Differentiable simulation of mass-spring simulations with contact

Magí Romanyà Serrasolsas
Supervisor: Dr. Miguel Ángel Otaduy Tristán

October, 2023

Contents

List of Figures

Abstract

The discontinuous nature of contact interactions poses a challenge for gradient-based control in physical systems. In our study, we derived and implemented a differentiable mass-spring simulation to investigate the predicted gradients and the process of optimization with and without contact. Our research introduces a novel derivation of the back-propagation method, adding physical meaning to adjoint variables. Our findings suggest that the anomalies in gradients during contact interactions may be attributed to sensitivities in initial conditions, rather than the contact discontinuities.

Chapter 1

Introduction

1.1 Motivation and previous work

In computer graphics, physics simulation is the foundation of physics based animations, and it allows to create realistic and complex animations, which would be impossible to do by hand. The lack of human involvement in physics simulation is one of its strengths, allowing both saving animation time, as well as creating more accurate results. Admittedly, automation naturally brings a lack of control for the animator. Physics simulations are controlled by multiple parameters, which can describe a wide range of potentially unrelated values, from the initial conditions to the interactions between systems in the simulation. The outcome of modifying said parameters is therefore often complex and unpredictable. Moreover, simulation parameters can rapidly grow in number as the complexity of the simulation increases. Ultimately, this complexity is suffered by the animator who is designing the scene.

Researchers have tackled this problem by transforming it into an optimization problem. This way, it is possible to define an arbitrary target, do the optimization process, and get a simulated system trajectory tailored to our needs. There is no longer a need to interact with the complexity of simulation parameters. For instance, the animator could want the system to take a certain configuration in a particular frame, which is similar to the traditional key-frame animation. After an optimization process, the framework computes the appropriate set of simulation parameters which best fits the target defined by the animator. Early examples of animation control can be found in the works of McNarma *et al.*[7], and Wojtan *et al.*[9], where they controlled fluid simulations, and particle systems respectively using keyframes.

The notion of defining a target for our simulations transcends the realm of animation

control. The same framework can be used to find the best simulation parameters which minimizes the oscillation of a bridge, or trying to match a real world physics material with a virtual one. The generality of the method makes it applicable to a broad range of problems in different domains of knowledge. For instance, in [4], researchers address the problem of controlling robots that cut soft materials, seeking to minimize cutting forces. Additionally, ongoing research endeavors are delving into the control of learned-based simulations, exemplified in [6], where convolutional neural networks are used to control a turbulence model.

The framework of differentiable simulation has emerged from the need of control over simulations. What sets it apart from conventional simulations, is its capacity of computing gradients, which have a positive impact in the optimization process, making it faster and more accurate. It is within this context that the research objectives in this study take shape, seeking to study differentiable simulation from its basic principles and delve deeper into the problems that arise when introducing contact interactions.

1.2 Objectives

In this work, our primary objectives encompass the investigation of differentiable simulation, with a specific focus on the influence of contact interactions on the optimization process and the resultant computed gradients. These objectives are outlined as follows:

- Understanding of the adjoint method: We aim to obtain a profound comprehension of the adjoint method and make a rigorous derivation of the mathematics involved in computing the loss function gradients.

- Implementation of a mass-spring cloth differentiable simulation: To test the efficacy of the adjoint method, we will develop a differentiable simulation framework for a mass-spring cloth system.

- Validation of the computed gradients. In order to validate the mathematical derivation and the implementation of our simulator, we aim to validate the gradient predictions generated by our differentiable simulator.

- Investigate the impact of contact on the computed gradients: We intend to examine how contact interactions affect the computation of gradients and the optimization process.

- Evaluation of the optimization effectiveness: This study will include an analysis

of the optimization process, assessing the general efficacy of the adjoint method, both in scenarios with and without contact interactions.

Chapter 2

Simulation

Classical mechanics is governed by ordinary differential equations, which predict how systems should evolve over time using Newton's laws. Let us define a position vector $\mathbf{x}(t)$ which describes how a general system evolves over time. From here we can define the velocity and the acceleration vectors as the first and second time derivatives of the position vector respectively $\mathbf{v}(t) = \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}$, $\mathbf{a}(t) = \frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2}$. Newton's second law states that the acceleration vector is proportional to the force, which we will call $\mathbf{f}$, and in general is a function of time, the positions and the velocities $\mathbf{f}(\mathbf{x}, \mathbf{v}, t)$. The proportionality constant is the mass, which measures the resistance to the change of speed of an object.

$$\mathbf{f} = M\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} \tag{2.1}$$

In this work, we use a generalized notation which fits all the degrees of freedom of our simulation inside a single position vector $\mathbf{x}$. A result of this notation is that mass is no longer a scalar value, and is represented by a generalized mass matrix M. Solving the second order ordinary differentiable equation for a given force function $\mathbf{f}(\mathbf{x}, \mathbf{v}, t)$, Newton's second law predicts the trajectory of the system $\mathbf{x}(t)$. However, the result of a differentiable equation is not a single curve but a family of curves. In order to get a specific trajectory, a set of initial conditions is needed in the description of the system. Thus, fundamentally, to simulate a certain system we need its initial conditions, a proper description of the forces present, and finally a solver of differential equations which can compute how the system will evolve over time. In this chapter present first how we solve the second order differentiable equation, and later the model we use which defines our forces and our state vector.

2.1 Implicit Euler integration

The implicit Euler method is a simple numerical method for solving ordinary differential equations. Let us have a general first order ordinary differential equation together with an initial condition:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y), \quad y_0 = y(x_0) \tag{2.2}$$

Solving this ODE, would mean to find the function $y(x)$ such that its derivative with respect to x is the function $f(x, y)$ defined in the ODE. However, the implicit Euler method is a numerical method, and instead of a function $y(x)$ we will compute a finite amount of values $y_i = y(x_i)$. The discretization is done starting from the initial condition x_0, and incrementing its value by a tiny amount Δx, so that $x_i = x_0 + i \cdot \Delta x, \quad i \in \mathbb{N}$. The equation which rules how to compute the y_i values is the implicit Euler method and reads:

$$y_{i+1} = y_i + \Delta x f(x_{i+1}, y_{i+1}) \tag{2.3}$$

In the setting of solving the Newton equations, we are interested in solving a second order ordinary differentiable equation. It is possible to solve higher order equations using a first order solver, by using the definition of the velocity vector $\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} = \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t}$. This trick splits the second order differentiable equation into two first order equations $M\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \mathbf{f}$, and $\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}$. Applying the implicit Newton method to the last two first order differentiable equations, and taking our discretization step to be $\Delta t = h$ we get:

$$M\mathbf{v}(t+h) = M\mathbf{v}(t) + h\mathbf{f}(\mathbf{x}(t+h), \mathbf{v}(t+h), t+h) \tag{2.4a}$$

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{v}(t+h) \tag{2.4b}$$

The challenge in these equations is that, in general, the force is a nonlinear function, and obtaining $\mathbf{x}$ and $\mathbf{v}$ requires solving a system of nonlinear equations. The problem with these equations is that, in general, we can not solve them. The force term in the velocity equation may very well be not a linear function, making solving for $\mathbf{v}(t+h)$ not possible. In graphics simulation it is common to use numerical root finding methods to approximate a solution of this non linear system of equations. Usually a Newton-Rhapson like method is used, which essentially linearizes the non linear troublesome term, in this case the force, using a Taylor's series expansion, and iterates until convergence. It is also common to only make a single iteration, as it can generally be good enough for certain simulations.

The process of iterating multiple Newton iterations is not difficult, but in the context

of simulation, can be confusing. It is important to distinguish between a simulation step and a Newton iteration. From this point onward, we will always work in the same simulation step labeled by time $t \to t+h$, but we will make several Newton iterations until we reach convergence.

We will start by Taylor expanding our force around the positions and velocities from the last simulation step:

$$\mathbf{f}(t+h) = \mathbf{f}(t) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\left(\mathbf{x}(t+h) - \mathbf{x}(t)\right) + \frac{\partial \mathbf{f}}{\partial \mathbf{v}}\left(\mathbf{v}(t+h) - \mathbf{v}(t)\right) + \mathcal{O}(2) \tag{2.5}$$

Now comes the tricky part where we have to label each variable with an index which tells the Newton iteration they belong to. We are solving for the state at the $t+h$ timestep, so that the variables that we have to iterate are $\mathbf{v}^i(t+h) \equiv \mathbf{v}^i$, $\mathbf{x}^i(t+h) \equiv \mathbf{x}^i$ and $\mathbf{f}^i(\mathbf{x}(t+h), \mathbf{v}(t+h), t+h) \equiv \mathbf{f}^i$.

$$M\mathbf{v}^i = M\mathbf{v}(t) + h\mathbf{f}^i \tag{2.6a}$$

$$\mathbf{x}^i = \mathbf{x}(t) + h\mathbf{v}^i \tag{2.6b}$$

Note that the $\mathbf{x}(t)$ and $\mathbf{v}(t)$ vectors are the known state from time t, and they will remain constant throughout the iterations. Now the force Taylor expansion will be around the last Newton iteration positions and velocities:

$$\mathbf{f}^{i+1} = \mathbf{f}^i + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_i \left(\mathbf{x}^{i+1} - \mathbf{x}^i\right) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right|_i \left(\mathbf{v}^{i+1} - \mathbf{v}^i\right) \tag{2.7}$$

The first state $\mathbf{x}^0$ and $\mathbf{v}^0$ can be chosen freely, and it is the initial guess for our convergence process. There exist *warm start approaches*, which try to make an educated initial guess to achieve a faster convergence. One reasonable approach is to use the result of explicit Euler as the first guess. In our implementation we set the first guess to be the last state $\mathbf{x}(t)$ and $\mathbf{v}(t)$. With this in mind, it is possible to substitute the force expansion into equation (2.6a) to get:

$$M\mathbf{v}^{i+1} = M\mathbf{v}(t) + h\left(\mathbf{f}^i + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_i \left(\mathbf{x}^{i+1} - \mathbf{x}^i\right) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right|_i \left(\mathbf{v}^{i+1} - \mathbf{v}^i\right)\right) \tag{2.8}$$

Now we can make use of the position update described in (2.6b), and move all terms containing the unknown $\mathbf{v}^{i+1}$ to the left side.

$$\left(M - h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right|_i - h^2\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_i\right)\mathbf{v}^{i+1} = M\mathbf{v}(t) + h\mathbf{f}^i + h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_i \left(\mathbf{x}(t) - \mathbf{x}^i\right) - h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right|_i \mathbf{v}^i \tag{2.9}$$

For completeness sake we will also write explicitly the Newton step to solve for $\Delta\mathbf{v}^{i+1} \equiv \mathbf{v}^{i+1} - \mathbf{v}^i$:

$$\left(M - h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right|_i - h^2\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_i\right)\Delta\mathbf{v}^{i+1} = M\mathbf{v}(t) + h\mathbf{f}^i + h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_i\left(\mathbf{x}(t) - \mathbf{x}^i\right) - \left(M - h^2\left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_i\right)\mathbf{v}^i \tag{2.10}$$

2.2 Energies, forces and Jacobians

Using the implicit Euler method requires not only to compute forces, but also to compute force Jacobians matrices $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$, and $\frac{\partial \mathbf{f}}{\partial \mathbf{v}}$. In this section we will study the present forces in the simulation as well as the energies and the force Jacobians. The simulations done in this work only require two forces aside from gravity.

First we will take a look at the spring interaction, which joins two nodes, a and b. The spring is defined by two parameters, the stiffness k, which controls the amount of force required to deform the spring, and the rest length L_0, which is the length in which the spring will make no force. To describe the energy, force and force Jacobians, it is useful to define some values first. The separation length of the nodes defined as $L = |\mathbf{x}_a - \mathbf{x}_b|$, and the unit vector which points from node b to node a $\mathbf{u} = \frac{\mathbf{x}_a - \mathbf{x}_b}{|\mathbf{x}_a - \mathbf{x}_b|}$. With this we can write:

$$E = \frac{1}{2}k\left(L - L_0\right)^2 \tag{2.11a}$$

$$\mathbf{f}_a = -k\left(L - L_0\right)\mathbf{u} \tag{2.11b}$$

$$\frac{\partial \mathbf{f}_a}{\partial \mathbf{x}_a} = -\frac{k}{L}\left[\left(L - L_0\right)I + L_0\mathbf{u}\mathbf{u}^T\right] \tag{2.11c}$$

Since the force only depends on positions there is no force velocity Jacobian. The computed values are with respect to node a, but in practice we would need also the values derived with respect to b. Thanks to Newton's third law this is trivial as $\mathbf{f}_a = -\mathbf{f}_b$.

$$\frac{\partial \mathbf{f}_b}{\partial \mathbf{x}_b} = \frac{\partial \mathbf{f}_a}{\partial \mathbf{x}_b}, \quad \frac{\partial \mathbf{f}_a}{\partial \mathbf{x}_b} = -\frac{\partial \mathbf{f}_a}{\partial \mathbf{x}_a}, \quad \frac{\partial \mathbf{f}_b}{\partial \mathbf{x}_a} = -\frac{\partial \mathbf{f}_a}{\partial \mathbf{x}_a} \tag{2.12}$$

The other interaction we need to describe is the contact force. To generalize contact we can assume that locally all contact is a point colliding with a plane, defined by a normal vector $\mathbf{n}$, and a point $\mathbf{x}_p$. This plane divides all 3D space in two regions, which in the context of collisions we are going to call inside and outside of the plane. We will model contact as a spring which will push our point $\mathbf{x}$ away from inside the plane. For this reason it is useful to define a signed distance d, which will tell us how far away our point

is from the plane. Moreover, its sign will tell us wether we are inside or outside the plane. With the diagram shown in figure 2.1, it is easy to compute the signed distance.

$$d = (\mathbf{x} - \mathbf{x}_p) \cdot \mathbf{n} \tag{2.13}$$

With a plane defined with its normal pointing outward, a positive d means outside and a negative d inside. The contact energy is defined in terms of the Heaviside step function

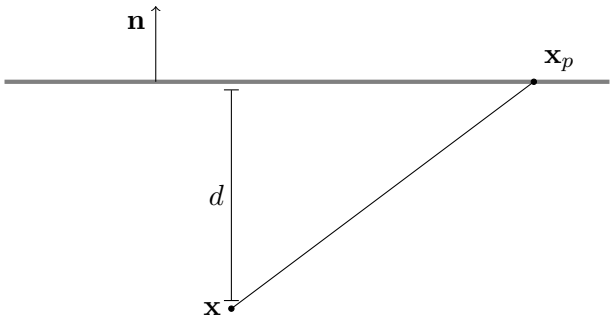


Figure 2.1: Plane contact schematic

which is inherently discontinuous $\Theta(x) = 1$ if $x > 0$, $\Theta(x) = 0$ otherwise. In order to compute the force and its Jacobian we need to know how to differentiate the signed length $\frac{\partial d}{\partial \mathbf{x}} = \mathbf{n}^T$.

$$E = \frac{1}{2} k_c \Theta(-d) d^2 \tag{2.14a}$$

$$\mathbf{f} = -k_c \Theta(-d) d\mathbf{n} \tag{2.14b}$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = -k_c \Theta(-d) \mathbf{n}\mathbf{n}^T \tag{2.14c}$$

However, the contact force itself is not discontinuous because as $d \to 0^-$, the force tends to zero. The result shows that the force Jacobian is indeed discontinuous because it does not tend to zero as $d \to 0^-$. In fact the Jacobian is a constant value inside of the plane and 0 outside. As discussed later in chapter 3, the discontinuity of this Jacobian poses a challenge to differentiable simulation because it can brake our assumptions of continuity that we need to define the derivatives.

2.3 Meshing a cloth model

There exist different ways of modeling a cloth, which in its core is a 2 dimensional elastic material. In this work, the cloth is made up of a mesh of nodes and springs which join the nodes. In the model, there are two kind of springs with separate purposes. The tension springs are responsible for the resistance to stretching and compression of the cloth. On the other hand, bending springs offer resistance to sharp bending angles, controlling the smoothness of the wrinkles. In our simulation environment, the bending springs are normal springs with reduced stiffness, and do not depend directly on the local bend angle of the cloth. However, using normal springs as bending springs is an approximation as they can also contribute to the stretch elasticity of the cloth. With a high enough k_b, bending springs would be exactly the same as tension springs. To position the springs, we start from a triangle mesh. The vertices of the mesh will be our mass nodes, and the edges of the triangles which join the vertices will be our tension springs. Bending springs are positioned joining the vertices which share an opposite edge with each other (see figure 2.2).

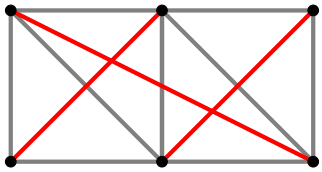


Figure 2.2: Arrangement of tension and bending springs in the mesh. The tension springs are positioned in the edges of the triangles (gray), while the bending springs join the vertices separated by an edge (red).

2.4 State formulation of the numerical integration

To close the simulation chapter, we will study how the simulation state in a particular timestep depends on the rest of states, which will be essential to understand in upcoming sections. The evolution of the states is entirely determined by the integration scheme of our simulation, which computes a new state each time-step. For readability purposes, we introduce the simulation state notation, which essentially joins positions and velocities in a single vector $\mathbf{s}_i \equiv (\mathbf{x}_i, \mathbf{v}_i)$. With this new notation, it is possible to write a general explicit integrator, which only depends on the previous sate, as $\mathbf{s}_{i+1} = \mathbf{g}(\mathbf{s}_i, \mathbf{p})$. The variable $\mathbf{p}$ represents a vector of simulation parameters, which remains constant throughout the simulation, and it is not of much importance in this particular section.

An implicit integrator, which computes the next state from both the current and the next state, would be written as $\mathbf{s}_{i+1} = \mathbf{h}(\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{p})$. The functions $\mathbf{g}$ and $\mathbf{h}$ will be defined by the integration scheme.

In this document we are interested in the implicit integration method which is the one we use. In particular we are interested in understanding how $\mathbf{s}_{i+1}$ depends on $\mathbf{s}_i$. While the description $\mathbf{s}_{i+1} = \mathbf{h}(\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{p})$ is correct, it can be difficult to work with because we have $\mathbf{s}_{i+1}$ on both sides of the equation, and in general there is no way to solve for $\mathbf{s}_{i+1}$. Fortunately, the implicit function theorem guarantees that there exists a function $\mathbf{F}$ so that $\mathbf{s}_{i+1} = \mathbf{F}(\mathbf{s}_i, \mathbf{p})$.

$$\mathbf{U}(\mathbf{p}, \mathbf{s}_i, \mathbf{s}_{i+1}) \equiv \mathbf{h}(\mathbf{p}, \mathbf{s}_i, \mathbf{s}_{i+1}) - \mathbf{s}_{i+1} = 0 \implies \exists \mathbf{F}(\mathbf{p}, \mathbf{s}_i) \quad \text{such that} \quad \mathbf{U}(\mathbf{p}, \mathbf{s}_i, \mathbf{F}(\mathbf{p}, \mathbf{s}_i)) = 0$$

In conclusion, we are allowed to treat an implicit method such as implicit Euler as if the next step depended only on the last $\mathbf{s}_{i+1} = \mathbf{F}(\mathbf{s}_i, \mathbf{p})$, as if it was an explicit method. The only difference is that now $\mathbf{F}$ is defined implicitly, which means that is not possible to write down as a mathematical expression. In the next chapter, we will see that the implicit function theorem also allows us to take first-order derivatives on $\mathbf{s}_{i+1} = \mathbf{F}(\mathbf{p}, \mathbf{s_i})$, which will be essential for the derivation of differentiable simulation.

Chapter 3

Differentiable simulation

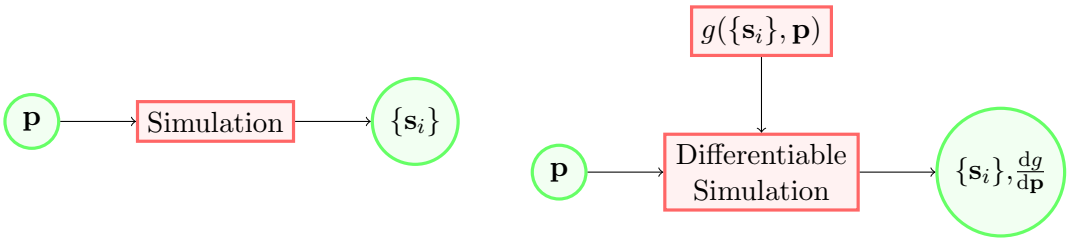


Figure 3.1: Schematic of forward simulation and differentiable simulation.

In the previous chapter we have described in detail how the simulation process works. The forward simulation problem goal is to make predictions of how a system should evolve over time, given an initial configuration of interactions and initial conditions. Consequently, it is possible to see the simulation process as a function, which accepts a configuration description, and returns a system trajectory. In this chapter, we unite all the system configuration in a parameters vector $\mathbf{p}$.

Differentiable simulation, on the other hand, is useful to solve the inverse or backward problem. Its goal is to find which initial system configuration is needed to obtain a desired trajectory from the simulation function. Mathematically, the target trajectory is defined by a loss or cost function. It is a scalar function that accepts a simulated trajectory and serves as a metric to measure how far away is it from our requirements. This description allows to define arbitrary requirements to our trajectory, which makes it very general, and in the end applicable to solve a wide range of problems. The approach to solve the inverse problem is to frame it as a minimization of the loss function, with the parameters as the control variables, and constrained by the physics. The existent minimization schemes, often require the computation of gradients, as they converge

faster and more accurately than their gradient free counterparts [5]. Consequently, we are greatly interested in the computation of these gradients, that will make the minimization process feasible. In conclusion, a differentiable simulation is a regular simulation which can also compute loss function gradients, required for the minimization routines.

In this chapter we are going too first derive the method we are going to use to make differentiable simulation fast and accurate as possible. Following the general method we are going to make the necessary computations to be able to apply the method to simulations which use the implicit Euler scheme. And last but not least, we compare our notation to the formulation found in literature and find the equivalences between them.

3.1 Derivation of the back propagation algorithm

In differentiable simulation we want to compute gradients of a given loss function, which in this document we will call g. The loss function in general will be a function of the parameters $\mathbf{p}$, which are the control variables, and the simulation state of each time step of the simulation $\mathbf{s}_i$.

$$g = g(\mathbf{p}, \{\mathbf{s}_i\}) \tag{3.1}$$

As discussed in 2.4, it is crucial to understand that the state variables $\mathbf{s}_i$ are not independent and themselves are a function of the last simulation step and also the parameters.

$$\mathbf{s}_{i+1} = \mathbf{s}_{i+1}(\mathbf{p}, \mathbf{s}_i) \tag{3.2}$$

There are different approaches to calculate the loss function gradient. The straight forward way would be to use finite differences, evaluating the loss function $N_{\text{parameters}}+1$ times, nudging a simulation parameter for each evaluation. This is not ideal as for each loss function evaluation one has to run an entire simulation making it computationally expensive.

In the remainder of this section we will derive a method, usually called the adjoint method or back propagation, to compute the loss gradient efficiently. This method does not scale badly with the number of parameters as finite difference methods do, and it has the approximate cost of running the simulation again after the forward pass has ended.

3.1.1 Forward propagation

A more efficient algorithm to find the loss function gradient requires the use of chain rule. Chain rule is useful because it breaks down a complicated derivative into more manageable parts. Knowing the dependence of the loss function and its variables written in equations (3.1) and (3.2) respectively, allows us to expand the loss gradient as follows:

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}} = \frac{\partial g}{\partial \mathbf{p}} + \sum_{i=0}^{N_s} \frac{\partial g}{\partial \mathbf{s}_i} \frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}} \tag{3.3}$$

Where N_s is the number of simulated steps. This makes $\mathbf{s}_0$, and thus $\mathbf{x}_0$ and $\mathbf{v}_0$, the initial conditions of the simulation.

Looking at the expansion, we know how to evaluate the partial derivatives of g. The problem now is that we do not know how to evaluate the total derivatives of the positions and the velocities with respect to parameters. Using the dependencies described in equation (3.2) we can also expand the total derivatives:

$$\frac{\mathrm{d}\mathbf{s}_{i+1}}{\mathrm{d}\mathbf{p}} = \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{p}} + \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i} \frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}} \tag{3.4}$$

Interestingly, we have found a way to compute $\frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}}$, from information of the previous simulation state. It is possible to get an intuition on why the state derivatives with respect to parameters depend on the previous states by thinking what happens when making a perturbation in the parameters. Changing the parameters in a simulation will change all the computed states and each state depends on the state before.

Provided that we know the derivatives of the initial state, one can calculate the derivatives of any ith state. We know that $\mathbf{s}_0$ is the initial conditions of the simulation, therefore it does not have a dependence on the previous state, making the derivative a direct calculation:

$$\frac{\mathrm{d}\mathbf{s}_0}{\mathrm{d}\mathbf{p}} = \frac{\partial \mathbf{s}_0}{\partial \mathbf{p}} \tag{3.5}$$

With this final result, we can compute the gradient of the loss function, computing the state derivatives with respect to parameters every simulation step from the step before. This algorithm is often called forward propagation, because to calculate the derivatives of a simulation state, it is necessary to know the derivatives of the state before, propagating the derivatives forward from the initial state.

The forward propagation algorithm has some drawbacks that could be improved. The derivative that we are calculating each state $\frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}}$, is a dense matrix with dimensions

nDoF $\times N_{\text{parameters}}$ where nDoF are the number of degrees of freedom of the simulation. In the literature the derivative is usually called sensitivity matrix, and it encodes all the information about how would the state change, under a perturbation of the parameters. As we will see in section 3.2, when using an implicit integrator, the matrix $\frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i}$ contains the inverse of a nDoF $\times$ nDoF matrix. Therefore, computing the matrix product that appears in equation (3.4) $\frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i}\frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}}$, requires the computation of nDoF linear solves.

One could think that this computation is required and that we need all the information that the sensitivity matrix contains. However, looking at the expansion of the loss gradient (3.1), we see that we only need a projection of the sensitivity matrix, because $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}}$ is a vector which lives in $N_{\text{parameters}}$ dimensions parameter space. In conclusion, it would be faster to have an algorithm which propagated vectors instead of matrices.

3.1.2 Backward propagation

There exists a different way of computing the loss function gradient, which avoids the computation of the full sensitivity matrix. Instead of making the direct chain rule expansion made in equation 3.3, it expands the loss function gradient as:

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}} = \frac{\partial g}{\partial \mathbf{p}} + \sum_{i=0}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i}\frac{\partial \mathbf{s}_i}{\partial \mathbf{p}} \tag{3.6}$$

The main difference is that the total derivatives are operating on a scalar function (the loss function), so that the result is a gradient vector. This vector is computed from vectors from following states, according to the relation:

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} = \frac{\partial g}{\partial \mathbf{s}_i} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{i+1}}\frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i} \tag{3.7}$$

This technique is named back-propagation, because we start with a known derivative in the last simulation state and propagate it backward to earlier state derivatives until finally reaching the initial state. Contrary to forward propagation, here we have a vector matrix product, which in the end it is going to require a single linear solve.

In forward propagation, we have obtained our expressions directly from the chain rule. Proving equations (3.6) and (3.7), requires a little more work, which is shown in the remainder of this section.

Proving (3.7)

Using the chain rule we can expand the derivatives of the loss function with respect to the simulation state to obtain:

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} = \frac{\partial g}{\partial \mathbf{s}_i} + \sum_{j=i+1}^{N_s} \frac{\partial g}{\partial \mathbf{s}_j} \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{s}_i} \tag{3.8}$$

There appears a summation because the loss function depends on all the states. It starts at $j = i + 1$ because all the states which follow $\mathbf{s}_i$, ultimately depend on $\mathbf{s}_i$, as a sort of initial condition. The special case where the state is the final state $\mathbf{s}_{N_s}$, the summation disappears and the full derivative is trivial to compute:

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{N_s}} = \frac{\partial g}{\partial \mathbf{s}_{N_s}} \tag{3.9}$$

Inside the summation in equation (3.8) there is a full derivative of the jth simulation state with respect to the ith simulation state ($j > i$). To understand this derivative it is helpful to understand first the simplest case, where $j = i + 1$.

$$\frac{\mathrm{d}\mathbf{s}_{i+1}}{\mathrm{d}\mathbf{s}_i} = \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i}$$

This is true by definition, because we have defined our state to depend only on the previous state $\mathbf{s}_{i+1}(\mathbf{s}_i, \mathbf{p})$, using the implicit function theorem. With this under our belt, we can use the chain rule to compute the general case $\frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{s}_i} = \frac{\partial \mathbf{s}_j}{\partial \mathbf{s}_{j-1}} \frac{\mathrm{d}\mathbf{s}_{j-1}}{\mathrm{d}\mathbf{s}_i}$, and keep expanding until we reach the base case.

$$\frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{s}_i} = \underbrace{\frac{\partial \mathbf{s}_j}{\partial \mathbf{s}_{j-1}} \frac{\partial \mathbf{s}_{j-1}}{\partial \mathbf{s}_{j-2}} \cdots \frac{\partial \mathbf{s}_{i+2}}{\partial \mathbf{s}_{i+1}}}_{\frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{s}_{i+1}}} \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i} = \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{s}_{i+1}} \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i} \quad (j > i) \tag{3.10}$$

Knowing how to take this last derivative, we can expand the summation in equation (3.8), do some algebra and get:

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} = \frac{\partial g}{\partial \mathbf{s}_i} + \frac{\partial g}{\partial \mathbf{s}_{i+1}} \frac{\mathrm{d}\mathbf{s}_{i+1}}{\mathrm{d}\mathbf{s}_i} + \underbrace{\sum_{j=i+2}^{N_s} \frac{\partial g}{\partial \mathbf{s}_j} \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{s}_{i+1}}}_{\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{i+1}} - \frac{\partial g}{\partial \mathbf{s}_{i+1}} (3.8)} \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i} = \frac{\partial g}{\partial \mathbf{s}_i} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{i+1}} \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i} \tag{3.11}$$

Finally, we have arrived at the expression that we wanted to prove.

Rearranging the loss gradient sum

In this section we will prove equation (3.6). Comparing it with the chain rule expansion in (3.3), we want to prove:

$$\sum_{i=0}^{N_s} \frac{\partial g}{\partial \mathbf{s}_i} \frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}} = \sum_{i=0}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{p}} \tag{3.12}$$

Note that this equality does not mean that each ith summation term is identical in each sum, we can only assume that the full sum is equal. That's why we are considering it as a rearrangement.

Proceeding from here is not obvious. For this reason we are going to formulate the following educated hypothesis.

$$\sum_{j=i}^{N_s} \frac{\partial g}{\partial \mathbf{s}_j} \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{p}} = \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} \frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}} + \sum_{j=i+1}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}} \tag{3.13}$$

We claim that this expression is the way to compute the last $N_s - i$ terms of the sum. This hypothesis equation is formulated with an arbitrary i index which can take integer values from $0 < i < N_s$. Consequently, we can study what does this hypothesis predict for the case of computing the whole summation, by taking $i = 0$.

$$\sum_{j=0}^{N_s} \frac{\partial g}{\partial \mathbf{s}_j} \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{p}} = \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_0} \underbrace{\frac{\mathrm{d}\mathbf{s}_0}{\mathrm{d}\mathbf{p}}}_{\frac{\partial \mathbf{s}_0}{\partial \mathbf{p}}} + \sum_{j=1}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}} = \sum_{j=0}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}} \tag{3.14}$$

This last expression confirms equation (3.12). In conclusion, proving that the sum can be rearranged is reduced to proving our hypothesis, which can be done by induction.

- Prove that the hypothesis (3.13) works for the base case $i = N_s$:

$$\sum_{j=N_s}^{N_s} \frac{\partial g}{\partial \mathbf{s}_j} \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{p}} = \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{N_s}} \frac{\mathrm{d}\mathbf{s}_{N_s}}{\mathrm{d}\mathbf{p}} + \cancelto{0}{\sum_{j=N_s+1}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}}} = \frac{\partial g}{\partial \mathbf{s}_{N_s}} \frac{\mathrm{d}\mathbf{s}_{N_s}}{\mathrm{d}\mathbf{p}} \quad \checkmark \tag{3.15}$$

  We are able to eliminate the sum because there are no states beyond N_s. We also made use of equation (3.9) to transform the total loss derivative into a partial derivative.

- Assume that the statement holds for i.

- Prove that the statement is true for $i-1$:

$$\sum_{j=i-1}^{N_s} \frac{\partial g}{\partial \mathbf{s}_j} \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{p}} = \frac{\partial g}{\partial \mathbf{s}_{i-1}} \frac{\mathrm{d}\mathbf{s}_{i-1}}{\mathrm{d}\mathbf{p}} + \sum_{j=i}^{N_s} \frac{\partial g}{\partial \mathbf{s}_j} \frac{\mathrm{d}\mathbf{s}_j}{\mathrm{d}\mathbf{p}} \tag{3.16a}$$

$$= \frac{\partial g}{\partial \mathbf{s}_{i-1}} \frac{\mathrm{d}\mathbf{s}_{i-1}}{\mathrm{d}\mathbf{p}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} \frac{\mathrm{d}\mathbf{s}_i}{\mathrm{d}\mathbf{p}} + \sum_{j=i+1}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}} \tag{3.16b}$$

$$= \frac{\partial g}{\partial \mathbf{s}_{i-1}} \frac{\mathrm{d}\mathbf{s}_{i-1}}{\mathrm{d}\mathbf{p}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} \left(\frac{\partial \mathbf{s}_i}{\partial \mathbf{p}} + \frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}} \frac{\mathrm{d}\mathbf{s}_{i-1}}{\mathrm{d}\mathbf{p}} \right) + \sum_{j=i+1}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}} \tag{3.16c}$$

$$= \underbrace{\left(\frac{\partial g}{\partial \mathbf{s}_{i-1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{s}_{i-1}} \right)}_{\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{i-1}} \text{ (3.7)}} \frac{\mathrm{d}\mathbf{s}_{i-1}}{\mathrm{d}\mathbf{p}} + \underbrace{\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{p}} + \sum_{j=i+1}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}}}_{\sum_{j=i}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}}} \tag{3.16d}$$

$$= \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{i-1}} \frac{\mathrm{d}\mathbf{s}_{i-1}}{\mathrm{d}\mathbf{p}} + \sum_{j=i}^{N_s} \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_j} \frac{\partial \mathbf{s}_j}{\partial \mathbf{p}} \quad \checkmark \tag{3.16e}$$

With this final proof, the derivation of the back propagation method is compleated.

3.2 Computing the partial derivatives

Backward propagation allows to calculate full total derivatives by applying the chain rule repeatedly, combining a series of partial derivatives. In this section we will focus on how to compute the required partial derivatives, to have the complete picture on how to evaluate simulation and loss function gradients. Looking back on equations (3.6) and (3.7) we see that we need to know how to calculate the derivatives:

$$\frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{p}}, \quad \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i}$$

The partial derivatives we want to compute depend on how we compute one state from the previous one, *i.e.* the integration scheme. In the case of this study, we have used an implicit integration method. The partial derivatives computed here will only stand for this kind of integrator and will change in simulators with other integration techniques.

As mentioned in section 2.1, the implicit Euler method equations can not be analytically solved, and some kind of numerical root finding algorithm must be used. This kind of algorithms, such as Newton-Rhapson, usually require multiple steps to converge, which are done in the forward simulation loop. Here we want to compute derivatives on how changing one simulation state affects the next one. If we were to take into account all the numerical root finding with multiple iterations, the computation of this derivative could get complicated fast. However, it is possible to skip this complexity by assuming that the states computed in forward simulation fulfill the implicit Euler equations:

$$M_i(\mathbf{v}_{i+1} - \mathbf{v}_i) - h\mathbf{f}(\mathbf{x}_{i+1}, \mathbf{v}_{i+1}, \mathbf{p}) = 0 \tag{3.17a}$$

$$\mathbf{x}_{i+1} - \mathbf{x}_i - h\mathbf{v}_{i+1} = 0 \tag{3.17b}$$

We have to keep in mind that this assumption is only an approximation. During the forward pass the states will be computed with a certain number of Newton iterations, which will affect their accuracy. Deviations from the exact result will ultimately induce error into the loss gradient computation, which could make the minimization process more difficult. To avoid this source of error, it is necessary to ensure that the implicit Euler equations are fulfilled each time-step, by doing multiple Newton steps until convergence.

Previously in section 2.4, we discussed how to define the states as implicit functions using the implicit function theorem, and mentioned that the theorem also allawed to compute derivatives. Now we are going to make use of the theorem to compute the

desired derivatives, assuming that equations (3.17) hold. This assumption can be written compactly in simulation state notation as $\mathbf{U}(\bar{\mathbf{s}}_i, \mathbf{s}_{i+1}, \mathbf{p}) = 0$. We have written $\bar{\mathbf{s}}_i$, because equations 3.17b assume it as a given and not a function $\bar{\mathbf{s}}_i \neq \mathbf{F}(\mathbf{s}_{i-1}, \mathbf{p})$. The function $\mathbf{s}_i$ is defined by another constraint $\mathbf{U}(\bar{\mathbf{s}}_{i-1}, \mathbf{s}_i, \mathbf{p})$, belonging to a different simulation step. Using the implicit function theorem we can write:

$$0 = \frac{\partial \mathbf{U}}{\partial \bar{\mathbf{s}}_i} + \frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}} \frac{\mathrm{d}\mathbf{F}(\bar{\mathbf{s}}_i, \mathbf{p})}{\mathrm{d}\bar{\mathbf{s}}_i}, \quad 0 = \frac{\partial \mathbf{U}}{\partial \mathbf{p}} + \frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}} \frac{\mathrm{d}\mathbf{F}(\bar{\mathbf{s}}_i, \mathbf{p})}{\mathrm{d}\mathbf{p}} \tag{3.18}$$

These equations are matrix equations, where we know the partial derivatives of $\mathbf{U}$ and the unknowns are the total derivatives. The variables $\hat{\mathbf{s}}_i$ and $\mathbf{p}$ are independent of each other, allowing us to transform the full derivatives of function $\mathbf{F}$ into partial derivatives. Moreover, in the partial derivative with respect to parameters, it is possible to change the constant state to the full description $\bar{\mathbf{s}}_i \to \mathbf{s}_i(\mathbf{s}_{i-1}, \mathbf{p})$, because the partial derivative leaves it constant by definition. In the derivative with respect to the state, it is also possible to introduce the full state without changing the expression.

$$\frac{\mathrm{d}\mathbf{F}(\bar{\mathbf{s}}_i, \mathbf{p})}{\mathrm{d}\bar{\mathbf{s}}_i} = \frac{\partial \mathbf{F}(\mathbf{s}_i, \mathbf{p})}{\partial \mathbf{s}_i}, \quad \frac{\mathrm{d}\mathbf{F}(\bar{\mathbf{s}}_i, \mathbf{p})}{\mathrm{d}\mathbf{p}} = \frac{\partial \mathbf{F}(\mathbf{s}_i, \mathbf{p})}{\partial \mathbf{p}} \tag{3.19}$$

The partial derivatives of $\mathbf{U}$ are straight forward to compute using the implicit Euler equations (3.17).

$$\frac{\partial \mathbf{U}}{\partial \bar{\mathbf{s}}_i} = \begin{pmatrix} -M_i & 0 \\ 0 & -I \end{pmatrix}, \quad \frac{\partial \mathbf{U}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \\ 0 \end{pmatrix}, \quad \frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}} = \begin{pmatrix} M_i - h\frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{v}_{i+1}} & -h\frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{x}_{i+1}} \\ I & -h \end{pmatrix} \tag{3.20}$$

Where $\mathbf{c}$ refears to equation (3.17a). We purposely left $\mathbf{c}$ in the final formulation and not compute the partial derivative because this way we can stay in a general and not induce parameter dependace to any specific function. In order to solve the matrix equation described in (3.20), we need to invert the matrix $\frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}}$ which results in:

$$\left(\frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}}\right)^{-1} = \begin{pmatrix} A_{i+1}^{-1} & hA_{i+1}^{-1}\frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{x}_{i+1}} \\ hA_{i+1}^{-1} & I + h^2 A_{i+1}^{-1}\frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{x}_{i+1}} \end{pmatrix} \tag{3.21}$$

In this equation we encounter the inverse of the matrix $A_{i+1} = M_i - h\frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{v}_{i+1}} - h^2\frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{x}_{i+1}}$, which already appeared in the implicit Euler integration scheme in section 2.1. The mass is an index lower with respect to the rest because it is the one used when solving the $i+1$ step.

With the this inverse matrix computed, it is trivial to solve for our unknowns $\frac{\partial \mathbf{F}(\mathbf{x}_i, \mathbf{p})}{\partial \mathbf{s}_i}$,

and $\frac{\partial \mathbf{F}(\mathbf{x}_i, \mathbf{p})}{\partial \mathbf{p}}$.

$$\frac{\partial \mathbf{F}(\mathbf{s}_i, \mathbf{p})}{\partial \mathbf{s}_i} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}}\right)^{-1} \frac{\partial \mathbf{U}}{\partial \bar{\mathbf{s}}_i} = \begin{pmatrix} A_{i+1}^{-1} M_i & h A_{i+1}^{-1} \frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{x}_{i+1}} \\ h A_{i+1}^{-1} M_i & I + h^2 A_{i+1}^{-1} \frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{x}_{i+1}} \end{pmatrix} \tag{3.22}$$

$$\frac{\partial \mathbf{F}(\mathbf{s}_i, \mathbf{p})}{\partial \mathbf{p}} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}}\right)^{-1} \frac{\partial \mathbf{U}}{\partial \mathbf{p}} = -\begin{pmatrix} A_{i+1}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \\ h A_{i+1}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \end{pmatrix} \tag{3.23}$$

With this, we have successfully computed the partial derivatives necessary to implement back-propagation in an implicit integration setting. The inverse of the A_{i+1} matrix, in our final expressions is a performance concern. Computing the inverse of a nDoF $\times$ nDoF sparse matrix in every backward step is as slow as forward propagation. However, in section 3.3 we will see that it is not necessary to do the inverse computation, and the problem can be solved with a single linear solve for each backward step.

To finish the section we can write the maths of back-propagation explicitly substituting the $\frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i}$ partial derivative in (3.7).

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_i} = \frac{\partial g}{\partial \mathbf{v}_i} + \left(h \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{i+1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_{i+1}}\right) A_{i+1}^{-1} M_i \tag{3.24a}$$

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_i} = \frac{\partial g}{\partial \mathbf{x}_i} + \left(h \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{i+1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_{i+1}}\right) A_{i+1}^{-1} h \frac{\partial \mathbf{f}_{i+1}}{\partial \mathbf{x}_{i+1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{i+1}} \tag{3.24b}$$

While computing $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i}$ in the backward pass, it is possible to also compute the summation that appears in equation (3.6), one term at a time.

$$S_{i+1} \equiv \frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_{i+1}} \frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{p}} = -\left(h \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{i+1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_{i+1}}\right) A_{i+1}^{-1} \frac{\partial \mathbf{c}}{\partial \mathbf{p}} \tag{3.25}$$

Here we compute the $i+1$th summation term to make the indices match with the backward iteration that computes the $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i}$. Because of this index offset, the S_0 term can not be computed during the backward pass, and it has to be computed immediately after. Furthermore, S_0 does not follow equation (3.25). The initial conditions are only a function of the parameters $\mathbf{s}_0(\mathbf{p})$, and are not defined from a previous state. Consequently, the derivative $\frac{\partial \mathbf{s}_0}{\partial \mathbf{p}}$ can not be computed by the general rule (3.23), and has to be computed for each particular case. This last S_0 term is crucial, because it is the one which holds all the gradient information regarding the initial conditions.

3.3 Final back propagation algorithm

Now that we have understood the math which describes how to propagate derivatives both forward and backward, we can think about the implementation. As discussed previously, backward propagation is the preferred method, as it is faster and more memory efficient to propagate vectors instead of whole matrices. For this reason, this section will only tackle the back-propagation algorithm.

The first implementation challenge is that the derivative propagation is backward, while the simulation loop is forward. This means that back-propagation can not be implemented inside the traditional simulation loop and it is necessary to introduce a secondary backward loop after the forward simulation finishes. In the backward loop we will require the following vectors and marices from the forward simulation pass:

$$\mathbf{x}_i, \mathbf{v}_i, \frac{\partial \mathbf{f}}{\partial \mathbf{x}_i}, A_i, \frac{\partial \mathbf{c}}{\partial \mathbf{p}}$$

The state has to be stored during the forward simulation loop, to avoid recomputing the simulation. The matrices can be computed from the saved state using the same machinery from ordinary simulation. Alternatively, they also can be stored during the forward pass to avoid computation, although it can be really memory intensive.

The second implementation challenge is that equations (3.24), implies that it is necessary to calculate the inverse of the A matrix every backward iteration, which is a slow operation. However, we can notice that the A^{-1} is always multiplied by the same row vector in all three equations. The result of the multiplication is another row vector that we will name α.

$$\alpha \equiv \left(h \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{i+1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_{i+1}} \right) A_{i+1}^{-1} \tag{3.26}$$

The new α vector is called the adjoint vector [7], and it is useful because it can be calculated without inverting the A matrix, and solving a linear system instead. This can be done by multiplying by A_{i+1} at both sides of the equation 3.26, and solving for α. In the same way as the forward linear solve, this system of linear equations is also sparse, making it suitable for certain specialized faster solvers.

Finally we can write down how to implement back-propagation to calculate the loss function gradient with respect to parameters.

In each iteration of the backward loop, we calculate the full derivative of the loss function with respect to the state. However, the loss function gradient with respect to parameters is not calculated completely until the backward loop has ended and the initial conditions term has been taken into account.

Algorithm 1 Back-propagation

Require: $\mathbf{x}_i, \mathbf{v}_i, \frac{\partial \mathbf{f}}{\partial \mathbf{x}_i}, \frac{\partial \mathbf{c}}{\partial \mathbf{p}}, A_i, \frac{\partial g}{\partial \mathbf{x}_i}, \frac{\partial g}{\partial \mathbf{v}_i}, \frac{\partial g}{\partial \mathbf{p}}$ ▷ Variables from the forward pass

$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}} \leftarrow \frac{\partial g}{\partial \mathbf{p}}$ ▷ Initialize gradients

$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{N_s}} \leftarrow \frac{\partial g}{\partial \mathbf{x}_{N_s}}$

$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_{N_s}} \leftarrow \frac{\partial g}{\partial \mathbf{v}_{N_s}}$

for $i = N_s - 1$ to 0 **do** ▷ Backward loop

Solve linear system $A_{i+1}\alpha = \left(h\frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{i+1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_{i+1}}\right)$

$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_i} \leftarrow \frac{\partial g}{\partial \mathbf{x}_i} + h\alpha\frac{\partial \mathbf{f}}{\partial \mathbf{x}_{i+1}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_{i+1}}$

$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_i} \leftarrow \frac{\partial g}{\partial \mathbf{v}_i} + \alpha M$

$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}} \mathrel{+}= -\alpha\frac{\partial \mathbf{c}}{\partial \mathbf{p}}$

end for

$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}} \mathrel{+}= \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_0}\frac{\partial \mathbf{x}_0}{\partial \mathbf{p}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_0}\frac{\partial \mathbf{v}_0}{\partial \mathbf{p}}$ ▷ Add the initial conditions term

3.4 Equivalence with other notations

In the literature, the back propagation method is also called the *adjoint method*, and it is normally formulated in terms of *adjoint variables*. Specifically Wojtan *et al.* [9], define the simulation state $\mathbf{s}_i$, and from there they introduce the adjoint variable, which they call $\hat{\mathbf{s}}_i$. While the state is the variable which gets updated in the forward simulation pass, the adjoint variable is computed in the backward pass, and it is used for the computation of the desired gradient $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}}$. In our derivation, we have not defined an adjoint variable, but we can define one now identifying the object we propagate backward.

$$\tilde{\mathbf{s}}_i = \left(\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i}\right)^T \tag{3.27}$$

Note that we introduced our adjoint with a different name $\tilde{\mathbf{s}}_i$ on purpose, because we do not know yet the relationship between ours and $\hat{\mathbf{s}}_i$. With this notation we can re-write our derived equations (3.6) and (3.7) in a similar way as in [9].

$$\tilde{\mathbf{p}} = \sum_i \tilde{\mathbf{s}}_i \left(\frac{\partial \mathbf{s}_i}{\partial \mathbf{p}}\right)^T + \left(\frac{\partial g}{\partial \mathbf{p}}\right)^T, \quad \tilde{\mathbf{s}}_i = \left(\frac{\partial \mathbf{s}_{i+1}}{\partial \mathbf{s}_i}\right)^T \tilde{\mathbf{s}}_{i+1} + \left(\frac{\partial g}{\partial \mathbf{s}_i}\right)^T \tag{3.28}$$

There exists a subtle difference between notations around how the simulation state dependence is defined. In our work, as discussed in section 2.4, we have defined the states as a function of the previous state and the parameters only $\mathbf{s}_{i+1} = \mathbf{F}(\mathbf{s}_i, \mathbf{p})$, using the implicit function theorem. On the other hand, Wojtan *et al.* define the state as an explicit function, which depends on both the present and previous sates, as well as the parameters $\mathbf{s}_{i+1} = \mathbf{h}(\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{p})$. Ultimately, this difference translates into the computation of the adjoint variables.

$$\hat{\mathbf{s}}_i = \left(\frac{\partial \mathbf{h}(\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{p})}{\partial \mathbf{s}_i}\right)^T \hat{\mathbf{s}}_{i+1} + \left(\frac{\partial \mathbf{h}(\mathbf{s}_{i-1}, \mathbf{s}_i, \mathbf{p})}{\partial \mathbf{s}_i}\right)^T \hat{\mathbf{s}}_i + \left(\frac{\partial g}{\partial \mathbf{s}_i}\right)^T \tag{3.29a}$$

$$\tilde{\mathbf{s}}_i = \left(\frac{\partial \mathbf{F}(\mathbf{s}_i, \mathbf{p})}{\partial \mathbf{s}_i}\right)^T \tilde{\mathbf{s}}_{i+1} + \cancel{\left(\frac{\partial \mathbf{F}(\mathbf{s}_{i-1}, \mathbf{p})}{\partial \mathbf{s}_i}\right)^T} \tilde{\mathbf{s}}_i + \left(\frac{\partial g}{\partial \mathbf{s}_i}\right)^T \tag{3.29b}$$

The first equation is the one used in Wojtan's paper while the second equation is equivalent to our equation (3.7). In the end, this difference generates slightly different adjoint variables for each definition. The exact relationship between both adjoints can be computed by understanding how the functions $\mathbf{h}$ and $\mathbf{F}$ are defined making use of the implicit function theorem.

$$\mathbf{U}(\mathbf{s}_i, \mathbf{F}(\mathbf{s}_i, \mathbf{p}), \mathbf{p}) = \mathbf{s}_{i+1} - \mathbf{h}(\mathbf{s}_i, \mathbf{F}(\mathbf{s}_i, \mathbf{p}), \mathbf{p}) = 0 \tag{3.30a}$$

$$\frac{\mathrm{d}\mathbf{U}}{\mathrm{d}\mathbf{s}_i} = \frac{\partial \mathbf{U}}{\partial \mathbf{s}_i} + \frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}} \frac{\partial \mathbf{F}(\mathbf{s}_i, \mathbf{p})}{\partial \mathbf{s}_i} = 0 \tag{3.30b}$$

$$\implies \frac{\partial \mathbf{F}(\mathbf{s}_i, \mathbf{p})}{\partial \mathbf{s}_i} = -\left(\frac{\partial \mathbf{U}}{\partial \mathbf{s}_{i+1}}\right)^{-1} \frac{\partial \mathbf{U}}{\partial \mathbf{s}_i} = \left(I - \frac{\partial \mathbf{h}(\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{p})}{\partial \mathbf{s}_{i+1}}\right)^{-1} \frac{\partial \mathbf{h}(\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{p})}{\partial \mathbf{s}_i} \tag{3.30c}$$

Knowing how the derivatives are related to each other, it is possible to find the relationship between both adjoint variables $\hat{\mathbf{s}}$ and $\tilde{\mathbf{s}}$.

$$\tilde{\mathbf{s}}_i = \left[I - \left(\frac{\partial \mathbf{h}(\mathbf{s}_{i-1}, \mathbf{s}_i, \mathbf{p})}{\partial \mathbf{s}_i}\right)^T\right] \hat{\mathbf{s}}_i \tag{3.31}$$

With this we have shown that the method we have derived is completely equivalent to the work done by Wojtan *et al.* [9]. Furthermore, by finding the connection between the adjoint states and the full derivative $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i}$, we have provided a meaning to the adjoint states, which were viewed only as a mathematical tool to compute $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}}$ efficiently. In the Siggraph differentiable simulation curse [2], Miles Macklin defines his adjoint variable as $\frac{\partial g}{\partial \mathbf{s}_i}$, which is a similar expression to our adjoint. However, in our notation this partial derivative does not depend on the simulation and only on the loss function, which would

not require any back propagation to compute but would not help to compute $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}}$ either. This final equation is the final conclusion when comparing Wojtan's adjoint states and ours, and it shows the equivalence between notations.

$$\frac{\mathrm{d}g}{\mathrm{d}\mathbf{p}} = \frac{\partial g}{\partial \mathbf{p}} + \sum_{i=0}^{N_s} \overbrace{\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s_i}}}^{\tilde{\mathbf{s}}_i} \underbrace{\left(I - \frac{\partial \mathbf{h}_i}{\partial \mathbf{s_i}}\right)^{-1}}_{\hat{\mathbf{s}}_i} \frac{\partial \mathbf{h}_i}{\partial \mathbf{p}} \tag{3.32}$$

Chapter 4

Results

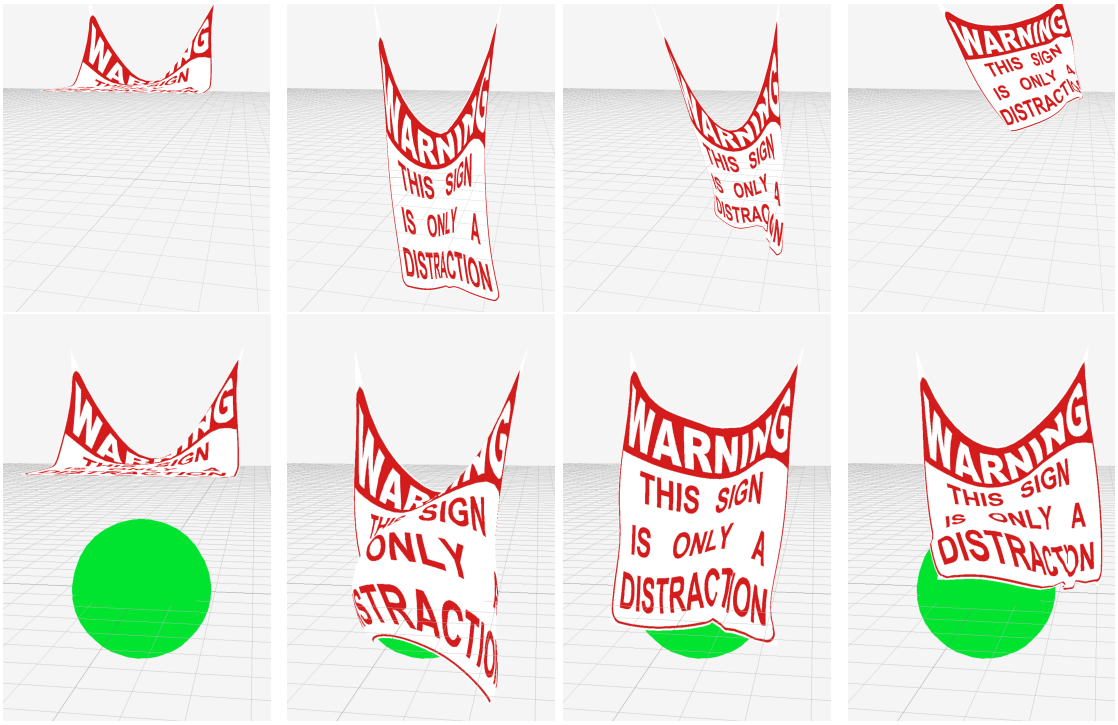


Figure 4.1: Rendered frames of the hanging cloth simulation setting without contact (top) and with contact (bottom). The frames are the 25, 50, 75, and 100, from left to right.

In this chapter we are going to use our differentiable simulation to test the back propagation algorithm. First, we validate that the predicted gradients are correct. Then we introduce contact and test how it affects the computation of the gradients. Lastly,

we study the minimization process and the effectiveness of the method, with different minimization schemes, both with and without contact.

4.1 Validation of gradients

As we have discussed in chapter 3, we want to calculate the gradient of the objective function efficiently. After the theoretical view in the subject we want to ensure that the back propagation algorithm 1 can indeed generate the correct gradients. This can be done by computing the objective function gradient using both finite differences and back propagation. Computing gradients using finite differences is somewhat trivial to implement, which is ideal for this situation as we can trust that the gradients will be correct. However finite differences gradients, unlike back propagated ones, are an approximation, and we expect some minor differences between the two results. We also expect that this difference will go to zero by using finer approximations in finite differences.

To test weather the gradients match one another, we need a testing environment. In the case of this study, we have a square cloth simulation hanged from both extremes. The cloth initial condition is flattened parallel to the ground. When the simulation starts, it is affected by gravity which makes the cloth oscillate back and forth, as depicted in the top frames in figure 4.1. The simulation uses the implicit Euler integration scheme with multiple steps.

In addition to our simulation environment, to test the gradients we need a definition of the objective function. In our testing we have used a simple mean-square loss function which computes the difference between the positions and velocities to some target values.

$$g(\{\mathbf{x}\},\{\mathbf{v}\},\mathbf{p}) = \sum_{i=0}^{N_s} w_i \left[(\mathbf{x}_i - \mathbf{x}_i^*)^2 + (\mathbf{v}_i - \mathbf{v}_i^*)^2\right] \tag{4.1}$$

The $\mathbf{x}_i^*$ and $\mathbf{v}_i^*$ variables represent the target position in frame i and the target velocity in frame i respectively. Using this definition of the loss function means that the target values must be defined beforehand. In a real-world scenario, the target values could be obtained from real-world data, and we could be trying to match our virtual cloth to some specific type of cloth. However, in our testing we use artificially generated target values. We run a forward simulation and save the positions and velocities of this forward pass, later to be used as the target variables. Generating the target data artificially means that the loss function could potentially be zero. This is virtually

impossible with real world data. In general, the simulated trajectory will not match with full accuracy the recorded trajectory, making the output of the objective function always > 0. Ultimately, this is the reason why the goal of differentiable simulation is to minimize the objective function, instead of finding its roots.

The weights w_i in the loss function definition, are used to give relative importance to each frame. To do our testing, we have selected the weights to be $w_{i<N_s} = 0$ and $w_{N_s} = 1$, which means that only the last state of the simulation is being constrained. This serves to stress test back propagation, which will have to propagate all the way from the last step until the initial conditions in order to compute the correct cost function gradient.

4.1.1 One dimensional gradients

The first tests involve computing the gradient with respect to a single parameter. These experiments are the simplest and are the first step at validating the back propagation gradients.

Gradients with respect to the tension stiffness k_s

The easiest parameter to start with is the stretch spring stiffness constant k_s. The spring force derivative with respect to this parameter is trivial to compute resulting in the force divided by the spring constant. In this simple one dimension case, the gradient we are looking for is not a vector but a scalar, and it is the derivative of the objective function with respect to the stiffness constant k_s. In figure 4.2 we can see the output gradients of a 100 steps simulation, as well as part of the objective function values. This test uses a 20×20 plane mesh which translates to 1200 degrees of freedom, and the target data is recorded from a simulation with a value of $k_s^{\text{target}} = 20$. In figure 4.2, we can appreciate that the cost function and its derivative do indeed reach zero at $k_s = k_s^{\text{target}} = 20$, as we would expect. Moreover, the gradients computed finite differences and the back propagated gradients follow the exact same curve, which is a direct validation of the back propagation mathod. The gradients nearer to zero stiffness do not match exactly, even though they follow the same curve. This discrepancy can be explained by errors in the finite differences method, which is not fine enough to compute correctly the changing slope of the loss function.

The results obtained in this simple situation show that the gradients match exactly. Nevertheless, there are situations where the situation can be less favourable. For instance, we have found that using a different loss function, which only measures differ-

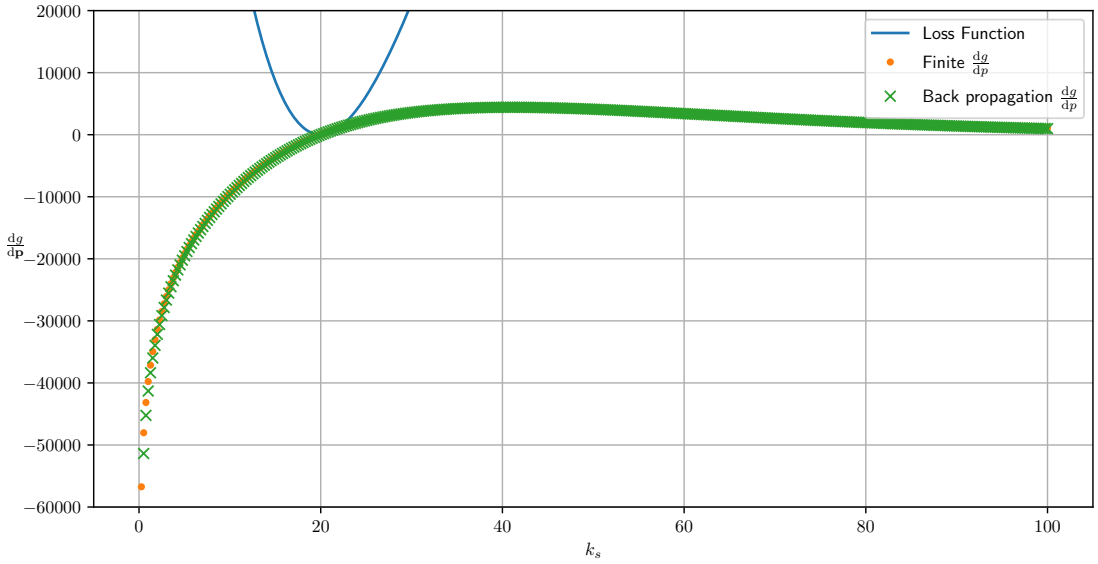


Figure 4.2: Objective function derivative with respect to the stiffness constant k_s computed with finite differences and back propagation. Forward pass solved with 3 Newton iterations, for 100 frames with a time step of $h = 0.1s$.

ences in positions and velocities of one particular node, can have substantial impact on the results, shown in figure 4.3. The first thing to notice, is the noise present in the finite differences gradient, and also the loss function, which starts to be noticeable after a stiffness of $k_s \sim 30$. We have expanded the plot domain to see how the noise evolves when evaluating gradients in high stiffness regimes, away from the target point. This noise is interesting because it is not reduced increasing the number of samples, making it not a discretization problem. It is possible that the noise exists because the system is chaotic, meaning that it is extremely sensitive to initial conditions. A slight perturbation on the tension stiffness could generate a trajectory seemingly identical in the beginning but as time passes the trajectories would start to diverge and become totally different. This would explain why the noise starts to appear with stiffer clothes, which reach complex dynamics sooner because they do not have to fall as far, as more elastic cloths. A double pendulum is an example of a really simple chaotic system. Our mass-spring cloth model can be thought of multiple pendulums coupled together with springs, which it probably makes it a chaotic system. The chaotic nature of the cloth is not noticeable when using the loss function which measures differences between all the nodes in figure 4.2, because the magnitude of the loss is immense in comparison to the chaotic noise.

Interestingly, the back propagated gradients follow a curve without noticeable noise.

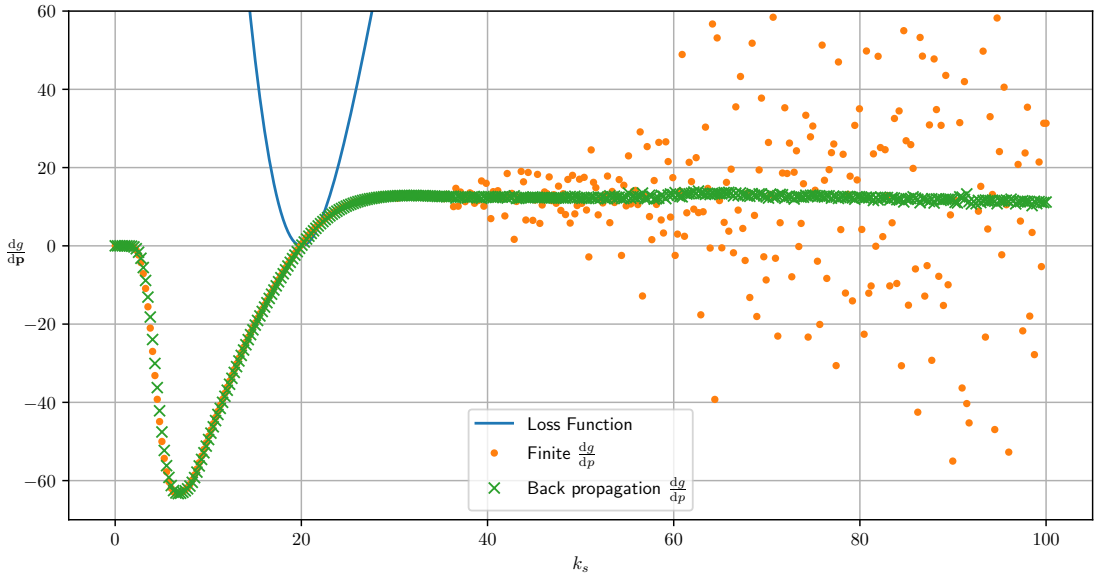


Figure 4.3: Objective function derivative with respect to the stiffness constant k_s computed with finite differences and back propagation. The loss function only measures differences of one of the extremes of the cloth. Forward pass solved with 3 Newton iterations, for 100 frames with a time step of $h = 0.1s$.

Assuming that the loss function noise comes form chaos, it makes sense that back propagated gradients are smoother. Back propagation computes analytic gradients, which are well-defined as long as the loss function is smooth. Chaotic systems suffer a transition from predictability to chaos, which in our case translates to smoother or coarser loss function. In this transition, back propagation is capable of computing gradients because the loss function is still smooth, even though it is extremely detailed, but finite differences is not fine enough to make out the detail. This can be extremely useful for minimization routines, because back propagation would serve as a way to smooth the gradients of the loss function, and partially skip the noise. Nevertheless, quasi-Newton methods could suffer from the coarseness of the loss function itself when estimating higher order derivatives from loss function values.

On the whole, the results show that the gradients computed with back propagation perfectly match the curve of the finite gradients. Hence it seems that the back propagation algorithm is capable of computing one dimensional gradients accurately. Also we have found that, in certain scales, it is possible that chaos starts to be noticeable, making the system unpredictable under slight perturbation of initial conditions, which can hinder or make impossible the minimization process.

Gradient with respect to initial conditions

Continuing with the testing of single parameter gradients, it is interesting to investigate one case involving initial conditions as the simulation parameters. The initial conditions case is interesting as it needs the back propagation machinery to work flawlessly. This is because the initial conditions term is computed using the adjoint variables $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_0}$ and $\frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_0}$, which require all the adjoint variables before to be correct. The only contribution on the cost function gradient in this case will be $\frac{\partial g}{\partial \mathbf{p}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}_0}\frac{\partial \mathbf{x}_0}{\partial \mathbf{p}} + \frac{\mathrm{d}g}{\mathrm{d}\mathbf{v}_0}\frac{\partial \mathbf{v}_0}{\partial \mathbf{p}}$, which means that any deviation in the gradient computation will be caused by a failure in propagating the adjoint variables in the backward loop. The system we are simulating is again a 20×20 flat cloth with 1200 degrees of freedom. To avoid having this many initial conditions as a parameter, we parameterized the positions of the nodes using an angle θ. The angle measures how is the cloth rotated around the x axis, which is where the 2 frozen nodes align as illustrated in figure 4.4. The rotation direction of angle θ is clockwise because we have a right handed system of coordinates, and we want to follow this convention.

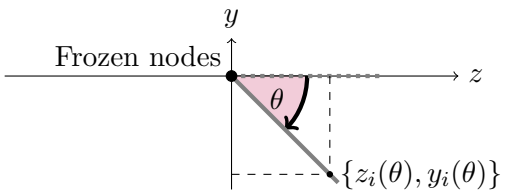


Figure 4.4: Graphical definition of the parameter θ. The gray thick line represents a side view of the flat cloth rotated by angle θ.

Defining the initial conditions using a single parameter has the advantage of not having to deal with a large number of parameters, but it also means that we have to know how to compute the derivatives $\frac{\partial \mathbf{x}_0}{\partial \theta}$ and $\frac{\partial \mathbf{v}_0}{\partial \theta}$. In this case it is trivial to compute this partial derivatives. The initial velocity derivative will be always zero as the angle only affects the positions, and in positions it will only affect the y and z coordinates of each node, as the x value will not change.

$$z(\theta) = L\cos(\theta) \quad \frac{\partial z}{\partial \theta} = -L\sin(\theta) = y \tag{4.2a}$$

$$y(\theta) = -L\sin(\theta) \quad \frac{\partial y}{\partial \theta} = -L\cos(\theta) = -z \tag{4.2b}$$

Note that the derivation here is done using θ defined in radians, and if we want to use degrees, we need an extra Jacobian to change the angle units. With the initial conditions partial derivative computed, there is nothing left to do and we can proceed and test how

back propagation handles initial conditions parameters. We have run the simulation as similarly as possible as the last test with values of $k_s = 20$, $k_b = 0.1$, $h = 0.1s$, and 3 Newton iterations for the forward solver. The study region is defined to cover the full rotation of the cloth, taking advantage of the fact that, unlike the spring stiffness in the previous test, the angle θ is set, and we can cover all its domain.

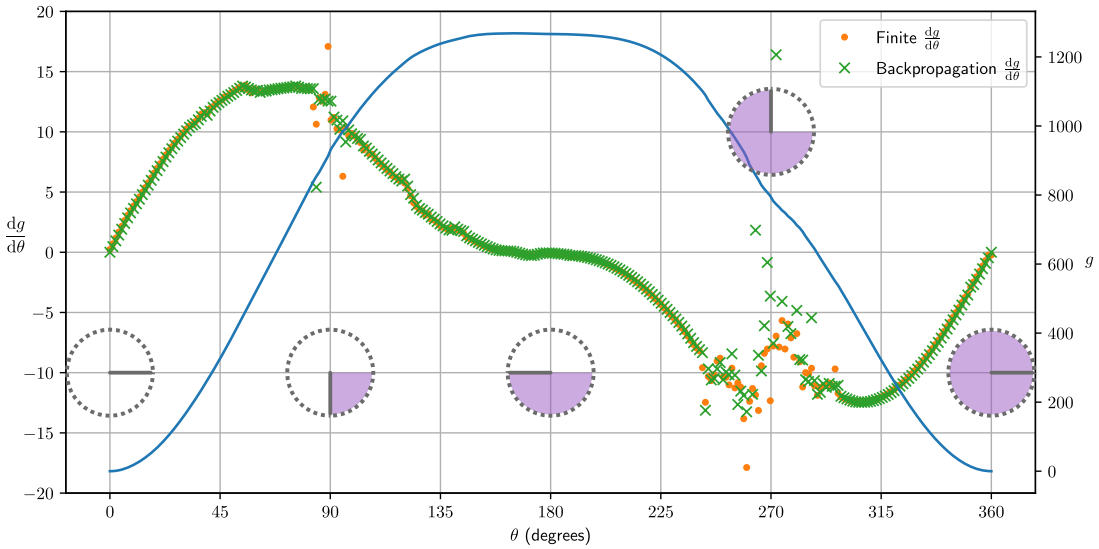


Figure 4.5: Objective function derivative with respect to the initial tilt angle θ of the cloth. The circles represent the starting orientation of the cloth. Forward pass solved with 3 Newton iterations, for 100 frames with a time step of $h = 0.1s$. 300 sample points.

In figure 4.5 we can see the results of the initial condition experiment. Again we can see that the objective function gradient computed with back propagation and with finite differences follow the same curve.

However, there is a region around $\theta = 270°$ and $\theta = 90°$, which seems to be problematic for the back propagated gradients. Interestingly, a rotation of 270° in our convention means that the cloth starts completely vertical, resting on top of the support frozen nodes (see figure 4.4), while $\theta = 90^0$, defines again a vertical cloth, hanging below the support points. Both problematic angles are related to fully vertical cloths. The $\theta = 270°$ case is an especially difficult initial state for our simulator, because predicting how the cloth is going to fall could be not well-defined. Our hypothesis is that the situation is more or less analogous to an inverse pendulum in its unstable equilibrium point, which is stable in theory but any perturbation will make it fall in a different direction. Consequently, a small change in initial angle in this situation can translate into totally different outcomes, which is what we see in our results. It happens something similar in the $\theta = 90°$ case,

where the cloth falls straight down, but tiny perturbations will probably make it bounce forward or backward. In the pendulum analogy, the cloth is in its stable equilibrium point, but because the cloth is elastic, it can fall under the influence of gravity, which will ultimately make the cloth oscillate. The behaviours we are describing around these two problematic points fall again under the definition of a chaotic system, where tiny perturbations have a huge effect on final conditions, and makes the system unpredictable. Making a closer scan of the region we can indeed see that the problem is focused around $\theta = 270°$.

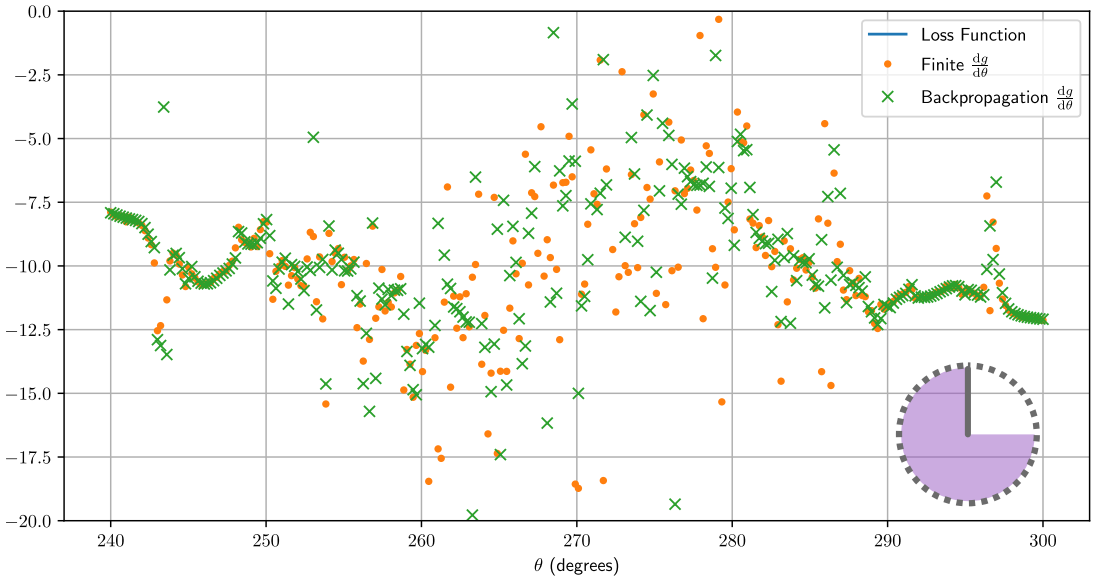


Figure 4.6: Finer scan closer to the problematic vertical angle $\theta = 270°$, using 300 samples.

The problem we have found around the problematic angles illustrates the importance of having a robust forward simulation in order to compute valid simulation gradients. Also the predictability of the system we want to control is crucial because chaos in the end makes the loss function not smooth and the gradients not well-defined. Both back propagation and finite differences predict irregular gradients when the forward simulation fails to make reliable predictions.

4.1.2 Multi-dimensional gradients

The main benefit of back propagation against finite differences is performance. Finite differences must run the forward simulation once in addition of a entire forward sim-

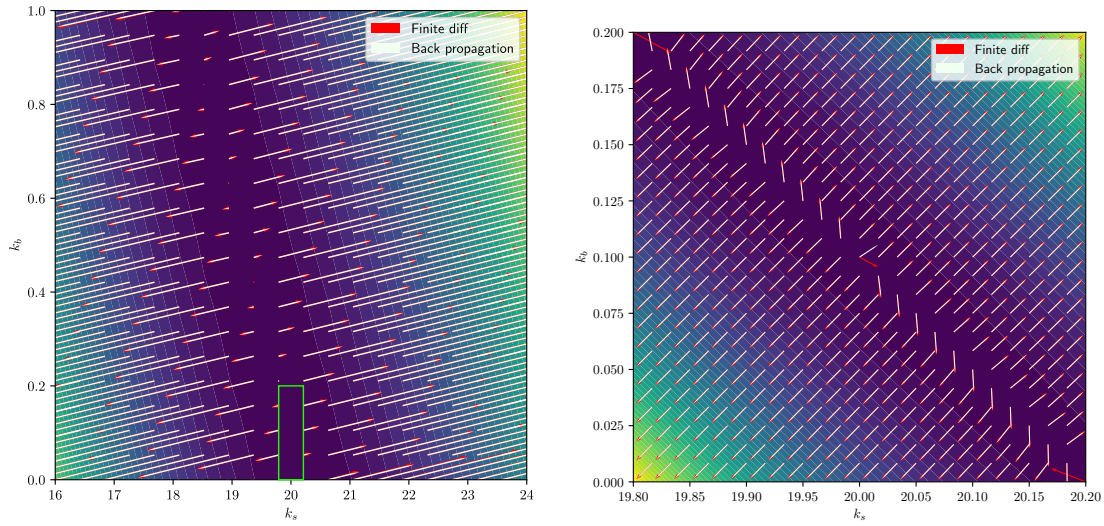
ulation for each parameter, while back propagation only has to run a forward and a backward pass, independently of the number of parameters. Nevertheless, this advantage is not particularly obvious when only computing the gradient with respect to a single parameter. Two forward passes is not much different from a forward and a backward pass. Therefore, it is our interest to compute accurately and performantly higher dimensional gradients using back propagation. In this section we are going to show different simulations, with different number of differential parameters, to study the computed gradients.

Gradient with respect to tension and bending stiffness k_s, k_b

The simplest multi-dimensional case is using 2 parameters. Having only 2 parameters has the advantage that it is still possible to represent the gradients graphically, which will be impossible in higher dimensions. A simple way of having 2 differential parameters is using the tension spring stiffness k_s and the bending spring stiffness k_b as two separated values. As back propagation assumes an arbitrary number of parameters, no change in implementation is needed when changing the dimensions of the parameter vector. Nevertheless, the simulator must handle multiple parameters by computing the derivatives $\frac{\partial \mathbf{c}}{\partial k_s}$ and $\frac{\partial \mathbf{c}}{\partial k_b}$, and placing them correctly in the general matrix $\frac{\partial \mathbf{c}}{\partial \mathbf{p}}$.

The results of the two parameter experiment are shown in figure 4.7, where both the values of the cost function and the gradients are shown. We can appreciate that the computed gradients are indeed perpendicular to the displayed level curves as one would expect. Furthermore, we can see that, overall, finite differences and back propagation predict matching gradient vector fields, which confirms that back propagation can compute correct gradients in two dimensions. Nevertheless, this does not mean that all gradients agree. In figure 4.8b, there are some minor discrepancies. First in the center point $k_s = 20$, and $k_b = 0.1$, which is also the target point, the gradients are not correctly aligned. However, the magnitude of the two gradients is insignificant, as we would expect when near to a minima. There are also two finite differences gradient vectors in the top-left and bottom-right edges, which do not make sense. This is an artifact caused because of the way finite differences are computed, which is not accurate in the edges of the domain.

There exists a challenge when visualizing gradient fields. We are interested in a way to represent the gradient direction and the gradient magnitude. However, plotting the raw vector field in figures 4.7, would not output perpendicular vectors because of the image scaling. Even though the figures are square images, the x and y do not share scale, which



(a) General landscape exploration. The rectangle shows the region of the finer scan in (b). $20 \times 20 = 400$ samples.

(b) Finer scan in the region where the target stiffness values are $(k_s = 20, k_b = 0.1)$. The gradients are normalized. $25 \times 25 = 625$ samples.

Figure 4.7: Two parameter cost function landscape exploration. The value of the cost function is represented as a heatmap with level curves (dark means lower values and bright higher values). The gradients are displayed as arrows vector fields. The back propagation gradient is displayed needle shaped to distinguish it from the finite differences. The simulation uses a timestep of $h = 0.1s$, 100 frames and 3 iterations per Newton step.

squeezes everything in one direction and stretches it in the other. Consequently, the deformation makes the gradient field appear not perpendicular in an unequal axis scale figure. One solution could be to force the figure to have the same scale in both axis, but this would make it too long or to tall for the document. The solution used here is to scale the gradient field in such a way that when the deformation occurs it stays perpendicular. Nevertheless, this method makes the magnitude of the vectors not accurate, which is not in itself a huge problem because the arrows are scaled to improve the readability of the visualization. Hence, the magnitudes only carry information relative to the other vectors, meaning that it is only useful to know if a gradient at a point is bigger that in a different point.

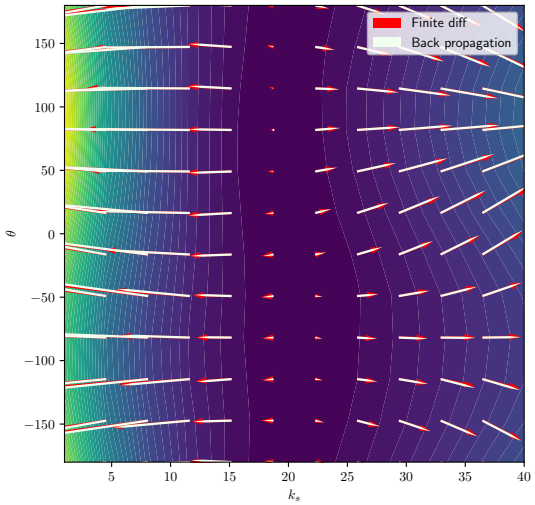
To conclude the study of this two stiffness setting, we can focus on the cost function landscape. The results show that the loss function is steep in one direction, and almost flat in the perpendicular direction. Even though the directions are not completely aligned

with our parameter axis, we can certainly see that the steep direction is more in line with parameter k_s and the flat direction to parameter k_b. The gradients point surely to the dark flat region of the landscape, but once in that region the gradients are small, and their direction changes significantly with respect to their neighbors. Figure 4.7b shows the normalized gradient field to see that few vectors point towards the target point. Moreover, the vectors more aligned in the diagonal of low magnitude loss function, have minuscule magnitudes compared to the vectors pointing away from it. This type of gradient fields can hinder the minimization process and make it unstable. The lower magnitudes mean that the minimization steps will be small, making the process slow, and the significant change in magnitudes and directions can induce instabilities to the process, which can make it halt before convergence. In practice, this might not be an issue. The almost flat valley ultimately means that any result in it is probably indistinguishable from our target. Therefore reaching the valley, which is easy with the obtained vector field, would be enough for our needs.

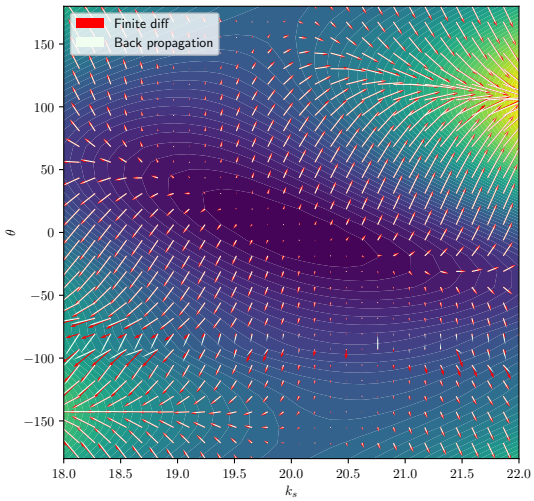
Gradient with respect to tension stiffness and initial conditions k_s, θ

Another interesting two parameter experiment is using the spring stiffness k_s and the tilt angle θ defined in figure 4.4. This setting is interesting because it combines an interaction parameter with the initial conditions. The two differentiable parameters we are using affect the simulation in drastically different ways which can not be said with the last experiment where k_s and k_b both influenced the rigidity of the cloth. In addition, the θ parameter is interesting in itself because it is defined between a finite interval $0 \leq \theta < 2\pi$, and thus we can study the parameter domain fully, and not limit ourselves to a certain region.

The results are shown in figure 4.8, and again the gradients are practically indistinguishable. The coarser scan shown in 4.8a, it shows that the cloth elasticity is significantly more important to reach our target, that the starting angle. This is explained because our loss function only measures the last state of our simulation and the cloth will fall regardless of the starting angle. The elasticity of the cloth controls how much the cloth stretches while falling, and thus controls how far it falls. In the finer elasticity scan done in figure 4.8b, the starting angle starts to become significant. Also in the figure we can see disagreeing vectors close to the starting angle $\theta = -90°$. This has already been observed in the single θ parameter study, where values around $\theta = 270$ were problematic. Here there exists the same problems, only in a wider range of elasticity. In the boarder scan in figure 4.8a, there is no noticeable effects around $\theta = -90°$, because



(a) General cost function landscape exploration. $12 \times 12 = 144$ samples.



(b) Finer scan in the region where the target values are $(k_s = 20, \theta = 0°)$. $30 \times 30 = 900$ samples.

Figure 4.8: Two parameter cost function landscape exploration. The value of the cost function is represented as a heatmap with level curves (dark means lower values and bright higher values). The gradients are displayed as an arrow field. The back propagation gradient is displayed needle shaped to distinguish it from the finite differences. The simulation uses a timestep of $h = 0.1s$, 100 frames and 3 iterations per Newton step.

of the scale and the coarseness of the grid.

Gradient with respect to multiple stiffness values

Until now all the experiments have been done using one or two parameters. To demonstrate that the method works independently of the number of parameters, we are going to significantly increase the number of parameters. In our simulation description, the easiest way to introduce a lot of parameters is to treat each spring independently with different stiffness parameters each. Our cloth is a plane mesh of triangles, which means that the number of tension springs in a $N \times M$ mesh is the number of edges in the mesh:

$$\#\text{Tension Springs} = (M-1)(N-1) + N(M-1) + M(N-1) \tag{4.3a}$$

$$\#\text{Bending Springs} = (M-1)(N-1) + (N-2)(M-1) + (M-2)(N-1) \tag{4.3b}$$

All the tests have used a 20×20, meaning that there are 1121 tension springs and 1045 bending springs which add up to a total of 2166 springs, and consequently parameters, to experiment with. In this higher dimensions study, we can no longer rely on visualizing a region of the domain to compare the results from both methods. Alternatively, we can use some numerical metric that can compare the results and return us some value which tells us how close the results are. This method has the advantage of being well-defined and predictable, but on the other hand it will not give us an intuition of how do the gradients differ from one another. It is still possible to compare visually the resulting gradient component-wise making a plot of the value of each component, shown in figure 4.9. Moreover, because of the large number of parameters, studying a region of the domain like we have done before, is out of computational reach $\mathcal{O}(n^{2166})$. Consequently, we will study single point instead of a region.

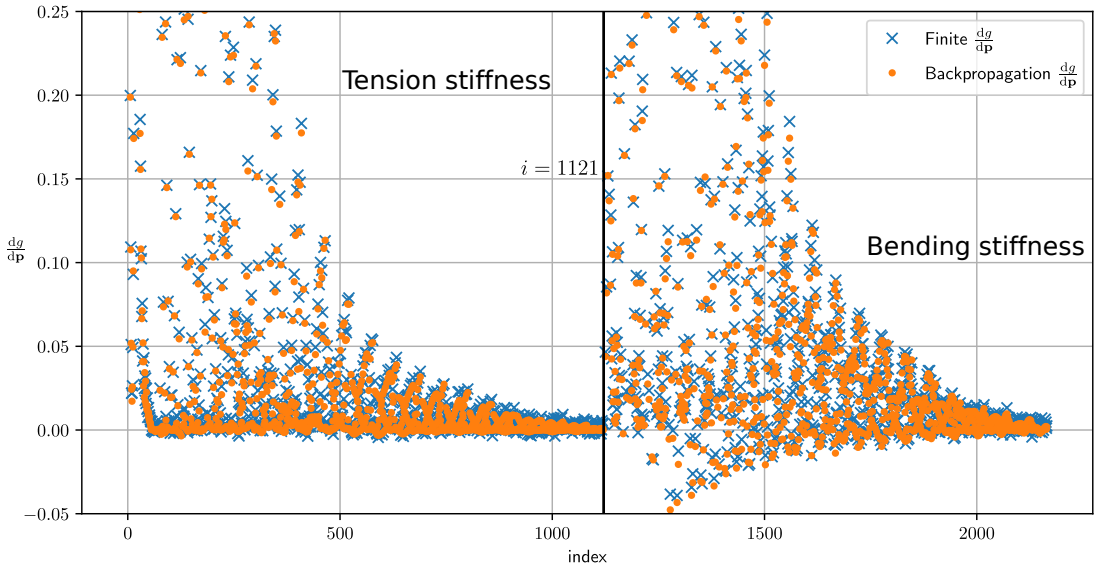


Figure 4.9: Graphic representation of a high dimensional gradient vector components, computed with back propagation and finite differences. The x axis represents the index of the gradient component, while the y axis shows the value of the component. The black line visually separates the tension stiffness (left) from bending stiffness (right). The gradient is evaluated at $\{k_s\} = 40$ and $\{k_b\} = 0.1$, while the target is set at $\{k_s\} = 70$ and $\{k_b\} = 0.1$. The simulation is done with 50 frames, and 2 Newton iterations per step.

The results shown in figure 4.9, display how, even in a 2166 dimensional parameter space, back propagation is capable of computing correct gradients. There exists a certain error between components of the gradient vector, but overall, the results match.

Interestingly, the components of the gradient with respect to bending stiffness, have a higher absolute values than the gradients with respect to tension springs. Moreover, the bending springs in the evaluated point and the target point are the same $k_b = 0.1$, so in practice one would expect a zero gradient. Plainly, the gradient with respect to the bending stiffness is different than zero, and this is explained because gradients are local, and they do not point away from absolute minima, but instead point towards maximum local increase of the function. In this particular point, moving away from the target surface $\{k_b\} = 0.1$, reduces our cost function.

In high number of parameter situations like the one at hand, is where having a differentiable simulation becomes most useful. Each point in 4.9, represents a full simulation that had to be done to obtain the finite difference derivative. Back propagation is capable of computing the full gradient in a single backward pass which has a similar cost to the forward simulation case. In conclusion in this situation it is capable of reducing the gradient computation time at least $\sim 1000\times$. The improvement increases with the number of Newton iterations in the forward simulation, because the performance of the backward pass is independent from them.

Bear in mind that the results here only show one particular point, and it is difficult to get the full picture of the whole domain when we are working with high parameter dimensionality. In later sections will study the minimization process using high dimension parameter spaces, which will also be a way to verify the predicted gradients in multiple points of the domain.

4.2 Introducing contact

Contact interactions are a challenge in differentiable simulation. The discontinuous nature of collisions goes against the smoothness required to compute well-defined derivatives. Computing useful gradients in this situation is an open problem in literature and has been tested by different works [10].

in this section we will introduce contact in our simulations to study how it affects the cost function landscape and the computation of gradients. The experiments are done again using a mass spring cloth simulation, with colliders in the scene. Contact has been implemented in the simplest way possible, being node-wise and it does not handle fancy edge or face collisions. The cloth can only collide with the colliders and not with itself, which makes cloth-cloth penetration possible. The collision response, as we have seen in section 2.2, is a simple spring force with a certain contact stiffness k_c, which pushes the node out of the collider with a force proportional to the penetration.

The contact force in itself is not discontinuous but the force position Jacobian is. Even though the contact model is simple, its discontinuous nature will be enough to pose a challenge to our derivative computations. In the bottom of figure 4.1, there is the visual representation of the physics scene at hand, where a hanging cloth collides with a sphere collider.

4.2.1 One dimensional gradients

In the remainder of the section, we repeated the same experiments of section 4.1, but with the collider, starting with the simplest one dimensional case, computing the gradient with respect to the tension spring stiffness. The experiments here have been done using a higher stiffness $k_s = 70$, to ensure that the cloth will interact with the sphere colliders in all situations.

Gradient with respect to tension stiffness k_s

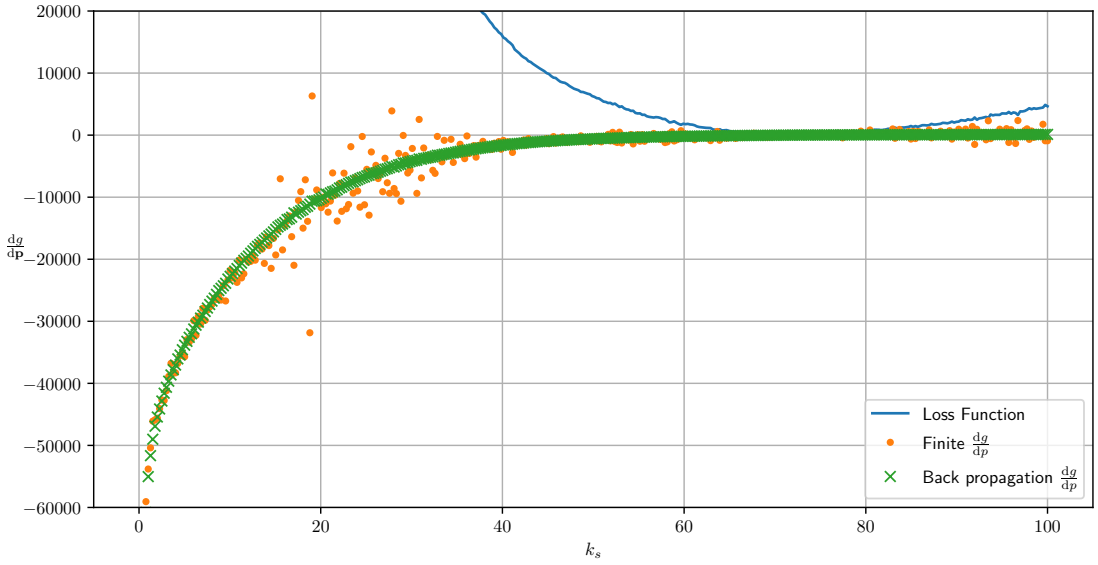


Figure 4.10: Gradient with respect to tension stiffness with contact. The target is a simulation with a stiffness of $k_s = 70$. Simulation done with a 20×20 node mesh, 3 Newton steps, 100 frames and a time step of $h = 0.1s$.

The first thing to notice in the results shown in figure 4.10, is the noise present in the finite differences gradient, which did not exist in the no contact experiment shown in figure 4.2. From this, we know that the noise comes from the introduction of contact,

but not yet what exactly caused it. The back propagation gradients are still able to follow a smooth curve even with the introduction of contact. An interesting feature of this experiment is that, because of how the cloth is set above the collider, there is almost no contact in lower stiffness and a lot of contact in higher stiffness. An elastic cloth is able to stretch a lot, and it is able to hang below the sphere collider, experiencing contact only in a few frames, while a stiffer cloth will not stretch as much and collide every time in a lot of frames. The transition between the described no-contact and contact regimes happens roughly around $k_s \sim 20$, which coincidentally is where the main noise is located.

In conclusion, the introduction of contact in this situation has not notably affected the computation of gradients using back propagation, which is still computing smooth and valid gradients.

Gradient with respect to initial conditions

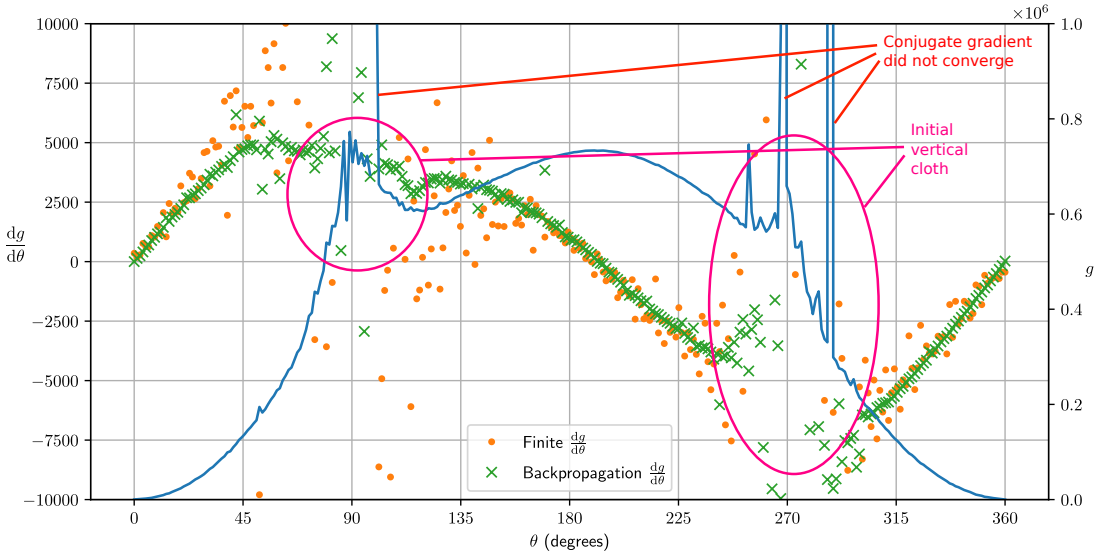


Figure 4.11: Gradient with respect to initial cloth inclination angle θ, with contact. There are 300 samples, done with 5 Newton iterations, 100 frames, and a time-step of $h = 0.1s$.

Continuing with one dimensional gradients, we will test the gradients computed by changing the starting cloth inclination angle θ, this time with the sphere collider present.

The results shown in figure 4.11, are computed using different loss weights than before. Particularly, we are now controlling the whole cloth trajectory, and not only the cloth final state $\{w_i\} = 1$. It is easy to notice that the loss function definition has

been altered, because its order of magnitude has increased significantly from $\sim 10^3$ in figure 4.5, to $\sim 10^6$ in this case. We have made this decision because using multiple frames, we are able to obtain clearer data, easier to analyse. Nevertheless, we tested a minimization process done controlling only the last simulation step, to check wether the computed gradients are useful in this particular situation.

The loss function presents discontinuities around the angles $\theta = 90°$ and $\theta = 270^0$, where the cloth is completely vertical. These discontinuities make the finite differences gradient inaccurate, and it presents noise throughout the domain, even though is more concentrated in the vertical angles regions. The back propagated gradients are overall smoother, but are discontinuous near the vertical angles. In the smoother regions of the loss function, away from the vertical angles, back propagation and finite differences make similar gradient predictions. Comparing the results to the non-contact case in figure 4.5, we can see that the introduction of contact has also introduced huge discontinuities in our loss function and has overall increased the noise. In the last experiment with contact, introducing contact has also increased the noise of the finite differences gradient significantly, which appears to be a common theme.

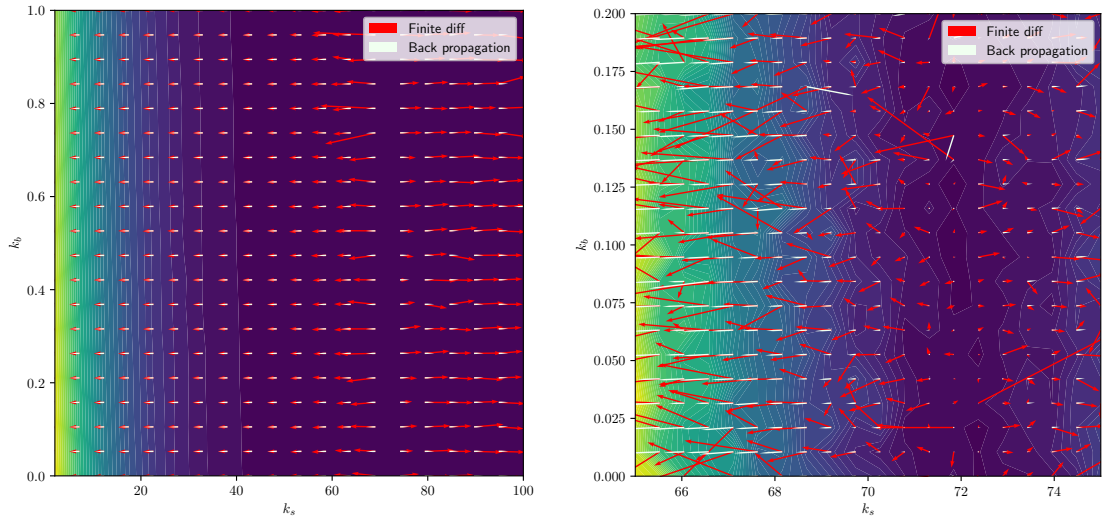
The noise in finite differences gradients comes from tiny discontinuities in the loss function. These discontinuities are not directly related to the discontinuities of contact force Jacobians. In fact, contact discontinuities should affect the back propagated gradients, which are computed making use of contact Jacobians. Nevertheless, the results show scattered finite differences and smoother gradients from back propagation. The tiny discontinuities in the loss function can be explained again by assuming that our cloth plus sphere-collider system is extremely sensitive to initial conditions. It is possible that the introduction of contact has made the system more chaotic than the single hanging cloth. The higher spikes in the loss function, are not originated solely by sensitivity to initial conditions, but by the forward simulation not being able to converge to a good solution. Our simulator uses the conjugate gradient method [3] for solving the systems of linear equations, and it only works when the system is defined positive and it is symmetric. When these conditions are not met, convergence is not guaranteed and the simulator computes wrong solutions.

The problematic zones when the cloth is vertical already appeared in the no-contact case, and can be explained again by the system not being well-defined in some cases. However, in the contact case the problematic zones have become more problematic. When the cloth falls vertically upon the sphere collider, it seems that the solver is not able to determine the direction the nodes must be pushed away from the sphere. With all of this complexity, it is difficult to say wether the errors in back propagated gradients

come from contact discontinuities, from the loss function not being smooth because of chaos, or because the forward simulation could not converge to a good solution.

4.2.2 Multi-dimensional gradients

Gradient with respect to tension and bending stiffness k_s, k_b



(a) General landscape exploration. Normalized back propagation gradients and scaled finite differences gradients.

(b) Finer scan closer to the target point $k_s = 70$, $k_b = 0.1$.

Figure 4.12: Gradient with respect to both tension and bending springs stiffness with contact. In both figures there are $20 \times 20 = 400$ samples, done with 3 Newton iterations and a time-step of $h = 0.1$.

We start again with the simplest two dimensions gradient case, differentiating with respect to both tension and bending stiffness. The results shown in figure 4.12, have been obtained controlling only the last state of the simulation, meaning the loss function weights again are set to $w_{i<N_s} = 0$ and $w_{N_s} = 1$.

Looking first at the general scan shown in figure 4.12a, the results have not changed much with respect to the no-contact case shown in figure 4.7a. Because of the different domain scanned and the change of target stiffness from $k_s = 20$ to $k_s = 70$, now it seems that the bending stiffness parameter does not play a significant role anymore. Moreover, the loss function starts to flatten at a tension stiffness around $k_s \sim 40$, and there is a large region with no level curves. This indicates that any simulation in that region will output similar results. It is possible to get an intuition of why this happens

by imagining how far will a mass on a spring fall as a function of the spring stiffness $\Delta x \sim \frac{gm}{k}$. Figure 4.12a follows roughly $\frac{1}{k_s}$, which justifies the flatter region at higher stiffness. The computed back propagated gradients are shown normalized, and the finite difference gradients are scaled relative to the same normalization. With this it is possible to see that the gradients match perfectly at lower stiffness but start to differ when passing $k_s \sim 60$. The direction of both gradients seem to be compatible, but the magnitude can be $\sim 3\times$ larger in finite differences. It is important to bear in mind that the disagreeing gradients are also the ones with lower magnitudes, as they rest on the flatter surface of the loss function, which makes the absolute difference between both methods small.

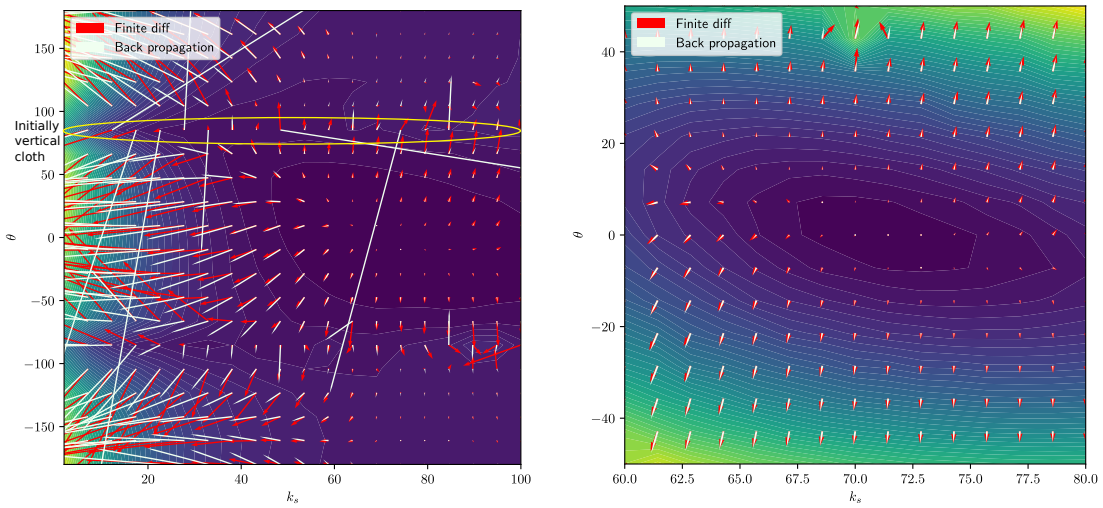
Figure 4.12b, shows a finer scan around our target point $k_s = 70$ and $k_b = 0.1$, even though the lower values of the loss function do not fall inside our target point. This is because the reference simulation is done using 5 Newton iterations instead of the 3 iterations that the simulations in the scan use. Ultimately, this means that 3 Newton iterations is not enough to reach convergence, because the results vary with the number of iterations. The loss function landscape in the figure is complex, but it can be coarsely described by a valley centered at $k_s = 72$, with high curvature in the k_s axis and low curvature in the k_b axis. Because of the high detail of the loss function, the finite difference gradients experiment a significant amount of noise. The back propagation vectors, are mostly aligned with the k_s axis, and do not change significantly between neighbors. There are two particular outliers back propagated gradients with seemingly wrong magnitude and direction. Again it is not easy to justify what caused this error, but its most probably originated by the forward simulation not converging in a difficult contact situation. There are obvious discrepancies between the two methods but the overall direction is correct. In fact, the situation reminds of the noise in the one parameter figure 4.10, where the finite differences were scattered around the back propagated values.

Gradient with respect to tension stiffness and initial conditions k_s, θ

To end this section, we will repeat the tests varying both the initial cloth inclination angle θ and the cloth tension stiffness k_s, now with the sphere collider. The results obtained here use a set of loss function weights of $w_i = 1$ to control all the cloth trajectory and not only its final state. We have done the same in the experiments with the single parameter θ, and all the same reasoning applies here.

The results shown in figure 4.13a, reveal a relative smooth landscape, where the starting angle matters at lower stiffness and becomes less relevant as the stiffness increases.

Remembering the experiment with contact and varying only the tension stiffness 4.10, in lower stiffness the cloth was elastic enough to hang below the sphere and avoid the collision. However, if the cloth starts vertically, it will collide regardless of the cloth stiffness. The target stiffness $k_s = 70$, is above the hang below the sphere stiffness threshold, and also will always collide. At lower stiffness, the vertical cloth starting angles $\theta = 90°$ and $\theta = 270°$ have a lower loss function because they force contact and consequently, the resulting trajectory is similar to our target. At higher stiffness, the cloth will always collide regardless of the starting angle and the elasticity, which explains the flatter region of the landscape.



(a) General loss function landscape.

(b) Finer scan closer to the target point at $k_s = 70$ and $\theta = 0$.

Figure 4.13: Gradient with respect to tension springs stiffness and initial cloth inclination angle with contact. Both plots use $20 \times 20 = 400$ samples, each using 3 Newton iterations, and a time-step of $h = 0.1s$.

Both gradient fields are consistent with one another, even though that there are obvious discrepancies. In this scenario it is possible that the differences have been caused by finite differences errors, meaning that our finite differences step is not fine enough to make out the highly non linear loss function. The discrepancies are more noticeable in the complex lower stiffness region than in the flatter more linear region of higher stiffness. In a similar way of the no-contact case, the vertical angles produce seemingly incorrect back propagated gradients.

In the finer scan shown in figure 4.13b, the gradients match almost perfectly, and the loss function is smooth and well-defined. There is an anomalous point in the top of the

domain, centered at $k_s = 70$. The neighboring finite difference gradients seem to notice it but not the back propagated ones, which remain unaltered. In conclusion, the study region in well behaved, and it is possible to obtain a smooth function with well-defined gradients. Making an optimization in the region should easily converge to an acceptable solution in few iterations.

4.3 Objective function minimization

With the previous tests we have seen that the back propagation algorithm is capable of computing accurate gradients, similar to the results of finite differences. However, the computation of these gradients is only a means to get to our goal, being able to find the best parameters to minimize a certain objective function. In this section, we study the minimization process in different scenarios and minimization schemes.

We have reduced our study to two single minimization schemes: a simple gradient descent and the quasi-Newton method L-BFGS [1], from the scipy library [8]. Both methods are iterative and aim to converge evaluating the cost function and the gradient repeatedly. Gradient descent is widely used in learning applications and it computes the next point moving some finite distance against the gradient each iteration. On the other hand, L-BFGS is a quasi-Newton method, which means that it approximates higher derivatives, in particular the hessian matrix, from the gradient and function value of previous iterations. This way the algorithm is capable of estimating the cost function curvature and theoretically converge faster. Both algorithms use line-search, which means that each new iteration the cost function is compared to the last iteration value, assuring that the value does not increase. This is useful for avoiding that the methods diverge, but it may make the methods more likely to converge to local minima.

The first test is the simplest one, where there is no contact present and we are optimizing only the tension stiffness k_s, to obtain the last state of a simulation done with $k_s = 20$. The optimization process does not present any problem in any method. In the L-BFGS case, it is able to reduce the cost function 6 orders of magnitude in the first iteration, and 3 orders of magnitude in the second iteration, where it converges to the exact solution $k_s = 20$. Gradient descent is able to rapidly and smoothly converge, even though without line search and with big learning rates, the method got to negative stiffness which the simulation can not handle.

The introduction of contact in this situation, hinders the convergence process but it is still able to find a good solution. L-BFGS is capable of reaching convergence in 5 iterations, and reduce the lost function 4 orders of magnitude in total. The optimized

parameter we get is $k_s = 72.9$. This means that the introduction of contact not only needs more iterations, but also decreases the accuracy of the method. Gradient descent is able to reach a value of $k_s = 71.8$ in 100 iterations, but it needs 77 iterations to match the L-BFGS solution. In conclusion, the minimization with respect to tension stiffness is successful with and without contact.

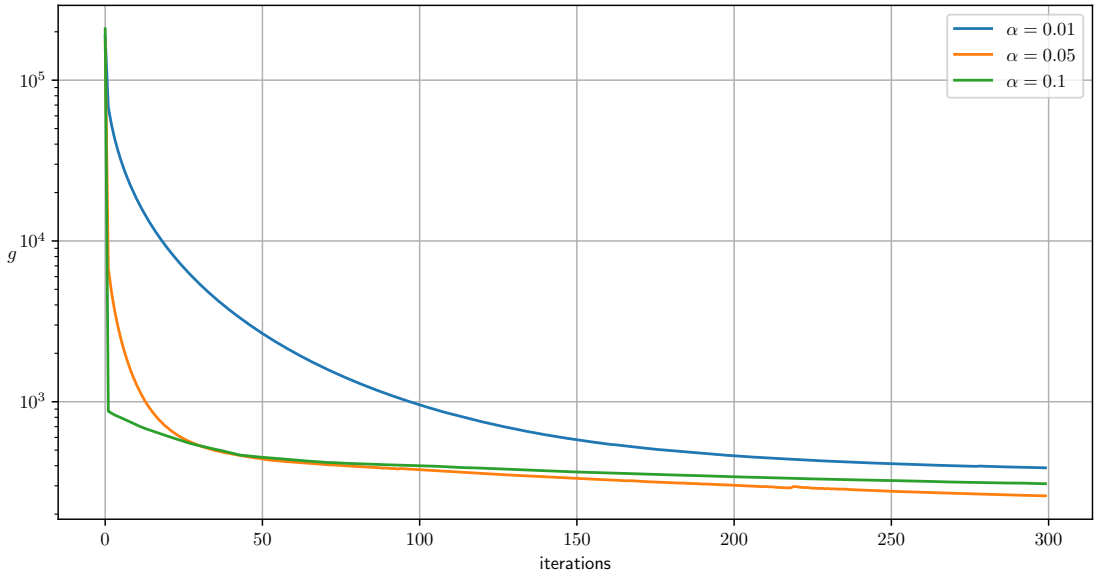
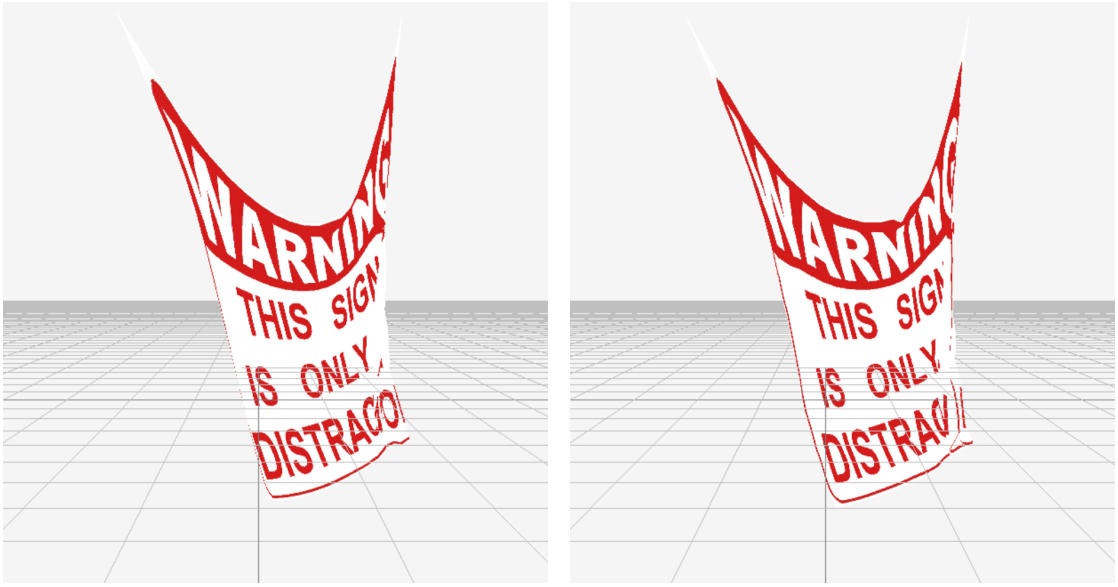


Figure 4.14: Convergence of the gradient descent method with different learning rates α. Here we optimize 2166 parameters with no contact. Simulations done with 3 Newton iterations and a time-step of $h = 0.1s$.

In section 4.1 and 4.2, we have stated the problems of studying a higher dimensional gradient field with finite differences because of the computational cost. We also stated that studying the minimization process will be a good metric for evaluating if the computed gradients are correct. For this reason, it is particularly interesting to do minimization tests with our 2166 dimensions parameter space. The first 1121 parameters describe each particular tension stiffness in the cloth, while the remaining 1045 parameters are the bending stiffness values. The target here is the last frame of a simulation done with $k_s = 70$ and $k_b = 0.1$. The minimization process will start with random stiffness values between 10 and 30 for the tension springs, and a fixed stiffness of $k_b = 0.1$ for the bending springs. In the no-contact case, the L-BFGS method is able to reduce the cost function by 4 orders of magnitude with 100 iterations. In the final iteration, the tension stiffness average to a value of ~ 20 and has a variance of ~ 50. The bending stiffness averages to ~ 13 with a variance of ~ 111. The values are not close to the

target $k_s = 70$, $k_b = 0.1$, but the loss function has been reduced regardless, and the ground truth and the result of the optimization process, displayed in figure 4.15, look the same, with a margin of error. This is possible because bending and tension springs



(a) Ground truth.

(b) Result from the optimization process.

Figure 4.15: Graphical comparison of the target final state (ground truth) and the last state obtained from the optimization process. The control variables of the optimization, are the multiple tension and bending stiffness.

are ultimately the same springs with different stiffness values. Here we are controlling each particular spring, so it is possible that some tension springs get lower stiffness and act like bending springs. The same can happen to bending springs which get higher stiffness and behave like tension springs. In conclusion, it may be that tension and bending springs have mixed, but the overall behaviour of the cloth remains similar to our target simulation. Looking at the results shown in figure 4.14, gradient descent is capable of reducing the loss function by 3 orders of magnitude in the first 100 iterations even in the more conservative learning rate of $\alpha = 0.01$. Each iteration in gradient descent always decreased the objective function without line-search enabled, which means that the back propagated gradients give a good minimization direction, and therefore backpropagation is probably computing correct gradients. There is an exception in the curve of learning rate $\alpha = 0.05$, where there is a small bump between 200 and 250 iterations, where the loss function increased. The results show that the best learning rate for this experiment is $\alpha = 0.05$, which was able to reach the lowest loss function value in 300 iterations.

When doing the same tests with the sphere collider, the L-BFGS method is only able to reduce the loss function by 2 orders of magnitude. It is possible to get better results by controlling an earlier state of the simulation, for instance the frame 75 instead of the 100. This way we get a 3 orders of magnitude reduction, getting a loss function value of $g \sim 40$, which is lower than the best optimization done by gradient descent in the no-contact case. The tests done in gradient descent, the method was not able to converge, and it got stuck in the line search, even with the easier case of controlling the 75 frame instead of the 100. In the end, the obtained results show how using the back propagation method, it is possible to compute valid and useful gradients and control a 2166 parameter simulation with contact present. With finite differences, this optimization would have been approximately three orders of magnitude slower, taking days instead of a few minutes.

The final convergence test is done by varying the initial cloth inclination angle θ. This case is interesting because we have already seen in figures 4.5 and 4.11 that the predicted gradient has anomalous behaviour around the vertical angles. The goal of this final experiment is to know weather the optimization is possible even with the anomalous gradients. Starting with the no-contact case, L-BFGS is able to converge without any problems reducing the loss function by 3 orders of magnitude in 5 iterations. The predicted starting angle obtained from this method is $\theta \sim 2.6°$, which is really close to our target $\theta = 0$. When introducing contact, L-BFGS is not capable to converge to any result. This is an interesting result, because even though we already knew that the gradients in this situation were noise for the most part, we now confirmed that the gradients are indeed not useful for the optimization process. However, in the same way as in the angle validation, we can change our loss function description to control the whole trajectory of our cloth and not only the last frame, by changing the weights to $w_i = 1$. With this change, L-BFGS is capable of reducing the loss function by 3 orders of magnitude, to a starting angle of $\theta \sim -0.9°$, which is even closer than in the no-contact case. The extra information from the whole cloth trajectory is capable to smooth the loss function enough to get valid gradients and converge to a perfectly acceptable solution. Gradient descent was not able to converge under any situation with contact, and always got stuck in the line search.

4.4 Demonstration of the method

To close the results chapter, we are going to use the method to control our mass-spring cloth in a less artificial environment. These tests will serve more as a demonstration of certain situations where the method can be used and less as experiments to validate the gradients and study the convergence process.

In this first demonstration, we want to throw our cloth in such a way that it lands on the sphere collider in a specific frame. In other words, in the setting of a minimization process, we want to find the initial velocities of each node of the cloth which minimizes the distance from the simulation's final state to our target. The target state is displayed in figure 4.17a, and in this case has been obtained by simulating a cloth falling on top of the sphere collider. In the end, the target is freely designed by the user of the differentiable simulation, and could be created manually by an artist or by any other means. After running the optimization process, our differentiable simulation is able to converge in few iterations and find a set of initial velocities which aproximate our desired target. Figure 4.16, shows the resulting trajectory of the cloth computed by the optimization

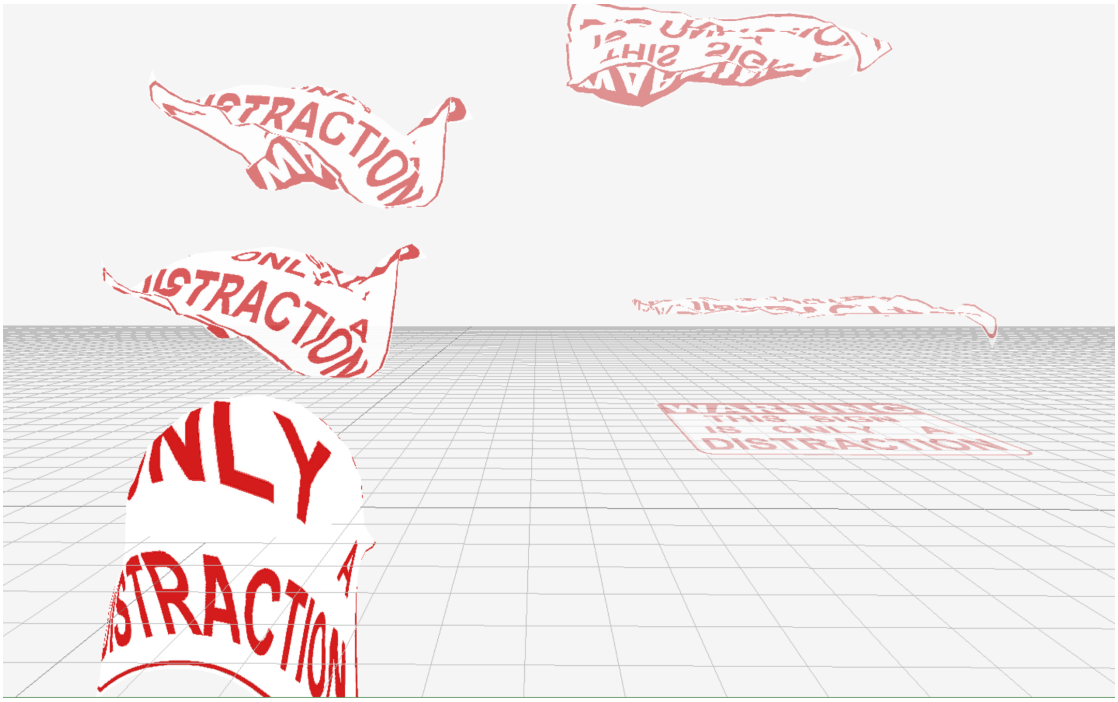
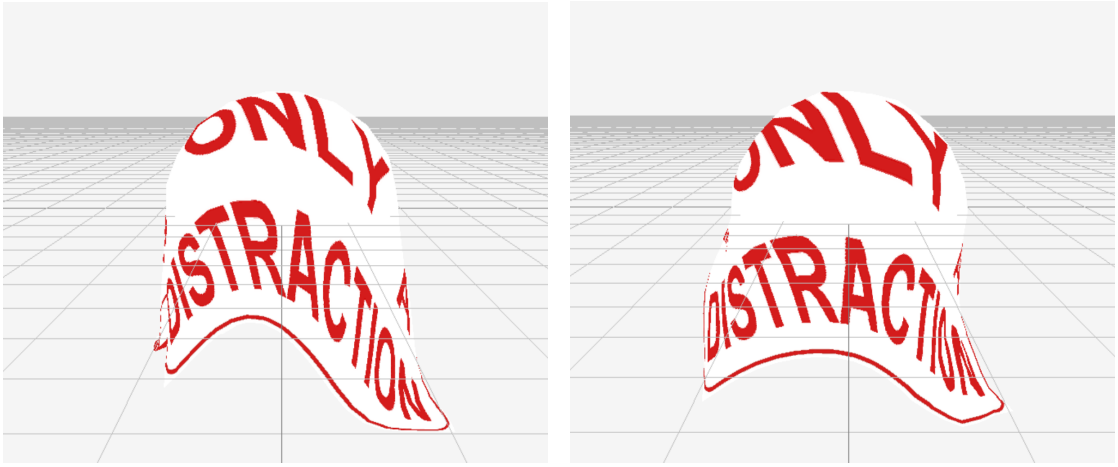


Figure 4.16: Trajectory of the cloth aiming the sphere collider, obtained by controlling the initial velocities of the cloth. The target state of the optimization can be seen in figure 4.17a.

process. This scene would have been very difficult to make without the tool of differentiable simulation. There would not be a straight forward way of finding the correct set of velocities, and the process would have been slow and less accurate. Computing the gradients with finite differences would have slowed our convergence significantly as we are working with a 20×20 node cloth, which adds up to 1200 initial velocities to optimize. However, with our differentiable simulation, we can define our target, and the optimization will compute the desired set of parameters. In figure 4.17, we can see that the optimization process not only was able to hit the sphere collider, but to match the shape the cloth takes resting upon the collider closely to our target state. This level of detail we can obtain is only possible because we are working with a large number of parameters, and thus the simulation can have a large space of possible outcomes. If for example we only controlled a 3 component global velocity of the cloth, finite differences could be fast enough for our needs, but the level of control will drop significantly, as we have less parameters to tweak and thus a smaller set of outcomes.



(a) Ground truth.

(b) Result from the optimization process.

Figure 4.17: Graphical comparison of the target final state (ground truth) and the last state obtained from the optimization process.

In another example, displayed in figure 4.18, we want to throw the cloth in such a way that it matches the frame in a certain frame. In this case we control not only the initial conditions but the general tension and bending stiffness of the cloth. The optimization process again is able to rapidly converge to a solution, and we obtain a scene which would be extremely tedious to do by hand.

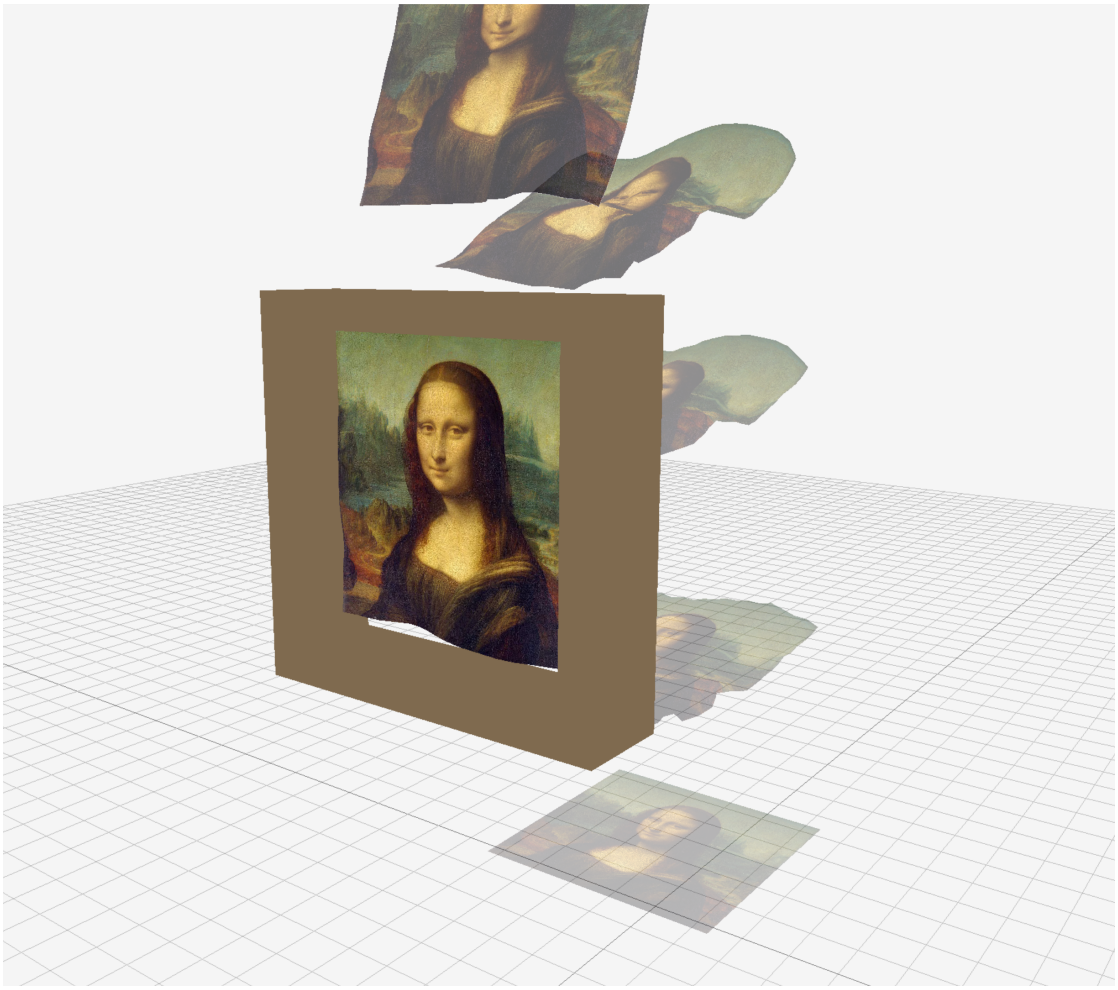


Figure 4.18: Trajectory of the cloth being thrown to fit the frame, obtained by controlling the initial velocities and stifness of the cloth.

Chapter 5

Conclusions and future work

This work can be virtually separated into a theoretical introduction to the mathematics involved in the back propagation derivation, and an applied validation of the method and its implementation.

We have derived the back propagation algorithm in a novel way, not yet seen in the state of the art. This has given us a deep understanding of the method and the formal details involved in the derivation. With this knowledge, we were able to compare and find a connection between our method and the adjoint method for particle systems presented by Wojtan *et. al.* in [9]. Moreover, our novel derivation can give new insights to the physical meaning of the adjoint variable, which was viewed only as a useful mathematical tool. In a similar way as Miles Macklin in the differentiable simulation Siggraph curse [2], we relate the adjoint variable with a derivative of the loss function with respect the state $\tilde{\mathbf{s}}_i = \left(\frac{\mathrm{d}g}{\mathrm{d}\mathbf{s}_i}\right)^T$, only that ours is a total derivative. The loss function gradients with respect to the state could be useful quantities in differentiable simulation, and with this connection, we know how to compute them for free from the adjoint variables.

We successfully implemented a differentiable simulator of a mass-spring cloth, which is seconded by the number of tests done in the results. In the no contact interaction scenarios, our implementation of back propagation, and consequently the method we mathematically derived from scratch, is able to predict correct loss function gradients, independently of the dimensions of the parameter space. From the validation tests done in section 4.1, we also observed the effects that sensitiveness to initial conditions can have in the contnext of simulation control. The unpredictability of a system translates to a not smooth loss function and consequently not well defined gradients. To circumvent this issue, we have found that controlling an earlier time, where the system has not yet diverged into its chaotic regime, helps significantly to smooth the loss function.

In addition, our results show that controlling more degrees of freedom of the system, increases the loss function scale, and the chaotic noise becomes negligible (see figures 4.2 and 4.3).

We have extensively tested the computation of gradients with the presence of contact interactions and observed its impact. Contact interactions did pose a problem to the computation of gradients but not necessarily for the reasons we expected. Our results do not clearly show that the errors in the back propagated gradients originate from the discontinuities of the contact force Jacobian as we expected. Instead, we have seen that contact may catalyze the transition to the unpredictability regime, and also it can pose a challenge to our system of linear equations solvers. It is possible that the situations we have studied are biased and further studying the effects of contact as well as how to improve the gradients when contact interactions are present is considered future work.

Finally, we successfully controlled our simulation in different situations, by completing multiple minimization processes. The results of the optimizations without contact, show how having a differentiable simulation can make the process orders of magnitude faster. As the back propagation algorithm grows linearly in complexity with the number of control variables, it remains fast even with ~ 2000 parameters. The introduction of contact does hinder the optimization, but the method is still able to converge to valid solutions in reasonable iterations. In these more complex scenarios, we found that simpler minimization algorithms like gradient descent, are not generally capable of reaching a good solution. The quasi-Newton method L-BFGS, did not get stuck and was able to converge.

Bibliography

[1] *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[2] Stelian Coros, Miles Macklin, Bernhard Thomaszewski, and Nils Thürey. Differentiable Simulation. In *SIGGRAPH Asia 2021 Courses*, SA '21, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Tokyo, Japan.

[3] Gaël Guennebaud, Benoît Jacob, and others. Eigen v3, 2010.

[4] Eric Heiden, Miles Macklin, Yashraj Narang, Dieter Fox, Animesh Garg, and Fabio Ramos. DiSECt: A Differentiable Simulation Engine for Autonomous Robotic Cutting, May 2021. arXiv:2105.12244 [cs].

[5] Junbang Liang and Ming C. Lin. Differentiable Physics Simulation. February 2020.

[6] Björn List, Li-Wei Chen, and Nils Thuerey. Learned Turbulence Modelling with Differentiable Fluid Solvers: Physics-based Loss-functions and Optimisation Horizons. *Journal of Fluid Mechanics*, 949:A25, October 2022. arXiv:2202.06988 [physics].

[7] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics*, 23(3):449–456, August 2004.

[8] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. Van Der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul Van Mulbregt,

SciPy 1.0 Contributors, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andreas Kloeckner, Anthony Scopatz, Antony Lee, Ariel Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric Martin, Eric Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, G. Young, Gavin A. Price, Gert-Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jörg P. Dietrich, Jacob Silterra, James T Webber, Janko Slavič, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius De Miranda Cardoso, Joscha Reimer, Joseph Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Tritz, Martin Thoma, Matthew Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per A. Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Takuya Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis R. Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, Utkarsh Upadhyay, Yaroslav O. Halchenko, and Yoshiki Vázquez-Baeza. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, March 2020.

[9] Chris Wojtan, Peter J. Mucha, and Greg Turk. Keyframe control of complex particle systems using the adjoint method. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 15–23, 2006.

[10] Yaofeng Desmond Zhong, Jiequn Han, and Georgia Olympia Brikis. Differentiable Physics Simulations with Contacts: Do They Have Correct Gradients w.r.t. Position, Velocity and Control?, July 2022. arXiv:2207.05060 [cs].