Spring 8-2018

On the well-posedness and global boundary controllability of a nonlinear beam model

Jessie Jamieson
University of Nebraska - Lincoln, jdjamieson@huskers.unl.edu

ON THE WELL-POSEDNESS AND GLOBAL BOUNDARY CONTROLLABILITY OF A NONLINEAR BEAM MODEL

by

Jessie D. Jamieson

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Mathematics

Under the Supervision of Professor George Avalos

Lincoln, Nebraska

August, 2018

ON THE WELL-POSEDNESS AND GLOBAL BOUNDARY CONTROLLABILITY OF A NONLINEAR BEAM MODEL

Jessie D. Jamieson, Ph.D.

University of Nebraska, 2018

Adviser: George Avalos

The theory of beams and plates has been long established due to works spanning many fields, and has been explored through many investigations of beam and plate mechanics, controls, stability, and the well-posedness of systems of equations governing the motions of plates and beams. Additionally, recent investigations of flutter phenomena by Dowell, Webster et al. have reignited interest into the mechanics and stability of nonlinear beams. In this thesis, we wish to revisit the seminal well-posedness results of Lagnese and Leugering for the one dimensional, nonlinear beam from their 1991 paper, "Uniform stabilization of a nonlinear beam by nonlinear boundary feedback." To date, these remarkable results of Lagnese and Leugering are the only such well-posedness treatments for this particular model. We will provide a modified version of the proofs of well-posedness which will also elucidate details omitted in the original proofs, and we will place the treatment of the nonlinear coupling in the now-standard context of locally Lipschitz perturbations of m-accretive generators. Additionally, the second part of this thesis is dedicated to numerical experiments that illustrate stability and controllability properties of this nonlinear problem.

DEDICATION

I wish to dedicate this work to William and Rocket, whose unconditional love and support lifted me when I fell.

ACKNOWLEDGMENTS

The path I have taken toward my doctorate in mathematics has been long, draining, and sometimes tumultuous, but it would have been absolutely impossible had it not been for the love and support of many people. This section is dedicated to those folks; although it would be impossible to name and acknowledge them all. What follows is a (long winded) attempt to acknowledge those who have contributed to my success, organized somewhat chronologically.

Technically first, chronologically, I owe absolutely all of my success to the love and support of my family. There have been many times, especially recently, when things have not always gone according to plan. We have been angry at each other, upset, and even disappointed sometimes, but at the end of the day, we are still family. My sister, Katie, has always supported me and given me the motivation that I needed to push through hard times. Although I do not tell her enough, I want her to know that I love her madly, and that I am so amazingly proud of her, despite her hitting me in the head with a doll when I was about 9 years old, which I still have not forgotten about and have not forgiven her for. Who knows, perhaps that bang on the noggin made me smarter (I doubt it), in which case, I really do owe her everything. The only other thing to tell her is how sorry I am. Sometimes, people have put her in my shadow and I have never been okay with that. Katie is an absolutely strong, resilient, smart person on her own, and one of the people I look up to most. No one can take that from her, and that is something I want everyone to know. Secondly, I owe thanks to my dad, Doug. For anyone who does not know him, the best way to describe him is by calling him the epitome of dedication, hard work, and love. Had it not been for his teachings and lessons, I would absolutely not be the person I am today. To him, I owe everything, and I am so thankful to have had him by my side through every

success and setback. Nothing I could say here could capture how much he means to me, so, instead, I'll just say, "Yes, dad, I'm a 'Mellon', but don't call me Shirley." Finally, I wish to thank my mom, Lisa. Things could have turned out so differently in the end, but I cannot give thanks without thanking her for all of the hard work she put into my development and growth as a person. Thanks, mom; I love you, and I miss you– miss us– all the time.

While we are on the subject of family, I should acknowledge my second family, the Jamiesons. My husband, William, has been my absolute rock during this entire process and I know for a fact that I could not have made it through my PhD had it not been for his unconditional love and support (and coding expertise). His family, especially Rob and Mary Jane, have also unconditionally supported me. To Mary Jane, thank you for serving as a role model for a strong, trail-blazing, professional woman. You have been an inspiration.

Next, throughout my K-12 education, I had some remarkable teachers and mentors who took a special interest in my success and education. I absolutely believe that had it not been for Jeannie Saunders quizzing me on my multiplication tables every morning and afternoon both to and from school, I would not have ever had as much interest in mathematics as I ended up with. I still love those wintergreen mints you kept in your van. Thanks, Coach.

In high school, my mathematical interest really took hold; I had some fantastic teachers- Jeff Kinsler, Chris Hayes, and Scott Bolton (and Brian McLaughlin), but I owe the most thanks to one of the best mentors and women I know, Mrs. Linda Sedlack. She took me under her wing when she did not have to, reinforced my drive and dedication to success and hard work, and was a shining example of a mother, friend, and teacher. Mrs. Sedlack, thanks for all that you did for me, for convincing me to go to college, for convincing me that I was capable of anything, and for showing

me what being a teacher and mentor is all about.

During my time at East Tennessee State University, some of the professors there took a special interest in my success; had it not been for the love and support of the faculty there, I would not have even considered attending graduate school. Dr. Anant Godbole rejecting me from his REU probably started the whole thing. Had he not rejected me, I would not have worked as hard; I took the rejection as a challenge and wanted to prove to him that I could keep up with his REU kids. Sometimes I could not, but he loved me anyway. The REU mentor, Bill Kay, ended up as one of my absolute best friends, and, as a friend who had been through the whole grad school thing, he gave me (and continues to give) support and an outside perspective sometimes when things went (go) wrong. Also, I cannot mention REUs without acknowledging Pete Johnson at Auburn University for his love and support. Speaking of Auburn, thanks to (Rev.?) Robert Gardner at ETSU, too (Archimedes!). I wish to also acknowledge Jeff Knisley for convincing me that python and machine learning were worth investing time into. Next, special thanks go out to Mark Giroux, Beverly Smith, Richard Ignace, and the rest of my ETSU physics family. Although I was not technically a physics major, you all adopted me as one of your own and **almost** convinced me to switch to physics. Almost.

The glaring omission from this list is Teresa Haynes, my undergraduate thesis advisor. I feel like nothing I could say here could possibly express how thankful I am for her guidance, support, trust, and the belief she put in me as a student and mathematician. From the talk I gave for her at the JMM in Boston to the papers we published together by the end of my senior year, Dr. Haynes believed in me when no one else did (including myself). She is probably the most important reason I decided to go to graduate school and pursue the doctorate, and she without a doubt shaped me into the researcher, writer, and person I am today. Thank you, Dr. Haynes, for

everything, including the mugs with Stephen Hedetniemi's face on them.

Last, but certainly not least, thanks to my ETSU friends for the good times: Brandon Sexton, Chance Brown, Matt Cannon, John Burdette, Olivia Miller, and especially, Erin Middlemas. We had some awesome times, especially if it was Matt's birthday, or we needed to get into the lab (shoutout to cbible182). Erin, you've proven to be one of the best friends I could ask for, despite the distance. I always enjoy our chats and visits, and every one picks up right where we left off, as if there was no separation at all. That's friendship, and for that, I'm thankful.

Finally, my time at UNL has brought with it its own version of ups and downs, but I would have been incapacitated by mental breakdowns had it not been for two groups of people, in particular. Ariel "Aubrey" Setniker and Laura "Brittany" White have been two of the best girlfriends a girl could imagine. From late night target runs to cry dates at each others' apartments, I would have been completely lost without you two. The other group of friends fell on the other end of the spectrum, my trivia guys: Nick Packauskas, Josh Pollitz, and Matt Reichenbach. You three have been the best guy friends I have ever had; I am going to miss trivia and hanging out with each of you dearly (and the basketball games, Josh). Thanks for being there for me through everything.

One of the last groups of people I owe the most to are my ORNL/Team Bridges family: John Goodall, Masha Vincent, Kelly Huffer, Joel Reed, Ben Thomas, Vlad Protopopescu, and especially Bobby Bridges. Each of you shaped my future, even if you do not realize you did, during my time at ORNL. I so very much appreciate your welcoming me onto the team and including me in your projects (and thank you, Vlad, for taking special interest in my dissertation!). Although my current trajectory takes me away from ORNL, I am sure I will return and that there will be projects we can work together on.

Finally, I owe a certainly substantial amount of thanks to my dissertation committee: Petronela Radu, George Avalos, Adam Larios, Hendrik Viljoen, and especially Daniel Toundykov. Your help and guidance through this long process has been invaluable and you have helped me and my career more than you may know. I appreciate all of the attention and advice each of you have given me over the last five years, and I know that each of you will impact the careers of others just as you have me.

In particular, thank you, Daniel, for continuing to advise me in your absence, even though it was absolutely not required. Your continuing support and advice has been one of the things continually motivating me to continue, and, sometimes, our video chats were the only thing that would have me laughing the entire week. I just hope you have enjoyed working with me as much as I have enjoyed (and miss) working with you.

GRANT INFORMATION

Jessie D. Jamieson was supported by National Science Foundation Graduate Research Fellowship under Grant No. 25-0517-0143-002. Any conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of the National Science Foundation.

Table of Contents

List of Figures

Chapter 1

Introduction

Many foundational mathematical models describing physical phenomena can be written as a first-order evolution equation

$$\frac{dy}{dt}(t) = A(y(t)), \quad y(0) = y_0$$

where t is time, y describes the stage of the system as encoded by an element of a suitably structured function space, and A is an operator on the function space. Thus, an ordinary differential equation or a system thereof, can be studied on a finite dimensional state space with $y(t) \in \mathbb{R}^d$. As an example, consider the simple harmonic oscillator, consisting of a weight attached to the end of a spring. The opposite end of the spring is attached to a rigid wall or support; if the mass is displaced from the equilibrium position, the spring will exert a restoring elastic force obeying Hooke's Law of Motion, which states that the force needed to extend or compress a spring by some distance scales linearly with respect to that distance. That is, $\mathbf{F} = -k\mathbf{x}$. Considering Newton's Second Law of Motion, when written as a first order equation,

the motion of the oscillator is governed by the system

$$\frac{d}{dt}\begin{pmatrix} x \\ \dot{x} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & 0 \end{pmatrix}\begin{pmatrix} x \\ \dot{x} \end{pmatrix},$$

where $x(t)$ is the displacement of the mass from equilibrium at time t, k is the spring constant, and m is the mass at the end of the spring. This ODE system does, indeed, take the form $\dot{y}(t) = A(y(t))$.

In turn, the state of a partial differential equation at each moment in time describes a function of other variables. This function can be considered as an element of an appropriate (infinite-dimensional) space which often, guided by both practicality and tractability, can be endowed with a suitable Hilbert or Banach structure. In the case of ODE systems, the fundamental solution is expressible by a group of linear operators given by matrix exponentials $t \mapsto e^{tA}$.

In many important cases, these results formally extend to infinite dimensions though the matrix exponential representation and must be considered in either a spectral or generalized form. Unlike matrix exponentials, these generalizations may represent a family of non-invertible operators (whence a group is typically referred as the semigroup). An example to which this is especially relevant is the heat equation; if $y(x,t)$ is the temperature at a point x at time t in a thin, homogeneous, laterally insulated rod, then the distribution of heat in the rod over time may be modeled by the equation

$$\frac{\partial y}{\partial t} = \alpha\ y_{xx},$$

where α is the thermal diffusivity. One method to show the existence of unique solutions to the heat equation (with suitable initial conditions) is to consider the operator $A = \partial_{xx}$, with an appropriate domain. It can be shown that the operator

A is the generator of a semigroup of contractions, which is the solution of the initial value problem. In this example, as is the case with many other applications, the operator A is linear; suitable generalizations exist even when A is nonlinear.

We will view such models from the perspective of mathematical control theory. Historically, famous examples of control engineering date back to waterclocks, *clepsydra*, of ancient Greece. These clocks would measure the passage of time using the flow of water either into or out of a vessel, where the height of water in the vessel was a function of time measured since the water started flowing. Early designs (c. 250 BCE) of in-flow clepsydra by the mathematician Ctesibius of Alexandria were involved enough to account for the effects of water height on the flow rate of water leaving the vessel. This system is considered one of the first instances of a system with feedback [May70]. Thanks to the feedback control system, these clocks remained the most accurate time measurement devices until Christiaan Huygens invented the pendulum clock in 1656. Formal analysis of controls began in 1868 when the physicist James Clerk Maxwell described the theoretical basis for the operation of centrifugal governors. In his seminal work [Max68], Maxwell presented differential equations of motion for various types of governors and moderators and supplied, for the first time, stability analysis for a feedback-controlled system of differential equations; such stability analysis is commonly the goal in control theoretical problems. In our context, the problem acquires the form

$$\frac{d}{dt}y = Ay + Bu$$

where the term u and operator B describe forces that are not intrinsic to the system in its simplest form. Indeed, Bu could represent friction, a force which stabilizes an otherwise conservative system, or it could represent artificial actuators built into the

system to achieve a desired behavior. As in these cases, the control function u at time t could depend on the state $y(t)$ (closed-loop control) or may be prescribed externally (open-loop control).

The control-theoretic framework can be formulated on a general state space and naturally encompasses both the finite and infinite dimensional, in either the linear or nonlinear context. While finite dimensional theory offers more flexible tools due to the less restrictive space structure, the infinite dimensional framework is naturally more general and is the primary focus of the modern applied control theory research.

1.1 Infinite-dimensional linear theory

When the evolution operator, e.g., as denoted A above, is linear, the system acquires structure. Mathematical control theory for such models has been very well-developed; focusing on infinite-dimensional theory, seminal results can be found in the works of David Russell [Rus73, DR77, LR99], including his early survey of the field in [Rus78], books by John Lagnese and Jacques-Louis Lions [LL88], Vilmos Komornik [Kom94], Andrey Fursikov [Fur00], Irena Lasiecka and Roberto Triggiani [LT00a, LT00b, Las02]. A comprehensive literature survey would be more exhaustive, yet these references provide a through guide to the primary foundations of the field.

In the context of applied PDEs, many questions are loosely centered around three classes of models, reflected in the structure of operator A:

1. Hyperbolic or hyperbolic-like systems. Such systems describe phenomena including, but not limited to, wave propagation, dynamic elasticity, and buckling of plates. Basic linear models for such processes often result in reversible dynamics (the semigroup of operators generated by the evolution operator A is actually a group). For such linear systems, exact controllability, from any ini-

tial state to any target, is equivalent to null-controllability, that is: steering the solution from any state to the origin.

2. Diffusion processes, e.g., heat transfer. Such processes often yield irreversible dynamics. An example would be the prescription of L^2 data to a heat equation on a rod resulting in subsequent C^∞ states; this is mathematically reflected in the analyticity property of the evolution semigroup generated by A. Generally, for diffusive systems the question of exact controllability on the state space can only be resolved approximately, since, for example, the aforementioned heat solution states can never exactly match an element of L^2. Hence, studies often focus on null-controllability of such systems (steering them to zero).

3. Coupled hyperbolic/parabolic systems. Coupled systems exhibit a more complex range of behaviors and may satisfy more general results than any of the system's constituents would have admitted individually.

For results on null-controllability of linear thermoelastic systems see, for example, I. Lasiecka and R. Triggiani [LT98, Tri03], G. Avalos and I. Lasiecka [AL03, AL04, AL05, Ava06]. In G. Avalos [Ava00], the author proves that a linear thermoelastic Kirchhoff model is exactly controllable, whether formulated without rotational terms (which yield parabolic dynamics) or with them (hyperbolic-like dynamics). For exact controllability results on Maxwell's equation of electromagnetism refer to Lagnese [Lag89], Kapitonov [Kap94], Zhou [Zho97], Zhang [Zha00], Phung [Phu00], Eller and Masters [EM02]; a more general result on boundary controllability of Maxwell's equations in anisotropic media was derived by Eller in [Ell07]. The leading results and an overview of the theory concerning observability and controllability of linear Schrödinger equations can be found in papers by Triggiani et al. [LTZ04a, LTZ04b, Tri07, TX07, Tri08].

An extensive treatise on control and stability theory for waves, plates and shells, as well as a comprehensive reference on applications of differential-geometric methods to these problems can be found in the monograph by Yao [Yao11]. For an extensive overview of numerical techniques in control of distributed-parameter systems see the book by Glowinski et al. [GLH08].

1.2 Nonlinear local controllability problems

When the evolution operator A is nonlinear, the control framework becomes much more restrictive. A natural approach would be to first consider perturbation and linearization techniques. Hence many results on the topic focus on controllability within a small set, typically a neighborhood of a stable equilibrium, or otherwise impose smallness assumptions on the parameters of the system. A discussion of pioneering developments in nonlinear control theory can be found in the survey [BD87] by Balachandran and Dauer, as well as in Russell [Rus78, Sec. 9a]. A more modern exposition of nonlinear control methods is given in the comprehensive treatise [Cor07] by Coron.

A vast body of literature has also been devoted to approximate and null controllability of Navier-Stokes equations. For example, see an overview by Fernández-Cara [FC99], Imanuvilov [Ima01] (and many references therein), Guerrero et. al [GIP12], Barbu [Bar03], Shorygin [Sho04], Fursikov [Fur06], Agrachev and Sarychev [AS06, AS05], Chapouly [Cha09]. The list here is certainly not exhaustive.

In the context of 1-dimensional PDE models in vibrational dynamics, we point out Lagnese's seminal investigation [Lag91] of the reachability set for a semilinear beam, which is one of the core references for this thesis. Numerous other approaches to such models have been studied over the years. For in-between equilibria controllability of

a nonlinear elastic string equation see Schmidt [Sch02]. Leiva in [Lei05] established a controllability theorem for a suspension bridge model under certain smallness restrictions imposed on the system. More recently Leugering and Schmidt addressed near-equilibrium controllability of a network of elastic strings in [LS12]. For a broader exposition of the controllability theory for 1D quasilinear hyperbolic models see the monograph by Li [Li10].

Within the paradigm of nonlinear plate dynamics, Liu in [Liu98] proved local controllability near the origin of a semilinear plate model. Exact boundary controllability near zero for the von Kármán plate equation was first established by Lagnese [Lag90]. Avalos subsequently demonstrated null-controllability of the thermoelastic von Kármán model in [Ava06]. Naso [Nas05] derived a controllability result for a thermoelastic plate proving the ability to steer the dynamics towards target states on certain trajectories. Local controllability near the origin for the Berger plate was later verified by Cîndea and Tuscnak in [CT09].

1.3 Global exact nonlinear controllability with arbitrary data

Fewer developments address *global* exact controllability of hyperbolic and hyperbolic-like dynamics without any "smallness" restrictions, or reliance on quasi-static deformations (in-between equilibria steering). Nonlinear terms in the form of a general *globally Lipschitz* feedback map were first addressed in the seminal work by Lasiecka and Triggiani [LT91, Tri05, LT05] with applications to boundary controllability of semilinear waves and plates. See also a more detailed exposition in Triggiani [Tri10]. Global exact controllability results from an interior patch were derived by Li and Zhang in [LZ00] for a semilinear wave problem, and by Zhang in [Zha01] for a semilinear plate equation. In both these latter papers the nonlinear terms in the form of

source feedbacks are allowed to grow super-linearly at infinity (hence stronger than Lipschitz condition), yet only by a sub-logarithmic margin: $o\big(|s|\sqrt{\ln|s|}\big)$ as $|s| \to \infty$.

The first application to general nonlinearities for plate systems was proposed in [ET15] where the authors generalize the methodology of [Tri10] to apply to arbitrary conservative nonlinear forcing terms. The resulting steering trajectory is glued together out of many pieces that nudge solutions between energy levels slowly enough so that a Lipschitz approximation of the system is possible.

Another fundamental strategy for global controllability goes back to the works of Russell and is prompted by the fact that for a linear system the null-controllability is equivalent to global exact controllability, provided the dynamics is time-reversible. In certain scenarios the same can be repeated for nonlinear systems. For some applications to nonlinear models see, e.g., Russell [Rus78, p. 722] or [ET15, p. 2486]. Our numerical investigations below will employ precisely this strategy.

1.4 New Interest in Nonlinear Beam Models

New interest in the nonlinear beam model examined herein is due to recent developments in the theory of flutter. The flutter phenomenon is one of self-excitation of a structure in the presence of a (not necessarily periodic) fluid flow; that is, under the right conditions, a structure that is nominally stable may become unstable. In particular, for a flexible structure, such as a beam, cycles of excitation may continue, resulting in a periodic motion strong enough to cause destruction or failure.

The canonical example of these destructive forces is that of the Tacoma Narrows Bridge. This bridge, whose construction completed in 1940, spanned Washington's Puget Sound, and, on windy days, was said to "dance"; waves would ripple across the bridge deck, constructed of steel and concrete. Unfortunately, by 9:00am November

7th, 1940, in the presence of winds up to 40 miles per hour, the bridge started twisting instead of rippling. By 10:30 that morning, the bridge had started ripping itself apart; the bridge totally collapsed 8 minutes later.

The leading, and incorrect, reasoning for the bridge's failure was that the wind prevailing across the bridge induced a wave motion which matched the internal resonances of the bridge; however, further investigations, culminating in a 2006 paper [GU06], yielded the discovery that the winds moving across the bridge did, in fact, induce a periodic oscillation, but not at resonant frequency. The bridge's response to the wind– her self-excitation and aeroelastic instability (flutter)– was her downfall.

As in the above example, flutter is a potentially dangerous phenomenon which is relevant to many industries: civil, mechanical, and architectural engineering, aerodynamics, and rocket science, briefly. To contrast, engineers have recently investigated a more positive consequence of flutter: the potential for power generation. Aeroelastic energy harvesters rely on flutter to transform oscillations of piezoelectric materials into electrical energy [DSMD11]. As with any power source, efficiency is key; to improve efficiency of such systems, it is necessary to understand the flutter phenomenon as completely as possible. Although some of the first papers on flutter were published out of the Cold War Era as a product of rocket and missile design investigations [flu57], we are finding that even the newest, most advanced models have issues fully emulating the flutter phenomenon, and that this is an area of research which could use more attention (see [TGD15], [HTW17], and the references therein). Key insights may be gained in this direction by focusing on one-dimensional beam models, many of which are still extremely challenging (again, see [TGD15]); there is a strong precedent to refresh and reexamine nonlinear beam models, which are some of the goals of this thesis.

1.5 Contributions and Goals of This Thesis

We wish to revisit the seminal well-posedness results of Lagnese and Leugering for the one dimensional, nonlinear beam [LL91]; to date, the remarkable results of Lagnese and Leugering are the only such well-posedness treatments. We will provide a modified version of the proofs of well-posedness which will also:

1. elucidate details omitted in the original proofs, such as the well-posedness of linear components and explicit maximality analysis for the in-plane subsystem.

2. focus on the setting of Kato's theorem (Theorem 4.4.1, [Sho97]) by identifying Hilbert spaces with their duals and using the framework of accretive operators on Hilbert spaces, rather than monotone operators from a space to its dual.

3. place the treatment of the nonlinear coupling in the now-standard context of locally Lipschitz perturbations of m-accretive generators.

Additionally, the second part of this thesis is dedicated to numerical experiments that illustrate stability and controllability properties of this nonlinear problem. In particular, we demonstrate

1. well-posedness and energy conservation in the unforced instance.

2. energy dissipation with linear and nonlinear damping.

3. global exact controllability of the system using a combination of reversed feedback controls and an approximation of exact control near the origin via linearization.

Chapter 2

Model Introduction and Transforming to an Abstract System

In the chapters which follow, we adhere to the notation of Lagnese and Leugering [LL91]; we refer the reader to the aforementioned publication for a more complete derivation of the model.

Consider a uniform prismatic beam of length L. Assume that the motion of this beam is planar and irrotational, and assume the aforementioned beam (in reference configuration) occupies the region whose rectangular coordinates are given by

$$\{(x, y, z)|\ 0 \leq x \leq L,\ -1 \leq y \leq 1,\ -h/2 \leq z \leq h/2\ \}.$$

The measurements $u(x,t)$ and $w(x,t)$ represent the longitudinal and lateral displacements of the point $(x, 0, 0)$, respectively, at time t. Furthermore, we assume the beam is clamped on the left end, i.e., $u(0,t) = w(0,t) = 0$ for all t, and free on the right end. The model herein considers in-plane and out-of-plane dynamics; in particular, $(u,\ u_t)$ corresponds to in-plane dynamics, while $(w,\ w_t)$ corresponds to out-of-plane dynamics. Furthermore, the model is extensible, that is, as the beam is deflected, local arc length is not necessarily conserved. This property manifests itself in the following term:

$$s'(x,t) = \sqrt{(1 + u'(x,t))^2 + (w'(x,t))^2} - 1,$$

where $s(x,t)$ is the axial stretching within the beam. A reasonable linearization of this expression may be taken by using the approximation $\sqrt{1+x} \approx 1 + x/2$, yielding

$$s'(x,t) = u'(x,t) + \frac{1}{2}\left(w'(x,t)\right)^2 + \frac{1}{2}\left(u'(x,t)\right)^2. \tag{2.1}$$

Finally, under the assumption that lateral beam displacement is small with respect to its length, we may identify the axial stretching with the first two terms of (2.1). This allows us to take

$$s'(x,t) = u'(x,t) + \frac{1}{2}\left(w'(x,t)\right)^2.$$

With this recurring term in mind, we appeal to the calculations of [LL91] and present the following homogeneous beam system and the associated boundary conditions:

$$\begin{cases} \rho A u_{tt} - EA(u' + \frac{1}{2}w'^2)' = 0 \\ \rho A w_{tt} - \rho I w''_{tt} + EI w'''' - EA\left[w'(u' + \frac{1}{2}w'^2)\right]' = 0 \\ u(0,t) = w(0,t) = w'(0,t) = 0, \\ EA(u' + \frac{1}{2}w'^2)(L,t) = 0, \\ EI w''(L,t) = 0, \\ \left[EI w''' - \rho I w'_{tt} - EA w'(u' + \frac{1}{2}w'^2)\right](L,t) = 0. \end{cases} \tag{2.2}$$

In (2.3), ρ is the mass density per unit volume of the beam in its reference configuration, $A = 2h$ is the cross sectional area of the beam, E is Young's modulus, and I is the beam's moment of inertia with respect to the y-axis. For the purposes of this model, each of these are constants.

Both boundary feedback controls and open loop controls will also be considered. For boundary controls, it is assumed that u_t, w_t, and the bending rate $(w_t)'$ can be measured at the boundary $x = L$ for any and all $t > 0$. These quantities, when measured, will be fed back through continuous functions, labeled G_1, G_2, and G_3, at the boundary. Thus, the closed loop system is

$$\begin{cases} \rho A u_{tt} - EA(u' + \frac{1}{2}w'^2)' = 0 \\ \rho A w_{tt} - \rho I w''_{tt} + EI w'''' - EA\left[w'(u' + \frac{1}{2}w'^2)\right]' = 0 \\ u(0,t) = w(0,t) = w'(0,t) = 0, \\ EA(u' + \frac{1}{2}w'^2)(L,t) = G_1, \\ EI w''(L,t) = G_2, \\ \left[EI w''' - \rho I w'_{tt} - EA w'(u' + \frac{1}{2}w'^2)\right](L,t) = G_3. \end{cases} \tag{2.3}$$

with initial conditions

$$\{u(0), u_t(0)\} = \left\{u^0, u^1\right\}, \quad \{w(0), w_t(0)\} = \left\{w^0, w^1\right\}.$$

Observe that if $G_i = 0$, then the system is unforced, while if G_i are prescribed, nonzero functions, then our system contains a control; in the case that G_i are determined by the state of the system, our control is a feedback control. On the other hand, open loop controls are independent of the state of the system.

2.1 Function Spaces

With the model in mind, we wish to next transform our system into an abstract evolution problem on appropriate function spaces. To this end, we introduce the

following spaces:

$$\begin{aligned} H &= L^2(0,L), \\ H_L^1 &= \left\{ v \;\middle|\; v \in H^1(0,L),\ v(L) = 0 \right\}, \\ {}_0H^1 &= \left\{ v \;\middle|\; v \in H^1(0,L),\ v(0) = 0 \right\}, \\ {}_{cl}H^2 &= \left\{ v \;\middle|\; v \in H^2(0,L),\ v(0) = v'(0) = 0 \right\}, \end{aligned}$$

with respective equivalent scalar products

$$\begin{aligned} (u,v) &= \int_0^L u(x)v(x)\ dx, \\ (u,v)_1 &= (u,v) + \gamma^2 \int_0^L u'(x)v'(x)\ dx, \\ (u,v)_2 &= \gamma^2 \int_0^L u''(x)v''(x)\ dx. \end{aligned}$$

Using our notation, left subscripts indicate boundary conditions on the left; in particular, functions in ${}_{cl}H^2$ have clamped boundary conditions on the left, meaning that functions in this space and their derivatives have value zero at the left endpoint, while ${}_0H^1$ indicates functions take the value zero on the left boundary. In a similar vein, right subscripts indicate that functions in the space take value zero at $x = L$. Additionally, for our problem we are working in the domain $\Omega = [0, L]$. Unless otherwise stated, assume spaces are taken with $\Omega = [0, L]$, as this notation may be suppressed.

Functionally, we may further inspect our system by considering the dual spaces of the H spaces above. Let $({}_0H^1)^*$ and $({}_{cl}H^2)^*$ be the dual spaces of ${}_0H^1$ and ${}_{cl}H^2$ respectively, with respect to H. This gives rise to the following containment:

$${}_{cl}H^2 \subset {}_0H^1 \subset H \subset ({}_0H^1)^* \subset ({}_{cl}H^2)^*.$$

Finally, the most relevant product space to our evolution problem is the product space

$$\mathscr{H} = (H \times H) \times ({}_{cl}H^2 \times {}_0H^1),$$

the state space for our system. Using this notation, we may denote an element of $\mathscr{H}$, for example, as $\{Z, W\}$, with $Z = \{z_0, z_1\} \in H \times H$ and $W = \{v, w\} \in {}_{cl}H^2 \times {}_0H^1$.

2.2 Operators

The issue of wellposedness of our system will be resolved by appealing to the theory of semigroups and locally Lipschitz perturbations of accretive operators. To this end, we define the following operators:

- $A_L = \partial_x$: this operator maps

  $$\mathcal{D}(A_L) = H^1_L \subset L^2 \to L^2$$

  by taking a partial derivative, that is, $A_L f = f_x$. The L^2 adjoint of this operator is $A_L^* g = -g_x$, with

  $$A_L^* : \mathcal{D}(A_L^*) = {}_0H^1 \subset L^2 \to L^2.$$

- $L = -\partial_{xx}$: this operator maps

  $$\mathcal{D}(L) = \{\phi \in H^2 : \ \phi(0) = 0, \ \phi_x(L) = 0\} \subset L^2 \to L^2.$$

  The operator L is self-adjoint on L^2 and positive definite. Because this operator is self-adjoint and positive, the spectrum contains only isolated, non-negative (and in this case, strictly positive) eigenvalues. We may define fractional powers

of the operator L. Further, one may describe the action of L as $(Lf, g) = (f_x, g_x)$; note that this expression defines an inner product on ${}_0H^1$, and

$$L : \mathcal{D}(L^{1/2}) = {}_0H^1 \to ({}_0H^1)^* = \mathcal{D}(L^{1/2})^*.$$

- $C = (I + \rho L)$, for $\rho > 0$: the domain of this operator coincides with that of L, with $Cf = f - \rho f_{xx}$. Like L, C is also self-adjoint on L^2 and positive definite. Also like L, we may also define fractional powers of C. We may view the action of C as $(Cf, g) = (f, g) + \rho(f_x, g_x)$, which also defines an inner product on ${}_0H^1$. The domain and range of C matches that of L, in that

$$C : \mathcal{D}(L^{1/2}) = {}_0H^1 \to ({}_0H^1)^* = \mathcal{D}(L^{1/2})^*.$$

  Additionally, we may write

$$(Cf, g) = (C^{1/2}C^{1/2}f, g) = (C^{1/2}f, C^{1/2*}g) = (C^{1/2}f, C^{1/2}g).$$

- $B = \partial_x^4$: this operator maps

$$\mathcal{D}(B) = {}_{cl}H^2 \cap \left\{ f \in H^4 : \ f_{xx}(L) = f_{xxx}(L) = 0 \right\} \to L^2.$$

  Like the other operators above, B is positive definite and self-adjoint on L^2. For this operator, the action may be viewed as $(Bf, g) = (f_{xx}, g_{xx})$; this product extends (by density) to $f \in {}_{cl}H^2$ and defines an equivalent inner product on ${}_{cl}H^2$:

$$B : {}_{cl}H^2 = \mathcal{D}(B^{1/2}) \to ({}_{cl}H^2)^* = (\mathcal{D}(B^{1/2}))^*.$$

Because we may also define fractional powers of B, $B^{1/2}$ is defined and self-adjoint, defining the equivalent H^2 inner product

$$(f_{xx}, g_{xx}) = (Bf, g) = (B^{1/2}B^{1/2}f, g) = (B^{1/2}f, B^{1/2*}g) = (B^{1/2}f, B^{1/2}g).$$

One may also write $(Bf, g) = -(f_{xxx}, g_x)$, taking care to notice that in this form,

$$B : {}_{cl}H^2 \cap \{f \in H^3 : \ f_{xx}(L) = 0\} \to ({}_0H^1)^*.$$

2.2.1 Extension Operators

Recall that we wish for the system to incorporate boundary controls; the framework to incorporate these controls relies on the construction of explicit trace maps to handle boundary values appropriately.

To this end, consider a vector with two scalar components, regarded formally as an element of $L^2(\{x = L\})$, so that

$$h = N \begin{bmatrix} a \\ b \end{bmatrix}.$$

If it must be the case that $h_x = 0$ and $h(L) = b$, then

$$N \begin{bmatrix} a \\ b \end{bmatrix} = b$$

This gives rise to the trace map for our z-system.

Proposition 1. *Let $g \in \mathscr{D}(A_L^*)$. Then*

$$(A_L N)^* g = \begin{bmatrix} 0 \\ -g(L) \end{bmatrix}.$$

That is, $N^* A_L^*$ is the trace map from Ω to $\{x = L\}$.

Proof. Let $\mathbf{v} = [a, b]^T$ be a fixed vector for any two real numbers a and b. Then

$$(g, A_L N \mathbf{v}) = (-g_x, N\mathbf{v}) = (-g_x, b) = -g(L)b$$

giving that

$$(A_L^* N^*, \mathbf{v})_{\mathbb{R}^2} = \begin{bmatrix} 0 \\ -g(L) \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

□

The construction of this trace operator for the G_1 control gives rise to the function $G_z = g_1(\mathrm{tr}(\cdot)) = g_1(N^* A_L^* \cdot)$, where

$$G_z : \begin{bmatrix} a \\ b \end{bmatrix} \mapsto \begin{bmatrix} 0 \\ -g_1(b) \end{bmatrix}$$

for any monotone increasing function g_1 for which $g_1(0) = 0$. This function will play a part in the construction of the evolution generator for our system.

A second trace map must be constructed to incorporate the G_2 and G_3 controls. Like before, for a vector with two scalar components a and b, if $h_{xxxx} = 0$, h satisfies the clamped-left boundary condition, $h_{xx}(L) = a$, and $h_{xxx}(L) = b$, then it must be

the case that

$$\tilde{N}\begin{bmatrix} a \\ b \end{bmatrix} = \frac{a - bL}{2}x^2 + \frac{b}{6}x^3. \tag{2.4}$$

This gives rise to the trace map for the w-system.

Proposition 2. *Let $g \in \mathscr{D}(B^*)$. Then*

$$(B_{ext}\tilde{N})^* g = \begin{bmatrix} g_x(L) \\ -g(L) \end{bmatrix}.$$

That is, $\tilde{N}^* B_{ext}^*$ is the trace map Ω to $\{x = L\}$.

Proof. Let $\mathbf{v} = [a, b]^T$ be a fixed vector for any two real numbers a and b. Because $\tilde{N}v$ has zero fourth derivative by construction, we use the B_{ext} map to get

$$\begin{aligned}
(g, B_{ext}\tilde{N}\mathbf{v}) &= (g_{xxxx}, \tilde{N}\mathbf{v}) \\
&= \left(g_{xxxx}, \frac{a - bL}{2}x^2 + \frac{b}{6}x^3\right) \\
&= \left(g_{xxxx},\ \frac{a}{2}x^2\right) + \left(g_{xxxx},\ \frac{b}{6}x^3 - \frac{bL}{2}x^2\right) \\
&= -a\left(g_{xxx},\ x\right) - \frac{b}{2}\left(g_{xxx},\ x^2 - 2Lx\right) \\
&= a(g_{xx},\ 1) + b(g_{xx},\ x - L) \\
&= (g_x(L) \cdot a) - (g(L) \cdot b),
\end{aligned}$$

giving that

$$(B_{ext}\tilde{N}^* g, \mathbf{v})_{\mathbb{R}^2} = \begin{bmatrix} g_x(L) \\ -g(L) \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix}$$

□

As above, this trace operator also gives rise to the nonlinear map G_w, whose action

is prescribed by

$$G : \begin{bmatrix} a \\ b \end{bmatrix} \mapsto \begin{bmatrix} -m(-a) \\ -g_2(b) \end{bmatrix}$$

for any monotone increasing functions m and g_2 for which $g_2(0) = m(0) = 0$.

Chapter 3

Wellposedness

One goal of this thesis is to explicitly show wellposedness of the full system (2.3) including the nonlinear boundary feedback controls. In particular, we prove the existence of strong and weak solutions to (2.3). The proof of the existence of weak and strong solutions relies on the representation of the dynamics of the system as a semi-flow with a dissipative evolution generator. We emphasize here that the original wellposedness result is due to Lagnese and Leugering [LL91]. Our work provides details omitted in the literature, and an alternative proof built upon the framework of m-accretive operators. To this end, we will describe an evolution generator for the system, which will include a locally Lipschitz perturbation. First, we will introduce an independent change of variables.

3.1 Change of Variables

Consider the independent change of variables which sets $\gamma^2 = I/A$ and takes $t \mapsto t\sqrt{\rho/E}$. This transforms the full controlled system into the form

$$\begin{cases} u_{tt} - \left(u' + \frac{1}{2}w'^2\right)' = 0, \\ w_{tt} - \gamma^2 w_{tt}^{"} + \gamma^2 w^{''''} - \left[w'\left(u' + \frac{1}{2}w'^2\right)\right]' = 0, \\ u(0,t) = w(0,t) = w'(0,t) = 0, \\ \left(u' + \frac{1}{2}w'^2\right)(L,t) = G_1 \\ \gamma^2 w"(L,t) = G_2 \\ \left[\gamma^2\left(w''' - w'_{tt}\right) - w'\left(u' + \frac{1}{2}w'^2\right)\right](L,t) = G_3, \end{cases} \tag{3.1}$$

Observing the recurring $s(x,t)$ term in the above equations inspires a substitution. Take

$$\begin{aligned} z_0 &= u' + \frac{1}{2}w'^2 \\ z_1 &= u_t, \\ v &= w_t \\ w &= w. \end{aligned}$$

Using this change, observe that the first equation of (3.1) above becomes $z_{1t} - z_0' = 0$. Additionally, directly calculating z_{0t} gives the equations

$$\begin{aligned} z_{1t} - z_0' &= 0, \\ z_{0t} - z_1' - v'w' &= 0, \end{aligned}$$

for $0 < x < L$. Appealing to the other conditions in the above system, we may rewrite the system taking advantage of our variable change, so that our full controlled system

becomes

$$\begin{cases} z_{0t} - z_1' - v'w' = 0, \\ z_{1t} - z_0' = 0 \\ v_t - \gamma^2 v_t'' + \gamma^2 w'''' - [w'z_0]' = 0, \\ w_t - v = 0, \end{cases} \tag{3.2}$$

with

$$\begin{cases} z_1(0,t) & = 0, \\ z_0(L,t) & = -g_1\left(z_1(L,t)\right), \\ v(0,t) & = w(0,t) = w'(0,t) = 0, \\ \gamma^2 w''(L,t) & = -m\left(v'(L,t)\right), \\ \left[\gamma^2\left(w''' - v_t'\right) - w'z_0\right](L,t) & = g_2\left(v(L,t)\right). \end{cases} \tag{3.3}$$

At this point, we are introducing more precise formulations of feedback controls on the right boundary. We also pass the variable change onto the initial conditions. The initial conditions for the system after the change of variables are

$$\begin{cases} z_0(x,0) & = (u^0)'(x) + \frac{1}{2}\left[(w^0)'(x)\right]^2 =: z_0^0(x), \\ z_1(x,0) & = u^1(x) =: z_1^0(x), \\ v(x,0) & = w^1(x) =: v^0(x), \\ w(x,0) & = w^0(x) =: w^0(x) \end{cases}$$

for $0 < x < L$.

Statements of wellposedness and other relevant results will henceforth refer to system (3.2-3.3) as it is written above. No other substitutions or variable changes will be introduced.

3.2 Statement of Wellposedness

Using the notation from the previous chapter, the full nonlinear, coupled system is of the form

$$\dot{\begin{bmatrix} z_0 \\ z_1 \\ w \\ v \end{bmatrix}} = \mathbb{A} \begin{bmatrix} z_0 \\ z_1 \\ w \\ v \end{bmatrix} + F \tag{3.4}$$

where

$$\mathbb{A} = \begin{bmatrix} & A_L & & \\ A_L & A_L N G_z(N^* A_L^* \cdot) & & \\ & & & I \\ & & -C^{-1}B & -C^{-1}\tilde{N} G_w(\tilde{N}^* B^* \cdot) \end{bmatrix}$$

and

$$F = \begin{bmatrix} v_x w_x \\ 0 \\ 0 \\ C^{-1}[w_x z_0]_x \end{bmatrix},$$

with $\mathcal{D}(\mathbb{A}) = H_L^1 \times {}_0H^1 \times H_w \times {}_{cl}H^2$, where

$$H_w := \left\{ w \in H^3 \cap {}_{cl}H^2 : \ \left[w + \tilde{N} G\left(\tilde{N}^* B^* v \right) \right]_{xx} = 0 \right\}.$$

Writing the system in this way allows us to define the notions of weak and strong solutions.

Definition 1 (Weak solution)**.** *We say $\{Z, W\}$ is a weak solution of (3.2)-(3.3) on*

the interval $[0,T)$*, including* $T=\infty$*, if*

$$Z=\{z_0,\ z_1\}\in C\left([0,T);H\times H\right),\quad W=\{w,\ v\}\in C\left([0,T);{}_0H^1\times {}_{cl}H^2\right).$$

In addition, for every $\psi_0\in H^1$*,* $\psi_1\in {}_0H^1$*,* $\phi_1,\ \phi_2\in {}_{cl}H^2$*, we have*

$$\begin{cases}\frac{d}{dt}\left(z_0,\psi_0\right)-\left(z_1,\psi_0'\right)-\left(v'w',\psi_0\right)=0,\\ \frac{d}{dt}\left(z_t,\psi_1\right)-\left(z_0,\psi_1'\right)+g_1(z_1(L,t))\psi_1(L)=0\\ \frac{d}{dt}\left[\left(v,\phi_1\right)+\gamma^2\left(v',\phi_1'\right)\right]+\gamma^2\left(w'',\phi_1''\right)+\left(w'z_0,\phi_1\right)\\ \qquad\qquad +g_2(v(L,t))\phi_1(L)+m(v'(L,t))\phi_1'(L)=0,\end{cases}\tag{3.5}$$

where d/dt *is taken in the distributional sense. Moreover,*

$$z_0(0,x)=z_0^0(x),\quad z_1(0,x)=z_1^0(x),\quad \forall x\in(0,L)$$
$$w(0,x)=w^0(x),\quad v(0,x)=v^0(x),\quad \forall x\in(0,L).$$

Note that (3.5) follows by formally multiplying (3.2) by the proper test functions, integrating by parts, and applying the conditions in (3.5). Likewise, we may also define a more regular solution.

Definition 2 (Strong solution)**.** *We say a solution* $\{Z,W\}$ *of (3.2)-(3.3) on* $[0,T)$ *is strong if* $\{Z,W\}$ *is a weak solution, and*

$$Z=\{z_0,\ z_1\}\in L^\infty\left(0,T;H_L^1\times {}_0H^1\right),\quad W=\{w,\ v\}\in L^\infty\left(0,T;H_w\times {}_{cl}H^2\right),$$

and

- $\{Z,W\}$ *is Lipschitz on* $[0,T]$*,*

- $\{Z(t), W(t)\} \in \mathcal{D}(\mathbb{A})$, $t \geq 0$,
- $\{Z, W\}$ *is strongly right differentiable on* $[0, T)$,
- $\{Z, W\}$ *is weakly differentiable and* $\{Z_t, W_t\}$ *is weakly continuous on* $(0, T)$.

These definitions allow us to state the following wellposedness result.

Theorem 1. *There is a nonlinear semigroup flow* S_t *on* $\mathscr{H}$ *such that if* $y_0 \in \mathscr{H}$, *then* $t \to S_t(y_0)$ *is a weak solution. Moreover, if* $y_0 \in \mathcal{D}(\mathbb{A})$, *then* $S_t(y_0)$ *is a strong solution. In addition, if* $y_0 \in \mathscr{H}$ *and* $(y_n) \to y_0$ *in* $\mathscr{H}$, *then strong solutions* $S_t(y_n)$ *congerge to* $S_t(y_0)$ *in* $C([0, T]; \mathscr{H})$ *for every* $T > 0$.

The proof of this result proceeds in the following steps:

1. Show that the generators for the independent w and z system components each satisfy m-accretivity conditions on appropriate state spaces.
2. Show that the coupling term F is a locally Lipschitz perturbation.
3. Demonstrate that the full model is conservative.
4. Appeal to nonlinear semigroup theory.

To this end, we begin by providing results on the accretivity of operators for this system.

3.3 Proof of the Main Result

Before proceeding with the proof, we define the notions of accretivity and m-accretivity, following [Sho97].

Definition 3. *For H a Hilbert space, a (possibly multivalued) operator $A : \mathcal{D}(A) \subset H \to 2^H$ is called accretive if*

$$(w_1 - w_2, x_1 - x_2)_H \geq 0$$

for all x_1 $x_2 \in \mathcal{D}(A)$ and for all $w_1 \in A(x_1)$, $w_2 \in A(x_2)$.

Although the definition of accretivity stated above is for possibly multivalued operators, we note that the operators we consider as part of this work are all single valued.

Definition 4. *An accretive operator A is m-accretive if $Range(I + \lambda A) = H$ for some $\lambda > 0$.*

3.3.1 Accretivity

We must show that the operator, $\mathbb{A}$ of the system (4.6) satisfies monotonicity and accretivity properties. To do so, we decompose $\mathbb{A}$ into two operators, $\mathbb{A}_z$ and $\mathbb{A}_w$, each acting on Z and W independently, which are the upper left and lower right blocks of $\mathbb{A}$, respectively. For completeness, they are reproduced here:

$$\mathbb{A}_z := \begin{bmatrix} & A_L \\ A_L & A_L N G_z(N^* A_L^* \cdot) \end{bmatrix}, \quad \mathbb{A}_w := \begin{bmatrix} & I \\ -C^{-1}B & -C^{-1}\tilde{N} G_w(\tilde{N}^* B^* \cdot) \end{bmatrix}.$$

First, we begin by showing the operator $-\mathbb{A}_z$ is accretive. Following the notation above, $-\mathbb{A}_z$ must be monotone and m-accretive on the state space $\mathscr{H}_z = L^2 \times L^2$.

Proposition 3. *The operator $-\mathbb{A}_z$ is an accretive operator on $\mathscr{H}_z$; that is,*

$$(\mathbb{A}_z \hat{z} - \mathbb{A}_z z, \hat{z} - z) \leq 0$$

for all $z,\ \hat{z} \in L^2 \times L^2$.

Proof. Let $(z_0, z_1),\ (\hat{z}_0, \hat{z}_1) \in L^2 \times L^2$. Then

$$\mathbb{A}_z \hat{z} = \mathbb{A}_z \begin{bmatrix} \hat{z}_0 \\ \hat{z}_1 \end{bmatrix} = \begin{bmatrix} A_L \hat{z}_1 \\ A_L \hat{z}_0 - A_L N g(N^* A_L^* \hat{z}_1) \end{bmatrix},$$

$$\mathbb{A}_z z = \mathbb{A}_z \begin{bmatrix} z_0 \\ z_1 \end{bmatrix} = \begin{bmatrix} A_L z_1 \\ A_L z_0 - A_L N g(N^* A_L^* z_1) \end{bmatrix}.$$

Therefore,

$$\mathbb{A}_z \hat{z} - \mathbb{A}_z z = \begin{bmatrix} A_L \hat{z}_1 - A_L z_1 \\ A_L \hat{z}_0 - A_L z_0 - A_L N g(N^* A_L^* \hat{z}_1) + A_L N g(N^* A_L^* z_1) \end{bmatrix}.$$

Hence,

$$\begin{aligned}(\mathbb{A}_z \hat{z} - \mathbb{A}_z z, \hat{z} - z)_{L^2 \times L^2} &= (A_L \hat{z}_1 - A_L z_1, \hat{z}_0 - z_0)_{L^2} \\ &+ (A_L \hat{z}_0 - A_L z_0 - A_L N g(N^* A_L^* \hat{z}_1) + A_L N g(N^* A_L^* z_1), \hat{z}_1 - z_1)_{L^2}.\end{aligned}$$

The second term may be split; this allows us to rewrite the inner product as

$$\begin{aligned}(\mathbb{A}_z \hat{z} - \mathbb{A}_z z, \hat{z} - z)_{L^2 \times L^2} &= (A_L \hat{z}_1 - A_L z_1, \hat{z}_0 - z_0)_{L^2} + (A_L \hat{z}_0 - A_L z_0, \hat{z}_1 - z_1)_{L^2} \\ &\quad - (A_L N g(N^* A_L^* \hat{z}_1) - A_L N g(N^* A_L^* z_1), \hat{z}_1 - z_1)_{L^2} \\ &= -(g(N^* A_L^* \hat{z}_1) - g(N^* A_L^* z_1), N^* A_L^* \hat{z}_1 - N^* A_L^* z_1)_{L^2}\end{aligned}$$

That is, the product is of the form

$$(\mathbb{A}_z \hat{z} - \mathbb{A}_z z, \hat{z} - z)_{L^2 \times L^2} = -(g(\hat{\mu}) - g(\mu), \hat{\mu} - \mu)_{L^2}$$

for $\mu = N^* A_L^* z_1$ and $\hat{\mu} = N^* A_L^* \hat{z}_1$. Note that the above quantity is always at most zero. This gives accretivity of $-\mathbb{A}_z$, as desired. □

Next, we must demonstrate that the operator $-\mathbb{A}_z$ is, in fact, m-accretive.

Proposition 4. *The operator* $(\lambda I - \mathbb{A}_z) : \mathscr{H}_z \to (L^2 \times L^2)$ *is a surjective operator for some* $\lambda > 0$.

Proof. First, recall that if $z_1 \in \mathscr{D}(A_L^*)$, then $A_L^* z_1 = \partial_x z_1$, and that the operator $N^* A_L^* z_1 = \mathrm{tr}(z_1)$ at $x = L$. Further, for a scalar r, we have that N applied to r is the constant function $f(x) = r$ for $x \in [0, L]$. Observe, then, that

$$A_L(z_0 + Ng(N^* A_L^* z_1)) = A_L(z_0 + r) = \partial_x z_0.$$

Therefore, proving that the operator $(\lambda I - \mathbb{A}_z)$ is onto the space $L^2 \times L^2$ is equivalent to showing that the system

$$\begin{aligned} \lambda z_0 - \partial_x z_1 &= q_1 \\ \lambda z_1 - \partial_x z_0 &= q_0 \end{aligned} \tag{3.6}$$

is satisfied for any $(q_0, q_1) \in (L^2 \times L^2)$, for some $\lambda > 0$, and where $z_1(0) = 0$, $z_0(L) = -g_1(z_1(L))$ for g_1 a monotone increasing function passing through the origin. For the purposes of the argument, let us take $\lambda = 1$. Observe that this system may be rewritten to be of the form

$$\frac{d}{dx} \begin{bmatrix} z_0 \\ z_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \end{bmatrix} - \begin{bmatrix} q_1 \\ q_0 \end{bmatrix},$$

or, alternatively, the system may be written in the general, recognizable form

$$\frac{dZ}{dx} = LZ - Q.$$

To solve this system, we will appeal to the method of variation of parameters; to this end, we will find the fundamental matrix of this system. Note that the characteristic equation for the matrix L is given by $\mu^2 - 1 = 0$, so that the eigenvalues are $\mu_1 = 1$ and $\mu_2 = -1$. Then the fundamental matrix will take the form

$$\Phi(x) = p_1(x)M_1 + p_2(x)M_2, \tag{3.7}$$

where M_1 is the appropriate identity matrix, $M_2 = L - \mu_1 I = L - I$, and p_1, p_2 solves the system

$$\frac{d}{dx}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix},$$

with $p_1(0) = 1$ and $p_2(0) = 0$. Solving this system directly yields $p_1(x) = e^x$ and $p_2(x) = \frac{1}{2}(e^x - e^{-x}) = \sinh(x)$. Therefore, by Equation (3.7), the fundamental matrix for this system takes the form

$$\Phi(x) = \begin{bmatrix} \cosh(x) & \sinh(x) \\ \sinh(x) & \cosh(x) \end{bmatrix}.$$

Using the fundamental matrix, we know that the system (3.6) has, as its solution,

$$\begin{aligned} Z(x) &= \Phi(x)C + \Phi(x)\int_0^x \Phi^{-1}(s)Q(s)ds \\ &= \begin{bmatrix} \cosh(x) & \sinh(x) \\ \sinh(x) & \cosh(x) \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} q_1(x) \\ q_0(x) \end{bmatrix} - \begin{bmatrix} q_1(0)\cosh(x) + q_0(0)\sinh(x) \\ q_1(0)\sinh(x) + q_0(0)\cosh(x) \end{bmatrix} \end{aligned}$$

for constants c_1 and c_2. Explicitly, this means that z_0 and z_1 take the form

$$z_0(x) = c_1 \cosh(x) + c_2 \sinh(x) + q_1(x) - q_1(0)\cosh(x) - q_0(0)\sinh(x)$$
$$z_1(x) = c_1 \sinh(x) + c_2 \cosh(x) + q_0(x) - q_1(0)\sinh(x) - q_0(0)\cosh(x)$$

Recall that the solution must satisfy the conditions $z_0(L) = -g_1(z_1(L))$ and $z_1(0) = 0$. The latter condition implies directly that $z_1(0) = c_2$, so that in order to satisfy the conditions, we must have $c_2 = 0$. With this information, the solution for $z_1(x)$ evaluated at $x = L$ simplifies; we may simplify it further by letting $d = \sinh(L)$ and $f = q_0(L) - q_1(0)\sinh(L) - q_0(0)\cosh(L)$. Then

$$z_1(L) = dc_1 + f,$$

where d and f are constants. Making similar substitutions for the expression $z_0(L)$ gives $z_0(L) = ac_1 + b$, where a and b are also constants. Combining these equations in light of the requirement that $z_0(L) = -g_1(z_1(L))$, we see that in order for the system to be solved while satisfying the function domain conditions, it must be the case that

$$ac_1 + b = -g_1(dc_1 + f).$$

The quantity on the left is a linear function of c_1 of slope a; since $a = \cosh(L)$, which is positive for any value of L, and since $-g_1$ is a monotone decreasing function passing through the origin, it must be the case that equality in the above expression holds for some value of c_1.

This shows that the system (3.6) does, indeed, have a unique solution satisfying the prescribed conditions, giving that the operator $(\lambda I - \mathbb{A}_z)$ is surjective for $\lambda = 1$; this result together with Proposition 5 yields m-accretivity of the operator $-\mathbb{A}_z$. $\square$

Like for the z-system, it must be shown that the operator $\mathbb{A}_w$ satisfies a few properties. In particular, $-\mathbb{A}_w$ must be monotone and m-accretive on the state space $\mathscr{H}_w = {}_{cl}H^2 \times {}_0H^1$.

Proposition 5. *The operator $-\mathbb{A}_w$ is an accretive operator on $\mathscr{H}_w$; that is,*

$$(\mathbb{A}_w\hat{y} - \mathbb{A}_w y, \hat{y} - y) \leq 0$$

for all $y,\ \hat{y} \in {}_{cl}H^2 \times {}_0H^1$.

Proof. Let $(w, v),\ (\hat{w}, \hat{v}) \in {}_{cl}H^2 \times {}_0H^1$. Then

$$\begin{aligned}(\mathbb{A}_w\hat{y}-\mathbb{A}_w y, \hat{y} - y) &= (\hat{w} - w, \hat{v} - v)_{{}_{cl}H^2} \\ &+ (C^{-1}B(\hat{w} - \tilde{N}G_w(\tilde{N}^*B^*\hat{v})) - C^{-1}B(w - \tilde{N}G_w(\tilde{N}^*B^*v)), \hat{v} - v)_{{}_0H^1}.\end{aligned}$$

We may eliminate the C^{-1} operator from the second term and use the B_{ext} operator instead of the B operator. Simplifying, and supressing the *ext* notation, this gives

$$\begin{aligned}(\mathbb{A}_w\hat{y}-\mathbb{A}_w y, \hat{y} - y) &= -(B\tilde{N}G_w(\tilde{N}^*B^*\hat{v}) - B\tilde{N}G_w(\tilde{N}^*B^*v), \hat{v} - v) \\ &= -(G_w(\tilde{N}^*B^*\hat{v}) - G_w(\tilde{N}^*B^*v), \tilde{N}^*B^*\hat{v} - \tilde{N}^*B^*v).\end{aligned}$$

If we assign $\nu := \tilde{N}^*B^*v$ and $\hat{\nu} = \tilde{N}^*B^*\hat{v}$, then this inner product is of the form

$$-(G_w(\hat{\nu}) - G_w(\nu), \hat{\nu} - \nu)$$

as an inner product in L^2. This expression is always negative, giving that $(\mathbb{A}_w\hat{y} - \mathbb{A}_w y, \hat{y} - y) \leq 0$, as desired. □

Next, we must demonstrate that the operator $-\mathbb{A}_w$ is, in fact, m-accretive; to this end, we prove the following lemma.

Lemma 1. *Let B be the operator as defined previously. If $f \in {}_{cl}H^2$ and $Bf \in ({}_0H^1)^*$, then $f \in \{\phi \in H^3 : \ \phi_{xx}(L) = 0\}$.*

Proof. First, for any $\eta \in H^{-1}(\Omega)$, that is, the dual space of the classical $H_0^1(\Omega)$ space, and $\partial_x \eta \in H^{-1}(\Omega)$, then $\eta \in L^2(\Omega)$ (Thm. 2.6.4, [Kes89]). In particular, since $f \in {}_{cl}H^2$, we have that $f_{xx} \in L^2(\Omega)$. We also know that $\partial_x[\partial_x(f_{xx})] = Bf \in {}_0H^1$. Thus, $\partial_x f_{xx} \in L^2$, meaning that $f \in H^3$.

To show that $f_{xx}(L) = 0$, note that by the above result, one should be able to extend the functional $Bf \in {}_{cl}H^2$, represented as $(Bf)(g) = (f_{xx}, g_{xx})$ to functions $g \in {}_0H^1$. Since $f \in H^3$, integration by parts gives

$$(f_{xx}, g_{xx}) = -(f_{xxx}, g_x) + [f_{xx}(L)g_x(L) - f_{xx}(0)g_x(0)].$$

Since $g \in {}_{cl}H^2$, this reduces to

$$(f_{xx}, g_{xx}) = -(f_{xxx}, g_x) + f_{xx}(L)g_x(L).$$

Since this expression represents a functional extending continuously to $g \in {}_0H^1$, and since such functions do not have a first order trace at $x = L$, it must be the case that $f_{xx}(L) = 0$. □

Proposition 6. *The operator $(\lambda I - \mathbb{A}_w) : \mathscr{H}_w \to (L^2 \times L^2)$ is a surjective operator for some $\lambda > 0$.*

Proof. Proving that the operator in question is surjective is equivalent to showing that the problem

$$\begin{aligned} v + C^{-1}B(w - \tilde{N}G_w(\tilde{N}^*B^*v)) &= q_2 \in {}_0H^1 \\ w - v &= q_1 \in {}_{cl}H^2 \end{aligned} \tag{3.8}$$

has a solution which satisfies $v \in {}_{cl}H^2$ and $w \in {}_{cl}H^2 \cap \{f \in H^3 : \ f_{xx}(L) = 0\}$. First, let us substitute $w = q_1 + v$ into the first equation. This gives

$$\begin{aligned} v + C^{-1}B(w - \tilde{N}G_w(\tilde{N}^*B^*v)) &= q_2 \\ \implies v + C^{-1}B(v + q_1 - \tilde{N}G_w(\tilde{N}^*B^*v)) &= q_2 \\ \implies Cv + Bv + Bq_1 - \tilde{N}G_w(\tilde{N}^*B^*v) &= Cq_2 \\ \implies Cv + Bv - \tilde{N}G_w(\tilde{N}^*B^*v) &= Cq_2 - Bq_1 \in ({}_{cl}H^2)^*. \end{aligned}$$

To prove our claim, it suffices to verify that the operator $C + B - \tilde{N}G_w(\tilde{N}^*B^*\cdot)$ is surjective onto $({}_{cl}H^2)^*$. This reduces further, since C is a linear operator of lower order; thus, it suffices to show $B - \tilde{N}G_w(\tilde{N}^*B^*\cdot)$ is surjective onto $({}_{cl}H^2)^*$. Observe that the operator B defines a Riesz isomorphism $J : {}_{cl}H^2 \to ({}_{cl}H^2)^*$ (recall: we are using the operator B_{ext}). This implies that we need to show $J - \tilde{N}G_w(\tilde{N}^*B^*\cdot)$ is surjective as a map ${}_{cl}H^2 \to ({}_{cl}H^2)^*$. This follows if we show the operator $\tilde{N}G_w(\tilde{N}^*B^*\cdot)$ is maximal monotone on this space.

To this end, consider the nonlinear functional from ${}_{cl}H^2$ into $\mathbb{R}$:

$$\mathcal{J} : v \mapsto \int_0^{v_x(L)} -m(-s)ds + \int_0^{v(L)} g_2(s)ds.$$

This is a convex functional, since the functions m and g_2 are monotone increasing functions through the origin. We can also find the derivative of this operator; indeed, observe that

$$D_v(\mathcal{J}(\phi)) = -G(\tilde{N}^*B^*v) \cdot \tilde{N}^*B^*\phi = -(B\tilde{N}G(\tilde{N}^*B^*v, \phi).$$

In particular, the operator $v \mapsto -B\tilde{N}G(\tilde{N}^*B^*v)$ is the subgradient of a convex functional on ${}_{cl}H^2$ and is therefore a maximal monotone operator mapping ${}_{cl}H^2 \to {}_{cl}H^2$

(Th. 2.8, [Bar10]).

Finally, rearranging the top equation in (3.8) gives

$$Bw = Cq_2 - Cv + B\tilde{N}G_w(\tilde{N}^*B^*v) \in ({}_0H^1)^*.$$

Additionally, we know that $w = q_1 + v \in {}_{cl}H^2$. By Lemma 1, we have that $w \in {}_{cl}H^2 \cap \{f \in H^3 : \ f_{xx}(L) = 0\}$, proving the result. □

3.3.2 Locally Lipschitz Perturbations

To prove the wellposedness of system (4.6), we must show, in addition to the results above, that the perturbation F as in (4.6) is, indeed, locally Lipschitz. Recall the definition of locally Lipschitz.

Definition 5. *A function $F : H_1 \to H_2$ is locally Lipschitz if for any $\rho \geq 0$, there exists a $M_\rho \geq 0$ such that for all x and y with $\|x\|, \|y\| \leq \rho$,*

$$\|F(x) - F(y)\| \leq M_\rho \|x - y\|.$$

The proof that F is locally Lipschitz also relies on the fact that an H^2 function is C^1 with an almost everywhere differentiable derivative and L^2 second derivative. moreover, note that for Banach spaces A and B, we have that the norm on $A \times B$ is equivalent to the following: for $(x, \ y) \in (A \times B)$, $\|(x, y)\|_{A\times B} = \|x\|_A + \|y\|_B$.

Proposition 7. *The perturbation F as in (4.6) is locally Lipschitz.*

Proof. Observe that F consists of two maps, namely, $F_0 : (w, v) \mapsto w_x v_x$, and $C^{-1} \circ F_1 : (w, z_0)(\cdot) \mapsto -(wz_0, \partial_x \cdot)$.

To see that F_0 is locally Lipschitz, let $(w, v),\ (w_0, v_0) \in ({}_{cl}H^2 \times {}_0H^1)$, with $\|(w,\ v)\|_2$ and $\|(w_0,\ v_0)\|_2 \leq \rho$ for some $\rho > 0$. Note that this implies $\|w_{0x}\|$, $\|v_x\| \leq \rho$. Then

$$\begin{aligned}\|F_0((w,v)) - F_0((w_0,v_0))\|_{L^2} &\leq \|v_x\|\|w_{0x} - w_x\| + \|w_{0x}\|\|v_{0x} - v_x\| \\ &\leq (\|v_x\| + \|w_{0x}\|)(\|w_{0x} - w_x\| + \|v_{0x} - v_x\|) \\ &\leq C(\|(w,v)\| + \|(w_0,v_0)\|)\ \|(w,v) - (w_0,v_0)\|_{{}_{cl}H^2 \times {}_0H^1},\end{aligned}$$

showing local Lipschitz continuity for F_0.

To see that $C^{-1} \circ F_1$ is also locally Lipschitz, let $(w, z),\ (w_0, z_0) \in ({}_{cl}H^2 \times L^2)$ with $\|(w,z)\|$ and $\|(w_0,z_0)\| \leq \rho$ for some $\rho > 0$, which, in particular, implies that $\|w_0\|_\infty,\ \|z\| \leq \rho$. We may define an equivalent inner product on ${}_0H^1$ via C, and

$$C : \mathscr{D}(\Delta^{1/2}) = ({}_0H^1)^{-1} \to {}_0H^1 = \mathscr{D}(\Delta^{1/2}).$$

Then to show

$$|C^{-1}(w_x z)_x - C^{-1}(w_{0x} z_0)_x| \leq C(\|w\|_2,\ \|z\|,\ \|w_0\|,\ \|z_0\|)\|(w,z) - (w_0,z_0)\|_{\mathscr{H}},$$

we need only show that $(w_x z)_x$ is locally Lipschitz on $({}_0H^1)^*$. Then

$$\|F_1((w,z)) - F_1((w_0,z_0))\| = |(wz, \partial_x f) - (w_0 z_0, \partial_x f)|$$

where $f \in {}_0H^1$ is a function of norm 1, which implies that $\|f_x\| \leq 1$. Then

$$\begin{aligned}
\|(wz, \partial_x f) - (w_0 z_0, \partial_x f)\| &= \left| \int wzf_x \, dx - \int w_0 z_0 f_x \, dx \right| \\
&= \left| - \int (w_0 z_0 f_x - wzf_x) \, dx \right| \\
&\leq \int |w_0 z_0 - wz||f_x| \, dx \\
&\leq \int |w_0 z_0 - w_0 z||f_x| \, dx + \int |w_0 z - wz||f_x| \, dx \\
&= \int |w_0||z_0 - z||f_x| \, dx + \int |z||w_0 - w||f_x| \, dx \\
&\leq \|w_0\|_\infty \int |z_0 - z||f_x| \, dx + \|w_0 - w\|_\infty \int |z||f_x| \, dx \\
&\leq \|w_0\|_\infty \|z_0 - z\| \|f_x\| + \|w_0 - w\|_\infty \|z\| \|f_x\| \\
&= \|f_x\| \left(\|w_0\|_\infty \|z_0 - z\| + \|w_0 - w\|_\infty \|z\| \right) \\
&\leq \|w_0\|_\infty \|z_0 - z\| + \|w_0 - w\|_\infty \|z\| \\
&\leq \rho \left(\|z_0 - z\| + \|w_0 - w\|_\infty \right),
\end{aligned}$$

giving the desired property. Since both F_0 and F_1 are locally Lipschitz functions, we have that F itself is locally Lipschitz, as desired. □

3.3.3 Energy Identity

To complete the proof of Theorem 1, we appeal to ([CEL02], Thm. 7.2), which relies on establishing that our system with the locally Lipschitz perturbation F is non-expansive. To this end, we define the energy functional associated with our system:

$$E(Z, W) := \frac{1}{2}\|z_0\|^2 + \frac{1}{2}\|z_1\|^2 + \frac{1}{2}\|w''\|^2 + \frac{1}{2}\|v_x\|^2. \tag{3.9}$$

Proposition 8. *Let $\mathcal{B}$ be a bounded subset of $\mathscr{H}$. Then for every initial condition*

y_0 in $\mathcal{B}$ the corresponding local weak solution to (3.2), (3.3) has a well-defined trace of z_1 at $x = L$ and satisfies for every t in its time-domain of existence the following identity:

$$E(t) + g_1(z_1(L,t))z_1(L,t) + m(v'(L,t))v'(L,t) + g_2(v(L,t))v(L,t) = E(0)$$

where $E(t) := E(Z(t), W(t))$. As an immediate consequence, $E(t)$ remains uniformly bounded in time by $E(0)$, whence the norm of $\{Z, W\}$ remains bounded by $\sqrt{c_\gamma E(0)}$.

Proof. Due to the regularity of the strong solution, we may use test functions $\psi_0 = z_0$, $\psi_1 = z_1$, and $\phi_1 = w_t$ in the weak formulation (3.5). Rewriting there

$$(w_{tt}, w_t) = \frac{d}{dt}\frac{1}{2}\|w_t\|^2,$$

the energy identity follows. □

Now, finally, the proof of the main wellposedness theorem follows from a result of Barbu [Bar10] concerning locally Lipschitz perturbations. A brief outline is presented here:

1. Choose any bounded ball $\mathcal{B}_r(0)$ in $\mathscr{H}$. Then consider a different system where the locally Lipschitz perturbation F is replaced by any globally Lipschitz perturbation which agrees with the former on a bounded ball $\mathcal{B}_{r_1}(0)$ (for instance, such a replacement can be defined by "truncations" of F as is done in [CEL02]).

2. The new system has a global solution by [Sho97], Cor. 4.4.1.

3. If the radius r_1 exceeds r , then for any initial data from $\mathcal{B} \cap \mathscr{D}(\mathbb{A})$, the truncated system satisfies (3.9), so that the corresponding strong solution remains in $\mathcal{B}$.

4. This means that the unique strong solution to the new system agrees with the solution to the original one. Likewise, any locally defined strong solution to the original system with initial conditions in $\mathcal{B}$ satisfies the same energy identity and must match the unique solution of the truncated system.

5. Therefore, both solutions are global and bounded. Since weak solutions are $C([0,T];\mathscr{H})$ limits of strong solutions then existence of global weak solutions follows.

As a consequence, we also obtain the following corollary:

Corollary 1. *Every weak solution of (3.2), (3.3) satisfies (3.9) for every* $t \in [0,\infty)$

Proof. First consider the family of strong solutions in $\mathcal{D}(\mathbb{A})$ such that $\{Z^{(n)}, W^{(n)}\}$ converges in $C([0,T];\mathscr{H})$ to weak solution $\{Z, W\}$. Every term of the energy identity, except for $\pi_n := g_1(z_1^{(n)}(L,t))z_1^{(n)}(L,t)$ is continuous with respect to the state space topology $\mathscr{H}$. Note that since g_1 is a monotone graph, then π_n is non-negative and the trace values of $z_1^{(n)}(L,t)$ converge to some number which may be obtained by taking the inverse of the monotone map $s \mapsto g_1(s)s$ at the value

$$E(0) - E(t) - m(v'(L,t))v'(L,t) - g_2(v(L,t))v(L,t).$$

Since $z^{(n)}$ converges to z_1 in L^2 (uniformly in time), and the traces $z_1^{(n)}$ at $x = L$ converge, we can consistently define the trace of $z_1(L,t)$ via this limit.

Thus, the energy identity for strong solutions extends to every weak solution originating in $\mathcal{B}$. □

Chapter 4

Numerical Investigations

In this chapter, we explore the numerical investigations of this model. The goals of the numerical work include to simulate the solution and controls for this system. The existence of solutions to this system has been treated in previous sections; in addition, we know that the original system satisfies the following stability and local controllability results, as given in [LL91] and [Lag91], respectively:

Theorem 2 ([LL91], [Lag91])**.** *Let the continuous functions $g = \{g_1,\ g_2\}$ and m be monotone as graphs satisfying the following prescribed collection of growth conditions:*

$$
\begin{aligned}
c_0|\eta|^{p+1} \le \eta m(\eta) \le C_0|\eta|^{\lambda+1}, \quad |\eta| \le 1 \\
c_0|\eta|^{2} \le \eta m(\eta) \le C_0|\eta|^{2}, \quad |\eta| > 1 \\
c_0|\eta|^{p+1} \le \eta g_1(\eta) \le C_0|\eta|^{\lambda+1}, \quad |\eta| \le 1 \\
c_0|\eta|^{2} \le \eta g_1(\eta) \le C_0|\eta|^{2}, \quad |\eta| > 1 \\
c_0|\eta|^{p+1} \le \eta g_2(\eta) \le C_0|\eta|^{\lambda+1}, \quad |\eta| \le 1 \\
c_0|\eta|^{\sigma+1} \le \eta g_2(\eta) \le C_0|\eta|^{r+1}, \quad |\eta| > 1
\end{aligned}
$$

for some constants c_0, $C_0 > 0$, $\lambda,\ \sigma \in (0, 1]$, $p \ge \lambda$, and $r \ge 1$. Let $\{z,\ w\}$ be any weak solution of (3.2), and let $M > 0$. Then there are constants $C > 0$, $\omega = \omega(M) > 0$

such that the following estimates hold, provided $E(0) \leq M$*:*

1. *If* $p = \lambda = 1$*, then*

$$E(t) \leq Ce^{-\omega t}E(0).$$

2. *if* $p + 1 > 2\lambda$*, then*

$$E(t) \leq C\left[1 + \omega t(E(0))^{\frac{p+1-2\lambda}{2\lambda}}\right]^{\frac{-2\lambda}{p+1-2\lambda}}.$$

Theorem 3 ([LL91], [Lag91])**.** *Let* $T > 0$ *be such that* $\mathbb{H} = H \times H \times {}_0H^1 \times {}_{cl}H^2$ *is in the reachable set of the linearized, decoupled controlled system at time* T*. Then there exists* $r > 0$ *such that the ball* $B_r \in \mathbb{H}$ *is in the reachable set of the nonlinear controlled system (3.2) at time* T*.*

Corollary 2 ([LL91], [Lag91])**.** *Let* $T > 0$ *be such that* $\chi = {}_0H^1 \times H \times {}_{cl}H^2 \times {}_0H^1$ *is in the reachable set at time* T *of the linearization of the controlled system about the zero solution. Then there exists* $r > 0$ *such that the ball* $B_r \in \chi$ *is in the reachable set of the nonlinear controlled system (3.2) at time* T*.*

A combination of feedback and null-controls can be used to steer the system from an arbitrary state to zero. In addition, since the unforced dynamics are reversible, then such a combination of controls can be used to design a single open-loop control that steers the system from zero to an arbitrary state.

We will numerically simulate the system and feedback controls, both linear and nonlinear. In addition, we will simulate control steering from the origin to a relatively high energy state. For the global controllability problem, we may choose what segment of trajectory is designed from feedback controls, and what segment is built from local null-controls. By assuring that the null-control is needed only very close to

zero, we will avoid numerical approximations of the local null control for the nonlinear problem and instead approximate them by null controls for the linearized problem.

This particular control construction is chosen precisely to avoid using minimal norm controls, since the dynamics they produce would not be regular enough for the nonlinear system; this issue is discussed with respect to this system by Lagnese in [Lag90]. In our case, the solution is defined in a larger space than our finite energy space. It is important that controls we employ will steer our solution, even an uncoupled one, in such a way that the solution remains in the finite energy space. The appropriate controls for this task are from a strategy employed by D.L. Russell [Rus73], of which we were made aware via a private communication with Roberto Triggiani [Tri].

Numerical simulations of partial differential equations employ a reduction of infinite dimensional dynamics to a finite-dimensional problem. This is analogous to the majority of exact theoretical results; the latter also utilizes the limits of such approximations to eliminate the approximation limitation. Numerous computational techniques have been devised to date; from the theoretical analysis perspective, the more natural set of methods are those which systematically approximate the underlying state space by a finite-dimensional family of functions. One way to do this is to employ a basis for the space based on the eigenfunctions of the differential operators involved, or, a "spectral" basis. This approach achieves both a natural discretization of differential operators by diagonalizing them, and often leads to many precise theoretical insights about the underlying dynamics. The separation of variables and Fourier series are fundamental and foundational examples of such a technique; truncations of Fourier series can be used for numerical simulations. Another family of methods, and the methods used herein, are the finite element methods (FEM), which were formulated in the mid-1950's, but whose ideas date back to the early 1940's.

These methods were originally introduced for structural analysis problems. FEM do not achieve diagonalization of the differential operators, but the basis functions used are easier to construct, and, unlike spectral bases, the supports of the basis elements are mostly disjoint, which ensures that the same basis yields a sparse matrix representation of many different differential operators on the corresponding subspace.

We will employ FEM approximations to simulate the beam dynamics and its controls. The set of elements for a beam problem is a collection of nonoverlapping, closed intervals covering the domain. The conforming finite element basis for the Sobolev space H^2 on a one dimensional domain consists of piecewise cubic Lagrange and Hermite functions [Šol06]. The coefficients of the Lagrange basis determines the values of the approximation at a discrete set of nodes, and the coefficients of a Hermite basis determine the values of the first-order derivatives (however, these will not necessarily exactly agree with the corresponding values for the actual function that is approximated). This same basis may be used for H^1 and L^2 subspaces, respectively, and will therefore constitute the global basis for all components of our problem.

4.1 Numerical Results

Recall that for our problem and the numerical investigations, we will be working on the 1D domain $\Omega = [0, 1]$. The notation we will employ is as follows: n is the number of points, h is the distance between nodes, and T is the simulation time. As mentioned previously, basis functions for the finite element approximation consist of piecewise cubic functions. Two examples of such piecewise cubic functions used may be seen below.

Construction of the matrices used for the finite element analysis relies on these basis functions, and the construction of "local" matrices for each element. For instance,

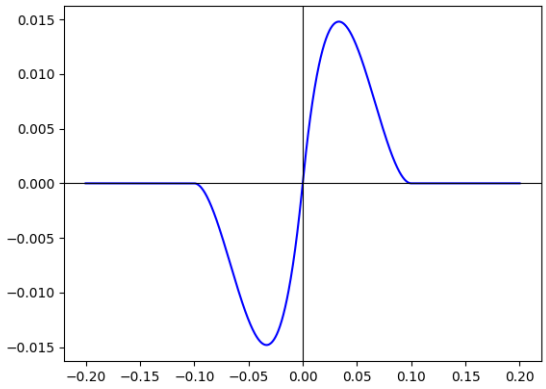

Figure 4.1: An example of a piecewise cubic basis function (centered at $x = 0$) used in the finite element analysis of our system. This function is $f(x) = \frac{x^3}{h^2} + \frac{2x^2}{h} + x$ if $-h \leq x < 0$, $f(x) = \frac{x^3}{h^2} - \frac{2x^2}{h} + x$ if $0 \leq x < h$, and $f(x) = 0$ otherwise. For this example, $h = 0.1$.

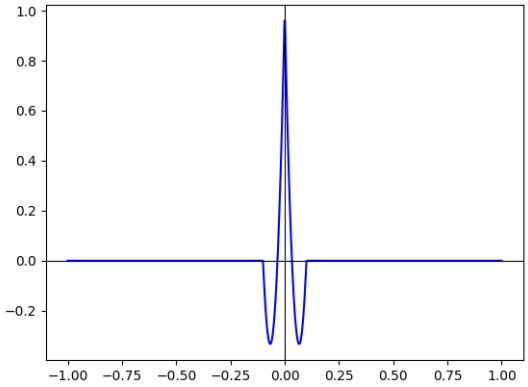

Figure 4.2: An example of a piecewise cubic basis function (centered at $x = 0$) used in the finite element analysis of our system. This function is $f(x) = \frac{3x^2}{h^2} + \frac{4x}{h} + 1$ if $-h \leq x < 0$, $f(x) = \frac{3x^2}{h^2} - \frac{4x}{h} + 1$ if $0 \leq x < h$, and $f(x) = 0$ otherwise. For this example, $h = 0.1$.

for the construction of the generic mass matrix, individual local element matrices are constructed, where the proper number of derivatives are taken on each basis function (in the case of the mass matrix, zero derivatives on either). For n points, the 1D domain is discretized into $n-1$ intervals, and for each interval/element e_k, there are two basis functions of each type with support on element e_k. For each element, this gives rise to four 2×2 matrices; if ϕ is a Lagrange basis function and ψ is a Hermite basis function, these local element matrices will have the form:

$$
\begin{aligned}
m_{LL,e_k} &= \begin{bmatrix} \int_{e_k} \phi_k^{d_1} \phi_k^{d_2} dx & \int_{e_k} \phi_k^{d_1} \phi_{k+1}^{d_2} dx \\ \int_{e_k} \phi_{k+1}^{d_1} \phi_k^{d_2} dx & \int_{e_k} \phi_{k+1}^{d_1} \phi_{k+1}^{d_2} dx \end{bmatrix} \\
m_{LH,e_k} &= \begin{bmatrix} \int_{e_k} \phi_k^{d_1} \psi_k^{d_2} dx & \int_{e_k} \phi_k^{d_1} \psi_{k+1}^{d_2} dx \\ \int_{e_k} \phi_{k+1}^{d_1} \psi_k^{d_2} dx & \int_{e_k} \phi_{k+1}^{d_1} \psi_{k+1}^{d_2} dx \end{bmatrix} \\
m_{HL,e_k} &= \begin{bmatrix} \int_{e_k} \psi_k^{d_1} \phi_k^{d_2} dx & \int_{e_k} \psi_k^{d_1} \phi_{k+1}^{d_2} dx \\ \int_{e_k} \psi_{k+1}^{d_1} \phi_k^{d_2} dx & \int_{e_k} \psi_{k+1}^{d_1} \phi_{k+1}^{d_2} dx \end{bmatrix} \\
m_{HH,e_k} &= \begin{bmatrix} \int_{e_k} \psi_k^{d_1} \psi_k^{d_2} dx & \int_{e_k} \psi_k^{d_1} \psi_{k+1}^{d_2} dx \\ \int_{e_k} \psi_{k+1}^{d_1} \psi_k^{d_2} dx & \int_{e_k} \psi_{k+1}^{d_1} \psi_{k+1}^{d_2} dx \end{bmatrix},
\end{aligned}
$$

where $d_1,\ d_2$ are the number of spacial derivatives to be taken on each basis function. These matrices are constructed for each element e_k, and are stored block-diagonal style in respective "global" Lagrange-Lagrange (LL), Lagrange-Hermite (LH), Hermite-Lagrange (HL), or Hermite-Hermite (HH) matrices. Since basis functions overlap on elements, diagonal entries of these global matrices will be the sums of respective products of basis functions (with the exception of the first and last diagonal entries of the global matrices, since no overlap occurs here).

As an example, if $n = 5$, then the 1D domain is broken up into four elements: e_1, e_2, e_3, and e_4. If we wish to construct a mass matrix, the local LL element matrix

for element e_1 is of the form

$$m_{LL,e_1} = \begin{bmatrix} \int_{e_1} \phi_1\phi_1 \, dx & \int_{e_1} \phi_1\phi_2 \, dx \\ \int_{e_1} \phi_2\phi_1 \, dx & \int_{e_1} \phi_2\phi_2 \, dx \end{bmatrix}$$

and the global Lagrange-Lagrange matrix is a matrix of the form

$$M_{LL} = \begin{bmatrix} \int_{e_1} \phi_1^2 \, dx & \int_{e_1} \phi_1\phi_2 \, dx & & & \\ \int_{e_1} \phi_2\phi_1 \, dx & \int_{e_1\cup e_2} \phi_2^2 \, dx & \int_{e_2} \phi_2\phi_3 \, dx & & \\ & \int_{e_2} \phi_3\phi_2 \, dx & \int_{e_2\cup e_3} \phi_3^2 \, dx & \int_{e_3} \phi_3\phi_4 \, dx & \\ & & \int_{e_3} \phi_4\phi_3 \, dx & \int_{e_3} \phi_4^2 \, dx & \int_{e_4} \phi_4\phi_5 \, dx \\ & & & \int_{e_4} \phi_5\phi_4 \, dx & \int_{e_4} \phi_5^2 \, dx \end{bmatrix}.$$

Observe that along the diagonal of the matrix above, integrals are taken over two elements, with the exception of the first and last diagonal entries. Once global LL, LH, HL, and HH matrices are constructed, they are finally assembled into a larger matrix of the form

$$M = \begin{bmatrix} LL & LH \\ HL & HH \end{bmatrix}$$

which, in our example, becomes a mass matrix. This matrix is banded, and often, matrices such as these may be sparse and represented by sparse matrices to make calculations less computationally demanding. For our purposes, FEM matrices are generally constructed in this manner, with the differences among matrices corresponding to differences in the derivatives applied to the basis functions. For a classical stiffness matrix, for example, d_1 and d_2 may be taken to be 1.

Matrix construction depends upon the weak formulation of a problem, which will be considered here. The weak formulation of (3.2), (3.3) is given by (3.5). This

formulation of our system, both forced and unforced, will be approximated via FEM. Many matrices are constructed for this problem following the formula above; we may use the notation $A_{d_1 d_2}$ to indicate matrices constructed with d_1 derivatives on the first test function and d_2 derivatives on the second. Hence, the FEM approximation for the uncoupled, unforced system has the form

$$\frac{d}{dt}\begin{bmatrix}\hat{z_0}\\ \hat{z_1}\\ \hat{v}\\ \hat{w}\end{bmatrix} = \begin{bmatrix} A_{00}^{-1}A_{10}^{*}z_1 \\ -A_{00}^{-1}A_{10}z_0 \\ (A_{00}+\rho A_{11})^{-1}(-\gamma^2 A_{22})w \\ Iv \end{bmatrix}$$

Likewise, the forced, decoupled system has the form

$$\frac{d}{dt}\begin{bmatrix}\hat{z_0}\\ \hat{z_1}\\ \hat{v}\\ \hat{w}\end{bmatrix} = \begin{bmatrix} A_{00}^{-1}A_{10}^{*}z_1 \\ -A_{00}^{-1}\left(A_{10}z_0 + A_{10}N_z g_z(N_z^* A_{10}^* z_1)\right) \\ (A_{00}+\rho A_{11})^{-1}(-\gamma^2 A_{22})\left(w - N_w g_w(N_w^* B^* v)\right) \\ Iv \end{bmatrix}$$

where $g_z,\ g_w$ are the appropriate control functions, and $N_z,\ N_w$ are as in the trace map formulations of Chapter 2, approximated here by taking the appropriate L^2, H^2 projections of the functions $f(x) = 1$, $f(x) = \frac{x^2}{2}$, and $f(x) = \frac{-x^2}{2} + \frac{x^3}{6}$, from (2.4). Incorporating the coupling of this system is achieved through the use of tensors, which are conveniently implemented through the use of python ndarray structures. For the coupling term found in the z_0 equation, a A_{110} ndarray is constructed, with indices (i, j, k) and whose kth entry is an augmented A_{11} matrix: an additional test function, ϕ_k or ψ_k, appears in each integrand of the matrix, so that, for example, an entry may be of the form $\int_e \phi_i^1 \phi_j^1 \phi_k \, dx$. A similar construction is used for the coupling term of

the w equation.

In the section that follows, various simulations are performed and energies plotted. In each case, the Runge-Kutte 4 method was used for iterations. Additionally, all computations were performed using the Python programming language [Fou], release 3.6.0, with the numpy and scipy libraries [JOP$^+$] . Development was done using the JetBrains PyCharm IDE [Jet].

4.1.1 The Unforced System

Using the FEM approximations, we may examine the energy of the coupled, unforced system. For this simulation, the initial conditions are as follows:

$$z_0(x,0) = 0, \quad z_1(x,0) = 0, \quad v(x,0) = 0, \quad w(x,0) = 25x.$$

The energies of the z and w components, as well as the total energy, is plotted in Figure 4.3. Observe the flat total energy curve; this indicates that the system is conservative. For another example, we may use the initial conditions

$$z_0(x,0) = z_1(x,0) = v(x,0) = w(x,0) = 1.$$

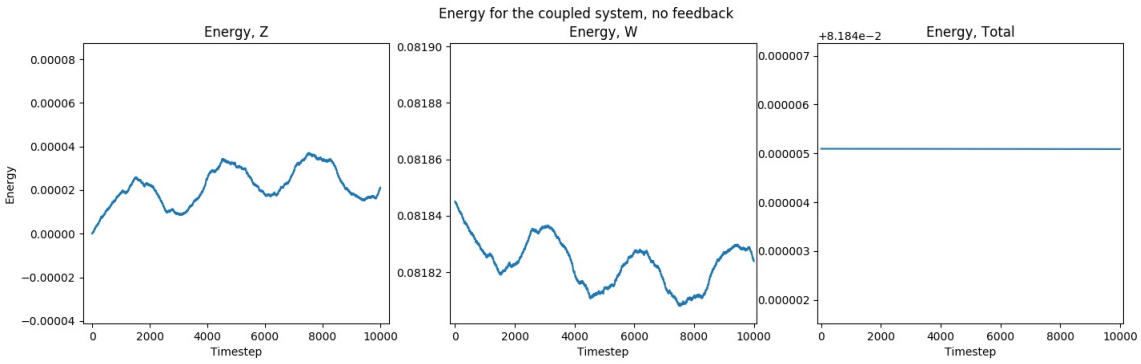

Figure 4.3: The energy of the system with initial condition $w(x,0) = 25x$ with all other initial conditions zero.

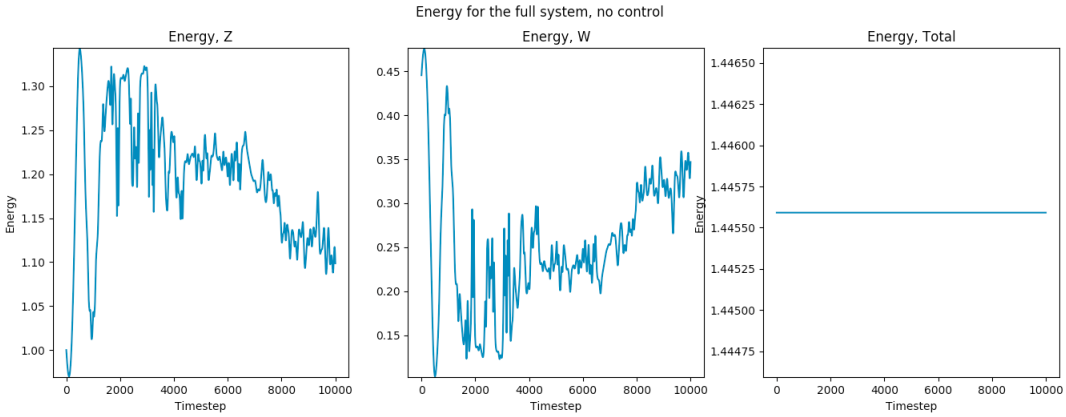

Figure 4.4: The energy of the system with initial conditions $z_0(x,0) = z_1(x,0) = v(x,0) = w(x,0) = 1$.

Again, energies are plotted in Figure 4.4.

4.1.2 The Forced System

The forced system can change depending on values of the functions g_1, g_2, and m from (3.3). These functions have the effect of boundary damping, and can be linear or nonlinear. For an example of the system with linear boundary damping, one may simply take the function values $g_1(z_1(L,t)) = g_1(x) = x$, $g_2(v(L,t)) = g_2(x) = x$, and $m(v'(L,t)) = m(x) = x$. The energies of the coupled, forced system with these boundary damping functions may be seen in Figure 4.5. To contrast, we may use nonlinear feedback functions. Figure 4.6 contains plots of energies for the same initial conditions as above, but with feedback functions $g_1(z_1(L,t)) = g_1(x) = x^3$, $g_2(v(L,t)) = g_2(x) = x^3$, and $m(v'(L,t)) = m(x) = x^3$

4.1.3 Null Controls

We may use a non-feedback control mechanism to force the energy of the system to zero. This mechanism, due to D. Russell [Rus73] and brought to our attention by

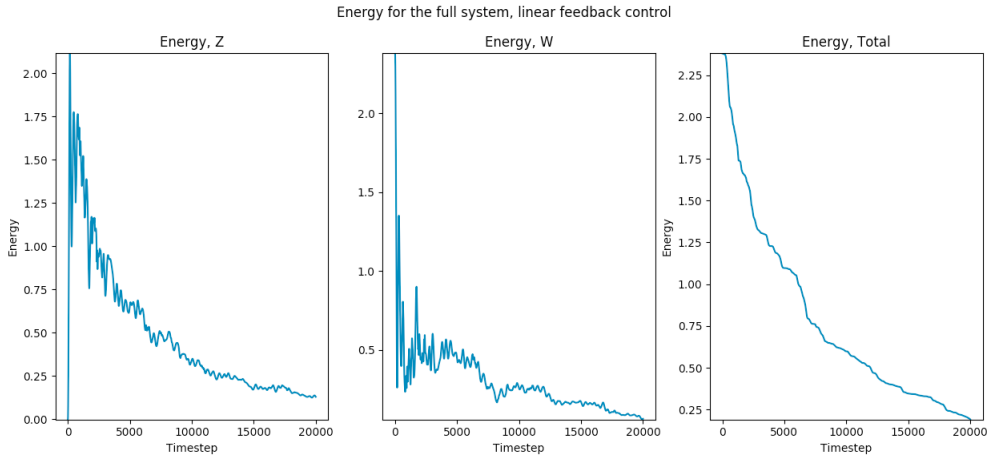

Figure 4.5: The energy of the system with initial conditions $z_0(x,0) = z_1(x,0) = 0$, $v(x,0) = w(x,0) = 5x^2$, with feedback control enabled. Each control function is the identity.

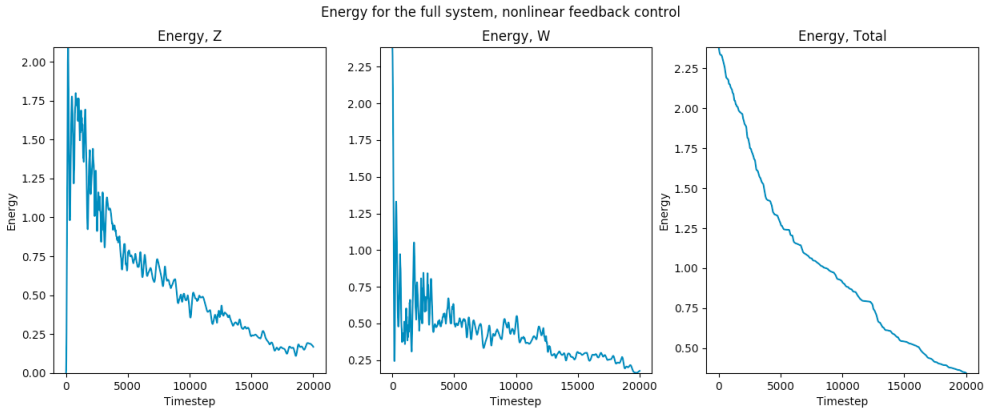

Figure 4.6: The energy of the system with initial conditions $z_0(x,0) = z_1(x,0) = 0$, $v(x,0) = w(x,0) = 5x^2$, with nonlinear feedback control enabled.

Roberto Triggiani [Tri], uses the matrix exponential to achieve exact controllability (to the origin) at some time T sufficiently large. This control is constructive; more precisely, the control

$$u(t) = F^{+}e^{(A+BF^{+})t}y_0 - F^{-}e^{(A+BF^{-})(t-T)}e^{(A+BF^{+})T}y_0 \in C([0,T];\mathbb{R}^m)$$

steers the solution $y(t;y_0;u)$ of $y_t = Ay+Bu$ to zero at time $t = T$, where A and B are $n \times n$ matrices. This control also works if A and B are operators, and, additionally, in the case of a conservative system, we have $F^{-} = 0$. The quantity y_0, in the case of a conservative system, is given by

$$y_0 = \left[I - e^{(A+BF^{+})T}\right]^{-1} y(0),$$

where $y(0)$ is the initial data for the system. Employing this control to our system, the energy at time $t = T$ does, indeed, go to zero. Since our testing domain is $[0,1]$, a time sufficiently large must allow for energy to travel the length of the domain (beam) and return; time $T = 3$ was chosen for these simulations. For the CFL condition of $\Delta t = (\Delta x)^2/10$, this means $t = 3$ occurs at time step 3000. The results of this simulation can be seen in Figure 4.7.

4.1.4 Reversed Control

Next, we wish to demonstrate exact controllability by combining the feedback controls and null controls to steer the energy of our system to a desired state. The idea for this goes back to Russell [Rus78]. This method relies on the fact that the unforced system is reversible; specifically, in the original u-w formulation, it can be seen that if $t \to u(t),\ w(t)$ is a solution, then so is $t \to u(K-t),\ w(K-t)$. It is worthwhile

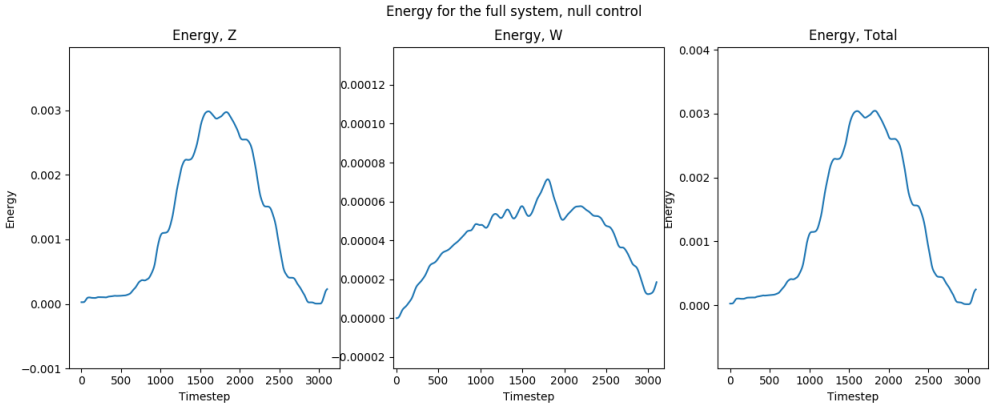

Figure 4.7: The energy of the coupled system with a null control. The initial conditions for this simulation were $z_0(x,0) = z_1(x,0) = 0.01x^2$, $v(x,0) = 0.0001x^2$, and $w(x,0) = 0.001x^3$. Because we are using an FEM approximation of our system, the energy at time $t = 3$ does, indeed, approach zero, and becomes very close, but may not exactly be zero.

to consider this exercise in the ordinary differential equation framework: consider the following initial value problem for a general second order ordinary differential equation:

$$\begin{cases} y''(t) + u(t) + y(t) = 0, \\ y(0) = a, \\ y'(0) = b, \end{cases} \tag{4.1}$$

and suppose the control is $u(t) = y'(t)$. Next, suppose $z(t) := y(T-t)$ for some time $T > 0$. Then

$$z'(t) = -y'(T-t) \quad \text{and} \quad z''(t) = y''(T-t),$$

and $z(T) = a$, $z'(T) = -b$. Hence, z satisfies the initial value problem

$$\begin{cases} z''(t) - z'(t) + z(t) = 0, \\ z(0) = y(T), \\ z'(0) = -y'(T). \end{cases} \tag{4.2}$$

This implies that to steer the system (4.1) above to the targets a and b, one may follow the following steps, making use of a reverse control:

1. Initialize the initial value problem (4.1) with the data $y(0) = a$ and $y'(0) = -b$.
2. Run the y problem to time T, at which point $y(T) = \alpha$ and $y'(T) = \beta$ for some α and β.
3. Initialize the initial value problem (4.2) with the data $z(0) = \alpha$ and $z'(0) = -\beta$.
4. Finally, run the z problem an additional time T, at which point $z(T) = a$ and $z'(T) = b$.

This same strategy is employed here, by making use of a combination of the feedback and null controls. The steps for our system are as follows:

1. Initialize our system to energy level E, but invert the signs on the velocity initial data.
2. Run the system using a feedback control until the energy level drops below a set threshold ϵ close to zero, while storing the feedback control values.
3. Run the system using a null control to steer the system to $E = 0$ while storing the null control values.

4. Initialize a new instance of the system whose initial data is the terminal data of the previous instance, with inverted signs on velocity initial data.

5. Run the new system forward using control values stored in steps (2) and (3), returning approximately to a state with energy level E.

We employ this strategy for our system. The result of one such simulation may be seen in Figure 4.8. For this simulation, the initial conditions are $z_0(x) = x^3$, $z_1(x) = x$, and $v(x) = w(x) = 5x^2$, and the boundary feedback controls are linear, with $g_1(x) = g_2(x) = m(x) = x$. At timestep t, we are plotting the energy of the system state at time t minus the target state, or E(state - target). Recall that until the reverse controls begin, the target state is the zero state; then the target switches to the initial state but with velocity of opposite signs.

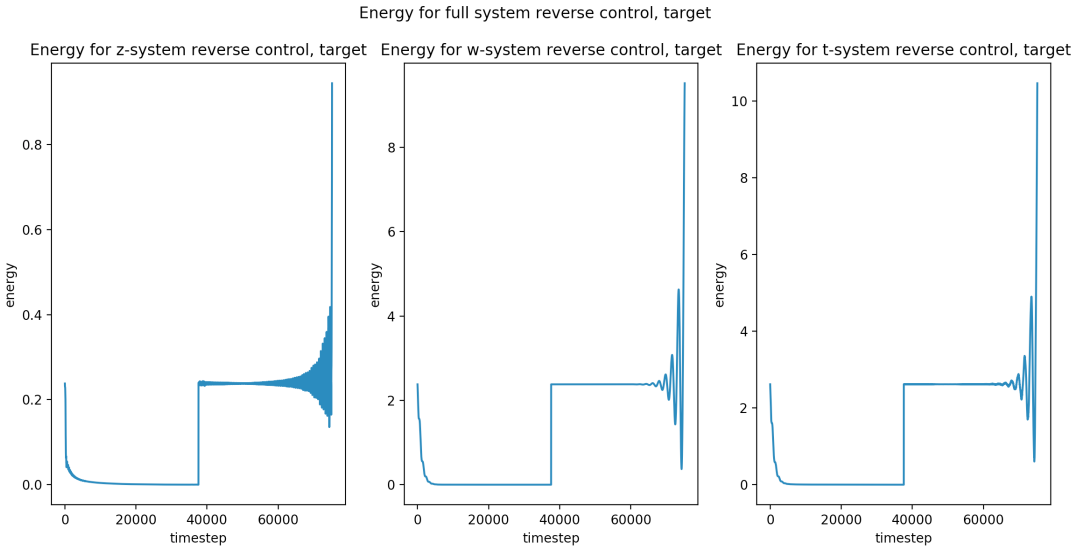

Figure 4.8: The energy of the state of the system at time t minus the target state.

The oscillatory behavior toward the end of the simulation is interesting, but not necessarily unexpected. This behavior is also seen when investigating harmonic oscil-

lators; it may be the case that our system takes an indirect trajectory to the target. In particular, the trajectory may be spiraling.

Chapter 5

Future Directions

There are a few directions that future research could go based off of the results of this thesis; a few examples will be listed here.

Based off of the results of the reverse control implementation, one may wonder if controls may be adapted from a coarse time scale to a refined one. After implementing the exact controls for a coarse time scale, store the values and convert them into a piecewise continuous function. Then interpolate between control values, and use those values in a system running with a smaller time step. What would the results be, and, would they match a simulation originally using the smaller time step?

The simplest instances of energy dissipation mechanisms, at least as far as elastic beams are concerned, are direct damping mechanisms, like those investigated by this thesis. However, an investigation of *indirect* damping mechanisms, as mentioned in [Rus93], may be worthwhile. In short, instead of inserting damping terms into the original equations of motion, the original equations may be further coupled with more equations describing other mechanics in the structure. That is, in the case of indirect controls, we no longer specify the control values; instead, the control function will, itself, be a solution to another system which we may attempt to influence. Some examples would be the coupling of beam equations to those describing thermal effects, or those describing the mechanics of a vibrating structure welded to the edge of a

beam. These are, in general, more difficult to investigate, especially in the nonlinear setting, but, using the framework we have constructed, may yield interesting results.

The null controls we have implemented here have been computed, technically, for the linear problem only, which is why our control only approaches zero and does not *exactly* reach zero. However, in a neighborhood of zero, there is, actually, a control which does steer the full nonlinear system to zero, as proven by Lagnese [Lag91]. One possible strategy would be to employ a fixed point argument on the nonlinear equation defining such a control, as in [Tri10], [LT81], where the control-to-state map $\mathcal{L}_T$ is replaced by the stabilization-based controls we use here. Investigating these controls may prove more difficult than those explored in this thesis, but, at the very least, these controls will be computationally time consuming, may not converge, and may not award significant improvement over the already quite precise null controls employed by our work. Nevertheless, an attempt to implement these controls may prove fruitful and intellectually rewarding.

An obvious future direction is to undertake a different numerical implementation of our system. We used the finite element method in our approach, but spectral methods are another alternative. Furthermore, implementing an alternative timestepping method may be of use, including a two-part algorithm such as Adams-Bashforth. This problem also lends itself to piston theory for a beam in a potential flow. In piston theory, for large flow velocities, the fluid pressure on the top surface of a structure, such as a beam, may be approximated by the fluid pressure on the head of a piston moving through a column of fluid. Investigating coupled models that make use of our system and piston theory is outside the scope of this thesis, but could make use of the work presented herein.

Appendix A

Code

This appendix contains all of the computer code used as part of this project. Each section of this appendix is a separate file, with file names as section names. Additionally, this code is hosted on GitHub at the following address: `https://github.com/jessiejamieson/dissertation_code`. For instance, to run an instance of the system using feedback controls, one would edit the feedback_run.py file and run the following terminal command:

```
$ python3 feedback_run.py
```

A.1 source/basis_function.py

```
1  import scipy.integrate as integrate
2
3
4  class MultiFunc(object):
5      """Class to describe multiply two functions as a function
6
7          Attributes:
8              _func0 := first function in product
9              _func1 := second function in product
10
11         Methods:
12             __init__ := constructor
13             __mul__  := multiply the function by another one
14             __call__ := return the result of the multiply function
15             integral := return the integral of the function
16     """
17
18     def __init__(self, func0, func1):
19         self._func0 = func0
20         self._func1 = func1
```

```
21
22      def __mul__(self, func):
23          """Define multiplication by a function
24
25              Args:
26                  func := the function to multiply by
27
28              Returns:
29                  MultiFunc of self and func
30          """
31
32          return MultiFunc(self, func)
33
34      def __call__(self, x):
35          """Call the function
36
37              Args:
38                  x := the input variable
39
40              Returns:
41                  product of results from both functions involved
42          """
43
44          value0 = self._func0(x)
45          value1 = self._func1(x)
46
47          return value0 * value1
48
49      def integral(self, lower, upper):
50          """Integrate function from lower to upper using quadrature
51
52              int_lower^upper self(x) dx
53
54              Args:
55                  lower := lower bound for integration
56                  upper := upper bound for integration
57
58              Returns:
59                  integral of self from lower to upper
60          """
61
62          return integrate.quad(self, lower, upper)[0]
63
64
65  class BasisFunction(object):
66      """Class to contain a single basis function
67
68          Attributes:
69              _index      := index of the basis function
70              _derivative := derivative of the basis function (0, 1, 2)
71              _num_points := number of points
72              _delta_x    := distance between the points
73
74          Properties:
75              delta_x_2 := _delta_x**2
76              delta_x_3 := _delta_x**3
77              center    := find the center of the function
78
79          Method:
80              __mul__  := multiply this by another function
81              __call__ := call the function for the result
82              new_x    := move input to reference interval
83              in_lower := determine if in lower half of interval
84              in_upper := determine if in upper half of interval
```

```
85
86              _basis    := combine the upper and lower functions
87              _lower_0  := no derivatives lower part
88              _upper_0  := no derivatives upper part
89              _basis_0  := no derivatives basis function
90              _lower_1  := one derivatives lower part
91              _upper_1  := one derivatives upper part
92              _basis_1  := one derivatives basis function
93              _lower_2  := two derivatives lower part
94              _upper_2  := two derivatives upper part
95              _basis_2  := two derivatives basis function
96              _evaluate := select the basis of correct derivative
97      """
98
99      def __init__(self, index, derivative, num_points, delta_x):
100         self._index = index
101         self._derivative = derivative
102         self._num_points = num_points
103         self._delta_x = delta_x
104
105     def __mul__(self, func):
106         """Define multiplication by a function
107
108             Args:
109                 func := the function to multiply by
110
111             Returns:
112                 MultiFunc of self and func
113         """
114
115         return MultiFunc(self, func)
116
117     @property
118     def delta_x_2(self):
119         """Find the square of delta_x
120
121             Returns:
122                 delta_x squared
123         """
124
125         return self._delta_x**2
126
127     @property
128     def delta_x_3(self):
129         """Find the cube of delta_x
130
131             Returns:
132                 delta_x cubed
133         """
134
135         return self._delta_x**3
136
137     @property
138     def center(self):
139         """Get the center of the function
140
141             Returns:
142                 the center point for the function
143         """
144
145         return self._index * self._delta_x
146
147     def new_x(self, x):
148         """Map an x to the reference interval
```

```
149
150             Returns:
151                 move x onto the reference interval
152         """
153
154         return x - self.center
155
156     def in_lower(self, x):
157         """Determine if x is in lower interval
158
159             Args:
160                 x := the shifted input variable
161
162             Returns:
163                 True:  if x is between -delta_x and 0.0
164                 False: otherwise
165         """
166
167         return (-self._delta_x <= x) and (x <= 0.0)
168
169     def in_upper(self, x):
170         """Determine if x is in upper interval
171
172
173             Args:
174                 x := the shifted input variable
175
176             Returns:
177                 True:  if x is between 0.0 and delta_x
178                 False: otherwise
179         """
180
181         return (self._delta_x >= x) and (x >= 0.0)
182
183     def _basis(self, x, lower, upper):
184         """Put basis function together
185
186             Args:
187                 x     := the shifted input variable
188                 lower := lower function
189                 upper := upper function
190
191             Returns:
192                 lower(x) if in lower part of interval
193                 upper(x) if in upper part of interval
194                 0.0 if not in interval
195         """
196
197         if self.in_lower(x):
198             return lower(x)
199         elif self.in_upper(x):
200             return upper(x)
201         else:
202             return 0.0
203
204     def _lower_0(self, x):
205         """The lower part with derivative 0:
206
207             Args:
208                 x := the shifted input variable
209
210             Returns:
211                 pass
212         """
```

```
213             pass
214
215         def _upper_0(self, x):
216             """The upper part with derivative 0:
217
218                 Args:
219                     x := the shifted input variable
220
221                 Returns:
222                     pass
223             """
224             pass
225
226         def _basis_0(self, x):
227             """The basis function for derivative 0
228
229                 Args:
230                     x := the shifted input variable
231
232                 Returns:
233                     the result of the basis function with 0 derivative
234             """
235
236             return self._basis(x, self._lower_0, self._upper_0)
237
238         def _lower_1(self, x):
239             """The lower part with derivative 1:
240
241                 Args:
242                     x := the shifted input variable
243
244                 Returns:
245                     pass
246             """
247             pass
248
249         def _upper_1(self, x):
250             """The upper part with derivative 1:
251
252                 Args:
253                     x := the shifted input variable
254
255                 Returns:
256                     pass
257             """
258             pass
259
260         def _basis_1(self, x):
261             """The basis function for derivative 1
262
263                 Args:
264                     x := the shifted input variable
265
266                 Returns:
267                     the result of the basis function with 1 derivative
268             """
269
270             return self._basis(x, self._lower_1, self._upper_1)
271
272         def _lower_2(self, x):
273             """The lower part with derivative 2:
274
275                 Args:
276                     x := the shifted input variable
```

```
277
278             Returns:
279                 pass
280         """
281         pass
282
283     def _upper_2(self, x):
284         """The upper part with derivative 2:
285
286             Args:
287                 x := the shifted input variable
288
289             Returns:
290                 pass
291         """
292         pass
293
294     def _basis_2(self, x):
295         """The basis function for derivative 2
296
297             Args:
298                 x := the shifted input variable
299
300             Returns:
301                 the result of the basis function with 2 derivative
302         """
303
304         return self._basis(x, self._lower_2, self._upper_2)
305
306     def _evaluate(self, x):
307         """Select the correct basis function via the internal derivative value
308
309             Args:
310                 x := the shifted input variable
311
312             Returns:
313                 result of _basis_0 if _derivative == 0
314                 result of _basis_1 if _derivative == 1
315                 result of _basis_2 if _derivative == 2
316                 error otherwise
317         """
318
319         if self._derivative == 0:
320             return self._basis_0(x)
321         elif self._derivative == 1:
322             return self._basis_1(x)
323         elif self._derivative == 2:
324             return self._basis_2(x)
325         else:
326             raise AttributeError('Invalid derivative')
327
328     def __call__(self, x):
329         """Evaluate the basis function at x
330
331             Args:
332                 x := input variable
333
334             Returns:
335                 value of the function at x
336         """
337
338         new_x = self.new_x(x)
339
340         return self._evaluate(new_x)
```

```
341
342
343 class Lagrange(BasisFunction):
344     """Class to contain a single Lagrange basis function
345
346         Inherits from Basis function
347     """
348
349     def _part0_0(self, x):
350         """Evaluate first part of a Lagrange basis function with 0 derivative
351
352             f00(x) = 2/delta_x^3 * x^3
353
354             Args:
355                 x := the shifted input variable
356
357             Returns:
358                 f00(x)
359         """
360
361         return (x**3 * 2.0) / self.delta_x_3
362
363     def _part1_0(self, x):
364         """Evaluate second part of a Lagrange basis function with 0 derivative
365
366             f10(x) = 3/delta_x^2 * x^2
367
368             Args:
369                 x := the shifted input variable
370
371             Returns:
372                 f10(x)
373         """
374
375         return (x**2 * 3.0) / self.delta_x_2
376
377     def _lower_0(self, x):
378         """Evaluate lower part of a Lagrange basis function with 0 derivative
379
380             fl0(x) = -f00(x) - f10(x) + 1
381
382             Args:
383                 x := the shifted input variable
384
385             Returns:
386                 fl0(x)
387         """
388
389         return -self._part0_0(x) - self._part1_0(x) + 1.0
390
391     def _upper_0(self, x):
392         """Evaluate upper part of a Lagrange basis function with 0 derivative
393
394             fu0(x) = f00(x) - f10(x) + 1
395
396             Args:
397                 x := the shifted input variable
398
399             Returns:
400                 fu0(x)
401         """
402
403         return self._part0_0(x) - self._part1_0(x) + 1.0
404
```

```
405     def _part0_1(self, x):
406         """Evaluate first part of a Lagrange basis function with 1 derivative
407 
408             f01(x) = 6/delta_x^3 * x^2
409 
410             Args:
411                 x := the shifted input variable
412 
413             Returns:
414                 f01(x)
415         """
416 
417         return (x**2 * 6.0) / self.delta_x_3
418 
419     def _part1_1(self, x):
420         """Evaluate second part of a Lagrange basis function with 1 derivative
421 
422             f11(x) = 6/delta_x^2 * x
423 
424             Args:
425                 x := the shifted input variable
426 
427             Returns:
428                 f11(x)
429         """
430 
431         return (x * 6.0) / self.delta_x_2
432 
433     def _lower_1(self, x):
434         """Evaluate lower part of a Lagrange basis function with 1 derivative
435 
436             fl1(x) = -f01(x) - f11(x)
437 
438             Args:
439                 x := the shifted input variable
440 
441             Returns:
442                 fl1(x)
443         """
444 
445         return -self._part0_1(x) - self._part1_1(x)
446 
447     def _upper_1(self, x):
448         """Evaluate upper part of a Lagrange basis function with 1 derivative
449 
450             fu1(x) = f01(x) - f11(x)
451 
452             Args:
453                 x := the shifted input variable
454 
455             Returns:
456                 fu1(x)
457         """
458 
459         return self._part0_1(x) - self._part1_1(x)
460 
461     def _part0_2(self, x):
462         """Evaluate first part of a Lagrange basis function with 2 derivative
463 
464             f02(x) = 12/delta_x^3 * x
465 
466             Args:
467                 x := the shifted input variable
468 
```

```
469             Returns:
470                 f02(x)
471         """
472
473         return (x * 12.0) / self.delta_x_3
474
475     def _part1_2(self, x):
476         """Evaluate second part of a Lagrange basis function with 2 derivative
477
478             f12(x) = 6/delta_x^2
479
480             Args:
481                 x := the shifted input variable
482
483             Returns:
484                 f12(x)
485         """
486
487         return 6.0 / self.delta_x_2
488
489     def _lower_2(self, x):
490         """Evaluate lower part of a Lagrange basis function with 2 derivative
491
492             fl2(x) = -f02(x) - f12(x)
493
494             Args:
495                 x := the shifted input variable
496
497             Returns:
498                 fl2(x)
499         """
500
501         return -self._part0_2(x) - self._part1_2(x)
502
503     def _upper_2(self, x):
504         """Evaluate upper part of a Lagrange basis function with 2 derivative
505
506             fu2(x) = f02(x) - f12(x)
507
508             Args:
509                 x := the shifted input variable
510
511             Returns:
512                 fu2(x)
513         """
514
515         return self._part0_2(x) - self._part1_2(x)
516
517
518 class Hermite(BasisFunction):
519     """Class to contain a single Hermite basis function
520
521         Inherits from Basis function
522     """
523
524     def _part0_0(self, x):
525         """Evaluate first part of a Hermite basis function with 0 derivative
526
527             f00(x) = 1/delta_x^2 * x^3
528
529             Args:
530                 x := the shifted input variable
531
532             Returns:
```

```
533                 f00(x)
534             """
535
536             return (x**3) / self.delta_x_2
537
538         def _part1_0(self, x):
539             """Evaluate second part of a Hermite basis function with 0 derivative
540
541                 f10(x) = 2/delta_x * x^2
542
543                 Args:
544                     x := the shifted input variable
545
546                 Returns:
547                     f10(x)
548             """
549
550             return (x**2 * 2.0) / self._delta_x
551
552         def _lower_0(self, x):
553             """Evaluate lower part of a Lagrange basis function with 0 derivative
554
555                 fl0(x) = f00(x) + f10(x) + x
556
557                 Args:
558                     x := the shifted input variable
559
560                 Returns:
561                     fl0(x)
562             """
563
564             return self._part0_0(x) + self._part1_0(x) + x
565
566         def _upper_0(self, x):
567             """Evaluate lower part of a Lagrange basis function with 0 derivative
568
569                 fu0(x) = f00(x) - f10(x) + x
570
571                 Args:
572                     x := the shifted input variable
573
574                 Returns:
575                     fu0(x)
576             """
577
578             return self._part0_0(x) - self._part1_0(x) + x
579
580         def _part0_1(self, x):
581             """Evaluate first part of a Hermite basis function with 1 derivative
582
583                 f10(x) = 3/delta_x^2 * x^2
584
585                 Args:
586                     x := the shifted input variable
587
588                 Returns:
589                     f10(x)
590             """
591
592             return (x**2 * 3.0) / self.delta_x_2
593
594         def _part1_1(self, x):
595             """Evaluate second part of a Hermite basis function with 1 derivative
596
```

```
597             f11(x) = 4/delta_x * x
598
599             Args:
600                 x := the shifted input variable
601
602             Returns:
603                 f11(x)
604         """
605
606         return (x * 4.0) / self._delta_x
607
608     def _lower_1(self, x):
609         """Evaluate lower part of a Lagrange basis function with 1 derivative
610
611             fl1(x) = f10(x) + f11(x) + 1
612
613             Args:
614                 x := the shifted input variable
615
616             Returns:
617                 fl1(x)
618         """
619
620         return self._part0_1(x) + self._part1_1(x) + 1.0
621
622     def _upper_1(self, x):
623         """Evaluate upper part of a Lagrange basis function with 1 derivative
624
625             fu1(x) = f10(x) - f11(x) + 1
626
627             Args:
628                 x := the shifted input variable
629
630             Returns:
631                 fu1(x)
632         """
633
634         return self._part0_1(x) - self._part1_1(x) + 1.0
635
636     def _part0_2(self, x):
637         """Evaluate first part of a Hermite basis function with 2 derivative
638
639             f20(x) = 6/delta_x^2 * x
640
641             Args:
642                 x := the shifted input variable
643
644             Returns:
645                 f20(x)
646         """
647
648         return (x * 6.0) / self.delta_x_2
649
650     def _part1_2(self, x):
651         """Evaluate second part of a Hermite basis function with 2 derivative
652
653             f21(x) = 4/delta_x
654
655             Args:
656                 x := the shifted input variable
657
658             Returns:
659                 f21(x)
660         """
```

```
661
662         return 4.0 / self._delta_x
663
664     def _lower_2(self, x):
665         """Evaluate lower part of a Lagrange basis function with 2 derivative
666
667             fl2(x) = f20(x) + f21(x)
668
669             Args:
670                 x := the shifted input variable
671
672             Returns:
673                 fl1(x)
674         """
675
676         return self._part0_2(x) + self._part1_2(x)
677
678     def _upper_2(self, x):
679         """Evaluate upper part of a Lagrange basis function with 2 derivative
680
681             fu2(x) = f20(x) - f21(x)
682
683             Args:
684                 x := the shifted input variable
685
686             Returns:
687                 fu2(x)
688         """
689
690         return self._part0_2(x) - self._part1_2(x)
```

A.2 source/element.py

```
1  import numpy as np
2
3  from source import basis_function as basis
4
5
6  class Element(object):
7      """Class to contain a single element and its related matrices
8
9          Attributes:
10             _index      := index
11             _num_points := num_points
12             _delta_x    := delta_x
13
14             LL_00 := Lagrange_0_diff-Lagrange_0_diff
15             LL_01 := Lagrange_0_diff-Lagrange_1_diff
16             LL_02 := Lagrange_0_diff-Lagrange_2_diff
17             LL_10 := Lagrange_1_diff-Lagrange_0_diff
18             LL_11 := Lagrange_1_diff-Lagrange_1_diff
19             LL_12 := Lagrange_1_diff-Lagrange_2_diff
20             LL_20 := Lagrange_2_diff-Lagrange_0_diff
21             LL_21 := Lagrange_2_diff-Lagrange_1_diff
22             LL_22 := Lagrange_2_diff-Lagrange_2_diff
23
24             LH_00 := Lagrange_0_diff-Hermite_0_diff
25             LH_01 := Lagrange_0_diff-Hermite_1_diff
26             LH_02 := Lagrange_0_diff-Hermite_2_diff
27             LH_10 := Lagrange_1_diff-Hermite_0_diff
28             LH_11 := Lagrange_1_diff-Hermite_1_diff
29             LH_12 := Lagrange_1_diff-Hermite_2_diff
```

```
30              LH_20 := Lagrange_2_diff-Hermite_0_diff
31              LH_21 := Lagrange_2_diff-Hermite_1_diff
32              LH_22 := Lagrange_2_diff-Hermite_2_diff
33
34              HL_00 := Hermite_0_diff-Lagrange_0_diff
35              HL_01 := Hermite_0_diff-Lagrange_1_diff
36              HL_02 := Hermite_0_diff-Lagrange_2_diff
37              HL_10 := Hermite_1_diff-Lagrange_0_diff
38              HL_11 := Hermite_1_diff-Lagrange_1_diff
39              HL_12 := Hermite_1_diff-Lagrange_2_diff
40              HL_20 := Hermite_2_diff-Lagrange_0_diff
41              HL_21 := Hermite_2_diff-Lagrange_1_diff
42              HL_22 := Hermite_2_diff-Lagrange_2_diff
43
44              HH_00 := Hermite_0_diff-Hermite_0_diff
45              HH_01 := Hermite_0_diff-Hermite_1_diff
46              HH_02 := Hermite_0_diff-Hermite_2_diff
47              HH_10 := Hermite_1_diff-Hermite_0_diff
48              HH_11 := Hermite_1_diff-Hermite_1_diff
49              HH_12 := Hermite_1_diff-Hermite_2_diff
50              HH_20 := Hermite_2_diff-Hermite_0_diff
51              HH_21 := Hermite_2_diff-Hermite_1_diff
52              HH_22 := Hermite_2_diff-Hermite_2_diff
53
54              LLL_110 := Lagrange_1-Lagrange_1-Lagrange_0
55              LLL_200 := Lagrange_1-Lagrange_0-Lagrange_1
56
57              LLH_110 := Lagrange_1-Lagrange_1-Hermite_0
58              LLH_200 := Lagrange_1-Lagrange_0-Hermite_1
59
60              LHL_110 := Lagrange_1-Hermite_1-Lagrange_0
61              LHL_200 := Lagrange_1-Hermite_0-Lagrange_1
62
63              LHH_110 := Lagrange_1-Hermite_1-Hermite_0
64              LHH_200 := Lagrange_1-Hermite_0-Hermite_1
65
66              HLL_110 := Hermite_1-Lagrange_1-Lagrange_0
67              HLL_200 := Hermite_1-Lagrange_0-Lagrange_1
68
69              HLH_110 := Hermite_1-Lagrange_1-Hermite_0
70              HLH_200 := Hermite_1-Lagrange_0-Hermite_1
71
72              HHL_110 := Hermite_1-Hermite_1-Lagrange_0
73              HHL_200 := Hermite_1-Hermite_0-Lagrange_1
74
75              HHH_110 := Hermite_1-Hermite_1-Hermite_0
76              HHH_200 := Hermite_1-Hermite_0-Hermite_1
77      """
78
79      def __init__(self, index, num_points, delta_x):
80          self._index = index
81          self._num_points = num_points
82          self._delta_x = delta_x
83
84      @classmethod
85      def setup(cls, index, num_points, delta_x):
86          """Setup the element
87
88              Args:
89                  index      := index
90                  num_points := number of points
91                  delta_x    := distance between points
92
93              Returns:
```

```
94                  fully built element
95          """
96
97          new_element = cls(index, num_points, delta_x)
98          new_element._build()
99
100         return new_element
101
102     @property
103     def lower_bound(self):
104         """Get the lower bound of interval
105
106             Returns:
107                 lower bound of interval
108         """
109
110         return self._index * self._delta_x
111
112     @property
113     def upper_bound(self):
114         """Get the upper bound of interval
115
116             Returns:
117                 upper bound of interval
118         """
119
120         return (self._index + 1) * self._delta_x
121
122     @property
123     def basis_index(self):
124         """Get the indices for nonzero basis functions on element
125
126             Returns:
127                 list if indices for non-zero functions
128         """
129
130         return [self._index, (self._index + 1)]
131
132     def build_lagrange(self, index, derivative):
133         """Build a Lagrange basis function
134
135             Args:
136                 index      := index
137                 derivative := derivative
138
139             Returns:
140                 the Lagrange basis function with index and derivative
141         """
142
143         return basis.Lagrange(index, derivative,
144                               self._num_points, self._delta_x)
145
146     def build_hermite(self, index, derivative):
147         """Build a Hermite basis function
148
149             Args:
150                 index      := index
151                 derivative := derivative
152
153             Returns:
154                 the Hermite Lagrange basis function with index and derivative
155         """
156
157         return basis.Hermite(index, derivative,
```

```
158                                 self._num_points, self._delta_x)
159
160     def func_pl(self, p, index0, diff0):
161         """Find the P-lagrange product
162
163             f = L(index0, diff0) * p
164
165             Args:
166                 p      := the function for product
167                 index0 := index of basis function
168                 diff0  := derivative of basis function
169
170             Returns:
171                 product of Lagrange basis with p i.e. f
172         """
173
174         function0 = self.build_lagrange(index0, diff0)
175         function1 = p
176
177         return function0 * function1
178
179     def func_ph(self, p, index0, diff0):
180         """Find the P-hermite product
181
182             g = H(index0, diff0) * p
183
184             Args:
185                 p      := the function for product
186                 index0 := index of basis function
187                 diff0  := derivative of basis function
188
189             Returns:
190                 product of Hermite basis with p i.e. g
191         """
192
193         function0 = self.build_hermite(index0, diff0)
194         function1 = p
195
196         return function0 * function1
197
198     def func_ll(self, index0, index1, diff0, diff1):
199         """Find the Lagrange-Lagrange product
200
201             fll = L(index0, diff0) * L(index1, diff1)
202
203             Args:
204                 index0 := index of first basis function
205                 index1 := index of second basis function
206                 diff0  := derivative of first basis function
207                 diff1  := derivative of second basis function
208
209             Returns:
210                 product of the basis functions i.e. fll
211         """
212
213         function0 = self.build_lagrange(index0, diff0)
214         function1 = self.build_lagrange(index1, diff1)
215
216         return function0 * function1
217
218     def func_lh(self, index0, index1, diff0, diff1):
219         """Find the Lagrange-Hermite product
220
221             flh = L(index0, diff0) * H(index1, diff1)
```

```
222
223             Args:
224                 index0 := index of first basis function
225                 index1 := index of second basis function
226                 diff0  := derivative of first basis function
227                 diff1  := derivative of second basis function
228
229             Returns:
230                 product of the basis functions i.e. flh
231         """
232
233         function0 = self.build_lagrange(index0, diff0)
234         function1 = self.build_hermite(index1, diff1)
235
236         return function0 * function1
237
238     def func_hl(self, index0, index1, diff0, diff1):
239         """Find the Hermite-Lagrange product
240
241             fhl = H(index0, diff0) * L(index1, diff1)
242
243             Args:
244                 index0 := index of first basis function
245                 index1 := index of second basis function
246                 diff0  := derivative of first basis function
247                 diff1  := derivative of second basis function
248
249             Returns:
250                 product of the basis functions i.e. fhl
251         """
252
253         function0 = self.build_hermite(index0, diff0)
254         function1 = self.build_lagrange(index1, diff1)
255
256         return function0 * function1
257
258     def func_hh(self, index0, index1, diff0, diff1):
259         """Find the Hermite-Hermite product
260
261             fhh = H(index0, diff0) * H(index1, diff1)
262
263             Args:
264                 index0 := index of first basis function
265                 index1 := index of second basis function
266                 diff0  := derivative of first basis function
267                 diff1  := derivative of second basis function
268
269             Returns:
270                 product of the basis functions i.e. fhh
271         """
272
273         function0 = self.build_hermite(index0, diff0)
274         function1 = self.build_hermite(index1, diff1)
275
276         return function0 * function1
277
278     def func_lll(self, index0, index1, index2, diff0, diff1, diff2):
279         """Find the Lagrange-Lagrange-Lagrange product
280
281             f_lll = L(index0, diff0) * L(index1, diff1) * L(index2, diff2)
282
283             Args:
284                 index0 := index of first basis function
285                 index1 := index of second basis function
```

```
286                 index2 := index of third basis function
287                 diff0  := derivative of first basis function
288                 diff1  := derivative of second basis function
289                 diff2  := derivative of third basis function
290 
291             Returns:
292                 product of the basis functions i.e. f_lll
293         """
294 
295         function0 = self.func_ll(index0, index1, diff0, diff1)
296         function1 = self.build_lagrange(index2, diff2)
297 
298         return function0 * function1
299 
300     def func_llh(self, index0, index1, index2, diff0, diff1, diff2):
301         """Find the Lagrange-Lagrange-Hermite product
302 
303             f_llh = L(index0, diff0) * L(index1, diff1) * H(index2, diff2)
304 
305             Args:
306                 index0 := index of first basis function
307                 index1 := index of second basis function
308                 index2 := index of third basis function
309                 diff0  := derivative of first basis function
310                 diff1  := derivative of second basis function
311                 diff2  := derivative of third basis function
312 
313             Returns:
314                 product of the basis functions i.e. f_llh
315         """
316 
317         function0 = self.func_ll(index0, index1, diff0, diff1)
318         function1 = self.build_hermite(index2, diff2)
319 
320         return function0 * function1
321 
322     def func_lhl(self, index0, index1, index2, diff0, diff1, diff2):
323         """Find the Lagrange-Hermite-Lagrange product
324 
325             f_lhl = L(index0, diff0) * H(index1, diff1) * L(index2, diff2)
326 
327             Args:
328                 index0 := index of first basis function
329                 index1 := index of second basis function
330                 index2 := index of third basis function
331                 diff0  := derivative of first basis function
332                 diff1  := derivative of second basis function
333                 diff2  := derivative of third basis function
334 
335             Returns:
336                 product of the basis functions i.e. f_lhl
337         """
338 
339         function0 = self.func_lh(index0, index1, diff0, diff1)
340         function1 = self.build_lagrange(index2, diff2)
341 
342         return function0 * function1
343 
344     def func_lhh(self, index0, index1, index2, diff0, diff1, diff2):
345         """Find the Lagrange-Hermite-Hermite product
346 
347             f_lhh = L(index0, diff0) * H(index1, diff1) * H(index2, diff2)
348 
349             Args:
```

```
350                 index0 := index of first basis function
351                 index1 := index of second basis function
352                 index2 := index of third basis function
353                 diff0  := derivative of first basis function
354                 diff1  := derivative of second basis function
355                 diff2  := derivative of third basis function
356
357             Returns:
358                 product of the basis functions i.e. f_lhh
359         """
360
361         function0 = self.func_lh(index0, index1, diff0, diff1)
362         function1 = self.build_hermite(index2, diff2)
363
364         return function0 * function1
365
366     def func_hll(self, index0, index1, index2, diff0, diff1, diff2):
367         """Find the Hermite-Lagrange-Lagrange product
368
369             f_hll = H(index0, diff0) * L(index1, diff1) * L(index2, diff2)
370
371             Args:
372                 index0 := index of first basis function
373                 index1 := index of second basis function
374                 index2 := index of third basis function
375                 diff0  := derivative of first basis function
376                 diff1  := derivative of second basis function
377                 diff2  := derivative of third basis function
378
379             Returns:
380                 product of the basis functions i.e. f_hll
381         """
382
383         function0 = self.func_hl(index0, index1, diff0, diff1)
384         function1 = self.build_lagrange(index2, diff2)
385
386         return function0 * function1
387
388     def func_hlh(self, index0, index1, index2, diff0, diff1, diff2):
389         """Find the Hermite-Lagrange-Hermite product
390
391             f_hlh = H(index0, diff0) * L(index1, diff1) * H(index2, diff2)
392
393             Args:
394                 index0 := index of first basis function
395                 index1 := index of second basis function
396                 index2 := index of third basis function
397                 diff0  := derivative of first basis function
398                 diff1  := derivative of second basis function
399                 diff2  := derivative of third basis function
400
401             Returns:
402                 product of the basis functions i.e. f_hlh
403         """
404
405         function0 = self.func_hl(index0, index1, diff0, diff1)
406         function1 = self.build_hermite(index2, diff2)
407
408         return function0 * function1
409
410     def func_hhl(self, index0, index1, index2, diff0, diff1, diff2):
411         """Find the Hermite-Hermite-Lagrange product
412
413             f_hhl = H(index0, diff0) * H(index1, diff1) * L(index2, diff2)
```

```
414
415             Args:
416                 index0 := index of first basis function
417                 index1 := index of second basis function
418                 index2 := index of third basis function
419                 diff0  := derivative of first basis function
420                 diff1  := derivative of second basis function
421                 diff2  := derivative of third basis function
422
423             Returns:
424                 product of the basis functions i.e. f_hhl
425         """
426
427         function0 = self.func_hh(index0, index1, diff0, diff1)
428         function1 = self.build_lagrange(index2, diff2)
429
430         return function0 * function1
431
432     def func_hhh(self, index0, index1, index2, diff0, diff1, diff2):
433         """Find the Hermite-Hermite-Hermite product
434
435             f_hhh = H(index0, diff0) * H(index1, diff1) * H(index2, diff2)
436
437             Args:
438                 index0 := index of first basis function
439                 index1 := index of second basis function
440                 index2 := index of third basis function
441                 diff0  := derivative of first basis function
442                 diff1  := derivative of second basis function
443                 diff2  := derivative of third basis function
444
445             Returns:
446                 product of the basis functions i.e. f_hhh
447         """
448
449         function0 = self.func_hh(index0, index1, diff0, diff1)
450         function1 = self.build_hermite(index2, diff2)
451
452         return function0 * function1
453
454     def pl_2(self, p):
455         """Build the PL_2 array
456
457             PL_2 = [L(0, 2) * p, L(1, 2)]
458
459             Args:
460                 p := the function to build array
461
462             Returns:
463                 array of products with Lagrange, i.e. PL_2
464         """
465
466         basis_index0 = self.basis_index[0]
467         basis_index1 = self.basis_index[1]
468
469         func0 = self.func_pl(p, basis_index0, 2)
470         func1 = self.func_pl(p, basis_index1, 2)
471
472         point_0 = func0.integral(self.lower_bound, self.upper_bound)
473         point_1 = func1.integral(self.lower_bound, self.upper_bound)
474
475         return np.array([point_0, point_1])
476
477     def ph_2(self, p):
```

```
478         """Build the PH_2 array
479 
480             PH_2 = [H(0, 2) * p, H(1, 2)]
481 
482             Args:
483                 p := the function to build array
484 
485             Returns:
486                 array of products with Hermite, i.e. PH_2
487         """
488 
489         basis_index0 = self.basis_index[0]
490         basis_index1 = self.basis_index[1]
491 
492         func0 = self.func_ph(p, basis_index0, 2)
493         func1 = self.func_ph(p, basis_index1, 2)
494 
495         point_0 = func0.integral(self.lower_bound, self.upper_bound)
496         point_1 = func1.integral(self.lower_bound, self.upper_bound)
497 
498         return np.array([point_0, point_1])
499 
500     def _build_ll_00(self):
501         """Build the LL_00 matrix
502 
503             LL_00 = [[L(0, 0) * L(0, 0), L(0, 0) * L(1, 0)],
504                      [L(1, 0) * L(0, 0), L(1, 0) * L(1, 0)]]
505 
506             Sets:
507                 LL_00
508 
509             Returns:
510                 None
511         """
512 
513         basis_index0 = self.basis_index[0]
514         basis_index1 = self.basis_index[1]
515 
516         func_00 = self.func_ll(basis_index0, basis_index0, 0, 0)
517         func_01 = self.func_ll(basis_index0, basis_index1, 0, 0)
518         func_10 = self.func_ll(basis_index1, basis_index0, 0, 0)
519         func_11 = self.func_ll(basis_index1, basis_index1, 0, 0)
520 
521         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
522         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
523         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
524         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
525 
526         self.LL_00 = np.array([[point_00, point_01],
527                                [point_10, point_11]])
528 
529     def _build_ll_01(self):
530         """Build the LL_01 matrix
531 
532             LL_01 = [[L(0, 0) * L(0, 1), L(0, 0) * L(1, 1)],
533                      [L(1, 0) * L(0, 1), L(1, 0) * L(1, 1)]]
534 
535             Sets:
536                 LL_01
537 
538             Returns:
539                 None
540         """
541 
```

```
542         basis_index0 = self.basis_index[0]
543         basis_index1 = self.basis_index[1]
544
545         func_00 = self.func_ll(basis_index0, basis_index0, 0, 1)
546         func_01 = self.func_ll(basis_index0, basis_index1, 0, 1)
547         func_10 = self.func_ll(basis_index1, basis_index0, 0, 1)
548         func_11 = self.func_ll(basis_index1, basis_index1, 0, 1)
549
550         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
551         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
552         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
553         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
554
555         self.LL_01 = np.array([[point_00, point_01],
556                                [point_10, point_11]])
557
558     def _build_ll_02(self):
559         """Build the LL_02 matrix
560
561             LL_02 = [[L(0, 0) * L(0, 2), L(0, 0) * L(1, 2)],
562                      [L(1, 0) * L(0, 2), L(1, 0) * L(1, 2)]]
563
564             Sets:
565                 LL_02
566
567             Returns:
568                 None
569         """
570
571         basis_index0 = self.basis_index[0]
572         basis_index1 = self.basis_index[1]
573
574         func_00 = self.func_ll(basis_index0, basis_index0, 0, 2)
575         func_01 = self.func_ll(basis_index0, basis_index1, 0, 2)
576         func_10 = self.func_ll(basis_index1, basis_index0, 0, 2)
577         func_11 = self.func_ll(basis_index1, basis_index1, 0, 2)
578
579         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
580         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
581         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
582         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
583
584         self.LL_02 = np.array([[point_00, point_01],
585                                [point_10, point_11]])
586
587     def _build_ll_10(self):
588         """Build the LL_10 matrix
589
590             LL_10 = [[L(0, 1) * L(0, 0), L(0, 1) * L(1, 0)],
591                      [L(1, 1) * L(0, 0), L(1, 1) * L(1, 0)]]
592
593             Sets:
594                 LL_10
595
596             Returns:
597                 None
598         """
599
600         basis_index0 = self.basis_index[0]
601         basis_index1 = self.basis_index[1]
602
603         func_00 = self.func_ll(basis_index0, basis_index0, 1, 0)
604         func_01 = self.func_ll(basis_index0, basis_index1, 1, 0)
605         func_10 = self.func_ll(basis_index1, basis_index0, 1, 0)
```

```
606         func_11 = self.func_ll(basis_index1, basis_index1, 1, 0)
607
608         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
609         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
610         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
611         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
612
613         self.LL_10 = np.array([[point_00, point_01],
614                                [point_10, point_11]])
615
616     def _build_ll_11(self):
617         """Build the LL_11 matrix
618
619             LL_11 = [[L(0, 1) * L(0, 1), L(1, 1) * L(1, 1)],
620                      [L(1, 1) * L(0, 1), L(1, 1) * L(1, 1)]]
621
622             Sets:
623                 LL_11
624
625             Returns:
626                 None
627         """
628
629         basis_index0 = self.basis_index[0]
630         basis_index1 = self.basis_index[1]
631
632         func_00 = self.func_ll(basis_index0, basis_index0, 1, 1)
633         func_01 = self.func_ll(basis_index0, basis_index1, 1, 1)
634         func_10 = self.func_ll(basis_index1, basis_index0, 1, 1)
635         func_11 = self.func_ll(basis_index1, basis_index1, 1, 1)
636
637         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
638         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
639         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
640         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
641
642         self.LL_11 = np.array([[point_00, point_01],
643                                [point_10, point_11]])
644
645     def _build_ll_12(self):
646         """Build the LL_12 matrix
647
648             LL_12 = [[L(0, 1) * L(0, 2), L(0, 1) * L(1, 2)],
649                      [L(1, 1) * L(0, 2), L(1, 1) * L(1, 2)]]
650
651             Sets:
652                 LL_12
653
654             Returns:
655                 None
656         """
657
658         basis_index0 = self.basis_index[0]
659         basis_index1 = self.basis_index[1]
660
661         func_00 = self.func_ll(basis_index0, basis_index0, 1, 2)
662         func_01 = self.func_ll(basis_index0, basis_index1, 1, 2)
663         func_10 = self.func_ll(basis_index1, basis_index0, 1, 2)
664         func_11 = self.func_ll(basis_index1, basis_index1, 1, 2)
665
666         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
667         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
668         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
669         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
```

```
670
671         self.LL_12 = np.array([[point_00, point_01],
672                                [point_10, point_11]])
673
674     def _build_ll_20(self):
675         """Build the LL_20 matrix
676
677             LL_20 = [[L(0, 2) * L(0, 0), L(0, 2) * L(1, 0)],
678                      [L(1, 2) * L(0, 0), L(1, 2) * L(1, 0)]]
679
680             Sets:
681                 LL_20
682
683             Returns:
684                 None
685         """
686
687         basis_index0 = self.basis_index[0]
688         basis_index1 = self.basis_index[1]
689
690         func_00 = self.func_ll(basis_index0, basis_index0, 2, 0)
691         func_01 = self.func_ll(basis_index0, basis_index1, 2, 0)
692         func_10 = self.func_ll(basis_index1, basis_index0, 2, 0)
693         func_11 = self.func_ll(basis_index1, basis_index1, 2, 0)
694
695         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
696         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
697         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
698         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
699
700         self.LL_20 = np.array([[point_00, point_01],
701                                [point_10, point_11]])
702
703     def _build_ll_21(self):
704         """Build the LL_21 matrix
705
706             LL_21 = [[L(0, 2) * L(0, 1), L(0, 2) * L(1, 1)],
707                      [L(1, 2) * L(0, 1), L(1, 2) * L(1, 1)]]
708
709             Sets:
710                 LL_21
711
712             Returns:
713                 None
714         """
715
716         basis_index0 = self.basis_index[0]
717         basis_index1 = self.basis_index[1]
718
719         func_00 = self.func_ll(basis_index0, basis_index0, 2, 1)
720         func_01 = self.func_ll(basis_index0, basis_index1, 2, 1)
721         func_10 = self.func_ll(basis_index1, basis_index0, 2, 1)
722         func_11 = self.func_ll(basis_index1, basis_index1, 2, 1)
723
724         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
725         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
726         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
727         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
728
729         self.LL_21 = np.array([[point_00, point_01],
730                                [point_10, point_11]])
731
732     def _build_ll_22(self):
733         """Build the LL_22 matrix
```

```
734
735             LL_22 = [[L(0, 2) * L(0, 2), L(0, 2) * L(1, 2)],
736                      [L(1, 2) * L(0, 2), L(1, 2) * L(1, 2)]]
737
738             Sets:
739                 LL_22
740
741             Returns:
742                 None
743         """
744
745         basis_index0 = self.basis_index[0]
746         basis_index1 = self.basis_index[1]
747
748         func_00 = self.func_ll(basis_index0, basis_index0, 2, 2)
749         func_01 = self.func_ll(basis_index0, basis_index1, 2, 2)
750         func_10 = self.func_ll(basis_index1, basis_index0, 2, 2)
751         func_11 = self.func_ll(basis_index1, basis_index1, 2, 2)
752
753         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
754         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
755         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
756         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
757
758         self.LL_22 = np.array([[point_00, point_01],
759                                [point_10, point_11]])
760
761     def _build_lh_00(self):
762         """Build the LH_00 matrix
763
764             LH_00 = [[L(0, 0) * H(0, 0), L(0, 0) * H(1, 0)],
765                      [L(1, 0) * H(0, 0), L(1, 0) * H(1, 0)]]
766
767             Sets:
768                 LH_00
769
770             Returns:
771                 None
772         """
773
774         basis_index0 = self.basis_index[0]
775         basis_index1 = self.basis_index[1]
776
777         func_00 = self.func_lh(basis_index0, basis_index0, 0, 0)
778         func_01 = self.func_lh(basis_index0, basis_index1, 0, 0)
779         func_10 = self.func_lh(basis_index1, basis_index0, 0, 0)
780         func_11 = self.func_lh(basis_index1, basis_index1, 0, 0)
781
782         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
783         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
784         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
785         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
786
787         self.LH_00 = np.array([[point_00, point_01],
788                                [point_10, point_11]])
789
790     def _build_lh_01(self):
791         """Build the LH_01 matrix
792
793             LH_01 = [[L(0, 0) * H(0, 1), L(0, 0) * H(1, 1)],
794                      [L(1, 0) * H(0, 1), L(1, 0) * H(1, 1)]]
795
796             Sets:
797                 LH_01
```

```
798
799             Returns:
800                 None
801         """
802
803         basis_index0 = self.basis_index[0]
804         basis_index1 = self.basis_index[1]
805
806         func_00 = self.func_lh(basis_index0, basis_index0, 0, 1)
807         func_01 = self.func_lh(basis_index0, basis_index1, 0, 1)
808         func_10 = self.func_lh(basis_index1, basis_index0, 0, 1)
809         func_11 = self.func_lh(basis_index1, basis_index1, 0, 1)
810
811         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
812         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
813         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
814         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
815
816         self.LH_01 = np.array([[point_00, point_01],
817                                [point_10, point_11]])
818
819     def _build_lh_02(self):
820         """Build the LH_02 matrix
821
822             LH_02 = [[L(0, 0) * H(0, 2), L(0, 0) * H(1, 2)],
823                      [L(1, 0) * H(0, 2), L(1, 0) * H(1, 2)]]
824
825             Sets:
826                 LH_02
827
828             Returns:
829                 None
830         """
831
832         basis_index0 = self.basis_index[0]
833         basis_index1 = self.basis_index[1]
834
835         func_00 = self.func_lh(basis_index0, basis_index0, 0, 2)
836         func_01 = self.func_lh(basis_index0, basis_index1, 0, 2)
837         func_10 = self.func_lh(basis_index1, basis_index0, 0, 2)
838         func_11 = self.func_lh(basis_index1, basis_index1, 0, 2)
839
840         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
841         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
842         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
843         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
844
845         self.LH_02 = np.array([[point_00, point_01],
846                                [point_10, point_11]])
847
848     def _build_lh_10(self):
849         """Build the LH_10 matrix
850
851             LH_10 = [[L(0, 1) * H(0, 0), L(0, 1) * H(1, 0)],
852                      [L(1, 1) * H(0, 0), L(1, 1) * H(1, 0)]]
853
854             Sets:
855                 LH_10
856
857             Returns:
858                 None
859         """
860
861         basis_index0 = self.basis_index[0]
```

```
862             basis_index1 = self.basis_index[1]
863
864             func_00 = self.func_lh(basis_index0, basis_index0, 1, 0)
865             func_01 = self.func_lh(basis_index0, basis_index1, 1, 0)
866             func_10 = self.func_lh(basis_index1, basis_index0, 1, 0)
867             func_11 = self.func_lh(basis_index1, basis_index1, 1, 0)
868
869             point_00 = func_00.integral(self.lower_bound, self.upper_bound)
870             point_01 = func_01.integral(self.lower_bound, self.upper_bound)
871             point_10 = func_10.integral(self.lower_bound, self.upper_bound)
872             point_11 = func_11.integral(self.lower_bound, self.upper_bound)
873
874             self.LH_10 = np.array([[point_00, point_01],
875                                    [point_10, point_11]])
876
877         def _build_lh_11(self):
878             """Build the LH_11 matrix
879
880                 LH_11 = [[L(0, 1) * H(0, 1), L(0, 1) * H(1, 1)],
881                          [L(1, 1) * H(0, 1), L(1, 1) * H(1, 1)]]
882
883                 Sets:
884                     LH_11
885
886                 Returns:
887                     None
888             """
889
890             basis_index0 = self.basis_index[0]
891             basis_index1 = self.basis_index[1]
892
893             func_00 = self.func_lh(basis_index0, basis_index0, 1, 1)
894             func_01 = self.func_lh(basis_index0, basis_index1, 1, 1)
895             func_10 = self.func_lh(basis_index1, basis_index0, 1, 1)
896             func_11 = self.func_lh(basis_index1, basis_index1, 1, 1)
897
898             point_00 = func_00.integral(self.lower_bound, self.upper_bound)
899             point_01 = func_01.integral(self.lower_bound, self.upper_bound)
900             point_10 = func_10.integral(self.lower_bound, self.upper_bound)
901             point_11 = func_11.integral(self.lower_bound, self.upper_bound)
902
903             self.LH_11 = np.array([[point_00, point_01],
904                                    [point_10, point_11]])
905
906         def _build_lh_12(self):
907             """Build the LH_12 matrix
908
909                 LH_12 = [[L(0, 1) * H(0, 2), L(0, 1) * H(1, 2)],
910                          [L(1, 1) * H(0, 2), L(1, 1) * H(1, 2)]]
911
912                 Sets:
913                     LH_12
914
915                 Returns:
916                     None
917             """
918
919             basis_index0 = self.basis_index[0]
920             basis_index1 = self.basis_index[1]
921
922             func_00 = self.func_lh(basis_index0, basis_index0, 1, 2)
923             func_01 = self.func_lh(basis_index0, basis_index1, 1, 2)
924             func_10 = self.func_lh(basis_index1, basis_index0, 1, 2)
925             func_11 = self.func_lh(basis_index1, basis_index1, 1, 2)
```

```
926
927             point_00 = func_00.integral(self.lower_bound, self.upper_bound)
928             point_01 = func_01.integral(self.lower_bound, self.upper_bound)
929             point_10 = func_10.integral(self.lower_bound, self.upper_bound)
930             point_11 = func_11.integral(self.lower_bound, self.upper_bound)
931
932             self.LH_12 = np.array([[point_00, point_01],
933                                    [point_10, point_11]])
934
935         def _build_lh_20(self):
936             """Build the LH_20 matrix
937
938                 LH_20 = [[L(0, 2) * H(0, 0), L(0, 2) * H(1, 0)],
939                          [L(1, 2) * H(0, 0), L(1, 2) * H(1, 0)]]
940
941                 Sets:
942                     LH_20
943
944                 Returns:
945                     None
946             """
947
948             basis_index0 = self.basis_index[0]
949             basis_index1 = self.basis_index[1]
950
951             func_00 = self.func_lh(basis_index0, basis_index0, 2, 0)
952             func_01 = self.func_lh(basis_index0, basis_index1, 2, 0)
953             func_10 = self.func_lh(basis_index1, basis_index0, 2, 0)
954             func_11 = self.func_lh(basis_index1, basis_index1, 2, 0)
955
956             point_00 = func_00.integral(self.lower_bound, self.upper_bound)
957             point_01 = func_01.integral(self.lower_bound, self.upper_bound)
958             point_10 = func_10.integral(self.lower_bound, self.upper_bound)
959             point_11 = func_11.integral(self.lower_bound, self.upper_bound)
960
961             self.LH_20 = np.array([[point_00, point_01],
962                                    [point_10, point_11]])
963
964         def _build_lh_21(self):
965             """Build the LH_21 matrix
966
967                 LH_21 = [[L(0, 2) * H(0, 1), L(0, 2) * H(1, 1)],
968                          [L(1, 2) * H(0, 1), L(1, 2) * H(1, 1)]]
969
970                 Sets:
971                     LH_21
972
973                 Returns:
974                     None
975             """
976
977             basis_index0 = self.basis_index[0]
978             basis_index1 = self.basis_index[1]
979
980             func_00 = self.func_lh(basis_index0, basis_index0, 2, 1)
981             func_01 = self.func_lh(basis_index0, basis_index1, 2, 1)
982             func_10 = self.func_lh(basis_index1, basis_index0, 2, 1)
983             func_11 = self.func_lh(basis_index1, basis_index1, 2, 1)
984
985             point_00 = func_00.integral(self.lower_bound, self.upper_bound)
986             point_01 = func_01.integral(self.lower_bound, self.upper_bound)
987             point_10 = func_10.integral(self.lower_bound, self.upper_bound)
988             point_11 = func_11.integral(self.lower_bound, self.upper_bound)
989
```

```
990             self.LH_21 = np.array([[point_00, point_01],
991                                    [point_10, point_11]])
992
993         def _build_lh_22(self):
994             """Build the LH_22 matrix
995
996                 LH_22 = [[L(0, 2) * H(0, 2), L(0, 2) * H(1, 2)],
997                          [L(1, 2) * H(0, 2), L(1, 2) * H(1, 2)]]
998
999                 Sets:
1000                    LH_22
1001
1002                Returns:
1003                    None
1004            """
1005
1006            basis_index0 = self.basis_index[0]
1007            basis_index1 = self.basis_index[1]
1008
1009            func_00 = self.func_lh(basis_index0, basis_index0, 2, 2)
1010            func_01 = self.func_lh(basis_index0, basis_index1, 2, 2)
1011            func_10 = self.func_lh(basis_index1, basis_index0, 2, 2)
1012            func_11 = self.func_lh(basis_index1, basis_index1, 2, 2)
1013
1014            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1015            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1016            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1017            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1018
1019            self.LH_22 = np.array([[point_00, point_01],
1020                                   [point_10, point_11]])
1021
1022        def _build_hl_00(self):
1023            """Build the HL_00 matrix
1024
1025                HL_00 = [[H(0, 0) * L(0, 0), H(0, 0) * L(1, 0)],
1026                         [H(1, 0) * L(0, 0), H(1, 0) * L(1, 0)]]
1027
1028                Sets:
1029                    HL_00
1030
1031                Returns:
1032                    None
1033            """
1034
1035            basis_index0 = self.basis_index[0]
1036            basis_index1 = self.basis_index[1]
1037
1038            func_00 = self.func_hl(basis_index0, basis_index0, 0, 0)
1039            func_01 = self.func_hl(basis_index0, basis_index1, 0, 0)
1040            func_10 = self.func_hl(basis_index1, basis_index0, 0, 0)
1041            func_11 = self.func_hl(basis_index1, basis_index1, 0, 0)
1042
1043            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1044            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1045            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1046            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1047
1048            self.HL_00 = np.array([[point_00, point_01],
1049                                   [point_10, point_11]])
1050
1051        def _build_hl_01(self):
1052            """Build the HL_01 matrix
1053
```

```
1054                HL_01 = [[H(0, 0) * L(0, 1), H(0, 0) * L(1, 1)],
1055                         [H(1, 0) * L(0, 1), H(1, 0) * L(1, 1)]]
1056
1057                Sets:
1058                    HL_01
1059
1060                Returns:
1061                    None
1062            """
1063
1064            basis_index0 = self.basis_index[0]
1065            basis_index1 = self.basis_index[1]
1066
1067            func_00 = self.func_hl(basis_index0, basis_index0, 0, 1)
1068            func_01 = self.func_hl(basis_index0, basis_index1, 0, 1)
1069            func_10 = self.func_hl(basis_index1, basis_index0, 0, 1)
1070            func_11 = self.func_hl(basis_index1, basis_index1, 0, 1)
1071
1072            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1073            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1074            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1075            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1076
1077            self.HL_01 = np.array([[point_00, point_01],
1078                                   [point_10, point_11]])
1079
1080        def _build_hl_02(self):
1081            """Build the HL_02 matrix
1082
1083                HL_02 = [[H(0, 0) * L(0, 2), H(0, 0) * L(1, 2)],
1084                         [H(1, 0) * L(0, 2), H(1, 0) * L(1, 2)]]
1085
1086                Sets:
1087                    HL_02
1088
1089                Returns:
1090                    None
1091            """
1092
1093            basis_index0 = self.basis_index[0]
1094            basis_index1 = self.basis_index[1]
1095
1096            func_00 = self.func_hl(basis_index0, basis_index0, 0, 2)
1097            func_01 = self.func_hl(basis_index0, basis_index1, 0, 2)
1098            func_10 = self.func_hl(basis_index1, basis_index0, 0, 2)
1099            func_11 = self.func_hl(basis_index1, basis_index1, 0, 2)
1100
1101            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1102            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1103            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1104            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1105
1106            self.HL_02 = np.array([[point_00, point_01],
1107                                   [point_10, point_11]])
1108
1109        def _build_hl_10(self):
1110            """Build the HL_10 matrix
1111
1112                HL_10 = [[H(0, 1) * L(0, 0), H(0, 1) * L(1, 0)],
1113                         [H(1, 1) * L(0, 0), H(1, 1) * L(1, 0)]]
1114
1115                Sets:
1116                    HL_10
1117
```

```
1118            Returns:
1119                None
1120        """
1121
1122        basis_index0 = self.basis_index[0]
1123        basis_index1 = self.basis_index[1]
1124
1125        func_00 = self.func_hl(basis_index0, basis_index0, 1, 0)
1126        func_01 = self.func_hl(basis_index0, basis_index1, 1, 0)
1127        func_10 = self.func_hl(basis_index1, basis_index0, 1, 0)
1128        func_11 = self.func_hl(basis_index1, basis_index1, 1, 0)
1129
1130        point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1131        point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1132        point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1133        point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1134
1135        self.HL_10 = np.array([[point_00, point_01],
1136                               [point_10, point_11]])
1137
1138    def _build_hl_11(self):
1139        """Build the HL_11 matrix
1140
1141            HL_11 = [[H(0, 1) * L(0, 1), H(0, 1) * L(1, 1)],
1142                     [H(1, 1) * L(0, 1), H(1, 1) * L(1, 1)]]
1143
1144            Sets:
1145                HL_11
1146
1147            Returns:
1148                None
1149        """
1150
1151        basis_index0 = self.basis_index[0]
1152        basis_index1 = self.basis_index[1]
1153
1154        func_00 = self.func_hl(basis_index0, basis_index0, 1, 1)
1155        func_01 = self.func_hl(basis_index0, basis_index1, 1, 1)
1156        func_10 = self.func_hl(basis_index1, basis_index0, 1, 1)
1157        func_11 = self.func_hl(basis_index1, basis_index1, 1, 1)
1158
1159        point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1160        point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1161        point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1162        point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1163
1164        self.HL_11 = np.array([[point_00, point_01],
1165                               [point_10, point_11]])
1166
1167    def _build_hl_12(self):
1168        """Build the HL_12 matrix
1169
1170            HL_12 = [[H(0, 1) * L(0, 2), H(0, 1) * L(1, 2)],
1171                     [H(1, 1) * L(0, 2), H(1, 1) * L(1, 2)]]
1172
1173            Sets:
1174                HL_12
1175
1176            Returns:
1177                None
1178        """
1179
1180        basis_index0 = self.basis_index[0]
1181        basis_index1 = self.basis_index[1]
```

```
1182
1183            func_00 = self.func_hl(basis_index0, basis_index0, 1, 2)
1184            func_01 = self.func_hl(basis_index0, basis_index1, 1, 2)
1185            func_10 = self.func_hl(basis_index1, basis_index0, 1, 2)
1186            func_11 = self.func_hl(basis_index1, basis_index1, 1, 2)
1187
1188            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1189            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1190            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1191            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1192
1193            self.HL_12 = np.array([[point_00, point_01],
1194                                   [point_10, point_11]])
1195
1196        def _build_hl_20(self):
1197            """Build the HL_20 matrix
1198
1199                HL_20 = [[H(0, 2) * L(0, 0), H(0, 2) * L(1, 0)],
1200                         [H(1, 2) * L(0, 0), H(1, 2) * L(1, 0)]]
1201
1202                Sets:
1203                    HL_20
1204
1205                Returns:
1206                    None
1207            """
1208
1209            basis_index0 = self.basis_index[0]
1210            basis_index1 = self.basis_index[1]
1211
1212            func_00 = self.func_hl(basis_index0, basis_index0, 2, 0)
1213            func_01 = self.func_hl(basis_index0, basis_index1, 2, 0)
1214            func_10 = self.func_hl(basis_index1, basis_index0, 2, 0)
1215            func_11 = self.func_hl(basis_index1, basis_index1, 2, 0)
1216
1217            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1218            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1219            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1220            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1221
1222            self.HL_20 = np.array([[point_00, point_01],
1223                                   [point_10, point_11]])
1224
1225        def _build_hl_21(self):
1226            """Build the HL_21 matrix
1227
1228                HL_21 = [[H(0, 2) * L(0, 1), H(0, 2) * L(1, 1)],
1229                         [H(1, 2) * L(0, 1), H(1, 2) * L(1, 1)]]
1230
1231                Sets:
1232                    HL_21
1233
1234                Returns:
1235                    None
1236            """
1237
1238            basis_index0 = self.basis_index[0]
1239            basis_index1 = self.basis_index[1]
1240
1241            func_00 = self.func_hl(basis_index0, basis_index0, 2, 1)
1242            func_01 = self.func_hl(basis_index0, basis_index1, 2, 1)
1243            func_10 = self.func_hl(basis_index1, basis_index0, 2, 1)
1244            func_11 = self.func_hl(basis_index1, basis_index1, 2, 1)
1245
```

```
1246            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1247            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1248            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1249            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1250
1251            self.HL_21 = np.array([[point_00, point_01],
1252                                   [point_10, point_11]])
1253
1254        def _build_hl_22(self):
1255            """Build the HL_22 matrix
1256
1257                HL_22 = [[H(0, 2) * L(0, 2), H(0, 2) * L(1, 2)],
1258                         [H(1, 2) * L(0, 2), H(1, 2) * L(1, 2)]]
1259
1260                Sets:
1261                    HL_22
1262
1263                Returns:
1264                    None
1265            """
1266
1267            basis_index0 = self.basis_index[0]
1268            basis_index1 = self.basis_index[1]
1269
1270            func_00 = self.func_hl(basis_index0, basis_index0, 2, 2)
1271            func_01 = self.func_hl(basis_index0, basis_index1, 2, 2)
1272            func_10 = self.func_hl(basis_index1, basis_index0, 2, 2)
1273            func_11 = self.func_hl(basis_index1, basis_index1, 2, 2)
1274
1275            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1276            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1277            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1278            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1279
1280            self.HL_22 = np.array([[point_00, point_01],
1281                                   [point_10, point_11]])
1282
1283        def _build_hh_00(self):
1284            """Build the HH_00 matrix
1285
1286                HH_00 = [[H(0, 0) * H(0, 0), H(0, 0) * H(1, 0)],
1287                         [H(1, 0) * H(0, 0), H(1, 0) * H(1, 0)]]
1288
1289                Sets:
1290                    HH_00
1291
1292                Returns:
1293                    None
1294            """
1295
1296            basis_index0 = self.basis_index[0]
1297            basis_index1 = self.basis_index[1]
1298
1299            func_00 = self.func_hh(basis_index0, basis_index0, 0, 0)
1300            func_01 = self.func_hh(basis_index0, basis_index1, 0, 0)
1301            func_10 = self.func_hh(basis_index1, basis_index0, 0, 0)
1302            func_11 = self.func_hh(basis_index1, basis_index1, 0, 0)
1303
1304            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1305            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1306            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1307            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1308
1309            self.HH_00 = np.array([[point_00, point_01],
```

```
1310                                [point_10, point_11]])
1311
1312     def _build_hh_01(self):
1313         """Build the HH_01 matrix
1314
1315             HH_01 = [[H(0, 0) * H(0, 1), H(0, 0) * H(1, 1)],
1316                      [H(1, 0) * H(0, 1), H(1, 0) * H(1, 1)]]
1317
1318             Sets:
1319                 HH_01
1320
1321             Returns:
1322                 None
1323         """
1324
1325         basis_index0 = self.basis_index[0]
1326         basis_index1 = self.basis_index[1]
1327
1328         func_00 = self.func_hh(basis_index0, basis_index0, 0, 1)
1329         func_01 = self.func_hh(basis_index0, basis_index1, 0, 1)
1330         func_10 = self.func_hh(basis_index1, basis_index0, 0, 1)
1331         func_11 = self.func_hh(basis_index1, basis_index1, 0, 1)
1332
1333         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1334         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1335         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1336         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1337
1338         self.HH_01 = np.array([[point_00, point_01],
1339                                [point_10, point_11]])
1340
1341     def _build_hh_02(self):
1342         """Build the HH_02 matrix
1343
1344             HH_02 = [[H(0, 0) * H(0, 2), H(0, 0) * H(1, 2)],
1345                      [H(1, 0) * H(0, 2), H(1, 0) * H(1, 2)]]
1346
1347             Sets:
1348                 HH_02
1349
1350             Returns:
1351                 None
1352         """
1353
1354         basis_index0 = self.basis_index[0]
1355         basis_index1 = self.basis_index[1]
1356
1357         func_00 = self.func_hh(basis_index0, basis_index0, 0, 2)
1358         func_01 = self.func_hh(basis_index0, basis_index1, 0, 2)
1359         func_10 = self.func_hh(basis_index1, basis_index0, 0, 2)
1360         func_11 = self.func_hh(basis_index1, basis_index1, 0, 2)
1361
1362         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1363         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1364         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1365         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1366
1367         self.HH_02 = np.array([[point_00, point_01],
1368                                [point_10, point_11]])
1369
1370     def _build_hh_10(self):
1371         """Build the HH_10 matrix
1372
1373             HH_10 = [[H(0, 1) * H(0, 0), H(0, 1) * H(1, 0)],
```

```
1374                     [H(1, 1) * H(0, 0), H(1, 1) * H(1, 0)]]
1375
1376             Sets:
1377                 HH_10
1378
1379             Returns:
1380                 None
1381         """
1382
1383         basis_index0 = self.basis_index[0]
1384         basis_index1 = self.basis_index[1]
1385
1386         func_00 = self.func_hh(basis_index0, basis_index0, 1, 0)
1387         func_01 = self.func_hh(basis_index0, basis_index1, 1, 0)
1388         func_10 = self.func_hh(basis_index1, basis_index0, 1, 0)
1389         func_11 = self.func_hh(basis_index1, basis_index1, 1, 0)
1390
1391         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1392         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1393         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1394         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1395
1396         self.HH_10 = np.array([[point_00, point_01],
1397                                [point_10, point_11]])
1398
1399     def _build_hh_11(self):
1400         """Build the HH_11 matrix
1401
1402             HH_11 = [[H(0, 1) * H(0, 1), H(0, 1) * H(1, 1)],
1403                      [H(1, 1) * H(0, 1), H(1, 1) * H(1, 1)]]
1404
1405             Sets:
1406                 HH_11
1407
1408             Returns:
1409                 None
1410         """
1411
1412         basis_index0 = self.basis_index[0]
1413         basis_index1 = self.basis_index[1]
1414
1415         func_00 = self.func_hh(basis_index0, basis_index0, 1, 1)
1416         func_01 = self.func_hh(basis_index0, basis_index1, 1, 1)
1417         func_10 = self.func_hh(basis_index1, basis_index0, 1, 1)
1418         func_11 = self.func_hh(basis_index1, basis_index1, 1, 1)
1419
1420         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1421         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1422         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1423         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1424
1425         self.HH_11 = np.array([[point_00, point_01],
1426                                [point_10, point_11]])
1427
1428     def _build_hh_12(self):
1429         """Build the HH_12 matrix
1430
1431             HH_12 = [[H(0, 1) * H(0, 2), H(0, 1) * H(1, 2)],
1432                      [H(1, 1) * H(0, 2), H(1, 1) * H(1, 2)]]
1433
1434             Sets:
1435                 HH_12
1436
1437             Returns:
```

```
1438                 None
1439         """
1440
1441         basis_index0 = self.basis_index[0]
1442         basis_index1 = self.basis_index[1]
1443
1444         func_00 = self.func_hh(basis_index0, basis_index0, 1, 2)
1445         func_01 = self.func_hh(basis_index0, basis_index1, 1, 2)
1446         func_10 = self.func_hh(basis_index1, basis_index0, 1, 2)
1447         func_11 = self.func_hh(basis_index1, basis_index1, 1, 2)
1448
1449         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1450         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1451         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1452         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1453
1454         self.HH_12 = np.array([[point_00, point_01],
1455                                [point_10, point_11]])
1456
1457     def _build_hh_20(self):
1458         """Build the HH_20 matrix
1459
1460             HH_20 = [[H(0, 2) * H(0, 0), H(0, 2) * H(1, 0)],
1461                      [H(1, 2) * H(0, 0), H(1, 2) * H(1, 0)]]
1462
1463             Sets:
1464                 HH_20
1465
1466             Returns:
1467                 None
1468         """
1469
1470         basis_index0 = self.basis_index[0]
1471         basis_index1 = self.basis_index[1]
1472
1473         func_00 = self.func_hh(basis_index0, basis_index0, 2, 0)
1474         func_01 = self.func_hh(basis_index0, basis_index1, 2, 0)
1475         func_10 = self.func_hh(basis_index1, basis_index0, 2, 0)
1476         func_11 = self.func_hh(basis_index1, basis_index1, 2, 0)
1477
1478         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1479         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1480         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1481         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1482
1483         self.HH_20 = np.array([[point_00, point_01],
1484                                [point_10, point_11]])
1485
1486     def _build_hh_21(self):
1487         """Build the HH_22 matrix
1488
1489             HH_22 = [[H(0, 2) * H(0, 2), H(0, 2) * H(1, 2)],
1490                      [H(1, 2) * H(0, 2), H(1, 2) * H(1, 2)]]
1491
1492             Sets:
1493                 HH_22
1494
1495             Returns:
1496                 None
1497         """
1498
1499         basis_index0 = self.basis_index[0]
1500         basis_index1 = self.basis_index[1]
1501
```

```
1502            func_00 = self.func_hh(basis_index0, basis_index0, 2, 1)
1503            func_01 = self.func_hh(basis_index0, basis_index1, 2, 1)
1504            func_10 = self.func_hh(basis_index1, basis_index0, 2, 1)
1505            func_11 = self.func_hh(basis_index1, basis_index1, 2, 1)
1506
1507            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1508            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1509            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1510            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1511
1512            self.HH_21 = np.array([[point_00, point_01],
1513                                   [point_10, point_11]])
1514
1515        def _build_hh_22(self):
1516            """Build the HH_22 matrix
1517
1518                HH_22 = [[H(0, 2) * H(0, 2), H(0, 2) * H(1, 2)],
1519                         [H(1, 2) * H(0, 2), H(1, 2) * H(1, 2)]]
1520
1521                Sets:
1522                    HH_22
1523
1524                Returns:
1525                    None
1526            """
1527
1528            basis_index0 = self.basis_index[0]
1529            basis_index1 = self.basis_index[1]
1530
1531            func_00 = self.func_hh(basis_index0, basis_index0, 2, 2)
1532            func_01 = self.func_hh(basis_index0, basis_index1, 2, 2)
1533            func_10 = self.func_hh(basis_index1, basis_index0, 2, 2)
1534            func_11 = self.func_hh(basis_index1, basis_index1, 2, 2)
1535
1536            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1537            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1538            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1539            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1540
1541            self.HH_22 = np.array([[point_00, point_01],
1542                                   [point_10, point_11]])
1543
1544        def _build_lll_matrix(self, index_k, diff_i, diff_j, diff_k):
1545            """Builds the LLL matrix for a fixed k
1546
1547                LLL = [[Lt(0, 0), Lt(0, 1)],
1548                       [Lt(1, 0), Lt(1, 1)]]
1549
1550                Where,
1551
1552                Lt(i, j) = L(i, diff_i) * L(j, diff_j) * L(index_k, diff_k)
1553
1554                Args:
1555                    index_k := index on third basis function
1556                    diff_i  := diff on first basis
1557                    diff_j  := diff on second basis
1558                    diff_k  := diff on third basis
1559
1560                Returns:
1561                    matrix LLL
1562            """
1563
1564            index0 = self.basis_index[0]
1565            index1 = self.basis_index[1]
```

```
1566
1567            func_00 = self.func_lll(index0, index0, index_k,
1568                                    diff_i, diff_j, diff_k)
1569            func_01 = self.func_lll(index0, index1, index_k,
1570                                    diff_i, diff_j, diff_k)
1571            func_10 = self.func_lll(index1, index0, index_k,
1572                                    diff_i, diff_j, diff_k)
1573            func_11 = self.func_lll(index1, index1, index_k,
1574                                    diff_i, diff_j, diff_k)
1575
1576            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1577            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1578            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1579            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1580
1581            return np.array([[point_00, point_01],
1582                             [point_10, point_11]])
1583
1584        def _build_llh_matrix(self, index_k, diff_i, diff_j, diff_k):
1585            """Builds the LLH matrix for a fixed k
1586
1587                LLH = [[Lt(0, 0), Lt(0, 1)],
1588                       [Lt(1, 0), Lt(1, 1)]]
1589
1590                Where,
1591
1592                Lt(i, j) = L(i, diff_i) * L(j, diff_j) * H(index_k, diff_k)
1593
1594                Args:
1595                    index_k := index on third basis function
1596                    diff_i  := diff on first basis
1597                    diff_j  := diff on second basis
1598                    diff_k  := diff on third basis
1599
1600                Returns:
1601                    matrix LLH
1602            """
1603
1604            index0 = self.basis_index[0]
1605            index1 = self.basis_index[1]
1606
1607            func_00 = self.func_llh(index0, index0, index_k,
1608                                    diff_i, diff_j, diff_k)
1609            func_01 = self.func_llh(index0, index1, index_k,
1610                                    diff_i, diff_j, diff_k)
1611            func_10 = self.func_llh(index1, index0, index_k,
1612                                    diff_i, diff_j, diff_k)
1613            func_11 = self.func_llh(index1, index1, index_k,
1614                                    diff_i, diff_j, diff_k)
1615
1616            point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1617            point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1618            point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1619            point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1620
1621            return np.array([[point_00, point_01],
1622                             [point_10, point_11]])
1623
1624        def _build_lhl_matrix(self, index_k, diff_i, diff_j, diff_k):
1625            """Builds the LHL matrix for a fixed k
1626
1627                LHL = [[Lt(0, 0), Lt(0, 1)],
1628                       [Lt(1, 0), Lt(1, 1)]]
1629
```

```
1630            Where,
1631
1632            Lt(i, j) = L(i, diff_i) * H(j, diff_j) * L(index_k, diff_k)
1633
1634            Args:
1635                index_k := index on third basis function
1636                diff_i  := diff on first basis
1637                diff_j  := diff on second basis
1638                diff_k  := diff on third basis
1639
1640            Returns:
1641                matrix LHL
1642        """
1643
1644        index0 = self.basis_index[0]
1645        index1 = self.basis_index[1]
1646
1647        func_00 = self.func_lhl(index0, index0, index_k,
1648                                diff_i, diff_j, diff_k)
1649        func_01 = self.func_lhl(index0, index1, index_k,
1650                                diff_i, diff_j, diff_k)
1651        func_10 = self.func_lhl(index1, index0, index_k,
1652                                diff_i, diff_j, diff_k)
1653        func_11 = self.func_lhl(index1, index1, index_k,
1654                                diff_i, diff_j, diff_k)
1655
1656        point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1657        point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1658        point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1659        point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1660
1661        return np.array([[point_00, point_01],
1662                         [point_10, point_11]])
1663
1664    def _build_lhh_matrix(self, index_k, diff_i, diff_j, diff_k):
1665        """Builds the LHH matrix for a fixed k
1666
1667            LHH = [[Lt(0, 0), Lt(0, 1)],
1668                   [Lt(1, 0), Lt(1, 1)]]
1669
1670            Where,
1671
1672            Lt(i, j) = L(i, diff_i) * H(j, diff_j) * H(index_k, diff_k)
1673
1674            Args:
1675                index_k := index on third basis function
1676                diff_i  := diff on first basis
1677                diff_j  := diff on second basis
1678                diff_k  := diff on third basis
1679
1680            Returns:
1681                matrix LHH
1682        """
1683
1684        index0 = self.basis_index[0]
1685        index1 = self.basis_index[1]
1686
1687        func_00 = self.func_lhh(index0, index0, index_k,
1688                                diff_i, diff_j, diff_k)
1689        func_01 = self.func_lhh(index0, index1, index_k,
1690                                diff_i, diff_j, diff_k)
1691        func_10 = self.func_lhh(index1, index0, index_k,
1692                                diff_i, diff_j, diff_k)
1693        func_11 = self.func_lhh(index1, index1, index_k,
```

```
1694                                diff_i, diff_j, diff_k)
1695
1696        point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1697        point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1698        point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1699        point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1700
1701        return np.array([[point_00, point_01],
1702                         [point_10, point_11]])
1703
1704    def _build_hll_matrix(self, index_k, diff_i, diff_j, diff_k):
1705        """Builds the HLL matrix for a fixed k
1706
1707            HLL = [[Lt(0, 0), Lt(0, 1)],
1708                   [Lt(1, 0), Lt(1, 1)]]
1709
1710            Where,
1711
1712            Lt(i, j) = H(i, diff_i) * L(j, diff_j) * L(index_k, diff_k)
1713
1714            Args:
1715                index_k := index on third basis function
1716                diff_i  := diff on first basis
1717                diff_j  := diff on second basis
1718                diff_k  := diff on third basis
1719
1720            Returns:
1721                matrix HLL
1722        """
1723
1724        index0 = self.basis_index[0]
1725        index1 = self.basis_index[1]
1726
1727        func_00 = self.func_hll(index0, index0, index_k,
1728                                diff_i, diff_j, diff_k)
1729        func_01 = self.func_hll(index0, index1, index_k,
1730                                diff_i, diff_j, diff_k)
1731        func_10 = self.func_hll(index1, index0, index_k,
1732                                diff_i, diff_j, diff_k)
1733        func_11 = self.func_hll(index1, index1, index_k,
1734                                diff_i, diff_j, diff_k)
1735
1736        point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1737        point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1738        point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1739        point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1740
1741        return np.array([[point_00, point_01],
1742                         [point_10, point_11]])
1743
1744    def _build_hlh_matrix(self, index_k, diff_i, diff_j, diff_k):
1745        """Builds the HLH matrix for a fixed k
1746
1747            HLH = [[Lt(0, 0), Lt(0, 1)],
1748                   [Lt(1, 0), Lt(1, 1)]]
1749
1750            Where,
1751
1752            Lt(i, j) = H(i, diff_i) * L(j, diff_j) * H(index_k, diff_k)
1753
1754            Args:
1755                index_k := index on third basis function
1756                diff_i  := diff on first basis
1757                diff_j  := diff on second basis
```

```
1758                 diff_k  := diff on third basis
1759
1760             Returns:
1761                 matrix HLH
1762         """
1763
1764         index0 = self.basis_index[0]
1765         index1 = self.basis_index[1]
1766
1767         func_00 = self.func_hlh(index0, index0, index_k,
1768                                 diff_i, diff_j, diff_k)
1769         func_01 = self.func_hlh(index0, index1, index_k,
1770                                 diff_i, diff_j, diff_k)
1771         func_10 = self.func_hlh(index1, index0, index_k,
1772                                 diff_i, diff_j, diff_k)
1773         func_11 = self.func_hlh(index1, index1, index_k,
1774                                 diff_i, diff_j, diff_k)
1775
1776         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1777         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1778         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1779         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1780
1781         return np.array([[point_00, point_01],
1782                          [point_10, point_11]])
1783
1784     def _build_hhl_matrix(self, index_k, diff_i, diff_j, diff_k):
1785         """Builds the HHL matrix for a fixed k
1786
1787             HHL = [[Lt(0, 0), Lt(0, 1)],
1788                    [Lt(1, 0), Lt(1, 1)]]
1789
1790             Where,
1791
1792             Lt(i, j) = H(i, diff_i) * H(j, diff_j) * L(index_k, diff_k)
1793
1794             Args:
1795                 index_k := index on third basis function
1796                 diff_i  := diff on first basis
1797                 diff_j  := diff on second basis
1798                 diff_k  := diff on third basis
1799
1800             Returns:
1801                 matrix HHL
1802         """
1803
1804         index0 = self.basis_index[0]
1805         index1 = self.basis_index[1]
1806
1807         func_00 = self.func_hhl(index0, index0, index_k,
1808                                 diff_i, diff_j, diff_k)
1809         func_01 = self.func_hhl(index0, index1, index_k,
1810                                 diff_i, diff_j, diff_k)
1811         func_10 = self.func_hhl(index1, index0, index_k,
1812                                 diff_i, diff_j, diff_k)
1813         func_11 = self.func_hhl(index1, index1, index_k,
1814                                 diff_i, diff_j, diff_k)
1815
1816         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1817         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1818         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1819         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1820
1821         return np.array([[point_00, point_01],
```

```
1822                            [point_10, point_11]])
1823
1824     def _build_hhh_matrix(self, index_k, diff_i, diff_j, diff_k):
1825         """Builds the HHH matrix for a fixed k
1826
1827             HHH = [[Lt(0, 0), Lt(0, 1)],
1828                    [Lt(1, 0), Lt(1, 1)]]
1829
1830             Where,
1831
1832             Lt(i, j) = H(i, diff_i) * H(j, diff_j) * H(index_k, diff_k)
1833
1834             Args:
1835                 index_k := index on third basis function
1836                 diff_i  := diff on first basis
1837                 diff_j  := diff on second basis
1838                 diff_k  := diff on third basis
1839
1840             Returns:
1841                 matrix HHH
1842         """
1843
1844         index0 = self.basis_index[0]
1845         index1 = self.basis_index[1]
1846
1847         func_00 = self.func_hhh(index0, index0, index_k,
1848                                 diff_i, diff_j, diff_k)
1849         func_01 = self.func_hhh(index0, index1, index_k,
1850                                 diff_i, diff_j, diff_k)
1851         func_10 = self.func_hhh(index1, index0, index_k,
1852                                 diff_i, diff_j, diff_k)
1853         func_11 = self.func_hhh(index1, index1, index_k,
1854                                 diff_i, diff_j, diff_k)
1855
1856         point_00 = func_00.integral(self.lower_bound, self.upper_bound)
1857         point_01 = func_01.integral(self.lower_bound, self.upper_bound)
1858         point_10 = func_10.integral(self.lower_bound, self.upper_bound)
1859         point_11 = func_11.integral(self.lower_bound, self.upper_bound)
1860
1861         return np.array([[point_00, point_01],
1862                          [point_10, point_11]])
1863
1864     def _build_lll(self, diff_i, diff_j, diff_k):
1865         """Builds a LLL tensor
1866
1867             T_LLL = [LLL(0), LLL(1)]
1868
1869             Where,
1870
1871             LLL(i) = LLL(i, diff_i, diff_j, diff_k)
1872
1873             Args:
1874                 diff_i := diff on first basis
1875                 diff_j := diff on second basis
1876                 diff_k := diff on third basis
1877
1878             Returns:
1879                 tensor T_LLL
1880         """
1881
1882         index0 = self.basis_index[0]
1883         index1 = self.basis_index[1]
1884
1885         matrix0 = self._build_lll_matrix(index0, diff_i, diff_j, diff_k)
```

```
1886            matrix1 = self._build_lll_matrix(index1, diff_i, diff_j, diff_k)
1887
1888            return [matrix0, matrix1]
1889
1890        def _build_llh(self, diff_i, diff_j, diff_k):
1891            """Builds a LLH tensor
1892
1893                T_LLH = [LLH(0), LLH(1)]
1894
1895                Where,
1896
1897                LLH(i) = LLH(i, diff_i, diff_j, diff_k)
1898
1899                Args:
1900                    diff_i := diff on first basis
1901                    diff_j := diff on second basis
1902                    diff_k := diff on third basis
1903
1904                Returns:
1905                    tensor T_LLH
1906            """
1907
1908            index0 = self.basis_index[0]
1909            index1 = self.basis_index[1]
1910
1911            matrix0 = self._build_llh_matrix(index0, diff_i, diff_j, diff_k)
1912            matrix1 = self._build_llh_matrix(index1, diff_i, diff_j, diff_k)
1913
1914            return [matrix0, matrix1]
1915
1916        def _build_lhl(self, diff_i, diff_j, diff_k):
1917            """Builds a LHL tensor
1918
1919                T_LHL = [LHL(0), LHL(1)]
1920
1921                Where,
1922
1923                LHL(i) = LHL(i, diff_i, diff_j, diff_k)
1924
1925                Args:
1926                    diff_i := diff on first basis
1927                    diff_j := diff on second basis
1928                    diff_k := diff on third basis
1929
1930                Returns:
1931                    tensor T_LHL
1932            """
1933
1934            index0 = self.basis_index[0]
1935            index1 = self.basis_index[1]
1936
1937            matrix0 = self._build_lhl_matrix(index0, diff_i, diff_j, diff_k)
1938            matrix1 = self._build_lhl_matrix(index1, diff_i, diff_j, diff_k)
1939
1940            return [matrix0, matrix1]
1941
1942        def _build_lhh(self, diff_i, diff_j, diff_k):
1943            """Builds a LHH tensor
1944
1945                T_LHH = [LHH(0), LHH(1)]
1946
1947                Where,
1948
1949                LHH(i) = LHH(i, diff_i, diff_j, diff_k)
```

```
1950
1951                Args:
1952                    diff_i := diff on first basis
1953                    diff_j := diff on second basis
1954                    diff_k := diff on third basis
1955
1956                Returns:
1957                    tensor T_LHH
1958            """
1959
1960            index0 = self.basis_index[0]
1961            index1 = self.basis_index[1]
1962
1963            matrix0 = self._build_lhh_matrix(index0, diff_i, diff_j, diff_k)
1964            matrix1 = self._build_lhh_matrix(index1, diff_i, diff_j, diff_k)
1965
1966            return [matrix0, matrix1]
1967
1968        def _build_hll(self, diff_i, diff_j, diff_k):
1969            """Builds a HLL tensor
1970
1971                T_HLL = [HLL(0), HLL(1)]
1972
1973                Where,
1974
1975                HLL(i) = HLL(i, diff_i, diff_j, diff_k)
1976
1977                Args:
1978                    diff_i := diff on first basis
1979                    diff_j := diff on second basis
1980                    diff_k := diff on third basis
1981
1982                Returns:
1983                    tensor T_HLL
1984            """
1985
1986            index0 = self.basis_index[0]
1987            index1 = self.basis_index[1]
1988
1989            matrix0 = self._build_hll_matrix(index0, diff_i, diff_j, diff_k)
1990            matrix1 = self._build_hll_matrix(index1, diff_i, diff_j, diff_k)
1991
1992            return [matrix0, matrix1]
1993
1994        def _build_hlh(self, diff_i, diff_j, diff_k):
1995            """Builds a HLH tensor
1996
1997                T_HLH = [HLH(0), HLH(1)]
1998
1999                Where,
2000
2001                HLH(i) = HLH(i, diff_i, diff_j, diff_k)
2002
2003                Args:
2004                    diff_i := diff on first basis
2005                    diff_j := diff on second basis
2006                    diff_k := diff on third basis
2007
2008                Returns:
2009                    tensor T_HLH
2010            """
2011
2012            index0 = self.basis_index[0]
2013            index1 = self.basis_index[1]
```

```
2014
2015            matrix0 = self._build_hlh_matrix(index0, diff_i, diff_j, diff_k)
2016            matrix1 = self._build_hlh_matrix(index1, diff_i, diff_j, diff_k)
2017
2018            return [matrix0, matrix1]
2019
2020        def _build_hhl(self, diff_i, diff_j, diff_k):
2021            """Builds a HHL tensor
2022
2023                T_HHL = [HHL(0), HHL(1)]
2024
2025                Where,
2026
2027                HHL(i) = HHL(i, diff_i, diff_j, diff_k)
2028
2029                Args:
2030                    diff_i := diff on first basis
2031                    diff_j := diff on second basis
2032                    diff_k := diff on third basis
2033
2034                Returns:
2035                    tensor T_HHL
2036            """
2037
2038            index0 = self.basis_index[0]
2039            index1 = self.basis_index[1]
2040
2041            matrix0 = self._build_hhl_matrix(index0, diff_i, diff_j, diff_k)
2042            matrix1 = self._build_hhl_matrix(index1, diff_i, diff_j, diff_k)
2043
2044            return [matrix0, matrix1]
2045
2046        def _build_hhh(self, diff_i, diff_j, diff_k):
2047            """Builds a HHH tensor
2048
2049                T_HHH = [HHH(0), HHH(1)]
2050
2051                Where,
2052
2053                HHH(i) = HHH(i, diff_i, diff_j, diff_k)
2054
2055                Args:
2056                    diff_i := diff on first basis
2057                    diff_j := diff on second basis
2058                    diff_k := diff on third basis
2059
2060                Returns:
2061                    tensor T_HHH
2062            """
2063
2064            index0 = self.basis_index[0]
2065            index1 = self.basis_index[1]
2066
2067            matrix0 = self._build_hhh_matrix(index0, diff_i, diff_j, diff_k)
2068            matrix1 = self._build_hhh_matrix(index1, diff_i, diff_j, diff_k)
2069
2070            return [matrix0, matrix1]
2071
2072        def _build_lll_arrays(self):
2073            """Build and set all the LLL arrays
2074
2075                Sets:
2076                    LLL_110 = T_LLL(1, 1, 0)
2077                    LLL_200 = T_LLL(1, 0, 1)
```

```
2078            """
2079
2080            self.LLL_110 = self._build_lll(1, 1, 0)
2081            self.LLL_200 = self._build_lll(1, 0, 1)
2082
2083        def _build_llh_arrays(self):
2084            """Build and set all the LLH arrays
2085
2086                Sets:
2087                    LLH_110 = T_LLH(1, 1, 0)
2088                    LLH_200 = T_LLH(1, 0, 1)
2089            """
2090
2091            self.LLH_110 = self._build_llh(1, 1, 0)
2092            self.LLH_200 = self._build_llh(1, 0, 1)
2093
2094        def _build_lhl_arrays(self):
2095            """Build and set all the LHL arrays
2096
2097                Sets:
2098                    LHL_110 = T_LHL(1, 1, 0)
2099                    LHL_200 = T_LHL(1, 0, 1)
2100            """
2101
2102            self.LHL_110 = self._build_lhl(1, 1, 0)
2103            self.LHL_200 = self._build_lhl(1, 0, 1)
2104
2105        def _build_lhh_arrays(self):
2106            """Build and set all the LHH arrays
2107
2108                Sets:
2109                    LHH_110 = T_LHH(1, 1, 0)
2110                    LHH_200 = T_LHH(1, 0, 1)
2111            """
2112
2113            self.LHH_110 = self._build_lhh(1, 1, 0)
2114            self.LHH_200 = self._build_lhh(1, 0, 1)
2115
2116        def _build_hll_arrays(self):
2117            """Build and set all the HLL arrays
2118
2119                Sets:
2120                    HLL_110 = T_HLL(1, 1, 0)
2121                    HLL_200 = T_HLL(1, 0, 1)
2122            """
2123
2124            self.HLL_110 = self._build_hll(1, 1, 0)
2125            self.HLL_200 = self._build_hll(1, 0, 1)
2126
2127        def _build_hlh_arrays(self):
2128            """Build and set all the HLH arrays
2129
2130                Sets:
2131                    HLH_110 = T_HLH(1, 1, 0)
2132                    HLH_200 = T_HLH(1, 0, 1)
2133            """
2134
2135            self.HLH_110 = self._build_hlh(1, 1, 0)
2136            self.HLH_200 = self._build_hlh(1, 0, 1)
2137
2138        def _build_hhl_arrays(self):
2139            """Build and set all the HHL arrays
2140
2141                Sets:
```

```
2142                HHL_110 = T_HHL(1, 1, 0)
2143                HHL_200 = T_HHL(1, 0, 1)
2144            """
2145
2146            self.HHL_110 = self._build_hhl(1, 1, 0)
2147            self.HHL_200 = self._build_hhl(1, 0, 1)
2148
2149        def _build_hhh_arrays(self):
2150            """Build and set all the HHH arrays
2151
2152                Sets:
2153                    HHH_110 = T_HHH(1, 1, 0)
2154                    HHH_200 = T_HHH(1, 0, 1)
2155            """
2156
2157            self.HHH_110 = self._build_hhh(1, 1, 0)
2158            self.HHH_200 = self._build_hhh(1, 0, 1)
2159
2160        def _build_arrays(self):
2161            """Builds all the arrays
2162
2163                Runs:
2164                    _build_lll_arrays
2165                    _build_llh_arrays
2166                    _build_lhl_arrays
2167                    _build_lhh_arrays
2168                    _build_hll_arrays
2169                    _build_hlh_arrays
2170                    _build_hhl_arrays
2171                    _build_hhh_arrays
2172            """
2173
2174            self._build_lll_arrays()
2175            self._build_llh_arrays()
2176            self._build_lhl_arrays()
2177            self._build_lhh_arrays()
2178            self._build_hll_arrays()
2179            self._build_hlh_arrays()
2180            self._build_hhl_arrays()
2181            self._build_hhh_arrays()
2182
2183        def _build_ll(self):
2184            """Build all the LL matrices
2185
2186                Runs:
2187                    _build_ll_00
2188                    _build_ll_01
2189                    _build_ll_02
2190                    _build_ll_10
2191                    _build_ll_11
2192                    _build_ll_12
2193                    _build_ll_20
2194                    _build_ll_21
2195                    _build_ll_22
2196            """
2197
2198            self._build_ll_00()
2199            self._build_ll_01()
2200            self._build_ll_02()
2201            self._build_ll_10()
2202            self._build_ll_11()
2203            self._build_ll_12()
2204            self._build_ll_20()
2205            self._build_ll_21()
```

```
2206            self._build_ll_22()
2207
2208        def _build_lh(self):
2209            """Build all the LH matrices
2210
2211                Runs:
2212                    _build_lh_00
2213                    _build_lh_01
2214                    _build_lh_02
2215                    _build_lh_10
2216                    _build_lh_11
2217                    _build_lh_12
2218                    _build_lh_20
2219                    _build_lh_21
2220                    _build_lh_22
2221            """
2222
2223            self._build_lh_00()
2224            self._build_lh_01()
2225            self._build_lh_02()
2226            self._build_lh_10()
2227            self._build_lh_11()
2228            self._build_lh_12()
2229            self._build_lh_20()
2230            self._build_lh_21()
2231            self._build_lh_22()
2232
2233        def _build_hl(self):
2234            """Build all the HL matrices
2235
2236                Runs:
2237                    _build_hl_00
2238                    _build_hl_01
2239                    _build_hl_02
2240                    _build_hl_10
2241                    _build_hl_11
2242                    _build_hl_12
2243                    _build_hl_20
2244                    _build_hl_21
2245                    _build_hl_22
2246            """
2247
2248            self._build_hl_00()
2249            self._build_hl_01()
2250            self._build_hl_02()
2251            self._build_hl_10()
2252            self._build_hl_11()
2253            self._build_hl_12()
2254            self._build_hl_20()
2255            self._build_hl_21()
2256            self._build_hl_22()
2257
2258        def _build_hh(self):
2259            """Build all the HH matrices
2260
2261                Runs:
2262                    _build_hh_00
2263                    _build_hh_01
2264                    _build_hh_02
2265                    _build_hh_10
2266                    _build_hh_11
2267                    _build_hh_12
2268                    _build_hh_20
2269                    _build_hh_21
```

```
2270                _build_hh_22
2271            """
2272
2273            self._build_hh_00()
2274            self._build_hh_01()
2275            self._build_hh_02()
2276            self._build_hh_10()
2277            self._build_hh_11()
2278            self._build_hh_12()
2279            self._build_hh_20()
2280            self._build_hh_21()
2281            self._build_hh_22()
2282
2283        def _build(self):
2284            """Build and set all matrices and tensors
2285
2286                Runs:
2287                    _build_ll
2288                    _build_lh
2289                    _build_hl
2290                    _build_hh
2291                    _build_arrays
2292            """
2293
2294            self._build_ll()
2295            self._build_lh()
2296            self._build_hl()
2297            self._build_hh()
2298
2299            self._build_arrays()
2300
2301        def function_l_product(self, f, index, diff):
2302            """Get the function Lagrange basis product
2303
2304                f_L = f * L(index, diff)
2305
2306                Args:
2307                    f     := function for inner product
2308                    index := index of basis
2309                    diff  := derivative on basis
2310
2311                Returns:
2312                    f_L
2313            """
2314
2315            func = self.build_lagrange(index, diff)
2316
2317            return func * f
2318
2319        def function_h_product(self, f, index, diff):
2320            """Get the function Hermite basis product
2321
2322                f_H = f * H(index, diff)
2323
2324                Args:
2325                    f     := function for inner product
2326                    index := index of basis
2327                    diff  := derivative on basis
2328
2329                Returns:
2330                    f_H
2331            """
2332
2333            func = self.build_hermite(index, diff)
```

```
2334
2335            return func * f
2336
2337        def function_l_vector(self, f, diff=0):
2338            """Get the vector of function-Lagrange evaluations
2339
2340                F_L(i) = f_L(i, diff)
2341
2342                Where,
2343
2344                vec_L = [F_L(0), F_L(1)]
2345
2346                Args:
2347                    f    := function for inner product
2348                    diff := derivative on basis
2349
2350                Returns:
2351                    returns vec_L
2352            """
2353
2354            basis_index0 = self.basis_index[0]
2355            basis_index1 = self.basis_index[1]
2356
2357            func_0 = self.function_l_product(f, basis_index0, diff)
2358            func_1 = self.function_l_product(f, basis_index1, diff)
2359
2360            point_0 = func_0.integral(self.lower_bound, self.upper_bound)
2361            point_1 = func_1.integral(self.lower_bound, self.upper_bound)
2362
2363            return np.array([[point_0], [point_1]])
2364
2365        def function_h_vector(self, f, diff=0):
2366            """Get the vector of function-Hermite evaluations
2367
2368                F_H(i) = f_H(i, diff)
2369
2370                Where,
2371
2372                vec_H = [F_H(0), F_H(1)]
2373
2374                Args:
2375                    f    := function for inner product
2376                    diff := derivative on basis
2377
2378                Returns:
2379                    returns vec_H
2380            """
2381
2382            basis_index0 = self.basis_index[0]
2383            basis_index1 = self.basis_index[1]
2384
2385            func_0 = self.function_h_product(f, basis_index0, diff)
2386            func_1 = self.function_h_product(f, basis_index1, diff)
2387
2388            point_0 = func_0.integral(self.lower_bound, self.upper_bound)
2389            point_1 = func_1.integral(self.lower_bound, self.upper_bound)
2390
2391            return np.array([[point_0], [point_1]])
```

A.3 source/finite_elements.py

```
1  import numpy as np
```

```
2  import scipy.sparse as sp
3  import scipy.sparse.linalg as lin
4
5  from source import element
6
7
8  class FiniteElements(object):
9      """Class to contain finite elements system
10
11         Attributes:
12             _num_points := number of points
13             _delta_x    := distance between points
14
15             _elements := list of elements
16
17             A_00 := the 00 FEM matrix
18             A_01 := the 01 FEM matrix
19             A_02 := the 02 FEM matrix
20             A_10 := the 10 FEM matrix
21             A_11 := the 11 FEM matrix
22             A_12 := the 12 FEM matrix
23             A_20 := the 20 FEM matrix
24             A_21 := the 21 FEM matrix
25             A_22 := the 22 FEM matrix
26     """
27
28     def __init__(self, num_points, delta_x):
29         self._num_points = num_points
30         self._delta_x = delta_x
31
32     @classmethod
33     def setup(cls, num_points, delta_x):
34         """Setup finite_elements
35
36             Args:
37                 num_points := number of points
38                 delta_x    := distance between points
39
40             Returns:
41                 fully built finite elements
42         """
43
44         new_finite = cls(num_points, delta_x)
45         new_finite._build()
46
47         return new_finite
48
49     @property
50     def num_elements(self):
51         """The number of elements
52
53             Returns:
54                 number of elements
55         """
56
57         return self._num_points - 1
58
59     def build_element(self, index):
60         """Build an element
61
62             Args:
63                 index := index of element
64
65             Returns:
```

```
66                  the fully built element with index
67          """
68
69          return element.Element.setup(index, self._num_points, self._delta_x)
70
71      def _build_elements(self):
72          """Build all the elements
73
74              Runs:
75                  build for every element upto index num_elements
76          """
77
78          self._elements = []
79
80          for index in range(self.num_elements):
81              self._elements.append(self.build_element(index))
82
83      def _local_pl_2(self, p, index):
84          """Get the local PL array from element
85
86              Args:
87                  p      := function for PL
88                  index := index of element
89
90              Returns:
91                  PL_2 array for element index
92          """
93
94          return self._elements[index].pl_2(p)
95
96      def _local_ph_2(self, p, index):
97          """Get the local PH array from element
98
99              Args:
100                 p      := function for PH
101                 index := index of element
102
103             Returns:
104                 PH_2 array for element index
105         """
106
107         return self._elements[index].ph_2(p)
108
109     def _local_ll_00(self, index):
110         """Get the local LL_00 matrix from element index
111
112             Args:
113                 index := index of element
114
115             Returns:
116                 LL_00 for element of index
117         """
118
119         return self._elements[index].LL_00
120
121     def _local_ll_01(self, index):
122         """Get the local LL_01 matrix from element index
123
124             Args:
125                 index := index of element
126
127             Returns:
128                 LL_01 for element of index
129         """
```

```
130
131         return self._elements[index].LL_01
132
133     def _local_ll_02(self, index):
134         """Get the local LL_02 matrix from element index
135
136             Args:
137                 index := index of element
138
139             Returns:
140                 LL_02 for element of index
141         """
142
143         return self._elements[index].LL_02
144
145     def _local_ll_10(self, index):
146         """Get the local LL_10 matrix from element index
147
148             Args:
149                 index := index of element
150
151             Returns:
152                 LL_10 for element of index
153         """
154
155         return self._elements[index].LL_10
156
157     def _local_ll_11(self, index):
158         """Get the local LL_11 matrix from element index
159
160             Args:
161                 index := index of element
162
163             Returns:
164                 LL_11 for element of index
165         """
166
167         return self._elements[index].LL_11
168
169     def _local_ll_12(self, index):
170         """Get the local LL_12 matrix from element index
171
172             Args:
173                 index := index of element
174
175             Returns:
176                 LL_12 for element of index
177         """
178
179         return self._elements[index].LL_12
180
181     def _local_ll_20(self, index):
182         """Get the local LL_20 matrix from element index
183
184             Args:
185                 index := index of element
186
187             Returns:
188                 LL_20 for element of index
189         """
190
191         return self._elements[index].LL_20
192
193     def _local_ll_21(self, index):
```

```
194             """Get the local LL_21 matrix from element index
195
196                 Args:
197                     index := index of element
198
199                 Returns:
200                     LL_21 for element of index
201             """
202
203             return self._elements[index].LL_21
204
205         def _local_ll_22(self, index):
206             """Get the local LL_22 matrix from element index
207
208                 Args:
209                     index := index of element
210
211                 Returns:
212                     LL_22 for element of index
213             """
214
215             return self._elements[index].LL_22
216
217         def _local_lh_00(self, index):
218             """Get the local LH_00 matrix from element index
219
220                 Args:
221                     index := index of element
222
223                 Returns:
224                     LH_00 for element of index
225             """
226
227             return self._elements[index].LH_00
228
229         def _local_lh_01(self, index):
230             """Get the local LH_01 matrix from element index
231
232                 Args:
233                     index := index of element
234
235                 Returns:
236                     LH_01 for element of index
237             """
238
239             return self._elements[index].LH_01
240
241         def _local_lh_02(self, index):
242             """Get the local LH_02 matrix from element index
243
244                 Args:
245                     index := index of element
246
247                 Returns:
248                     LH_02 for element of index
249             """
250
251             return self._elements[index].LH_02
252
253         def _local_lh_10(self, index):
254             """Get the local LH_10 matrix from element index
255
256                 Args:
257                     index := index of element
```

```
258
259             Returns:
260                 LH_10 for element of index
261         """
262
263         return self._elements[index].LH_10
264
265     def _local_lh_11(self, index):
266         """Get the local LH_11 matrix from element index
267
268             Args:
269                 index := index of element
270
271             Returns:
272                 LH_11 for element of index
273         """
274
275         return self._elements[index].LH_11
276
277     def _local_lh_12(self, index):
278         """Get the local LH_12 matrix from element index
279
280             Args:
281                 index := index of element
282
283             Returns:
284                 LH_12 for element of index
285         """
286
287         return self._elements[index].LH_12
288
289     def _local_lh_20(self, index):
290         """Get the local LH_20 matrix from element index
291
292             Args:
293                 index := index of element
294
295             Returns:
296                 LH_20 for element of index
297         """
298
299         return self._elements[index].LH_20
300
301     def _local_lh_21(self, index):
302         """Get the local LH_21 matrix from element index
303
304             Args:
305                 index := index of element
306
307             Returns:
308                 LH_21 for element of index
309         """
310
311         return self._elements[index].LH_21
312
313     def _local_lh_22(self, index):
314         """Get the local LH_22 matrix from element index
315
316             Args:
317                 index := index of element
318
319             Returns:
320                 LH_22 for element of index
321         """
```

```
322
323         return self._elements[index].LH_22
324
325     def _local_hl_00(self, index):
326         """Get the local HL_00 matrix from element index
327
328             Args:
329                 index := index of element
330
331             Returns:
332                 HL_00 for element of index
333         """
334
335         return self._elements[index].HL_00
336
337     def _local_hl_01(self, index):
338         """Get the local HL_01 matrix from element index
339
340             Args:
341                 index := index of element
342
343             Returns:
344                 HL_01 for element of index
345         """
346
347         return self._elements[index].HL_01
348
349     def _local_hl_02(self, index):
350         """Get the local HL_02 matrix from element index
351
352             Args:
353                 index := index of element
354
355             Returns:
356                 HL_02 for element of index
357         """
358
359         return self._elements[index].HL_02
360
361     def _local_hl_10(self, index):
362         """Get the local HL_10 matrix from element index
363
364             Args:
365                 index := index of element
366
367             Returns:
368                 HL_10 for element of index
369         """
370
371         return self._elements[index].HL_10
372
373     def _local_hl_11(self, index):
374         """Get the local HL_11 matrix from element index
375
376             Args:
377                 index := index of element
378
379             Returns:
380                 HL_11 for element of index
381         """
382
383         return self._elements[index].HL_11
384
385     def _local_hl_12(self, index):
```

```
386             """Get the local HL_12 matrix from element index
387
388                 Args:
389                     index := index of element
390
391                 Returns:
392                     HL_12 for element of index
393             """
394
395             return self._elements[index].HL_12
396
397         def _local_hl_20(self, index):
398             """Get the local HL_20 matrix from element index
399
400                 Args:
401                     index := index of element
402
403                 Returns:
404                     HL_20 for element of index
405             """
406
407             return self._elements[index].HL_20
408
409         def _local_hl_21(self, index):
410             """Get the local HL_21 matrix from element index
411
412                 Args:
413                     index := index of element
414
415                 Returns:
416                     HL_21 for element of index
417             """
418
419             return self._elements[index].HL_21
420
421         def _local_hl_22(self, index):
422             """Get the local HL_22 matrix from element index
423
424                 Args:
425                     index := index of element
426
427                 Returns:
428                     HL_22 for element of index
429             """
430
431             return self._elements[index].HL_22
432
433         def _local_hh_00(self, index):
434             """Get the local HH_00 matrix from element index
435
436                 Args:
437                     index := index of element
438
439                 Returns:
440                     HH_00 for element of index
441             """
442
443             return self._elements[index].HH_00
444
445         def _local_hh_01(self, index):
446             """Get the local HH_01 matrix from element index
447
448                 Args:
449                     index := index of element
```

```
450
451             Returns:
452                 HH_01 for element of index
453         """
454
455         return self._elements[index].HH_01
456
457     def _local_hh_02(self, index):
458         """Get the local HH_02 matrix from element index
459
460             Args:
461                 index := index of element
462
463             Returns:
464                 HH_02 for element of index
465         """
466
467         return self._elements[index].HH_02
468
469     def _local_hh_10(self, index):
470         """Get the local HH_10 matrix from element index
471
472             Args:
473                 index := index of element
474
475             Returns:
476                 HH_10 for element of index
477         """
478
479         return self._elements[index].HH_10
480
481     def _local_hh_11(self, index):
482         """Get the local HH_11 matrix from element index
483
484             Args:
485                 index := index of element
486
487             Returns:
488                 HH_11 for element of index
489         """
490
491         return self._elements[index].HH_11
492
493     def _local_hh_12(self, index):
494         """Get the local HH_12 matrix from element index
495
496             Args:
497                 index := index of element
498
499             Returns:
500                 HH_12 for element of index
501         """
502
503         return self._elements[index].HH_12
504
505     def _local_hh_20(self, index):
506         """Get the local HH_20 matrix from element index
507
508             Args:
509                 index := index of element
510
511             Returns:
512                 HH_20 for element of index
513         """
```

```
514
515         return self._elements[index].HH_20
516
517     def _local_hh_21(self, index):
518         """Get the local HH_21 matrix from element index
519
520             Args:
521                 index := index of element
522
523             Returns:
524                 HH_21 for element of index
525         """
526
527         return self._elements[index].HH_21
528
529     def _local_hh_22(self, index):
530         """Get the local HH_22 matrix from element index
531
532             Args:
533                 index := index of element
534
535             Returns:
536                 HH_22 for element of index
537         """
538
539         return self._elements[index].HH_22
540
541     def _local_lll_110(self, index):
542         """Gets the local LLL_110 array from element of index
543
544             Args:
545                 index := index of element
546
547             Returns:
548                 LLL_110 for element of index
549         """
550
551         return self._elements[index].LLL_110
552
553     def _local_lll_200(self, index):
554         """Gets the local LLL_200 array from element of index
555
556             Args:
557                 index := index of element
558
559             Returns:
560                 LLL_200 for element of index
561         """
562
563         return self._elements[index].LLL_200
564
565     def _local_llh_110(self, index):
566         """Gets the local LLH_110 array from element of index
567
568             Args:
569                 index := index of element
570
571             Returns:
572                 LLH_110 for element of index
573         """
574
575         return self._elements[index].LLH_110
576
577     def _local_llh_200(self, index):
```

```
578             """Gets the local LLH_200 array from element of index
579
580                 Args:
581                     index := index of element
582
583                 Returns:
584                     LLH_200 for element of index
585             """
586
587             return self._elements[index].LLH_200
588
589         def _local_lhl_110(self, index):
590             """Gets the local LHL_110 array from element of index
591
592                 Args:
593                     index := index of element
594
595                 Returns:
596                     LHL_110 for element of index
597             """
598
599             return self._elements[index].LHL_110
600
601         def _local_lhl_200(self, index):
602             """Gets the local LHL_200 array from element of index
603
604                 Args:
605                     index := index of element
606
607                 Returns:
608                     LHL_200 for element of index
609             """
610
611             return self._elements[index].LHL_200
612
613         def _local_lhh_110(self, index):
614             """Gets the local LLL_110 array from element of index
615
616                 Args:
617                     index := index of element
618
619                 Returns:
620                     LLL_110 for element of index
621             """
622
623             return self._elements[index].LHH_110
624
625         def _local_lhh_200(self, index):
626             """Gets the local LLL_200 array from element of index
627
628                 Args:
629                     index := index of element
630
631                 Returns:
632                     LLL_200 for element of index
633             """
634
635             return self._elements[index].LHH_200
636
637         def _local_hll_110(self, index):
638             """Gets the local HLL_110 array from element of index
639
640                 Args:
641                     index := index of element
```

```
642
643             Returns:
644                 HLL_110 for element of index
645         """
646
647         return self._elements[index].HLL_110
648
649     def _local_hll_200(self, index):
650         """Gets the local HLL_200 array from element of index
651
652             Args:
653                 index := index of element
654
655             Returns:
656                 HLL_200 for element of index
657         """
658
659         return self._elements[index].HLL_200
660
661     def _local_hlh_110(self, index):
662         """Gets the local HLH_110 array from element of index
663
664             Args:
665                 index := index of element
666
667             Returns:
668                 HLH_110 for element of index
669         """
670
671         return self._elements[index].HLH_110
672
673     def _local_hlh_200(self, index):
674         """Gets the local HLH_200 array from element of index
675
676             Args:
677                 index := index of element
678
679             Returns:
680                 HLH_200 for element of index
681         """
682
683         return self._elements[index].HLH_200
684
685     def _local_hhl_110(self, index):
686         """Gets the local HHL_110 array from element of index
687
688             Args:
689                 index := index of element
690
691             Returns:
692                 HHL_110 for element of index
693         """
694
695         return self._elements[index].HHL_110
696
697     def _local_hhl_200(self, index):
698         """Gets the local HHL_200 array from element of index
699
700             Args:
701                 index := index of element
702
703             Returns:
704                 HHL_200 for element of index
705         """
```

```
706
707             return self._elements[index].HHL_200
708
709         def _local_hhh_110(self, index):
710             """Gets the local HHH_110 array from element of index
711
712                 Args:
713                     index := index of element
714
715                 Returns:
716                     HHH_110 for element of index
717             """
718
719             return self._elements[index].HHH_110
720
721         def _local_hhh_200(self, index):
722             """Gets the local HHH_200 array from element of index
723
724                 Args:
725                     index := index of element
726
727                 Returns:
728                     HHH_200 for element of index
729             """
730
731             return self._elements[index].HHH_200
732
733         def _local_function_l(self, f, index, diff=0):
734             """Get the local Lagrange projection of f
735
736                 Args:
737                     f     := function for projection
738                     index := index of elements
739                     diff  := derivative of basis function
740
741                 Returns:
742                     projection array
743             """
744
745             return self._elements[index].function_l_vector(f, diff=diff)
746
747         def _local_function_h(self, f, index, diff=0):
748             """Get the local Hermite projection of f
749
750                 Args:
751                     f     := function for projection
752                     index := index of elements
753                     diff  := derivative of basis function
754
755                 Returns:
756                     projection array
757             """
758
759             return self._elements[index].function_h_vector(f, diff=diff)
760
761         def _init_block(self):
762             """Create an initial block matrix
763
764                 Returns:
765                     empty sparse matrix
766             """
767
768             return sp.lil_matrix((self._num_points, self._num_points),
769                                  dtype=np.float64)
```

```
770
771     def _build_block(self, local_matrix):
772         """Build one of the block matrices using local_matrix
773
774             Args:
775                 local_matrix := function to construct local (to element) matrix
776
777             Returns:
778                 the block matrix
779         """
780
781         block = self._init_block()
782         for index in range(self.num_elements):
783             local_block = local_matrix(index)
784             block[index:(index + 2), index:(index + 2)] += local_block
785
786         return block
787
788     def _build_ll_00(self):
789         """Build the LL_00 block matrix
790
791             Returns:
792                 block LL_00 matrix
793         """
794
795         return self._build_block(self._local_ll_00)
796
797     def _build_ll_01(self):
798         """Build the LL_01 block matrix
799
800             Returns:
801                 block LL_01 matrix
802         """
803
804         return self._build_block(self._local_ll_01)
805
806     def _build_ll_02(self):
807         """Build the LL_02 block matrix
808
809             Returns:
810                 block LL_02 matrix
811         """
812
813         return self._build_block(self._local_ll_02)
814
815     def _build_ll_10(self):
816         """Build the LL_10 block matrix
817
818             Returns:
819                 block LL_10 matrix
820         """
821
822         return self._build_block(self._local_ll_10)
823
824     def _build_ll_11(self):
825         """Build the LL_11 block matrix
826
827             Returns:
828                 block LL_11 matrix
829         """
830
831         return self._build_block(self._local_ll_11)
832
833     def _build_ll_12(self):
```

```
834             """Build the LL_12 block matrix
835
836                 Returns:
837                     block LL_12 matrix
838             """
839
840             return self._build_block(self._local_ll_12)
841
842         def _build_ll_20(self):
843             """Build the LL_20 block matrix
844
845                 Returns:
846                     block LL_20 matrix
847             """
848
849             return self._build_block(self._local_ll_20)
850
851         def _build_ll_21(self):
852             """Build the LL_21 block matrix
853
854                 Returns:
855                     block LL_21 matrix
856             """
857
858             return self._build_block(self._local_ll_21)
859
860         def _build_ll_22(self):
861             """Build the LL_22 block matrix
862
863                 Returns:
864                     block LL_22 matrix
865             """
866
867             return self._build_block(self._local_ll_22)
868
869         def _build_lh_00(self):
870             """Build the LH_00 block matrix
871
872                 Returns:
873                     block LH_00 matrix
874             """
875
876             return self._build_block(self._local_lh_00)
877
878         def _build_lh_01(self):
879             """Build the LH_01 block matrix
880
881                 Returns:
882                     block LH_01 matrix
883             """
884
885             return self._build_block(self._local_lh_01)
886
887         def _build_lh_02(self):
888             """Build the LH_02 block matrix
889
890                 Returns:
891                     block LH_02 matrix
892             """
893
894             return self._build_block(self._local_lh_02)
895
896         def _build_lh_10(self):
897             """Build the LH_10 block matrix
```

```
898
899             Returns:
900                 block LH_10 matrix
901         """
902
903         return self._build_block(self._local_lh_10)
904
905     def _build_lh_11(self):
906         """Build the LH_11 block matrix
907
908             Returns:
909                 block LH_11 matrix
910         """
911
912         return self._build_block(self._local_lh_11)
913
914     def _build_lh_12(self):
915         """Build the LH_12 block matrix
916
917             Returns:
918                 block LH_12 matrix
919         """
920
921         return self._build_block(self._local_lh_12)
922
923     def _build_lh_20(self):
924         """Build the LH_20 block matrix
925
926             Returns:
927                 block LH_20 matrix
928         """
929
930         return self._build_block(self._local_lh_20)
931
932     def _build_lh_21(self):
933         """Build the LH_21 block matrix
934
935             Returns:
936                 block LH_21 matrix
937         """
938
939         return self._build_block(self._local_lh_21)
940
941     def _build_lh_22(self):
942         """Build the LH_22 block matrix
943
944             Returns:
945                 block LH_22 matrix
946         """
947
948         return self._build_block(self._local_lh_22)
949
950     def _build_hl_00(self):
951         """Build the HL_00 block matrix
952
953             Returns:
954                 block HL_00 matrix
955         """
956
957         return self._build_block(self._local_hl_00)
958
959     def _build_hl_01(self):
960         """Build the HL_01 block matrix
961
```

```
962             Returns:
963                 block HL_01 matrix
964         """
965
966         return self._build_block(self._local_hl_01)
967
968     def _build_hl_02(self):
969         """Build the HL_02 block matrix
970
971             Returns:
972                 block HL_02 matrix
973         """
974
975         return self._build_block(self._local_hl_02)
976
977     def _build_hl_10(self):
978         """Build the HL_10 block matrix
979
980             Returns:
981                 block HL_10 matrix
982         """
983
984         return self._build_block(self._local_hl_10)
985
986     def _build_hl_11(self):
987         """Build the HL_11 block matrix
988
989             Returns:
990                 block HL_11 matrix
991         """
992
993         return self._build_block(self._local_hl_11)
994
995     def _build_hl_12(self):
996         """Build the HL_12 block matrix
997
998             Returns:
999                 block HL_12 matrix
1000        """
1001
1002        return self._build_block(self._local_hl_12)
1003
1004    def _build_hl_20(self):
1005        """Build the HL_20 block matrix
1006
1007            Returns:
1008                block HL_20 matrix
1009        """
1010
1011        return self._build_block(self._local_hl_20)
1012
1013    def _build_hl_21(self):
1014        """Build the HL_21 block matrix
1015
1016            Returns:
1017                block HL_21 matrix
1018        """
1019
1020        return self._build_block(self._local_hl_21)
1021
1022    def _build_hl_22(self):
1023        """Build the HL_22 block matrix
1024
1025            Returns:
```

```
1026                 block HL_22 matrix
1027             """
1028
1029             return self._build_block(self._local_hl_22)
1030
1031         def _build_hh_00(self):
1032             """Build the HH_00 block matrix
1033
1034                 Returns:
1035                     block HH_00 matrix
1036             """
1037
1038             return self._build_block(self._local_hh_00)
1039
1040         def _build_hh_01(self):
1041             """Build the HH_01 block matrix
1042
1043                 Returns:
1044                     block HH_01 matrix
1045             """
1046
1047             return self._build_block(self._local_hh_01)
1048
1049         def _build_hh_02(self):
1050             """Build the HH_02 block matrix
1051
1052                 Returns:
1053                     block HH_02 matrix
1054             """
1055
1056             return self._build_block(self._local_hh_02)
1057
1058         def _build_hh_10(self):
1059             """Build the HH_10 block matrix
1060
1061                 Returns:
1062                     block HH_10 matrix
1063             """
1064
1065             return self._build_block(self._local_hh_10)
1066
1067         def _build_hh_11(self):
1068             """Build the HH_11 block matrix
1069
1070                 Returns:
1071                     block HH_11 matrix
1072             """
1073
1074             return self._build_block(self._local_hh_11)
1075
1076         def _build_hh_12(self):
1077             """Build the HH_12 block matrix
1078
1079                 Returns:
1080                     block HH_12 matrix
1081             """
1082
1083             return self._build_block(self._local_hh_12)
1084
1085         def _build_hh_20(self):
1086             """Build the HH_20 block matrix
1087
1088                 Returns:
1089                     block HH_20 matrix
```

```
1090            """
1091
1092            return self._build_block(self._local_hh_20)
1093
1094        def _build_hh_21(self):
1095            """Build the HH_21 block matrix
1096
1097                Returns:
1098                    block HH_21 matrix
1099            """
1100
1101            return self._build_block(self._local_hh_21)
1102
1103        def _build_hh_22(self):
1104            """Build the HH_22 block matrix
1105
1106                Returns:
1107                    block HH_22 matrix
1108            """
1109
1110            return self._build_block(self._local_hh_22)
1111
1112        def _init_array(self):
1113            """Builds initial block array
1114
1115                Returns:
1116                    empty sparse tensor array
1117            """
1118
1119            array_list = []
1120
1121            for _ in range(self._num_points):
1122                array_list.append(self._init_block())
1123
1124            return array_list
1125
1126        def _build_block_array(self, local_array):
1127            """Builds one of the block arrays
1128
1129                Args:
1130                    local_array := function to construct local (to element) array
1131
1132                Returns:
1133                    one of the block arrays
1134            """
1135
1136            block = self._init_array()
1137
1138            for index in range(self.num_elements):
1139                local_block_arrays = local_array(index)
1140                for l_index, l_block in enumerate(local_block_arrays):
1141                    block[index + l_index][index:(index + 2), index:(index + 2)] =\
1142                        l_block
1143
1144            return block
1145
1146        def _build_local_block_array_lll_110(self):
1147            """Builds the local LLL 110 block array
1148
1149                Returns:
1150                    block LLL_110 array
1151            """
1152
1153            return self._build_block_array(self._local_lll_110)
```

```
1154
1155        def _build_local_block_array_lll_200(self):
1156            """Builds the local LLL 200 block array
1157
1158                Returns:
1159                    block LLL_200 array
1160            """
1161
1162            return self._build_block_array(self._local_lll_200)
1163
1164        def _build_local_block_array_llh_110(self):
1165            """Builds the local LLH 110 block array
1166
1167                Returns:
1168                    block LLH_110 array
1169            """
1170
1171            return self._build_block_array(self._local_llh_110)
1172
1173        def _build_local_block_array_llh_200(self):
1174            """Builds the local LLH 200 block array
1175
1176                Returns:
1177                    block LLH_200 array
1178            """
1179
1180            return self._build_block_array(self._local_llh_200)
1181
1182        def _build_local_block_array_lhl_110(self):
1183            """Builds the local LHL 110 block array
1184
1185                Returns:
1186                    block LHL_110 array
1187            """
1188
1189            return self._build_block_array(self._local_lhl_110)
1190
1191        def _build_local_block_array_lhl_200(self):
1192            """Builds the local LHL 200 block array
1193
1194                Returns:
1195                    block LHL_200 array
1196            """
1197
1198            return self._build_block_array(self._local_lhl_200)
1199
1200        def _build_local_block_array_lhh_110(self):
1201            """Builds the local LHH 110 block array
1202
1203                Returns:
1204                    block LHH_110 array
1205            """
1206
1207            return self._build_block_array(self._local_lhh_110)
1208
1209        def _build_local_block_array_lhh_200(self):
1210            """Builds the local LHH 200 block array
1211
1212                Returns:
1213                    block LHH_200 array
1214            """
1215
1216            return self._build_block_array(self._local_lhh_200)
1217
```

```
1218        def _build_local_block_array_hll_110(self):
1219            """Builds the local HLL 110 block array
1220
1221                Returns:
1222                    block HLL_110 array
1223            """
1224
1225            return self._build_block_array(self._local_hll_110)
1226
1227        def _build_local_block_array_hll_200(self):
1228            """Builds the local HLL 200 block array
1229
1230                Returns:
1231                    block HLL_200 array
1232            """
1233
1234            return self._build_block_array(self._local_hll_200)
1235
1236        def _build_local_block_array_hlh_110(self):
1237            """Builds the local HLH 110 block array
1238
1239                Returns:
1240                    block HLH_110 array
1241            """
1242
1243            return self._build_block_array(self._local_hlh_110)
1244
1245        def _build_local_block_array_hlh_200(self):
1246            """Builds the local HLH 200 block array
1247
1248                Returns:
1249                    block HLH_200 array
1250            """
1251
1252            return self._build_block_array(self._local_hlh_200)
1253
1254        def _build_local_block_array_hhl_110(self):
1255            """Builds the local HHL 110 block array
1256
1257                Returns:
1258                    block HHL_110 array
1259            """
1260
1261            return self._build_block_array(self._local_hhl_110)
1262
1263        def _build_local_block_array_hhl_200(self):
1264            """Builds the local HHL 200 block array
1265
1266                Returns:
1267                    block HHL_200 array
1268            """
1269
1270            return self._build_block_array(self._local_hhl_200)
1271
1272        def _build_local_block_array_hhh_110(self):
1273            """Builds the local HHH 110 block array
1274
1275                Returns:
1276                    block HHH_110 array
1277            """
1278
1279            return self._build_block_array(self._local_hhh_110)
1280
1281        def _build_local_block_array_hhh_200(self):
```

```
1282            """Builds the local HHH 200 block array
1283
1284                Returns:
1285                    block HHH_200 array
1286            """
1287
1288            return self._build_block_array(self._local_hhh_200)
1289
1290        def _init_vec_block(self):
1291            """Create an initial vector block
1292
1293                Returns:
1294                    initial block vector
1295            """
1296
1297            return np.zeros((self._num_points, 1))
1298
1299        def _build_vec_block(self, f, local_vec, diff=0):
1300            """Build the vector block
1301
1302                Args:
1303                    f         := function for projection
1304                    local_vec := function to create the local vector
1305                    diff      := derivative of basis function
1306
1307                Returns:
1308                    the block vector
1309            """
1310
1311            block = self._init_vec_block()
1312            for index in range(self.num_elements):
1313                local_block = local_vec(f, index, diff=diff)
1314                block[index:(index + 2)] += local_block
1315
1316            return block
1317
1318        def _build_vec_l(self, f, diff=0):
1319            """Build the local vec Lagrange for f
1320
1321                Args:
1322                    f    := function for projection
1323                    diff := derivative of basis function
1324
1325                Returns:
1326                    the Lagrange block vector
1327            """
1328
1329            return self._build_vec_block(f, self._local_function_l, diff=diff)
1330
1331        def _build_vec_h(self, f, diff=0):
1332            """Build the local vec Hermite for f
1333
1334                Args:
1335                    f    := function for projection
1336                    diff := derivative of basis function
1337
1338                Returns:
1339                    the Hermite block vector
1340            """
1341
1342            return self._build_vec_block(f, self._local_function_h, diff=diff)
1343
1344        @staticmethod
1345        def _build_matrix(ll, lh, hl, hh):
```

```
1346            """Build the block matrix
1347
1348                Args:
1349                    ll := Lagrange-Lagrange matrix
1350                    lh := Lagrange-Hermite matrix
1351                    hl := Hermite-Lagrange matrix
1352                    hh := Hermite-Hermite matrix
1353
1354                Returns:
1355                    the assembled matrix of blocks
1356            """
1357
1358            return sp.bmat([[ll, lh], [hl, hh]])
1359
1360        def _build_00(self):
1361            """Build the A_00 matrix
1362
1363                Sets:
1364                    A_00 matrix
1365            """
1366
1367            ll = self._build_ll_00()
1368            lh = self._build_lh_00()
1369            hl = self._build_hl_00()
1370            hh = self._build_hh_00()
1371
1372            self.A_00 = self._build_matrix(ll, lh, hl, hh)
1373
1374        def _build_01(self):
1375            """Build the A_01 matrix
1376
1377                Sets:
1378                    A_01 matrix
1379            """
1380
1381            ll = self._build_ll_01()
1382            lh = self._build_lh_01()
1383            hl = self._build_hl_01()
1384            hh = self._build_hh_01()
1385
1386            self.A_01 = self._build_matrix(ll, lh, hl, hh)
1387
1388        def _build_02(self):
1389            """Build the A_02 matrix
1390
1391                Sets:
1392                    A_02 matrix
1393            """
1394
1395            ll = self._build_ll_02()
1396            lh = self._build_lh_02()
1397            hl = self._build_hl_02()
1398            hh = self._build_hh_02()
1399
1400            self.A_02 = self._build_matrix(ll, lh, hl, hh)
1401
1402        def _build_10(self):
1403            """Build the A_10 matrix
1404
1405                Sets:
1406                    A_10 matrix
1407            """
1408
1409            ll = self._build_ll_10()
```

```
1410            lh = self._build_lh_10()
1411            hl = self._build_hl_10()
1412            hh = self._build_hh_10()
1413
1414            self.A_10 = self._build_matrix(ll, lh, hl, hh)
1415
1416        def _build_11(self):
1417            """Build the A_11 matrix
1418
1419                Sets:
1420                    A_11 matrix
1421            """
1422
1423            ll = self._build_ll_11()
1424            lh = self._build_lh_11()
1425            hl = self._build_hl_11()
1426            hh = self._build_hh_11()
1427
1428            self.A_11 = self._build_matrix(ll, lh, hl, hh)
1429
1430        def _build_12(self):
1431            """Build the A_12 matrix
1432
1433                Sets:
1434                    A_12 matrix
1435            """
1436
1437            ll = self._build_ll_12()
1438            lh = self._build_lh_12()
1439            hl = self._build_hl_12()
1440            hh = self._build_hh_12()
1441
1442            self.A_12 = self._build_matrix(ll, lh, hl, hh)
1443
1444        def _build_20(self):
1445            """Build the A_20 matrix
1446
1447                Sets:
1448                    A_20 matrix
1449            """
1450
1451            ll = self._build_ll_20()
1452            lh = self._build_lh_20()
1453            hl = self._build_hl_20()
1454            hh = self._build_hh_20()
1455
1456            self.A_20 = self._build_matrix(ll, lh, hl, hh)
1457
1458        def _build_21(self):
1459            """Build the A_21 matrix
1460
1461                Sets:
1462                    A_21 matrix
1463            """
1464
1465            ll = self._build_ll_21()
1466            lh = self._build_lh_21()
1467            hl = self._build_hl_21()
1468            hh = self._build_hh_21()
1469
1470            self.A_21 = self._build_matrix(ll, lh, hl, hh)
1471
1472        def _build_22(self):
1473            """Build the A_22 matrix
```

```
1474
1475                Sets:
1476                    A_22 matrix
1477            """
1478
1479            ll = self._build_ll_22()
1480            lh = self._build_lh_22()
1481            hl = self._build_hl_22()
1482            hh = self._build_hh_22()
1483
1484            self.A_22 = self._build_matrix(ll, lh, hl, hh)
1485
1486        def _build_a_matrices(self):
1487            """Build the A matrices
1488
1489                Runs:
1490                    _build_00
1491                    _build_01
1492                    _build_02
1493                    _build_10
1494                    _build_11
1495                    _build_12
1496                    _build_20
1497                    _build_21
1498                    _build_22
1499            """
1500
1501            self._build_00()
1502            self._build_01()
1503            self._build_02()
1504            self._build_10()
1505            self._build_11()
1506            self._build_12()
1507            self._build_20()
1508            self._build_21()
1509            self._build_22()
1510
1511        def _block_assembler_array(self, ll, lh, hl, hh):
1512            """Assembles a array from blocks
1513
1514                Args:
1515                    ll := the ll block array
1516                    lh := the lh block array
1517                    hl := the hl block array
1518                    hh := the hh block array
1519
1520                Returns:
1521                    the full array
1522            """
1523
1524            array_list = []
1525
1526            for index in range(self._num_points):
1527                block = sp.bmat([[ll[index], lh[index]],
1528                                 [hl[index], hh[index]]]).tocsc()
1529                array_list.append(block)
1530
1531            return array_list
1532
1533        def _build_block_array_l_110(self):
1534            """Builds the big block array for Lagrange 110
1535
1536                Returns:
1537                    Lagrange A_110
```

```
1538            """
1539
1540            ll = self._build_local_block_array_lll_110()
1541            lh = self._build_local_block_array_lhl_110()
1542            hl = self._build_local_block_array_hll_110()
1543            hh = self._build_local_block_array_hhl_110()
1544
1545            return self._block_assembler_array(ll, lh, hl, hh)
1546
1547        def _build_block_array_l_200(self):
1548            """Builds the big block array for Lagrange 200
1549
1550                Returns:
1551                    Lagrange A_200
1552            """
1553
1554            ll = self._build_local_block_array_lll_200()
1555            lh = self._build_local_block_array_lhl_200()
1556            hl = self._build_local_block_array_hll_200()
1557            hh = self._build_local_block_array_hhl_200()
1558
1559            return self._block_assembler_array(ll, lh, hl, hh)
1560
1561        def _build_block_array_h_110(self):
1562            """Builds the big block array for Hermite 110
1563
1564                Returns:
1565                    Hermite A_110
1566            """
1567
1568            ll = self._build_local_block_array_llh_110()
1569            lh = self._build_local_block_array_lhh_110()
1570            hl = self._build_local_block_array_hlh_110()
1571            hh = self._build_local_block_array_hhh_110()
1572
1573            return self._block_assembler_array(ll, lh, hl, hh)
1574
1575        def _build_block_array_h_200(self):
1576            """Builds the big block array for Hermite 200
1577
1578                Returns:
1579                    Hermite A_200
1580            """
1581
1582            ll = self._build_local_block_array_llh_200()
1583            lh = self._build_local_block_array_lhh_200()
1584            hl = self._build_local_block_array_hlh_200()
1585            hh = self._build_local_block_array_hhh_200()
1586
1587            return self._block_assembler_array(ll, lh, hl, hh)
1588
1589        def _build_array_110(self):
1590            """Builds the collection of all 110 arrays
1591
1592                Sets:
1593                    A_110
1594            """
1595
1596            l_array = self._build_block_array_l_110()
1597            h_array = self._build_block_array_h_110()
1598
1599            self.A_110 = l_array + h_array
1600
1601        def _build_array_200(self):
```

```
1602            """Builds the collection of all 200 arrays
1603
1604                Sets:
1605                    A_200
1606            """
1607
1608            l_array = self._build_block_array_l_200()
1609            h_array = self._build_block_array_h_200()
1610
1611            self.A_200 = l_array + h_array
1612
1613        def build_vec(self, f, diff=0):
1614            """Build the vector for function f
1615
1616                Args:
1617                    f    := function for projection
1618                    diff := derivative of basis function
1619
1620                Returns:
1621                    the block vector
1622            """
1623
1624            l_vec = self._build_vec_l(f, diff=diff)
1625            h_vec = self._build_vec_h(f, diff=diff)
1626
1627            return np.concatenate((l_vec, h_vec))
1628
1629        def _build(self):
1630            """Build all sub components
1631
1632                Runs:
1633                    _build_elements
1634                    _build_a_matrices
1635                    _build_array_110
1636                    _build_array_200
1637            """
1638
1639            self._build_elements()
1640            self._build_a_matrices()
1641            self._build_array_110()
1642            self._build_array_200()
1643
1644        def l2_project(self, f, identify):
1645            """Find l2 projection of f
1646
1647                P = A_00^-1 * (f, b_j)_j=0^num_points-1
1648
1649                Args:
1650                    f        := function to project
1651                    identify := decide if we identify
1652
1653                Returns:
1654                    Projection of f
1655            """
1656
1657            b = self.build_vec(f)
1658            M = self.A_00.tocsc()
1659            Mtemp = M.copy()
1660
1661            if identify:
1662                for k in range(M.shape[0]):
1663                    Mtemp[k, 0] = 0.0
1664                    Mtemp[0, k] = 0.0
1665                    Mtemp[k, self._num_points] = 0.0
```

```
1666                 Mtemp[self._num_points, k] = 0.0
1667             Mtemp[0, 0] = 1.0
1668             Mtemp[self._num_points, self._num_points] = 1.0
1669 
1670         return lin.spsolve(Mtemp, b)
1671 
1672     def h1_project(self, f):
1673         """Find h1 projection of f
1674 
1675             P = (A_00 - A_01)^-1 * (f, b_j)_j=0^num_points-1
1676 
1677             Args:
1678                 f := function to project
1679 
1680             Returns:
1681                 Projection of f
1682         """
1683 
1684         b = self.build_vec(f)
1685         M = self.A_00.tocsc() - self.A_01.tocsc()
1686 
1687         return lin.spsolve(M, b)
1688 
1689     def h2_project(self, f):
1690         """Find h1 projection of f
1691 
1692             P = (A_00 - A_01 + A_02)^-1 * (f, b_j)_j=0^num_points-1
1693 
1694             Args:
1695             f := function to project
1696 
1697             Returns:
1698             Projection of f
1699         """
1700 
1701         b = self.build_vec(f)
1702         M = self.A_00.tocsc() - self.A_01.tocsc() + self.A_02.tocsc()
1703 
1704         return lin.spsolve(M, b)
1705 
1706     def l2_norm(self, u):
1707         """Find the l2 norm of vec
1708 
1709             Args:
1710                 u := function to take norm of
1711 
1712             Returns:
1713                 norm
1714         """
1715 
1716         v = self.A_00 * u
1717         value = np.dot(u, v)
1718 
1719         return value * 0.5
```

A.4 source/solver.py

```
1 class RK4(object):
2     """Class to contain a Runge-Kutta 4th Order method:
3 
4         Solves d/dt u = f(u)
5 
```

```
6           Attributes:
7               _delta_t := timestep length
8       """
9
10      def __init__(self, delta_t):
11          self._delta_t = delta_t
12
13          self.storagez = []
14          self.storagew = []
15
16      def make_storage(self):
17          """Create/reset a storage"""
18
19          self.storagez = []
20          self.storagew = []
21
22      def k1(self, f, t, u, usereverse=False):
23          """First stage of time stepper
24
25              Args:
26                  f := function to step
27                  t := time
28                  u := previous value
29
30                  usereverse := decide if reverse storage is used
31
32              Returns:
33                  K1 stage value
34          """
35
36          if usereverse:
37              output, storagez, storagew = f(t, u)
38
39              self.storagez.append(storagez)
40              self.storagew.append(storagew)
41
42              return output
43          else:
44              return f(t, u)
45
46      def k2(self, f, t, u, k1, usereverse=False):
47          """Second stage of time stepper
48
49              Args:
50                  f  := function to step
51                  t  := time
52                  u  := previous value
53                  k1 := k1 stage value
54
55                  usereverse := decide if reverse storage is used
56
57              Returns:
58                  K2 stage
59          """
60
61          h = self._delta_t / 2.0
62
63          v = u + h * k1
64          tt = t + h
65
66          if usereverse:
67              output, storagez, storagew = f(tt, v)
68
69              self.storagez.append(storagez)
```

```
70              self.storagew.append(storagew)
71
72              return output
73          else:
74              return f(tt, v)
75
76      def k3(self, f, t, u, k2, usereverse=False):
77          """Third stage of time stepper
78
79              Args:
80                  f  := function to step
81                  t  := time
82                  u  := previous value
83                  k2 := k2 stage value
84
85                  usereverse := decide if reverse storage is used
86
87              Returns:
88                  K3 stage
89          """
90
91          h = self._delta_t / 2.0
92
93          v = u + h * k2
94          tt = t + h
95
96          if usereverse:
97              output, storagez, storagew = f(tt, v)
98
99              self.storagez.append(storagez)
100             self.storagew.append(storagew)
101
102             return output
103         else:
104             return f(tt, v)
105
106     def k4(self, f, t, u, k3, usereverse=False):
107         """Fourth stage of time stepper
108
109             Args:
110                 f  := function to step
111                 t  := time
112                 u  := previous value
113                 k3 := k3 stage value
114
115                 usereverse := decide if reverse storage is used
116
117             Returns:
118                 K4 stage
119         """
120
121         v = u + self._delta_t * k3
122         tt = t + self._delta_t
123
124         if usereverse:
125             output, storagez, storagew = f(tt, v)
126
127             self.storagez.append(storagez)
128             self.storagew.append(storagew)
129
130             return output
131         else:
132             return f(tt, v)
133
```

```
134         @staticmethod
135         def combine(k1, k2, k3, k4):
136             """Combine the k1, k2, k3, k4
137 
138                 Args:
139                     k1 := k1 stage value
140                     k2 := k2 stage value
141                     k3 := k3 stage value
142                     k4 := k4 stage value
143 
144                 Returns:
145                     K combination
146             """
147 
148             return k1 + (k2 * 2.0) + (k3 * 2.0) + k4
149 
150         def new(self, u, k1, k2, k3, k4):
151             """Create new step from the 4 stages
152 
153                 Args:
154                     u  := current step value
155                     k1 := k1 stage value
156                     k2 := k2 stage value
157                     k3 := k3 stage value
158                     k4 := k4 stage value
159 
160                 Returns:
161                     New step value
162             """
163 
164             v = self.combine(k1, k2, k3, k4)
165 
166             return u + (self._delta_t / 6.0) * v
167 
168         def __call__(self, f, t, u, usereverse=False):
169             """Make stepping forward one step the call
170 
171                 Args:
172                     f  := function to step
173                     t  := time
174                     u  := previous value
175 
176                     usereverse := decide if reverse storage is used
177 
178                 Returns:
179                     new_step if usereverse is false
180                     new_step, z_values, w_values if usereverse is true
181             """
182 
183             if usereverse:
184                 self.make_storage()
185 
186             k1 = self.k1(f, t, u, usereverse)
187             k2 = self.k2(f, t, u, k1, usereverse)
188             k3 = self.k3(f, t, u, k2, usereverse)
189             k4 = self.k4(f, t, u, k3, usereverse)
190 
191             if usereverse:
192                 return self.new(u, k1, k2, k3, k4), self.storagez, self.storagew
193             else:
194                 return self.new(u, k1, k2, k3, k4)
```

A.5 source/initial.py

```
1  class InitV(object):
2      """Class to contain the initial function for v"""
3
4      def __call__(self, x):
5          """Run the function"""
6
7          return 5.0 * x**2
8          #return 0.01 * x**2
9
10
11 class InitW(object):
12     """Class to contain the initial function for w"""
13
14     def __call__(self, x):
15         """Run the function"""
16
17         return 5.0 * x**3
18         #return 0.01 * x**3
19         #return 5.0 * x**2
20
21
22 class InitZ0(object):
23     """Class to contain the initial function for z0"""
24
25     def __call__(self, x):
26         """Run the function"""
27
28         return x**3
29         #return 0.01 * x**2
30         #return 0.0
31
32
33 class InitZ1(object):
34     """Class to contain the initial function for z1"""
35
36     def __call__(self, x):
37         """Run the function"""
38
39         return x
40         #return 0.01 * x ** 2
41         #return 0.0
42
43
44 class G1(object):
45     """Class to contain the g1 control function"""
46
47     def __call__(self, x):
48         """Run the function"""
49
50         return x
51
52
53 class G2(object):
54     """Class to contain the g2 control function"""
55
56     def __call__(self, x):
57         """Run the function"""
58
59         return x
60
61
```

```
62  class M(object):
63      """Class to contain the m control function"""
64
65      def __call__(self, x):
66          """Run the function"""
67
68          return x
```

A.6 source/feedback.py

```
1   import numpy as np
2   import scipy.sparse as sp
3   import scipy.sparse.linalg as lin
4
5   from source import finite_elements as fem
6   from source import solver
7
8
9   class OnesFunction(object):
10      """Class to contain the ones-function system"""
11
12      def __call__(self, x):
13          """Return the value of the function"""
14
15          return 1.0
16
17
18  class SquareFunction(object):
19      """Class to contain the square-function system"""
20
21      def __init__(self, diff):
22          self._diff = diff
23
24      @staticmethod
25      def diff_0(x):
26          """Square function with 0 derivatives"""
27
28          return x**2 / 2.0
29
30      @staticmethod
31      def diff_1(x):
32          """Square function with 1 derivatives"""
33
34          return x
35
36      @staticmethod
37      def diff_2(x):
38          """Square function with 2 derivatives"""
39
40          return 1.0
41
42      def __call__(self, x):
43          """Call the correct function"""
44
45          if self._diff == 0:
46              return self.diff_0(x)
47          elif self._diff == 1:
48              return self.diff_1(x)
49          elif self._diff == 2:
50              return self.diff_2(x)
51
52
```

```
53  class CubeFunction(object):
54      """Class to contain the cube-function system"""
55
56      def __init__(self, diff):
57          self._diff = diff
58
59      @staticmethod
60      def diff_0(x):
61          """Cube function with 0 derivatives"""
62
63          return x**3 / 6.0 - x**2 / 2.0
64
65      @staticmethod
66      def diff_1(x):
67          """Cube function with 1 derivatives"""
68
69          return x**2 / 2.0 - x
70
71      @staticmethod
72      def diff_2(x):
73          """Cube function with 2 derivatives"""
74
75          return x - 1.0
76
77      def __call__(self, x):
78          """Call the correct function"""
79
80          if self._diff == 0:
81              return self.diff_0(x)
82          elif self._diff == 1:
83              return self.diff_1(x)
84          elif self._diff == 2:
85              return self.diff_2(x)
86
87
88  class FeedbackControl(object):
89      """Class to contain the feedback control system
90
91          Attributes:
92              _num_points := the number of space points (includes endpoints)
93              _delta_x    := distance between space points
94              _num_steps  := number of time steps to take
95              _delta_t    := amount of time to take in one timestep
96
97              _gamma := constant for w-system
98              _rho   := constant for w-system
99
100             _init_v  := initial condition for v
101             _init_w  := initial condition for w
102             _init_z0 := initial condition for z0
103             _init_z1 := initial condition for z1
104
105             _g1 := feedback function for z-system
106             _g2 := feedback function for w-system
107             _m  := feedback function for w-system
108
109             _use_feedback := turn on and off feedback
110             _use_coupling := turn on and off coupling
111             _use_reverse  := turn on and off reverse system
112
113             _epsilon   := feedback target energy is less than this
114             _max_steps := max_steps to attempt to achieve target
115
116             _storage_w := reverse control pre computed values for w
```

```
117             _storage_z := reverse control pre computed values for z
118
119             _is_forward := Determine reverse state
120         """
121
122         def __init__(self, num_points, delta_x, num_steps, delta_t,
123                      gamma, rho,
124                      init_v, init_w, init_z0, init_z1,
125                      g1, g2, m,
126                      use_feedback, use_coupling, use_reverse,
127                      epsilon, max_steps,
128                      storage_w, storage_z):
129             self._num_points = num_points
130             self._delta_x = delta_x
131             self._num_steps = num_steps
132             self._delta_t = delta_t
133
134             self._gamma = gamma
135             self._rho = rho
136
137             self._init_v = init_v
138             self._init_w = init_w
139             self._init_z0 = init_z0
140             self._init_z1 = init_z1
141
142             self._g1 = g1
143             self._g2 = g2
144             self._m = m
145
146             self._use_feedback = use_feedback
147             self._use_coupling = use_coupling
148             self._use_reverse = use_reverse
149
150             self._epsilon = epsilon
151             self._max_steps = max_steps
152
153             self._storage_w = storage_w
154             self._storage_z = storage_z
155
156             if storage_w is None:
157                 self._is_forward = True
158             else:
159                 self._is_forward = False
160
161         @classmethod
162         def setup(cls, num_points, delta_x, num_steps, delta_t,
163                   gamma, rho,
164                   init_v, init_w, init_z0, init_z1,
165                   g1, g2, m,
166                   use_feedback, use_coupling, use_reverse,
167                   epsilon, max_steps,
168                   storage_w, storage_z):
169             """Setup the system completely"""
170
171             new = cls(num_points, delta_x, num_steps, delta_t,
172                       gamma, rho,
173                       init_v, init_w, init_z0, init_z1,
174                       g1, g2, m,
175                       use_feedback, use_coupling, use_reverse,
176                       epsilon, max_steps,
177                       storage_w, storage_z)
178             new._build()
179
180             return new
```

```
181
182     @property
183     def vec_num_points(self):
184         """Get the number of points in a vec"""
185
186         return 2 * self._num_points
187
188     def _build(self):
189         """Build the feedback control system"""
190
191         print("Build the solver system")
192         self._build_solver()
193
194         print("Build the FEM system")
195         self._build_fem()
196
197         print("Build the w-system")
198         self._build_a00_w()
199         self._build_a01_w()
200         self._build_a02_w()
201         self._build_a11_w()
202         self._build_a22_w()
203         self._build_ml_w()
204         self._build_m_w()
205         self._build_a_w()
206         self._build_e_w()
207
208         print("Build the z-system")
209         self._build_a10_z()
210         self._build_a_z()
211         self._build_a00_z()
212         self._build_m_z()
213         self._build_a00_ez()
214         self._build_m_ez()
215
216         print("Build the init_w")
217         self._build_init_w0()
218         self._build_init_w1()
219         self._build_init_w()
220
221         print("Build the init_z")
222         self._build_init_z0()
223         self._build_init_z1()
224         self._build_init_z()
225
226         print("Build the feedback_w")
227         self._build_n_square_w()
228         self._build_n_cube_w()
229         self._build_f_plus_w()
230         self._build_n_w()
231
232         print("Build the feedback_z")
233         self._build_n_z()
234         self._build_f_plus_z()
235
236         print("Build the coupling_z")
237         self._build_a110_z()
238
239         print("Build the coupling_w")
240         self._build_a200_w()
241
242     def _build_solver(self):
243         """Build the time-solver"""
244
```

```
245             self._solver = solver.RK4(self._delta_t)
246
247         def _build_fem(self):
248             """Build the fem system"""
249
250             self._fem = fem.FiniteElements.setup(self._num_points, self._delta_x)
251
252         def split_eqn(self, u):
253             """Splits the equation u into it's two parts
254
255                 Args:
256                     u := equation to split
257
258                 Returns:
259                     the two parts
260             """
261
262             u0 = u[:self.vec_num_points]
263             u1 = u[self.vec_num_points:]
264
265             return u0, u1
266
267         def split_solution(self, u):
268             """Splits the solution u into the w and z systems
269
270                 Args:
271                     u := equation to split
272
273                 Returns:
274                     the two parts
275             """
276
277             z = u[:2 * self.vec_num_points]
278             w = u[2 * self.vec_num_points:]
279
280             return z, w
281
282         @staticmethod
283         def identify_matrix(matrix, index):
284             """Identify row and column of matrix corresponding to index
285
286                 Args:
287                     matrix := matrix to identify
288                     index  := index of row/column to identify
289             """
290
291             matrix[:, index] = 0.0
292             matrix[index, :] = 0.0
293             matrix[index, index] = 1.0
294
295         def _build_a00_w(self):
296             """Build the A_00 matrix for w-system
297
298                 A_00 -> identified for B.C.s
299             """
300
301             A = self._fem.A_00.copy().tolil()
302
303             self.identify_matrix(A, 0)
304             self.identify_matrix(A, self._num_points)
305
306             self.A_00_w = A.tocsc()
307
308         def _build_a01_w(self):
```

```
309             """Builds the A_01 matrix for the w-system
310
311                 A_01 -> identified for B.C.s
312             """
313
314             A = self._fem.A_01.copy().tolil()
315
316             self.identify_matrix(A, 0)
317             self.identify_matrix(A, self._num_points)
318
319             self.A_01_w = A.tocsc()
320
321         def _build_a02_w(self):
322             """Builds the A_02 matrix for the w-system
323
324                 A_02 -> identified for B.C.s
325             """
326
327             A = self._fem.A_02.copy().tolil()
328
329             self.identify_matrix(A, 0)
330             self.identify_matrix(A, self._num_points)
331
332             self.A_02_w = A.tocsc()
333
334         def _build_a11_w(self):
335             """Builds the A_11 (Laplacian) matrix for the w-system
336
337                 A_11 -> identified for B.C.s
338             """
339
340             A = self._fem.A_11.copy().tolil()
341
342             self.identify_matrix(A, 0)
343             self.identify_matrix(A, self._num_points)
344
345             A[0, 0] = 0.0
346             A[self._num_points, self._num_points] = 0.0
347
348             self.A_11_w = A.tocsc()
349
350         def _build_a22_w(self):
351             """Builds the A_22 (Biharmonic) matrix for the w-system
352
353                 A_22 -> identified for B.C.s
354             """
355
356             A = self._fem.A_22.copy().tolil()
357
358             self.identify_matrix(A, 0)
359             self.identify_matrix(A, self._num_points)
360
361             self.A_22_w = A.tocsc()
362
363         def _build_ml_w(self):
364             """Builds the mass/lapacian operator matrix for the w-system
365
366                 ML_w = A_00_w + (rho * A_11_w)
367             """
368
369             M = self.A_00_w.copy()
370             L = self.A_11_w.copy()
371
372             self.ML_w = (M + (self._rho * L)).tocsc()
```

```
373
374         def _build_m_w(self):
375             """Builds the mass matrix for the w-system
376
377                 M_w = [[ML_w,  None],
378                        [None,  Id   ]]
379             """
380
381             top = self.ML_w.copy()
382             bottom = sp.identity(self.vec_num_points).tocsc()
383
384             self.M_w = sp.bmat([[top,   None],
385                                 [None,  bottom]]).tocsc()
386
387         def _build_a_w(self):
388             """Builds the A matrix for the w-system
389
390                 A_w = [[None, -gamma^2 * A_22_w],
391                        [  Id,  None]]
392             """
393
394             B = self.A_22_w.copy()
395
396             top = -self._gamma**2 * B
397             bottom = sp.identity(self.vec_num_points)
398
399             self.A_w = sp.bmat([[None,    top],
400                                 [bottom,  None]]).tocsc()
401
402         def _build_e_w(self):
403             """Builds the energy matrix for w-system
404
405                 E_w = [[ML_w,  None],
406                        [None,  gamma^2 * A_22_w]]
407             """
408
409             B = self.A_22_w.copy()
410
411             top = self.ML_w.copy()
412             bottom = self._gamma**2 * B
413
414             self.E_w = sp.bmat([[top,   None],
415                                 [None,  bottom]]).tocsc()
416
417         def _build_a10_z(self):
418             """Build the A_10 matrix for the z-system
419
420                 A_10 -> identified for B.C.s
421             """
422
423             A = self._fem.A_10.copy().tolil()
424
425             self.identify_matrix(A, 0)
426             self.identify_matrix(A, self._num_points)
427
428             self.A_10_z = A.tocsc()
429
430         def _build_a_z(self):
431             """Build the A matrix for the z-system
432
433                 A_z = [[       0,  A_10_z.transpose()],
434                        [-A_10_z,  0                   ]]
435             """
436
```

```
437             A = self.A_10_z.copy()
438
439             top = A.copy().transpose()
440             bottom = -A.copy()
441
442             self.A_z = sp.bmat([[None,    top],
443                                 [bottom, None]]).tocsc()
444
445         def _build_a00_z(self):
446             """Build the A_00 matrix for z-system
447
448                 A_00 -> identified for B.C.s
449             """
450
451             A = self._fem.A_00.copy().tolil()
452
453             self.identify_matrix(A, 0)
454             self.identify_matrix(A, self._num_points)
455
456             self.A_00_z = A.tocsc()
457
458         def _build_m_z(self):
459             """Build the 'mass' matrix for z-system
460
461                 M_z = [[A_00_z,       0],
462                        [     0, A_00_z]]
463             """
464
465             A = self.A_00_z.copy()
466
467             self.M_z = sp.bmat([[A,     None],
468                                 [None, A]]).tocsc()
469
470         def _build_a00_ez(self):
471             """Build the A_00 matrix for z-system energy
472
473                 A_00 -> identified for B.C.s
474             """
475
476             A = self._fem.A_00.copy().tolil()
477
478             self.A_00_ez = A.tocsc()
479
480         def _build_m_ez(self):
481             """Build the 'mass' matrix for z-system energy
482
483                 M_ez = [[A_00_ez,       0],
484                        [     0, A_00_ez]]
485             """
486
487             A = self.A_00_ez.copy()
488
489             self.M_ez = sp.bmat([[A,     None],
490                                 [None, A]]).tocsc()
491
492         def _build_init_w0(self):
493             """Build initial vector v"""
494
495             if isinstance(self._init_v, np.ndarray):
496                 self.init_w0 = self._init_v
497             else:
498                 vec = self._fem.build_vec(self._init_v)
499
500                 vec[0] = 0.0
```

```
501             vec[self._num_points] = 0.0
502
503             M = self.ML_w.copy()
504
505             self.init_w0 = lin.spsolve(M, vec)
506
507     def _build_init_w1(self):
508         """Build initial vector w"""
509
510         if isinstance(self._init_w, np.ndarray):
511             self.init_w1 = self._init_w
512         else:
513             vec = self._fem.build_vec(self._init_w)
514
515             vec[0] = 0.0
516             vec[self._num_points] = 0.0
517
518             M = self.A_22_w.copy()
519
520             self.init_w1 = lin.spsolve(M, vec)
521
522     def _build_init_w(self):
523         """Build the initial vector for w-system"""
524
525         vec0 = self.init_w0
526         vec1 = self.init_w1
527
528         self.init_w = np.concatenate((vec0, vec1))
529
530     def _build_init_z0(self):
531         """Build initial vector z0"""
532
533         if isinstance(self._init_z0, np.ndarray):
534             self.init_z0 = self._init_z0
535         else:
536             self.init_z0 = self._fem.build_vec(self._init_z0)
537
538     def _build_init_z1(self):
539         """Build initial vector z1"""
540
541         if isinstance(self._init_z1, np.ndarray):
542             self.init_z1 = self._init_z1
543         else:
544             self.init_z1 = self._fem.build_vec(self._init_z1)
545
546     def _build_init_z(self):
547         """Build initial vector for z-system"""
548
549         vec0 = self.init_z0
550         vec1 = self.init_z1
551
552         vec = np.concatenate((vec0, vec1))
553
554         if isinstance(self._init_z0, np.ndarray):
555             self.init_z = vec
556         else:
557             M = self.M_z.copy()
558             vec = lin.spsolve(M, vec)
559
560             vec[0] = 0.0
561             vec[self._num_points] = 0.0
562             vec[2 * self._num_points] = 0.0
563             vec[3 * self._num_points] = 0.0
564
```

```
565             self.init_z = vec
566
567     def _build_n_square_w(self):
568         """Builds the N_w vector corresponding to the square function"""
569
570         f0 = SquareFunction(0)
571         f1 = SquareFunction(1)
572         f2 = SquareFunction(2)
573
574         vec0 = self._fem.build_vec(f0, diff=0)
575         vec1 = self._fem.build_vec(f1, diff=1)
576         vec2 = self._fem.build_vec(f2, diff=2)
577
578         self.N_square_w = vec0 + vec1 + vec2
579
580     def _build_n_cube_w(self):
581         """Builds the N_w vector corresponding to the cube function"""
582
583         f0 = CubeFunction(0)
584         f1 = CubeFunction(1)
585         f2 = CubeFunction(2)
586
587         vec0 = self._fem.build_vec(f0, diff=0)
588         vec1 = self._fem.build_vec(f1, diff=1)
589         vec2 = self._fem.build_vec(f2, diff=2)
590
591         self.N_cube_w = vec0 + vec1 + vec2
592
593     def _build_f_plus_w(self):
594         """Builds the F_plus vector for w-system"""
595
596         vec = np.zeros((2, self.vec_num_points))
597
598         vec[0, -1] = 1.0
599         vec[1, self._num_points - 1] = 1.0
600
601         self.F_plus_w = vec
602
603     def _build_n_w(self):
604         """Builds the N_w matrix"""
605
606         F_plus = self.F_plus_w.copy()
607         B = self.A_22_w.copy()
608
609         Bt = B.transpose().tocsc()
610         Bt_inv = lin.inv(Bt)
611         Nt = F_plus * Bt_inv
612
613         self.N_w = Nt.transpose()
614
615     def g_w(self, w):
616         """Builds the G_w function for the w-system control
617
618             Args:
619                 w := current w-system vec
620
621             Returns:
622                 g_w(w) result vector
623         """
624
625         F_plus = self.F_plus_w
626
627         prod = np.matmul(F_plus, np.reshape(w, (self.vec_num_points, 1)))
628
```

```
629             m = self._m(prod[0])
630             g = self._g2(prod[1])
631 
632             return np.array([-m, -g])
633 
634         def feedback_w(self, w):
635             """Feedback control term for w-system
636 
637                 Args:
638                     w := current value of system
639 
640                 Returns:
641                     control term for w-system
642             """
643 
644             N = self.N_w
645 
646             v = w[:self.vec_num_points]
647 
648             g_w = self.g_w(v)
649 
650             left = np.zeros((self.vec_num_points, 1))
651             right = np.matmul(N, g_w)
652 
653             value = np.concatenate((left, right))
654 
655             return np.reshape(value, (2 * self.vec_num_points, ))
656 
657         def control_w_forward(self, t, w):
658             """Determines the forward control term for w
659 
660                 Args:
661                     t := current timestep value
662                     w := current value of system
663 
664                 Returns:
665                     w modified by the control term
666             """
667 
668             if self._use_feedback:
669                 feedback = self.feedback_w(w)
670 
671                 return w - feedback, feedback
672             else:
673                 feedback = [0.0, 0.0, 0.0, 0.0]
674 
675                 return w, feedback
676 
677         def control_w_reverse(self, t, w):
678             """Determines the reverse control term for w
679 
680                 Args:
681                     t := current timestep value
682                     w := current value of system
683 
684                 Returns:
685                     w modified by the control term
686             """
687 
688             feedback = self._storage_w.pop()
689 
690             return w + feedback, feedback
691 
692         def control_w(self, t, w):
```

```
693             """Determines the control term for w
694
695                 Args:
696                     t := current timestep value
697                     w := current value of system
698
699                 Returns:
700                     w modified by the control term
701             """
702
703             if self._is_forward:
704                 return self.control_w_forward(t, w)
705             else:
706                 return self.control_w_reverse(t, w)
707
708         def _build_n_z(self):
709             """Builds the N_z vector
710
711                 l2 projection of ones-function
712             """
713
714             f = OnesFunction()
715
716             self.N_z = self._fem.l2_project(f, identify=False)
717
718         def _build_f_plus_z(self):
719             """Builds the F_plus vector for z-system
720
721                 F_plus_z = [-N^t * A^t, 0]
722             """
723
724             A = self.A_10_z
725             N = self.N_z
726
727             left = np.zeros((self.vec_num_points, 1))
728             right = np.reshape(-np.transpose(N) * A.transpose(),
729                                (self.vec_num_points, 1))
730
731             self.F_plus_z = np.concatenate((left, right))
732
733         def feedback_z(self, z):
734             """Feedback control term for z-system
735
736                 Args:
737                     z := current value of system
738
739                 Returns:
740                     feedback control term
741             """
742
743             F_plus = self.F_plus_z
744             N = self.N_z
745
746             prod = np.dot(z, F_plus)[0]
747
748             top = np.zeros((self.vec_num_points, ))
749             bottom = self._g1(prod) * N
750
751             return np.concatenate((bottom, top))
752
753         def control_z_forward(self, t, z):
754             """Determines the forward control term for z
755
756                 Args:
```

```
757                 t := current timestep value
758                 z := current value of system
759
760             Returns:
761                 z modified by the control term
762         """
763
764         if self._use_feedback:
765             feedback = self.feedback_z(z)
766
767             return z - feedback, feedback
768         else:
769             feedback = [0.0, 0.0, 0.0, 0.0]
770
771             return z, feedback
772
773     def control_z_reverse(self, t, z):
774         """Determines the reverse control term for z
775
776             Args:
777                 t := current timestep value
778                 z := current value of system
779
780             Returns:
781                 z modified by the control term
782         """
783
784         feedback = self._storage_z.pop()
785
786         return z + feedback, feedback
787
788     def control_z(self, t, z):
789         """Determines the control term for z
790
791             Args:
792                 t := current timestep value
793                 z := current value of system
794
795             Returns:
796                 z modified by the control term
797         """
798
799         if self._is_forward:
800             return self.control_z_forward(t, z)
801         else:
802             return self.control_z_reverse(t, z)
803
804     def _build_a200_w(self):
805         """Build the A_200 tensor for w-system"""
806
807         self.A_200_w = self._fem.A_200.copy()
808
809     def _coupling_w(self, z, w):
810         """Returns the coupling term for w-system
811
812             Args:
813                 z := current z-system vector
814                 w := current w-system vector
815
816             Returns:
817                 coupling term
818         """
819
820         array = self.A_200_w
```

```
821
822             z0, z1 = self.split_eqn(z)
823             w0, w1 = self.split_eqn(w)
824
825             coupling = np.zeros((self.vec_num_points, ))
826             zeros = np.zeros((self.vec_num_points, ))
827
828             for index, A_k in enumerate(array):
829                 temp = A_k * z0
830                 term = np.dot(w1, temp)
831                 coupling[index] = -term
832
833             return np.concatenate((coupling, zeros))
834
835         def coupling_w(self, t, v, z, w):
836             """Add coupling term for w-system
837
838                 Args:
839                     t := current time
840                     v := current vector
841                     z := current z-system vec
842                     w := current w-system vec
843
844                 Returns:
845                     w modified by coupling
846             """
847
848             if self._use_coupling:
849                 coupling = self._coupling_w(z, w)
850                 return v + coupling
851             else:
852                 return v
853
854         def _build_a110_z(self):
855             """Build the A_110 tensor for z-system"""
856
857             self.A_110_z = self._fem.A_110.copy()
858
859         def _coupling_z(self, u):
860             """Returns the coupling term for z-system
861
862                 Args:
863                     u := the w-system vector
864
865                 Returns:
866                     coupling term
867             """
868
869             array = self.A_110_z
870             v, w = self.split_eqn(u)
871
872             coupling = np.zeros((self.vec_num_points, ))
873             zeros = np.zeros((self.vec_num_points, ))
874
875             for index, A_k in enumerate(array):
876                 temp = A_k * w
877                 coupling[index] = np.dot(v, temp)
878
879             return np.concatenate((coupling, zeros))
880
881         def coupling_z(self, t, z, w):
882             """Add coupling term for z-system
883
884                 Args:
```

```
885                 t := current time
886                 z := current z-system vec
887                 w := current w-system vec
888 
889             Returns:
890                 z modified by coupling
891         """
892 
893         if self._use_coupling:
894             coupling = self._coupling_z(w)
895             return z + coupling
896         else:
897             return z
898 
899     def rhs_w(self, t, z, w):
900         """Computes the right hand side of equation for w-system
901 
902             Args:
903                 t := current time
904                 z := current z-system vec
905                 w := current w-system vec
906 
907             Returns:
908                 w rhs
909         """
910 
911         M = self.M_w
912         A = self.A_w
913 
914         v0, storage = self.control_w(t, w)
915 
916         v1 = A * v0
917 
918         b = self.coupling_w(t, v1, z, w)
919 
920         return lin.spsolve(M, b), storage
921 
922     def rhs_z(self, t, z, w):
923         """Computes the right hand side of equation for z-system
924 
925             Args:
926                 t := current time
927                 z := current z-system vec
928                 w := current w-system vec
929 
930             Returns:
931                 z rhs
932         """
933 
934         M = self.M_z
935         A = self.A_z
936 
937         v0, storage = self.control_z(t, z)
938 
939         v1 = A * v0
940 
941         b = self.coupling_z(t, v1, w)
942 
943         return lin.spsolve(M, b), storage
944 
945     def rhs(self, t, u):
946         """Computes the right hand side of whole system
947 
948             Args:
```

```
949                 t := current time
950                 u := current system vec
951
952             Returns:
953                 rhs
954         """
955
956         z, w = self.split_solution(u)
957
958         new_z, storage_z = self.rhs_z(t, z, w)
959         new_w, storage_w = self.rhs_w(t, z, w)
960
961         return np.concatenate((new_z, new_w)), storage_z, storage_w
962
963     def step(self, t, u):
964         """Computes the new step for whole system
965
966             Args:
967                 t := current time
968                 u := current system vec
969
970             Returns:
971                 new system vec, storage_z, storage_w
972         """
973
974         return self._solver(self.rhs, t, u, usereverse=True)
975
976     def energy_w(self, w):
977         """Calculates the energy for w-system
978
979             Args:
980                 w := current system vec
981
982             Returns:
983                 energy for w
984         """
985
986         E = self.E_w
987         v = E * w
988         value = np.dot(w, v)
989
990         return 0.5 * value
991
992     def energy_z(self, z):
993         """Calculates the energy for z-system
994
995             Args:
996                 z := current system vec
997
998             Returns:
999                 energy for z
1000          """
1001
1002         M = self.M_ez
1003         v = M * z
1004         value = np.dot(z, v)
1005
1006         return 0.5 * value
1007
1008    def energy(self, u):
1009        """Calculates all of the energies
1010
1011            Args:
1012                u := current system vec
```

```
1013
1014                Returns:
1015                    energy for z, energy for w, total energy
1016            """
1017
1018            z, w = self.split_solution(u)
1019
1020            e_z = self.energy_z(z)
1021            e_w = self.energy_w(w)
1022
1023            return e_z, e_w, e_z + e_w
1024
1025        def run_normal(self):
1026            """Run the solver"""
1027
1028            storage_z = []
1029            storage_w = []
1030
1031            solution = np.zeros((self._num_steps + 1, 4 * self.vec_num_points))
1032            solution[0, :] = np.concatenate((self.init_z, self.init_w))
1033
1034            e_z = np.zeros((self._num_steps + 1, 1))
1035            e_w = np.zeros((self._num_steps + 1, 1))
1036            e_t = np.zeros((self._num_steps + 1, 1))
1037
1038            e_z[0], e_w[0], e_t[0] = self.energy(solution[0, :])
1039
1040            if self._use_feedback:
1041                step_str = "Step feedback: "
1042            else:
1043                step_str = "Step: "
1044
1045            t = 0.0
1046            for index in range(self._num_steps):
1047                print(step_str + str(index + 1))
1048
1049                t += self._delta_t
1050                u = solution[index, :]
1051
1052                new, s_z, s_w = self.step(t, u)
1053
1054                solution[index + 1, :] = new
1055                storage_z.extend(s_z)
1056                storage_w.extend(s_w)
1057
1058                e_z[index + 1], e_w[index + 1], e_t[index + 1] = self.energy(new)
1059
1060            return solution, e_z, e_w, e_t, storage_z, storage_w
1061
1062        def run_reverse(self):
1063            """Run the solver forward for reverse control"""
1064
1065            storage_z = []
1066            storage_w = []
1067
1068            solution = np.zeros((self._max_steps + 1, 4 * self.vec_num_points))
1069            solution[0, :] = np.concatenate((self.init_z, self.init_w))
1070
1071            e_z = np.zeros((self._max_steps + 1, 1))
1072            e_w = np.zeros((self._max_steps + 1, 1))
1073            e_t = np.zeros((self._max_steps + 1, 1))
1074
1075            e_z[0], e_w[0], e_t[0] = self.energy(solution[0, :])
1076
```

```
1077            if self._use_feedback:
1078                step_str = "Step feedback: "
1079            else:
1080                step_str = "Step: "
1081
1082            t = 0.0
1083            count = 0
1084            e_c = e_t[0]
1085            while (e_c > self._epsilon) and (count < self._max_steps):
1086                t += self._delta_t
1087                u = solution[count, :]
1088
1089                count += 1
1090                print(step_str + str(count))
1091
1092                new, s_z, s_w = self.step(t, u)
1093
1094                solution[count, :] = new
1095                storage_z.extend(s_z)
1096                storage_w.extend(s_w)
1097
1098                e_z[count], e_w[count], e_t[count] = self.energy(new)
1099
1100                e_c = e_t[count]
1101
1102            solution_trim = solution[:count]
1103
1104            e_z_trim = e_z[:count]
1105            e_w_trim = e_w[:count]
1106            e_t_trim = e_t[:count]
1107
1108            return solution_trim, e_z_trim, e_w_trim, e_t_trim, storage_z, storage_w
1109
1110        def run(self):
1111            """Choose correct run system"""
1112
1113            if self._use_reverse:
1114                print("Running reverse feedback calculation")
1115                return self.run_reverse()
1116            else:
1117                print("Running calculation")
1118                return self.run_normal()
```

A.7 feedback_run.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from source import initial
4  from source import feedback
5
6
7  if __name__ == '__main__':
8      print("Running the feedback system")
9
10     use_feedback = True
11     use_coupling = False
12     use_reverse = False
13
14     epsilon = 0.001
15     max_steps = 50000
16
17     num_points = 11
```

```
18      delta_x = 1.0 / (num_points - 1)
19
20      delta_t = 1.0 / 10.0 * delta_x
21
22      num_steps = 4000
23
24      gamma = 0.1
25      rho = gamma**2
26
27      init_v = initial.InitV()
28      init_w = initial.InitW()
29      init_z0 = initial.InitZ0()
30      init_z1 = initial.InitZ1()
31
32      g1 = initial.G1()
33      g2 = initial.G2()
34      m = initial.M()
35
36      init_storage_w = None
37      init_storage_z = None
38
39      system = feedback.FeedbackControl.setup(num_points, delta_x,
40                                              num_steps, delta_t,
41                                              gamma, rho,
42                                              init_v, init_w, init_z0, init_z1,
43                                              g1, g2, m,
44                                              use_feedback, use_coupling,
45                                              use_reverse,
46                                              epsilon, max_steps,
47                                              init_storage_w, init_storage_z)
48
49      solution, e_z, e_w, e_t, storage_z, storage_w = system.run()
50
51      min_z = np.amin(e_z)
52      max_z = np.amax(e_z)
53      min_w = np.amin(e_w)
54      max_w = np.amax(e_w)
55      min_t = np.amin(e_t)
56      max_t = np.amax(e_t)
57
58      print('Min energy_z: ' + str(min_z))
59      print('Max energy_z: ' + str(max_z))
60      print('Min energy_w: ' + str(min_w))
61      print('Max energy_w: ' + str(max_w))
62      print('Min energy_t: ' + str(min_t))
63      print('Max energy_t: ' + str(max_t))
64
65      lower_plot = 0.001
66      upper_plot = 0.001
67
68      fig = plt.figure()
69      full_title = 'Energy for full system'
70      if use_feedback:
71          full_title += ', with feedback'
72      if use_coupling:
73          full_title += ', with coupling'
74      plt.suptitle(full_title)
75
76      plt.subplot(1, 3, 1)
77      plt.ylim([min_z - lower_plot, max_z + upper_plot])
78      plt.plot(e_z, linewidth=1.5)
79      z_title = 'Energy for z-system'
80      if use_feedback:
81          z_title += ', with feedback'
```

```
82      if use_coupling:
83          z_title += ', with coupling'
84      plt.title(z_title)
85      plt.xlabel('Timestep')
86      plt.ylabel('Energy')
87
88      plt.subplot(1, 3, 2)
89      plt.ylim([min_w - lower_plot, max_w + upper_plot])
90      plt.plot(e_w, linewidth=1.5)
91      w_title = 'Energy for w-system'
92      if use_feedback:
93          w_title += ', with feedback'
94      if use_coupling:
95          w_title += ', with coupling'
96      plt.title(w_title)
97      plt.xlabel('Timestep')
98      plt.ylabel('Energy')
99
100     plt.subplot(1, 3, 3)
101     plt.ylim([min_t - lower_plot, max_t + upper_plot])
102     plt.plot(e_t, linewidth=1.5)
103     t_title = 'Total energy for system'
104     if use_feedback:
105         t_title += ', with feedback'
106     if use_coupling:
107         t_title += ', with coupling'
108     plt.title(t_title)
109     plt.xlabel('Timestep')
110     plt.ylabel('Energy')
111
112     plt.show(block=False)
```

A.8 source/null.py

```
1  import numpy as np
2  import scipy.sparse as sp
3  import scipy.sparse.linalg as lin
4
5  from source import finite_elements as fem
6  from source import solver
7
8
9  class OnesFunction(object):
10     """Class to contain the ones-function system"""
11
12     def __call__(self, x):
13         """Return the value of the function"""
14
15         return 1.0
16
17
18 class NullControl(object):
19     """Class to contain the null control system
20
21         Attributes:
22             _num_points := the number of space points (includes endpoints)
23             _delta_x    := distance between space points
24             _num_steps  := number of time steps to take
25             _delta_t    := amount of time to take in one timestep
26
27             _gamma := constant for w-system
28             _rho   := constant for w-system
```

```
29              _T := null control target time
30
31              _init_v  := initial condition for v
32              _init_w  := initial condition for w
33              _init_z0 := initial condition for z0
34              _init_z1 := initial condition for z1
35
36              _use_null     := turn on and off null control
37              _use_coupling := turn on and off coupling
38
39              _storage_w := reverse control pre computed values for w
40              _storage_z := reverse control pre computed values for z
41
42              _is_forward := Determine reverse state
43      """
44
45      def __init__(self, num_points, delta_x, num_steps, delta_t,
46                   gamma, rho, T,
47                   init_v, init_w, init_z0, init_z1,
48                   use_null, use_coupling,
49                   storage_w, storage_z):
50          self._num_points = num_points
51          self._delta_x = delta_x
52          self._num_steps = num_steps
53          self._delta_t = delta_t
54
55          self._gamma = gamma
56          self._rho = rho
57          self._T = T
58
59          self._init_v = init_v
60          self._init_w = init_w
61          self._init_z0 = init_z0
62          self._init_z1 = init_z1
63
64          self._use_null = use_null
65          self._use_coupling = use_coupling
66
67          self._storage_w = storage_w
68          self._storage_z = storage_z
69
70          if storage_w is None:
71              self._is_forward = True
72          else:
73              self._is_forward = False
74
75      @classmethod
76      def setup(cls, num_points, delta_x, num_steps, delta_t,
77                gamma, rho, T,
78                init_v, init_w, init_z0, init_z1,
79                use_null, use_coupling,
80                storage_w, storage_z):
81          """Setup the system completely"""
82
83          new = cls(num_points, delta_x, num_steps, delta_t,
84                    gamma, rho, T,
85                    init_v, init_w, init_z0, init_z1,
86                    use_null, use_coupling,
87                    storage_w, storage_z)
88          new._build()
89
90          return new
91
92      @property
```

```
93      def vec_num_points(self):
94          """Get the number of points in a vec"""
95
96          return 2 * self._num_points
97
98      def _build(self):
99          """Build the feedback control system"""
100
101         print("Build the solver system")
102         self._build_solver()
103
104         print("Build the FEM system")
105         self._build_fem()
106
107         print("Build the w-system")
108         self._build_a00_w()
109         self._build_a01_w()
110         self._build_a02_w()
111         self._build_a11_w()
112         self._build_a22_w()
113         self._build_ml_w()
114         self._build_m_w()
115         self._build_a_w()
116         self._build_e_w()
117
118         print("Build the z-system")
119         self._build_a10_z()
120         self._build_a_z()
121         self._build_a00_z()
122         self._build_m_z()
123
124         print("Build the init_w")
125         self._build_init_w0()
126         self._build_init_w1()
127         self._build_init_w()
128
129         print("Build the init_z")
130         self._build_init_z0()
131         self._build_init_z1()
132         self._build_init_z()
133
134         print("Build the null_w")
135         self._build_m_w_inv()
136         self._build_f_plus_w()
137         self._build_n_w()
138         self._build_b_w()
139         self._build_abf_w()
140         self._build_y0_w()
141
142         print("Build the null_z")
143         self._build_m_z_inv()
144         self._build_n_z()
145         self._build_f_plus_z()
146         self._build_b_z()
147         self._build_abf_z()
148         self._build_y0_z()
149
150         print("Build the coupling_z")
151         self._build_a110_z()
152
153         print("Build the coupling_w")
154         self._build_a200_w()
155
156     def _build_solver(self):
```

```
157             """Build the time-solver"""
158
159             self._solver = solver.RK4(self._delta_t)
160
161         def _build_fem(self):
162             """Build the fem system"""
163
164             self._fem = fem.FiniteElements.setup(self._num_points, self._delta_x)
165
166         def split_eqn(self, u):
167             """Splits the equation u into it's two parts
168
169                 Args:
170                     u := equation to split
171
172                 Returns:
173                     the two parts
174             """
175
176             u0 = u[:self.vec_num_points]
177             u1 = u[self.vec_num_points:]
178
179             return u0, u1
180
181         def split_solution(self, u):
182             """Splits the solution u into the w and z systems
183
184                 Args:
185                     u := equation to split
186
187                 Returns:
188                     the two parts
189             """
190
191             z = u[:2 * self.vec_num_points]
192             w = u[2 * self.vec_num_points:]
193
194             return z, w
195
196         @staticmethod
197         def identify_matrix(matrix, index):
198             """Identify row and column of matrix corresponding to index
199
200                 Args:
201                     matrix := matrix to identify
202                     index  := index of row/column to identify
203             """
204
205             matrix[:, index] = 0.0
206             matrix[index, :] = 0.0
207             matrix[index, index] = 1.0
208
209         def _build_a00_w(self):
210             """Build the A_00 matrix for w-system
211
212                 A_00 -> identified for B.C.s
213             """
214
215             A = self._fem.A_00.copy().tolil()
216
217             self.identify_matrix(A, 0)
218             self.identify_matrix(A, self._num_points)
219
220             self.A_00_w = A.tocsc()
```

```
221
222         def _build_a01_w(self):
223             """Builds the A_01 matrix for the w-system
224
225                 A_01 -> identified for B.C.s
226             """
227
228             A = self._fem.A_01.copy().tolil()
229
230             self.identify_matrix(A, 0)
231             self.identify_matrix(A, self._num_points)
232
233             self.A_01_w = A.tocsc()
234
235         def _build_a02_w(self):
236             """Builds the A_02 matrix for the w-system
237
238                 A_02 -> identified for B.C.s
239             """
240
241             A = self._fem.A_02.copy().tolil()
242
243             self.identify_matrix(A, 0)
244             self.identify_matrix(A, self._num_points)
245
246             self.A_02_w = A.tocsc()
247
248         def _build_a11_w(self):
249             """Builds the A_11 (Laplacian) matrix for the w-system
250
251                 A_11 -> identified for B.C.s
252             """
253
254             A = self._fem.A_11.copy().tolil()
255
256             self.identify_matrix(A, 0)
257             self.identify_matrix(A, self._num_points)
258
259             A[0, 0] = 0.0
260             A[self._num_points, self._num_points] = 0.0
261
262             self.A_11_w = A.tocsc()
263
264         def _build_a22_w(self):
265             """Builds the A_22 (Biharmonic) matrix for the w-system
266
267                 A_22 -> identified for B.C.s
268             """
269
270             A = self._fem.A_22.copy().tolil()
271
272             self.identify_matrix(A, 0)
273             self.identify_matrix(A, self._num_points)
274
275             self.A_22_w = A.tocsc()
276
277         def _build_ml_w(self):
278             """Builds the mass/lapacian operator matrix for the w-system
279
280                 ML_w = A_00_w + (rho * A_11_w)
281             """
282
283             M = self.A_00_w.copy()
284             L = self.A_11_w.copy()
```

```
285
286             self.ML_w = (M + (self._rho * L)).tocsc()
287
288         def _build_m_w(self):
289             """Builds the mass matrix for the w-system
290
291                 M_w = [[ML_w,  None],
292                        [None,  Id   ]]
293             """
294
295             top = self.ML_w.copy()
296             bottom = sp.identity(self.vec_num_points).tocsc()
297
298             self.M_w = sp.bmat([[top,   None],
299                                 [None,  bottom]]).tocsc()
300
301         def _build_a_w(self):
302             """Builds the A matrix for the w-system
303
304                 A_w = [[None, -gamma^2 * A_22_w],
305                        [  Id,  None]]
306             """
307
308             B = self.A_22_w.copy()
309
310             top = -self._gamma**2 * B
311             bottom = sp.identity(self.vec_num_points)
312
313             self.A_w = sp.bmat([[None,    top],
314                                 [bottom,  None]]).tocsc()
315
316         def _build_e_w(self):
317             """Builds the energy matrix for w-system
318
319                 E_w = [[ML_w,  None],
320                        [None,  gamma^2 * A_22_w]]
321             """
322
323             B = self.A_22_w.copy()
324
325             top = self.ML_w.copy()
326             bottom = self._gamma**2 * B
327
328             self.E_w = sp.bmat([[top,   None],
329                                 [None,  bottom]]).tocsc()
330
331         def _build_a10_z(self):
332             """Build the A_10 matrix for the z-system
333
334                 A_10 -> identified for B.C.s
335             """
336
337             A = self._fem.A_10.copy().tolil()
338
339             self.identify_matrix(A, self._num_points - 1)
340
341             self.A_10_z = A.tocsc()
342
343         def _build_a_z(self):
344             """Build the A matrix for the z-system
345
346                 A_z = [[       0,  A_10_z.transpose()],
347                        [-A_10_z,  0                   ]]
348             """
```

```
349
350         A = self.A_10_z.copy()
351
352         top = A.copy().transpose()
353         bottom = -A.copy()
354
355         self.A_z = sp.bmat([[None,    top],
356                             [bottom, None]]).tocsc()
357
358     def _build_a00_z(self):
359         """Build the A_00 matrix for z-system
360
361             A_00 -> identified for B.C.s
362         """
363
364         A = self._fem.A_00.copy().tolil()
365
366         self.A_00_z = A.tocsc()
367
368     def _build_m_z(self):
369         """Build the 'mass' matrix for z-system
370
371             M_z = [[A_00_z,       0],
372                    [     0, A_00_z]]
373         """
374
375         A = self.A_00_z.copy()
376
377         self.M_z = sp.bmat([[A,     None],
378                             [None, A]]).tocsc()
379
380     def _build_init_z0(self):
381         """Build initial vector z0"""
382
383         if isinstance(self._init_z0, np.ndarray):
384             self.init_z0 = self._init_z0
385         else:
386             self.init_z0 = self._fem.build_vec(self._init_z0)
387
388     def _build_init_z1(self):
389         """Build initial vector z1"""
390
391         if isinstance(self._init_z1, np.ndarray):
392             self.init_z1 = self._init_z1
393         else:
394             self.init_z1 = self._fem.build_vec(self._init_z1)
395
396     def _build_init_w0(self):
397         """Build initial vector v"""
398
399         if isinstance(self._init_v, np.ndarray):
400             self.init_w0 = self._init_v
401         else:
402             vec = self._fem.build_vec(self._init_v)
403
404             vec[0] = 0.0
405             vec[self._num_points] = 0.0
406
407             M = self.ML_w.copy()
408
409             self.init_w0 = lin.spsolve(M, vec)
410
411     def _build_init_w1(self):
412         """Build initial vector w"""
```

```
413
414             if isinstance(self._init_w, np.ndarray):
415                 self.init_w1 = self._init_w
416             else:
417                 vec = self._fem.build_vec(self._init_w)
418
419                 vec[0] = 0.0
420                 vec[self._num_points] = 0.0
421
422                 M = self.A_22_w.copy()
423
424                 self.init_w1 = lin.spsolve(M, vec)
425
426         def _build_init_w(self):
427             """Build the initial vector for w-system"""
428
429             vec0 = self.init_w0
430             vec1 = self.init_w1
431
432             self.init_w = np.concatenate((vec0, vec1))
433
434         def _build_init_z(self):
435             """Build initial vector for z-system"""
436
437             vec0 = self.init_z0
438             vec1 = self.init_z1
439
440             vec = np.concatenate((vec0, vec1))
441
442             if isinstance(self._init_z0, np.ndarray):
443                 self.init_z = vec
444             else:
445                 M = self.M_z.copy()
446                 vec = lin.spsolve(M, vec)
447
448                 self.init_z = vec
449
450         def _build_m_w_inv(self):
451             """Build the inverse of M_w for w-system control"""
452
453             M = self.M_w.copy()
454
455             self.M_w_inv = lin.inv(M).tocsc()
456
457         def _build_f_plus_w(self):
458             """Build the F_plus for the w-system"""
459
460             F = np.zeros((2, 2 * self.vec_num_points))
461
462             F[0, self.vec_num_points - 1] = 1.0
463             F[1, self._num_points - 1] = 1.0
464
465             self.F_plus_w = F
466
467         def _build_n_w(self):
468             """Build the N_w matrix"""
469
470             F = np.zeros((2, self.vec_num_points))
471             F[0, self.vec_num_points - 1] = 1
472             F[1, self._num_points - 1] = 1
473
474             B = self.A_22_w.copy()
475
476             A = -self._gamma**2 * B
```

```
477             A_inv = lin.inv(A)
478
479             Nt = F * A_inv
480
481             N = Nt.transpose()
482             fill = np.zeros(N.shape)
483
484             self.N_w = np.concatenate((N, fill), axis=0)
485
486         def _build_b_w(self):
487             """Build the B matrix for null control in w-system"""
488
489             N = self.N_w.copy()
490             B = self.A_22_w.copy()
491             fill = np.zeros(B.shape)
492
493             A = sp.bmat([[B,     None],
494                          [None, fill]]).tocsc()
495
496             self.B_w = A * N
497
498         def _build_abf_w(self):
499             """Build the ABF (exponent) for the exp function for
500                 w-system control
501
502                 ABF_w = M_w^-1 (A_w + B_w*F_plus_w)
503             """
504
505             A = self.A_w.copy()
506             B = self.B_w.copy()
507             F = self.F_plus_w.copy()
508             M = self.M_w_inv.copy()
509
510             BF = sp.csc_matrix(np.matmul(B, F))
511             ABF = A + BF
512
513             self.ABF_w = M * ABF
514
515         def exp_w(self, t):
516             """Computes e^(A + BF)t for w-system:
517
518                 f(t) = exp(inv(M_w)(A_w + B_w * F_plus_w)t )
519
520                 Args:
521                     t := time
522
523                 Returns
524                     f(t)
525             """
526
527             ABF = self.ABF_w
528             exp = ABF * t
529
530             return lin.expm(exp)
531
532         def _build_y0_w(self):
533             """Build the y0 term for w-system control"""
534
535             I = sp.identity(2 * self.vec_num_points)
536             A = self.A_w.copy()
537             M = self.M_w_inv.copy()
538             T = self._T
539             MA = M * A
540
```

```
541             exp_ABF_T = self.exp_w(T)
542             exp_A_T = lin.expm(-T * MA)
543             exp = exp_A_T * exp_ABF_T
544             I_exp = I - exp
545 
546             y0 = self.init_w.copy()
547             y0_new = np.reshape(y0, (2 * self.vec_num_points, 1))
548 
549             sol = lin.spsolve(I_exp, y0_new)
550 
551             self.y0_w = np.reshape(sol, (2 * self.vec_num_points, 1))
552 
553         def u_w(self, t):
554             """Computes the u(t) term for the w-system control
555 
556                 u(t) = F_plus_w * exp_w(t) * y0_w
557 
558                 Args:
559                     t := time
560 
561                 Returns:
562                     u(t)
563             """
564 
565             F = self.F_plus_w
566             y0 = self.y0_w
567             exp = self.exp_w(t)
568 
569             prod = F * exp
570 
571             return np.matmul(prod, y0)
572 
573         def null_w(self, t):
574             """Computes the null control term for w-system
575 
576                 N(t) = B_w * u_w(t)
577 
578                 Args:
579                     t := time
580 
581                 Returns:
582                     N(t)
583             """
584 
585             B = self.B_w
586             u = self.u_w(t)
587 
588             null = np.matmul(B, u)
589 
590             return np.reshape(null, (2 * self.vec_num_points, ))
591 
592         def control_w_forward(self, t, w):
593             """Determines the forward control term for w
594 
595                 Args:
596                     t := current timestep value
597                     w := current value of system
598 
599                 Returns:
600                     w modified by the control term
601             """
602 
603             if self._use_null:
604                 null = self.null_w(t)
```

```
605
606             return w + null, null
607         else:
608             null = [0.0, 0.0, 0.0, 0.0]
609
610             return w, null
611
612     def control_w_reverse(self, t, w):
613         """Determines the reverse control term for w
614
615             Args:
616                 t := current timestep value
617                 w := current value of system
618
619             Returns:
620                 w modified by the control term
621         """
622
623         null = self._storage_w.pop()
624
625         return w - null, null
626
627     def control_w(self, t, w):
628         """Determines the control term for w
629
630             Args:
631                 t := current timestep value
632                 w := current value of system
633
634             Returns:
635                 w modified by the control term
636         """
637
638         if self._is_forward:
639             return self.control_w_forward(t, w)
640         else:
641             return self.control_w_reverse(t, w)
642
643     def _build_m_z_inv(self):
644         """Build the inverse of M for z-system"""
645
646         M = self.M_z.copy()
647
648         self.M_z_inv = lin.inv(M).tocsc()
649
650     def _build_n_z(self):
651         """Builds the N_z vector
652
653             l2 projection of ones-function
654         """
655
656         f = OnesFunction()
657
658         self.N_z = self._fem.l2_project(f, identify=False)
659
660     def _build_f_plus_z(self):
661         """Builds the F_plus vector for z-system"""
662
663         N = self.N_z.copy()
664         A = self.A_10_z.copy().transpose().tocsc()
665
666         prod = np.transpose(-N) * A
667
668         left = np.zeros((1, self.vec_num_points))
```

```
669             right = np.reshape(prod, (1, self.vec_num_points))
670
671             self.F_plus_z = np.concatenate((left, right), axis=1)
672
673         def _build_b_z(self):
674             """Builds the B matrix for z-system control
675
676                 B = -[0, A_10_z * N_z]^t
677             """
678
679             A = self.A_10_z.copy()
680             N = self.N_z.copy()
681
682             fill = np.zeros((self.vec_num_points, ))
683             b = A * N
684
685             self.B_z = np.concatenate((fill, b))
686
687         def _build_abf_z(self):
688             """Build the ABF (exponent) for the exp function for
689                 z-system control
690
691                 ABF_z = M_z^-1 (A_z + B_z*F_plus_z)
692             """
693
694             A = self.A_z.copy()
695             B = self.B_z.copy()
696             F = self.F_plus_z.copy()
697             M = self.M_z_inv.copy()
698
699             B_new = np.reshape(B, (2 * self.vec_num_points, 1))
700             BF = sp.csc_matrix(B_new * F)
701             ABF = A + BF
702
703             self.ABF_z = M * ABF
704
705         def exp_z(self, t):
706             """Computes e^(A + BF)t for z-system:
707
708                 f(t) = exp(inv(M_z)(A_z + B_z * F_plus_z)t )
709
710                 Args:
711                     t := time
712
713                 Returns
714                     f(t)
715             """
716
717             ABF = self.ABF_z
718             exp = ABF * t
719
720             return lin.expm(exp)
721
722         def _build_y0_z(self):
723             """Build the y0 term for z-system control"""
724
725             I = sp.identity(2 * self.vec_num_points)
726             A = self.A_z.copy()
727             M = self.M_z_inv.copy()
728             T = self._T
729             MA = M * A
730
731             exp_ABF_T = self.exp_z(T)
732             exp_A_T = lin.expm(-T * MA)
```

```
733             exp = exp_A_T * exp_ABF_T
734             I_exp = (I - exp).tocsc()
735 
736             y0 = self.init_z.copy()
737 
738             sol = lin.spsolve(I_exp, y0)
739 
740             self.y0_z = np.reshape(sol, (2 * self.vec_num_points, 1))
741 
742         def u_z(self, t):
743             """Computes the u(t) term for the z-system control
744 
745                 u(t) = F_plus_z * exp_z(t) * y0_z
746 
747                 Args:
748                     t := time
749 
750                 Returns:
751                     u(t)
752             """
753 
754             F = self.F_plus_z
755             y0 = self.y0_z
756             exp = self.exp_z(t)
757 
758             prod = F * exp
759 
760             return np.matmul(prod, y0)[0, 0]
761 
762         def null_z(self, t):
763             """Computes the null control term for z-system
764 
765                 N(t) = B_z * u_z(t)
766 
767                 Args:
768                     t := time
769 
770                 Returns:
771                     N(t)
772             """
773 
774             B = self.B_z
775             u = self.u_z(t)
776 
777             return B * u
778 
779         def control_z_forward(self, t, z):
780             """Determines the forward control term for z
781 
782                 Args:
783                     t := current timestep value
784                     z := current value of system
785 
786                 Returns:
787                     z modified by the control term
788             """
789 
790             if self._use_null:
791                 null = self.null_z(t)
792 
793                 return z + null, null
794             else:
795                 null = [0.0, 0.0, 0.0, 0.0]
796 
```

```
797             return z, null
798
799     def control_z_reverse(self, t, z):
800         """Determines the reverse control term for z
801
802             Args:
803                 t := current timestep value
804                 z := current value of system
805
806             Returns:
807                 w modified by the control term
808         """
809
810         null = self._storage_z.pop()
811
812         return z + null, null
813
814     def control_z(self, t, z):
815         """Determines the control term for z
816
817             Args:
818                 t := current timestep value
819                 z := current value of system
820
821             Returns:
822                 z modified by the control term
823         """
824
825         if self._is_forward:
826             return self.control_z_forward(t, z)
827         else:
828             return self.control_z_reverse(t, z)
829
830     def _build_a200_w(self):
831         """Build the A_200 tensor for w-system"""
832
833         self.A_200_w = self._fem.A_200.copy()
834
835     def _coupling_w(self, z, w):
836         """Returns the coupling term for w-system
837
838             Args:
839                 z := current z-system vector
840                 w := current w-system vector
841
842             Returns:
843                 coupling term
844         """
845
846         array = self.A_200_w
847
848         z0, z1 = self.split_eqn(z)
849         w0, w1 = self.split_eqn(w)
850
851         coupling = np.zeros((self.vec_num_points, ))
852         zeros = np.zeros((self.vec_num_points, ))
853
854         for index, A_k in enumerate(array):
855             temp = A_k * z0
856             term = np.dot(w1, temp)
857             coupling[index] = -term
858
859         return np.concatenate((coupling, zeros))
860
```

```
861     def coupling_w(self, t, v, z, w):
862         """Add coupling term for w-system
863 
864             Args:
865                 t := current time
866                 v := current vector
867                 z := current z-system vec
868                 w := current w-system vec
869 
870             Returns:
871                 w modified by coupling
872         """
873 
874         if self._use_coupling:
875             coupling = self._coupling_w(z, w)
876             return v + coupling
877         else:
878             return v
879 
880     def _build_a110_z(self):
881         """Build the A_110 tensor for z-system"""
882 
883         self.A_110_z = self._fem.A_110.copy()
884 
885     def _coupling_z(self, u):
886         """Returns the coupling term for z-system
887 
888             Args:
889                 u := the w-system vector
890 
891             Returns:
892                 coupling term
893         """
894 
895         array = self.A_110_z
896         v, w = self.split_eqn(u)
897 
898         coupling = np.zeros((self.vec_num_points, ))
899         zeros = np.zeros((self.vec_num_points, ))
900 
901         for index, A_k in enumerate(array):
902             temp = A_k * w
903             coupling[index] = np.dot(v, temp)
904 
905         return np.concatenate((coupling, zeros))
906 
907     def coupling_z(self, t, z, w):
908         """Add coupling term for z-system
909 
910             Args:
911                 t := current time
912                 z := current z-system vec
913                 w := current w-system vec
914 
915             Returns:
916                 z modified by coupling
917         """
918 
919         if self._use_coupling:
920             coupling = self._coupling_z(w)
921             return z + coupling
922         else:
923             return z
924 
```

```
925     def rhs_w(self, t, z, w):
926         """Computes the right hand side of equation for w-system
927 
928             Args:
929                 t := current time
930                 z := current z-system vec
931                 w := current w-system vec
932 
933             Returns:
934                 w rhs
935         """
936 
937         M = self.M_w
938         A = self.A_w
939 
940         v0 = A * w
941 
942         v1, storage = self.control_w(t, v0)
943 
944         b = self.coupling_w(t, v1, z, w)
945 
946         return lin.spsolve(M, b), storage
947 
948     def rhs_z(self, t, z, w):
949         """Computes the right hand side of equation for z-system
950 
951             Args:
952                 t := current time
953                 z := current z-system vec
954                 w := current w-system vec
955 
956             Returns:
957                 z rhs
958         """
959 
960         M = self.M_z
961         A = self.A_z
962 
963         v0 = A * z
964 
965         v1, storage = self.control_z(t, v0)
966 
967         b = self.coupling_z(t, v1, w)
968 
969         return lin.spsolve(M, b), storage
970 
971     def rhs(self, t, u):
972         """Computes the right hand side of whole system
973 
974             Args:
975                 t := current time
976                 u := current system vec
977 
978             Returns:
979                 rhs
980         """
981 
982         z, w = self.split_solution(u)
983 
984         new_z, storage_z = self.rhs_z(t, z, w)
985         new_w, storage_w = self.rhs_w(t, z, w)
986 
987         return np.concatenate((new_z, new_w)), storage_z, storage_w
988 
```

```
989         def step(self, t, u):
990             """Computes the new step for whole system
991 
992                 Args:
993                     t := current time
994                     u := current system vec
995 
996                 Returns:
997                     new system vec, storage_z, storage_w
998             """
999 
1000            return self._solver(self.rhs, t, u, usereverse=True)
1001
1002        def energy_w(self, w):
1003            """Calculates the energy for w-system
1004
1005                Args:
1006                    w := current system vec
1007
1008                Returns:
1009                    energy for w
1010            """
1011
1012            E = self.E_w
1013            v = E * w
1014            value = np.dot(w, v)
1015
1016            return 0.5 * value
1017
1018        def energy_z(self, z):
1019            """Calculates the energy for z-system
1020
1021                Args:
1022                    z := current system vec
1023
1024                Returns:
1025                    energy for z
1026            """
1027
1028            M = self.M_z
1029            v = M * z
1030            value = np.dot(z, v)
1031
1032            return 0.5 * value
1033
1034        def energy(self, u):
1035            """Calculates all of the energies
1036
1037                Args:
1038                    u := current system vec
1039
1040                Returns:
1041                    energy for z, energy for w, total energy
1042            """
1043
1044            z, w = self.split_solution(u)
1045
1046            e_z = self.energy_z(z)
1047            e_w = self.energy_w(w)
1048
1049            return e_z, e_w, e_z + e_w
1050
1051        def run(self):
1052            """Run the solver"""
```

```
1053
1054            storage_z = []
1055            storage_w = []
1056
1057            solution = np.zeros((self._num_steps + 1, 4 * self.vec_num_points))
1058            solution[0, :] = np.concatenate((self.init_z, self.init_w))
1059
1060            e_z = np.zeros((self._num_steps + 1, 1))
1061            e_w = np.zeros((self._num_steps + 1, 1))
1062            e_t = np.zeros((self._num_steps + 1, 1))
1063
1064            e_z[0], e_w[0], e_t[0] = self.energy(solution[0, :])
1065
1066            if self._use_null:
1067                step_str = "Step null: "
1068            else:
1069                step_str = "Step: "
1070
1071            t = 0.0
1072            for index in range(self._num_steps):
1073                print(step_str + str(index + 1))
1074
1075                t += self._delta_t
1076                u = solution[index, :]
1077
1078                new, s_z, s_w = self.step(t, u)
1079
1080                solution[index + 1, :] = new
1081                storage_z.extend(s_z)
1082                storage_w.extend(s_w)
1083
1084                e_z[index + 1], e_w[index + 1], e_t[index + 1] = self.energy(new)
1085
1086            return solution, e_z, e_w, e_t, storage_z, storage_w
```

A.9 null_run.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from source import initial
4  from source import null
5
6
7  if __name__ == '__main__':
8      print("Running the null system")
9
10     use_null = True
11     use_coupling = True
12
13     T = 10.0
14
15     num_points = 11
16     delta_x = 1.0 / (num_points - 1)
17
18     delta_t = 1.0 / 10.0 * delta_x
19
20     num_steps = int(T / delta_t)
21
22     gamma = 0.1
23     rho = gamma**2
24
25     init_v = initial.InitV()
```

```
26      init_w = initial.InitW()
27      init_z0 = initial.InitZ0()
28      init_z1 = initial.InitZ1()
29
30      init_storage_w = None
31      init_storage_z = None
32
33      system = null.NullControl.setup(num_points, delta_x,
34                                      num_steps, delta_t,
35                                      gamma, rho, T,
36                                      init_v, init_w, init_z0, init_z1,
37                                      use_null, use_coupling,
38                                      init_storage_w, init_storage_z)
39
40      solution, e_z, e_w, e_t, storage_z, storage_w = system.run()
41
42      min_z = np.amin(e_z)
43      max_z = np.amax(e_z)
44      min_w = np.amin(e_w)
45      max_w = np.amax(e_w)
46      min_t = np.amin(e_t)
47      max_t = np.amax(e_t)
48
49      print('Min energy_z: ' + str(min_z))
50      print('Max energy_z: ' + str(max_z))
51      print('Min energy_w: ' + str(min_w))
52      print('Max energy_w: ' + str(max_w))
53      print('Min energy_t: ' + str(min_t))
54      print('Max energy_t: ' + str(max_t))
55
56      lower_plot = 0.001
57      upper_plot = 0.001
58
59      fig = plt.figure()
60      full_title = 'Energy for full system'
61      if use_null:
62          full_title += ', with null'
63      if use_coupling:
64          full_title += ', with coupling'
65      plt.suptitle(full_title)
66
67      plt.subplot(1, 3, 1)
68      plt.ylim([min_z - lower_plot, max_z + upper_plot])
69      plt.plot(e_z)
70      z_title = 'Energy for z-system'
71      if use_null:
72          z_title += ', with null'
73      if use_coupling:
74          z_title += ', with coupling'
75      plt.title(z_title)
76      plt.xlabel('timestep')
77      plt.ylabel('energy')
78
79      plt.subplot(1, 3, 2)
80      plt.ylim([min_w - lower_plot, max_w + upper_plot])
81      plt.plot(e_w)
82      w_title = 'Energy for w-system'
83      if use_null:
84          w_title += ', with null'
85      if use_coupling:
86          w_title += ', with coupling'
87      plt.title(w_title)
88      plt.xlabel('timestep')
89      plt.ylabel('energy')
```

```
90
91      plt.subplot(1, 3, 3)
92      plt.ylim([min_t - lower_plot, max_t + upper_plot])
93      plt.plot(e_t)
94      t_title = 'Energy for t-system'
95      if use_null:
96          t_title += ', with null'
97      if use_coupling:
98          t_title += ', with coupling'
99      plt.title(t_title)
100     plt.xlabel('timestep')
101     plt.ylabel('energy')
102
103     plt.show(block=False)
```

A.10 source/reverse.py

```
1   import numpy as np
2   from source import null
3   from source import feedback
4
5
6   class ReverseControl(object):
7       """Class to contain the reverse control
8
9           Attributes:
10              _num_points := the number of space points (includes endpoints)
11              _delta_x    := distance between space points
12              _delta_t    := amount of time to take in one timestep
13
14              _gamma := constant for w-system
15              _rho   := constant for w-system
16              _T     := wall time for null control
17
18              _init_v  := initial condition for v
19              _init_w  := initial condition for w
20              _init_z0 := initial condition for z0
21              _init_z1 := initial condition for z1
22
23              _g1 := feedback function for z-system
24              _g2 := feedback function for w-system
25              _m  := feedback function for w-system
26
27              _epsilon   := feedback target energy is less than this
28              _max_steps := max_steps to attempt to achieve target
29
30              _use_coupling := turn on and off coupling
31      """
32
33      def __init__(self, num_points, delta_x, delta_t,
34                   gamma, rho, T,
35                   init_v, init_w, init_z0, init_z1,
36                   g1, g2, m,
37                   epsilon, max_steps,
38                   use_coupling):
39          self._num_points = num_points
40          self._delta_x = delta_x
41          self._delta_t = delta_t
42
43          self._gamma = gamma
44          self._rho = rho
45          self._T = T
```

```
46
47          self._init_v = init_v
48          self._init_w = init_w
49          self._init_z0 = init_z0
50          self._init_z1 = init_z1
51
52          self._g1 = g1
53          self._g2 = g2
54          self._m = m
55
56          self._epsilon = epsilon
57          self._max_steps = max_steps
58
59          self._use_coupling = use_coupling
60
61          self._system_ff = None
62          self._system_fn = None
63          self._system_rn = None
64          self._system_rf = None
65
66      @property
67      def num_steps(self):
68          """Calculate the number of steps"""
69
70          return int(self._T / self._delta_t)
71
72      def build_feedback(self, num_steps,
73                         init_v, init_w, init_z0, init_z1,
74                         use_reverse, storage_w, storage_z):
75          """Build a complete feedback system as needed"""
76
77          return feedback.FeedbackControl.\
78              setup(self._num_points, self._delta_x, num_steps, self._delta_t,
79                    self._gamma, self._rho,
80                    init_v, init_w, init_z0, init_z1,
81                    self._g1, self._g2, self._m,
82                    True, self._use_coupling, use_reverse,
83                    self._epsilon, self._max_steps,
84                    storage_w, storage_z)
85
86      def build_null(self, init_v, init_w, init_z0, init_z1,
87                     storage_w, storage_z):
88          """Build a complete null system as needed"""
89
90          return null.NullControl.\
91              setup(self._num_points, self._delta_x,
92                    self.num_steps, self._delta_t,
93                    self._gamma, self._rho, self._T,
94                    init_v, init_w, init_z0, init_z1,
95                    True, self._use_coupling,
96                    storage_w, storage_z)
97
98      def run_forward_feedback(self):
99          """Run the forward feedback system"""
100
101         system = self.build_feedback(None,
102                                      self._init_v, self._init_w,
103                                      self._init_z0, self._init_z1,
104                                      True, None, None)
105
106         self._system_ff = system
107
108         return system.run()
109
```

```
110     def energy_ff(self, u):
111         """Compute the energies of u"""
112
113         time, space = u.shape
114
115         e_z = np.zeros((time, 1))
116         e_w = np.zeros((time, 1))
117         e_t = np.zeros((time, 1))
118
119         for index in range(time):
120             e_z[index], e_w[index], e_t[index] =\
121                 self._system_ff.energy(u[index])
122
123         return e_z, e_w, e_t
124
125     def run_forward_null(self, init_v, init_w, init_z0, init_z1):
126         """Run the forward null system"""
127
128         system = self.build_null(init_v, init_w, init_z0, init_z1, None, None)
129
130         self._system_fn = system
131
132         return system.run()
133
134     def energy_fn(self, u):
135         """Compute the energies of u"""
136
137         time, space = u.shape
138
139         e_z = np.zeros((time, 1))
140         e_w = np.zeros((time, 1))
141         e_t = np.zeros((time, 1))
142
143         for index in range(time):
144             e_z[index], e_w[index], e_t[index] = \
145                 self._system_fn.energy(u[index])
146
147         return e_z, e_w, e_t
148
149     def run_reverse_null(self, init_v, init_w, init_z0, init_z1,
150                          storage_w, storage_z):
151         """Run the reverse null system"""
152
153         system = self.build_null(init_v, init_w, init_z0, init_z1,
154                                  storage_w, storage_z)
155
156         self._system_rn = system
157
158         return system.run()
159
160     def energy_rn(self, u, target):
161         """Compute the energies of u"""
162
163         time, space = u.shape
164
165         e_z = np.zeros((time, 1))
166         e_w = np.zeros((time, 1))
167         e_t = np.zeros((time, 1))
168
169         for index in range(time):
170             e_z[index], e_w[index], e_t[index] = \
171                 self._system_rn.energy(u[index] - target)
172
173         return e_z, e_w, e_t
```

```
174
175     def run_reverse_feedback(self, num_steps, init_v, init_w, init_z0, init_z1,
176                              storage_w, storage_z):
177         """Run the reverse feedback system"""
178
179         system = self.build_feedback(num_steps,
180                                      init_v, init_w, init_z0, init_z1,
181                                      False, storage_w, storage_z)
182
183         self._system_rf = system
184
185         return system.run()
186
187     def energy_rf(self, u, target):
188         """Compute the energies of u"""
189
190         time, space = u.shape
191
192         e_z = np.zeros((time, 1))
193         e_w = np.zeros((time, 1))
194         e_t = np.zeros((time, 1))
195
196         for index in range(time):
197             e_z[index], e_w[index], e_t[index] = \
198                 self._system_rf.energy(u[index] - target)
199
200         return e_z, e_w, e_t
201
202     def split_solution(self, u):
203         """Split u into its 4 components"""
204
205         vec_num_points = 2 * self._num_points
206
207         z_sys = u[:2 * vec_num_points]
208         w_sys = u[2 * vec_num_points:]
209
210         z0 = z_sys[:vec_num_points]
211         z1 = z_sys[vec_num_points:]
212
213         v = w_sys[:vec_num_points]
214         w = w_sys[vec_num_points:]
215
216         return v, w, z0, z1
217
218     def run(self):
219         """Run the full system"""
220
221         solution_ff, e_z_ff, e_w_ff, e_t_ff, storage_z_ff, storage_w_ff =\
222             self.run_forward_feedback()
223
224         init_fn = solution_ff[-1, :]
225         init_v_fn, init_w_fn, init_z0_fn, init_z1_fn =\
226             self.split_solution(init_fn)
227
228         solution_fn, e_z_fn, e_w_fn, e_t_fn, storage_z_fn, storage_w_fn = \
229             self.run_forward_null(init_v_fn, init_w_fn, init_z0_fn, init_z1_fn)
230
231         init_rn = solution_fn[-1, :]
232         init_v_rn, init_w_rn, init_z0_rn, init_z1_rn = \
233             self.split_solution(init_rn)
234
235         init_v_rn_new = np.negative(init_v_rn)
236         init_z1_rn_new = np.negative(init_z1_rn)
237
```

```
238         solution_rn, e_z_rn, e_w_rn, e_t_rn, storage_z_rn, storage_w_rn = \
239             self.run_reverse_null(init_v_rn_new, init_w_rn,
240                                   init_z0_rn, init_z1_rn_new,
241                                   storage_w_fn, storage_z_fn)
242 
243         init_rf = solution_rn[-1, :]
244         init_v_rf, init_w_rf, init_z0_rf, init_z1_rf = \
245             self.split_solution(init_rf)
246 
247         num_rf_steps, sys_num_points = solution_ff.shape
248 
249         solution_rf, e_z_rf, e_w_rf, e_t_rf, storage_z_rf, storage_w_rf = \
250             self.run_reverse_feedback(num_rf_steps,
251                                       init_v_rf, init_w_rf,
252                                       init_z0_rf, init_z1_rf,
253                                       storage_w_ff, storage_z_ff)
254 
255         # TODO: Process the solutions to get the correct energies (for reverse)
256 
257         target = solution_ff[0, :]
258         target_v, target_w, target_z0, target_z1 = self.split_solution(target)
259 
260         target_v_new = np.negative(target_v)
261         target_z1_new = np.negative(target_z1)
262 
263         target_new_z = np.concatenate((target_z0, target_z1_new))
264         target_new_w = np.concatenate((target_v_new, target_w))
265 
266         target_new = np.concatenate((target_new_z, target_new_w))
267 
268         e_z_rn_new, e_w_rn_new, e_t_rn_new = self.energy_rn(solution_rn,
269                                                             target_new)
270         e_z_rf_new, e_w_rf_new, e_t_rf_new = self.energy_rf(solution_rf,
271                                                             target_new)
272 
273         e_z_target = np.concatenate((e_z_ff, e_z_fn, e_z_rn_new, e_z_rf_new))
274         e_w_target = np.concatenate((e_w_ff, e_w_fn, e_w_rn_new, e_w_rf_new))
275         e_t_target = np.concatenate((e_t_ff, e_t_fn, e_t_rn_new, e_t_rf_new))
276 
277         e_z = np.concatenate((e_z_ff, e_z_fn, e_z_rn, e_z_rf))
278         e_w = np.concatenate((e_w_ff, e_w_fn, e_w_rn, e_w_rf))
279         e_t = np.concatenate((e_t_ff, e_t_fn, e_t_rn, e_t_rf))
280 
281         return e_z, e_w, e_t, e_z_target, e_w_target, e_t_target,\
282                e_z_ff, e_w_ff, e_t_ff, e_z_fn, e_w_fn, e_t_fn,\
283                e_z_rn, e_w_rn, e_t_rn, e_z_rf, e_w_rf, e_t_rf
```

A.11 reverse_run.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from source import initial
4  from source import reverse
5 
6 
7  if __name__ == '__main__':
8      print("Running the reverse system")
9 
10     use_coupling = False
11 
12     epsilon = 0.0001
13     max_steps = 100000
```

```
14
15      num_points = 11
16      delta_x = 1.0 / (num_points - 1)
17
18      delta_t = 1.0 / 10.0 * delta_x
19
20      gamma = 0.1
21      rho = gamma**2
22      T = 20.0
23
24      init_v = initial.InitV()
25      init_w = initial.InitW()
26      init_z0 = initial.InitZ0()
27      init_z1 = initial.InitZ1()
28
29      g1 = initial.G1()
30      g2 = initial.G2()
31      m = initial.M()
32
33      system = reverse.ReverseControl(num_points, delta_x, delta_t,
34                                      gamma, rho, T,
35                                      init_v, init_w, init_z0, init_z1,
36                                      g1, g2, m,
37                                      epsilon, max_steps,
38                                      use_coupling)
39
40      e_z, e_w, e_t, e_z_target, e_w_target, e_t_target, e_z_ff, e_w_ff, e_t_ff, e_z_fn, e_w_fn, e_t_f
41
42      min_z = np.amin(e_z)
43      max_z = np.amax(e_z)
44      min_w = np.amin(e_w)
45      max_w = np.amax(e_w)
46      min_t = np.amin(e_t)
47      max_t = np.amax(e_t)
48
49      print('Min energy_z: ' + str(min_z))
50      print('Max energy_z: ' + str(max_z))
51      print('Min energy_w: ' + str(min_w))
52      print('Max energy_w: ' + str(max_w))
53      print('Min energy_t: ' + str(min_t))
54      print('Max energy_t: ' + str(max_t))
55
56      lower_plot = 0.001
57      upper_plot = 0.001
58
59      fig = plt.figure()
60      full_title = 'Energy for full system reverse control'
61      if use_coupling:
62          full_title += ', with coupling'
63      plt.suptitle(full_title)
64
65      plt.subplot(1, 3, 1)
66      # plt.ylim([min_z - lower_plot, max_z + upper_plot])
67      plt.plot(e_z)
68      z_title = 'Energy for z-system reverse control'
69      if use_coupling:
70          z_title += ', with coupling'
71      plt.title(z_title)
72      plt.xlabel('timestep')
73      plt.ylabel('energy')
74
75      plt.subplot(1, 3, 2)
76      # plt.ylim([min_w - lower_plot, max_w + upper_plot])
77      plt.plot(e_w)
```

```
78      w_title = 'Energy for w-system reverse control'
79      if use_coupling:
80          w_title += ', with coupling'
81      plt.title(w_title)
82      plt.xlabel('timestep')
83      plt.ylabel('energy')
84
85      plt.subplot(1, 3, 3)
86      # plt.ylim([min_t - lower_plot, max_t + upper_plot])
87      plt.plot(e_t)
88      t_title = 'Energy for t-system reverse control'
89      if use_coupling:
90          t_title += ', with coupling'
91      plt.title(t_title)
92      plt.xlabel('timestep')
93      plt.ylabel('energy')
94
95      plt.show(block=False)
96
97      fig = plt.figure()
98      full_title = 'Energy for full system reverse control, target'
99      if use_coupling:
100         full_title += ', with coupling'
101     plt.suptitle(full_title)
102
103     plt.subplot(1, 3, 1)
104     # plt.ylim([min_z - lower_plot, max_z + upper_plot])
105     plt.plot(e_z_target)
106     z_title = 'Energy for z-system reverse control, target'
107     if use_coupling:
108         z_title += ', with coupling'
109     plt.title(z_title)
110     plt.xlabel('timestep')
111     plt.ylabel('energy')
112
113     plt.subplot(1, 3, 2)
114     # plt.ylim([min_w - lower_plot, max_w + upper_plot])
115     plt.plot(e_w_target)
116     w_title = 'Energy for w-system reverse control, target'
117     if use_coupling:
118         w_title += ', with coupling'
119     plt.title(w_title)
120     plt.xlabel('timestep')
121     plt.ylabel('energy')
122
123     plt.subplot(1, 3, 3)
124     # plt.ylim([min_t - lower_plot, max_t + upper_plot])
125     plt.plot(e_t_target)
126     t_title = 'Energy for t-system reverse control, target'
127     if use_coupling:
128         t_title += ', with coupling'
129     plt.title(t_title)
130     plt.xlabel('timestep')
131     plt.ylabel('energy')
132
133     plt.show(block=False)
134
135     fig = plt.figure()
136     full_title = 'Energy for full system reverse control, forward feedback'
137     if use_coupling:
138         full_title += ', with coupling'
139     plt.suptitle(full_title)
140
141     plt.subplot(1, 3, 1)
```

```
142     # plt.ylim([min_z - lower_plot, max_z + upper_plot])
143     plt.plot(e_z_ff)
144     z_title = 'Energy for z-system reverse control, forward feedback'
145     if use_coupling:
146         z_title += ', with coupling'
147     plt.title(z_title)
148     plt.xlabel('timestep')
149     plt.ylabel('energy')
150
151     plt.subplot(1, 3, 2)
152     # plt.ylim([min_w - lower_plot, max_w + upper_plot])
153     plt.plot(e_w_ff)
154     w_title = 'Energy for w-system reverse control, forward feedback'
155     if use_coupling:
156         w_title += ', with coupling'
157     plt.title(w_title)
158     plt.xlabel('timestep')
159     plt.ylabel('energy')
160
161     plt.subplot(1, 3, 3)
162     # plt.ylim([min_t - lower_plot, max_t + upper_plot])
163     plt.plot(e_t_ff)
164     t_title = 'Energy for t-system reverse control, forward feedback'
165     if use_coupling:
166         t_title += ', with coupling'
167     plt.title(t_title)
168     plt.xlabel('timestep')
169     plt.ylabel('energy')
170
171     plt.show(block=False)
172
173     fig = plt.figure()
174     full_title = 'Energy for full system reverse control, forward null'
175     if use_coupling:
176         full_title += ', with coupling'
177     plt.suptitle(full_title)
178
179     plt.subplot(1, 3, 1)
180     # plt.ylim([min_z - lower_plot, max_z + upper_plot])
181     plt.plot(e_z_fn)
182     z_title = 'Energy for z-system reverse control, forward null'
183     if use_coupling:
184         z_title += ', with coupling'
185     plt.title(z_title)
186     plt.xlabel('timestep')
187     plt.ylabel('energy')
188
189     plt.subplot(1, 3, 2)
190     # plt.ylim([min_w - lower_plot, max_w + upper_plot])
191     plt.plot(e_w_fn)
192     w_title = 'Energy for w-system reverse control, forward null'
193     if use_coupling:
194         w_title += ', with coupling'
195     plt.title(w_title)
196     plt.xlabel('timestep')
197     plt.ylabel('energy')
198
199     plt.subplot(1, 3, 3)
200     # plt.ylim([min_t - lower_plot, max_t + upper_plot])
201     plt.plot(e_t_fn)
202     t_title = 'Energy for t-system reverse control, forward null'
203     if use_coupling:
204         t_title += ', with coupling'
205     plt.title(t_title)
```

```
206     plt.xlabel('timestep')
207     plt.ylabel('energy')
208
209     plt.show(block=False)
210
211     fig = plt.figure()
212     full_title = 'Energy for full system reverse control, reverse null'
213     if use_coupling:
214         full_title += ', with coupling'
215     plt.suptitle(full_title)
216
217     plt.subplot(1, 3, 1)
218     # plt.ylim([min_z - lower_plot, max_z + upper_plot])
219     plt.plot(e_z_rn)
220     z_title = 'Energy for z-system reverse control, reverse null'
221     if use_coupling:
222         z_title += ', with coupling'
223     plt.title(z_title)
224     plt.xlabel('timestep')
225     plt.ylabel('energy')
226
227     plt.subplot(1, 3, 2)
228     # plt.ylim([min_w - lower_plot, max_w + upper_plot])
229     plt.plot(e_w_rn)
230     w_title = 'Energy for w-system reverse control, reverse null'
231     if use_coupling:
232         w_title += ', with coupling'
233     plt.title(w_title)
234     plt.xlabel('timestep')
235     plt.ylabel('energy')
236
237     plt.subplot(1, 3, 3)
238     # plt.ylim([min_t - lower_plot, max_t + upper_plot])
239     plt.plot(e_t_rn)
240     t_title = 'Energy for t-system reverse control, reverse null'
241     if use_coupling:
242         t_title += ', with coupling'
243     plt.title(t_title)
244     plt.xlabel('timestep')
245     plt.ylabel('energy')
246
247     plt.show(block=False)
248
249     fig = plt.figure()
250     full_title = 'Energy for full system reverse control, reverse feedback'
251     if use_coupling:
252         full_title += ', with coupling'
253     plt.suptitle(full_title)
254
255     plt.subplot(1, 3, 1)
256     # plt.ylim([min_z - lower_plot, max_z + upper_plot])
257     plt.plot(e_z_rf)
258     z_title = 'Energy for z-system reverse control, reverse feedback'
259     if use_coupling:
260         z_title += ', with coupling'
261     plt.title(z_title)
262     plt.xlabel('timestep')
263     plt.ylabel('energy')
264
265     plt.subplot(1, 3, 2)
266     # plt.ylim([min_w - lower_plot, max_w + upper_plot])
267     plt.plot(e_w_rf)
268     w_title = 'Energy for w-system reverse control, reverse feedback'
269     if use_coupling:
```

```
270             w_title += ', with coupling'
271         plt.title(w_title)
272         plt.xlabel('timestep')
273         plt.ylabel('energy')
274 
275         plt.subplot(1, 3, 3)
276         # plt.ylim([min_t - lower_plot, max_t + upper_plot])
277         plt.plot(e_t_rf)
278         t_title = 'Energy for t-system reverse control, reverse feedback'
279         if use_coupling:
280             t_title += ', with coupling'
281         plt.title(t_title)
282         plt.xlabel('timestep')
283         plt.ylabel('energy')
284 
285         plt.show(block=False)
```

Bibliography

[AL03] George Avalos and Irena Lasiecka. Mechanical and thermal null controllability of thermoelastic plates and singularity of the associated minimal energy function. *Control Cybernet.*, 32(3):473, 2003.

[AL04] George Avalos and Irena Lasiecka. The null controllability of thermoelastic plates and singularity of the associated minimal energy function. *J. Math. Anal. Appl.*, 294(1):34–61, 2004.

[AL05] George Avalos and Irena Lasiecka. Asymptotic rates of blowup for the minimal energy function for the null controllability of thermoelastic plates: the free case. In *Control theory of partial differential equations*, volume 242 of *Lect. Notes Pure Appl. Math.*, pages 1–27. Chapman & Hall/CRC, Boca Raton, FL, 2005.

[AS05] Andrey A. Agrachev and Andrey V. Sarychev. Navier-Stokes equations: controllability by means of low modes forcing. *J. Math. Fluid Mech.*, 7(1):108–152, 2005.

[AS06] Andrey A. Agrachev and Andrey V. Sarychev. Controllability of 2D Euler and Navier-Stokes equations by degenerate forcing. *Comm. Math. Phys.*, 265(3):673–697, 2006.

[Ava00] George Avalos. Exact controllability of a thermoelastic system with control in the thermal component only. *Differential Integral Equations*, 13(4-6):613–630, 2000.

[Ava06] George Avalos. Null controllability of von Kármán thermoelastic plates under the clamped or free mechanical boundary conditions. *J. Math. Anal. Appl.*, 318(2):410–432, 2006.

[Bar03] Viorel Barbu. On local controllability of Navier-Stokes equations. *Adv. Differential Equations*, 8(12):1481–1498, 2003.

[Bar10] Viorel Barbu. *Nonlinear differential equations of monotone types in Banach spaces*. Springer Monographs in Mathematics. Springer, New York, 2010.

[BD87] K. Balachandran and J. P. Dauer. Controllability of nonlinear systems via fixed-point theorems. *J. Optim. Theory Appl.*, 53(3):345–352, 1987.

[CEL02] Igor Chueshov, Matthias Eller, and Irena Lasiecka. On the attractor for a semilinear wave equation with critical exponent and nonlinear boundary dissipation. *Comm. Partial Differential Equations*, 27(9-10):1901–1951, 2002.

[Cha09] Marianne Chapouly. On the global null controllability of a Navier-Stokes system with Navier slip boundary conditions. *J. Differential Equations*, 247(7):2094–2123, 2009.

[Cor07] Jean-Michel Coron. *Control and nonlinearity*, volume 136 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2007.

[CT09] Nicolae Cîndea and Marius Tucsnak. Local exact controllability for Berger plate equation. *Math. Control Signals Systems*, 21(2):93–110, 2009.

[DR77] Szymon Dolecki and David L. Russell. A general theory of observation and control. *SIAM J. Control Optimization*, 15(2):185–220, 1977.

[DSMD11] J.A. Dunnmon, S.C. Stanton, B.P. Mann, and E.H. Dowell. Power extraction from aeroelastic limit cycle oscillations. *Journal of Fluids and Structures*, 27(8):1182 – 1198, 2011.

[Ell07] Matthias Eller. Continuous observability for the anisotropic Maxwell system. *Appl. Math. Optim.*, 55(2):185–201, 2007.

[EM02] Matthias Eller and J. E. Masters. Exact boundary controllability of electromagnetic fields in a general region. *Appl. Math. Optim.*, 45(1):99–123, 2002.

[ET15] Matthias Eller and Daniel Toundykov. Semiglobal exact controllability of nonlinear plates. *SIAM J. Control Optim.*, 53:2480–2513, 2015.

[FC99] Enrique Fernández-Cara. On the approximate and null controllability of the Navier-Stokes equations. *SIAM Rev.*, 41(2):269–277 (electronic), 1999.

[flu57] Flutter of rectangular simply supported panels at high supersonic speeds. *Journal of the Aeronautical Sciences*, 24(8):563–573, 2018/04/23 1957.

[Fou] The Python Software Foundation. Python language reference, ver. 3.6.0.

[Fur00] Andrei V. Fursikov. *Optimal control of distributed systems. Theory and applications*, volume 187 of *Translations of Mathematical Monographs.*

American Mathematical Society, Providence, RI, 2000. Translated from the 1999 Russian original by Tamara Rozhkovskaya.

[Fur06] Andrei V. Fursikov. Exact controllability and feedback stabilization from a boundary for the Navier-Stokes equations. In *Control of fluid flow*, volume 330 of *Lecture Notes in Control and Inform. Sci.*, pages 173–188. Springer, Berlin, 2006.

[GIP12] Sergio Guerrero, O. Yu. Imanuvilov, and J.-P. Puel. A result concerning the global approximate controllability of the Navier-Stokes system in dimension 3. *J. Math. Pures Appl. (9)*, 98(6):689–709, 2012.

[GLH08] Roland Glowinski, Jacques-Louis Lions, and Jiwen He. *Exact and approximate controllability for distributed parameter systems: A numerical approach*, volume 117. Cambridge University Press, Cambridge, 2008.

[GU06] Daniel Green and William G. Unruh. The failure of the tacoma bridge: A physical model. *American Journal of Physics*, 74(8):706–716, 2006.

[HTW17] J. Howell, D. Toundykov, and J. T. Webster. A Cantilevered Extensible Beam in Axial Flow: Semigroup Well-posedness and Post-flutter Regimes. *ArXiv e-prints*, July 2017.

[Ima01] Oleg Yu. Imanuvilov. Remarks on exact controllability for the Navier-Stokes equations. *ESAIM Control Optim. Calc. Var.*, 6:39–72 (electronic), 2001.

[Jet] JetBrains. Pycharm ver. 2018.1.

[JOP+] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed ¡today¿].

[Kap94] B. V. Kapitonov. Stabilization and exact boundary controllability for Maxwell's equations. *SIAM J. Control Optim.*, 32(2):408–420, 1994.

[Kes89] S. Kesavan. *Topics in functional analysis and applications*. John Wiley & Sons Inc., New York, 1989.

[Kom94] Vilmos Komornik. *Exact controllability and stabilization*. Masson, Paris, 1994.

[Lag89] John E. Lagnese. Exact boundary controllability of Maxwell's equations in a general region. *SIAM J. Control Optim.*, 27(2):374–388, 1989.

[Lag90] John E. Lagnese. Local controllability of dynamic von Kármán plates. *Control Cybernet.*, 19(3-4):155–168 (1991), 1990. Optimal design and control of structures (Jabłonna, 1990).

[Lag91] John E. Lagnese. Boundary controllability of nonlinear beams to bounded states. *Proc. Roy. Soc. Edinburgh Sect. A*, 119(1-2):63–72, 1991.

[Las02] Irena Lasiecka. *Mathematical control theory of coupled PDEs*, volume 75. Society for Industrial and Applied Mathematics, SIAM; Philadelphia, PA, 2002.

[Lei05] Hugo Leiva. Exact controllability of the suspension bridge model proposed by Lazer and McKenna. *J. Math. Anal. Appl.*, 309(2):404–419, 2005.

[Li10] Tatsien Li. *Controllability and observability for quasilinear hyperbolic systems*, volume 3 of *AIMS Series on Applied Mathematics*. American Institute of Mathematical Sciences (AIMS), Springfield, MO, 2010.

[Liu98] Weijiu Liu. Local boundary controllability for the semilinear plate equation. *Comm. Partial Differential Equations*, 23(1-2):201–221, 1998.

[LL88] J. Lagnese and J.-L. Lions. *Modelling analysis and control of thin plates*, volume 6 of *Recherches en Mathématiques Appliquées [Research in Applied Mathematics]*. Masson, Paris, 1988.

[LL91] J. E. Lagnese and G. Leugering. Uniform stabilization of a nonlinear beam by nonlinear boundary feedback. *J. Differential Equations*, 91(2):355–388, 1991.

[LR99] Kangsheng Liu and David L. Russell. New meaning of exact controllability of linear systems in Hilbert spaces. In *Control of distributed parameter and stochastic systems (Hangzhou, 1998)*, pages 103–110. Kluwer Acad. Publ., Boston, MA, 1999.

[LS12] Günter R. Leugering and E. J. P. Georg Schmidt. On exact controllability of networks of nonlinear elastic strings in 3-dimensional space. *Chin. Ann. Math. Ser. B*, 33(1):33–60, 2012.

[LT81] Irena Lasiecka and Roberto Triggiani. A cosine operator approach to modeling $L_2(0,\, T;\; L_2(\Gamma))$—boundary input hyperbolic equations. *Appl. Math. Optim.*, 7(1):35–93, 1981.

[LT91] Irena Lasiecka and Roberto Triggiani. Exact controllability of semilinear abstract systems with application to waves and plates boundary control problems. *Appl. Math. Optim.*, 23(2):109–154, 1991.

[LT98] Irena Lasiecka and Roberto Triggiani. Exact null controllability of structurally damped and thermo-elastic parabolic models. *Atti Accad. Naz. Lincei Cl. Sci. Fis. Mat. Natur. Rend. Lincei (9) Mat. Appl.*, 9(1):43–69, 1998.

[LT00a] Irena Lasiecka and Roberto Triggiani. *Control theory for partial differential equations: continuous and approximation theories. I*, volume 74 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2000. Abstract parabolic systems.

[LT00b] Irena Lasiecka and Roberto Triggiani. *Control theory for partial differential equations: continuous and approximation theories. II*, volume 75. Cambridge University Press, Cambridge, 2000. Abstract hyperbolic-like systems over a finite time horizon.

[LT05] Irena Lasiecka and Roberto Triggiani. Global exact controllability of semilinear wave equations by a double compactness/uniqueness argument. *Discrete Contin. Dyn. Syst.*, (suppl.):556–565, 2005.

[LTZ04a] Irena Lasiecka, Roberto Triggiani, and X. Zhang. Global uniqueness, observability and stabilization of nonconservative Schrödinger equations via pointwise Carleman estimates. I. $H^1(\Omega)$-estimates. *J. Inverse Ill-Posed Probl.*, 12(1):43–123, 2004.

[LTZ04b] Irena Lasiecka, Roberto Triggiani, and X. Zhang. Global uniqueness, observability and stabilization of nonconservative Schrödinger equations via pointwise Carleman estimates. II. $L_2(\Omega)$-estimates. *J. Inverse Ill-Posed Probl.*, 12(2):183–231, 2004.

[LZ00] Liangyu Li and Xu Zhang. Exact controllability for semilinear wave equations. *J. Math. Anal. Appl.*, 250(2):589–589–597, 2000.

[Max68] James Clerk Maxwell. I. on governors. *Proceedings of the Royal Society of London*, 16:270–283, 1868.

[May70] Otto Mayr. *The Origins of Feedback Control*. MIT Press, Cambridge, MA, USA, 1970.

[Nas05] Maria Grazia Naso. Controllability to trajectories for semilinear thermoelastic plates. *Discrete Contin. Dyn. Syst.*, (suppl.):672–681, 2005.

[Phu00] Kim Dang Phung. Contrôle et stabilisation d'ondes électromagnétiques. *ESAIM Control Optim. Calc. Var.*, 5:87–137 (electronic), 2000.

[Rus73] David L. Russell. A unified boundary controllability theory for hyperbolic and parabolic partial differential equations. *Studies in Appl. Math.*, 52:189–211, 1973.

[Rus78] David L. Russell. Controllability and stabilizability theory for linear partial differential equations: recent progress and open questions. *SIAM Rev.*, 20(4):639–739, 1978.

[Rus93] David L. Russell. A general framework for the study of indirect damping mechanisms in elastic systems. *J. Math. Anal. Appl.*, 173(2):339–358, 1993.

[Sch02] E. J. P. Georg Schmidt. On a non-linear wave equation and the control of an elastic string from one equilibrium location to another. *J. Math. Anal. Appl.*, 272(2):536–554, 2002.

[Sho97] R. E. Showalter. *Monotone operators in Banach space and nonlinear partial differential equations*, volume 49. American Mathematical Society, Providence, RI, 1997.

[Sho04] P. O. Shorygin. Approximate controllability of the Navier-Stokes system specified in an unbounded domain. *Vestnik Moskov. Univ. Ser. I Mat. Mekh.*, (1):52–55, 63, 2004.

[Šol06] Pavel Šolín. *Partial differential equations and the finite element method*. Pure and Applied Mathematics (New York). Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, 2006.

[TGD15] Deman Tang, S. Chad Gibbs, and Earl H. Dowell. Nonlinear aeroelastic analysis with inextensible plate theory including correlation with experiment. *AIAA Journal*, 53(5):1299–1308, 2018/04/23 2015.

[Tri] Roberto Triggiani. Lecture notes, private communication.

[Tri03] Roberto Triggiani. Optimal estimates of norms of fast controls in exact null controllability of two non-classical abstract parabolic systems. *Adv. Differential Equations*, 8(2):189–229, 2003.

[Tri05] Roberto Triggiani. *Global exact controllability on $H^1_{\Gamma_0}(\Omega)\times L_2(\Omega)$ of semilinear wave equations with Neumann $L_2(0,T;L_2(\Gamma_1))$-boundary control*, volume 242 of *Control theory of partial differential equations; Lect. Notes Pure Appl. Math.*, pages 273–336. Chapman & Hall/CRC, Boca Raton, FL, 2005.

[Tri07] Roberto Triggiani. The role of an $L_2(\Omega)$-energy estimate in the theories of uniform stabilization and exact controllability for Schrödinger equations with Neumann boundary control. *Bol. Soc. Parana. Mat. (3)*, 25(1-2):109–138, 2007.

[Tri08] Roberto Triggiani. Exact controllability in $L_2(\Omega)$ of the Schrödinger equation in a Riemannian manifold with $L_2(\Sigma_1)$-Neumann boundary control. In *Functional analysis and evolution equations*, pages 613–636. Birkhäuser, Basel, 2008.

[Tri10] Roberto Triggiani. Nonlinear exact controllability and nonlinear stabilization of hyperbolic or hyperbolic-like evolution equations. *Mat. Contemp.*, 38:1–192, 2010.

[TX07] Roberto Triggiani and Xiangjin Xu. Pointwise Carleman estimates, global uniqueness, observability, and stabilization for Schrödinger equations on Riemannian manifolds at the $H^1(\Omega)$-level. In *Control methods in PDE-dynamical systems*, volume 426 of *Contemp. Math.*, pages 339–404. Amer. Math. Soc., Providence, RI, 2007.

[Yao11] Peng-Fei Yao. *Modeling and control in vibrational and structural dynamics*. Chapman & Hall/CRC Applied Mathematics and Nonlinear Science Series. CRC Press, Boca Raton, FL, 2011. A differential geometric approach.

[Zha00] Xu Zhang. Exact internal controllability of Maxwell's equations. *Appl. Math. Optim.*, 41(2):155–170, 2000.

[Zha01] Xu Zhang. Exact controllability of semilinear plate equations. *Asymptot. Anal.*, 27(2):95–125, 2001.

[Zho97] Qi Zhou. Exact internal controllability of Maxwell's equations. *Japan J. Indust. Appl. Math.*, 14(2):245–256, 1997.