1, virtual_k2, u1, u2, M,
         gc, r1 ,r2)
      b4 = dt / M * (p0 + a3)

      a = 1/6 * (a1 + 2*a2 + 2*a3 + a4)
      b = 1/6 * (b1 + 2*b2 + 2*b3 + b4)

      R1 = R0 + b
      p1 = p0 + a



   # Calculating the energies
   E0 = energy(R0, p0, virtual_k1, virtual_k2, u1, u2, M, gc,
      r1, r2)
   E1 = energy(R1, p1, virtual_k1, virtual_k2, u1, u2, M, gc,
      r1, r2)
```

```

    # Initiating the data-arrays with the first 2 values.
    dataR = [R0, R1]
    dataP = [p0, p1]
    dataE = [E0, E1]
    steps = []
    prev_step = None

    for i in range(1,N): # Running the simulation

        # Conditions for termination
        if stop_conditions == True:
            if dataR[i][2] < r1[2] or dataR[i][2] > r2[2]: # Stop
                if beyond feet
                break

            elif dataR[i][1] < 0: # Stop if y position goes
                through the ground
                break

            elif dataR[i][0] > 2.1**step_size and make_steps ==
                False: # Then a step has been completed
                break # TODO this 2.1* is ugly and should be made
                    more precize, but it works...

        # There is no spring force if extended:
        if norm(dataR[i] - r1) >= u1:# extend == no spring force
            virtual_k1 = 0
        else:
            virtual_k1 = k1

        if norm(dataR[i] - r2) >= u2:
            virtual_k2 = 0
        else:
            virtual_k2 = k2

        # Do a step if the next position is closer, stop if there
            are two same steps in a row!
        if make_steps == True:
            if norm(dataR[i] - r1) > norm(dataR[i] - (r1 +
                step_array)):
                if prev_step == "left":
                    print("Tried a double left step!")
                    break
                steps.append(i)
                print("Step left!", i)
                r1 += step_array
                prev_step = "left"

```

```python
        if norm(dataR[i] - r2) > norm(dataR[i] - (r2 + step_array)):
            if prev_step == "right":
                print("Tried a double right step!")
                break

            r2 += step_array
            steps.append(i)
            print("Step right!", i)
            prev_step = "right"
    if integration_method == "central":
        dataR.append(dataR[i-1] + 2*dt/M * dataP[i])
        dataP.append(dataP[i-1] + 2*dt*f(dataR[i], virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2))
        dataE.append(energy(dataR[i], dataP[i], virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2))

    elif integration_method == "RK4":
        R0 = dataR[i]
        p0 = dataP[i]

        a1 = dt * f(R0, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
        b1 = dt / M * p0

        a2 = dt * f(R0 + b1/2, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
        b2 = dt / M * (p0 + a1/2)

        a3 = dt * f(R0 + b2/2, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
        b3 = dt / M * (p0 + a2/2)

        a4 = dt * f(R0 + b3, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
        b4 = dt / M * (p0 + a3)

        a = 1/6 * (a1 + 2*a2 + 2*a3 + a4)
        b = 1/6 * (b1 + 2*b2 + 2*b3 + b4)

        R1 = R0 + b
        p1 = p0 + a

        dataR.append(R1)
        dataP.append(p1)
        dataE.append(energy(dataR[i], dataP[i], virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2))

if make_steps:
        return dataP, dataR, dataE, steps
    else:
        return dataP, dataR, dataE


def iterative_simulation(parameters, y0_range, z0_range): #
    Brute force grid search within given parameters
    # Unpacking parameters:
    spring_parameters, system_parameters, numerical_parameters =
        parameters
    k1, k2, u1, u2 = spring_parameters
    gc, M, r1, r2, step_size, step_width, p0 = system_parameters
    dt, N, mesh_discr, integration_method, _,
        minimum_improvement = numerical_parameters

    best_score = 1e100 # Initially a large number

    all_good_scores = [ [], [], []]

    for y0, z0 in product(np.linspace(*y0_range, mesh_discr),
        np.linspace(*z0_range, mesh_discr)):
        # Initial conditions

        R0 = np.array([0,y0, z0])
        # p0 = M*np.array([5.83, 0, 0])
        p_half_period = p0
        R_half_period = np.array([R0[0] + step_size, R0[1],
            -1*R0[2]])

        R1 = R0 + dt/M * p0

        if norm(R1 - r1) >= u1: # only spring force if contracting
            virtual_k1 = 0
        else:
            virtual_k1 = k1

        if norm(R1 - r2) >= u2:
            virtual_k2 = 0
        else:
            virtual_k2 = k2


        p1 = p0 + dt * f(R0, virtual_k1, virtual_k2, u1, u2, M,
            gc, r1, r2)

        R_half_period_plus1 = np.array([R1[0] + step_size, R1[1],
            -1*R1[2]])
        p_half_period_plus1 = p1

        dataP, dataR, dataE = simulate(R0, p0, parameters) # We assume that x is monotone increasing, so only have to check last to elements but this need not be the best value of the simulation!!!
        dataP, dataR, dataE = np.array(dataP), np.array(dataR), np.array(dataE)

        # Determine score and compare with current best score. Only past 100 coordinates are taken into



        # score = norm(score_R - R_half_period)**2 + norm(score_P/M - p_half_period/M)**2 #/M to give less preference to momentum, make them the same order of magnitude
        indx = np.argmin(abs(dataR[:, 0] - R_half_period[0]))
        score = norm(dataR[indx] - R_half_period)**2 + norm(dataP[indx]/M - p_half_period/M)**2 #/M to give less preference to momentum, make them the same order of magnitude

        indx2 = np.argmin(abs(dataR[:, 0] - R_half_period_plus1[0]))
        score += norm(dataR[indx2] - R_half_period_plus1)**2 + norm(dataP[indx2]/M - p_half_period_plus1/M)**2 #/M to give less preference to momentum, make them the same order of magnitude

        if score < best_score:
            best_score = score
            best_R = dataR
            best_p = dataP
            best_E = dataE
            best_R_half = R_half_period
            best_P_half = p_half_period
            all_good_scores = [y0, z0, score]
            # print("New best score", score, indx, "(y0, z0) = ({}, {})".format(round(y0,5), round(z0,5)))

        # if score < 10: # arbitrary bound on 'good'
        #    # print("Found good score:", score, "(y0, z0) = ({}, {})".format( round(y0, 3), round(z0, 3)) )
        #    all_good_scores[0].append(y0)
        #    all_good_scores[1].append(z0)
        #    all_good_scores[2].append(score)

    return best_R, best_p, best_E, best_R_half, best_P_half, all_good_scores
```

```


def pos_mom_energy_plots(dataR, dataP, dataE, parameters, string = "None", save = False, pos = True, mom = True, E = True):
    dataR, dataP, dataE = np.array(dataR), np.array(dataP), np.array(dataE)
    spring_parameters, system_parameters, numerical_parameters = parameters
    k1, _, u1, _ = spring_parameters
    _, _, _, _, step_size, step_width, _ = system_parameters
    dt, _, _, _, _, _ = numerical_parameters
    if pos:
        plt.figure() # (x/y/z, t) plot
        plt.plot(dataR[:, 0], label = "x position")
        plt.plot(dataR[:, 1], label = "y position")
        plt.plot(dataR[:, 2], label = "z position")
        plt.legend()

        plt.xlabel("Time index [-]")
        plt.ylabel("$r$ [m]")
        plt.title("Position, dt={}, k={}, u={}, sz={}, sw={}".format(dt, k1, u1, step_size, step_width))
        if save: plt.savefig("20230410 position {} gait k={}, dt={}.pdf".format(string, k1, dt))
        plt.show()

    Etot = [sum(elm) for elm in list(dataE)]

    if mom:
        plt.figure() # (x/y/z, t) plot
        plt.plot(dataP[:, 0], label= "x momentum")
        plt.plot(dataP[:, 1], label= "y momentum")
        plt.plot(dataP[:, 2], label= "z momentum")
        plt.legend()

        plt.xlabel("Time index [-]")
        plt.ylabel("$p$ [kgms$^{-1}$]")
        plt.title("Momenta, dt={}, k={}, u={}, sz={}, sw={}".format(dt, k1, u1, step_size, step_width))
        if save: plt.savefig("20230410 momenta {} gait k={}, dt={}.pdf".format(string, k1, dt))
        plt.show()

    if E:
        plt.figure()
        plt.plot(dataE[:, 0], label = "Gravitational energy")
        plt.plot(dataE[:, 1], label = "Kinetic energy")
        plt.plot(dataE[:, 2], label = "Spring energy 1")
```

```python
    plt.plot(dataE[:, 3], label = "Spring energy 2")
    plt.plot(Etot,   label = "Total energy of system")
    plt.legend()

    plt.xlabel("Time index [-]")
    plt.ylabel("$E$ [J]")
    plt.title("Energies, dt={}, k={}, u={}, sz={}, sw={}".format(dt, k1, u1, step_size, step_width))
    if save: plt.savefig("20230410 Energies {} gait k={}, dt={}.pdf".format(string, k1, dt))
    plt.show()

def iterative_mesh_grid(parameters, y0_range, z0_range, verbose=True):
  _, _, numerical_parameters = parameters
  _, _, mesh_discr, _, convergence_rate, minimum_improvement = numerical_parameters
  virtual_parameters = copy.deepcopy(parameters)
  step_y0 = abs(y0_range[1] - y0_range[0]) / (mesh_discr - 1)
  step_z0 = abs(z0_range[1] - z0_range[0]) / (mesh_discr - 1)
  old_best_score = 10

  while step_y0 > 1e-12 and step_z0 > 1e-12: # A bit above machine precision

    if verbose: print("Reiterating, step y = {:#.3e} and step z = {:#.3e}".format(step_y0, step_z0))
    # Run the simulation:
    dataR, dataP, dataE, best_R_half, best_P_half, all_good_scores = iterative_simulation(copy.deepcopy(virtual_parameters), y0_range, z0_range)
    try:
      best_score = all_good_scores[2]
    except ValueError:
      best_score = 100

    if True:#
      # Update the mesh grid around the best found point
      y0_range = [dataR[0][1] - convergence_rate * step_y0, dataR[0][1] + convergence_rate*step_y0]
      z0_range = [dataR[0][2] - convergence_rate*step_z0, dataR[0][2] + convergence_rate*step_z0]

      # Update the new step size:
      step_y0 = abs(y0_range[1] - y0_range[0]) / (mesh_discr - 1)
      step_z0 = abs(z0_range[1] - z0_range[0]) / (mesh_discr - 1)
```

```
            if best_score != old_best_score and abs(best_score - old_best_score) / old_best_score < minimum_improvement: # if new coordinate only yields a very small (relative) improvement, stop.
                if verbose: print("Insufficient improvement")
                break
            old_best_score = best_score
        if verbose:
            print("Best score: {:#.3e}".format(best_score))
            print("where y0={} and z0={}".format(dataR[0][1], dataR[0][2]))
            print("y dist = [ {:#.3e}, {:#.3e}]".format(y0_range[0] - dataR[0][1], y0_range[1] - dataR[0][1]))
            print("z dist = [ {:#.3e}, {:#.3e}]".format(z0_range[0] - dataR[0][2], z0_range[1] - dataR[0][2]))
            print()
    return dataR, dataP, dataE, best_R_half, best_P_half, all_good_scores
```

C.2 Data Fitting Algorithm

```
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
from itertools import product
import copy
import pickle
import ezc3d
def f(R, k1, k2, u1, u2, M, gc, r1, r2): # -dH/dR, helper function to keep everything manageable.
    part_spring1 = -k1 * (norm(R - r1) - u1) * (R - r1) / norm(R-r1)
    part_spring2 = -k2 * (norm(R - r2) - u2) * (R - r2) / norm(R-r2)
    gravity = np.array([0, -M*gc, 0])
    return part_spring1 + part_spring2 + gravity


def energy(R, p, k1, k2, u1, u2, M, gc, r1, r2):
    return [M*gc*R[1],\
            sum(p**2) / (2*M),\
            k1/2 * (norm(R - r1) - u1)**2,\
            k2/2 * (norm(R - r2) - u2)**2] # all the energies


```

```python
def simulate(R0, p0, parameters, make_steps = False, stop_conditions = True):
   spring_parameters, system_parameters, numerical_parameters = parameters
   k1, k2, u1, u2 = spring_parameters
   gc, M, r1, r2, step_size, step_width = system_parameters
   dt, N, _, integration_method, _, _= numerical_parameters
   # Data arrays for position, momentum and energy, the latter being 2d
   step_array = np.array([2*step_size, 0, 0])

   if norm(R0 - r1) >= u1: # only spring force if contracting
         virtual_k1 = 0
   else:
      virtual_k1 = k1

   if norm(R0 - r2) >= u2:
      virtual_k2 = 0
   else:
      virtual_k2 = k2

   # First step for both position and momentum using Backwards + Central
   if integration_method == "central":
      R1 = R0 + dt/M * p0
      p1 = p0 + dt * f(R0, virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2)


   elif integration_method == "RK4":

      a1 = dt * f(R0, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
      b1 = dt / M * p0

      a2 = dt * f(R0 + b1/2, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
      b2 = dt / M * (p0 + a1/2)

      a3 = dt * f(R0 + b2/2, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
      b3 = dt / M * (p0 + a2/2)

      a4 = dt * f(R0 + b3, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
      b4 = dt / M * (p0 + a3)

      a = 1/6 * (a1 + 2*a2 + 2*a3 + a4)
      b = 1/6 * (b1 + 2*b2 + 2*b3 + b4)
```

```

        R1 = R0 + b
        p1 = p0 + a


    # Calculating the energies
    E0 = energy(R0, p0, virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2)
    E1 = energy(R1, p1, virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2)

    # Initiating the data-arrays with the first 2 values.
    dataR = [R0, R1]
    dataP = [p0, p1]
    dataE = [E0, E1]
    steps = []
    prev_step = None

    for i in range(1,N): # Running the simulation

        # Conditions for termination
        if stop_conditions == True:
            if dataR[i][2] < r1[2] or dataR[i][2] > r2[2]: # Stop if beyond feet
                break

            elif dataR[i][1] < 0: # Stop if y position goes through the ground
                break

            elif dataR[i][0] > 1.1*step_size and make_steps == False: # Then a half a step has been completed
                break

        # There is no spring force if extended:
        if norm(dataR[i] - r1) >= u1:# extend == no spring force
            virtual_k1 = 0
        else:
            virtual_k1 = k1

        if norm(dataR[i] - r2) >= u2:
            virtual_k2 = 0
        else:
            virtual_k2 = k2

        # Do a step if the next position is closer, stop if there are two same steps in a row!
        if make_steps == True:
            if norm(dataR[i] - r1) > norm(dataR[i] - (r1 + step_array)):
                if prev_step == "left":
                    print("Tried a double left step!")
                    break
                steps.append(i)
                print("Step left!", i)
                r1 += step_array
                prev_step = "left"

            if norm(dataR[i] - r2) > norm(dataR[i] - (r2 + step_array)):
                if prev_step == "right":
                    print("Tried a double right step!")
                    break

                r2 += step_array
                steps.append(i)
                print("Step right!", i)
                prev_step = "right"
        if integration_method == "central":
            dataR.append(dataR[i-1] + 2*dt/M * dataP[i])
            dataP.append(dataP[i-1] + 2*dt*f(dataR[i], virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2))
            dataE.append(energy(dataR[i], dataP[i], virtual_k1, virtual_k2, u1, u2, M, gc, r1, r2))

        elif integration_method == "RK4":
            R0 = dataR[i]
            p0 = dataP[i]

            a1 = dt * f(R0, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
            b1 = dt / M * p0

            a2 = dt * f(R0 + b1/2, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
            b2 = dt / M * (p0 + a1/2)

            a3 = dt * f(R0 + b2/2, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
            b3 = dt / M * (p0 + a2/2)

            a4 = dt * f(R0 + b3, virtual_k1, virtual_k2, u1, u2, M, gc, r1 ,r2)
            b4 = dt / M * (p0 + a3)

            a = 1/6 * (a1 + 2*a2 + 2*a3 + a4)
            b = 1/6 * (b1 + 2*b2 + 2*b3 + b4)

            R1 = R0 + b
            p1 = p0 + a

            dataR.append(R1)
            dataP.append(p1)
            dataE.append(energy(dataR[i], dataP[i], virtual_k1,
               virtual_k2, u1, u2, M, gc, r1, r2))

    if make_steps:
        return dataP, dataR, dataE, steps
    else:
        return dataP, dataR, dataE


def iterative_simulation(parameters, k_range, u_range): # Brute
    force grid search within given parameters
    # Unpacking parameters:
    dt_exp_arr, exp_data_R, exp_data_P, system_parameters,
        numerical_parameters = parameters
    data_x, data_y, data_z = np.array(exp_data_R).T
    data_px, data_py, data_pz = np.array(exp_data_P).T
    dt, _, _, _, _, _ = numerical_parameters

    # Now try to refind the k & u

    gc, M, r1, r2, step_size, step_width = system_parameters
    dt, N, mesh_discr, integration_method, _, _ =
        numerical_parameters

    best_score = 1e100 # Initially a large number


    for k, u in product(np.linspace(*k_range, mesh_discr),
        np.linspace(*u_range, mesh_discr)):
        # Initial conditions
        u1 = u2 = u
        k1 = k2 = k


        simulate_parameters = [ [k1, k2, u1, u2],
            system_parameters, numerical_parameters]
        dataP, dataR, dataE = simulate(exp_data_R[0],
            exp_data_P[0], simulate_parameters)
        dataP, dataR, dataE = np.array(dataP), np.array(dataR),
            np.array(dataE)

        # Determine score and compare with current best score.

```

```python
        score = 0
        dataR = np.array(dataR)
        dataP = np.array(dataP)
        exp_data_R = np.array(exp_data_R)
        exp_data_P = np.array(exp_data_P)
        dt_arr = np.arange(0, dt*len(dataR), dt)

        for i in range(0, len(exp_data_R)):
            dt_exp_i = dt_exp_arr[i]
            j = np.argmin(abs(dt_arr - dt_exp_i))

            # print(exp_data_R[indx], indx)
            score += norm(exp_data_R[i] - dataR[j])**2 + norm(exp_data_P[i]/M - dataP[j]/M)**2 #/M to give less preference to momentum, make them the same order of magnitude

        if score < best_score:
            best_score = score
            best_R = dataR
            best_p = dataP
            best_E = dataE
            best_param = [k, u, score]

    return best_R, best_p, best_E, best_param


def pos_mom_energy_plots(dataR, dataP, dataE, parameters, string = "None", save = False, pos = True, mom = True, E = True):
    dataR, dataP, dataE = np.array(dataR), np.array(dataP), np.array(dataE)
    spring_parameters, system_parameters, numerical_parameters = parameters
    k1, _, u1, _ = spring_parameters
    _, _, _, _, step_size, step_width = system_parameters
    dt, _, _, _ = numerical_parameters
    if pos:
        plt.figure() # (x/y/z, t) plot
        plt.plot(dataR[:, 0], label = "x position")
        plt.plot(dataR[:, 1], label = "y position")
        plt.plot(dataR[:, 2], label = "z position")
        plt.legend()

        plt.xlabel("Time index [-]")
        plt.ylabel("$r$ [m]")
        plt.title("Position, dt={}, k={}, u={}, sz={}, sw={}".format(dt, k1, u1, step_size, step_width))
        if save: plt.savefig("20230410 position {} gait k={}, dt={}.pdf".format(string, k1, dt))
        plt.show()

    Etot = [sum(elm) for elm in list(dataE)]

    if mom:
        plt.figure() # (x/y/z, t) plot
        plt.plot(dataP[:, 0], label= "x momentum")
        plt.plot(dataP[:, 1], label= "y momentum")
        plt.plot(dataP[:, 2], label= "z momentum")
        plt.legend()

        plt.xlabel("Time index [-]")
        plt.ylabel("$p$ [kgms$^{-1}$]")
        plt.title("Momenta, dt={}, k={}, u={}, sz={}, sw={}".format(dt, k1, u1, step_size, step_width))
        if save: plt.savefig("20230410 momenta {} gait k={}, dt={}.pdf".format(string, k1, dt))
        plt.show()

    if E:
        plt.figure()
        plt.plot(dataE[:, 0], label = "Gravitational energy")
        plt.plot(dataE[:, 1], label = "Kinetic energy")
        plt.plot(dataE[:, 2], label = "Spring energy 1")
        plt.plot(dataE[:, 3], label = "Spring energy 2")
        plt.plot(Etot,  label = "Total energy of system")
        plt.legend()

        plt.xlabel("Time index [-]")
        plt.ylabel("$E$ [J]")
        plt.title("Energies, dt={}, k={}, u={}, sz={}, sw={}".format(dt, k1, u1, step_size, step_width))
        if save: plt.savefig("20230410 Energies {} gait k={}, dt={}.pdf".format(string, k1, dt))
        plt.show()

def iterative_mesh_grid(parameters, k_range, u_range):
    _, _, _, _, numerical_parameters = parameters
    _, _, mesh_discr, _, convergence_rate, minimum_improvement = numerical_parameters
    virtual_parameters = copy.deepcopy(parameters)
    step_k = abs(k_range[1] - k_range[0]) / (mesh_discr - 1)
    step_u = abs(u_range[1] - u_range[0]) / (mesh_discr - 1)
    old_best_score = 1000

    counter = 0
    while step_k > 1e-8 and step_u > 1e-8: # A bit above machine precision
        counter += 1
        print("Reiterating, step y = {:#.3e} and step z = {:#.3e}".format(step_k, step_u))
        # Run the simulation:
        dataR, dataP, dataE, best_param = iterative_simulation(copy.deepcopy(virtual_parameters), k_range, u_range)
        best_score = best_param[2]

        # Update the mesh grid around the best found point
        k_range = [best_param[0] - convergence_rate * step_k, best_param[0] + convergence_rate*step_k]
        u_range = [best_param[1] - convergence_rate*step_u, best_param[1] + convergence_rate*step_u]

        # Update the new step size:
        step_k = abs(k_range[1] - k_range[0]) / (mesh_discr - 1)
        step_u = abs(u_range[1] - u_range[0]) / (mesh_discr - 1)
        if best_score != old_best_score and abs(best_score - old_best_score) / old_best_score < minimum_improvement and counter > 10: # if new coordinate only yields a very small (relative) improvement, stop.
            print("Insufficient improvement")
            break
        old_best_score = best_score

        print("Best score: {:#.4e}".format(best_score))
        print("where k={} and u={}".format(best_param[0], best_param[1]))
        print()
    return dataR, dataP, dataE, best_param
```