Development and benchmarking of a Particle System framework for structural modeling of soft-wing kites

A.R. Batchelor

Delft University of Technology

TU Delft

Development and benchmarking of a Particle System framework for structural modeling of soft-wing kites

by

A.R. Batchelor

To obtain the degree of Master of Science in Sustainable Energy Technology at Delft University of Technology

Supervisor:	Dr. Ir. R. Schmehl	
Daily supervisor:	Ir. J. Poland	
Thesis committee:	Dr. Ir. R. Schmehl	TU Delft, chair
	Dr. Ir. U. Fechner	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

Cover: Painting in the style of Kazimir Malevitsj, by stable diffusion AI art, with prompt: ”airborne kite, sea, setting sun, Kazemir Malevitsj, suprematism composition, blue purple”

TU Delft

Preface

Choosing the direction for my academic career was never solely based on passion, as some people might feel. I've always been able to cultivate interest in almost any subject I encounter, so finding my vocation had to be a rational decision. Although that may appear paradoxical, I do feel a strong sense of responsibility considering the privilege of my circumstances. I've consistently enjoyed the support and patience of my loving parents and girlfriend and was given the opportunity to develop and apply my intellectual abilities, which is not something to take for granted. Therefore, my choices were always based on the acquisition of a wide array of knowledge to find a way by which I perhaps could make some positive contribution to societal issues. This eventually led me to the Airborne Wind Energy research group.
Despite not meeting many of the prerequisites, Roland managed to find a project that was somewhat aligned with my background. It would require extra effort on my part, but I was drawn by the charm of a relatively young industry. There are still many uncertainties because it is a field that is still very much in development, which offers a lot of opportunities to add value. The journey has not been easy, but as the saying goes: "Nothing worthwhile is ever easy.".

With that said, I owe this experience to my parents, Wilma and Simon. The only way that I can thank them is to give my future children the same care and love they gave me. I also want to thank my girlfriend, Iris. She has been a supportive and patient emotional anchor to the stressful and rational bubble that Delft can be.
On a practical note, I would like to thank the AWE group who could always be called upon if any problems arose, and for the outings that were a nice distraction from the daily grind. Dylan Eijkelhof has my gratitude, for the gusto with which he wanted to help when I approached him with a question. I want to particularly acknowledge my daily supervisor, Jelle Poland. In our meetings, I could always rely on his infectious and unrelenting enthusiasm. My main supervisor, Roland Schmehl, also deserves my gratitude for entrusting me with this project, and who, when I approached him with a technical problem, could always point me in the right direction with an almost uncanny intuition.

A.R. Batchelor
Delft, November 2023

Abstract

The Airborne Wind Energy industry is evolving rapidly, aiming to establish its position among the major players in the conventional renewable energy sector. Since this industry is relatively young, it faces various challenges with scaling and integration into society. One aspect that can be improved is the design process of soft-wing kites. The aim is to shift from design iterations based on experience-based hardware modifications and experimental testing to design iterations based on optimization strategies using computational simulations. For optimization-based design to be a feasible option, it requires computationally efficient underlying models. This report presents the results of thesis research that aimed to develop a fast and reasonably accurate framework for structural modeling of soft-wing kites and connected tethers.

Deformation of leading-edge inflatable kites is essential for steering and controlling aerodynamic loads. Modeling of deformation is therefore indispensable, resulting in a highly non-linear fluid-structure interaction problem. Currently, a particle system model (PSM) is the preferred choice for modeling soft-wing kites because it can simulate deformations while converging faster than other methods. The existing PSM code in Java is outdated and the current academic preference has switched to developing in Python. This led to the central research question: “Can the PSM be implemented in Python, using a mix of Object-Oriented (OO) and non-OO programming techniques, to efficiently predict deformation?”.

While a framework was implemented based on the original Java code, it was also investigated whether it is possible to modify the solution-finding method for the structural model to improve convergence times without loss of accuracy. The aeroelastic problem can be simplified into a series of form-finding problems by assuming a quasi-steady flow. Three main families of form-finding methodologies are presented that are capable of finding solutions. Based on their properties and computational efficiency, the kinetic damping algorithm was selected as having the most potential for improving the current particle system implementation.
The kinetic algorithm was originally developed for a dynamic relaxation method that used an explicit integration scheme. Kinetic damping has never been combined with an implicit integration scheme. Slight adjustments needed to be made to the algorithm’s implementation as the current particle system employs an implicit scheme. Analysis of verification tests revealed characteristics of the utilized implicit Euler scheme. One notable effect was numerical damping, which hurt the performance of the kinetic damping algorithm. It was discovered that the algorithm’s performance was more consistent by disabling the quadratic correction.

Validation testing showed that each method is capable of accurately predicting the shape of tether and bridle line systems. To accurately simulate the deformation of membranes, more consideration must be given to the development of the PSM that represents the considered membrane continuum.
Next, the runtimes of the self-coded solver with explicit computation of Jacobian matrices and the framework with kinetic damping algorithm were compared to the performance of a black-box solver. It was found that the framework outperformed a black-box solver in runtime, demonstrating the advantage of using the self-coded solver.
Finally, benchmarking results indicated that the modified kinetic damping algorithm, despite some limitations, generally improved runtimes. Further investigation of the scaling of the runtime against increasing amounts of DOF indicated that the current framework could be fast enough for simulation in the range of 15 to 60 particles, despite the framework’s current non-optimized state.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AWE	Airborne Wind Energy
AWEC	Airborne Wind Energy Conference
BDF	Backwards Differentiation Formulae
BE	Bunny Earing
BE	Backward Euler
BiCGSTAB	BiConjugate Gradient Stabilized Method
CB	Canopy Buckling
CFD	Computational Fluid Dynamics
CS	Coordinate System
DOF	Degrees Of Freedom
DR	Dynamic Relaxation
FD	Flight Dynamic
FDM	Force Density Method
FE	Finite Element
FEM	Finite Element Method
FFT	Fast-Fourier Transform
FSI	Fluid-Structure Interaction
GSM	Geometric Stiffness Method
HM	Homotopy Mapping
IDE	Integrated Development Environment
IDM	Leading-Edge Indentation
IVP	Initial Value Problem
JF	JellyFishing
KCU	Kite Control Unit
LE	Leading Edge
LEI	Leading Edge Inflatable
MFDF	Multi-step Force Density Method with Force Adjustment
MIT	Massachusetts Institute of Technology
OO	Object-Oriented
PS	Particle-Spring System
PSM	Particle System Model
RK4	Runge-Kutta 4
QC	Quadratic Correction
SM	Stiffness Matrix Methods
SR	Seam-Rippling
TE	Trailing Edge
TEF	Trailing Edge Flutter
TLF	Total Lagrangian Formulation
TUD	Delft University of Technology
ULF	Updated Lagrangian Formulation
URS	Updated Reference Strategy

Latin symbols

Scalars	Definition	Unit
A	Area	m^2
c_d	Damping coefficient	N s/m
C_D	Drag coefficient	-
C_L	Lift coefficient	-
d_t	Diameter tether	m
E	Young's modulus	Pa
f_a^*	Characteristic aerodynamic frequency	Hz
f_F	Kite response frequency to change in force	Hz
F_D	Drag force	N
F_L	Lift force	N
f_r	Reduced aerodynamic frequency	Hz
F_s	Spring force	N
g	Gravitational acceleration constant	m/s^2
h	Timestep numerical integration	s
h	Vertical sag	m
H	Height	m
k_s	Spring stiffness	N/m
k	Matrix conditioning number	-
L	Length	m
l_0	Rest length	m
m_i	Mass of particle *i*	kg
n	Number of particles	-
n	Simulation timestep number	-
q	Force density	N/m^{dim}
q	Quadratic correction factor	-
t	Time	s
w	Thickness	m
W_{kin}^t	Kinetic energy of system at time t	J

Vectors and matrices		
$\boldsymbol{F}_{d,i}$	Damping force vector experienced by particle *i*	N
$\boldsymbol{F}_{\text{ext}}$	External force vector	N
$\boldsymbol{F}_{s,i}$	Spring force vector experienced by particle *i*	N
$\boldsymbol{I}$	Identity matrix	-
$\boldsymbol{J}_v$	Velocity Jacobian	N s/m
$\boldsymbol{J}_x$	Position Jacobian	N/m
$\boldsymbol{K}$	Global stiffness matrix	N/m
$\boldsymbol{K}_e$	Elastic stiffness matrix	N/m
$\boldsymbol{K}_g$	Geometrical stiffness matrix	N/m
$\boldsymbol{M}$	Mass matrix	kg
$\boldsymbol{P}$	Linear momentum	kg m/s
$\boldsymbol{R}$	Residual force vector	N
$\boldsymbol{T}$	Tensile force vector	N
$\boldsymbol{u}$	Displacement vector	m
$\hat{\boldsymbol{u}}_{i,j}$	Unit vector pointing from particle *j* to particle *i*	m
$\boldsymbol{v}_a$	Apparent velocity vector	m/s
$\boldsymbol{v}_i$	Velocity vector of particle *i*	m/s
$\boldsymbol{v}_k$	Kite velocity vector	m/s
$\boldsymbol{v}_n$	Velocity state vector at timestep n	m/s

$\boldsymbol{v}_{s,a}$	Apparent velocity vector of spring element	m/s
$\boldsymbol{v}_t$	Apparent velocity vector of tether segment	m/s
$\boldsymbol{v}_w$	Wind velocity vector	m/s
$\boldsymbol{x}_i$	Position vector of particle *i*	m
$\boldsymbol{x}_n$	Position state vector at timestep n	m

Greek symbols

Symbol	Definition	Unit
α	Angle of attack	°
α	Explicit integration scheme coefficient	-
γ	Damping term	N s/m/kg
ϵ	Strain	-
$\dot{\epsilon}$	Strain rate	s^{-1}
η	Viscosity	Pa s
λ	Eigenvalue	-
ν	Poisson's ratio	-
ρ	Density	kg/m^3
σ	Standard deviation	-
ω	Natural frequency	s^{-1}

Vector definitions

Throughout this report, kinematic relations and mathematical derivations will be shown. This requires a set of vector relations between particles *i* and *j*, which are defined in Equations 2 - 6. Equation 6 can be derived and simplified by making use of the quotient rule and equation 5 [45].

$$\boldsymbol{v} = \dot{\boldsymbol{x}}, \quad \boldsymbol{a} = \ddot{\boldsymbol{x}} \tag{1}$$

$$\boldsymbol{x}_{i,j} = \boldsymbol{x}_i - \boldsymbol{x}_j \tag{2}$$

$$\boldsymbol{v}_{i,j} = \boldsymbol{v}_i - \boldsymbol{v}_j \tag{3}$$

$$\hat{\boldsymbol{u}}_{i,j} = \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|} = \frac{\boldsymbol{x}_{i,j}}{|\boldsymbol{x}_{i,j}|} \tag{4}$$

$$\frac{\partial |\boldsymbol{x}|_{i,j}}{\partial \boldsymbol{x}_{i,j}} = \left(\frac{\boldsymbol{x}_{i,j}}{|\boldsymbol{x}|_{i,j}} \right)^T = \hat{\boldsymbol{u}}_{i,j}^T \tag{5}$$

$$\frac{\partial \hat{\boldsymbol{u}}_{i,j}}{\partial \boldsymbol{x}_{i,j}} = \frac{\partial \frac{\boldsymbol{x}_{i,j}}{|\boldsymbol{x}_{i,j}|}}{\partial \boldsymbol{x}_{i,j}} = \frac{\boldsymbol{I} - \hat{\boldsymbol{u}}_{i,j} \hat{\boldsymbol{u}}_{i,j}^T}{|\boldsymbol{x}_{i,j}|} \tag{6}$$

List of Figures

List of Tables

1 Introduction

To contribute to a solution to the current climate crisis, numerous technologies are being developed to improve the energy yield from renewable resources. Several of those technologies belong to the airborne wind energy (AWE) system category.
AWE systems use kites to harvest energy from altitudes beyond the reach of conventional wind turbines. Generally speaking, wind speed increases with altitude. The power density of the wind is a function of the cube of the wind speed. The average available power increases with altitude in most places, as explained more in-depth in [6]. Archer and Caldeira [5] identified the global potential of wind energy at altitudes of 500 m and above, an operational altitude as envisioned by the developers of the first AWE systems. An analysis of the resource potential at these altitudes and typical turbine heights was conducted by Bechtle et al. [9]. They found that the resources at the altitude that AWE systems operate in are more consistent, and an increase in locations where these resources can be harvested compared to typical cut-in wind speeds for turbines. A more recent study by Schelbergen et al. [61] presented a methodology based on empirical data that offers a better representation of wind conditions at potential site locations. They showed that their method identified more realistic wind profile shapes at these locations, addressing the uncertainties involved with extrapolating ground-level wind data to higher altitudes. While harvesting resources at high altitudes is still challenging itself, the promise that AWE holds is driving the search for solutions in this relatively young industry. A more in-depth introduction to the current state of AWE research and its advantages over conventional wind harvesting devices can be found in [62, 63].
To summarize, there is an untapped resource potential in wind energy AWE systems can be the key to unlocking it.

There are several AWE technologies currently in development, several of which are presented concisely in the Airborne Wind Energy Conference (AWEC) book of abstracts by Schmehl and Tulloch [64]. The present research focuses on soft-wing kites that fly cross-wind patterns in a so-called pumping cycle. They rely on tether traction to drive a ground-based generator which converts the mechanical energy into electricity.
Unlike a wind turbine's nacelle and rotor that rely on a static structure to provide support, kites, and thus AWE systems, are for the largest part highly dynamic. For soft-wing kites, the ability to deform the aerodynamically loaded tensile membrane structure is a requirement for steering and changing the wing's pitch angle to modulate the generated pulling force. This makes modeling the system very complex, and a trade-off needs to be considered between accuracy and simulation time. Simulating structural deformation accurately and quickly is necessary to optimize design, performance, and control, among others. Historically, a particle system model (PSM) has been the option of choice as it elegantly combines the speed and ease of implementation of lumped mass models with the possibility of modeling the deformation of tensile membrane structures [15, 22, 25, 35, 36, 39, 51, 52, 55, 71]. A visual example of a particle system model of a kite is illustrated in Figure 1.1.

A particle system (PS) framework for a flexible-wing kite was implemented by Van der Knaap [39], based on a generic processing Java code developed by Simon Greenwold [66] at Massachusetts In-

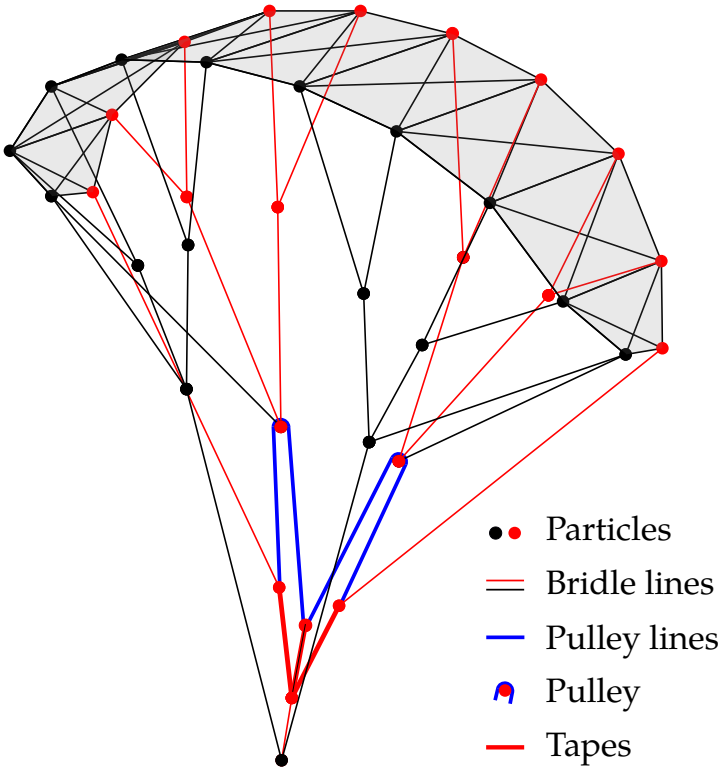


Figure 1.1: Particle system model of the TU Delft V3 LEI kite [52].

stitute of Technology (MIT). The Java code is currently outdated and the coding language of choice at Delft University of Technology (TUD) has shifted to Python, which forms a roadblock to further development. Combining different languages has been tried, but resulted in problems with coupling modules and performance issues [25]. For efficient future development, implementing the code in Python is a requirement. The primary goal of this thesis research is to implement, validate, and benchmark a PS framework in Python for further development as a structural model for soft-wing kites.

The computational efficiency of the PS framework was also tested by Van der Knaap [39]. He found that by adding rotational springs to the particle system framework, the inflatable support structures of the kite can be accurately modeled but at a high computational cost.
To simplify the simulation, a quasi-steady flow can be assumed. Leuthold [40] found that the validity of this assumption is justified. External conditions are kept constant, while the new position and shape of the kite are found by dynamic simulation of the PS until an equilibrium position is reached. Efficiently calculating the equilibrium position of comparable cable-net and membrane structures is exactly what the research field of structural form-finding tries to achieve. The literature in this field discusses various potential improvements to the PS framework and other methods used to find equilibrium positions. Many of these methods have not yet been tested for application in the structural modeling of soft-wing kites. Occasionally, researchers might find casual references to one of these methods in AWE literature, but the connection between the two subject areas has not yet been made. A review of form-finding literature has been conducted to identify methods or algorithms that show the potential of improving the framework.

The report begins with a literature review on the structural modeling of soft-wing kites and the quasi-steady-state assumption, which be read in Chapter 2. The chapter concludes by stating the main research question and several sub-questions. It is followed by a review of form-finding methods in Chapter 3. Chapter 4 describes and illustrates the physics behind and the development of the framework. It also covers external forces, which are not part of the inner workings, but paint a complete picture of all of the involved forces. Finally, the implementation of the kinetic damping algorithm that has the potential to improve the framework is explained. Chapter 5 follows and contains a justification

for the choice of solver with a derivation of the system of equations. To solve the system of equations, Jacobians are required, which are treated next. An analysis of the stability and accuracy of the chosen methods concludes the chapter. The implemented framework was verified with trivial test cases, which revealed characteristics specific to the implicit Euler scheme. These results are presented in Chapter 6. A selection of test cases was used to validate the PS, which are explained in detail and illustrated in Chapter 7. In Chapter 8 the benchmarking results of the framework are presented and analyzed. Finally, Chapter 9 concludes the report by answering the research questions and making recommendations for future researchers and developers of the framework.

2

Literature study

Miles L. Loyd [44] was the first to publish about the potential of harvesting wind energy by flying kites crosswind and presented his findings as two analytical models. The two concepts, called 'lift mode' and 'drag mode', rely on either aerodynamic drag or lift to generate electricity, and realizations for both can be seen in Figure 2.1.
Figure 2.1a shows the M600, a 600 kW fixed-wing kite developed by Makani, based on the 'drag-mode' concept. The kite flies in a circular trajectory to continuously generate electricity. The onboard rotors act as turbines and induce high aerodynamic drag forces on the kite. The tether also acts as the wiring to transfer the generated electricity to the ground station. Makani was a subsidiary of Google until the company was dissolved in 2020.
Figure 2.1b shows the Falcon, a 100 kW soft-wing AWE system by Kitepower, based on the 'lift-mode' concept. During the reel-out phase, soft-wing kites fly crosswind in typical figures of eight, illustrated by the black dashed line. The kite repeatedly flies these patterns while its radial distance in the downwind direction from the ground station simultaneously increases by reeling out of the tether. The tether is attached to a drum at the ground station, which, in turn, drives a generator to produce electricity.

This research project will focus on the structural model and simulation of leading-edge inflatable (LEI) soft-wing kite systems. The following sections present the results of the literature study on the requirements of an efficient computer model with reasonable fidelity and explain the need for such a model. An opportunity to develop a framework to aid future research is recognized and used to formulate a research statement with three sub-questions.

(a) M600 Prototype in Norwegian water [83] during the first offshore test flight of an AWE system.

(b) The Falcon harvesting electricity in former naval airbase Valkenburg [38] with an added figure of eight illustration.

Figure 2.1: Examples of realized AWE systems based on the two analytical models presented by Miles Loyd

2.1. Soft-wing kite system

Figure 2.2 identifies typical components in soft-wing kite systems. Bridle lines connected to the front of the kite, called leading edge (LE), support and transfer most of the aerodynamic load generated by the airfoil. Bridle lines connected to the rear, called trailing edge (TE), can be actuated to steer and (de)power the kite. The current theory regarding steering of soft-wing kites was devised by Breukels [14] and states that steering is achieved by a combination of roll and asymmetric deformation of the soft membrane structure. By deforming asymmetrically, the aerodynamic force vectors rotate, which creates an arm to the center of mass of the kite and results in a yawing moment. Actuating the amount of tension in the TE bridle lines, and thus the amount of asymmetrical deformation, allows the kite to be precisely controlled. This actuation can be performed at the kite control unit (KCU) or from the ground station, requiring an additional tether.

Powering or depowering the kite means adjusting the aerodynamic load that the kite experiences and can be achieved by (a combination of) several options. Altering the kite's velocity $\boldsymbol{v}_k$ results in a change in the apparent wind velocity $\boldsymbol{v}_a$ that the kite experiences, which directly influences the aerodynamic load as can be directly seen from Equations 2.1 and 2.2. The lift-to-drag ratio is, among others, a function of the membrane shape and the angle between the kite and the wind flow, called the angle of attack α. As the shape of the airfoil and α can be controlled by the actuation of the TE bridle lines, the lift-to-drag ratio of the kite can be adjusted. Finally, during flight, the aerodynamic properties of the kite can be modified by changing its structure, e.g. by introducing bleed-air spoilers.

$$\boldsymbol{v}_a = \boldsymbol{v}_w - \boldsymbol{v}_k \tag{2.1}$$

$$F_L = \frac{1}{2}\rho_{\text{air}} A C_l |\boldsymbol{v}_a|^2, \qquad F_D = \frac{1}{2}\rho_{\text{air}} A C_d |\boldsymbol{v}_a|^2 \tag{2.2}$$

Flying the kite in a crosswind pattern, e.g. the before-mentioned figure of eight, increases apparent wind speed. The lift force that the kite experiences scales with the magnitude of the apparent wind speed cubed, resulting in a tether tension that is orders of magnitude larger than could be achieved by flying directly into the wind. While the kite is in this powered state, the tether is reeled out and is consequently called the reel-out phase. After the reel-out phase, when the drum is fully unwound and the tether has reached its maximum length, the kite is depowered and transitions to the reel-in phase. By depowering, the reeling-in of the kite consumes less energy than the powered kite can generate during reel-out, which makes a net positive energy cycle possible. The process of reeling in and out and transitioning between these phases is referred to as a pumping cycle, as illustrated in Figure 2.3.

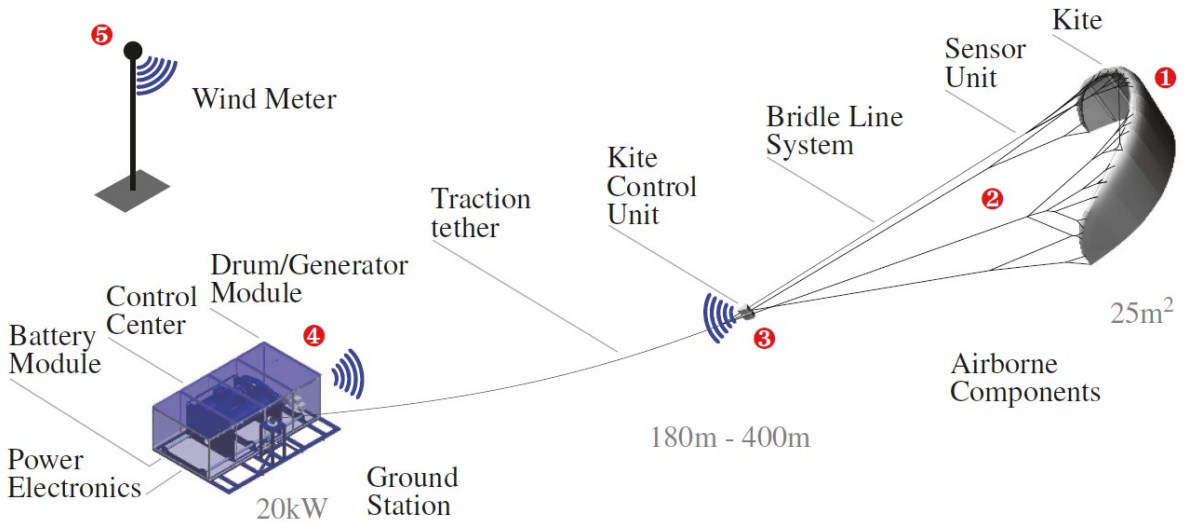


Figure 2.2: Identification of AWE system components, with communication and sensor locations (red numbered dots) [77].

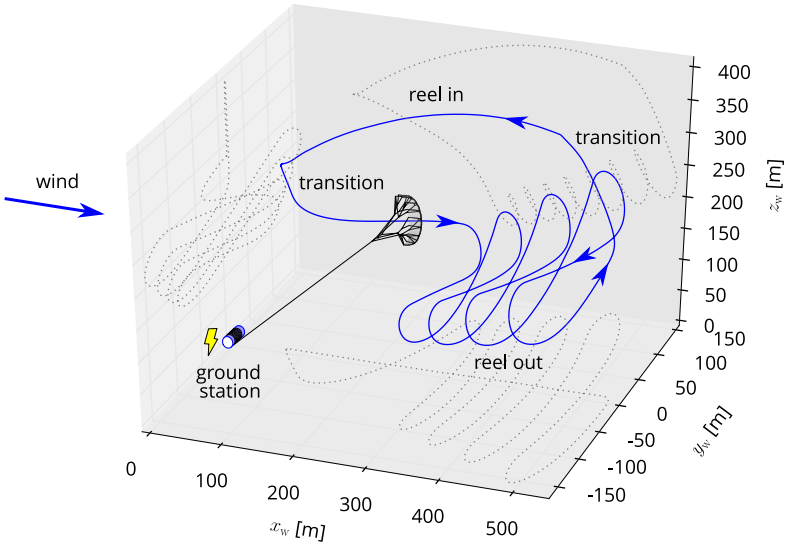


Figure 2.3: Flight path of a soft-wing kite system, with the distinct phases of a pumping cycle indicated [21]

Optimizing the ratio between energy yield and expenditure is one of the challenges that the industry currently faces. As Poland [51] argues, transitioning from experience-based design to simulation-based design that allows for optimization algorithms would accelerate the current design process of the industry. This would require a simulation framework that must at least be accurate enough to capture the main interactions between wind and kite but also must converge to results in a reasonable time. Generally, real-time or even faster simulation is set as a goal, since the design process is likely to take many iterations [51, 52, 58]. Designing a fast and accurate enough model requires the interaction between the airfoil and wind flow to be dissected to justify whether the problem should be modeled fully dynamic or quasi-steady.

2.2. Kite deformation modes

Having a high lift-to-weight ratio is advantageous for kites as it makes flight at lower wind speeds possible and lowers the energy expenditure to keep the kite airborne. LEI kites are made up of pressurized tubular chambers, called struts, that determine the shape of the kite. These struts are joined by fabric, called the canopy, to provide structure to the kite. Generally, savings in kite weight are coupled with reduced structural strength. Soft-wing kites are therefore easily deformed, which is advantageously used for steering and (de)powering. However, during flight, the kite is also deformed by the aerodynamic load that it experiences. In turn, this deformation affects the aerodynamics of the kite. This strong two-way coupled effect between structural deformation and the wind flow around the structure is called aeroelasticity, a subset of fluid-structure interaction (FSI). The various modes of unintentional deformation that occur during operation are illustrated in Figure 2.4 and were divided by Leuthold [40] into local sub-scale and global large-scale deformation modes. While subscale deformations can be neglected as they have no significant effects on flight dynamics, as stated by Bosch et al. [12] and Breukels [14], the opposite is true for large-scale deformations. The ability of the model to capture large-scale deformations is required for accurate results. This was also further substantiated by Leuthold [40], who estimated a typical frequency band for each of the deformation modes as well as for the flight path frequencies, which are sorted in Figure 2.5. The frequencies subscript are abbreviated as follows: trailing-edge flutter (*TEF*), seam-rippling (*SR*), canopy buckling (*CB*), jellyfishing (*JF*), bunny earing (*BE*), leading-edge indentation (*IDM*), figure eight flight (*8*), flight dynamic (*FD*) and aerodynamic (*a*).

2.2.1. Quasi-steady flow

The reduced aerodynamic frequency (f_r) serves as an indicator to categorize fluid dynamic problems as quasi-steady or unsteady. The dimensionless quantity depends on two frequencies: the characteristic aerodynamics frequency (f_a*), which represents the inverse of the time it takes for the airflow to traverse the kite's chord, and the frequency at which the kite responds to the change in force (f_F). The latter

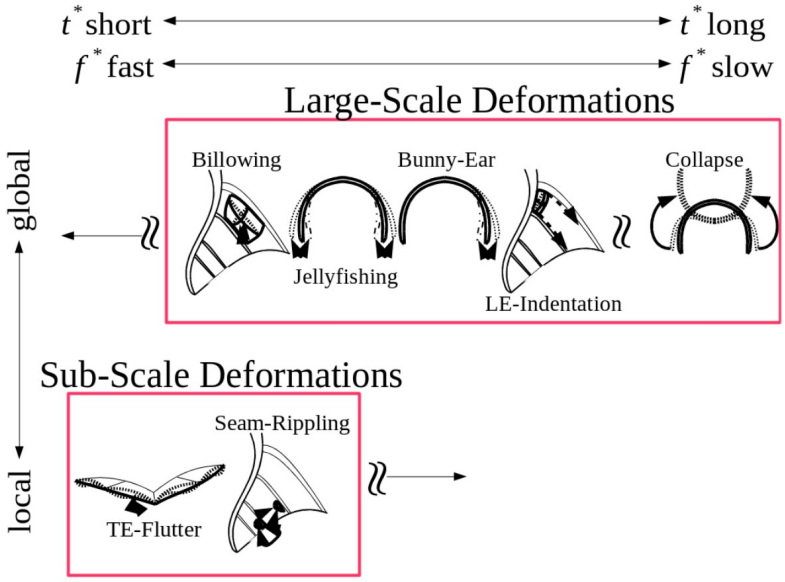


Figure 2.4: LEI kite deformation modes, grouped by scale and ranked according to characteristic frequency [40].

encompasses the sub-scale, large-scale, and flight path frequency modes.

Equation 2.3 shows the definition of f_r. If f_a^* is much larger than f_F ($f_r \ll 1$), the problem can be considered quasi-steady, as the fluid element effectively experiences a steady kite while it passes over the airfoil. Conversely, if f_a^* isn't much larger than f_F ($f_r \not\ll 1$), the problem must be treated as unsteady, signifying that the kite deforms as the fluid element is passing over it.

$$f_r = \frac{f_F}{f_a^*} \tag{2.3}$$

Leuthold [40] found that the large-scale deformations could be resolved with a quasi-steady model, illustrated in Figure 2.5. These findings are in line with the results of Kappel [34], who neglected the sub-scale deformation in his model and generated results similar to the experimental data. Based on these findings, assuming a quasi-steady-state model is deemed acceptable, which simplifies implementation and, additionally lowers computational cost in comparison to a fully dynamic model [13].

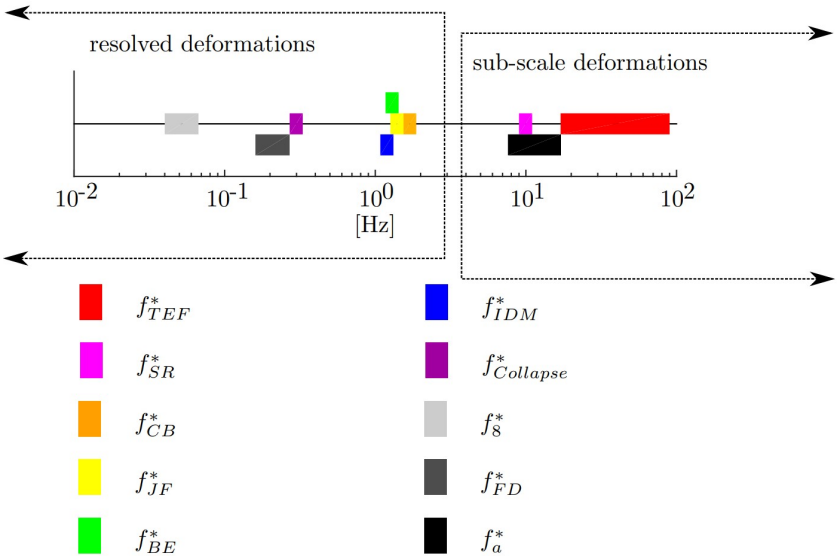


Figure 2.5: Characteristic frequencies of operational modes (black and white tones) and deformation modes (coloured) [40].

2.3. Simulation framework

The field of FSI solvers can be divided into two categories: monolithic and partitioned. Monolithic solvers represent the aeroelastic behavior of the system, including structural deformations influenced by aerodynamic forces, in a single set of equations. On the other hand, partitioned FSI solvers separate the aeroelastic model into aerodynamic and structural components, which is graphically illustrated in Figure 2.6. These models interact at their interfaces, but their mechanics do not directly interfere with each other. The aerodynamic model calculates the pressure distribution for a given body shape, while the structural model computes the deformation for a given aerodynamic loading. Consistent boundary conditions are applied through the coupling between the fluid and structural solvers to ensure accuracy and stability.

Monolithic solvers have the advantage of customization for specific problems and accurate modeling of all interface effects. However, their application to arbitrary physical problems can be challenging, and updating them to incorporate field-specific advancements may also prove difficult and time-consuming. On the contrary, partitioned FSI solvers offer modularity between the fluid and structural models. This allows developers to modify either or both models to improve accuracy and efficiency or explore new concepts. This flexibility is particularly valuable during earlier stages of development, as the tradeoff between accuracy and speed can be explored and reviewed, guiding future development effectively.

In 2020, Folkersma, Schmehl, and Viré [24] built a two-way coupled aeroelastic solver for ram-air kites, based on a computational fluid dynamics (CFD) solver and a finite element (FE) solver. Similar success has not yet been achieved for LEI kites, despite the fact that research efforts have been devoted to developing a similar solver approach. Poland [51] concluded that for design optimization of LEI kites a solver based on the combination of CFD and FE is too computationally expensive, due to their intricate geometry and attachment points. One of the issues with existing FSI models is the trade-off

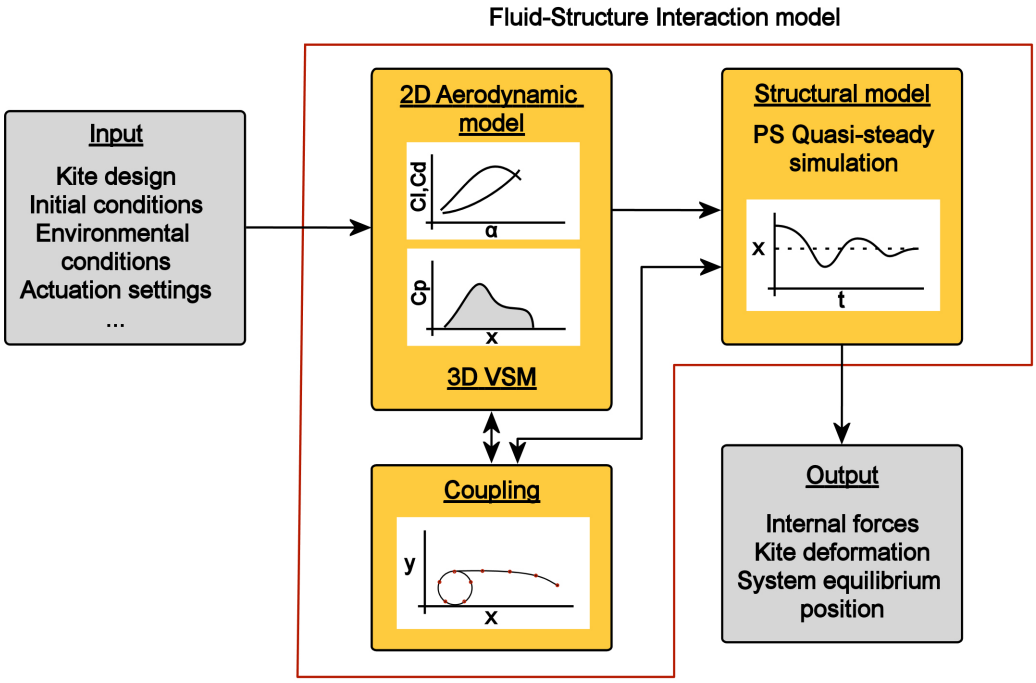


Figure 2.6: Diagram of a possible partitioned FSI solver, where the aerodynamic and structural models are modularly implemented

between computational cost and level of detail. Some models use rigid body assumptions, which do not accurately represent the in-flight behavior of systems with significant deformations, such as LEI kites. Therefore, to accurately study the aerodynamic performance of kites and efficiently develop designs to enhance it, a less accurate but faster structural and aerodynamic model is needed. Additionally, including the tether and bridle lines separately is essential for simulating a pumping cycle operation [25].

In brief, using a partitioned FSI solver offers benefits when research and development are still in an early stage. To ensure an accurate representation of the interaction between wind and airfoil, it is essential to have a structural model that can simulate deformation. Existing combinations of aerodynamic and structural models are either not accurate enough or not fast enough.

2.3.1. Aerodynamic model

As the aerodynamic model is not the main topic of this investigation, its implementation will be limited. The aerodynamic model is the subject of a separate research paper by Watchorn [80]. Due to a lack of experimentally determined data on kite deformation, no kite model will be implemented and thus no model for its aerodynamic load is required. Regarding the aerodynamics of the tether, the calculation of forces and distribution to the particles follow the model that Geschiere [25] presented.

2.3.2. Structural Model

Ruppert [58] reviewed the options for structural modeling, in efforts to develop a model for LEI inflatable kites, which can be seen in Figure 2.7. To this day, unless for specific reasons higher-fidelity models are required, most studies on LEI kite modeling have argued for the use of a PSM [15, 22, 25, 35, 36, 39, 51, 52, 55, 71] because its computational efficiency while still being capable of simulating kite deformation. The open-source Java PSM library, developed by Simon Greenwold [66], was utilized by the Kite Power research group at TUD to develop the KiteSim framework, which was capable of simulating crosswind LEI kite power systems. Currently, both the library and framework are outdated and do not run anymore. An open-source kite simulator was written in Python by Uwe Fechner [73], but the code was translated to the Julia programming language [74] and thus the Python version also became outdated. Having a structural model written in Python would be a merit, as it currently is the language that students learn and thus simplifies further development. In terms of programming languages, as Python is a general-purpose language it is more versatile than Julia, and it has been around for much longer, solidifying its robustness.

A PSM approximates the structure by discretizing it into lumped masses, in this context more often called particles, which are connected by weightless spring-damper systems. To find the quasi-steady equilibrium position, i.e. $\ddot{x}$ and $\dot{x}$ are 0, the system is dynamically simulated over a pseudo-time, i.e. numerical integration steps, until convergence criteria are met. This solution-finding method is based on Newton's second law, where the imposed forces on the particles can be coupled to the internal forces and resulting accelerations (Eq. 2.4). This dynamic solution is computationally expensive when considering that only the steady-state solution is of importance. The data between the initial condition and steady-state solution has no physical meaning and therefore no value. A solution-finding method that applies this knowledge in an intelligent algorithm or even completely skips the dynamic phase would increase the computational efficiency, which is an obvious advantage.

$$\begin{aligned} \Sigma \boldsymbol{F} &= \boldsymbol{M}\ddot{\boldsymbol{x}} \\ \boldsymbol{M}\ddot{\boldsymbol{x}} + c_d\dot{\boldsymbol{x}} + k_s\boldsymbol{x} &= \boldsymbol{F_{ext}} \end{aligned} \tag{2.4}$$

To dynamically reach a steady-state solution, numerical integration of stiff differential equations is required, which limits the value of the spring stiffness k_s and can result in stability issues [43]. This is also reported by Poland and Schmehl [52] and Cayon, Gaunaa, and Schmehl [15], who determine the value of k_s by limiting the elongation to around 2-3 %. Improving the solution-finding method to where the true material stiffness can be used would be of value, as no more trial-and-error search would be required to stabilize the simulation based on an estimated strain limit.
An effort by Poland [51] to develop a novel geometric structural model required additional empirical relations between bridle line actuation and kite deformation to fully geometrically define the shape without the use of forces. Although the geometric-based model had a runtime of three orders of magnitude

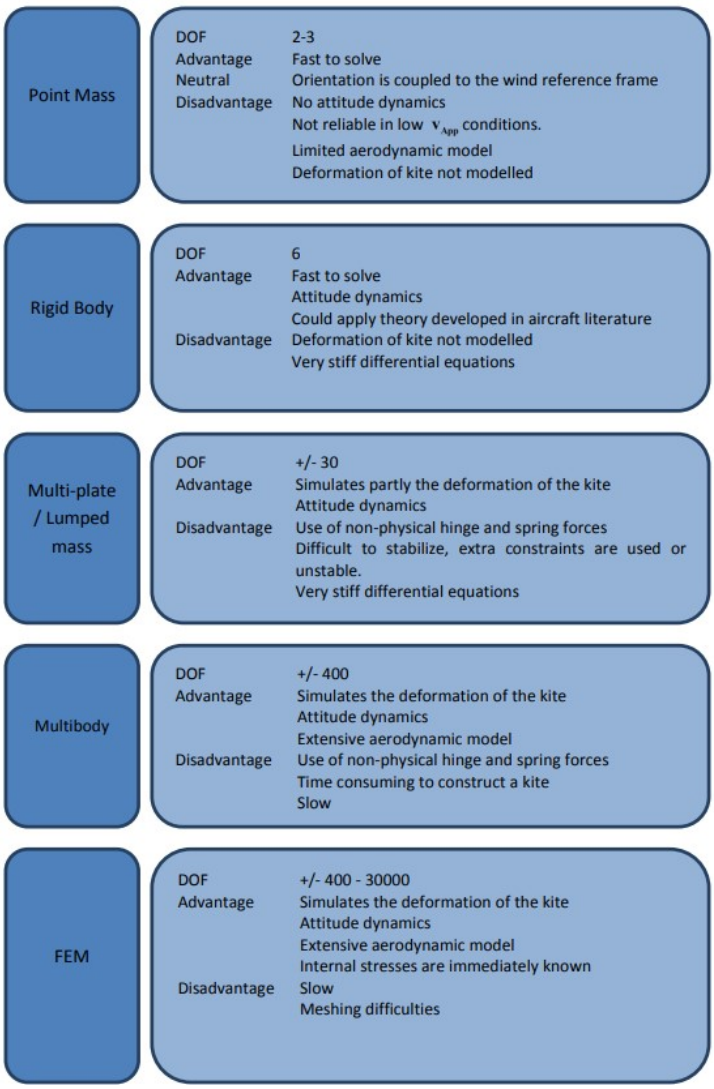


Figure 2.7: Overview of structural modeling options [58].

smaller than that of his implemented PSM with a black-box solver, he determined that the latter is better suited as a structural model because it can simulate asymmetric deformation without relying on empirical relations. Recent research using a PSM as the structural model reinforced its suitability to predict asymmetric deformation [15, 52].

A PSM is the preferred structural modeling method for LEI kite research, despite its associated drawbacks. Improving its computational efficiency would be of great value, as would implementing alternative solution-finding methods that could speed up convergence times. As Julia is a fairly new programming language a more robust and established language is preferred, but currently no functioning open-source code exists in such a coding language. To enable future research and development, coding a new framework is necessary. Python will be utilized for this purpose, as it is presently the most widely used programming language.

2.3.3. Coding architecture

The existing Java implementation of the PSM follows an object-oriented architecture. This architecture is used because it is very suitable to build a problem from basic building blocks, in this case, massless spring-damper elements and lumped mass elements. On the other hand, the core of the program uses a one-dimensional vector data structure for the most efficient solution finding. The one-dimensional

vector data structure is ideal for numerically solving the linear system of equations with a sparsely populated system matrix and is capable of achieving faster than real-time simulation speeds. The reason for this efficiency gain is avoiding time-consuming traversing of deeply nested object-oriented data structures. For these reasons, this basic code architecture will be reused. A repository for the code and resulting figures was made and can be found online [3]. A copy of the developed framework can be found in appendix A.1.

2.4. Simulation validation

The conventional method for assessing the aerodynamic properties and resulting shapes of airfoils involves conducting wind tunnel tests using scaled-down models or employing high-fidelity simulations. Wind tunnel tests can be expensive, and because of the complex interaction between fluid dynamics and structural properties, the results from scaled models often can't be directly applied to full-sized systems. Additionally, accurately simulating the aerodynamics of flexible membrane wings that include the aeroelastic effects is still quite tricky for current computational methods [78]. Therefore, the results of the simulation usually are validated with specific test cases.
Several test cases can be found in literature for validation and benchmarking the particle system framework of springs, dampers, and masses. The selected test cases isolate as much as possible a specific functionality of the developed software. While for some cases experimental measurements exist, other cases have to be compared to the theoretical expected outcome. The following test cases are selected for validating the framework.

1. A hanging tether, fixed at its top end, that exhibits longitudinal elastic vibrations as a result of an attached mass being dropped from a certain height.
2. A tether, fixed at both ends, that is deflected by a perpendicular air flow and experiences no gravity.
3. A horizontally suspended tether, fixed at both ends, that is deflected by gravity in the vertical direction.
4. Hencky's problem, where a flat circular membrane is subjected to uniform pressure at one side, resulting in nonlinear forces.

These validation tests can be performed with meshes that vary in coarseness by discretizing the structure into varying amounts of particles, which essentially determine the degrees of freedom (DOF) in the system. This methodology will be used to benchmark the performance of the framework. The convergence criteria are based on the sum of residual forces in the system. Furthermore, the parameters that govern the dynamic formulation ($\boldsymbol{M}$, c_d, k_s) can be varied to analyze how they affect accuracy, stability, and runtime.

2.5. Research questions

To accurately simulate LEI kites the model has to be capable of predicting shape. Assuming quasi-steady state flow conditions is justified and simplifies the model and lowers computational cost. In this context a PSM is the model of choice, as it is capable of simulating deformation while having a low computational cost. The existing PSM code in Java is outdated and academic preference has shifted to Python. This leads to the central research question:

Can the PSM be implemented in Python, using a combination of Object-Oriented (OO) and non-OO programming techniques, to efficiently predict deformation?.

Three sub-questions are defined, listed as follows.

1. Is it possible to modify the existing PS framework to improve convergence times, without loss of accuracy?
2. How does a black-box solver perform compared to the self-coded solver with explicit computation of Jacobian matrices?
3. How does the runtime scale when increasing the amount of DOF?

To answer these research questions, four distinct research objectives are identified and listed below.

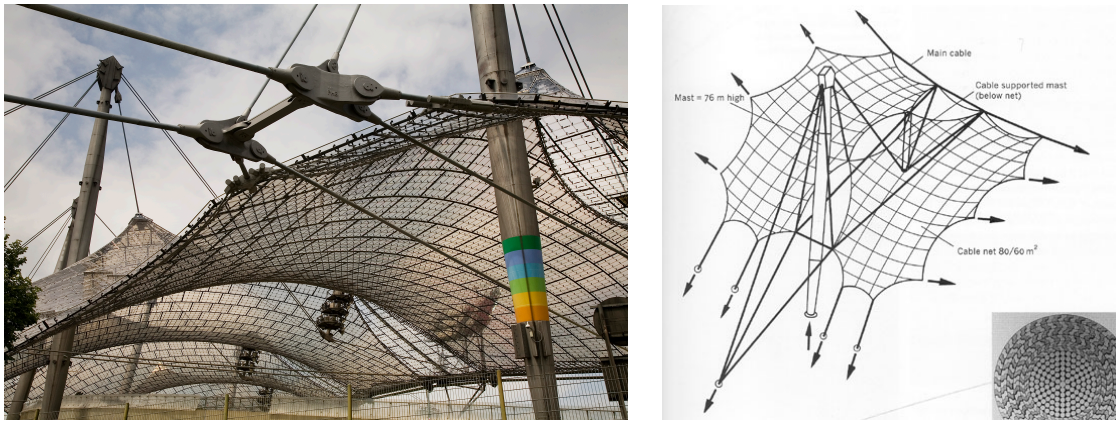
1. Implementation of PSM framework in Python.
2. Incorporating kinetic damping into the PSM.
3. Validating the framework with the identified test cases.
4. Conducting performance testing of the implemented framework for an increasing number of DOF.

3

Structural form finding

The ambitious designs for the Munich Olympic complex triggered the need for computational models of shell structures and ushered in a new era for the science of structural form-finding. Structural form-finding is concerned with developing methods that can efficiently and accurately calculate the shape, or static equilibrium, of thin structures that transfer load through axial forces. A distinction can be made between unstrained gridshells when the elements of the structure solely experience compressive forces, cable-nets when solely tensile forces are experienced, and tensegrity when both compressive and tensile forces are present.
Part of the built complex and one of the design drawings are visible in Figures 3.1a and 3.1b respectively. These images serve to illustrate the similarities between this particular problem and the deformation modeling of LEI soft-wing kites. Both are concerned with finding the equilibrium position of a membrane structure held together by (for the most part) tensile elements. Quasi-steady simulation of an LEI kite system can be seen as a chain of form-finding solutions, with imposed forces varying from step to step. Note that this comparison only holds when quasi-steady state is assumed, a fully dynamic simulation of a kite system requires a different approach.

The field of structural form-finding is characterized by researchers directing their efforts at individual methods. An absence of standardization resulted in a branch of science that uses a multitude of nomenclatures, mathematical structuring, and symbolic notation. Impartial comparisons of the performance of methods in one mathematical framework didn't exist. To this day this complicates research on the topic and the selection of optimal methods for specific applications. Precisely for this reason Veenendaal and Block [76] conducted a review, improving accessibility and clarity on structural form-finding methods.



(a) Plexiglas canopy and suspension system formed by cables and masts [31]. **(b)** Design of the suspension system of the canopy [46].

Figure 3.1: Detailed view of the roof of the Munich Olympic complex and design drawing of shown roof.

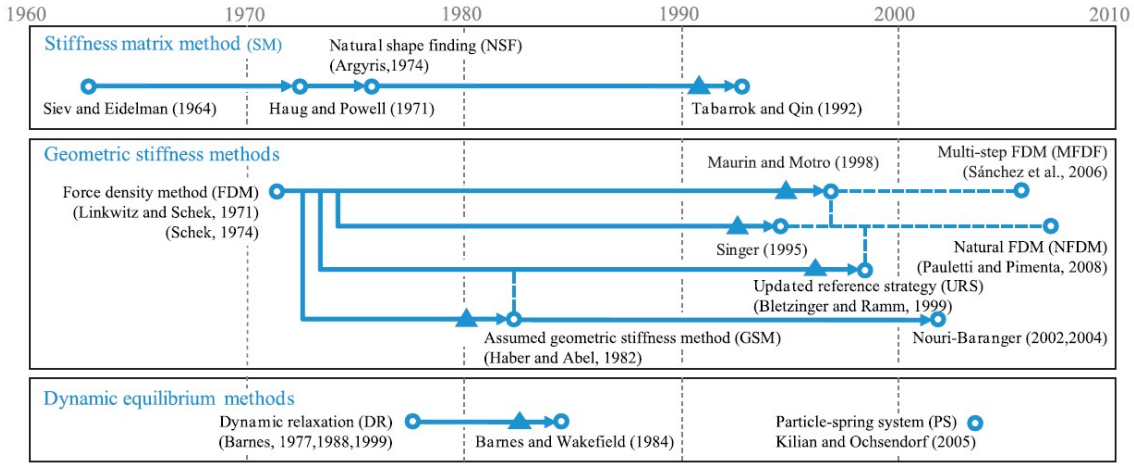


Figure 3.2: Categorized timeline of the development of form-finding methods including key references. Circles denote methods, arrows denote descendence, triangles denote an extension of the method to surface elements, and blue dotted lines link related but independent methods [76].

Veenendaal and Block classified form-finding methods into one of three main families and structured the chronology accordingly, shown in Figure 3.2. Notably, a PSM based on Simon Greenwold's Java code library was used for form-finding by Kilian and Ochsendorf [37] and is listed as a distinct method in the review. Next to the main methods many more variations exist, specifically adapted to the needs of certain applications.

3.1. Form-finding methods

Structural form-finding is closely related to techniques for structural analysis such as FEM and, as such, often requires the use of similar properties. The elastic and geometric stiffness matrices, $\boldsymbol{K}_e$ and $\boldsymbol{K}_g$ respectively, are examples of such properties and are used to categorize families in the suggested classification. Elastic stiffness is derived from Hooke's law which links applied forces or stress with deformation based on material properties. It is the dominant factor when structural deformations are small. Geometric stiffness stems from geometric nonlinearity in a structure, meaning that as the structure deforms, it also affects the distribution of forces and the stiffness of the structure. This is primarily the case when structural deformations are large. Tether dynamics and sag are an example of non-linear behavior in a structure. Sagging of the structure affects the direction of resistance, and the aerodynamic forces are a function of orientation as described in Chapter 4. This means that finding an analytical solution for its form is extremely challenging, if not impossible. Kite deformation has even more sources of non-linear behavior. Similar to tether and bridle lines, the aerodynamic load and structural stiffness change as the structure deforms. Furthermore, the inflatable struts are characterized by non-linear bending stiffness [11]. In theory, each family of methods should be able to accurately predict non-linear behavior and converge, but in practice, they have their limitations. According to the classification proposed by Veenendaal and Block [76], the families can be characterized and distinguished as follows.

The stiffness matrix (SM) methods rely on the use of both elastic and geometric stiffness matrices. From these matrices, a global stiffness matrix $\boldsymbol{K}$ is assembled, representing the stiffness of the whole structure. Equation 3.1 is used to find the displacements of the structural nodes $\boldsymbol{u}$ from their starting position. The equation does this by balancing internal stress with the residual load vector $\boldsymbol{R}$. Then, from the current geometry, for small fractions of the residual force vector $\Delta\boldsymbol{R}$ intermediate solutions $\Delta\boldsymbol{u}$ are found. This way small displacement is preserved, which is an underlying assumption of SM. Convergence is reached by iterating until $\boldsymbol{u}$ is found. It may be clear that this family of methods has a historical foundation in structural analysis.

$$(\boldsymbol{K}_e + \boldsymbol{K}_g)\boldsymbol{u} = \boldsymbol{K}\boldsymbol{u} = \boldsymbol{R} \tag{3.1}$$

Although SM methods utilize physical material properties, a drawback is that these properties are not required to find a solution and are computationally expensive [41, 48]. Experience is required in convergence control, as small or even zero stiffness in certain directions can cause large displacements or divergence.

Geometric stiffness methods disregard the material properties and rely solely on the geometric stiffness of the structure. These techniques stem from the force density method (FDM) [60], which, as its name suggests, employs force densities. The FDM attempts to find the shape of a structure by resolving a set of non-linear equations that balances tension force $\boldsymbol{T}_i$ for each element in the three primary directions of a Cartesian coordinate system (CS) (Eq. 3.2). The force density q is not a unit, but a concept used to describe the distribution of force in the structure. It is a measure of force per unit of dimensionality of the problem, used to linearize the set of equations that determine the position of the nodes. For the given example $q_i = \frac{\boldsymbol{T}}{L}$ with L being the element length, where the operator will set the value of q_i to control the outcome surface. Subsequent studies [10, 28, 50] present methods that prescribe forces rather than non-intuitive force densities, as the resulting shapes are difficult to predict.

$$\sum \frac{\boldsymbol{T}_i(\boldsymbol{u})\boldsymbol{u}_{\boldsymbol{e}_x}}{L_i} = 0, \quad \sum \frac{\boldsymbol{T}_i(\boldsymbol{u})\boldsymbol{u}_{\boldsymbol{e}_y}}{L_i} = 0, \quad \sum \frac{\boldsymbol{T}_i(\boldsymbol{u})\boldsymbol{u}_{\boldsymbol{e}_z}}{L_i} = 0 \tag{3.2}$$

Lewis [41] states that the linear application of geometric stiffness methods may produce results with stresses that are beyond material limits and can serve only as a preliminary result. These results are also dependent on the anisotropy of the considered material and meshing strategy. Additional constraints or different formulations are required to produce results with practicable internal stresses. This changes the set of equations to a non-linear form, for which additional iterations are necessary.

Dynamic equilibrium methods solve a dynamic formulation over a (pseudo-)time to arrive at a steady-state solution, rather than using stiffness matrices. The steady-state solution is equivalent to the static equilibrium solution. The stiffness is embedded in the geometry of the structure, which updates every iteration. This makes these methods particularly well-suited for problems with high degrees of bending, which is the case for LEI kites. The kinetic damping algorithm has proven to be inherently stable for highly nonlinear problems.
Nouri-Baranger [48] Noted a few criticisms on DR as follows. These methods require many parameters, such as the time step, to control stability and convergence. The mass and damping parameters are fictitious, have no physical representation, and may therefore not be meaningful. The latter is explained more in-depth in chapter 4, and doesn't necessarily constitute a disadvantage. The formerly stated disadvantage depends on the method of numerical integration and control may be reduced to a singular parameter. The author would argue that the largest disadvantage of DR would be the dynamic process required to reach the steady-state solution. It is an iteration-heavy process, and therefore time-consuming, when the transient phase isn't of value, which is the case when the quasi-steady state approximation is assumed.

3.2. Computational performance

Veenendaal and Block [76] tested the performance of the distinct methods that they identified in their framework, the results of which are listed in table 3.1. Convergence times are normalized relative to the convergence time of the fastest method t_{min}. The tested methods are abbreviated as follows: stiffness method (SM), multi-step force density method with force adjustment (MFDF), geometric stiffness method (GSM), updated reference strategy (URS), dynamic relaxation (DR), and particle spring system (PS). The corresponding subscripts are abbreviated as follows: updated Lagrangian formulation (ULF), homotopy mapping (HM), viscous damping (vis), kinetic damping (kin), Runge-Kutta 4 integration scheme (RK4), and backward-Euler integration scheme (BE).

From this table, several observations can be made. The standard SM could not converge at higher amounts of DOF, emphasizing the difficulty with convergence control. ULF is a framework where intermediate solutions are found and used as new reference shapes for which a new stiffness matrix needs

Table 3.1: Solution times of implemented methods for increasing DOF, normalized with respect to the fastest method in bold, and number of iterations in parenthesis. Adapted from [76].

DOF	**75**	**183**	**339**	**543**
SM	1.25 (**8**)	N/A	N/A	N/A
SM_{ULF}	1.44 (13)	4.22 (16)	6.26 (18)	9.93 (17)
MFDF	**1.00** (14)	**1.00** (16)	**1.00** (18)	**1.00** (17)
GSM	1.52 (16)	2.64 (15)	3.77 (14)	3.73 (13)
URS_{HM}	1.38 (10)	3.13 (**8**)	4.31 (**9**)	5.36 (**8**)
DR_{vis}	1.22 (**8**)	5.40 (34)	15.54 (63)	24.66 (96)
DR_{kin}	1.41 (16)	5.19 (32)	10.36 (42)	13.16 (50)
$PS_{RK4,vis}$	1.78 (17)	3.68 (22)	6.74 (28)	11.30 (50)
PS_{RK4}	4.10 (39)	6.58 (39)	13.57 (60)	22.73 (67)
PS_{BE}	6.31 (37)	11.30 (32)	14.44 (30)	20.33 (30)
t_{min} [s]	0.007	0.015	0.026	0.040

to be calculated. Although this framework made convergence possible, it could also be a reason for the longer convergence times.
The performance of the geometric stiffness methods (middle three) is prevalent, with MFDF in particular. Veenendaal and Block [76] state that its performance can be explained by the fact that the algorithm begins with a force density-controlled iteration before switching to force control. The other methods begin iteration from prescribed forces, which makes their performance more dependent on the initial geometry. The relative performance of the MFDF method may decrease considering that in quasi-steady kite simulation, the consecutively found shapes are near each other, instead of being found from an arbitrary initial geometry.
Dynamic equilibrium methods (bottom five) are among the worst in terms of performance, certainly at higher amounts of DOF. The addition of the kinetic damping algorithm to DR seems to worsen performance at lower amounts of DOF. The change appears to lower the scaling coefficient, as the method has lower runtimes from the 183 DOF mark. The PS with an implicit BE scheme also scales better with increasing DOF. The number of iterations remains relatively constant, while computational cost per iteration doesn't grow as much compared to the increase of iterations that methods with implicit schemes experience. This is in line with the findings of Baraff and Witkin [7] who reported that implicit schemes outperformed explicit based on examples with 7806 DOF. Reducing the number of iterations would benefit methods with implicit schemes, which might be achieved as the transient phase holds no value.

3.3. Method selection

Considering the time constraints of a single thesis project, a choice in which method to implement and test was made based on applicability, difficulty of implementation, and computational efficiency. The reasoning behind this choice is as follows.

Implementing a LEI model based on SM has been attempted (successfully) by Bosch, Thedens [13, 71]. Thedens [71] used a FE formulation in a curvilinear CS, but couldn't find a stable solution with the static Newton-Rhapson method. As such he had to resort to DR for which he included kinetic damping. The FE analysis that Bosch [13] performed on LEI kites was based on the direct stiffness method (DSM). He had to develop a control algorithm to get the method to converge, as out-of-plane forces cause large deformations with low bending stiffness or singular matrices with zero bending stiffness. The author encountered similar problems in an attempt to implement the DSM, also noting that in certain cases multiple solutions might be possible, adding a layer of difficulty. Implementing an SM is time- and resource-intensive and combined with its mediocre performance-wise scaling it is not considered as a feasible option for optimization-based design.

Although the geometric stiffness methods are a promising topic to pursue, they demand additional effort on the side of researchers and developers. Familiarity and expertise in force-density control or force control need to be gained and extra steps are required to make sure that the results are practicable. This unfortunately requires too much time for the current research project, but it could be an interesting topic for a future thesis project. An open-source FDM-based code in Python was implemented by Vahid Moosavi [75] and could be used as a starting point.

That leaves the DR methods, of which the addition of a kinetic damping algorithm to the framework is a promising option that could potentially speed up runtimes. This algorithm has been used in combination with explicit integration schemes, but to the author's knowledge not yet with an implicit scheme. In the following chapter, readers can find the theory behind the algorithm, its application in the developed framework and the anticipated effects.

4

Particle System Framework

In the early stages of their application, particle systems relied mainly on the internal properties of particles to simulate the dynamic characteristics of ”fuzzy objects” such as fire or smoke [56]. In an effort to achieve an accurate simulation of collective animal behavior using a particle system, Reynolds [57] introduced the concept of particles that could also be influenced by external factors, including other particles. Over time, this idea transformed the particle system into a tool for cloth simulation and form-finding of net structures and membrane surfaces.
This chapter opens with a brief high-level summary of setting up a PSM and simulating it in a PS framework. This is followed by a more in-depth explanation of the objects that hold the structure together, namely the springs and dampers. Next, the externally imposed forces are described that drive the shape in the selected test cases, as per the *form follows force* principle. Finally, the kinetic damping algorithm intended to improve the performance of the framework is explained.

4.1. Particle System

The first step in setting up a PSM is to discretize the structure of interest into lumped masses or particles of infinitesimal size. These particles are then connected by massless springs and dampers, the data of which is stored in a connectivity matrix. This way, a larger structure can be split up into a finite amount of particles n that can be described by idealized fundamental equations, which makes analysis of the structure more manageable. The absence of a body of any size or shape for the particles means that they can be fully described by a position vector $\boldsymbol{x}_i$, a velocity vector $\boldsymbol{v}_i$ in the case of dynamic simulation, and a mass m_i. Here, the subscript i denotes the index of the corresponding particle. Depending on the requirements and applicability the model can coupled with either a dynamic or direct solution-finding method.
During its initial stages, the PS was simulated dynamically with explicit integration schemes. Later, preference shifted when Baraff and Witkin [7] presented the advantages of using implicit schemes as explicit integration schemes run into stability issues when certain limits to the step size are exceeded. The consideration between explicit and implicit integration schemes will be covered in Chapter 5. Diagrams that visualize the structure and the behavior of the implemented PS framework can be found in Appendix B.2.

4.2. Internal Forces

The aim of connecting particles with massless spring-damper systems is to accurately recreate material properties and behavior. There is a multitude of spring-damper configurations that try to replicate material behavior, but an energy-dissipating component is required as the system otherwise would keep oscillating indefinitely after initial excitation. In the developed PS framework the connection is formed by one spring element and one damper element in parallel, which is also known as a Kelvin-Voigt material [82]. In this configuration, the strain is equal for both spring and damper elements. The resulting constitutive relation (Eq. 4.1) dictates that the stress on the material σ is a sum of the product of Young’s modulus E and the strain ϵ and the product of the viscosity η and the strain rate $\dot{\epsilon}$.

$$\sigma = E\epsilon + \eta\dot{\epsilon} \tag{4.1}$$

The strain terms on the right-hand side of Equation 4.1 can be related to the state ($\boldsymbol{x}$, $\boldsymbol{v}$) of connected particles. This is how the framework calculates the values for the internal spring and damping forces, which are described in the following subsections.

4.2.1. Spring force

Linear springs are used, with the spring force F_s calculated according to Hooke's law (Eq. 4.2) that relates force and strain. Here, A is the cross-sectional area of the material, l_0 the initial- or rest length, and Δl the elongation. The equation can be rewritten to a vector form as in Equation 4.3. Here the spring force between particles *i* and *j*, as felt by particle *i* is a function of their respective position vectors. The equation follows the convention that elongation of the spring results in a negative force and a positive force when the spring is compressed. For the modeling of a LEI kite, it is also assumed that tether, bridle lines, and membrane do not have any resistance against compression. Therefore an additional conditional statement is added to the code that nullifies the spring force when springs are compressed beyond their initial length l_0.

$$F_s = \frac{EA\Delta l}{l_0} = k_s\epsilon \tag{4.2}$$

$$\boldsymbol{F}_{s,i} = -k_s(|\boldsymbol{x}_{i,j}| - l_0)\hat{\boldsymbol{u}}_{i,j} \tag{4.3}$$

A graphical representation for arbitrary positions of particles *i* and *j* can be seen in Figure 4.1. As spring force is a conservative force, when particle i moves to a new position $\boldsymbol{x}_i^* = \boldsymbol{x}_i + d\boldsymbol{l}$, the energy potential stored is independent of what path is taken to reach this position. To add bending resistance when out-of-plane forces are applied, rotational springs could be added. Van der Knaap [39] found that including rotational springs generated very accurate results, but they came with a computational cost that was deemed too much. Therefore, this framework only uses three translational DOF $\boldsymbol{e}_x$, $\boldsymbol{e}_y$, and $\boldsymbol{e}_z$, and no rotational DOF are considered.

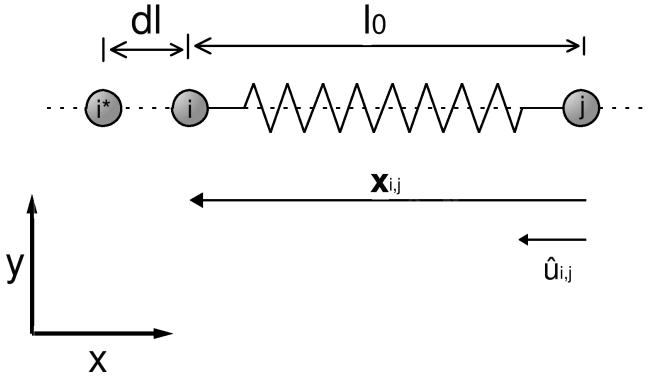


Figure 4.1: Visual representation of the variables used to calculate the spring force between particles

4.2.2. Material damping

From Equation 4.1 it can be understood that the material damping specifically acts on the strain rate or elongation velocity of the cable. Breukels [14] evaluated the effects of both external aerodynamic damping and internal material damping. He found that internal damping mostly affects the high-frequency oscillations that occur due to numerical instabilities, while external aerodynamic damping mostly affects low-frequency modes.

The damping force vector $\boldsymbol{F}_{d,i}$ between particles i and j, as felt by particle i is a function of their respective position and velocity vectors (Eq. 4.4).

$$\boldsymbol{F}_{d,i} = -c_d(\boldsymbol{v}_{i,j} \cdot \hat{\boldsymbol{u}}_{i,j})\hat{\boldsymbol{u}}_{i,j} \tag{4.4}$$

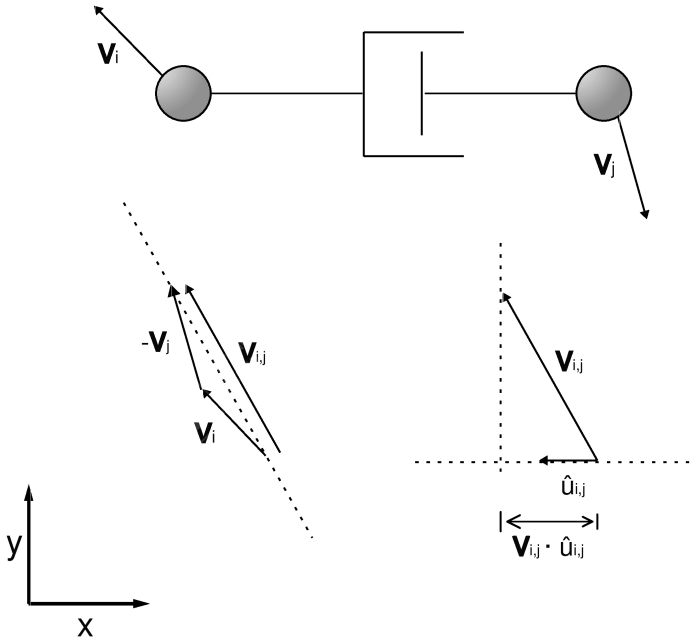


Figure 4.2: Schematic representation of damper between particles with arbitrary velocity vectors and vector calculations required to determine $\boldsymbol{F}_d$.

The orientation of the damping force vector is always opposite to that of the relative velocity between particles. As the damper elements are assumed to only have resistance in the axial direction, the relative velocity is projected on the relative position between particles. This calculation is illustrated in Figure 4.2. When the simulation reaches a steady-state solution, the velocity approaches zero and consequently, the damping force diminishes. Because the damping force doesn't affect the final shape, which can also be interpreted from Equation 2.4, the damping coefficient c_d can be used as a tuning parameter for stability and convergence speed.

4.3. External Forces

To drive the simulation of part of the test cases two external forces are required, namely gravity and tether drag. Gravity is the only force that is a function of mass and is calculated before the simulation runs. It is combined into the external force constant $\boldsymbol{F}_{ext}$ (Eq. 2.4). Again, for the steady-state solution of the simulation, the acceleration is near zero. This means that changing the mass matrix during the simulation won't affect the steady-state position and therefore the inertia can be used as a tuning parameter.

The computation of the tether drag follows the methodology that Geschiere [25] presented. First, the velocity of a tether segment $\boldsymbol{v}_t$ between particles i and j is approximated by the average velocity of the two respective particles (Eq. 4.5). Then it is used to calculate the apparent velocity $\boldsymbol{v}_{s,a}$ of the spring element with respect to the wind velocity $\boldsymbol{v}_w$ (Eq. 4.6). The effective tether area A_{eff} that faces perpendicular to the wind vector is calculated with Equation 4.7, and is visualized in Figure 4.3. Finally, the tether drag force vector $\boldsymbol{F}_d$ can be determined (Eq. 4.8). Its value is divided by two and then added to the external force vectors that work on the respective particles.

$$\boldsymbol{v}_t = \frac{\boldsymbol{v}_i + \boldsymbol{v}_j}{2} \tag{4.5}$$

$$\boldsymbol{v}_{s,a} = \boldsymbol{v}_w - \boldsymbol{v}_t \tag{4.6}$$

$$A_{\text{eff}} = d_s \cdot l_{\text{eff}} = d_t \cdot L_{\text{spring}}(1 - |\hat{\boldsymbol{u}}_{i,j} \frac{\boldsymbol{v}_{s,a}}{|\boldsymbol{v}_{s,a}|}|) \tag{4.7}$$

$$\boldsymbol{F}_d = C_D \frac{1}{2} \rho_{\text{air}} |\boldsymbol{v}_{s,a}| A_{\text{eff}} \boldsymbol{v}_{s,a} \tag{4.8}$$

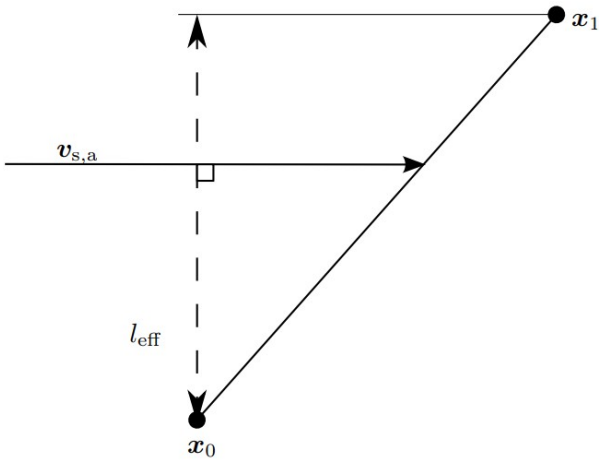


Figure 4.3: Visualization of parameters used to calculate the tether area faced perpendicular to the wind velocity vector [25].

4.4. Kinetic damping

The concept of applying kinetic damping to DR was first introduced by Barnes [8]. Kinetic damping stems from the idea that a system's kinetic energy (KE) peaks when it is on, or near, its equilibrium position. This is best illustrated with a simple undamped harmonic oscillator (Fig. 4.4), where the KE (green) peaks at the exact moment when the particle reaches its equilibrium position (indicated by blue dashed lines). The equilibrium position can be found by resetting the velocity of the system to zero at the peak of the kinetic energy. For a system with more than one spring multiple of these resets are required as it is unrealistic to assume that the springs oscillate perfectly in phase. However, by resetting the velocities of the particles, a large portion of the internal energy is instantly "dissipated" as it were. Implementing material damping in parallel would be possible, but it isn't clear if this has any advantages. Considering that the system simulates inertia, adding material damping decreases the acceleration from a standstill at resets, slowing down convergence and thus increasing runtime. Based on the results of Veenendaal and Block [76], the algorithm should outperform regular material damping when a certain amount of DOF is reached.
In his effort to develop an FSI model for ram-air kite simulations, Thedens [71] followed the implementation of the kinetic damping algorithm as presented by Barnes [8]. He found that the method accurately determines membrane shapes and internal stress, but while the shape of the structure stabilizes, force equilibrium is not reached. This highlights a limitation of the algorithm, which cannot resolve cases where the system loops by resetting at two KE peaks and bouncing back and forth between them.

The KE energy curve of the system W_{kin} is assumed to behave as a quadratic function to estimate at which moment it peaked. Other functions, e.g. cubic, could be assumed to base estimations on, but are disregarded for the sake of simplicity. Predicting where the KE would peak, based on the assumption that the curve is shaped a certain way would likely result in unstable simulations. Equation 4.9 is used to calculate W_{kin} at time t, where $\boldsymbol{v}$ is the velocity state vector of the system $[\boldsymbol{v}_1, ..., \boldsymbol{v}_n]$.

$$W_{\text{kin}}^t = (\boldsymbol{v}^t)^T \boldsymbol{M} \boldsymbol{v}^t \tag{4.9}$$

If the algorithm detects that the value of W_{kin}^t is less than W_{kin}^{t-h}, the quadratic correction factor q is calculated (Eq. 4.10). Barnes [8] and Thedens [71] both used a 2$^{\text{nd}}$-order leap-frog explicit integration scheme, also known as centered finite difference form, which uses estimates of the system's state at times $t - \frac{1}{2}h$ and $t + \frac{1}{2}h$ to find a solution for $t + h$. Here, h is the value of the timestep $h = \Delta t = t_{n+1} - t_n$ with n the simulation step number. As the implicit Euler scheme only evaluates the state of the system at integer multiples of h, the calculation and application of q were slightly modified.

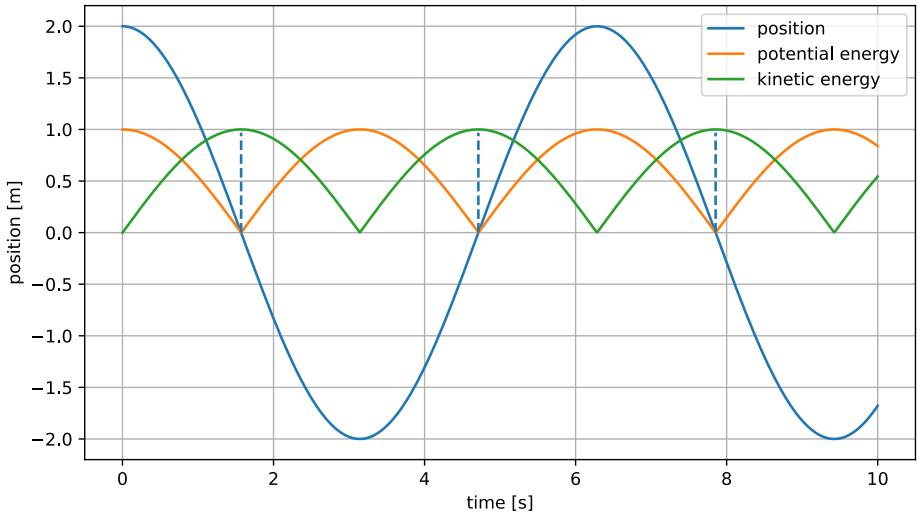


Figure 4.4: Graph illustrating normalized energies for position of a simple harmonic oscillator system

$$q = \frac{W_{\text{kin}}^{t-h} - W_{\text{kin}}^{t}}{2W_{\text{kin}}^{t-h} - W_{\text{kin}}^{t-2h} - W_{\text{kin}}^{t}} \tag{4.10}$$

The value of q is in essence the ratio of the slopes between points $t-2h$ and $t-h$ and between $t-h$ and t, illustrated by Figure 4.5. As this correction factor is calculated over a time span of $2h$, the KE peak W^* at time t^* is either positioned between $t-2h$ and $t-h$, between $t-h$ and t, or exactly at $t-h$ (Eq. 4.11). The new state of the system is determined through linear interpolation between the corresponding states. When the state of the system at t^* is calculated and updated, regular integration with stepsize h is resumed until a following reset occurs or convergence criteria are satisfied.

$$\begin{aligned} q < 0.5: &\quad t-2h < t^* < t-h \\ q = 0.5: &\quad t^* = t-h \\ q > 0.5: &\quad t-h < t^* < t \end{aligned} \tag{4.11}$$

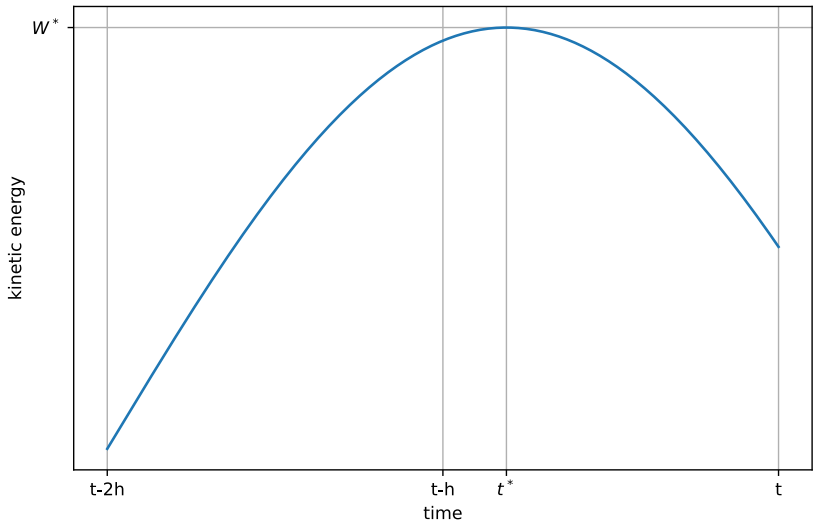


Figure 4.5: Illustrating showing how the KE peak is likely to be positioned between discrete timesteps of the simulation

5

Numerical Simulation

Nature Portfolio describes numerical simulation as "*A numerical simulation is a calculation that is run on a computer following a program that implements a mathematical model for a physical system. Numerical simulations are required to study the behavior of systems whose mathematical models are too complex to provide analytical solutions, as in most nonlinear systems.*" [67].
This chapter will formulate a set of equations that model the kite's physics to simulate its behavior over time and present the methodology used to find solutions to these equations. Even though the problem is described by a system of linearized equations, a solver is required to efficiently calculate solutions. A solver is considered to be a combination of a numerical integration scheme with either direct or iterative method, which need to be chosen based on the properties of the problem. The chapter ends with an stability analysis of the chosen integration scheme.

5.1. Solver

An Initial Value Problem (IVP) is a process that has a solution that evolves over time and that usually can be described mathematically, e.g. as a differential equation. The solution $f(t = t_0) = \boldsymbol{x}_0$ at the starting time t_0 is known, and from this initial state, solutions $f(t = t_0 + nh) = \boldsymbol{x}_t$ are calculated. The Python programming language has a library of IVP solvers that can calculate solutions $\boldsymbol{x}(t)$ for Equation 5.6, and requires little rewriting. However, it is crucial to analyze and reformulate the problem as that should lead to better performance over using a highly optimized library to brute-force solutions for a nonlinear kite and tether model. Therefore this section breaks the problem down to make a choice in numerical integration scheme, used to approximate solutions to the problem over time, and numerical technique, that calculates the values of these solutions.

Griffiths and Higham describe schemes used for solving IVPs that fall into one of two categories: one-step methods and multistep methods [27]. These families can again be sub-categorized into implicit and explicit schemes. Explicit schemes approximate the solution at the next timestep based on the solution of the current timestep: $f(x_{n+1}, t_{n+1}) = f(x_n, t_n)$. Implicit schemes also use an approximation of the future state(s) to find the solution at the next timestep: $f(x_{n+1}, t_{n+1}) = f(x_n, t_n, x_{n+1}, t_{n+1})$. This requires extra computational steps and makes their implementation less straightforward than explicit schemes. However, previous research on particle systems argued the necessity and proved the advantage of using implicit methods for time integration of a PSM [7, 65]. Implicit methods come with a higher computational cost per timestep, but their improved stability with respect to explicit methods makes larger timesteps possible, resulting in overall faster simulation. Explicit methods have their timestep *h* bounded inversely proportional to the natural frequency ω of the particles, which makes them unfeasible to use for cables that have very high material stiffness and thus very high frequencies, as shown in Equations 5.1 and 5.2 [65]. Here, α is a coefficient depending on the scheme that is considered. Discretizing the cables with an increasing number of particles n lowers h, which can be seen by plugging Equation 5.2 into Equation 5.1. Implicit schemes are therefore not considered as a feasible option to utilize in the framework.

$$h < \frac{\alpha}{\omega} \tag{5.1}$$

$$\omega(n) = \sqrt{\frac{nE}{l_0 \rho}} \tag{5.2}$$

The equations that model the problem remain nonlinear when using an implicit scheme. To find forces at a future state a linear approximation is used, consequently resulting in a linear set of equations that accelerates calculations. However, linearization generally is paired with a loss in accuracy. As the accuracy of the transient phase is not of interest and the steady-state solution isn't affected, linearization is only considered beneficial. To solve a linear set of equations expressed as a matrix-vector product, either a direct method involving matrix inversion or an iterative method can be employed. The choice depends on the properties of the matrix.

5.1.1. Integration scheme

Certain differential equations can be classified as stiff. There is no set of precise conditions to classify a differential equation as such, but there are general descriptions.

- A problem that, for certain numerical schemes, has its stepsize limited by stability issues rather than being determined by a required accuracy.
- A problem with a sought solution that varies on a large timescale, while intermediate solutions vary on lower timescales.
- A problem where the eigenvalues of the Jacobian(s) of the force vector have a large ratio between the lowest and highest value.

Generally, implicit schemes are suited for finding solutions to stiff problems, in particular the family of backward differentiation formula (BDF). They belong to the family of linear multistep methods and were developed specifically for solving stiff equations by Curtiss and Hirschfelder [19]. As the set of equations that describe the particles' motion has been identified as stiff [7], a BDF will be used.

Implicit Euler

The first order BDF is also known as the implicit or backward Euler integration scheme, which uses a single future state in its approximation. The current state is estimated from the perspective of the future state, i.e. looking backward. The integration scheme is found by rewriting the future state as a first two exponents of its Taylor series (Eq. 5.3). By formulating this equation in discrete terms (Eq. 5.4) and rearranging for x_{n+1}, the integration scheme is found (Eq. 5.5).

$$x_n \approx x(t_{n+1} - \Delta t) = x(t_{n+1}) - \Delta t \frac{dx(t_{n+1})}{dt} + O(\Delta t^2) \tag{5.3}$$

$$x_n = x_{n+1} - h\dot{x}_{n+1} \tag{5.4}$$

$$\begin{aligned} x_{n+1} &= x_n + h\dot{x}_{n+1} \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + h\boldsymbol{v}_{n+1} \end{aligned} \tag{5.5}$$

From this derivation, it can also be seen that the local truncation error $O(h^{p+1})$ of the scheme is of order two. A visual example of this error can be seen in Figure 5.1. The scheme approximates a solution by estimating the derivative of the next time step and using that gradient to find a solution. Choi and Ko [17] used a second-order BDF for their PSM-based cloth simulation. Higher-order methods are more accurate but come with an increase in computational cost and difficulty in implementation. As accuracy during the transient phase is not required, higher-order methods are disregarded and the Implicit Euler scheme will be used for numerical integration.

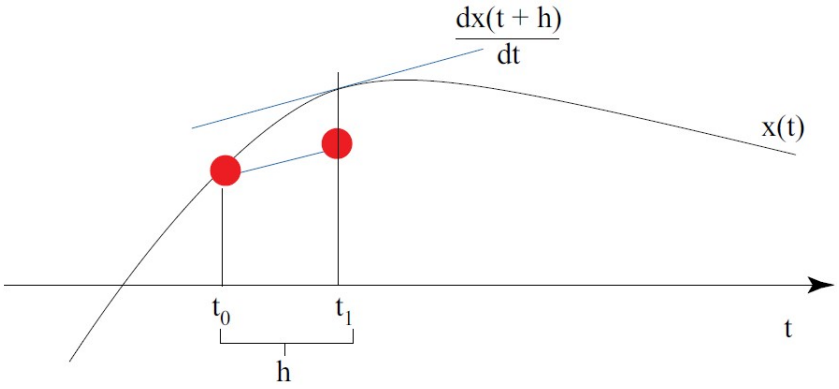


Figure 5.1: Visual example of a linear estimate of the state at the next time step using implicit Euler [68].

5.1.2. Linear system of equations

To find a value for $\boldsymbol{v}_{n+1}$ and thus find $\boldsymbol{x}_{n+1}$, Van der Knaap [39] presented the following derivation. Newton's second law of motion describes the change in motion that a body with mass m experiences when force is applied to it. Equation 2.4 is rewritten as Equation 5.6 to isolate $\ddot{\boldsymbol{x}}(t)$. Here $\boldsymbol{x}$ is the position state vector, describing the position of each particle in three dimensions: $[\boldsymbol{x}_1, ..., \boldsymbol{x}_n]$. $\boldsymbol{M}$ is a square and sparse matrix of size $3n$, containing the masses of each particle in each dimension on the diagonal: $[m_{1,x},\ m_{1,y},\ m_{1,z}, ..., m_{n,x},\ m_{n,y},\ m_{n,z}]$. As mass can be used as a tuning parameter, it doesn't need to be kept constant or be the same value in each direction. For the sake of simplicity, the masses of a particle are kept constant with the same value in all directions. Lastly, the force vector $\boldsymbol{F}$ holds the sum of internal and external force acting on each particle for all dimensions of motion. These are the forces that were covered in Chapter 4.

$$\ddot{\boldsymbol{x}}(t) = \boldsymbol{M}^{-1}\Sigma\boldsymbol{F}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t) \tag{5.6}$$

This compactly written second-order, non-homogeneous, ordinary differential equation (ODE) can be rewritten to a set of coupled first-order ODEs, shown in Equation 5.7.

$$\begin{pmatrix} \dot{\boldsymbol{x}} \\ \ddot{\boldsymbol{x}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{v} \end{pmatrix} \frac{d}{dt} = \begin{pmatrix} \boldsymbol{v}(t) \\ \boldsymbol{M}^{-1}\boldsymbol{F}(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t) \end{pmatrix} \tag{5.7}$$

The derivatives from Equation 5.7 are approximated by the implicit Euler scheme in discrete form as Equation 5.8. Using these approximations, Equation 5.7 is rewritten to its discrete form, as shown in Equation 5.9.

$$\begin{aligned} \frac{d\boldsymbol{x}}{dt} &\approx \frac{\Delta\boldsymbol{x}}{h} = \frac{\boldsymbol{x}_{n+1} - \boldsymbol{x}_n}{h} \\ \frac{d\boldsymbol{v}}{dt} &\approx \frac{\Delta\boldsymbol{v}}{h} = \frac{\boldsymbol{v}_{n+1} - \boldsymbol{v}_n}{h} \end{aligned} \tag{5.8}$$

$$\begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{v} \end{pmatrix} = h \begin{pmatrix} \boldsymbol{v}_n \\ \boldsymbol{M}^{-1}\boldsymbol{F}(\boldsymbol{x}_{n+1}, \boldsymbol{v}_{n+1}) \end{pmatrix} \tag{5.9}$$

Looking at the bottom half of the equation, the only nonlinear term is $\boldsymbol{F}(\boldsymbol{x}_{n+1}, \boldsymbol{v}_{n+1},)$. To find a value for this term a linear approximation is used, i.e. by approximating the term with the first two exponents of its Taylor series (Eq. 5.10).

$$\boldsymbol{F}(\boldsymbol{x}_{n+1}, \boldsymbol{v}_{n+1}) = \boldsymbol{F}(\boldsymbol{x}_n + \Delta\boldsymbol{x}, \boldsymbol{v}_n + \Delta\boldsymbol{v}) = \boldsymbol{F}_n + \frac{\partial\boldsymbol{F}}{\partial\boldsymbol{x}}\Delta\boldsymbol{x} + \frac{\partial\boldsymbol{F}}{\partial\boldsymbol{v}}\Delta\boldsymbol{v} \tag{5.10}$$

The partial derivatives of the force vector with respect to the position or velocity vectors are also known as Jacobians. Their respective symbolic notations are $\boldsymbol{J}_x$ and $\boldsymbol{J}_v$. With this approximation of F the bottom half of Equation 5.9 can now be written out to a new expression sorted for $\Delta\boldsymbol{v}$ (Eq. 5.11).

$$(\boldsymbol{I} - h\boldsymbol{M}^{-1}\boldsymbol{J}_v - h^2\boldsymbol{M}^{-1}\boldsymbol{J}_x)\Delta\boldsymbol{v} = h\boldsymbol{M}^{-1}(\boldsymbol{F}_n + h\boldsymbol{J}_x\boldsymbol{v}_n) \tag{5.11}$$

By evaluating the force vector and Jacobians at time n the only unknown left is $\Delta \boldsymbol{v}$, from which $\boldsymbol{v}_{n+1}$ can be calculated, as $\boldsymbol{v}_{n+1} = \Delta \boldsymbol{v} + \boldsymbol{v}_n$.

5.1.3. Jacobians

Jacobians are mathematical objects which can be used for sensitivity analysis in multiple disciplines, e.g. in economics, since they give a measure of change of one vector with respect to change in another vector. In this context, they can in a sense be interpreted as a geometrical stiffness. The analytically worked out solution provides the problem with the sensitivity of nodal forces with respect to their positions which is the geometry of the model.

Position Jacobian

With use of vector Equation 6 the equation for the position Jacobian can be derived. Between the internal forces, only the spring force is a function of position. The derivation is done for the change in force of a spring between particles *i* and *j* by moving particle *i*, as felt by particle *i*. A shortened version of the derivation as presented by Macklin [45] can be seen in Equation 5.12. This can be simplified further to Equation 5.13, by using the product rule and setting $\hat{\boldsymbol{u}}_{i,j}\hat{\boldsymbol{u}}_{i,j}^T = \boldsymbol{T}$ as suggested by E. van der Knaap [39].

$$\begin{aligned}\frac{\partial \boldsymbol{F}_{s,i}}{\partial \boldsymbol{x}_{i,j}} &= -k_s \left[(|\boldsymbol{x}_{i,j}| - l_0) \frac{\partial \hat{\boldsymbol{u}}_{i,j}}{\partial \boldsymbol{x}_{i,j}} + \frac{(\partial |\boldsymbol{x}_{i,j}| - l_0)}{\partial \boldsymbol{x}_{i,j}} \hat{\boldsymbol{u}}_{i,j} \right] \\ &= -k_s \left[(|\boldsymbol{x}_{i,j}| - l_0) \frac{\boldsymbol{I} - \hat{\boldsymbol{u}}_{i,j}\hat{\boldsymbol{u}}_{i,j}^T}{|\boldsymbol{x}_{i,j}|} + \hat{\boldsymbol{u}}_{i,j}\hat{\boldsymbol{u}}_{i,j}^T \right]\end{aligned} \tag{5.12}$$

$$\boldsymbol{J}_x = -k_s \left[(1 - \frac{l_0}{|\boldsymbol{x}_{i,j}|})(\boldsymbol{I} - \boldsymbol{T}) + \boldsymbol{T} \right] \tag{5.13}$$

From Figure 4.1 it can be seen that if the perspective between particles *j* and *i* is reversed, the change in force felt by particle *j* by moving particle *j* results in the same Jacobian as derived above. When the opposite particle is moved, it results in a change of force that is equal in magnitude, but opposite in direction, i.e. $\boldsymbol{J}_{x,i,i} = \boldsymbol{J}_{x,j,j} = -\boldsymbol{J}_{x,i,j} = -\boldsymbol{J}_{x,j,i}$. Conveniently, this means only one sub-Jacobian has to be calculated per spring. The sparse Jacobian matrix that contains the results for each spring in the system can be constructed by placing these sub-Jacobians at their respective indices. A typical resulting sparse Jacobian matrix can be seen in Equation 5.14, for a tether where particles are connected to only direct neighboring particles. Here, $J_{x,i,j}$ denotes the Jacobian for the spring between particles *i* and *j*.

$$\begin{pmatrix} J_{x,1,2} & -J_{x,1,2} & 0 & \cdots & 0 \\ -J_{x,1,2} & J_{x,1,2} + J_{x,2,3} & & & \vdots \\ 0 & & \ddots & & 0 \\ \vdots & & & J_{x,n-2,n-1} + J_{x,n-1,n} & -J_{x,n-1,n} \\ 0 & \cdots & 0 & -J_{x,n-1,n} & J_{x,n-1,n} \end{pmatrix} \tag{5.14}$$

Velocity Jacobian

The derivation of the velocity Jacobian of the internal forces is more straightforward since only the damping force is a function of velocity. The partial derivative of Equation 4.4 with respect to velocity results in Equation 5.15, which subsequently can be simplified into Equation 5.16.

$$\frac{\partial \boldsymbol{F}_{d,i}}{\partial \boldsymbol{v}_{i,j}} = -c_d(\hat{\boldsymbol{u}}_{i,j}^T \cdot \hat{\boldsymbol{u}}_{i,j}) \tag{5.15}$$

$$\boldsymbol{J}_v = -c_d \boldsymbol{I} \tag{5.16}$$

5.1.4. Iterative method

Section 5.1.2 ended with the derivation of Equation 5.11 from the system of equations. With every variable now known, except for $\Delta \boldsymbol{v}$, the linear system can be reduced to the form shown in Equation 5.17. Here, the matrix $\boldsymbol{A}$ is a known, sparse, square, and positive-definite matrix. Vector $\boldsymbol{b}$ is also known. Vector $\boldsymbol{x}$ represents $\Delta \boldsymbol{v}$, which is the unknown to solve the system for.

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{5.17}$$

The iterative biconjugate gradient stabilized method (BiCGSTAB) was developed by Vorst [79] and is part of the family of biconjugate gradient (BiCG) algorithms. This family of algorithms is used for solving linear systems of equations. He found that the BiCGSTAB method has more stable convergence behavior, while also converging faster in most situations than regular conjugate squared methods (CGS). Therefore the BiCGSTAB is used to find numerical solutions for $\Delta \boldsymbol{v}$.

To save development time, the BiCGSTAB module from the *Scipy* Python library was used. The exact algorithm can be found in the online documentation [70] and is also added to Appendix A.2. The method is executed without preconditioning. The addition of preconditioning could lead to lower runtime [69] if in future research the BiCGSTAB algorithm is shown to have a relatively large share in the overall computational cost.

5.2. Stability analysis

A numerical integration scheme is called stable if the global error doesn't increase over time. Stability of a scheme is an important factor in ensuring convergence. As a test, Equation 5.18 is used to analyze the stability of the BE integration scheme. It is an ODE known as Dahlquist's equation, with initial condition $y(0) = y_0$ and eigenvalue $\lambda \in \mathbb{C}$. The analytical solution for this problem is shown in Equation 5.19. From this solution, it can be seen that the real part a of λ should be negative for the solution to be bounded when t approaches infinity.

$$\dot{y} = \lambda y(t) \tag{5.18}$$

$$y(t) = y_0 e^{\lambda t} = y_0 e^{at}(cos(bt) - i\ sin(bt)) \tag{5.19}$$

The derivation shown in Equation 5.20 starts by plugging Equation 5.18 into the numerical scheme (Eq. 5.3). By rearranging for the term y_{n+1} and substituting the initial condition, a recurrence relation can be determined.

$$\begin{aligned} y_{n+1} &= y_n + h\lambda y_{n+1} \\ y_{n+1} &= \frac{1}{1-h\lambda} y_n \\ y_k &= \frac{1}{1-h\lambda}^{\ k}\ y_0 \end{aligned} \tag{5.20}$$

From this relation, the requirements for stability can be deduced. The timestep h is always greater than 0. When combined with the condition that the real part of λ is negative, it follows that $|\frac{1}{1+h\lambda}| \leq 1$. This leads to the conclusion that with any real λ the implicit BE integration scheme is unconditionally stable for any value of h. However, it is important to note that stable results are not the same as accurate results.

5.2.1. Matrix conditioning

Backward stable integration schemes such as BE are expected to find accurate results for systems in the form of Equation 5.17, when matrix $\boldsymbol{A}$ is well-conditioned. The condition number k is a measure of sensitivity, giving insight into how much the found solution $\boldsymbol{x}$ varies for changes in the input. It is defined in Equation 5.21. A certain threshold value is used to define if the problem is well-conditioned or ill-conditioned. Numbers in literature vary but are usually in an order of magnitude of $1e3\ 1000\ 1000$ As a rule of thumb: the larger the condition number of the matrix, the less well-conditioned the problem.

$$k(\boldsymbol{A}) = |\boldsymbol{A}|\ |\boldsymbol{A}^{-1}| \tag{5.21}$$

For a given geometry, timestep, and mass matrix, it can be seen from Equation 5.11 that matrix $\boldsymbol{A}$ is a function of k_s and c_d.

For the test case where the tether is deflected by a perpendicular flow, a change of the spring coefficient from 1000 N/m to a realistic material value of 6720 N/m for a tether of 10 m resulted in a change of the condition number from 32 to ~ 2000. This result indicates that the conditioning of the problem could be (one of the) causes that realistic values of spring stiffness cause problems with accurate simulation.

6 Verification

Four intermediate tests were used to verify the correct implementation of individual components before running the more complex validation tests. The first three tests focus on isolating imposed load and internal spring and damping forces as much as possible. Trivial cases are used as they allow for comparison to analytical solutions, which simplifies analysis. They also serve to provide insight into the numerical effects of simulation with an integration scheme. These test cases are in essence one-dimensional problems that are simulated in three dimensions. The results showed that the framework didn't introduce any errors by doing this. The fourth verification case is used as an indication that the framework is capable of handling a more complex mesh in 3D space. No adjustments were made to the framework to run these cases.

6.1. External load

The first case imposes a constant external load, namely gravity. Internal spring- and damping forces are excluded by setting their respective coefficients to zero. The position of a falling body can be derived by integrating the gravitational acceleration twice with respect to time and using the particle's initial position $\boldsymbol{x}_0$ and initial velocity $\boldsymbol{v}_0$ as boundary conditions (Eq. 6.1).

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \boldsymbol{v}_0 t - 0.5\boldsymbol{g}t^2 \tag{6.1}$$

The initial position and velocity vector components are all set to zero. The framework requires a minimum of two particles with one connection to run. Therefore, one particle was anchored at the origin, and the other particle was free to accelerate. The result of three simulations and the analytical solution (green) can be seen in Figure 6.1, with gravity accelerating in the $-\boldsymbol{e}_z$ direction. No movement of the particle in either the $\boldsymbol{e}_x$ or $\boldsymbol{e}_y$ direction occurred. The simulations were run with time steps of 0.01 s (dashed black), 0.1 s (blue), and 1 s (orange). With an increase in the value of h, a clear decrease in precision can be observed. This is an expected effect for the BE scheme, where the local error is proportional to the square of h. The error also shows growth over time, as the local error occurs every iteration. This growth over time can most clearly be seen by comparing the difference between the simulation with a time step of 1 s and the analytical solution.

Figure 6.2 visualizes both observations more clearly. The effect of local error can be visualized by taking the absolute error after one iteration for each simulation and normalizing, as shown in the graph on the left. The measured errors exactly match the curve of h squared. The graph on the right shows that the absolute error of the simulation with a time step of 1 s grows linearly.

6.2. Spring force

To test the spring force, and in the next section the internal damping, a system that is well-known is simulated, namely the simple harmonic oscillator. This system consists of a particle with mass m that is connected by a spring to an anchored particle. No external loads are imposed on the particle. With the rest length of the spring set to 0, releasing the particle from any nonzero distance results in

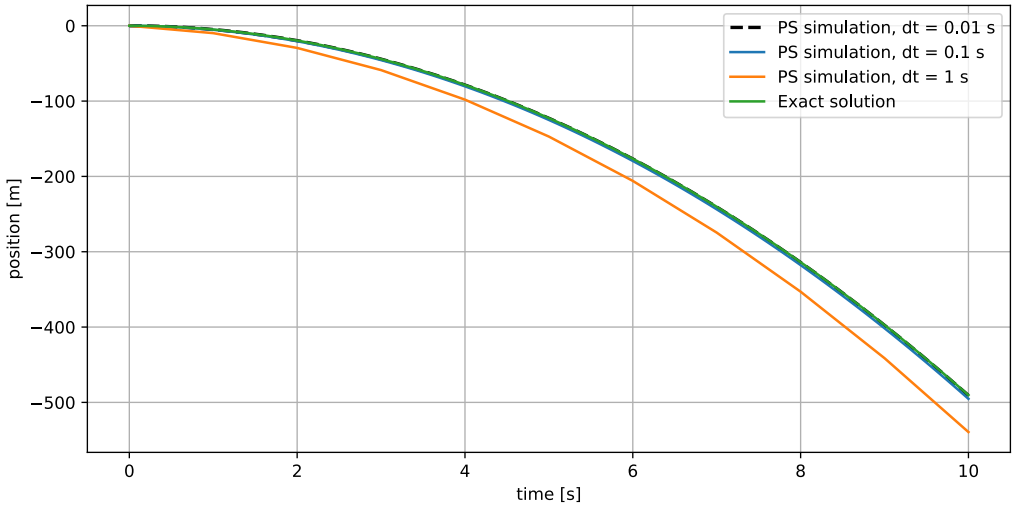


Figure 6.1: Analytical solution and particle position over time for varying values of time step

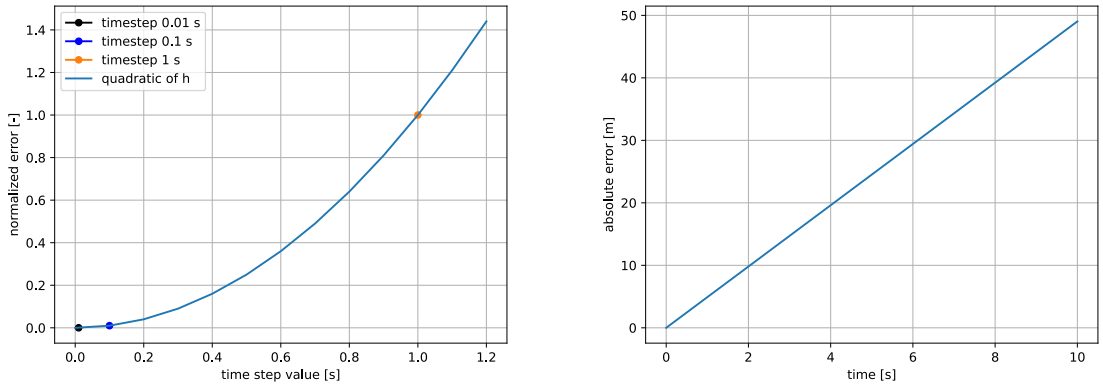


Figure 6.2: Normalized errors after one simulation step against the curve of h^2 (L) and absolute error growth over time for a simulation with a time step of 1 s (R).

the oscillatory motion of the particle around its equilibrium position. The damping coefficient is set to zero, so there is no energy dissipated from the system. The derivations of [18, 72] are followed to find the analytical solution for this system. It starts with a second-order homogeneous differential equation, again based on Newton's second law (Eq. 6.2). The natural angular frequency ω_0 is defined in Equation 6.3. A solution over time in the form of Equation 6.4 is proposed. The values of coefficients *A* and *B* can be found by calculating the first derivative of this solution (Eq. 6.5) and satisfying the boundary conditions in Equation 6.6. Solving for these boundary conditions leads to the general solution (Eq. 6.7). As the initial velocity for the particles is set to 0, only the cosine term remains on the right-hand side of the equation.

$$\ddot{\boldsymbol{x}} + \omega_0^2 \boldsymbol{x} = 0 \tag{6.2}$$

$$\omega_0 \equiv \sqrt{\frac{k}{m}} \tag{6.3}$$

$$\boldsymbol{x}(t) = A\,\mathsf{cos}(\omega_0 t) + B\,\mathsf{sin}(\omega_0 t) \tag{6.4}$$

$$\boldsymbol{v}(t) \equiv \frac{d\boldsymbol{x}(t)}{dt} = -\omega_0 A\,\mathsf{sin}(\omega_0 t) + \omega_0 B\,\mathsf{cos}(\omega_0 t) \tag{6.5}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0, \quad \boldsymbol{v}(t_0) = \boldsymbol{v}_0 \tag{6.6}$$

$$\boldsymbol{x}(t) = \boldsymbol{x}_0\,\mathsf{cos}(\omega_0 t) + \frac{\boldsymbol{v}_0}{\omega_0}\,\mathsf{sin}(\omega_0 t) \tag{6.7}$$

To simulate this system, the anchored particle is fixed at the origin, and the second particle is released from $\boldsymbol{x}_0 = [1,\ 0,\ 0]$. This results in oscillations purely along the $\boldsymbol{e}_x$ axis. A simulation of this system (blue) with parameters set to h = 1×10^{-3} s, m = 1 kg, and k_s = 2×10^5 N/m can be seen together with the analytical solution (green) in Figure 6.3. Remarkable is the amount of energy that seems to be dissipated from the simulated system, despite the absence of damping. This is an effect known as numerical damping and can be explained as follows.

First, Equation 6.2 is rewritten to a system of first-order ODEs (Eq. 6.8). Then, by using equation 5.8 this system is rewritten to its discrete BE approximation (Eq. 6.9). The eigenvalues of the matrix on the right side of the equation are $\lambda_{1,2} = 1 \pm ih\omega$. Assuming the initial vector is exactly the eigenvector belonging to the eigenvalue $\lambda = 1 + ih\omega$, and by sorting terms the recurrence relation is found (Eq. 6.10).

$$\begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix} \frac{d}{dt} = \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix} \tag{6.8}$$

$$\begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix}_{n+1} - \begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix}_{n} = h \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix}_{n+1} \tag{6.9}$$

$$\begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix}_{n} = (1 - ih\omega)^{-n} \begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix}_{0} \tag{6.10}$$

So far, this is relatively similar to the derivation in Section 5.2. By examining the coefficient in more detail, some characteristic properties of the BE Euler scheme can be revealed. A phase shift is found when comparing the phase of the coefficient against that of the analytical solution (Eq. 6.11).

$$\mathsf{phase}((1 - ih\omega)^{-n}) = -n * \mathsf{phase}(1 - ih\omega) = -n * \mathsf{arctan}(-h\omega) = n * \mathsf{arctan}(h\omega) < nh\omega = t\omega \tag{6.11}$$

The coefficient can be rewritten to a product of two exponents, where one exponent contains the imaginary terms with *i* and the other purely real numbers (Eq. 6.12). This result can be found by approximating the exponent by a Taylor series around the point $ih\omega = 0$. A more detailed derivation can be viewed in Appendix B.1. The exponent including i reveals that a frequency reduction occurs,

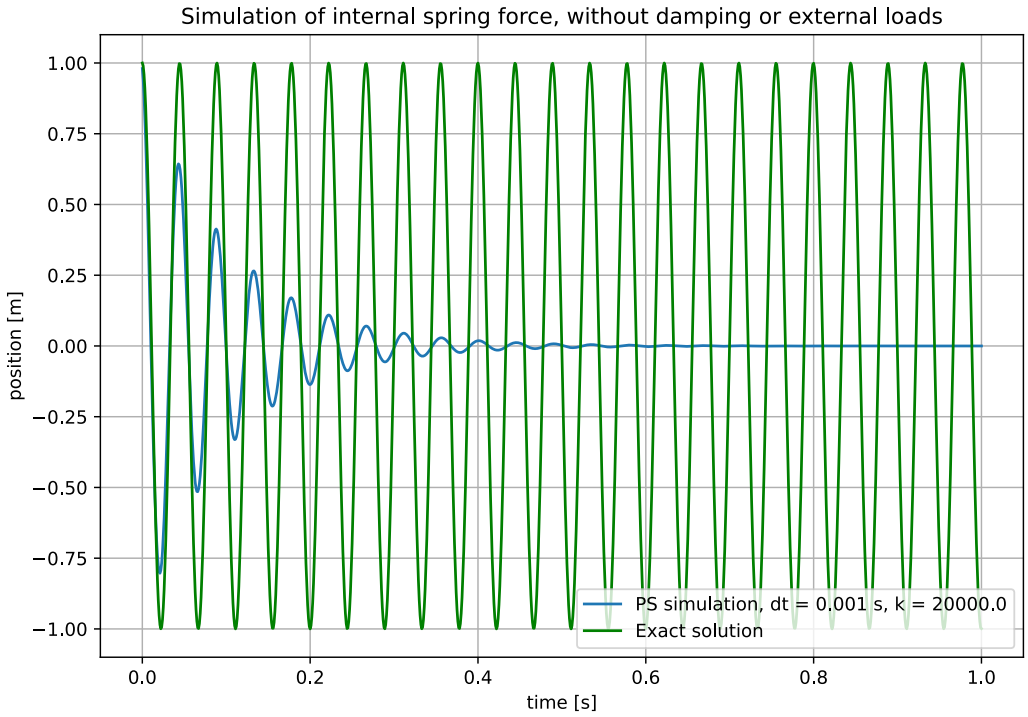


Figure 6.3: Plot of exact solution of harmonic oscillator (green) and simulation (orange)

consistent with the earlier observation. The exponent with real number reveals the effect of numerical damping, which varies with the size of the time step h. Both effects can be plotted as functions of t as an estimate of the phase shift over time and the decay rate that Implicit Euler introduces.

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_n = e^{-it\omega(1-\frac{1}{3}\omega^2)} e^{-\frac{1}{2}*th\omega} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_0 \tag{6.12}$$

The system was simulated with the same values for h and k_s as before (green), together with the estimated decay rate (red), and a corrected simulation (dotted black) (Fig. 6.4). The estimated decay decreases in accuracy over time. The increase of the phase shift over time can also be seen more clearly.

6.3. Internal damping

The same system, a harmonic oscillator, is used to check the correct implementation of internal damping. The addition of a damping term γ to the differential equation (Eq. 6.13) makes the derivation of its analytical solution slightly more complicated. The analytical solution over time varies, depending on whether the system is under-, over-, or critically damped (Eq. 6.15) [18]. The steady-state solution can be calculated by setting $\ddot{x}$ and $\dot{x}$ to zero. Without imposed forces, the steady-state solution of x is 0, which the methods should converge to. As an alternative to the analytical solution, the differential equation can be split into a system of coupled first-order ODEs (Eq. 6.16).

$$\ddot{\boldsymbol{x}} + \gamma\dot{\boldsymbol{x}} + \omega^2\boldsymbol{x} = 0 \tag{6.13}$$

$$\gamma \equiv \frac{c_d}{2m} \tag{6.14}$$

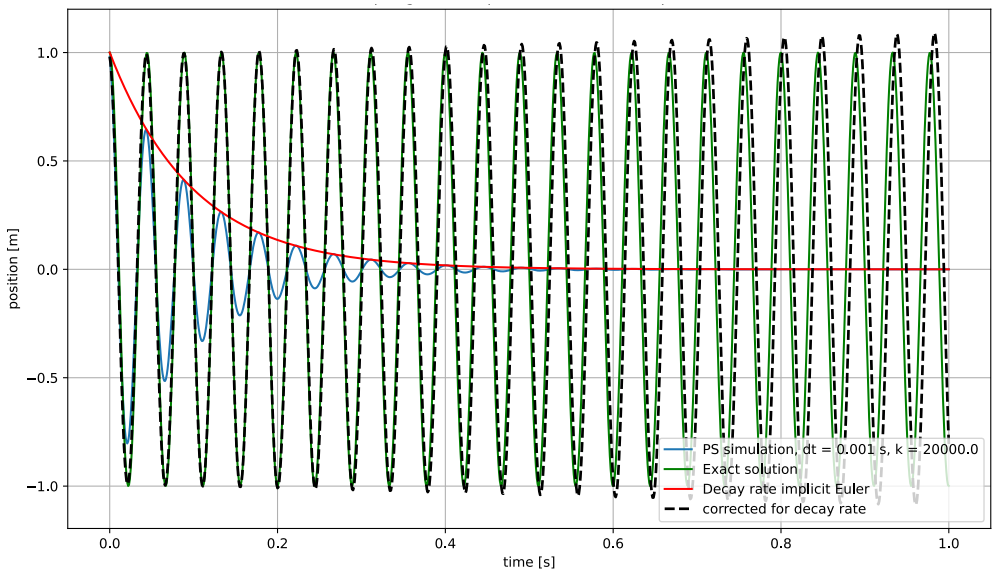


Figure 6.4: Simulation corrected for decay rate

$$\begin{aligned} \text{Underdamped}: &\quad \omega_0^2 > \gamma^2 \\ \text{Critically damped}: &\quad \omega_0^2 = \gamma^2 \\ \text{Overdamped}: &\quad \omega_0^2 < \gamma^2 \end{aligned} \tag{6.15}$$

$$\begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix} \frac{d}{dt} = \begin{pmatrix} 0 & 1 \\ -\omega^2 & -\gamma \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \dot{\boldsymbol{x}} \end{pmatrix} \tag{6.16}$$

This system will be solved with a built-in function, $solve\_ivp()$, from the Python *Scipy* library. This is meant to serve as a demonstration of the computational cost of this methodology to find solutions. This test also introduces kinetic damping, since the algorithm doesn't apply to previous cases. The algorithm has been split into two versions, one version where the system's state is recalculated based on the value of q, whereas the other version disregards recalculation. These versions will be denoted by with- or without q-correction, respectively.

The results of simulating a critically damped system can be seen in Figure 6.5. Parameter values were set to k_s = 1×10^3 N/m , c_d = 63.2 N s/m, m = 1 kg and h = 0.1 s. The value of h was set lower than 1 s, since the simulation otherwise converges in a single step, which would not allow differences to be distinguished.

The classic PS with viscous damping (blue) and PS with kinetic damping without q-correction algorithm (red) converged before the kinetic damping with q-correction algorithm (yellow). The kinetic damping algorithm is implemented such that the user can decide whether to use the correction factor q to calculate the new geometry $\boldsymbol{x}^*$ at reset. If q is not used, the algorithm sets the velocity to zero without correcting the geometry. As the BE scheme introduces numerical damping, the peak of the kinetic energy is not aligned with the equilibrium position. The peak is shifted to an earlier moment, which can be seen in the graph where the yellow and red lines split. The algorithm takes effect at this time step before the equilibrium is reached. Because q-correction assumes the equilibrium position has already been reached, it ends up placing the system further from the steady state, which introduces more energy into the system. This is the reason that the kinetic damping with q-correction takes longer to converge than the other methods. It was found that not using q-correction and only setting velocity to 0 led to better results. Each method did eventually converge to the correct steady-state solution.

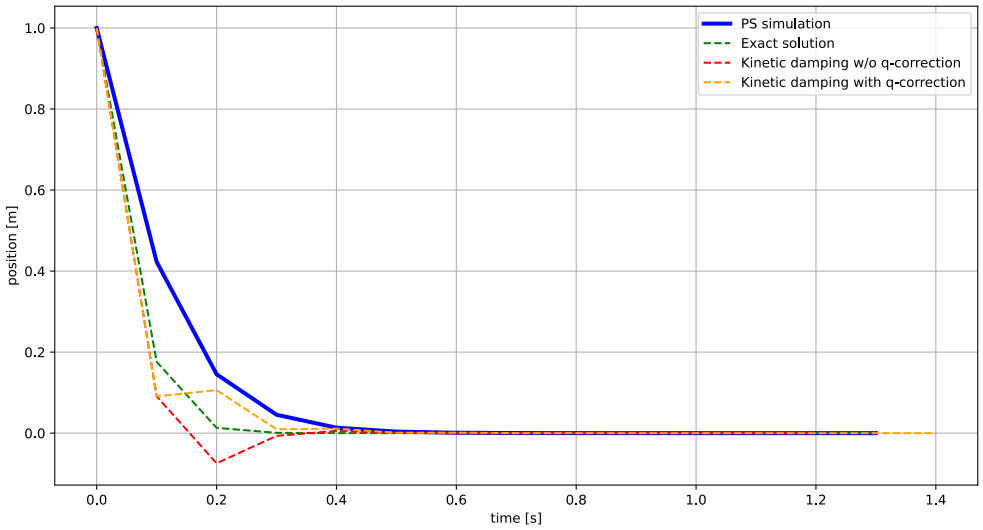


Figure 6.5: Simulation of a critically damped harmonic oscillator (c_d = $63.2\,\mathrm{N\,s/m}$), without external load

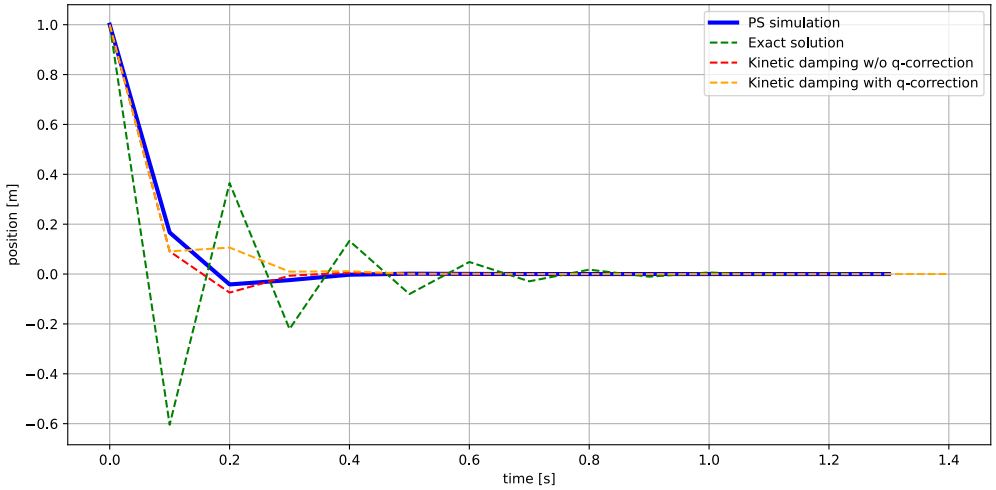


Figure 6.6: Simulation of an underdamped harmonic oscillator (c_d = $10\,\mathrm{N\,s/m}$), without external load

The simulation was repeated for both an underdamped and overdamped system. The results can be seen in Figures 6.6 and 6.7, respectively. The IVP function (green) was set to solve up to the time step that the classic PS converged on. The IVP function took 9.7×10^{-3} to 1.23×10^{-2} s to run while the classic PS took 5×10^{-3} to 7×10^{-3} s. This is a reduction factor of 1.4 to 2.5 for a very basic system with only 6 DOF and sub-optimal tuning for the PS. Overdamping slows the convergence of classic PS, while the simulations with kinetic damping are unaffected by varying the viscous damping parameter. This reduces the amount of tuning parameters, which is considered advantageous since it either requires less user tuning or simplifies the development of an automated tuning algorithm.

6.4. Complex meshing

A final case was simulated, similar to the test that Veenendaal and Block [76] used to compare the performance of the distinct methods they identified. Here, its use is mostly to confirm that the implemented framework is capable of handling nodes that have connections other than with their two direct neighbors, as in the previous benchmarks. The test case is to find the shape of a self-stressed network (no imposed loads), where the boundary conditions are displaced to a slanted line. The initial problem (L) and the found solution (R) can be seen in Figure 6.8.

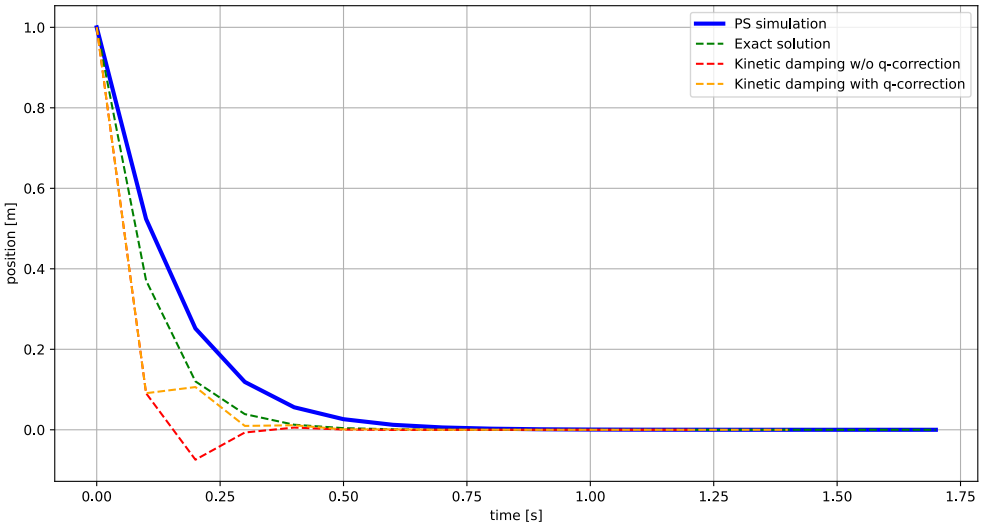


Figure 6.7: Simulation of an overdamped harmonic oscillator (c_d = $100\,\mathrm{N\,s/m}$), without external load

The result was visually inspected, as no exact data was provided on the final shape. The simulated membrane appears to converge to the expected shape, which is the case for varying mesh densities. No instabilities or other unexpected problems were encountered during test simulations. This concludes the verification process, as the implemented framework behaves as expected.

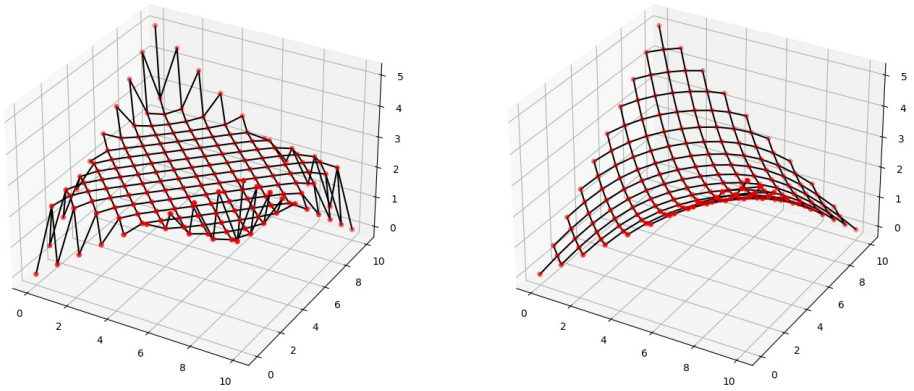


Figure 6.8: Flat net-structure with displaced boundary conditions (L) and found shape (R)

7

Validation

Four main test cases are used to validate that the framework can simulate tether behavior and membrane deformation accurately. The first three cases target different modes of tether and bridle line loading. Namely, longitudinal loading, perpendicular loading, and nonlinear loading. The final case is meant to serve as a substitute for kite deformation, as there is little to no experimental data available that can be used as a comparison. The material continuum within either the tether or membrane is represented by a PSM using a finite mesh of connected springs and dampers. It is expected that increasing mesh density results in increasingly precise outcomes.

7.1. Modeling

Figure 7.1 illustrates how discrete elements of tether and membrane continuum are translated into a PSM. The tether is discretized into segments of length L, where it's Young's modulus E, diameter d_t, and length L are used to calculate the spring stiffness of the PS spring element. Equation 7.1 shows the relation between these parameters. Each particle is also assigned half of the mass of the tether segment.

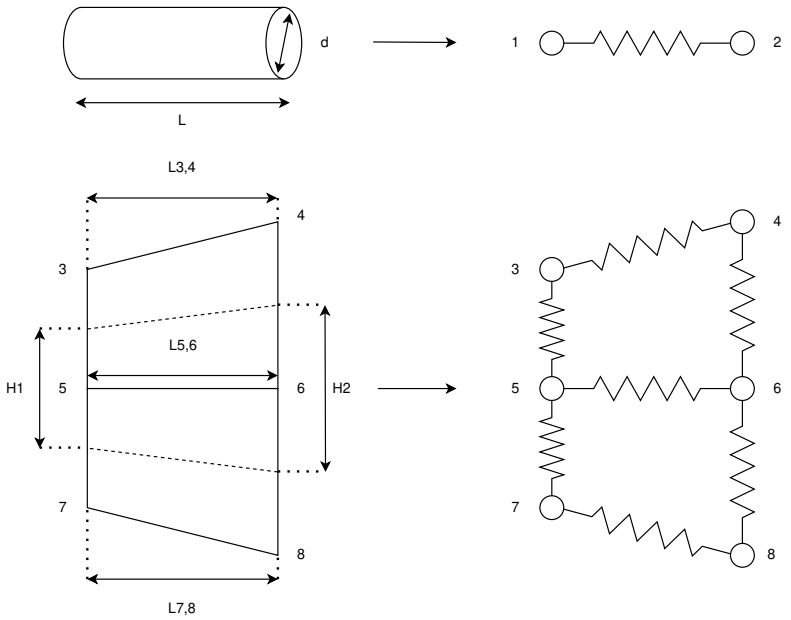


Figure 7.1: Translation of rope and membrane continuum elements into particle system representation.

$$k_{\text{tether}} = \frac{E_{\text{tether}} A}{L} = \frac{E_{\text{tether}} \frac{1}{4} \pi d_t^2}{L} \tag{7.1}$$

For a membrane of a certain size and thickness w, the choice was made to discretize the material into quadrilateral elements. To calculate spring stiffness for the spring between particles 5 and 6, the average height and length of the quadrilateral membrane area around the spring are calculated (Eq. 7.2).
The Poisson's ratio ν is set to 0 in Hencky's problem since strain in one of the primary directions of the utilized mesh doesn't affect strain in a perpendicular direction. Also, no masses are calculated from the quadrilateral elements as gravity is disregarded in this validation case.

$$k_{\text{membrane}} = \frac{E_{\text{membrane}} A}{L} = \frac{E_{\text{membrane}} w \frac{1}{2} (H_1 + H_2)}{\frac{1}{4} (\frac{1}{4} L_{3,4} + \frac{1}{2} L_{5,6} + \frac{1}{4} L_{7,8})} \tag{7.2}$$

7.2. Validation test 1

The tether and bridle line systems are usually made from Dyneema®. The first validation test case is based on the experimental setup of Ruppert [59], used to determine the properties of this material. Here, the tether was hung vertically, with a mass M attached to its bottom end. The tether was left to rest for an hour to let the effects of material creep reside. The experiment was then carried out by lifting part of the mass m to a height of 10 cm, dropping it, and measuring the longitudinal vibrations of the mass. A visualization of the PSM of this validation case can be seen in Figure 7.2. The stiffness value for each spring element is calculated according to Equation 7.3.

$$k_{\text{spring}} = (n-1) k_{\text{tether}} \tag{7.3}$$

The additional mass $M + m$ is added to the bottom particle. To simulate the effect of a dropped mass, the linear impulse-momentum relation is used (Eq. 7.4). It is assumed that the dropped mass experiences zero air resistance. This simplifies the calculation of the time t that the mass took to drop from its initial condition $\boldsymbol{x}_0 = 0.1$ m (Eq. 6.1). The velocity at impact $\boldsymbol{v}$ can be found by multiplying t with $\boldsymbol{g}$, as the mass is dropped from a stationary position. Perfect energy transfer is assumed, which is added to the bottom particle by imposing an equivalent load over the course of one simulation step (Eq. 7.5).

$$\boldsymbol{P} = m\boldsymbol{v} \tag{7.4}$$

$$\boldsymbol{F}_{transfer} = \frac{\Delta \boldsymbol{P}}{\Delta t} \tag{7.5}$$

To find an analytical steady-state solution, the validation case is viewed as a superposition of harmonic oscillator systems. The individual converged positions of the particles can then be compared to their respective harmonic oscillation steady-state positions. The results of a simulation for the particle system with material damping can be seen in Figure 7.3. Similar graphs for both kinetic algorithms can be viewed in Appendix B.4. Every method converged to the calculated analytical values. The parameter values for the current and following validation tests are listed in Table A.1 in Appendix A.3.

A simulation was conducted with parameter values matching those of Ruppert's experiment and a small time step (h = 1×10^{-4} s). This minimizes numerical damping making it possible to check if oscillations match physical behavior. A fast-Fourier transform (FFT) analysis of the resulting simulation revealed a peak for the primary frequency at $3.000\,06$ Hz (Fig. 7.4). No secondary frequencies were found, as the initial impulse diminished quickly and there was no introduction of artificial noise during the simulation. Due to some remaining numerical damping, the primary frequency of the simulation is slightly decreased compared to the 3.0041 to 3.0518 Hz found in the experiment. The largest absolute error found between the steady-state and analytical position of a particle is 1.9×10^{-6} m. For this validation case, it appears that the use of the physical material stiffness doesn't result in instability or inaccuracy.

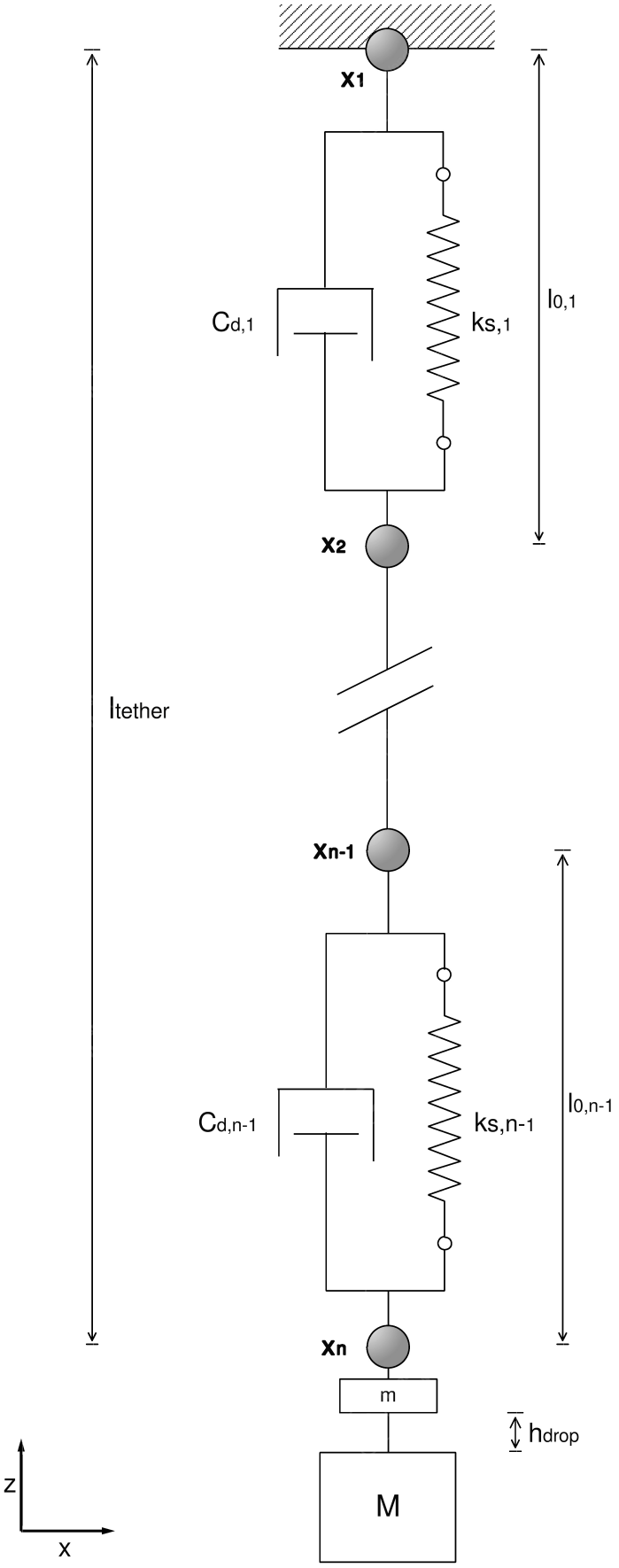


Figure 7.2: PSM model for simulation of validation case 1

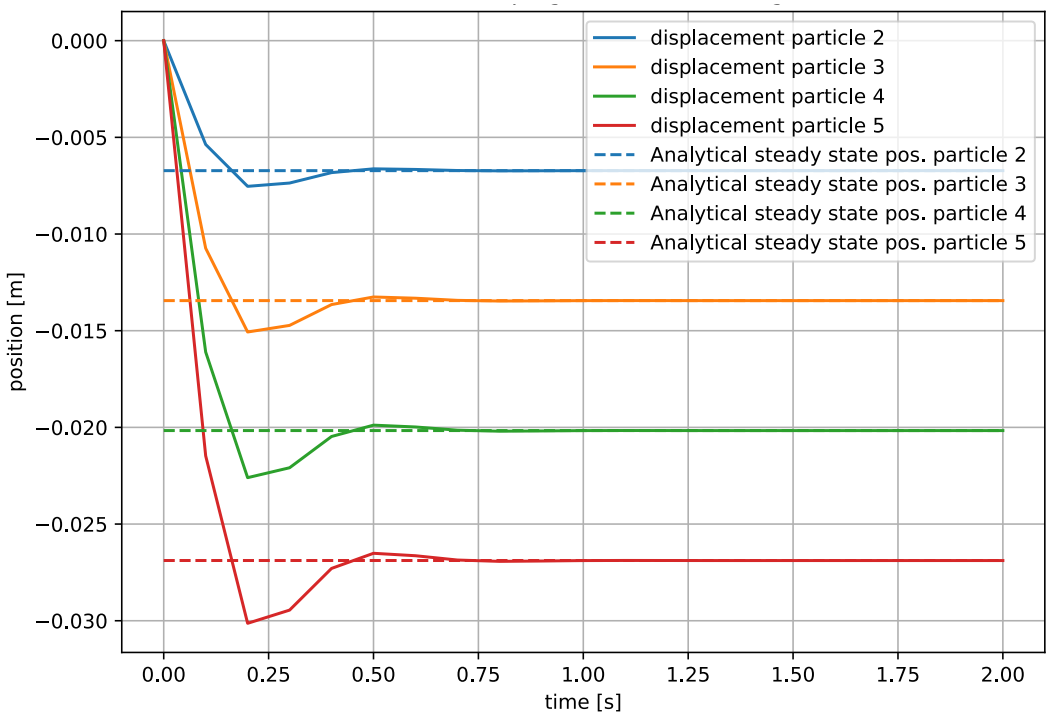


Figure 7.3: Particle e_z positions over time for PS with material damping, simulating validation case 1

7.3. Validation test 2

In validation case 2 the tether is suspended from both ends and experiences a perpendicular and constant external load, namely gravity. Under these idealized conditions, the resulting shape is known as a catenary. It is proven that a catenary is the state with the lowest potential energy and thus the shape that the tether will assume [53]. The resulting parabola is purely a function of geometrical parameters, material properties do not affect the shape (Eq. 7.6) [81]. Here, h is the vertical sag of the line and L is the total length of the arc.

$$\begin{aligned} y(x) &= a \cosh(\frac{x}{a}) \\ a &= \frac{\frac{1}{4}L^2 - h^2}{2h} \end{aligned} \tag{7.6}$$

The PSM of this validation case can be seen in Figure 7.5. Figure 7.6 shows the result of PS with material damping, where the number of particles is increased from n = 5 for the left graph to n = 20 in the right graph. The shapes found by the kinetic damping algorithms are indistinguishable from this result and can be seen in Appendix B.4. By simulating with a finer mesh, the shape of the tether is more accurately approximated. For both levels of discretization, the particles converged to the analytical steady-state position. The difference between kinetic damping algorithms becomes clear by plotting the particle displacement over time (Fig. 7.7). The length of the x-axis reveals at what time step either algorithm converged. For the same reasons as stated in Section 6.3, the kinetic damping with q-correction (right graph) requires more iterations to converge and thus takes longer to simulate.

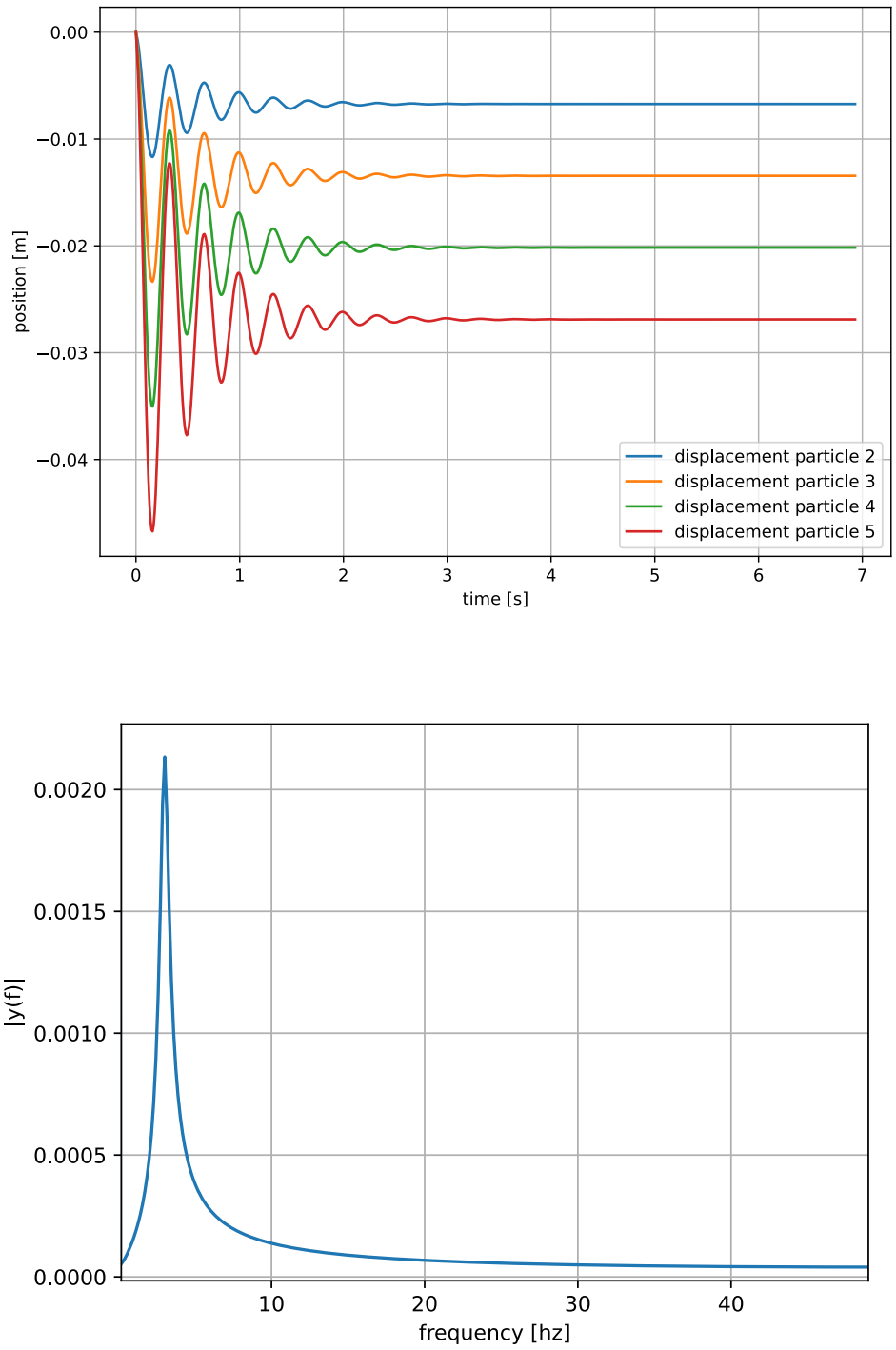


Figure 7.4: Particle positions of PS with material damping, simulation with a low timestep of h = 0.01 s (upper) and FFT of simulated longitudinal tether oscillations (lower).

7.4. Validation test 3

In validation test 3 the shape of a tether that is deflected by a perpendicular wind flow without being affected by gravity is sought. The PSM for this validation case can be viewed in Figure 7.8. Although this might appear as a rotated case of validation case 2 at first glance, they differ in their loading. The

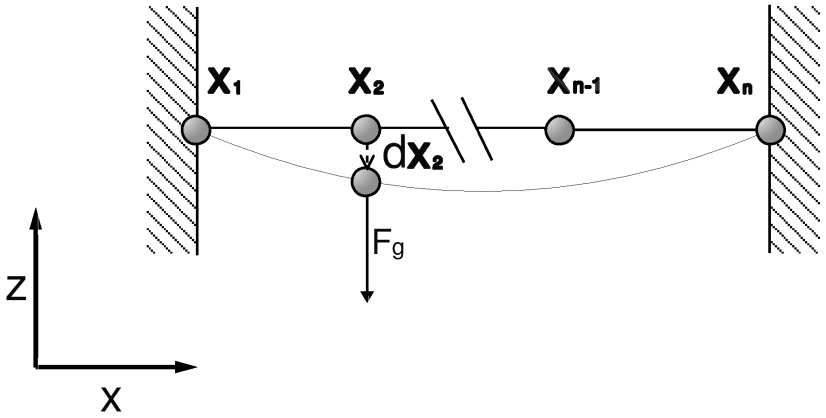


Figure 7.5: PSM model for simulation of validation case 2

tether drag is a function of the angle that the tether makes with respect to the wind field. The load is thus dependent on the geometry of the tether and the resulting problem is non-linear. The calculation of the tether drag is explained earlier, in Section 4.3. The drag forces are recalculated each iteration, and imposed on the respective particles. There does not exist an analytical solution for this problem, and to the author's knowledge, no experimental wind tunnel data exist for the tether shape. Therefore the resulting shape is compared to the simulation results of J. H. Baayen [30]. The value of the tether drag coefficient $C_{D,t}$ was taken from Jung [33] for a smooth cylinder at an attack angle of 90 degrees. The wind field vector was directed purely in the $\boldsymbol{e}_x$ direction with a value of 6 m/s.

The resulting shapes can be seen in Figure 7.9. The methods converged to a consistent shape with slight deviations on the cm scale, which is considered negligible. The maximum deflection, however, is approximately 5.8 m compared to the 4.5 m found by J. H. Baayen [30]. This 1.3 m discrepancy could be caused by differently calculated force values, as the found solutions visually appear similar in shape. Interestingly, the non-linear problem converges about twice as fast compared to cases where the imposed forces were kept constant. This is considered a side effect of the external viscous damping that opposes too much momentum since the overshoot in the linear problem is visibly larger.

7.5. Hencky's case

Hencky's problem can be used to evaluate how accurately the developed framework can predict membrane deformation under nonlinear loading conditions. The benefit of this test is that an analytical solution exists, which is rare for boundary value problems of nonlinear equations [42]. In this problem, a circular flat membrane with isotropic material properties is subjected to uniform pressure from one side. The uniform pressure results in nodal forces that are oriented in the transverse direction of the membrane, also known as a follower force. An analytical solution for the final deformed shape was first presented by Hencky [29] with an error in the computations later corrected by Chien [16] and Alekseev [2]. The analytical solution equations and parameter values can be found in Appendix B.3.

As mentioned in Section 7.1, the choice was made to discretize membrane-like material in quadrilateral elements. This choice was made because utilizing triangular elements results in a mesh with a particle in the center of the membrane, which might result in a more cone-like final shape. It was also thought that the use of quadrilateral elements would make for a simpler calculation of material properties for the translation to PSM.
Figure 7.10 illustrates how the forces evolve and how the direction and value of the forces in the problem are calculated. The four particles enclosing a quadrilateral element do not necessarily lie on a single plane, since three points in space define a plane. The direction of the force vector is calculated by taking the cross-product of the vectors between diagonal nodes (Eq. 7.7). The magnitude of the force is equivalent to the pressure times the area of the parallelogram formed by the two vectors. After calculation, the force is equally distributed amongst the four cornering particles.

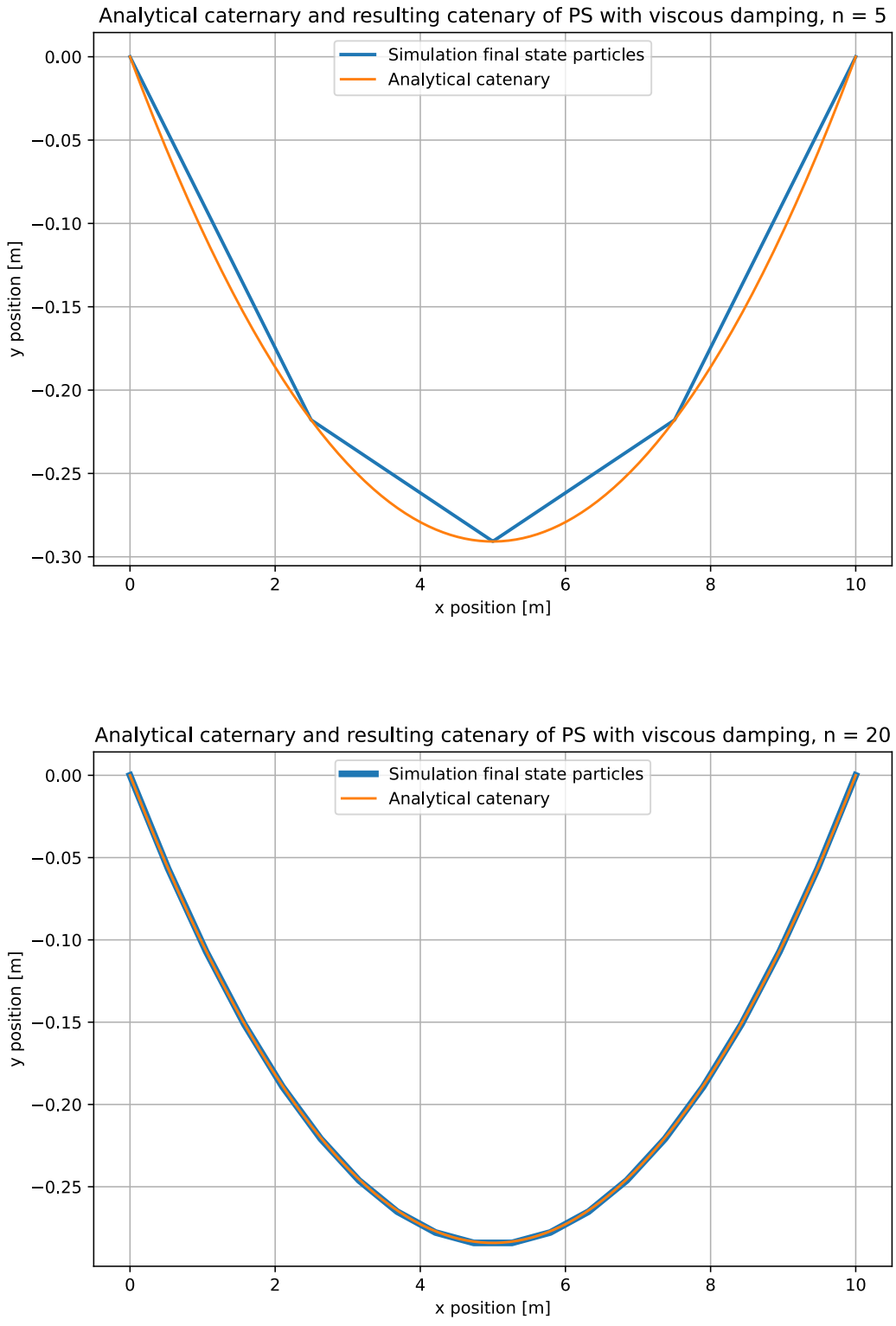


Figure 7.6: Found shapes of PS with material damping compared to analytically determined catenary, validation case 2.

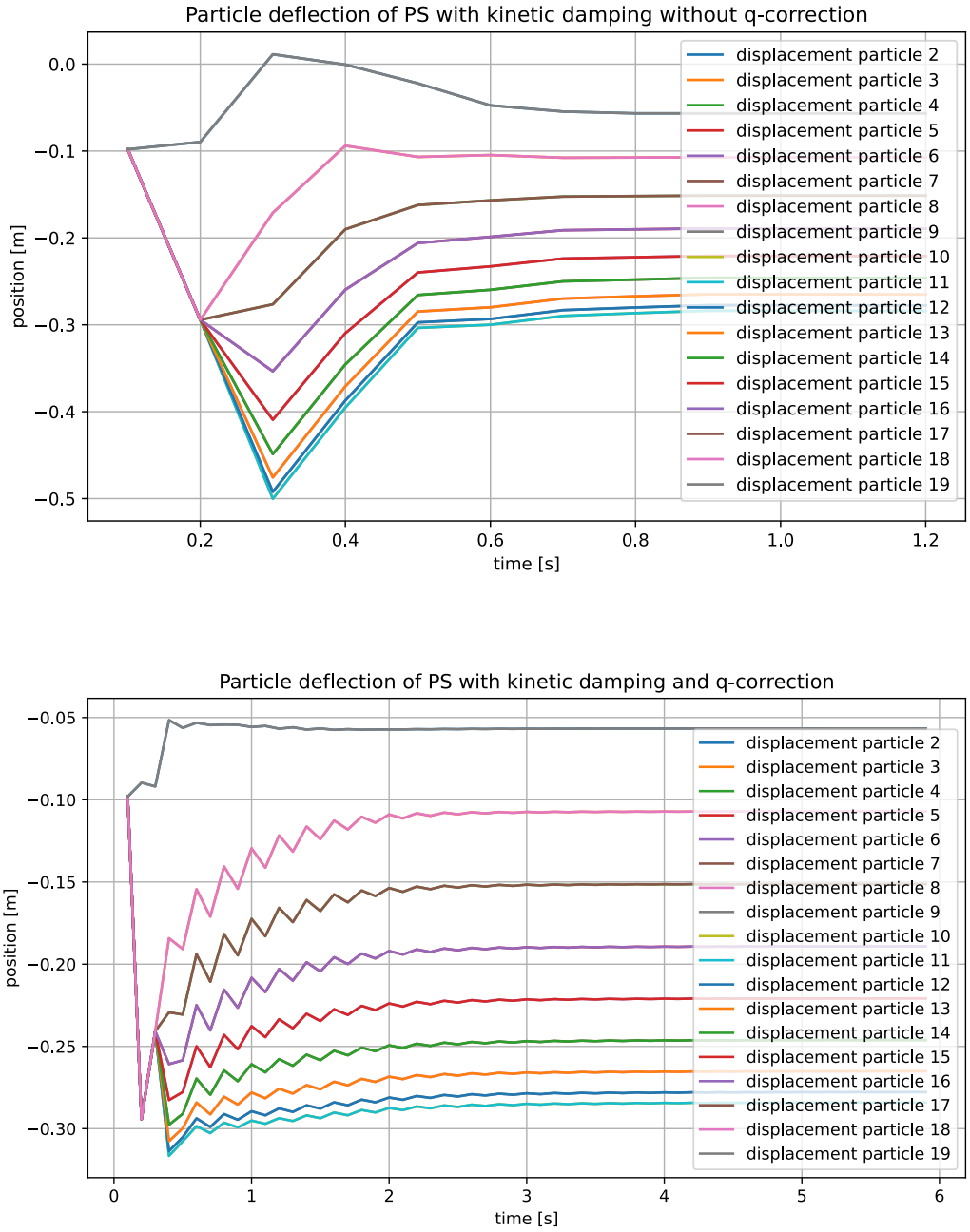


Figure 7.7: Particle e_z positions over time for PS with kinetic damping algorithms, simulating validation case 2

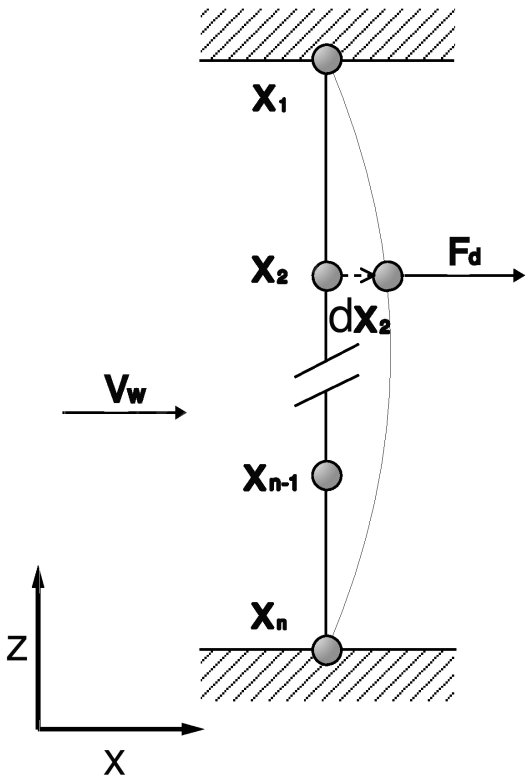


Figure 7.8: PSM model for simulation of validation case 3

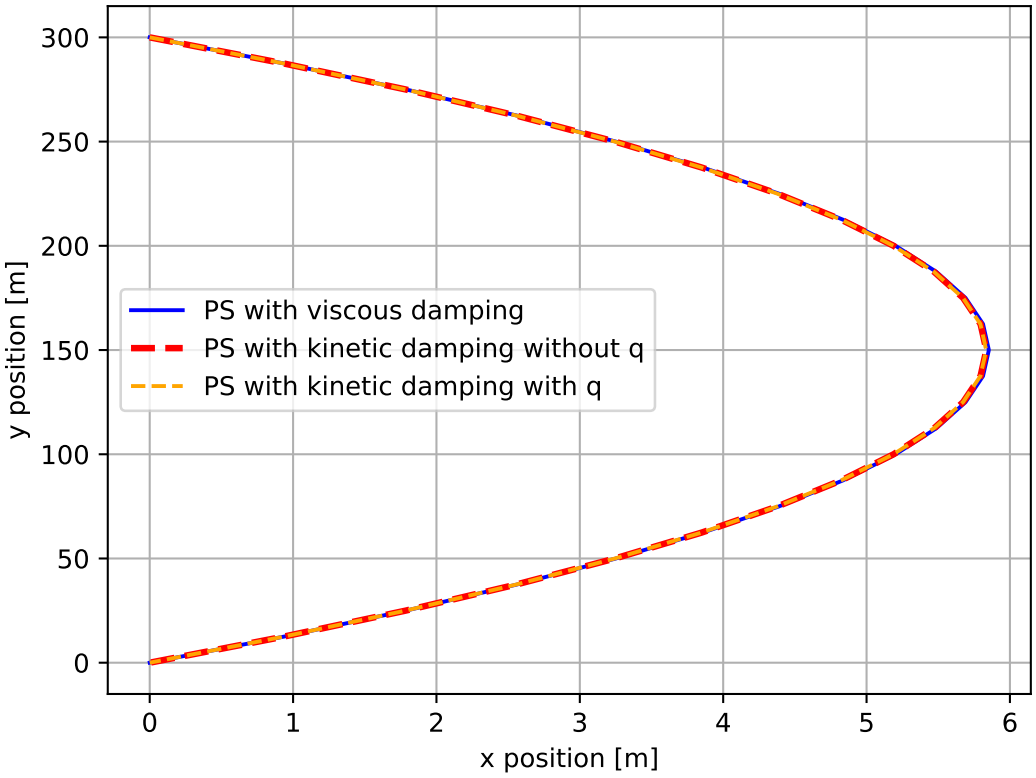


Figure 7.9: Found tether shapes with n = 25, simulation of validation case 3

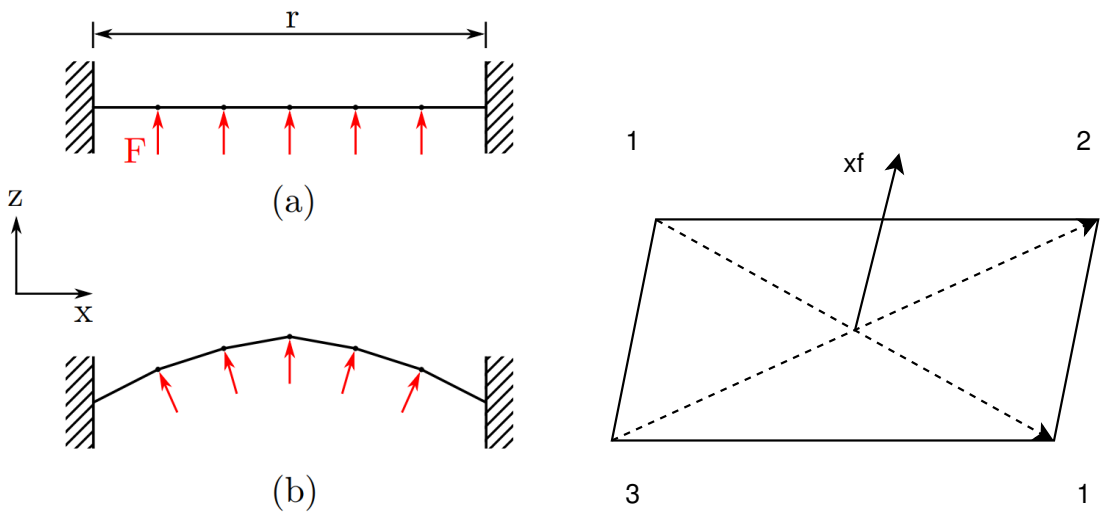


Figure 7.10: Cross-section of the Hencky problem (L) [1] and calculation of follower force for quadrilateral elements (R).

$$\boldsymbol{F}_p = pA\boldsymbol{x}_f = p\,|\boldsymbol{x}_{1,4} \times \boldsymbol{x}_{3,2}|\frac{\boldsymbol{x}_{1,4} \times \boldsymbol{x}_{3,2}}{|\boldsymbol{x}_{1,4} \times \boldsymbol{x}_{3,2}|} = p(\boldsymbol{x}_{1,4} \times \boldsymbol{x}_{3,2}) \tag{7.7}$$

Two PSMs were used for the simulation of the Hencky problem, one with a coarser mesh and one with a finer mesh (Fig 7.11). The open-source software Gmsh was used to generate the meshes [26]. The dimensions and material properties, except for the Poisson's ratio ν, were kept the same as in the validation test performed by Adam [1].

The results of simulations with material damping and kinetic damping can be seen in Figures 7.12 and 7.13, respectively. Both systems converged to the same shape, which had a center deflection of around 0.0218 m. As the deflection of the analytical solution is 0.017 m, there is a difference of 0.0048 m which is a large relative error of 28 %. After inspection, it was found that the method to determine the forces resulted in orientations that were not perpendicular to their respective surfaces (Fig. 7.14). At least part of the error could be attributed to this faulty approximation. The lack of bending and shearing resistance could also be factors that add to this discrepancy in deflection. The simulations with a finer mesh converged to a similar shape and deflection (Fig. 7.15 and 7.16). The error is therefore not a result of using a PSM with too few particles. For this increased amount of particles, the runtime of the PSM with kinetic damping was about 4 times faster than that of the PSM with material damping (ĩ2s compared to ẽ0s).

7.6. Conclusion

Based on the results of the validation cases, it can be concluded that the implemented framework can accurately predict the shape of a tether. While making a PSM for cables works naturally well, the Hencky case highlights one of the drawbacks of using a PS to simulate membrane deformation. Translating membrane-like material to a PSM needs a more thoughtful approach for accurate results. While the Hencky problem can be adjusted to facilitate the lack of Poisson's effect in the applied PSM, this approach wouldn't be accurate for actual canopy materials. Eischen et al. [20] showed that by adding shearing and out-of-plane bending relations to quadrilateral elements their PS implementation could predict cloth shape with similar accuracy to a FEM simulation, without the use of rotational springs. This reinforces the notion that sufficient accuracy can be achieved whilst preserving the computational efficiency of the framework by devoting additional efforts towards developing a PSM beforehand.
It was also found that q-correction as initially conceived for explicit schemes doesn't function as effectively for the BE scheme. This is attributed to the presence of numerical damping. For these validation cases, convergence speed was improved by skipping q-correction altogether. The accuracy of the framework wasn't affected by the addition of either kinetic damping algorithm.

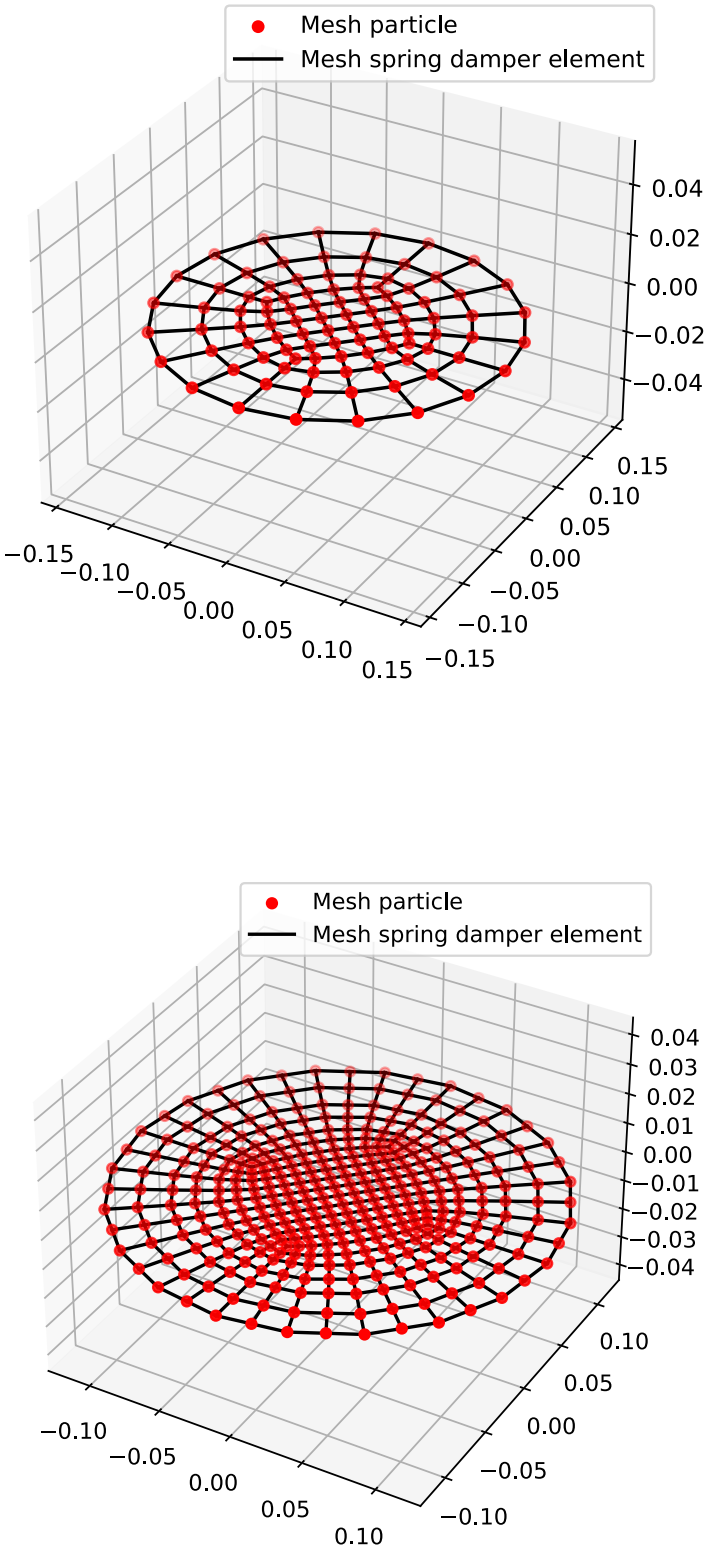


Figure 7.11: Meshes used to simulate the Hencky problem. A coarser mesh with 96 particles (upper) and a finer mesh with 361 particles (lower).

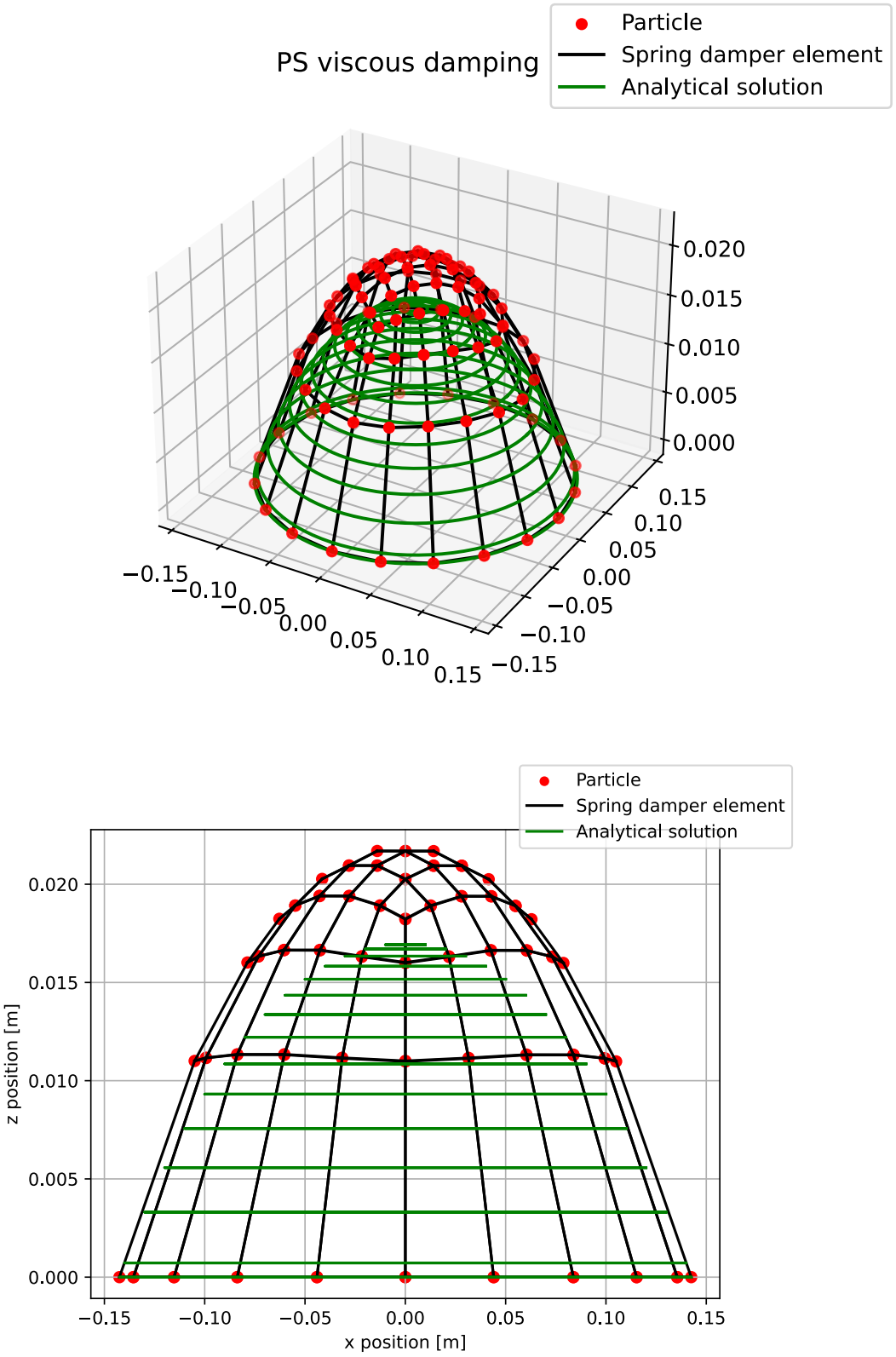


Figure 7.12: Resulting 3D shape of PS with material damping (upper) and 2D projection of the found shape (lower), for PSM with 96 particles.

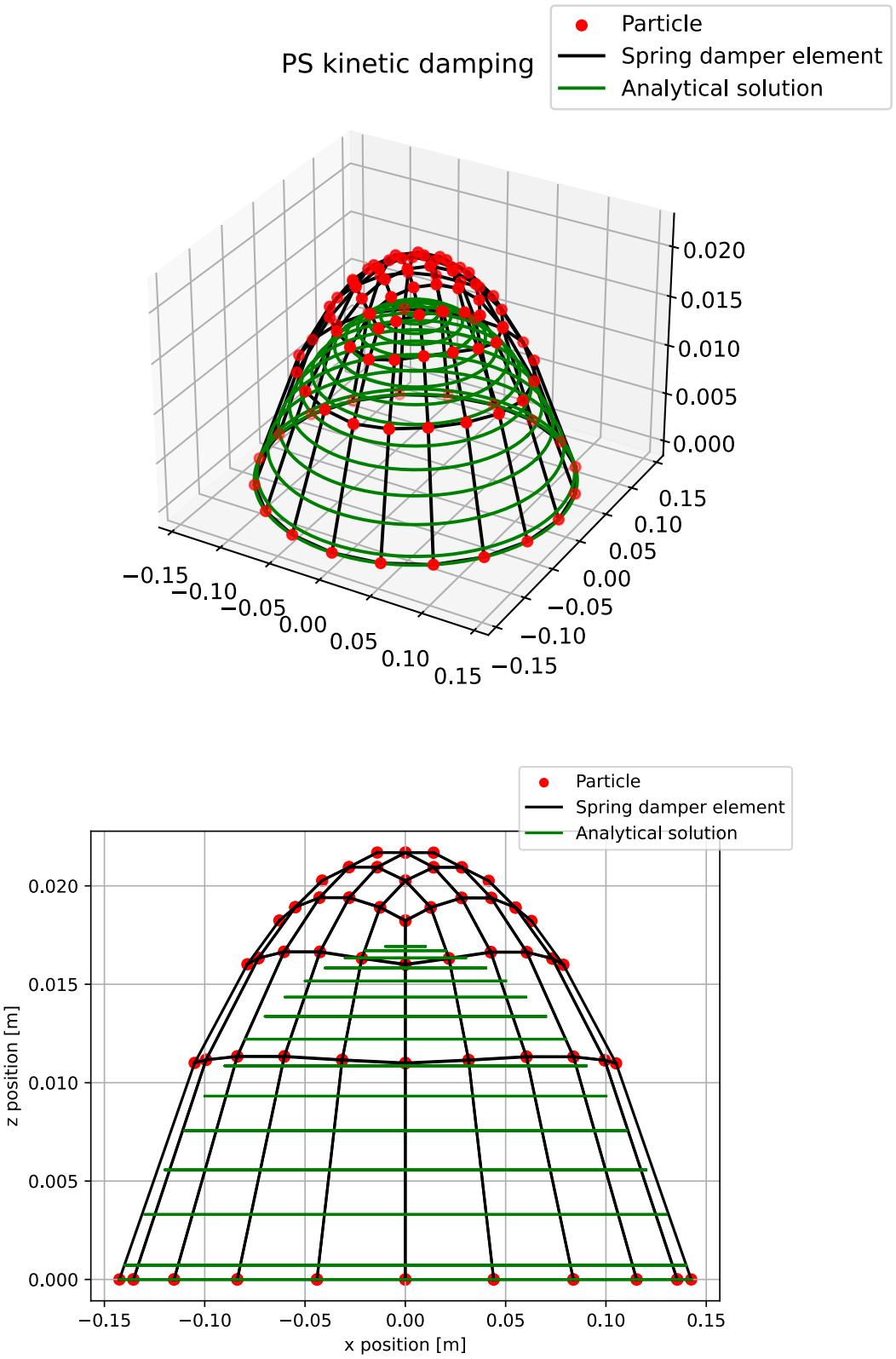


Figure 7.13: Resulting 3D shape of PS with kinetic damping (upper) and 2D projection of the found shape (lower), for PSM with 96 particles.

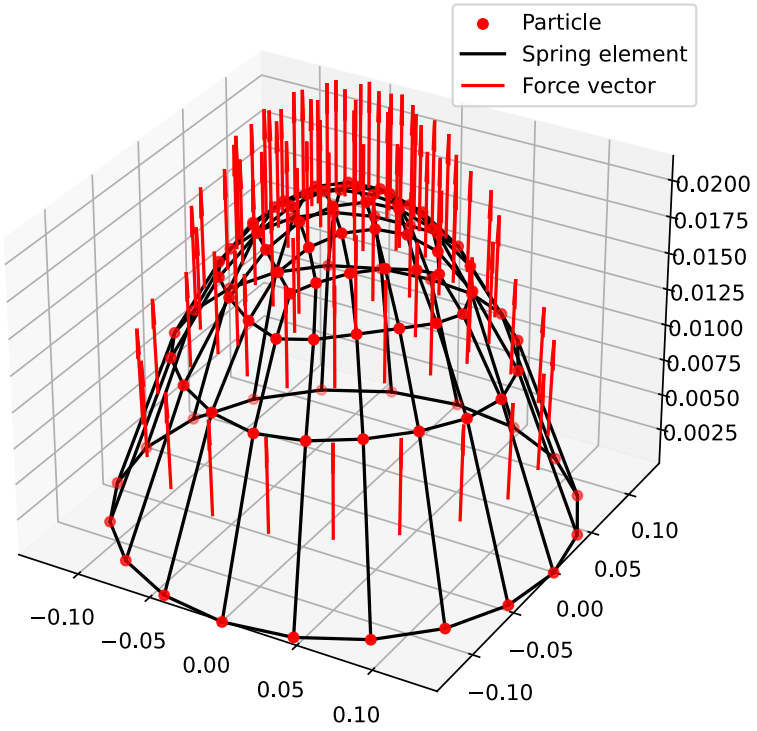


Figure 7.14: Scaled force vectors after convergence.

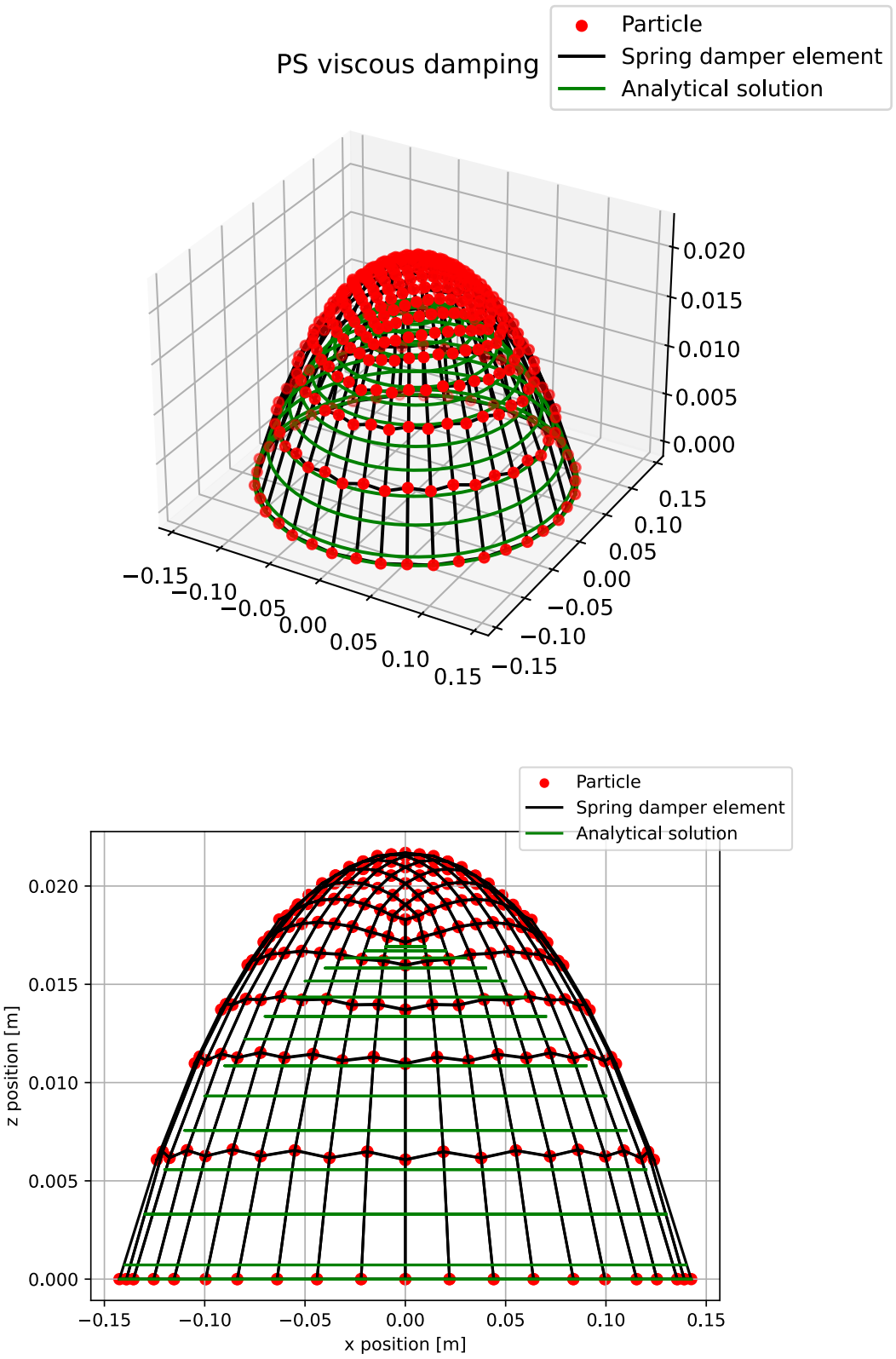


Figure 7.15: Resulting 3D shape of PS with material damping (upper) and 2D projection of the found shape (lower), for PSM with 361 particles.

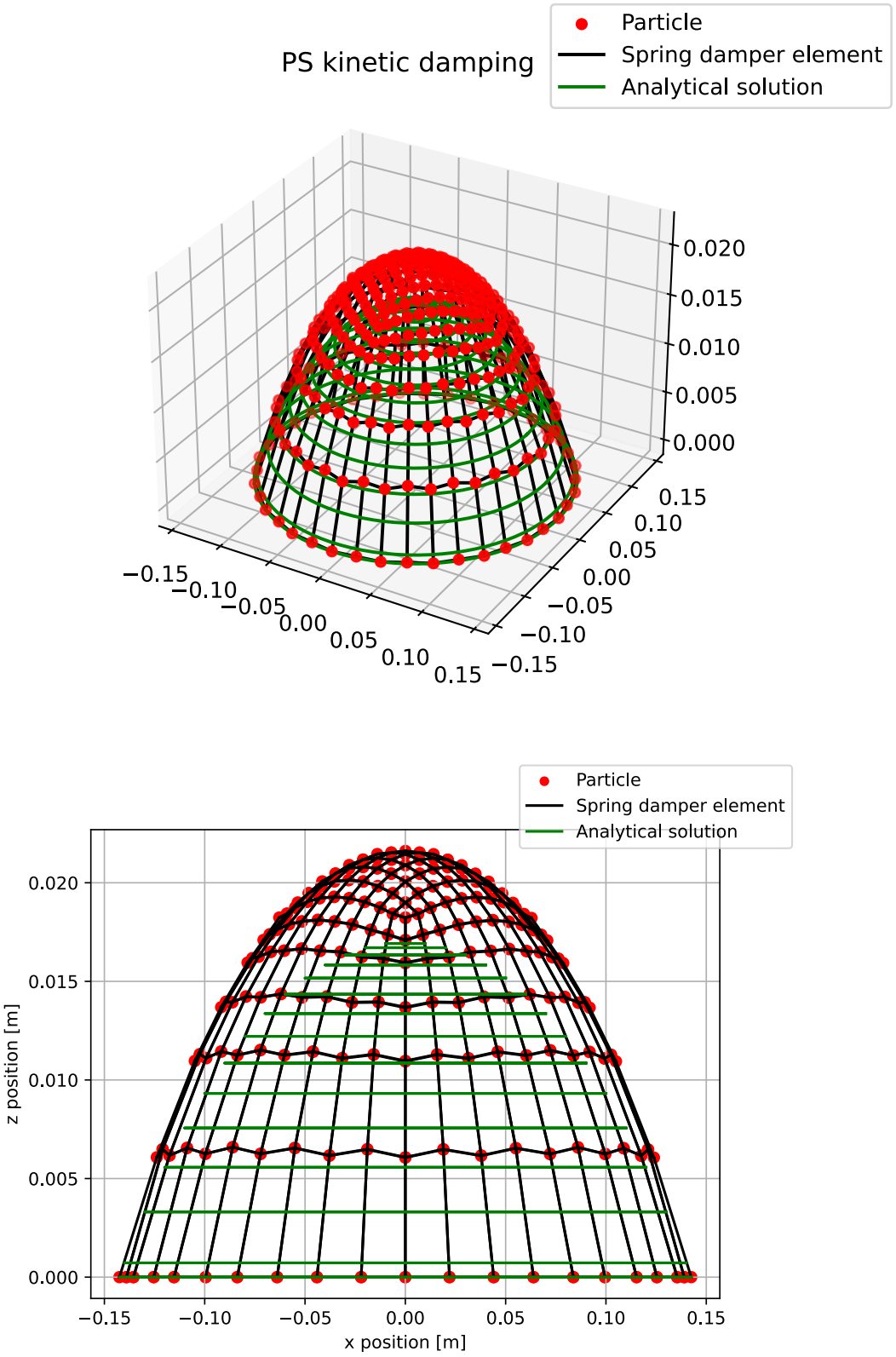


Figure 7.16: Resulting 3D shape of PS with kinetic damping (upper) and 2D projection of the found shape (lower), for PSM with 361 particles.

During testing with material stiffness values, no instabilities or decrease in accuracy were found. However, further experimentation is needed to determine the precise boundaries and conditions at which stiffness values affect the outcome of the framework.
Finally, the numerical damping effect grows with increasing values of timestep. This affects the PS with material damping most, as the system gets overdamped quickly, resulting in worse runtimes. The best performance is achieved with slightly underdamped systems.

8

Benchmarking results

The standard Python *time* module offers $\pm$16 ms precision on a Windows system, which is not sufficient for simulations that will converge in only a few milliseconds. Therefore, the *timeit* module will be used, which offers improved precision by selecting the most accurate timer compatible with your operating system. Furthermore, it disables the garbage collector to prevent it from influencing the results by running at inopportune moments [54]. The instancing of the PS object is done outside of the timing function, as this is regarded as a one-time cost and therefore not important to include in the resulting runtimes. The results presented in this chapter are average convergence times of 10 simulation runs unless specified otherwise.
For reference, the specifics of the software and hardware used to run the simulations are listed below.

- Laptop: *HP ZBook Studio G5*
- Processor: Intel Core i7-9750H (2.60 Ghz clock speed)
- RAM memory: 24 GB SSD
- Operating System: Windows 11 Home (64-bit version)
- Integrated Development Environment (IDE): PyCharm 2023.1.4 (Community Edition)
- Python v3.10.1

8.1. Black box solver comparison

The runtimes of the framework and a black-box solver *scipy.integrate.solve_ivp()* were tested to confirm that the former offers improved performance. The test was performed by increasing the number of particles for the validation case where a hanging tether was given an initial impulse and measuring convergence times. The results are shown in Figure 8.1. The graphs show that the framework performs better at low amounts of particles, and scales better when increasing the amount of particles. This confirmed the suspicions arising from the initial results obtained in Section 6.3. It is expected that the performance difference increases for membrane problems with non-linear loading. Furthermore, it can be seen that both kinetic damping algorithms outperform the framework with material damping and scale better. As this particular problem suits the kinetic damping algorithm, this is not indicative of the relative performance in other test cases.

8.2. Benchmarking

Benchmarking is conducted with three selected tests. Validation case 2, in which the tether experiences deflection due to perpendicular gravity, serves as an example of constant imposed loads. Validation case 3, in which the tether is deflected by perpendicular wind flow, illustrates nonlinear imposed loads. Lastly, the additional test case where the shape of a self-stressed net structure was sought, representing a case more similar to how the canopy could be modeled. In the following tables, these benchmarks are abbreviated as b2, b3, and b4, respectively. Hencky's problem is not used, as it is difficult to specify the amount of particles used to discretize the membrane.

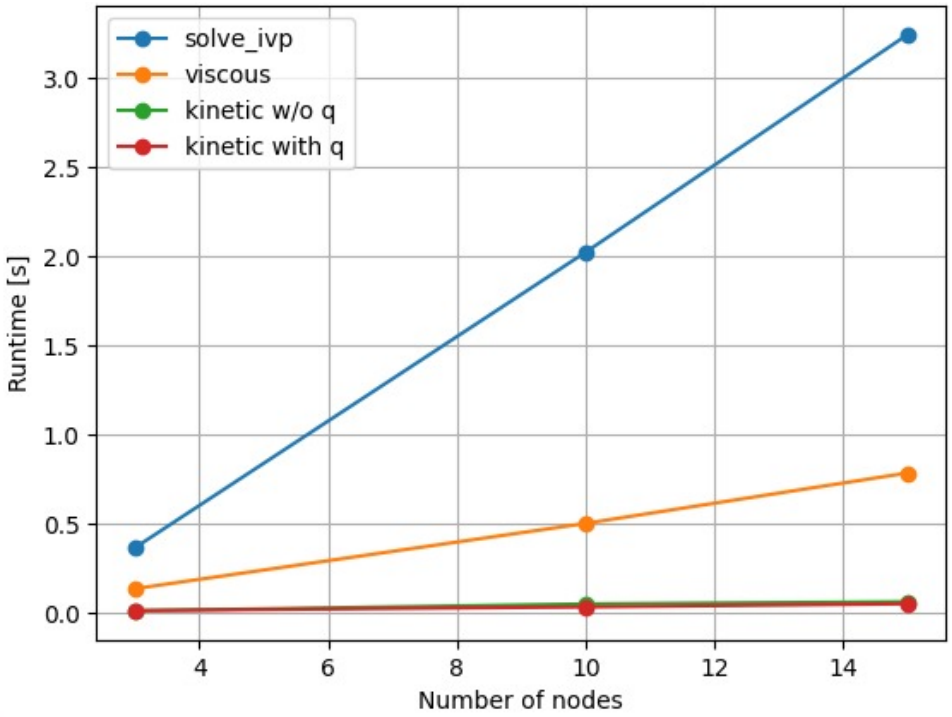


Figure 8.1: Comparison of runtimes for the implemented framework and a black-box solver.

The three methods that will be evaluated are denoted by PS_{vis}, PS_{kin}, and $PS_{kin,\,q}$ for the PS with material damping, for the PS with kinetic damping without the q-correction, and for the PS with kinetic damping and q-correction, respectively.

The main variable that will be varied to evaluate runtimes is the amount of DOF, which can be determined by the number of particles n. The value of n is not expected to be in the single digits when modeling a kite including tether and bridle lines. For example, the PSM of the V3 LEI kite shown in Figure 1.1 including tether and bridle lines consists of 37 particles. That is why performance testing starts with values of n in the double digits. In the results the amount of DOF is specified first, followed by n in parenthesis.
The spring stiffness k_s will also be varied to examine whether it affects performance and, more importantly, to check if the models are capable of converging for high stiffness values. Four values were chosen, two extreme values of $1\,\mathrm{N/m}$ and $1 \times 10^6\,\mathrm{N/m}$, one realistic material value of $6 \times 10^4\,\mathrm{N/m}$ and a value between the lower extreme and realistic stiffness, of $100\,\mathrm{N/m}$.
Finally, h, c_d and m are set to one and the number of simulation steps is limited to 1000. The fastest runtimes for each benchmark case and the combination of stiffness and DOF are highlighted in bold. If any simulations couldn't converge, either due to instability or surpassing the number of permitted steps, their time will be denoted with N/A. An extensive overview of results can be seen in Table 8.1.

The first observation is that the $PS_{kin,\,q}$ performed worst on almost every test case and combination of DOF and k. Additionally, in five cases the method failed to converge, some of which were due to the algorithm being trapped in a loop between states as Thedens [71] described. Given these results, it can not be justified to use the algorithm in combination with the BE scheme and it is henceforth disregarded. The following analysis and results will only include the two remaining methods.

The PS_{vis} outperforms the PS_{kin} on b3 with $k = 1\,\mathrm{N/m}$, although both methods couldn't converge for the lowest amount of particles. Upon closer examination, it was discovered that the increased runtime of the PS_{kin} was an indirect result of the low stiffness. The kinetic damping algorithm was activated due to the particles accelerating and decelerating quickly from a standstill. This cycle repeats, which can be seen as the bumpy trajectory of the particle positions in Figure 8.2, resulting in a slowed-down

Table 8.1: Resulting runtimes for varying DOF and k, with h = 1 s, c_d = 1 N s/m, m = 1 kg. The best runtimes of each combination of DOF and k are highlighted in **bold**.

DOF (n)	k = 1 N/m			k = 100 N/m		
PS_{vis}	b2	b3	b4	b2	b3	b4
39 (13)	0.051	N/A	0.053	1.068	N/A	0.375
123 (41)	0.489	**0.650**	0.308	0.216	0.643	0.138
255 (85)	2.454	**1.828**	1.444	**0.328**	**1.035**	0.320
543 (181)	10.366	**3.110**	3.533	10.433	5.098	0.607
PS_{kin}	b2	b3	b4	b2	b3	b4
39 (13)	**0.022**	N/A	**0.022**	**0.199**	**0.670**	**0.122**
123 (41)	**0.191**	0.931	**0.106**	**0.140**	**0.482**	**0.086**
255 (85)	**0.499**	2.090	**0.331**	0.748	1.079	**0.237**
543 (181)	**1.782**	5.146	**1.060**	17.491	**3.900**	**0.535**
$PS_{kin, q}$	b2	b3	b4	b2	b3	b4
39 (13)	N/A	N/A	0.030	0.274	0.780	6.882
123 (41)	0.946	3.114	0.234	0.188	0.946	0.761
255 (85)	1.937	2.879	N/A	0.433	1.550	0.307
543 (181)	2.429	6.890	N/A	**1.917**	5.918	0.538

DOF (n)	k = 6×10^4 N/m			k = 1×10^6 N/m		
PS_{vis}	b2	b3	b4	b2	b3	b4
39 (13)	1.117	0.633	0.073	0.571	6.511	0.046
123 (41)	**0.487**	1.381	**0.054**	**0.740**	3.424	0.051
255 (85)	**0.649**	6.950	0.201	2.026	**10.899**	0.153
543 (181)	3.008	63.818	0.336	5.135	N/A	0.402
PS_{kin}	b2	b3	b4	b2	b3	b4
39 (13)	1.791	**0.467**	0.067	**0.302**	**1.267**	**0.038**
123 (41)	0.491	**1.343**	0.055	0.827	**2.933**	**0.047**
255 (85)	2.530	**4.481**	**0.153**	**1.133**	12.034	**0.151**
543 (181)	**2.479**	**11.807**	**0.270**	**4.280**	**143.754**	**0.271**
$PS_{kin, q}$	b2	b3	b4	b2	b3	b4
39 (13)	**0.279**	0.707	**0.064**	0.607	5.312	**0.038**
123 (41)	0.869	4.225	0.058	1.646	17.037	0.055
255 (85)	1.469	10.784	0.157	4.614	26.056	0.154
543 (181)	5.181	51.851	0.275	9.474	N/A	0.282

convergence.

On the other hand, at higher stiffness values the kinetic damping algorithm performs the best for b3. In this specific test, a large stiffness in the system functions as a resistance against acceleration in this test case, which benefits the PS_{kin}. Further testing was conducted of b2 with k = 1 N/m, where particle masses were varied. This produced results that were in line with the aforementioned theory. Lower particle masses, and thus lower inertia caused the kinetic damping algorithm to activate more frequently than with higher particle masses. The other benchmark tests b2 and b4 with k = 1 N/m aren't subject to this effect, as there the particles aren't both accelerated and decelerated by the imposed force. In practice, it isn't likely that a stiffness value of 1 N/m will be used. However, it is important to note that in simulations where high accelerations and external damping are present, the PS_{kin} may perform worse than the PS_{vis}.

The performance comparison between the two remaining methods can made more clear by normalizing runtimes with the fastest time for each combination, $t_{norm} = \frac{t}{t_{min}}$ (Table 8.2). It was found that in 77% of combinations where either method performed best, the PS_{kin} outperformed the PS_{vis}. When the PS_{kin} outperformed the PS_{vis}, it converged on average a factor 2.3 faster with a standard deviation of $sigma$ = 1.49. The variation is significant, as evidenced by the normalized times ranging from 1.01 to 5.82.

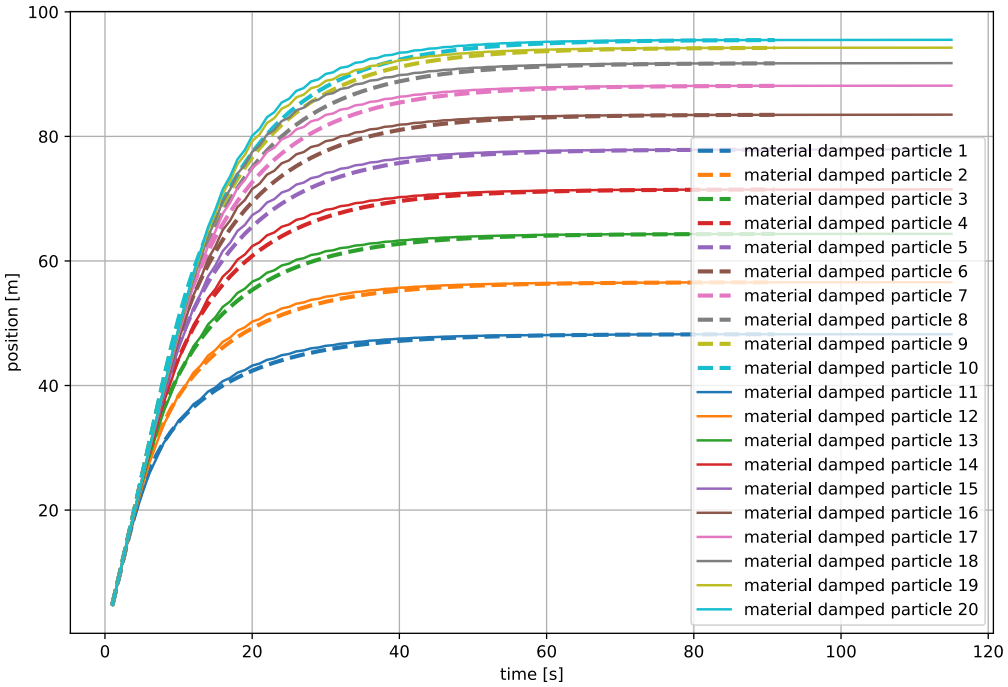


Figure 8.2: Particle position over time for b3 with low stiffness, illustrating the kinetic damping algorithm activating at each bump.

The method used to measure the runtimes is not without flaws, as the measured times are considerably affected by other processes that are concurrently running on the device that runs the simulations. Nonetheless, there were ten runs averaged and conducted for four stiffness values, four amounts of DOF, three benchmark cases, and two methods, resulting in a total of 960 simulations. For this quantity of simulations, it is highly unlikely that the found improvement of a factor 2.3 ± 1.49 faster in 77% of the cases is entirely random.

One possibility that has not yet been considered is that the improved convergence times of PS_{kin} can be attributed to the absence of material damping. Therefore, the benchmark tests were repeated with c_d set to zero, thus solely relying on numerical damping. The resulting runtimes can be viewed in Table 8.3. Other than an additional case where the PS with material damping couldn't converge, the outcome is nearly identical to previously obtained results. Therefore it is concluded that the addition of kinetic damping does predominantly affect runtimes beneficially. However, further analysis should be carried out to define more precisely the circumstances in which the kinetic damping algorithm provides a significant improvement.

8.3. Runtime scaling

As a final result, Figure 8.3 shows the runtimes of both PS_{vis} and PS_{kin} against increasing DOF for benchmark b4. To obtain this result, the stiffness was set to the value that is estimated to most closely resemble what will be used for kite simulation, namely $k = 6 \times 10^4\,\mathrm{N/m}$. Other parameters were again set to one, $dt = 1\,\mathrm{s}$, $m = 1\,\mathrm{kg}$, $c_d = 1\,\mathrm{N\,s/m}$. Benchmark b4 was chosen, as is the closest proxy for membrane deformation that could easily be varied in DOF.

It can be seen that both methods follow a similar path, but the curve of the kinetic damping algorithm is shifted to lower runtimes. This is believed to be due to the effect of the kinetic damping algorithm initially removing a large amount of the KE from the system, after which it converges in a similar manner to the viscous method.

Table 8.2: Normalized resulting runtimes for varying DOF and k, with h = 1 s, c_d = 1 N s/m, m = 1 kg. The best runtimes of each combination of DOF and k are highlighted in **bold**.

DOF (n)	k = 1 N/m			k = 100 N/m		
PS_{vis}	b2	b3	b4	b2	b3	b4
39 (13)	2.32	N/A	2.41	5.37	N/A	3.07
123 (41)	2.56	**1.00**	2.91	1.54	1.33	1.60
255 (85)	4.92	**1.00**	4.36	**1.00**	**1.00**	1.35
543 (181)	5.82	**1.00**	3.33	5.44	1.31	1.13
PS_{kin}	b2	b3	b4	b2	b3	b4
39 (13)	**1.00**	N/A	**1.00**	**1.00**	**1.00**	**1.00**
123 (41)	**1.00**	1.43	**1.00**	**1.00**	**1.00**	**1.00**
255 (85)	**1.00**	1.14	**1.00**	2.28	1.04	**1.00**
543 (181)	**1.00**	1.65	**1.00**	9.12	**1.00**	**1.00**

DOF (n)	k = 6×10^4 N/m			k = 1×10^6 N/m		
PS_{vis}	b2	b3	b4	b2	b3	b4
39 (13)	4.00	1.36	1.14	1.89	5.14	1.21
123 (41)	**1.00**	1.03	**1.00**	**1.00**	1.17	1.09
255 (85)	**1.00**	1.55	1.31	1.79	**1.00**	1.01
543 (181)	1.21	5.41	1.24	1.20	N/A	1.48
PS_{kin}	b2	b3	b4	b2	b3	b4
39 (13)	6.42	**1.00**	1.05	**1.00**	**1.00**	**1.00**
123 (41)	1.01	**1.00**	1.02	1.12	**1.00**	**1.00**
255 (85)	3.90	**1.00**	**1.00**	**1.00**	1.10	**1.00**
543 (181)	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 8.3: Resulting runtimes for varying DOF and k, with h = 1 s, c_d = 0 N s/m, m = 1 kg. The best runtimes of each combination of DOF and k are highlighted in **bold**.

DOF (n)	k = 1 N/m			k = 100 N/m		
PS_{vis}	b2	b3	b4	b2	b3	b4
39 (13)	0.057	N/A	0.056	1.507	N/A	0.349
123 (41)	0.654	**0.751**	0.356	0.313	0.579	0.134
255 (85)	4.377	**1.922**	1.606	**0.882**	**1.015**	0.326
543 (181)	14.712	**3.778**	3.273	18.534	4.680	0.604
PS_{kin}	b2	b3	b4	b2	b3	b4
39 (13)	**0.023**	N/A	**0.022**	**0.303**	**0.691**	**0.128**
123 (41)	**0.332**	1.032	**0.139**	**0.186**	**0.420**	**0.092**
255 (85)	**0.955**	2.218	**0.409**	0.922	1.037	**0.249**
543 (181)	**2.744**	5.618	**1.118**	14.865	**3.622**	**0.556**

DOF (n)	k = 6×10^4 N/m			k = 1×10^6 N/m		
PS_{vis}	b2	b3	b4	b2	b3	b4
39 (13)	1.904	0.583	0.075	0.280	4.077	0.049
123 (41)	0.478	**1.112**	**0.055**	**0.651**	3.222	**0.053**
255 (85)	**0.856**	**4.496**	0.195	1.647	**14.354**	**0.161**
543 (181)	**2.131**	N/A	0.357	5.362	N/A	0.441
PS_{kin}	b2	b3	b4	b2	b3	b4
39 (13)	**1.526**	**0.446**	**0.068**	**0.267**	**1.556**	**0.041**
123 (41)	**0.447**	1.202	0.056	0.794	**2.872**	**0.053**
255 (85)	2.107	4.879	**0.164**	**1.016**	17.251	**0.161**
543 (181)	**2.139**	**14.620**	**0.291**	**4.340**	**123.532**	**0.279**

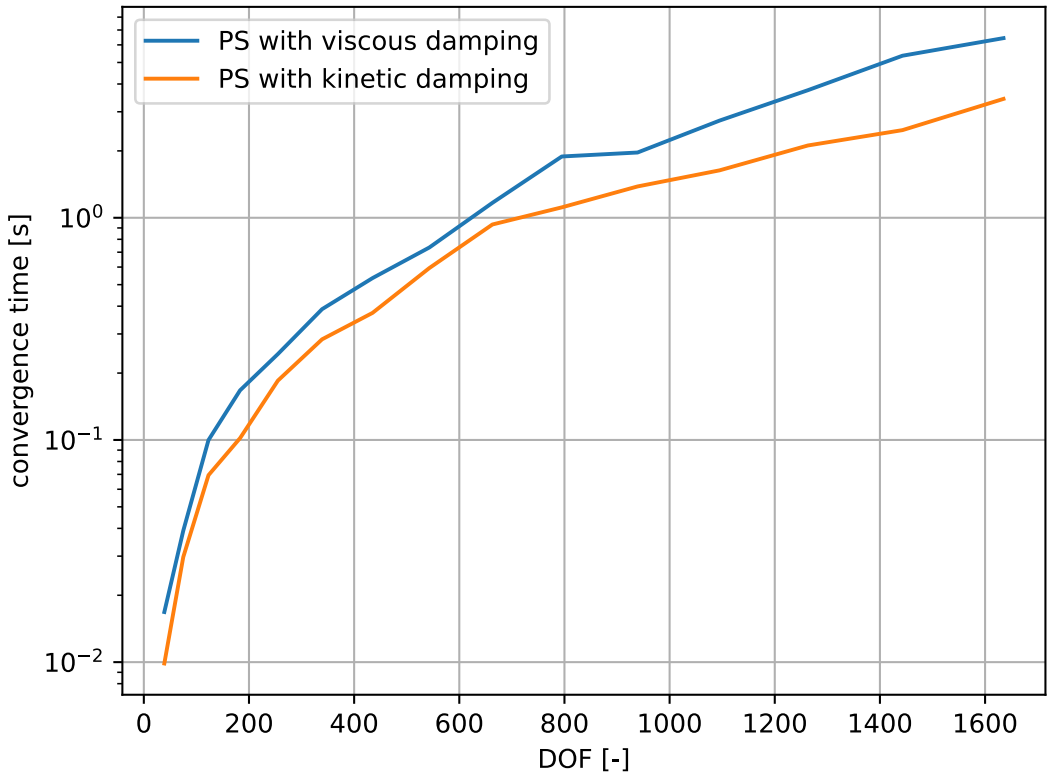


Figure 8.3: Semi-logarithmic plot showing the increase in runtime for increasing DOF

A quasi-steady modeling framework was developed by vlugt et al. [78] to predict generated power during pumping cycles. They found that the error between the resulting power prediction and reference converged to less than 3% if the time-step between the quasi-steady states was smaller than 0.1 s. They opted to use a value of 0.01 s to be on the safe side. If these values are taken as a range to indicate efficient runtime, the current framework should be able to simulate from 15 up to 60 particles given the values used for the scaling shown in Figure 8.3. Referring to the 37 particles used in the PSM of the V3 LEI kite, using the current developed framework appears feasible. Further optimization of the framework could be achieved, as memory allocation is not optimized, and no optimal tuning for masses, damping, and timestep was executed.

9

Conclusion and recommendations

9.1. Conclusions

The aim of the research was to develop a fast and reasonably accurate structural model of soft-wing kites and connected tethers. Various options were examined and weighed, leading to the choice of a previously used PS framework. It was revealed that incorporating a kinetic damping algorithm into the framework has the potential to enhance runtime, without affecting accuracy. Thus, a PS framework was developed in Python and the option to run simulations with a kinetic damping algorithm was added. During verification testing, it was observed that the kinetic damping algorithm in combination with the BE scheme did not work as anticipated due to numerical damping. Therefore, a slight modification was made to the algorithm, to bypass the q-correction.

Next, the framework and kinetic damping algorithms were validated with four selected test cases. Three cases isolated tether functionalities and one case tested the accuracy of membrane deformation. It was found that each method is capable of accurately predicting the shape of tether and bridle line systems. This partially addressed one of the sub-research questions, demonstrating that the framework could be modified without affecting accuracy. A noteworthy conclusion is that to accurately simulate the deformation of membranes, more consideration must be given to the development of the PSM that represents the considered membrane continuum.

Benchmark testing was carried out for three cases with varying values of k, c, and most importantly with increasing DOF of the system by increasing the number of particles n. The three test cases that were utilized for evaluation were chosen for their loading conditions. The tether deflected by gravity is an example where imposed loads are constant, the tether deflected by wind flow is an example of nonconstant imposed loads, and finally the shape-finding case for unloaded self-stressed networks.
A comparison between the framework and a black-box solver demonstrated that the framework outperforms the latter in terms of runtime. This showed the advantage of using the self-coded solver with explicit computation of Jacobian matrices over a black-box solver.
Benchmarking revealed that the kinetic damping algorithm with q-correction underperformed in almost any condition. This variant of the kinetic damping algorithm was subsequently disregarded. Furthermore, it was found that normalized runtimes between the PS_{vis} and PS_{kin} vary widely. Part of the random nature and variance in result is believed to be a consequence of the methodology used to measure performance. A specific set of conditions was discovered where the PS_{kin} adversely affected runtime. This is the case when the system has low inertia and is subjected to external forces that cause both high acceleration and deceleration of particles. Except for these specific conditions, in most instances, the PS_{kin} offered improved runtimes. Further analysis should be performed to identify the conditions for which the kinetic damping algorithm does and does not offer improvement.
Finally, the scaling of runtime against increasing DOF was plotted for representative conditions of kite simulation. The PS_{kin} showed similar scaling to the PS_{vis} but shifted to lower runtimes. This is expected to be due to the algorithm removing a large fraction of energy from the system, and subsequently following regular convergence of the framework. Benchmarking resulted in strong indications in favor of

using the kinetic damping algorithm with the PS, especially for the settings that are believed to best represent the kite model. Combining these results, an answer can be formulated to the main research question: “Can the PSM be implemented in Python, using a mix of Object-Oriented (OO) and non-OO programming techniques, to efficiently predict deformation?”.

The validation tests demonstrated that the utilized PSM can predict tether shapes accurately in the developed framework. However, to achieve sufficient accuracy for non-linear, asymmetrical membrane deformation, a more thoughtful approach is required. By doing this, the advantage of utilizing the PS framework as opposed to, for example, FEM is maintained. Benchmarking results indicated that the current framework could be fast enough for simulation in the range of 15 to 60 particles. Considering the current V3 LEI kite PSM is discretized by 37 particles and the developed framework is far from optimized, the PS framework is considered fast enough, and further development is encouraged.

9.2. Recomendations

In the broader context of research on structural models, it would be worthwhile investigating whether a variant of FDM could achieve faster runtimes while maintaining accuracy. Implementation as a sole method for solution-finding could be possible, but if this isn’t feasible due to for example unpracticable results, a hybrid methodology such as the implementation of Veenendaal and Block [76] could lead to improved convergence times. A good starting point would be the open-source FDM implemented by Vahid Moosavi [75] in Python.

The implemented framework and algorithms have undergone comprehensive testing for the selected validation cases, but it is clear that membrane PSMs require further examination. While linear springs are used, any form of mesh that approximates membrane-like material will exhibit non-linear behavior. Comparison against empirical data on deformation would be preferred, but proxy cases are the next best option in the absence of such data. The Hencky problem is not the most suitable test case as it assumes isotropic material properties, which is incredibly challenging, if not impossible, to achieve when developing a PSM. A potential alternative could be a validation case where anisotropic fabric is tested for tensile strength, as performed by Anton et al. [4]. The results could then directly be compared to their developed non-linear model.
This ties into the next recommendation, namely the option of adding additional relations or elements to the framework. For example, non-linear springs based on either physical equations or empirically determined stress-strain relations for certain ranges of strain. Other relations, such as the shear resistance and out-of-plane bending resistance as implemented by Eischen et al. [20], could also increase accuracy, without significantly affecting runtime.

Regarding optimization of the developed framework, the most promising suggestion for lowering runtimes would be to integrate an automated tuning parameter algorithm. Papadrakakis [49] described an algorithm to evaluate tuning parameter values for DR methods with either viscous or kinetic damping. Other methods, such as adaptive time-stepping [32] could be considered if automatic parameter control can’t be applied to the framework.
The relative share of the BiCGSTAB iterative method used to solve the linear system of equations could be investigated. If it amounts to a large portion of the floating point operations, performance could be improved by making use of preconditioning. Notably, the preconditioning technique developed by [84] provides the added benefit of addressing inaccuracy related to the conditioning of matrix $\boldsymbol{A}$.

References

[1] Niklas Johannes Adam. “Computational simulation of fluid-structure interaction of soft kites”. MA thesis. Stuttgart : University of Stuttgart, Institute of Mechanics, Structural Analysis and Dynamics, 2018. URL: http://dx.doi.org/10.18419/opus-10628.

[2] SA Alekseev. “Elastic circular membranes under the uniformly distributed loads. Eng”. In: *Corpus* 14 (1953), pp. 196–198.

[3] Alexander Richard Batchelor. *Msc_Alexander_Batchelor*. https://github.com/ARBatchelor/Msc_Alexander_Batchelor. Accessed 24 May 2023. 2023.

[4] Sabin V. Anton et al. “Development and Validation of a Nonlinear Fabric Model for Subsonic Parachute Aerodynamics”. In: *Journal of Spacecraft and Rockets* 0.0 (0), pp. 1–21. DOI: 10.2514/1.A35583. eprint: https://doi.org/10.2514/1.A35583. URL: https://doi.org/10.2514/1.A35583.

[5] Cristina Archer and Ken Caldeira. “Global Assessment of High-Altitude Wind Power”. In: *Energies* 2 (June 2009). DOI: 10.3390/en20200307.

[6] Cristina L. Archer. “An Introduction to Meteorology for Airborne Wind Energy”. In: *Airborne Wind Energy*. Ed. by Uwe Ahrens, Moritz Diehl, and Roland Schmehl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 81–94. ISBN: 978-3-642-39965-7. DOI: 10.1007/978-3-642-39965-7_5. URL: https://doi.org/10.1007/978-3-642-39965-7_5.

[7] David Baraff and Andrew Witkin. “Large Steps in Cloth Simulation”. In: *Proceedings of SIGGRAPH* 98 (June 2001). DOI: 10.1145/280814.280821.

[8] Michael R. Barnes. “Form Finding and Analysis of Tension Structures by Dynamic Relaxation”. In: *International Journal of Space Structures* 14.2 (1999), pp. 89–104. DOI: 10.1260/026635191491494722. eprint: https://doi.org/10.1260/0266351991494722. URL: https://doi.org/10.1260/0266351991494722.

[9] Philip Bechtle et al. “Airborne wind energy resource analysis”. In: *Renewable Energy* 141 (2019), pp. 1103–1116. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2019.03.118. URL: https://www.sciencedirect.com/science/article/pii/S0960148119304306.

[10] Kai-Uwe Bletzinger and Ekkehard Ramm. “A General Finite Element Approach to the form Finding of Tensile Structures by the Updated Reference Strategy”. In: *International Journal of Space Structures* 14.2 (1999), pp. 131–145. DOI: 10.1260/0266351991494759. eprint: https://doi.org/10.1260/0266351991494759. URL: https://doi.org/10.1260/0266351991494759.

[11] Allert Bosch et al. “Dynamic Nonlinear Aeroelastic Model of a Kite for Power Generation”. In: *Journal of Guidance, Control, and Dynamics* in press (Sept. 2014). DOI: 10.2514/1.G000545.

[12] Allert Bosch et al. “Nonlinear Aeroelasticity, Flight Dynamics and Control of a Flexible Membrane Traction Kite”. In: *Airborne Wind Energy*. Ed. by Uwe Ahrens, Moritz Diehl, and Roland Schmehl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 307–323. ISBN: 978-3-642-39965-7. DOI: 10.1007/978-3-642-39965-7_17. URL: https://doi.org/10.1007/978-3-642-39965-7_17.

[13] H.A. Bosch. “Finite Element Analysis of a Kite for Power Generation”. MA thesis. Delft University of Technology, 2012. URL: http://resolver.tudelft.nl/uuid:888fe64a-b101-438c-aa6f-8a0b34603f8e.

[14] Jeroen Breukels. “An engineering methodology for kite design”. PhD thesis. Jan. 2011. URL: http://resolver.tudelft.nl/uuid:cdece38a-1f13-47cc-b277-ed64fdda7cdf.

[15] Oriol Cayon, Mac Gaunaa, and Roland Schmehl. “Fast Aero-Structural Model of a Leading-Edge Inflatable Kite”. In: *Energies* 16 (Mar. 2023), p. 3061. DOI: 10.3390/en16073061.

[16] Wei Zang Chien. “Asymptotic behavior of a thin clamped circular plate under uniform normal pressure at very large deflection”. In: *Sci. Rep. Natl. Tsinghua Univ* 5 (1948), pp. 193–208.

[17] Kwang-Jin Choi and Hyeong-Seok Ko. “Stable but Responsive Cloth”. In: *ACM Transactions on Graphics* 21 (July 2002). DOI: 10.1145/1198555.1198571.

[18] Chris Meyer. *The Harmonic Oscillator*. https://faculty.washington.edu/seattle/physics227/reading/reading-2b.pdf. Accessed 23 March 2023. 2008.

[19] C. F. Curtiss and J. O. Hirschfelder. “Integration of Stiff Equations*”. In: *Proceedings of the National Academy of Sciences* 38.3 (1952), pp. 235–243. DOI: 10.1073/pnas.38.3.235. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.38.3.235. URL: https://www.pnas.org/doi/abs/10.1073/pnas.38.3.235.

[20] Jeffrey Eischen et al. “Continuum versus particle representations”. In: *Cloth Modeling and Animation* (Jan. 2000).

[21] Uwe Fechner. “A Methodology for the Design of Kite-Power Control Systems”. PhD thesis. Nov. 2016. DOI: 10.4233/uuid:85efaf4c-9dce-4111-bc91-7171b9da4b77.

[22] Uwe Fechner et al. “Dynamic model of a pumping kite power system”. In: *Renewable Energy* 83 (2015), pp. 705–716. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2015.04.028. URL: https://www.sciencedirect.com/science/article/pii/S0960148115003080.

[23] W. B. Fichter. “Some Solutions for the Large Deflections of Uniformly Loaded Circular Membranes”. In: 1997. URL: https://api.semanticscholar.org/CorpusID:117977936.

[24] Mikko Folkersma, Roland Schmehl, and Axelle Viré. “Steady-state aeroelasticity of a ram-air wing for airborne wind energy applications”. In: *Journal of Physics: Conference Series* 1618.3 (Sept. 2020), p. 032018. DOI: 10.1088/1742-6596/1618/3/032018. URL: https://dx.doi.org/10.1088/1742-6596/1618/3/032018.

[25] Nick Geschiere. “Dynamic modelling of a flexible kite for power generation”. MA thesis. Delft University of Technology, 2014. URL: http://resolver.tudelft.nl/uuid:6478003a-3c77-40ce-862e-24579dcd1eab.

[26] Christophe Geuzaine and Jean-François Remacle. *Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*. https://gmsh.info/. Accessed 29 Oct 2023. 2023.

[27] David Griffiths and Desmond Higham. *Numerical Methods for Ordinary Differential Equations: Initial Value Problems*. Jan. 2010. ISBN: 978-0-85729-147-9. DOI: 10.1007/978-0-85729-148-6.

[28] R.B. Haber and J.F. Abel. “Initial equilibrium solution methods for cable reinforced membranes part I—formulations”. In: *Computer Methods in Applied Mechanics and Engineering* 30.3 (1982), pp. 263–284. ISSN: 0045-7825. DOI: https://doi.org/10.1016/0045-7825(82)90080-9. URL: https://www.sciencedirect.com/science/article/pii/0045782582900809.

[29] HJZMP Hencky. “Uber den Spannungszustand in kreisrunden Platten mit verschwindender Biegungssteifigkeit”. In: *Z. Math. Phys.* 63 (1915), pp. 311–317.

[30] J. H. Baayen. *Modeling a kite on a variable length flexible tether*. Accessed 14 June 2023, internal paper at Delft University of Technology. 2011.

[31] Jorge Royan. *Munich - Frei Otto Tensed structures*. https://commons.wikimedia.org/wiki/File:Munich_-_Frei_Otto_Tensed_structures_-_5249.jpg. Accessed 12 July 2023. 2007.

[32] Samuel Jung, Tae-Yun Kim, and Wan-Suk Yoo. “Adaptive Step-Size Control for Dynamic Relaxation Using Continuous Kinetic Damping”. In: *Mathematical Problems in Engineering* (2018). URL: https://api.semanticscholar.org/CorpusID:59461497.

[33] Timothy Jung. “Wind Tunnel Study of Drag of Various Rope Designs”. In: June 2009. ISBN: 978-1-62410-130-4. DOI: 10.2514/6.2009-3608.

[34] Robin van Kappel. “Aerodynamic Analysis Tool for Dynamic Leading Edge Inflated Kite Models”. MA thesis. Delft University of Technology, 2012. URL: http://resolver.tudelft.nl/uuid:385d316b-c997-4a02-b0f3-b30c40fffc32.

[35] Mustafa Can Karadayıt. “Particle System Modelling and Dynamic Simulation of a Tethered Rigid Wing Kite for Power Generation”. MA thesis. Delft University of Technology, 2016. URL: http://resolver.tudelft.nl/uuid:d6a2fcf8-7fce-4eb8-857b-209b9faac755.

[36] Axel Kilian and John Ochsendorf. “Particle-spring systems for structural form finding”. In: *Journal of the International Association for Shell and Spatial Structures* 46 (Aug. 2005), pp. 77–84.

[37] Axel Kilian and John Ochsendorf. “Particle-spring systems for structural form finding”. In: *Journal of the International Association for Shell and Spatial Structures* 46 (Aug. 2005), pp. 77–84.

[38] Kitepower. *Kitepower Falcon*. https://thekitepower.com/product/. Accessed 24 April 2023. 2023.

[39] Eric van der Knaap. “A particle system approach for modelling flexible wings with inflatable support structures”. MA thesis. Delft University of Technology, 2023. URL: http://resolver.tudelft.nl/uuid:c77c5c6a-0bf7-47d5-b5bf-c5efac0c2d83.

[40] Rachel Leuthold. “Multiple-Wake Vortex Lattice Method for Membrane Wing Kites”. MA thesis. Delft University of Technology, 2015. URL: http://resolver.tudelft.nl/uuid:4c2f34c2-d465-491a-aa64-d991978fedf4.

[41] W. J. Lewis. “Computational form-finding methods for fabric structures”. In: *Proceedings of the Institution of Civil Engineers - Engineering and Computational Mechanics* 161.3 (2008), pp. 139–149. DOI: 10.1680/eacm.2008.161.3.139. eprint: https://doi.org/10.1680/eacm.2008.161.3.139. URL: https://doi.org/10.1680/eacm.2008.161.3.139.

[42] Xue Li et al. “A New Solution to Well-Known Hencky Problem: Improvement of In-Plane Equilibrium Equation”. In: *Mathematics* 8.5 (2020). ISSN: 2227-7390. DOI: 10.3390/math8050653. URL: https://www.mdpi.com/2227-7390/8/5/653.

[43] Tom Lolies et al. “Numerical Methods for Efficient Fluid–Structure Interaction Simulations of Paragliders”. In: *Aerotecnica Missili & Spazio* 98 (July 2019). DOI: 10.1007/s42496-019-00017-2.

[44] Miles L. Loyd. “Crosswind kite power (for large-scale wind power production)”. In: *Journal of Energy* 4.3 (1980), pp. 106–111. DOI: 10.2514/3.48021. URL: https://doi.org/10.2514/3.48021.

[45] Miles Macklin. *Implicit Springs*. https://blog.mmacklin.com/2012/05/04/implicitsprings/. Accessed 16 January 2023. 2012.

[46] Marijke M. Mollaert. *OLYMPIC GAMES 1972 (MUNICH): OLYMPIC STADIUM*. https://www.tensinet.com/index.php/projects-database/projects?view=project&id=3779. Accessed 12 July 2023. 2023.

[47] Mermaid Chart. *Mermaid Chart — The next evolution in smart diagramming for enterprise teams*. https://www.mermaidchart.com/. Accessed 25 Oct 2023. 2023.

[48] Thouraya Nouri-Baranger. “Computational methods for tension-loaded structures”. In: *Archives of Computational Methods in Engineering* 11.2 (2004), pp. 143–186. ISSN: 1886-1784. DOI: https://doi.org/10.1007/BF02905937. URL: https://link.springer.com/article/10.1007/bf02905937.

[49] M. Papadrakakis. “A method for the automatic evaluation of the dynamic relaxation parameters”. In: *Computer Methods in Applied Mechanics and Engineering* 25.1 (1981), pp. 35–48. ISSN: 0045-7825. DOI: https://doi.org/10.1016/0045-7825(81)90066-9. URL: https://www.sciencedirect.com/science/article/pii/0045782581900669.

[50] Ruy Pauletti and Paulo Pimenta. “The natural force density method for the shape finding of taut structures”. In: *Computer Methods in Applied Mechanics and Engineering* 197 (Sept. 2008), pp. 4419–4428. DOI: 10.1016/j.cma.2008.05.017.

[51] Jelle Poland. “Modelling aeroelastic deformation of soft wing membrane kites”. MA thesis. Delft University of Technology, 2022. URL: http://resolver.tudelft.nl/uuid:39d67249-53c9-47b4-84c0-ddac948413a5.

[52] Jelle A. W. Poland and Roland Schmehl. “Modelling Aero-Structural Deformation of Flexible Membrane Kites”. In: *Energies* 16.14 (2023). ISSN: 1996-1073. DOI: 10.3390/en16145264. URL: https://www.mdpi.com/1996-1073/16/14/5264.

[53] ProofWiki. *Equation of Catenary*. https://proofwiki.org/wiki/Equation_of_Catenary. Accessed 06 Oct 2023. 2022.

[54] Python Software Foundation. *timeit — Measure execution time of small code snippets*. https://docs.python.org/3/library/timeit.html#timeit.default_timer. Accessed 20 Oct 2023. 2023.

[55] Rachel Leuthold. *Java-Based Surf-Kite Flight Model for Application to a Kite-Surfing Simulator*. Accessed 27 Aug 2022, internal paper at Delft University of Technology. 2013.

[56] W. T. Reeves. “Particle Systems—a Technique for Modeling a Class of Fuzzy Objects”. In: *ACM Trans. Graph.* 2.2 (Apr. 1983), pp. 91–108. ISSN: 0730-0301. DOI: 10.1145/357318.357320. URL: https://doi.org/10.1145/357318.357320.

[57] Craig W. Reynolds. “Flocks, Herds and Schools: A Distributed Behavioral Model”. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 25–34. ISBN: 0897912276. DOI: 10.1145/37401.37406. URL: https://doi.org/10.1145/37401.37406.

[58] Marien Ruppert. “Development and validation of a real time pumping kite model”. MA thesis. Delft University of Technology, 2012. URL: http://resolver.tudelft.nl/uuid:56f1aef6-f337-4224-a44e-8314e9efbe83.

[59] Marien Ruppert. *PROPERTIES OF THE 4 MM DYNEEMA TETHER*. Tech. rep. TU Delft, 2012.

[60] H.-J. Schek. “The force density method for form finding and computation of general networks”. In: *Computer Methods in Applied Mechanics and Engineering* 3.1 (1974), pp. 115–134. ISSN: 0045-7825. DOI: https://doi.org/10.1016/0045-7825(74)90045-0. URL: https://www.sciencedirect.com/science/article/pii/0045782574900450.

[61] M. Schelbergen et al. “Clustering wind profile shapes to estimate airborne wind energy production”. In: *Wind Energy Science* 5.3 (2020), pp. 1097–1120. DOI: 10.5194/wes-5-1097-2020. URL: https://wes.copernicus.org/articles/5/1097/2020/.

[62] Roland Schmehl, ed. *Airborne Wind Energy: Advances in Technology Development and Research*. English. Green Energy and Technology. Springer Science+Business Media, 2018. ISBN: 978-981-10-1946-3. DOI: 10.1007/978-981-10-1947-0.

[63] Roland Schmehl, Uwe Ahrens, and Moritz Diehl. *Airborne Wind Energy*. Nov. 2013. ISBN: 978-3-642-39964-0. DOI: 10.1007/978-3-642-39965-7.

[64] Roland Schmehl and Oliver Tulloch, eds. *8th Airborne Wind Energy Conference (AWEC 2019): Book of Abstracts*. Glasgow, United Kingdom: Delft University of Technology, 2019.

[65] Martin Servin and Claude Lacoursière. “Massless Cable for Real-time Simulation”. In: *Comput. Graph. Forum* 26 (June 2007), pp. 172–184. DOI: 10.1111/j.1467-8659.2007.01014.x.

[66] Simon Greenwold. *PSystem: Particle Systems Plugin for Processing*. https://www.cs.rpi.edu/~cutler/gaudi/gaudi_from_simon/presentations/particleIntro/media/particle/index.html. Accessed 26 January 2023. 2015.

[67] Springer Nature Limited. *Numerical simulations articles from across Nature Portfolio*. https://www.nature.com/subjects/numerical-simulations#:~:text=A%20numerical%20simulation%20is%20a,as%20in%20most%20nonlinear%20systems.. Accessed 13 April 2023. 2023.

[68] Tuur Stuyck. “Cloth Simulation for Computer Graphics”. In: *Synthesis Lectures on Visual Computing* 10 (Aug. 2018), pp. 1–121. DOI: 10.2200/S00867ED1V01Y201807VCP032.

[69] The MathWorks, Inc. *Iterative Methods for Linear Systems*. https://www.mathworks.com/help/matlab/math/iterative-methods-for-linear-systems.html. Accessed 30 Sep 2023. 2023.

[70] The SciPy community. *scipy.sparse.linalg.bicgstab*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.bicgstab.html. Accessed 10 December 2022. 2023.

[71] Paul Thedens. “An integrated aero-structural model for ram-air kite simulations”. PhD thesis. Apr. 2022. DOI: https://doi.org/10.4233/uuid:16e90401-62fc-4bc3-bf04-7a8c7bb0e2ee.

[72] UCSC. *Solving the Simple Harmonic Oscillator*. http://scipp.ucsc.edu/~haber/ph5B/sho09.pdf. Accessed 12 April 2023. 2009.

[73] Uwe Fechner. *FreeKiteSim*. https://bitbucket.org/ufechner/freekitesim/src/master/. Accessed 4 July 2023. 2022.

[74] Uwe Fechner. *KiteSimulators*. https://github.com/aenarete/KiteSimulators.jl. Accessed 4 July 2023. 2023.

[75] Vahid Moosavi. *Force_Density_Method*. https://github.com/sevamoo/Force_Density_Method. Accessed 11 July 2023. 2023.

[76] D. Veenendaal and P. Block. “An overview and comparison of structural form finding methods for general networks”. In: *International Journal of Solids and Structures* 49.26 (2012), pp. 3741–3753. ISSN: 0020-7683. DOI: https://doi.org/10.1016/j.ijsolstr.2012.08.008. URL: https://www.sciencedirect.com/science/article/pii/S002076831200337X.

[77] Rolf van der Vlugt, Johannes Peschel, and Roland Schmehl. “Design and Experimental Characterization of a Pumping Kite Power System”. In: *Airborne Wind Energy*. Ed. by Uwe Ahrens, Moritz Diehl, and Roland Schmehl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 403–425. ISBN: 978-3-642-39965-7. DOI: 10.1007/978-3-642-39965-7_23. URL: https://doi.org/10.1007/978-3-642-39965-7_23.

[78] Rolf van der vlugt et al. “Quasi-steady model of a pumping kite power system”. In: *Renewable Energy* 131 (July 2019), pp. 83–99. DOI: 10.1016/j.renene.2018.07.023.

[79] H. A. van der Vorst. “Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems”. In: *SIAM Journal on Scientific and Statistical Computing* 13.2 (1992), pp. 631–644. DOI: 10.1137/0913035. URL: https://doi.org/10.1137/0913035.

[80] John Watchorn. “Aerodynamic Load Modelling for Leading Edge Inflatable Kites”. MA thesis. Delft University of Technology, 2023. URL: http://resolver.tudelft.nl/uuid:42f611a2-ef79-4540-a43c-0ea827700388.

[81] Wikpedia. *Catenary*. https://en.wikipedia.org/wiki/Catenary. Accessed 10 July 2023. 2023.

[82] Windpact. *Overview: Viscoelastic Material Models*. https://windpact.com/viscoelastic-material-models/. Accessed 09 Nov 2023. 2023.

[83] X - The Moonshot Factory. *Makani*. https://x.company/projects/makani/. Accessed 24 April 2023. 2023.

[84] Qiang Ye. *Preconditioning for Accurate Solutions of Linear Systems and Eigenvalue Problems*. 2017. arXiv: 1705.04340 [math.NA].

A

Code and settings

Appendix containing a copy of the source code, settings for test cases, and other relevant information required to reproduce the PS framework and results.
The code can also be found online at `https://github.com/ARBatchelor/3D_PS.git`.

A.1. Source code

The PS framework source code *particleSystem* is comprised of 6 Python files, which are presented below.

Force.py

```
"""
Child Abstract Base Class 'Force', for force objects to be instantiated in ParticleSystem
"""
from Msc_Alexander_Batchelor.src.particleSystem.SystemObject import SystemObject
from abc import abstractmethod


class Force(SystemObject):

    def __init__(self):
        super().__init__()
        return

    def __str__(self):
        return

    @abstractmethod
    def force_value(self):
        return
```

ImplicitForce.py

```
"""
Child Abstract Base Class 'ImplicitForce', for implicit force objects to be instantiated in
    ParticleSystem
"""
from Msc_Alexander_Batchelor.src.particleSystem.Force import Force
from Msc_Alexander_Batchelor.src.particleSystem.Particle import Particle
from abc import abstractmethod
from abc import abstractproperty

class ImplicitForce(Force):

    def __init__(self, p1: Particle, p2: Particle):
        self.__p1 = p1
        self.__p2 = p2
        super().__init__()
        return
```

```

    def __str__(self):
        return

    @abstractmethod
    def calculate_jacobian(self):
        return

    @property
    def p1(self):
        return self.__p1

    @property
    def p2(self):
        return self.__p2
```

Particle.py

```
"""
Child Class 'Particle', for particle objects to be instantiated in ParticleSystem
"""
from Msc_Alexander_Batchelor.src.particleSystem.SystemObject import SystemObject
import numpy as np
import numpy.typing as npt


class Particle(SystemObject):

    def __init__(self, x: npt.ArrayLike, v: npt.ArrayLike, m: float, fixed: bool):
        self.__x = np.array(x)
        self.__v = np.array(v)
        self.__m = m
        self.__fixed = fixed
        super().__init__()
        return

    def __str__(self):
        return f"Particle Object, position [m]: [{self.__x[0]}, {self.__x[1]}, {self.__x
            [2]}], " \
               f"velocity [m/s]: [{self.__v[0]}, {self.__v[1]}, {self.__v[2]}], mass [kg]: {
                   self.__m}" \
               f", fixed: {self.__fixed}"

    def update_pos(self, new_pos: npt.ArrayLike):
        if not self.__fixed:
            self.__x = np.array(new_pos)
        return

    def update_vel(self, new_vel: npt.ArrayLike):
        if not self.__fixed:
            self.__v = np.array(new_vel)
        return

    @property
    def x(self):
        return self.__x

    @property
    def v(self):
        return self.__v

    @property
    def m(self):
        return self.__m

    @property
    def fixed(self):
        return self.__fixed
```

ParticleSystem.py

```
"""
```

```
ParticleSystem framework
...
"""
import numpy as np
import numpy.typing as npt
from Msc_Alexander_Batchelor.src.particleSystem.Particle import Particle
from Msc_Alexander_Batchelor.src.particleSystem.SpringDamper import SpringDamper
from scipy.sparse.linalg import bicgstab


class ParticleSystem:
    def __init__(self, connectivity_matrix: npt.ArrayLike, initial_conditions: npt.ArrayLike,
                 sim_param: dict):
        """
        Constructor for ParticleSystem object, model made up of n particles
        :param connectivity_matrix: sparse n-by-n matrix, where an 1 at index (i,j) means
                                    that particle i and j are connected
        :param initial_conditions: Array of n arrays to instantiate particles. Each array
            must contain the information
                                   required for the particle constructor: [initial_pos,
                                       initial_vel, mass, fixed: bool]
        :param sim_param: Dictionary of other parameters required (k, l0, dt, ...)
        """
        self.__connectivity_matrix = np.array(connectivity_matrix)
        self.__k = sim_param["k"]
        self.__l0 = sim_param["l0"]
        self.__c = sim_param["c"]
        self.__dt = sim_param["dt"]
        self.__g = sim_param["g"]
        self.__n = sim_param["n"]

        self.__rtol = sim_param["rel_tol"]
        self.__atol = sim_param["abs_tol"]
        self.__maxiter = sim_param["max_iter"]

        # allocate memory
        self.__particles = []
        self.__springdampers = []
        self.__f = np.zeros((self.__n * 3, ))
        self.__jx = np.zeros((self.__n * 3, self.__n * 3))
        self.__jv = np.zeros((self.__n * 3, self.__n * 3))

        self.__instantiate_particles(initial_conditions)
        self.__m_matrix = self.__construct_m_matrix()
        self.__instantiate_springdampers()

        # Variables required for kinetic damping
        self.__w_kin = self.__calc_kin_energy()
        self.__w_kin_min1 = self.__calc_kin_energy()
        self.__w_kin_min2 = self.__calc_kin_energy()
        self.__vis_damp = True
        self.__x_min1 = np.zeros(self.__n, )
        self.__x_min2 = np.zeros(self.__n, )

        return

    def __str__(self):
        print("ParticleSystem object instantiated with attributes\nConnectivity matrix:")
        print(self.__connectivity_matrix)
        print("Instantiated particles:")
        n = 1
        for particle in self.__particles:
            print(f" p{n}: ", particle)
            n += 1
        return ""

    def __instantiate_particles(self, initial_conditions):
        for set_of_initial_cond in initial_conditions:
            x = set_of_initial_cond[0]
            v = set_of_initial_cond[1]
            m = set_of_initial_cond[2]
```

```
            f = set_of_initial_cond[3]
            self.__particles.append(Particle(x, v, m, f))
        return

    def __instantiate_springdampers(self):
        b = np.nonzero(np.triu(self.__connectivity_matrix))
        self.__b = np.column_stack((b[0], b[1]))
        for index in self.__b:
            self.__springdampers.append(SpringDamper(self.__particles[index[0]], self.
                __particles[index[1]],
                                       self.__k, self.__l0, self.__c, self.__dt))
        return

    def __construct_m_matrix(self):
        matrix = np.zeros((self.__n * 3, self.__n * 3))

        for i in range(self.__n):
            matrix[i*3:i*3+3, i*3:i*3+3] += np.identity(3)*self.__particles[i].m

        return matrix

    def __calc_kin_energy(self):
        v = self.__pack_v_current()
        w_kin = np.matmul(np.matmul(v, self.__m_matrix), v)      # Kinetic energy, 0.5
            constant can be neglected
        return w_kin

    def simulate(self, f_external: npt.ArrayLike = ()):
        if not len(f_external):
            f_external = np.zeros(self.__n * 3, )
        f = self.__one_d_force_vector() + f_external

        v_current = self.__pack_v_current()
        x_current = self.__pack_x_current()

        jx, jv = self.__system_jacobians()

        # constructing A matrix and b vector for solver
        A = self.__m_matrix - self.__dt * jv - self.__dt ** 2 * jx
        b = self.__dt * f + self.__dt ** 2 * np.matmul(jx, v_current)

        # checking conditioning of A and b
        # print("conditioning A:", np.linalg.cond(A))

        for i in range(self.__n):
            if self.__particles[i].fixed:
                A[i * 3: (i + 1) * 3] = 0        # zeroes out row i to i + 3
                A[:, i * 3: (i + 1) * 3] = 0     # zeroes out column i to i + 3
                b[i * 3: (i + 1) * 3] = 0        # zeroes out row i

        # BiCGSTAB from scipy library
        dv, _ = bicgstab(A, b, tol=self.__rtol, atol=self.__atol, maxiter=self.__maxiter)

        v_next = v_current + dv
        x_next = x_current + self.__dt * v_next

        # function returns the pos. and vel. for the next timestep, but for fixed particles
            this value doesn't update!
        self.__update_x_v(x_next, v_next)
        return x_next, v_next

    def kin_damp_sim(self, f_ext: npt.ArrayLike, q_correction: bool = False):       # kinetic
         damping alghorithm
        if self.__vis_damp:         # Condition resetting viscous damping to 0
            self.__c = 0
            self.__springdampers = []
            self.__instantiate_springdampers()
            self.__vis_damp = False

        if len(f_ext):              # condition checking if an f_ext is passed as argument
            self.__save_state()
```

```
138             x_next, v_next = self.simulate(f_ext)
139         else:
140             self.__save_state()
141             x_next, v_next = self.simulate()
142 
143         w_kin_new = self.__calc_kin_energy()
144 
145         if w_kin_new > self.__w_kin:    # kin damping algorithm, takes effect when decrease
                in kin energy is detected
146             self.__update_w_kin(w_kin_new)
147         else:
148             v_next = np.zeros(self.__n*3, )
149 
150             if q_correction:            # statement to check if q_correction is desired,
                    standard is turned off
151                 q = (self.__w_kin - w_kin_new)/(2*self.__w_kin - self.__w_kin_min1 -
                        w_kin_new)
152                 # print(q)
153                 # print(self.__w_kin, w_kin_new)
154                 # !!! Not sure if linear interpolation between states is the way to determine
                         new x_next !!!
155                 if q < 0.5:
156                     x_next = self.__x_min2 + (q / 0.5) * (self.__x_min1 - self.__x_min2)
157                 elif q == 0.5:
158                     x_next = self.__x_min1
159                 elif q < 1:
160                     x_next = self.__x_min1 + ((q - 0.5) / 0.5) * (x_next - self.__x_min1)
161 
162                 # Can also use this q factor to recalculate the state for certain timestep h
163 
164             self.__update_x_v(x_next, v_next)
165             self.__update_w_kin(0)
166 
167 
168         return x_next, v_next
169 
170     def __pack_v_current(self):
171         return np.array([particle.v for particle in self.__particles]).flatten()
172 
173     def __pack_x_current(self):
174         return np.array([particle.x for particle in self.__particles]).flatten()
175 
176     def __one_d_force_vector(self):
177         self.__f[self.__f != 0] = 0
178 
179         for n in range(len(self.__springdampers)):
180             fs, fd = self.__springdampers[n].force_value()
181             i, j = self.__b[n]
182 
183             self.__f[i*3: i*3 + 3] += fs + fd
184             self.__f[j*3: j*3 + 3] -= fs + fd
185 
186         return self.__f
187 
188     def __system_jacobians(self):
189         self.__jx[self.__jx != 0] = 0
190         self.__jv[self.__jv != 0] = 0
191 
192         for n in range(len(self.__springdampers)):
193             jx, jv = self.__springdampers[n].calculate_jacobian()
194             i, j = self.__b[n]
195 
196             self.__jx[i * 3:i * 3 + 3, i * 3:i * 3 + 3] += jx
197             self.__jx[j * 3:j * 3 + 3, j * 3:j * 3 + 3] += jx
198             self.__jx[i * 3:i * 3 + 3, j * 3:j * 3 + 3] -= jx
199             self.__jx[j * 3:j * 3 + 3, i * 3:i * 3 + 3] -= jx
200 
201             self.__jv[i * 3:i * 3 + 3, i * 3:i * 3 + 3] += jv
202             self.__jv[j * 3:j * 3 + 3, j * 3:j * 3 + 3] += jv
203             self.__jv[i * 3:i * 3 + 3, j * 3:j * 3 + 3] -= jv
204             self.__jv[j * 3:j * 3 + 3, i * 3:i * 3 + 3] -= jv
```

```
205
206         return self.__jx, self.__jv
207
208     def __update_x_v(self, x_next: npt.ArrayLike, v_next: npt.ArrayLike):
209         for i in range(self.__n):
210             self.__particles[i].update_pos(x_next[i * 3:i * 3 + 3])
211             self.__particles[i].update_vel(v_next[i * 3:i * 3 + 3])
212         return
213
214     def __update_w_kin(self, w_kin_new: float):
215         self.__w_kin_min2 = self.__w_kin_min1
216         self.__w_kin_min1 = self.__w_kin
217         self.__w_kin = w_kin_new
218         return
219
220     def __save_state(self):
221         self.__x_min2 = self.__x_min1
222         self.__x_min1 = self.__pack_x_current()
223         return
224
225     @property
226     def particles(self):            # Temporary solution to calculate external aerodynamic
        forces
227         return self.__particles
228
229     @property
230     def springdampers(self):
231         return self.__springdampers
232
233     @property
234     def stiffness_m(self):
235         self.__system_jacobians()
236         return self.__jx
237
238     @property
239     def f_int(self):
240         f_int = self.__f.copy()
241         for i in range(len(self.__particles)):
242             if self.__particles[i].fixed:
243                 f_int[i*3:(i+1)*3] = 0
244
245         return f_int
246
247     @property
248     def x_v_current(self):
249         return self.__pack_x_current(), self.__pack_v_current()
```

SpringDamper.py

```
1  """
2  Child Class 'SpringDamper', for spring-damper objects to be instantiated in ParticleSystem
3  """
4  from Msc_Alexander_Batchelor.src.particleSystem.ImplicitForce import ImplicitForce
5  from Msc_Alexander_Batchelor.src.particleSystem.Particle import Particle
6  import numpy as np
7
8
9  class SpringDamper(ImplicitForce):
10
11     def __init__(self, p1: Particle, p2: Particle, k: float, l0: float, c: float, dt: float):
12         self.__k = k
13         self.__c = c
14         self.__l0 = l0
15         self.__dt = dt
16         super().__init__(p1, p2)
17         return
18
19     def __str__(self):
20         return f"SpringDamper object, spring stiffness [n/m]: {self.__k}, rest length [m]: {
               self.__l0}\n" \
21                f"Damping coefficient [N s/m]: {self.__c}" \
22                f"Assigned particles\n  p1: {self.p1}\n  p2: {self.p2}"
```

```

    def __relative_pos(self):
        return np.array([self.p1.x - self.p2.x])

    def __relative_vel(self):
        return np.array([self.p1.v - self.p2.v])

    def force_value(self):
        return self.__calculate_f_spring(), self.__calculate_f_damping()

    def __calculate_f_spring(self):
        relative_pos = self.__relative_pos()
        norm_pos = np.linalg.norm(relative_pos)

        if norm_pos != 0:
            unit_vector = relative_pos / norm_pos
        else:
            unit_vector = np.array([0, 0, 0])

        f_spring = -self.__k * (norm_pos - self.__l0) * unit_vector
        return np.squeeze(f_spring)

    def __calculate_f_damping(self):
        relative_pos = self.__relative_pos()
        relative_vel = np.squeeze(self.__relative_vel())
        norm_pos = np.linalg.norm(relative_pos)

        if norm_pos != 0:
            unit_vector = np.squeeze(relative_pos / norm_pos)
        else:
            unit_vector = np.squeeze(np.array([0, 0, 0]))

        f_damping = -self.__c * np.dot(relative_vel, unit_vector) * unit_vector
        return np.squeeze(f_damping)

    def calculate_jacobian(self):
        relative_pos = self.__relative_pos()
        norm_pos = np.linalg.norm(relative_pos)

        if norm_pos != 0:
            unit_vector = relative_pos / norm_pos
        else:
            norm_pos = 1
            unit_vector = np.array([0, 0, 0])

        i = np.identity(3)
        T = np.matmul(np.transpose(unit_vector), unit_vector)
        jx = -self.__k * ((self.__l0 / norm_pos - 1) * (T - i) + T)

        jv = -self.__c*i

        return jx, jv
```

SystemObject.py

```
"""
Parent Abstract Base Class 'SystemObject', for objects to be instantiated in ParticleSystem
"""
from abc import ABC


class SystemObject(ABC):

    def __init__(self):
        return

    def __str__(self):
        return
```

A.2. BiCGSTAB Alghorithm

Unpreconditioned Bi-CGStAB alghorithm [79].

Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, choose $\boldsymbol{r}'_0$ such that $\boldsymbol{r}_0 \cdot \boldsymbol{r}'_0 \neq 0$
Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$
for $j = 0, 1, \cdots$ **do**
    $\alpha_j = (\boldsymbol{r}_j \cdot \boldsymbol{r}'_0) \,/\, ((A\boldsymbol{p}_j) \cdot \boldsymbol{r}'_0)$
    $\boldsymbol{s}_j = \boldsymbol{r}_j - \alpha_j A\boldsymbol{p}_j$
    $\omega_j = ((A\boldsymbol{s}_j) \cdot \boldsymbol{s}_j) \,/\, ((A\boldsymbol{s}_j) \cdot (A\boldsymbol{s}_j))$
    $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j + \omega_j \boldsymbol{s}_j$
    $\boldsymbol{r}_{j+1} = \boldsymbol{s}_j - \omega_j A\boldsymbol{s}_j$
    if $\|\boldsymbol{r}_{j+1}\| < \varepsilon_0$ **then**
        Break;
    end if
    $\beta_j = (\alpha_j/\omega_j) \times (\boldsymbol{r}_{j+1} \cdot \boldsymbol{r}'_0) \,/\, (\boldsymbol{r}_j \cdot \boldsymbol{r}'_0)$
    $\boldsymbol{p}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \,(\boldsymbol{p}_j - \omega_j A\boldsymbol{p}_j)$
end for
Set $\boldsymbol{x} = \boldsymbol{x}_{j+1}$

A.3. Simulation input parameters

Settings used for validation tests.

Table A.1: Input parameter values for shown simulations of validation cases

Parameter	Validation case 1	Validation case 2	Validation case 3
n [-]	5	5, 20	25
k_S [N/m]	119575.9	2e3	2933.3
c_d [N s/m]	92	100	100
l_{tether} [m]	5.14	10	300
ρ_{tether} [kg/m]	0.012	0.1	0.012
g [m/s^2]	9.807	9.807	N/A
m_{block} [kg]	327.8	N/A	N/A
$C_{D,t}$ [-]	N/A	N/A	1.22
h [s]	0.1, 0.0001	0.1	0.01
n_{steps} [-]	1000, 1000000	1000	10000
abs_tol [m/s]	1e-5	1e-5	1e-5
rel_tol [-]	1e-4	1e-5	1e-5
max_iter [-]	1e5	1e5	1e5
$\boldsymbol{F}_{res}$ [N]	1e-3	1e-3	1e-3

B

Additional results

B.1. Full derivations

Decay rate and phase shift for undamped harmonic oscillator

1. From second order differential equation tot system of two first order differential equations.
2. Approximation of this systems solution over time with implicit Euler scheme.
3. Rewrite to a recursive formulation (i.e. express $\boldsymbol{x}_n$ as a function of initial values $\boldsymbol{x}_0$).
4. Find eigenvalues of the matrix ($\lambda = 1 \pm ih\omega$) and use the property $\boldsymbol{Av} = \lambda \boldsymbol{v}$, to find the scalar value. Note that this assumes that the initial value is equal to the eigenvector belonging to the eigenvalue $\lambda = 1 - ih\omega$.
5. Raise the term before the initial value to an exponent.
6. Use $ln(1-a) = \int \frac{1}{1+a}$, and a Taylor expansion around the point $a = b$ to approximate the integral as a series: $f(a) + \sum_{n \geq 0} \frac{-1^n (a-b)^{n+1}}{(n+1)(1+b)^n}$. With b=0, this simplifies to $\sum_{n \geq 0} \frac{-1^n a^{n+1}}{(n+1)}$.
7. Use $t = hn$ and split the term into exponents with and without *i*.

$$\ddot{x} + \omega^2 x = 0, \quad \omega \equiv \sqrt{\frac{k}{m}} \tag{B.1}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix} \frac{d}{dt} = \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix} \tag{B.2}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n+1} - \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n} = h \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n+1} \tag{B.3}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n} = \left[\boldsymbol{I} - h \begin{pmatrix} 0 & 1 \\ -\omega^2 & 0 \end{pmatrix} \right]^{-n} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{0} \tag{B.4}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n} = (1 + ih\omega)^{-n} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{0} \tag{B.5}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n} = e^{-n * ln(1-a)} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{0} \tag{B.6}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n} = e^{-n \sum_{n \geq 0} \frac{-1^n a^{n+1}}{(n+1)}} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{0} \tag{B.7}$$

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{n} = e^{-it\omega(1-\frac{1}{3}\omega^2)} e^{-\frac{1}{2} * th\omega} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}_{0} \tag{B.8}$$

B.2. Code visualization

A behavioral diagram of the code can be seen in Figure B.1. A structural diagram is presented in Figure B.2. Both diagrams were made with the Mermaid Chart software [47].

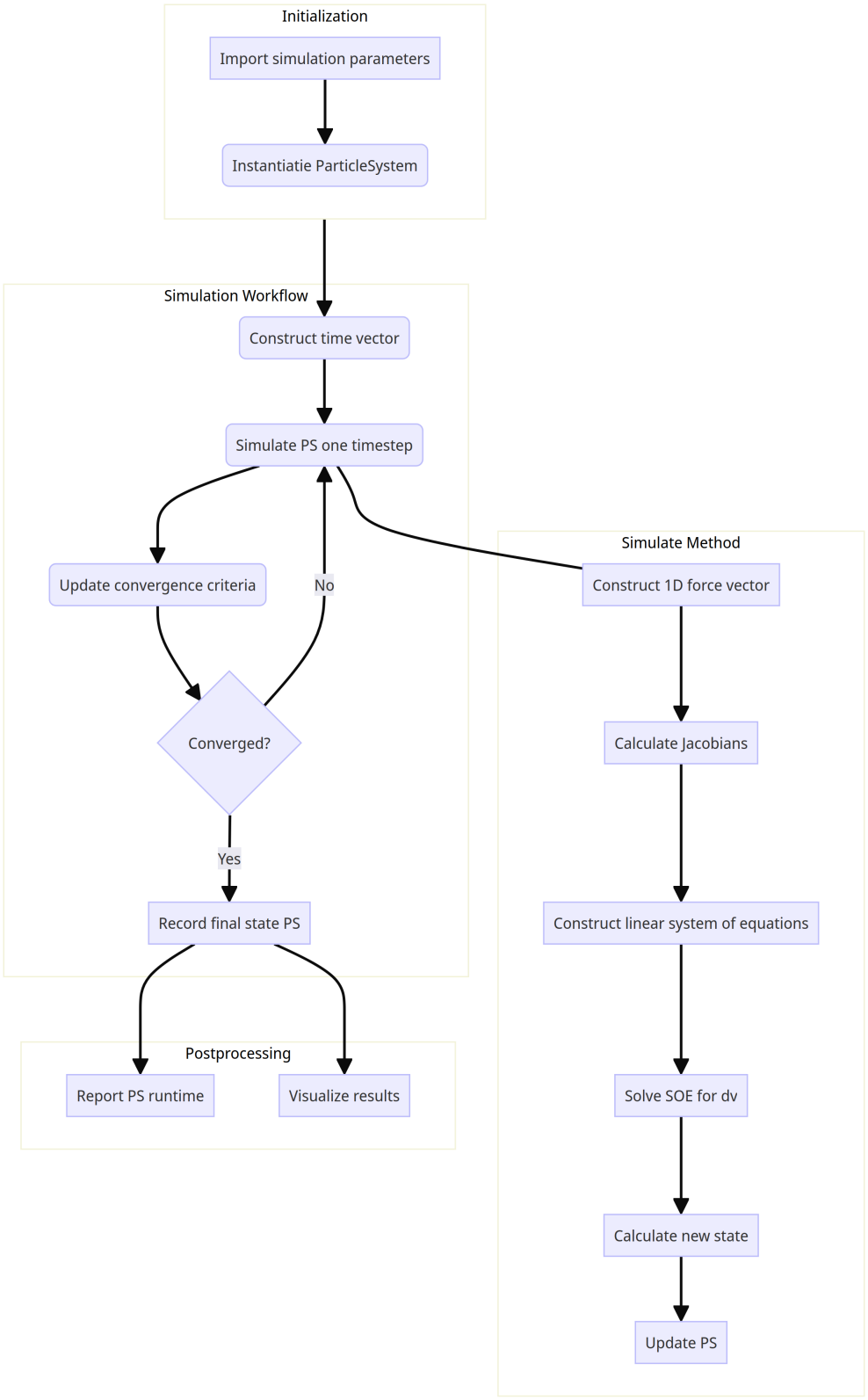


Figure B.1: Functional flow block diagram of the PS framework

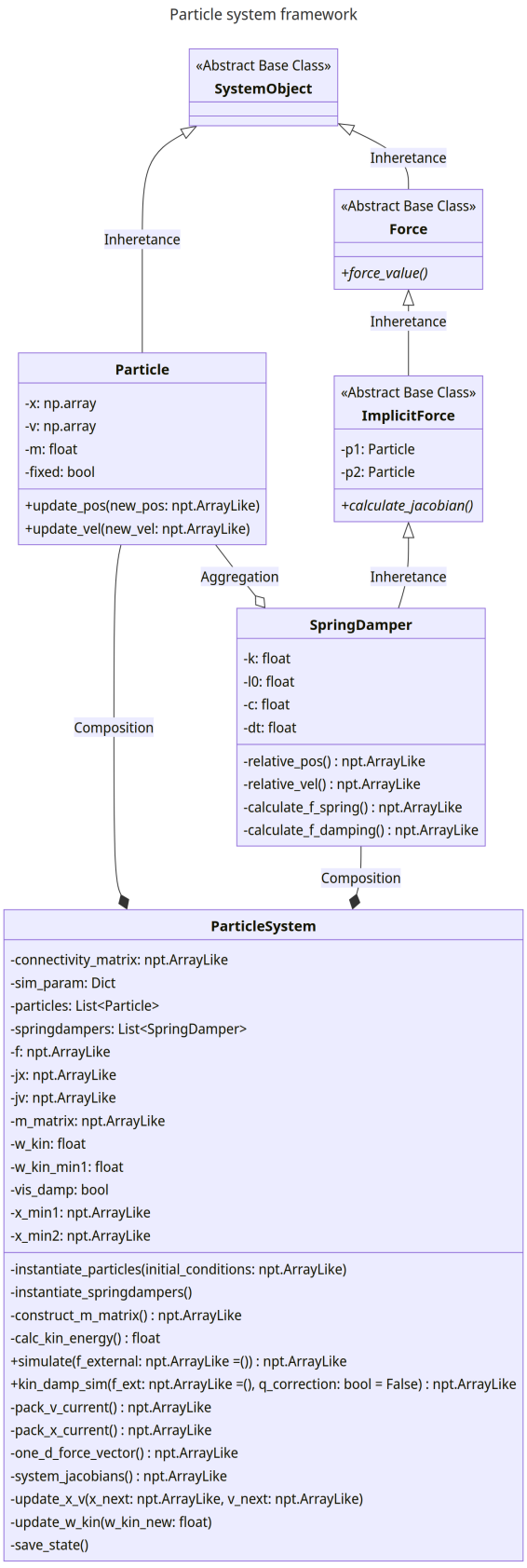


Figure B.2: UML class diagram of the PS framework

B.3. Hencky problem

Fichter [23] presented a solution for Hencky's problem in the form of Equation B.9.

$$W = q^{\frac{1}{3}} \sum_{0}^{\infty} a_{2n}(1 - \rho^{2n+2}) \tag{B.9}$$

The dimensionless quantities W and ρ can be related to the displacement w and radial coordinate c, respectively, with Equation B.10. Here, r is the radius of the membrane.

$$W = \frac{w}{r}, \quad \rho = \frac{c}{r} \tag{B.10}$$

The loading parameter q depends on the radius of the membrane, the Young's modulus of the material and the pressure p (Eq. B.11).

$$q = \frac{pa}{Ew} \tag{B.11}$$

To approximate a solution for W, the coefficients a_{2n} for n up to 10 were calculated by Fichter and can be seen in Equation B.12.

$$\begin{aligned}
a_0 &= \frac{1}{b_0} \\
a_2 &= \frac{1}{2\, b_0^4} \\
a_4 &= \frac{5}{9\, b_0^7} \\
a_6 &= \frac{55}{72\, b_0^{10}} \\
a_8 &= \frac{7}{6\, b_0^{13}} \\
a_{10} &= \frac{205}{108\, b_0^{16}} \\
a_{12} &= \frac{17051}{5292\, b_0^{19}} \\
a_{14} &= \frac{2864485}{508032\, b_0^{22}} \\
a_{16} &= \frac{103863265}{10287648\, b_0^{25}} \\
a_{18} &= \frac{27047983}{1469664\, b_0^{28}} \\
a_{20} &= \frac{42367613873}{1244805408\, b_0^{31}}
\end{aligned} \tag{B.12}$$

The following Matlab code was used to calculate numerical values of b_0, which depends on the Poisson's ratio of the material. In table B.1 resulting values of b_0 for a range of ν are listed.

```
clear all
clc

syms b_0
nu = 0;           % [-] Poisson's ratio
eqn = 0 == (1-nu) * b_0 - (3-nu) / (b_0 ^ 2) - (5 - nu) * 2 / (3 * b_0
    ^ 5) - (7 - nu) * 13 / (18 * b_0 ^ 8)     - (9 - nu) * 17 / (18 *
   b_0 ^ 11) - (11 - nu) * 37 / (27 * b_0 ^ 14) - (13 - nu) * 1205 /
   (567 * b_0 ^ 17)     - (15 - nu) * 219241 / (63504 * b_0 ^ 20) -
   (17 - nu) * 6634069 / (1143072 * b_0 ^ 23) - (19 - nu) * 51523763 /
```

```
        (5143824 * b_0 ^ 26) - (21 - nu) * 998796305 / (56582064 * b_0 ^
    29);
sol = solve(eqn, b_0);

numeric_sol = double(sol);
tolerance = 1e-10;  % Filter out solutions with an imaginary part
    defined as 0
real_solutions = numeric_sol(abs(imag(numeric_sol)) < tolerance)
```

Table B.1: Calculated values of b_0 for a range of ν

ν	b_0
0.0	1.6204
0.1	1.6487
0.2	1.6827
0.3	1.7244
0.34	1.7439
0.4	1.7769

Finally, the values of the material properties and geometry, except for ν, were taken from [1]. They are listed in Table B.2.

Table B.2: Remaining parameter values to fully define the Hencky problem

parameter	value
p	$100\,\mathrm{kPa}$
E	$311\,488\,\mathrm{Pa}$
a	$0.1425\,\mathrm{m}$
ν	0
w	$0.01\,\mathrm{m}$

B.4. Additional figures

Validation cases

Additional results of validation case 1.

Additional results of validation case 2.

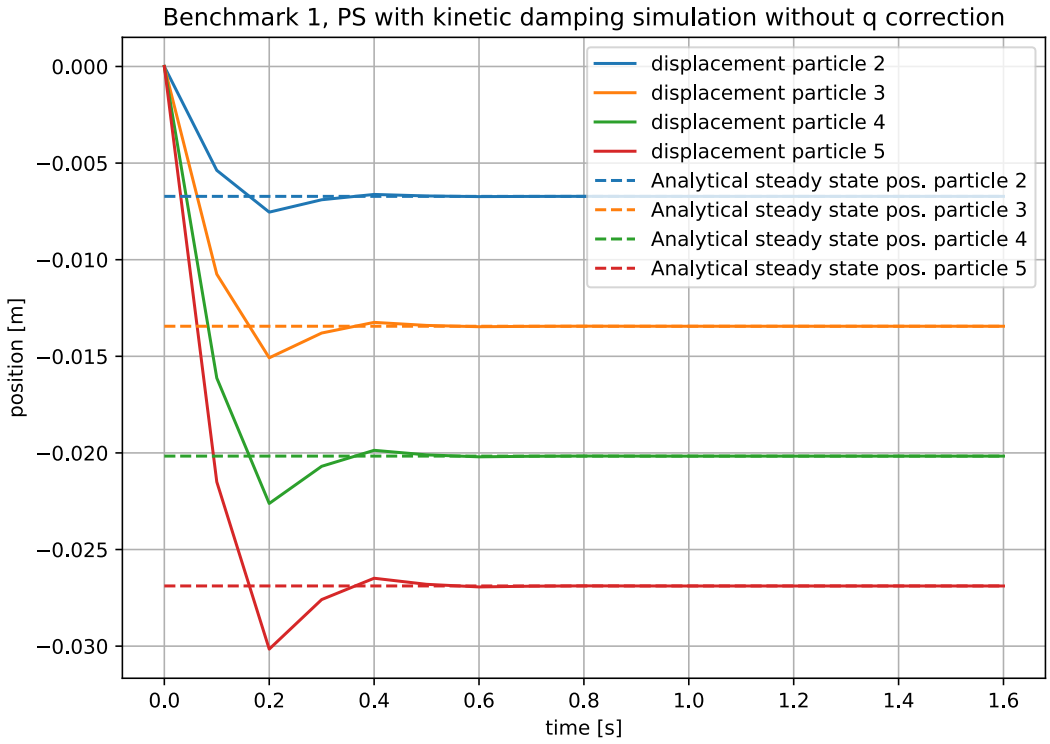


Figure B.3: Simulation of kinetically damped PS without q correction

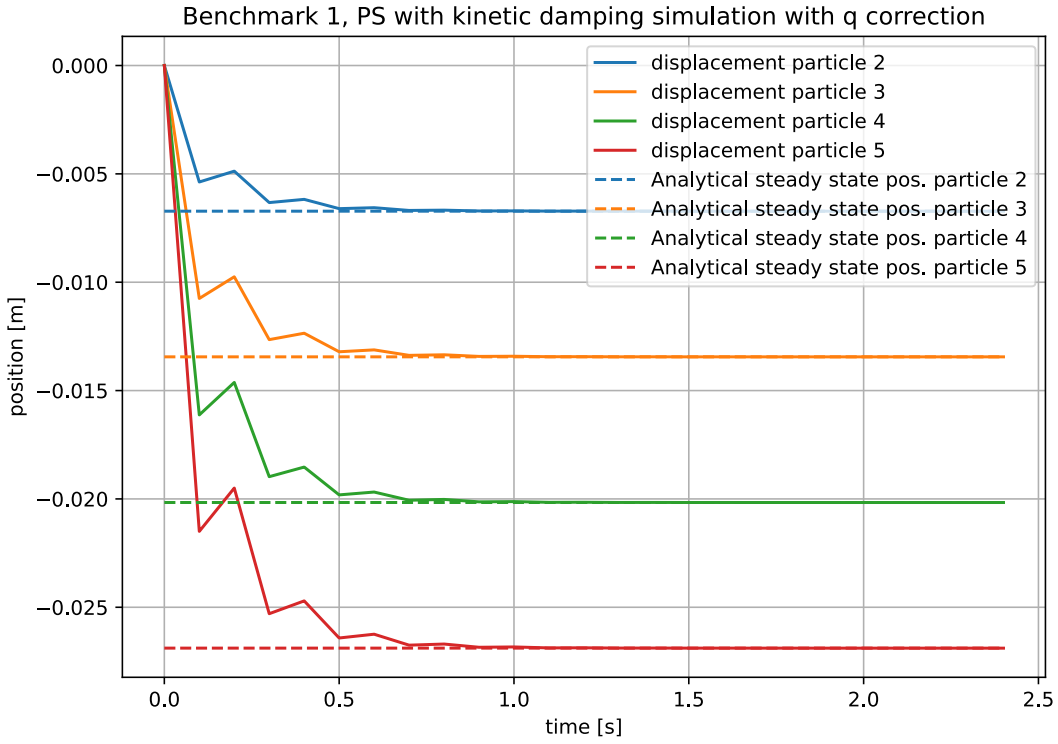
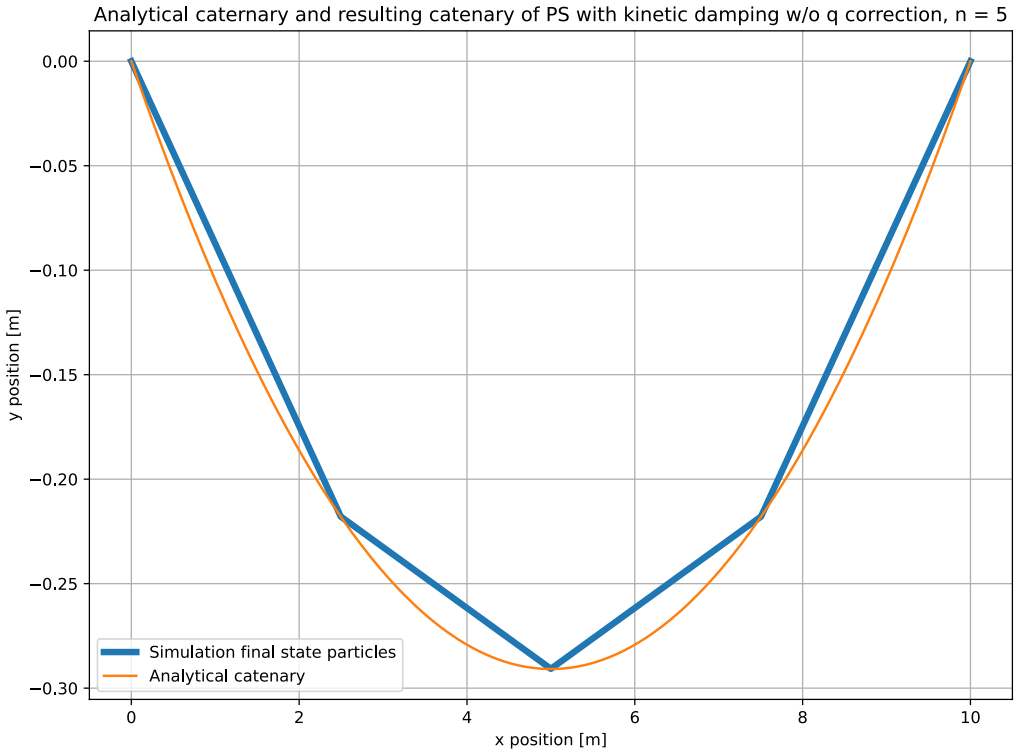


Figure B.4: Simulation of kinetically damped PS with q correction



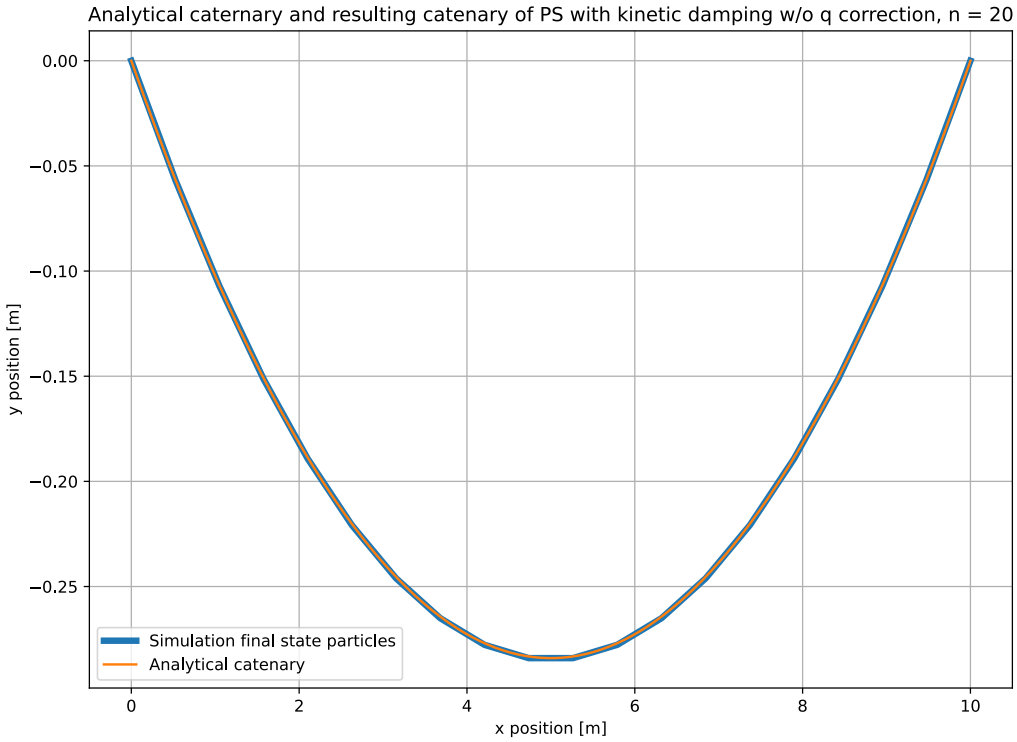


Figure B.5: Found catenary shape of PS with kinetic damping without q correction compared to analytical shape for 5 particles (upper) and 20 particles (lower).

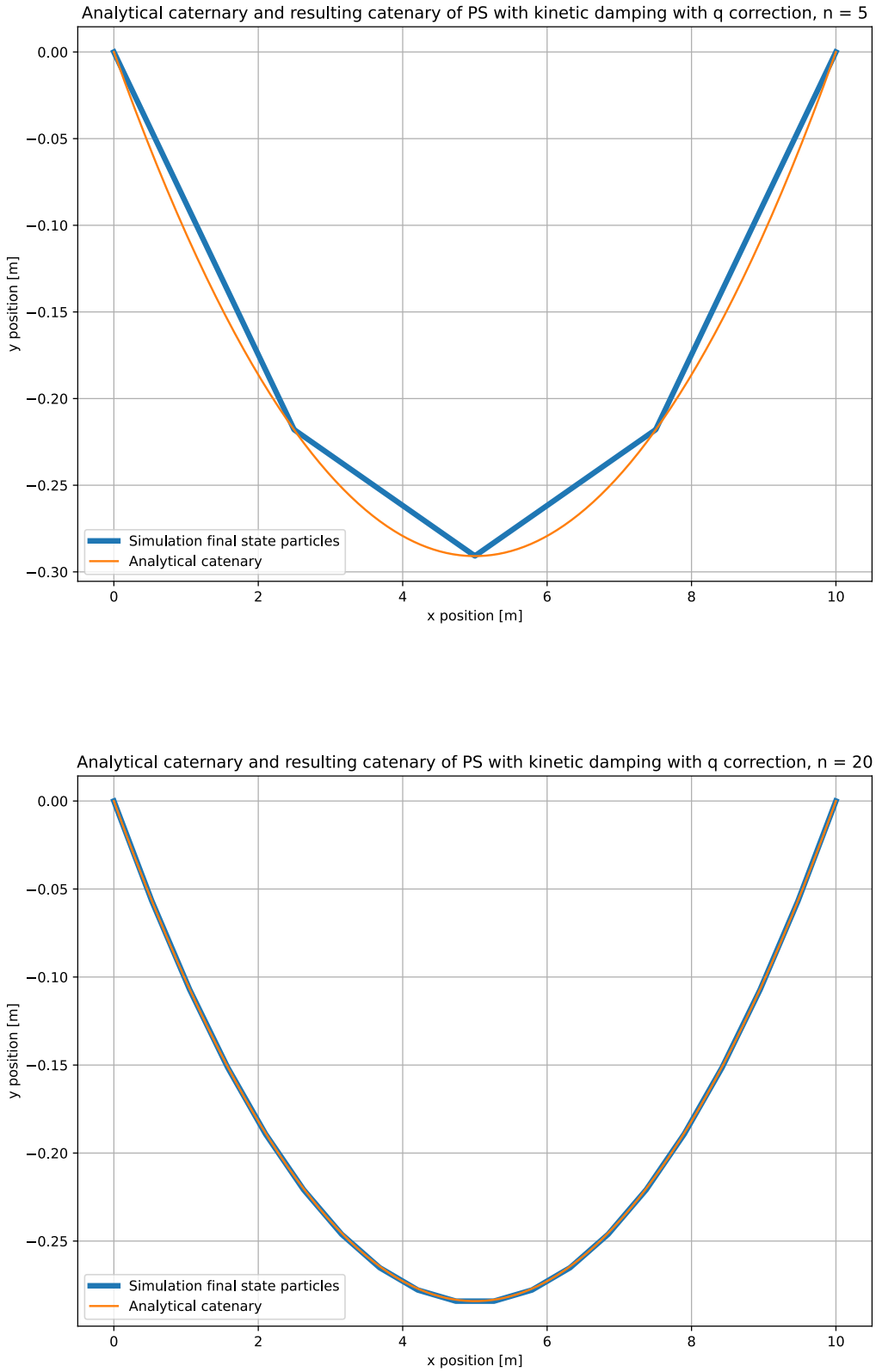


Figure B.6: Found catenary shape of PS with kinetic damping and q correction compared to analytical shape for 5 particles (upper) and 20 particles (lower).