The Emulated Ensemble

Real-Time Simulation of Musical Instruments using Finite-Difference Time-Domain Methods

Ph.D. Dissertation

Silvin Willemsen

Dissertation submitted May 24, 2021

Thesis submitted:	May 24, 2021
PhD Supervisor:	Prof. Stefania Serafin Aalborg University
PhD Committee:	Prof. X, Y University Prof. X, Y University Prof. X, Y University
PhD Series:	Technical Faculty of IT and Design, Aalborg University
Department:	Department of Architecture, Design and Media Technology

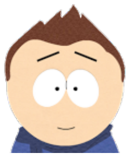
ISSN: xxxx-xxxx
ISBN: xxx-xx-xxxx-xxx-x

Published by:
Aalborg University Press
Skjernvej 4A, 2nd floor
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2021

Curriculum Vitae

Silvin Willemsen



Here is the CV text.

Acknowledgements

I would like to thank my mom..

List of Publications

Listed below are the publications that I (co)authored during the Ph.D. project. These are grouped by: the main publications, which are also included in Part IX, papers where I had a supervisory role, and finally, miscellaneous publications.

Main Publications

[A] S. Willemsen, N. Andersson, S. Serafin, and S. Bilbao, “Real-time control of large-scale modular physical models using the sensel morph,” in *Proceedings of the 16th Sound and Music Computing (SMC) Conference*, 2019, pp. 275–280.

[B] S. Willemsen, S. Bilbao, N. Andersson, and S. Serafin, “Physical models and real-time control with the sensel morph,” in *Proceedings of the 16th Sound and Music Computing (SMC) Conference*, 2019, pp. 95–96.

[C] S. Willemsen, S. Bilbao, and S. Serafin, “Real-time implementation of an elasto-plastic friction model applied to stiff strings using finite difference schemes,” in *Proceedings of the 22nd International Conference on Digital Audio Effects (DAFx-19)*, 2019, pp. 40–46.

[D] S. Willemsen, S. Serafin, S. Bilbao, and M. Ducceschi, “Real-time implementation of a physical model of the tromba marina,” in *Proceedings of the 17th Sound and Music Computing (SMC) Conference*, 2020, pp. 161–168.

[E] S. Willemsen, R. Paisa, and S. Serafin, “Resurrecting the tromba marina: A bowed virtual reality instrument using haptic feedback and accurate physical modelling,” in *Proceedings of the 17th Sound and Music Computing (SMC) Conference*, 2020, pp. 300–307.

[F] S. Willemsen, A.-S. Horvath, and M. Nascimben, “Digidrum: A haptic-based virtual reality musical instrument and a case study,” in *Proceedings of the 17th Sound and Music Computing (SMC) Conference*, 2020, pp. 292–299.

[G] S. Willemsen, S. Bilbao, M. Ducceschi, and S. Serafin, "Dynamic grids for finite-difference schemes in musical instrument simulations," in *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx2020in21)*, 2021.

[H] ——, "A physical model of the trombone using dynamic grids for finite-difference schemes," in *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx2020in21)*, 2021.

Publications with a Supervisory Role

[S1] R. S. Alecu, S. Serafin, S. Willemsen, E. Parravicini, and S. Lucato, "Embouchure interaction model for brass instruments," in *Proceedings of the 17th Sound and Music Computing (SMC) Conference*, 2020, pp. 153–160.

[S2] T. Lasickas, S. Willemsen, and S. Serafin, "Real-time implementation of the shamisen using finite difference schemes," in *Proceedings of the 18th Sound and Music Computing (SMC) Conference*, 2021.

[S3] M. G. Onofrei, S. Willemsen, and S. Serafin, "Implementing complex physical models in real-time using partitioned convolution: An adjustable spring reverb," in *Proceedings of the 18th Sound and Music Computing (SMC) Conference*, 2021.

[S4] ——, "Real-time implementation of an elasto-plastic friction drum using finite difference schemes," in *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx2020in21)*, 2021.

Miscellaneous Publications

[M1] J. M. Hjerrild, S. Willemsen, and M. G. Christensen, "Physical models for fast estimation of guitar string, fret and plucking position," *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pp. 155–159, 2019.

[M2] K. Prawda, S. Willemsen, S. Serafin, and V. Välimäki, "Flexible real-time reverberation synthesis with accurate parameter control," in *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx-20)*, 2020.

[M3] M. Ducceschi, S. Bilbao, S. Willemsen, and S. Serafin, "Linearly-implicit schemes for collisions in musical acoustics based on energy quadratisation," *Journal of the Acoustical Society of America (JASA)*, 2021.

Abstract

English abstract

Resumé

Danish Abstract

Contents

exactly as in [3]

FULL DOC SWEEP: check capitalisation of headings throughout document

FULL DOC SWEEP: check hyphen in titles

Todo list

Preface

Starting this Ph.D. project, I did not have a background in mathematics, physics or computer science, which were three equally crucial components in creating the result of this project. After the initial steep learning curve of notation and terminology, I was surprised to find that the methods used for physical modelling are actually quite straightforward!

Of course it should take a bit of time to learn these things, but

Many concepts that seemed impossible at the beginning

I feel that the literature lacks a lot of the intuition needed for readers without a background in any of these topics. Rather, much of the literature I came across assumes that the reader has a degree in at least one of the aforementioned topics. This is why I decided to write this work a bit more pedagogical than what could be expected.

I believe that anyone with some basic skills in mathematics and programming is able to create a simulation based on physics within a short amount of time, given the right tools, which I hope that this dissertation could be.

The knowledge dissemination of this dissertation is thus not only limited to the research done and publications made over the course of the project, but also its pedagogical nature hopefully allowing future (or past) students to benefit from.

As with a musical instrument itself, a low entry level, a gentle learning curve along with a high virtuosity level is desired. Take a piano, for instance. Most will be able to learn a simple melody — such as "Frère Jacques" — in minutes, but to become virtuous requires years of practice.

This is the way I wanted to write this dissertation: easy to understand the basic concepts, but many different aspects touched upon to allow for virtuosity. Hopefully by the end, the reader will at least grasp some highly complex concepts in the fields of mathematics, physics and computer science (which will hopefully take less time than it takes to become virtuous at playing the piano).

Some basic calculus knowledge is assumed.

I wanted to show my learning process and (hopefully) explain topics such as *Energy Analysis*, *Stability Analysis*, etc. in a way that others lacking the same knowledge will be able to understand.

Make physical modelling more accessible to the non-physicist.

Interested in physically impossible manipulations of now-virtual instruments.

Silvin Willemsen
Aalborg University, May 24, 2021

Part I

Introduction

Chapter 1

Physical Modelling of Musical Instruments

The history of physical modelling of musical instruments

Exciter-resonator approach.

The time-evolution of dynamic systems can be conveniently described by differential equations. Examples of a dynamic systems are a guitar string, a drum-membrane, or a concert hall; three very different concepts, but all based on the same types of equations of motion.

Though these equations are very powerful, only few have a closed-form solution. What this means is that in order for them to be implemented, they need to be approximated. There exist different approximation techniques to do this

1.1 Physical Modelling Techniques

- Modal Synthesis
- Finite-difference Time-domain methods
- Finite Element Methods
- Digital waveguides
- Mass-spring systems
- Functional transformation method
- State-space
- Wave-domain

- Energy-based

Advantages of finite-difference methods

Using FDTD methods can be quite computationally heavy. Moore's law [15]

1.2 Real-Time Implementation

Although many techniques to digitally simulate musical instruments exist proving that we have only recently reached the computing power in personal computers to make real-time playability of these models an option. The biggest challenge in real-time audio applications as opposed to those only involving graphics, is that the sample rate is extremely high. As Nyquist's sampling theory tells us, a sampling rate of at least 40 kHz is necessary to produce frequencies up to the human hearing limit of 20 kHz **[Nyquist]**. Visuals

Real-time: no noticeable latency

exactly as in [3]

1.3 Why?

1.3.1 Audio plugins

Samples, or recordings, of real instruments are static and unable to adapt to changes in performance. Moreover, capturing the the entire interaction space of an instrument is nearly impossible. Imagine recording a violin with every single combination of bowing force, velocity, position, duration and other aspects such as vibrato, pizzicato. Even if a complete sample library could be created, this would contain an immense amount of data.

Samples vs. Physical Modelling:

Trade off between storage and speed

Using musical instrument simulations, on the other hand, allows the sound to be generated on the spot based on physical parameters that the user can interact with.

1.3.2 Resurrect old or rare instruments

Even popular instruments require maintenance and might need to be replaced after years of usage.

1.3.3 Go beyond what is physically possible

1.4 Thesis Objectives and Main Contributions

Over the past few decades, much work has been done on the accurate modelling of physical phenomena. In the field of sound and musical instruments..

From [9] to [5]

The main objective of this thesis is to implement existing physical models in real time using FDTD methods. Many of the physical models and methods presented in this thesis are taken from the literature and are thus not novel.

Secondly, to combine the existing physical models to get complete instruments and be able to control them in real time.

As FDTD methods are quite rigid, changing parameters on the fly, i.e., while the instrument simulation is running, is a challenge. Other techniques, such as modal synthesis, are much more suitable for this, but come with the drawbacks mentioned in Section 1.1. Therefore, a novel method was devised to smoothly change parameters over time, introducing this to FDTD methods.

Put this work into perspective of the literature (higher level)

1.5 Thesis Outline

Introduction to finite-difference methods and analysis techniques

Models used over the course of the project divided into resonators in part II, exciters in part III and the interactions between them in IV.

Focus on real-time implementation and control of the models in part VI

- Physical models
  - Resonators
  - Exciters
  - Interactions
- Dynamic Grids
- Real-Time Implementation and Control
- Complete instruments
  - Large-scale physical models
  - Tromba Marina
  - Trombone

Notes

- Think about how to define real-time.
- Create an intuition for different parts of the equation
- Talk about input and output locations and how that affects frequency content (modes).

One over number $\rightarrow$ reciprocal of number

Example: When the waveform consists entirely of harmonically related frequencies, it will be periodic, with a period equal to the reciprocal of the fundamental frequency (from An Introduction to the Mathematics of Digital Signal Processing Pt 2 by F. R. Moore)

Chapter 2

Introduction to FDTD Methods

"Since Newton, mankind has come to realize that the laws of physics are always expressed in the language of differential equations."
- Steven Strogatz

This chapter introduces some important concepts needed to understand the physical models presented later on in this document. By means of a simple mass-spring system and the 1D wave equation, the notation and terminology used throughout this document will be explained. Before diving into the mathematics, let us go over some useful terminology.

2.1 Differential Equations

Differential equations are used to describe the motion of dynamic systems including vibrations in musical instruments. In this work, these equations are used, among others, to describe the movement of a string, an instrument body and the air pressure in an acoustic tube.

A characteristic feature of these equations is that, rather than an absolute value or *state* of a system, the time derivative of its state – its velocity – or the second time derivative – its acceleration – is described. From this, the absolute state of the system can then be computed. This state is usually described by the variable u which is (nearly) always a function of time, i.e., $u = u(t)$. If the system is distributed in space, u also becomes a function of space, i.e., $u = u(x,t)$, or with two spatial dimensions, $u = u(x,y,t)$, etc. Though this work only describes systems of up to two spatial dimensions, one can easily maybe not such a relevant reference..

extend to three dimensions [12] and potentially higher-dimensional systems [16]! See Section 2.1.1 for more information on this.

If u is univariate, and only a function of time, the differential equation that describes the motion of this system is called an *ordinary differential equation* (ODE). Various ways to describe the second derivative in time of u, or the acceleration of u are

$$\begin{aligned} &\frac{d^2u}{dt^2} && \text{(Leibniz's notation)},\\ &\ddot{u} && \text{(Newton's notation)},\\ &D_t^2 u && \text{(Euler's notation)}. \end{aligned}$$

Leibniz' notation could be considered the most standard notation but is not necessarily compact. Newton's notation on the other hand allows for an ultra compact notation using a dot above the function to denote a derivative. However, this notation can only be used for univariate functions which it will be used for in this document. Finally, Euler's notation uses an operator which can be applied to a function and indicates a derivative.

If u is also a function of at least one spatial dimension, the equation of motion is a called a *partial differential equation* (PDE). The literature uses different types of notation for taking (continuous-time) partial derivatives. Applied to a state variable u these can look like

$$\begin{aligned} &\frac{\partial^2 u}{\partial t^2} && \text{(Leibniz's notation)}\\ &u_{tt} && \text{(subscript notation)}\\ &\partial_t^2 u && \text{(Euler's notation)} \end{aligned}$$

where the subscript notation could be seen as the partial derivative counterpart to Newton's notation due to its compactness. In the remainder of this document, the operator notation will be used, due to their similarity to the discrete operators (introduced shortly) and as it allows for creation of bigger operators for more compactness when working with multiple (connected) systems (see e.g. Chapter 15).

maybe not yet as this is super general still

Often-used partial derivatives and their meanings are shown below

$\partial_t^2 u$	(acceleration)	$\partial_x^2 u$	(curvature)
$\partial_t u$	(velocity)	$\partial_x u$	(slope)

2.1.1 Dimensions and Degrees of Freedom

All objects in the physical world are 3-dimensional (3D) as they have a non-zero width, length and depth. Moreover, these objects can move in these three dimensions and thus have three translational *degrees of freedom (DoF)* (the three rotational DoF are ignored here). To reduce the complexity of the model as well as computational complexity, simplifications can be made to reduce both the dimensionality of the spatial distribution of a physical object as well as that of the translational DoF.

Generally, the spatial distribution of an object can be simplified if one (or more) of the dimensions is orders of magnitude smaller than the others. A guitar string, for instance, has much greater length than its width or depth and can therefore be reduced to a one-dimensional (1D) system. If a 3D description were to be kept, the relative displacement between two locations on one cross-section along the length of the string would be taken into account. One could imagine that this displacement will always be orders of magnitude smaller than the relative displacement of two points along the string length and is thus negligible. Similarly, the thickness of a drum membrane is much smaller than its length and width and can therefore be simplified to a two-dimensional (2D) system.

The translational DoF, on the other hand, describe now many "coordinates" a state variable includes. In much of the literature on FDTD methods in the field of musical acoustics, the state variable only has one coordinate. In most string models, for example, only the transverse displacement in one polarisation is considered (see Chapter 4) and the other polarisation as well as the longitudinal motion of the string is ignored. In other words, every point along the string can only move up and down, not side-to-side and not forward and back. Although this greatly simplifies the system at hand and reduces computational complexity, this is not what happens in reality, and non-linear effects such as phantom partials and pitch glides due to tension modulation are not present in the simplified model.

check whether this needs to be per point along a system

Work has been done on strings with dual polarisation by Desvages [8] and Desvages and Bilbao [7] using FDTD methods. Models including longitudinal string vibration, where the longitudinal and transversal displacements are couples can be found in [3]. In [17], Villeneuve and Leonard present a mass-spring network where the state of every individual mass has three translational DoF. Due to these additional DoF, these networks would capture the aforementioned effects, but greatly increase the computational complexity of the models.

Although the dimensionality reduction ignoring some of the physical processes, surprisingly realistic sounding models can be made despite these simplifications. Due to computational considerations, all models used in this work thus only have 1 translational DoF.

Notation

When describing the state of a system, the spatial dimensions it is distributed over appears in the argument of the state variable, whereas the translational

DoF determines the amount of coordinates of the state variable describes. For example, the state of a 2D system, with 1 translational DoF is written as $u(x, y, t)$. A 1D system with 3 translational DoF can be written as $\mathbf{u}(x, t)$ where $\mathbf{u}$ is a vector containing the coordinates for all three translational DoF.

2.1.2 Domains

When describing physical systems using state variables, *domains* over which they are defined need to be provided. This means that for the state of a 1D system $u = u(x, t)$, ranges of definition must be given for x and t. Usually, time $t \geq 0$, meaning that the system is defined for non-negative time. For the spatial dimension, we may define a finite domain $\mathcal{D}$ over which x is defined, which is written as $x \in \mathcal{D}$. For analysis purposes, infinite domains ($\mathcal{D} = \mathbb{R} = (-\infty, \infty)$) or semi-infinite domains ($\mathcal{D} = \mathbb{R}^+ = [0, \infty)$) may be used, but for implementation purposes, a finite domain needs to be established.

2.2 Discretisation using FDTD methods

Differential equations are powerful tools that describe the motion of physical systems. Despite this, only few of these have a closed-form or analytical solution (such as the simpler ones that will be presented in this chapter). More complex systems require methods that do not perfectly solve, but rather *approximate* the solutions to these equations. Finite-difference time-domain (FDTD) are considered one of the in terms of generality and flexibility...

Unless denoted otherwise, the equations and theory used in this chapter has been taken from [3].

It is important to note that a discrete FD scheme is an *approximation* to a continuous PDE, not a sampled version of it. This means that the resulting schemes are rarely an exact solution to the original continuous equation.

These methods essentially subdivide a continuous differential equation into discrete points in time and space, a process called *discretisation*. Once an ODE or PDE is discretised using these methods it is now called a *finite-difference (FD) scheme* which approximates the original differential equation. In the following, for generality and ease of explanation, a 1D system will be used.

Figure and caption are not done yet

FULL DOC SWEEP: check capitalisation of headings throughout document

grid figure

2.2.1 Grid Functions

We start by defining a discrete *grid* over time and space which we will use to approximate our continuous equations . A system described by state $u = u(x, t)$ defined over time t and one spatial dimension x, can be discretised to a *grid*

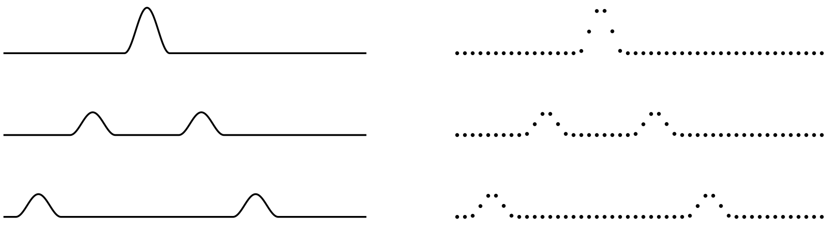


Fig. 2.1: A continuous PDE is discretised...

function u_l^n. Here, integers l and superscript n describe the spatial and temporal indices respectively and arise from the discretisation of the continuous variables x and t according to $x = lh$ and $t = nk$. The spatial step h, also called the *grid spacing* describes the distance (in m) between two neighbouring *grid points* and the temporal step k, or *time step* is the time (in s) between two consecutive temporal indices. The latter can be calculated $k = 1/f_s$ for a sample rate f_s (in Hz). In many audio applications $f_s = 44100$ Hz which will be used in this work (unless denoted otherwise).

As mentioned in Section 2.1.2, a 1D system needs to be defined over a temporal and one spatial domain. In discrete time, the temporal domain $t \geq 0$ is discretised to $n \in \mathbb{N}^0$.[1] The spatial domain $\mathcal{D}$ can be subdivided into N equal sections, or intervals, of length h (see Figure 2.2). The grid points describing the state of the system are placed between and around these intervals. The spatial range of interest then becomes $l \in \{0, \dots, N\}$ and the total number of grid points is $N+1$, one more than the number of intervals.

To summarise, for a 1D system

this might be unnecessary, but I thought that it might be nice to have this in an equation for clarity

$$u(x,t) \approx u_l^n \quad \text{with} \quad x = lh \quad \text{and} \quad t = nk,$$

$$l \in \{0, \dots, N\} \quad \text{and} \quad n \in \mathbb{N}^0.$$

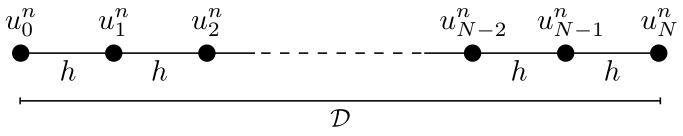


Fig. 2.2: When a 1D system $u(x,t)$ with $x \in \mathcal{D}$ is discretised to a grid function u_l^n, the spatial domain $\mathcal{D}$ is divided into N intervals of length h and spatial range of interest $l = \{0, \dots, N\}$.

[1]In this work, $\mathbb{N}^0$ is used to denote the set of non-negative integers ($\mathbb{N}^0 = 0, 1, 2, \dots$).

2.2.2 Finite-Difference Operators

Now that the state variable has a discrete counterpart, this leaves the derivatives to be discretised, or approximated. We start by introducing shift operators that can be applied to a grid function and 'shifts' its indexing, either temporally or spatially. Forward and backward shifts in time, together with the identity operation are

$$e_{t+}u_l^n = u_l^{n+1}, \quad e_{t-}u_l^n = u_l^{n-1}, \quad \text{and} \quad 1u_l^n = u_l^n. \tag{2.1}$$

Similarly, forward and backward shifts in space are

$$e_{x+}u_l^n = u_{l+1}^n, \quad \text{and} \quad e_{x-}u_l^n = u_{l-1}^n. \tag{2.2}$$

many figures for shift and FD operators

These shift operators are rarely used in isolation, though they do appear in energy analysis techniques detailed in Section 3.4. The operators do, however, form the basis of commonly used *finite-difference (FD) operators*. The first-order derivative in time can be discretised three different ways. The forward, backward and centred difference operators are

FULL DOC SWEEP: check centred instead of centered

these spacings are different in overleaf...

$$\partial_t \approxeq \begin{cases} \delta_{t+} \triangleq \frac{1}{k}\left(e_{t+} - 1\right), & (2.3a)\\ \delta_{t-} \triangleq \frac{1}{k}\left(1 - e_{t-}\right), & (2.3b)\\ \delta_{t\cdot} \triangleq \frac{1}{2k}\left(e_{t+} - e_{t-}\right), & (2.3c)\end{cases}$$

where "$\triangleq$" means "equal to by definition". These operators can then be applied to grid function u_l^n to get

$$\partial_t u \approxeq \begin{cases} \delta_{t+}u_l^n = \frac{1}{k}\left(u_l^{n+1} - u_l^n\right), & (2.4a)\\ \delta_{t-}u_l^n = \frac{1}{k}\left(u_l^n - u_l^{n-1}\right), & (2.4b)\\ \delta_{t\cdot}u_l^n = \frac{1}{2k}\left(u_l^{n+1} - u_l^{n-1}\right), & (2.4c)\end{cases}$$

and all approximate the first-order time derivative of u. Note that the centred difference has a division by $2k$ as the time difference between $n+1$ and $n-1$ is, indeed, twice the time step.

figure here visualising operators (with reference to grid figure)

Similar operators exist for a first-order derivative in space, where the forward, backward and centred difference are

$$\partial_x \approxeq \begin{cases} \delta_{x+} \triangleq \frac{1}{h}\left(e_{x+} - 1\right), & (2.5a)\\ \delta_{x-} \triangleq \frac{1}{h}\left(1 - e_{x-}\right), & (2.5b)\\ \delta_{x\cdot} \triangleq \frac{1}{2h}\left(e_{x+} - e_{x-}\right), & (2.5c)\end{cases}$$

and when applied to u_l^n are

$$\partial_x u \approxeq \begin{cases} \delta_{x+}u_l^n = \frac{1}{h}\left(u_{l+1}^n - u_l^n\right), & (2.6a)\\ \delta_{x-}u_l^n = \frac{1}{h}\left(u_l^n - u_{l-1}^n\right), & (2.6b)\\ \delta_{x\cdot}u_l^n = \frac{1}{2h}\left(u_{l+1}^n - u_{l-1}^n\right). & (2.6c)\end{cases}$$

Higher order differences can be approximated through a composition of first-order difference operators. The second-order difference in time may be approximated using

$$\partial_t^2 \approxeq \delta_{t+}\delta_{t-} = \delta_{tt} \triangleq \frac{1}{k^2}\left(e_{t+} - 2 + e_{t-}\right), \tag{2.7}$$

where "2" is the identity operator applied twice. This can similarly be done for the second-order difference in space

$$\partial_x^2 \approxeq \delta_{x+}\delta_{x-} = \delta_{xx} \triangleq \frac{1}{h^2}\left(e_{x+} - 2 + e_{x-}\right), \tag{2.8}$$

both of which can be applied to a grid function u_l^n in a similar fashion. Further information on combining operators can be found in Section 4.2.1.

Also useful are averaging operators, all of which approximate the identity operation. The temporal forward, backward and centred averaging operators are

in energy analysis, interleaved grids, etc.

$$1 \approxeq \begin{cases} \mu_{t+} \triangleq \frac{1}{2}\left(e_{t+} + 1\right), & (2.9a)\\ \mu_{t-} \triangleq \frac{1}{2}\left(1 + e_{t-}\right), & (2.9b)\\ \mu_{t\cdot} \triangleq \frac{1}{2}\left(e_{t+} + e_{t-}\right). & (2.9c)\end{cases}$$

Notice how these definitions are different than the difference operators in (2.3): the terms in the parentheses are added rather than subtracted, and rather than a division by the time step k there is a division by 2. Finally, the centred averaging operator does not have an extra division by 2 as in (2.3c). Applied to u_l^n, Eqs. (2.9) become

$$u_l^n \approxeq \begin{cases} \mu_{t+}u_l^n = \frac{1}{2}\left(u_l^{n+1} + u_l^n\right), & (2.10a)\\ \mu_{t-}u_l^n = \frac{1}{2}\left(u_l^n + u_l^{n-1}\right), & (2.10b)\\ \mu_{t\cdot}u_l^n = \frac{1}{2}\left(u_l^{n+1} + u_l^{n-1}\right). & (2.10c)\end{cases}$$

Similarly, spatial averaging operators are

$$1 \approxeq \begin{cases} \mu_{x+} \triangleq \frac{1}{2}\left(e_{x+} + 1\right), & (2.11a)\\ \mu_{x-} \triangleq \frac{1}{2}\left(1 + e_{x-}\right), & (2.11b)\\ \mu_{x\cdot} \triangleq \frac{1}{2}\left(e_{x+} + e_{x-}\right), & (2.11c)\end{cases}$$

and when applied to u_l^n

$$u_l^n \approxeq \begin{cases} \mu_{x+}u_l^n = \frac{1}{2}\left(u_{l+1}^n + u_l^n\right), & (2.12a)\\ \mu_{x-}u_l^n = \frac{1}{2}\left(u_l^n + u_{l-1}^n\right), & (2.12b)\\ \mu_{x\cdot}u_l^n = \frac{1}{2}\left(u_{l+1}^n + u_{l-1}^n\right). & (2.12c)\end{cases}$$

Finally, using forward and backward averaging operators, second-order temporal and spatial averaging operators can be created according to

$$1 \approxeq \mu_{tt} = \mu_{t+}\mu_{t-} \triangleq \frac{1}{4}\left(e_{t+} + 2 + e_{t-}\right), \tag{2.13}$$

and

$$1 \approxeq \mu_{xx} = \mu_{x+}\mu_{x-} \triangleq \frac{1}{4}\left(e_{x+} + 2 + e_{x-}\right). \tag{2.14}$$

Operators and derivatives in 2D will be discussed in Chapter 6.

Accuracy

As FDTD methods approximate continuous systems, the resulting solution is rarely 100% accurate. To determine the accuracy of the FD operators above, one can perform a Taylor series analysis. The Taylor series is an infinite sum and its expansion of a function f about a point a is defined as

$$f(x) = \sum_{n=0}^{\infty} \frac{(x-a)^n}{n!} f^{(n)}(a) \tag{2.15}$$

where superscript (n) denotes the n^{th} derivative of f with respect to x. The analysis will be performed on the temporal operators in this section, but also apply to the spatial operators presented.

Using continuous function $u = u(t)$ and following Bilbao's "slight abuse of notation" in [3], one may apply FD operators to continuous functions according to

$$\delta_{t+}u(t) = \frac{u(t+k) - u(t)}{k}\,. \tag{2.16}$$

Assuming that u is infinitely differentiable, $u(t+k)$, i.e., u at the next time step (but in continuous time), can be approximated using a Taylor series expansion of u about t according to

$$u(t+k) = u(t) + k\dot{u} + \frac{k^2}{2}\ddot{u} + \frac{k^3}{6}\dddot{u} + \mathcal{O}(k^4). \tag{2.17}$$

Here, (following Newton's notation) the dot describes a single temporal derivative and $\mathcal{O}$ includes additional terms in the expansion. The power of k in the argument of $\mathcal{O}$ describes the order of accuracy, the higher the power of k the more accurate the approximation. Equation (2.17) can be rewritten to

$$\begin{aligned} \frac{u(t+k) - u(t)}{k} &= \dot{u} + \frac{k}{2}\ddot{u} + \frac{k^2}{6}\dddot{u} + \mathcal{O}(k^3), \\ \delta_{t+}u(t) &= \dot{u} + \mathcal{O}(k), \end{aligned} \tag{2.18}$$

which says that the forward difference operator approximates the continuous first order derivative with an additional error term. As the power of k in

$\mathcal{O}$'s argument is 1, it can be concluded that the forward operator is first-order accurate. One can also observe that, as expected, the error gets smaller as the time step k gets smaller and indicates that higher sample rates result in more accurate simulations (through $k = 1/f_\text{s}$).

We can arrive at a similar result for the backward operator. Applying Eq. (2.3b) to u yields

$$\delta_{t-}u(t) = \frac{u(t) - u(t-k)}{k} \tag{2.19}$$

and performing a Taylor series expansion of u about t yields

$$\begin{aligned} u(t-k) &= u(t) + (-k)\dot{u} + \frac{(-k)^2}{2}\ddot{u} + \frac{(-k)^3}{6}\dddot{u} + \mathcal{O}(k^4), \quad (2.20)\\ \frac{u(t-k) - u(t)}{k} &= -\dot{u} + \frac{k}{2}\ddot{u} - \frac{k^2}{6}\dddot{u} + \mathcal{O}(k^3),\\ \delta_{t-}u(t) &= \dot{u} + \mathcal{O}(k). \quad (2.21)\end{aligned}$$

Notice that the sign of $\mathcal{O}$ does not matter.

Applying the centred operator in Eq. (2.3c) to u yields

$$\delta_{t\cdot}u(t) = \frac{u(t+k) - u(t-k)}{2k}, \tag{2.22}$$

indicating that to find the order of accuracy for this operator, both Eqs. (2.17) and (2.20) are needed. Subtracting these and filling in their definitions yields

$$\begin{aligned} u(t+k) - u(t-k) &= 2k\dot{u} - \frac{2k^3}{6}\dddot{u} + 2\mathcal{O}(k^5),\\ \frac{u(t+k) - u(t-k)}{2k} &= \dot{u} + \mathcal{O}(k^2),\\ \delta_{t\cdot}u(t) &= \dot{u} + \mathcal{O}(k^2), \quad (2.23)\end{aligned}$$

and shows that the centred difference operator is second-order accurate.

As a first-order derivative indicates the *slope* of a function, the differences in accuracy between the above operators can be visualised as in Figure 2.3. It can be observed that the derivative approximation of the centred operator much more closely matches the the true derivative of u at t.

Higher-order differences, such as the second-order difference in time operator in Eq. (2.7) can also be applied to $u(t)$ to get

$$\delta_{tt}u(t) = \frac{u(t+k) - 2u(t) + u(t-k)}{k^2}, \tag{2.24}$$

and be proven to be second-order accurate by adding Eqs. (2.17) and (2.20):

$$\begin{aligned} u(t+k) + u(t-k) &= 2u(t) + k^2\ddot{u} + \mathcal{O}(k^4),\\ \frac{u(t+k) - 2u(t) + u(t-k)}{k^2} &= \ddot{u} + \mathcal{O}(k^2),\\ \delta_{tt}u(t) &= \ddot{u} + \mathcal{O}(k^2). \quad (2.25)\end{aligned}$$

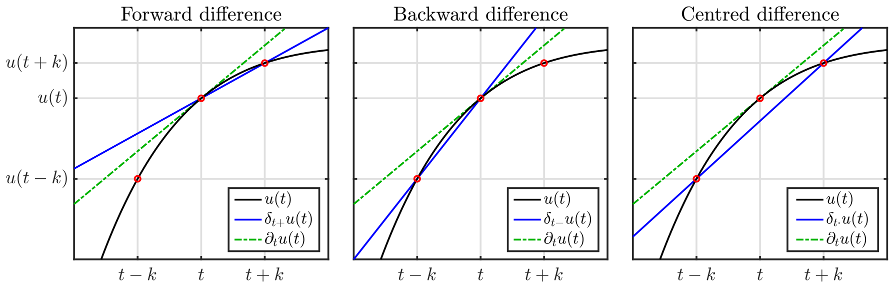


Fig. 2.3: The accuracy of the forward, backward and centred difference operators in (2.3) visualised. One can observe that the centred difference operator much more closely approximates the derivative, or the slope, of u at t than the forward and backward difference operators.

The accuracy of averaging operators can also be found and follow a similar pattern.

$$\begin{aligned} \mu_{t+}u(t) = u(t) + \mathcal{O}(k), \quad \mu_{t-}u(t) = u(t) + \mathcal{O}(k), \\ \mu_{t\cdot}u(t) = u(t) + \mathcal{O}(k), \quad \mu_{tt}u(t) = u(t) + \mathcal{O}(k^2). \end{aligned} \tag{2.26}$$

2.2.3 Identities

For working with FD schemes, either for implementation or analysis, it can be extremely useful to rewrite the operators presented above to equivalent versions of themselves. These are called identities and for future reference, some useful ones are listed below

$$\delta_{tt} = \frac{2}{k}\left(\delta_{t\cdot} - \delta_{t-}\right) \tag{2.27a}$$

$$\delta_{t\cdot} = \delta_{t+}\mu_{t-} = \delta_{t-}\mu_{t+} \tag{2.27b}$$

$$\mu_{t+} = \frac{k}{2}\delta_{t+} + 1 \tag{2.27c}$$

see whether the negative version of identity (2.27c) is also used later on

That these equalities hold can easily be proven by expanding the operators defined in Section 2.2.2. Naturally, these identities also hold for spatial operators by simply substituting the 't' subscripts for 'x'.

2.3 The Mass-Spring System

Though a complete physical modelling field on its own (see Chapter 1), mass-spring systems are also sound-generating systems themselves and lend themselves well to illustrating and explaining FDTD methods in practice. Starting with the continuous-time ODE, this section continues to discretise it to an FD scheme using the operators described in Section 2.2.2. Finally, the scheme is rewritten to an update equation that can be implemented and the output of the system is shown.

2.3.1 Continuous-time

Using dots to indicate a temporal derivative, the ODE of a simple mass-spring system is defined as

$$M\ddot{u} = -Ku, \tag{2.28}$$

where $u = u(t)$ is the distance from the equilibrium position (in m), $M > 0$ is the mass of the mass (in kg) and $K \geq 0$ is the spring constant (in N/m). In the literature [3, **?**], Eq. (2.28) is often written as

$$\ddot{u} = -\omega_0^2 u \tag{2.29}$$

with

$$\omega_0 = \sqrt{K/M}. \tag{2.30}$$

This way of writing the mass-spring ODE is more compact and more straightforward to relate to a fundamental frequency $f_0 = \omega_0/2\pi$ (in Hz).

Apart from the choices of K and M, the behaviour of the mass-spring system is determined by its *initial conditions*, being $u(0)$ and $\partial_t u(0)$, i.e., the displacement and velocity of the mass at $t = 0$. If the initial conditions are non-zero, the path that the mass follows over time is sinusoidal (see Figure 2.4), which is also why the mass-spring system is often referred to as the *simple harmonic oscillator*. The amplitude of the sinusoid is determined by the initial conditions, whereas the frequency is determined by M and K.

Intuition

The behaviour of the mass-spring system in Eq. (2.28) arises from two basic laws of physics: *Newton's second law* and *Hooke's law*.

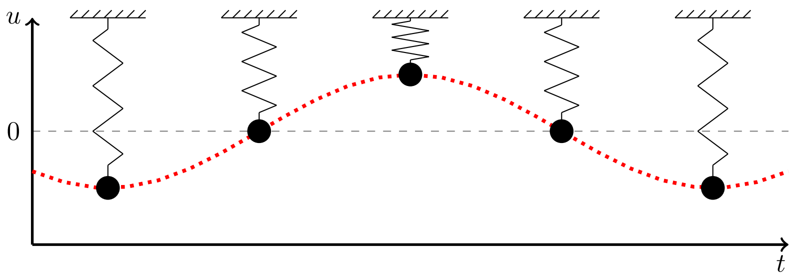


Fig. 2.4: Mass spring system over time. The system follows a harmonic (sinusoidal) motion.

Starting with Newton's second law *force equals mass times acceleration*, and relating this to the variables used in Eq. (2.28) yields an expression for force

$$F = M\ddot{u}. \tag{2.31}$$

This equation in isolation can be used to, fx., calculate the force necessary to accelerate a mass of M kg to $\ddot{u}$ m/s^2. Next, the force generated by the spring follows Hooke's law:

$$F = -Ku, \tag{2.32}$$

which simply states that the force generated by a spring with stiffness K is negatively proportional to the value of u. In other words, the further the spring is extended (from the equilibrium $u = 0$), the more force will be generated in the opposite direction. Finally, as the sole force acting on the mass is the one generated by the spring, the two expressions for the force F can be set equal to each other and yields the equation for the mass-spring system in (2.28).

The sinusoidal behaviour of the mass-spring system, or a least the fact that the mass "gets pulled back" to the equilibrium, is apparent from the minus-sign in Eq. (2.32). The frequency of the sinusoid, depends on the value of K as the "pull" happens to a higher degree for a higher spring stiffness. That the frequency of the system is also dependent on the mass M can be explained by the fact that a lighter object is more easily moved and vice versa, which is apparent from Eq. (2.31). In other words, the pull of the spring has a greater effect on the acceleration of a lighter object than a heavier one.

Finally, if $u = 0$ there is no spring force present and the acceleration remains unchanged. If the mass is not in motion, this means that it remains stationary, but if it is, the velocity is unchanged and it will continue moving with the same speed.

2.3.2 Discrete-time

The displacement of the mass is approximated using

$$u(t) \approx u^n, \tag{2.33}$$

with time $t = nk$, time step $k = 1/f_\text{s}$, sample rate f_s and temporal index and $n \in \mathbb{N}^0$. Note that the "grid function" does not have a subscript l as u is not distributed in space and is now simply called a *time series*.

Using the operators found in Section 2.2.2, Eq. (2.28) can be discretised as follows:

$$M\delta_{tt}u^n = -Ku^n, \tag{2.34}$$

which is the first appearance of a FD scheme in this work. Expanding the δ_{tt} operator yields

$$\frac{M}{k^2}\left(u^{n+1} - 2u^n + u^{n-1}\right) = -Ku^n,$$

and solving for u^{n+1} results in the following recursion or *update equation*:

$$u^{n+1} = \left(2 - \frac{Kk^2}{M}\right) u^n - u^{n-1}, \tag{2.35}$$

which can be implemented in a programming language such as `MATLAB`.

2.3.3 Implementation and Output

A simple `MATLAB` script implementing the mass-spring system described in this section is shown in Appendix C.1. The most important part of the algorithm happens in a for loop recursion, where update equation (2.35) is implemented. At the end of each loop, the system states are updated and prepared for the next iteration.

To be able to calculate the scheme at $n = 0$ (the first time index of the simulation), values must be provided for u^0 and u^{-1}. These are determined by the initial conditions mentioned in Section 2.3.1. A simple way to obtain a sinusoidal motion with an amplitude of 1, is to set the initial conditions as follows (using the backwards time difference operator for discretising the first-order time derivative):

$$u^0 = 1 \quad \text{and} \quad \delta_{t-}u^n = 0. \tag{2.36}$$

The latter equality can be solved for u^{-1} to obtain its definition:

$$\begin{aligned} &\frac{1}{k}\left(u^0 - u^{-1}\right) = 0, \\ \overset{u^0=1}{\Longleftrightarrow} \quad &1 - u^{-1} = 0, \\ &u^{-1} = 1. \end{aligned}$$

To sum up, simply setting $u^0 = u^{-1} \neq 0$ yields an oscillatory behaviour.

maybe a bit short...

The values for K and M are restricted by a stability condition which will be elaborated on in Section 3.3.

The output of the system can be obtained by listening to the displacement of the mass at the given sample rate f_s. An example of this can be found in Figure 2.5 where the frequency of oscillation $f_0 = 440$ Hz.

2.4 The 1D Wave Equation

Arguably the most important PDE in the field of physical modelling for sound synthesis is the 1D wave equation. It can be used to describe transverse vibration in an ideal string, longitudinal vibration in an ideal bar or the pressure in an acoustic tube (see Chapter 5). Although the behaviour of this equation alone does not appear in the real world as such – as no physical system is ideal – it is extremely useful as a test case and a basis for more complicated models.

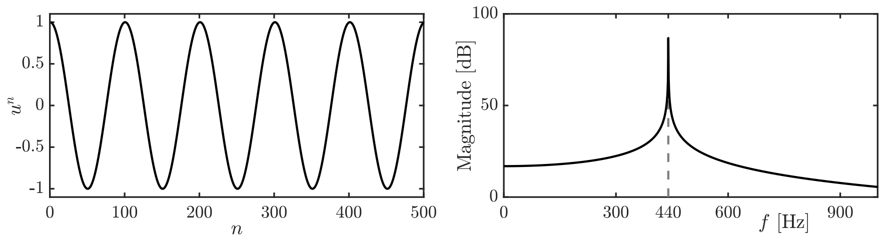


Fig. 2.5: The time-domain and frequency-domain output of a mass-spring system with $f_0 = 440$ Hz.

FULL DOC SWEEP: check hyphen in titles

2.4.1 Continuous time

The 1D wave equation is a PDE that describes the motion of a system distributed in one dimension of space. Consider the state of a 1D system $u = u(x, t)$ of length L defined for time $t \geq 0$ and $x \in \mathcal{D}$ with $D = [0, L]$. The PDE describing its motion is

$$\partial_t^2 u = c^2 \partial_x^2 u, \tag{2.37}$$

where c is the wave speed (in m/s).

Intuition

As with the mass-spring system in Section 2.3 the working of the PDE in (2.37) arises from Newton's second law, even though this connection might be less apparent.

The 1D wave equation in (2.37) states that the acceleration of $u = u(x, t)$ at location x is determined by the second-order spatial derivative of u at that same location (scaled by a constant c^2). In the case that u describes the transverse displacement of an ideal string, this second-order derivative denotes the *curvature* of this string. As c^2 is always positive, the sign (or direction) of the acceleration is fully determined by the sign of the curvature. In other words, a 'positive' curvature at location x along the ideal string yields a 'positive' or upwards acceleration at that same location.

What a 'positive' or 'negative' curvature implies is more easily seen when we take a simple function describing a parabola, $y(x) = x^2$, and take its second derivative to get $y''(x) = 2$. The answer is a positive number which means that y has a positive curvature.

So, what does this mean for the 1D wave equation? As a positive curvature implies a positive or upwards acceleration as per Eq. (2.37), u with a positive curvature at a location x will start to move upwards and vice versa. Of course, the state of a physical system such as u will rarely have a perfect parabolic shape, but the argument still applies. See Figure 2.6.

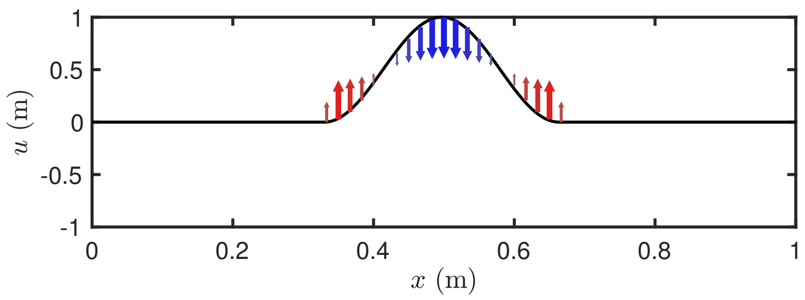


Fig. 2.6: The forces acting on the 1D wave equation due to curvature. The arrows indicate the direction and magnitude of the force, and simultaneously the acceleration as these are connected through Eq. (2.37).

different wording in caption

Boundary Conditions

When a system is distributed in space, *boundary conditions* must be determined. Consider a system of length L (in m) defined over space $x \in \mathcal{D}$ where domain $\mathcal{D} = [0, L]$. Two often-used alternatives are

$$u(0,t) = u(L,t) = 0 \quad \text{(Dirichlet, fixed)}, \tag{2.38a}$$
$$\partial_x u(0,t) = \partial_x u(L,t) = 0 \quad \text{(Neumann, free)}. \tag{2.38b}$$

The Dirichlet boundary condition says that at the end points of the system, the state is 0 at all times. The Neumann condition on the other hand, says that rather the slope of these points needs to be 0, but that the end points are free to move transversely. In the former case, incoming waves to invert after reaching the boundary whereas in the latter incoming waves are reflected un-inverted. See Figure 2.7.

If both boundaries of the 1D wave equation share the same condition, the fundamental frequency of the simulation can be calculated using

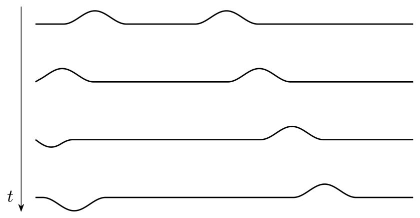
$$f_0 = \frac{c}{2L}\,. \tag{2.39}$$

2.4.2 Discrete time

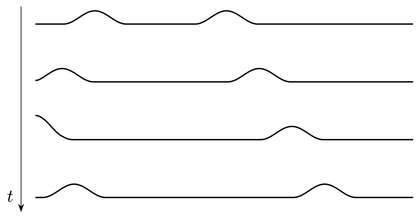
The most straightforward discretisation of Eq. (2.37) is the following FD scheme

$$\delta_{tt} u_l^n = c^2 \delta_{xx} u_l^n, \tag{2.40}$$

with $l \in \{0, \ldots, N\}$ and number of grid points $N+1$. Other schemes exist (see fx. [3]), but are excluded as they have not been used in this work. Expanding



(a) The Dirichlet boundary condition in Eq. (2.38a) fixes the boundary, which causes the incoming waves to invert.



(b) The Neumann or free boundary condition in Eq. (2.38b) fixes the slope at the boundary, causing the incoming waves to not invert.

Fig. 2.7: The behaviour of the 1D wave equation with (a) Dirichlet or (b) Neumann boundary conditions.

the operators yields

$$\frac{1}{k^2}\left(u_l^{n+1} - 2u_l^n + u_l^{n-1}\right) = \frac{c^2}{h^2}\left(u_{l+1}^n - 2u_l^n + u_{l-1}^n\right). \tag{2.41}$$

and solving for u_l^{n+1} yields

$$u_l^{n+1} = \left(2 - 2\lambda^2\right)u_l^n + \lambda^2\left(u_{l+1}^n + u_{l-1}^n\right) - u_l^{n-1}. \tag{2.42}$$

Here,

$$\lambda = \frac{ck}{h} \tag{2.43}$$

is called the *Courant number* and plays a big role in stability quality of the FD scheme. The Courant number needs to abide the (famous) Courant-Friedrichs-Lewy or *CFL condition* for short [6]

$$\lambda \leq 1, \tag{2.44}$$

which acts as a stability condition for scheme (2.40). If $\lambda = 1$, Eq. (2.40) is an exact solution to Eq. (2.37). Exact solutions are actually quite uncommon in the realm of differential equations! If $\lambda < 1$, the quality of the simulation quality decreases and dispersive and bandlimiting effects occur (see Section 2.4.3).

Boundary Conditions

The end points of the discrete domain are located at $l = 0$ and $l = N$. Substituting these locations into Eq. (2.42) seemingly shows that grid points outside of the defined domain are needed, namely u_{-1}^n and u_{N+1}^n. These can be referred to as *virtual grid points* and can be accounted for by discretising the boundary conditions in Eq. (2.38). Discretising these (using the centred spatial difference operator for the Neumann condition) yields

$$u_0^n = u_N^n = 0, \quad \text{(Dirichlet)} \tag{2.45a}$$
$$\delta_{x\cdot} u_0^n = \delta_{x\cdot} u_N^n = 0. \quad \text{(Neumann)} \tag{2.45b}$$

Expanding the operators in Eq. (2.45b) and solving for u_{-1}^n and u_{N+1}^n provides the definitions for these virtual grid points based on values in the discrete domain:

$$\begin{aligned}
&\frac{1}{2h}\left(u_1^n - u_{-1}^n\right) = 0, && \frac{1}{2h}\left(u_{N+1}^n - u_{N-1}^n\right) = 0,\\
&u_1^n - u_{-1}^n = 0, && u_{N+1}^n - u_{N-1}^n = 0,\\
&u_{-1}^n = u_1^n. && u_{N+1}^n = u_{N-1}^n.
\end{aligned}$$

If Dirichlet boundary conditions are used, the states of the boundary points will always be zero and can therefore can be excluded from the calculations. The range of calculation then simply becomes $l \in \{1, \dots, N-1\}$. If, on the other hand, Neumann conditions are used, the update equation in (2.42) at the boundaries will have the the above definitions for the virtual grid points substituted at $l = 0$ and $l = N$ and will become

$$u_0^{n+1} = \left(2 - 2\lambda^2\right) u_0^n + 2\lambda^2 u_1^n - u_0^{n-1}, \tag{2.46}$$

and

$$u_N^{n+1} = \left(2 - 2\lambda^2\right) u_N^n + 2\lambda^2 u_{N-1}^n - u_N^{n-1}, \tag{2.47}$$

at the left and right boundary respectively.

2.4.3 Implementation and Output

As c, k and h are interdependent due to the CFL condition in (2.44), it is useful to rewrite this condition in terms of known variables. As the time step k is based on the sample rate and thus (usually) fixed, and c is a user-defined wave speed, Eq. (2.44) can be rewritten in terms of the grid spacing h:

$$h \geq ck, \tag{2.48}$$

which, in implementation, is used as a stability condition for the scheme. See Section 3.3 for more information on how to derive the stability condition from a FD scheme. Again, if Eq. (2.48) is satisfied with equality, the FD scheme is an exact solution to the PDE, and if h deviates from this condition, the quality of the simulation decreases.

So why would the stability condition not be satisfied with equality? As mentioned in Section 2.2.1, a continuous domain $\mathcal{D} = [0, L]$ for a system of length L needs to be divided into N equal sections of length h. A logical step to calculate N would be to divide L by h calculated using (2.48) satisfied with equality. However, this calculation might not result in an integer value, which N should be! To stay as close to the stability condition as possible, the following calculations are performed in order:

$$h := ck, \quad N := \left\lfloor \frac{L}{h} \right\rfloor, \quad h := \frac{L}{N}, \quad \lambda := \frac{ck}{h}, \tag{2.49}$$

where $\lfloor \cdot \rfloor$ denotes the flooring operation. In other words, Eq. (2.48) is satisfied with equality and used to calculate integer N. After this, h is recalculated based on N and used to calculate the Courant number λ. This process assures that N is an integer and that the CFL condition is satisfied, though not necessarily with equality.

If this recalculation of h is not carried out, the behaviour of the system will not be correct. For example, we want our 1D wave equation to be 1 m long ($L = 1$) and produce a fundamental frequency of $f_0 = 750$ Hz according to (2.39). This requires a wave speed of $c = 1500\,\mathrm{m/s}$. If we use a commonly-used sample rate of $f_\mathrm{s} = 44100$ Hz, and recalling that $k = 1/f_\mathrm{s}$, these values can be filled into (2.48) satisfied with equality yielding $h \approx 0.034$. If we divide the length by the grid spacing, we get $L/h = 29.4$, meaning that exactly 29.4 intervals of size h fit in the domain $\mathcal{D}$. However, the number of intervals needs to be an integer and – using the above values – we get $N = 29$. If h is not recalculated according to (2.49), the total length will be 29 times the grid spacing h. This results in $L \approx 0.986$ and is slightly less than the original length of 1. Although the CFL condition will be satisfied with equality, the fundamental frequency will be slightly tuned up: $f_0 \approx 760.34$ Hz. If h is recalculated based on N, L and f_0 will be unchanged, and $\lambda \approx 0.986$ which is still very close to satisfying condition (2.44).

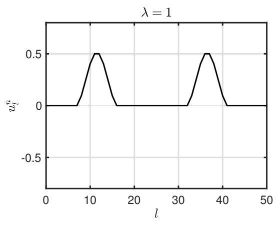
Output

After the system is excited (see III), one can retrieve the output of the system by selecting a grid point and listening to that at at the given sample rate f_s. The amount of modes is determined by the number of moving points in the system. If Dirichlet boundary conditions are used this means that there are $N-1$ modes, and $N+1$ modes for Neumann boundary conditions.

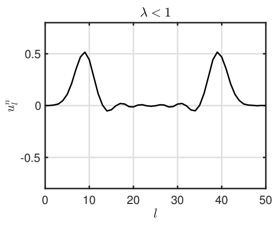
If the CFL condition is satisfied with equality, these modes are integer multiples of the fundamental: $f_m = mf_0$ for mode number $m \in \{1, \dots, N-1\}$ for Dirichlet and $m \in \{0, \dots, N\}$ for Neumann boundary conditions. The frequency of the harmonic partials can also be analytically / numerically derived using modal analysis as will be explained in Section 3.5.

The amplitude of the different modes, on the other hand, depends on the location of excitation and output. modes

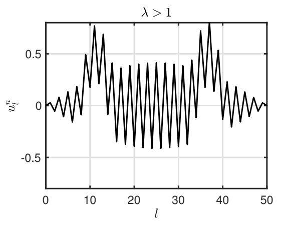
See Appendix C.2MATLAB implementation of the 1D wave equation



(a) $\lambda = 1$

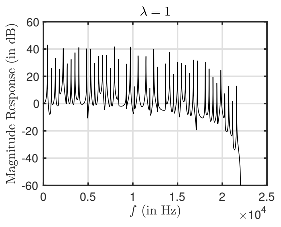


(b) $\lambda = 0.9$

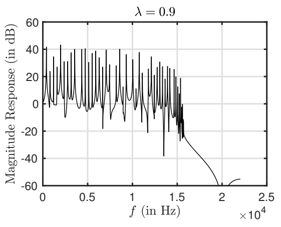


(c) $\lambda = 1.001$

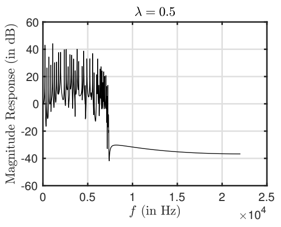
Fig. 2.8: State u_l^n with $N = 50$ and $f_s = 44100$ visualised ~ 100 samples after excitation. (a) If $\lambda = 1$, the solution is exact. (b) If $\lambda < 1$ dispersive behaviour shows. (c) If $\lambda > 1$ the CFL condition in Eq. (2.44) is not satisfied and the system is unstable.caption is straight from paper



(a) $\lambda = 1$



(b) $\lambda = 0.9$



(c) $\lambda = 1.001$

Fig. 2.9: Bandwidths of the simulation output at $l = 16$ with $f_s = 44100$ Hz and $N = 50$ excited with a raised cosine with a width of 5 at center-location $N = 25$. The Courant number is set to (a) $\lambda = 1$, (b) $\lambda = 0.9$ and (c) $\lambda = 0.5$.caption is straight from paper

Chapter 3

Analysis Techniques

This chapter provides some techniques to analyses FD schemes .

Techniques to analyse PDEs also exist, but the focus here is on the discrete schemes.

3.1 Preamble

Before going into analysis techniques of the FD schemes presented in this chapter, some mathematical tools used for these techniques will be presented.

Continuous time

For two functions $f(x)$ and $g(x)$ defined for $x \in \mathcal{D}$, their l_2 inner product and l_2 norm are defined as

$$\langle f, g\rangle_{\mathcal{D}} = \int_{\mathcal{D}} fg dx \quad \text{and} \quad \|f\|_{\mathcal{D}} = \sqrt{\langle f, f\rangle_{\mathcal{D}}} \tag{3.1}$$

Integration by parts

Discrete-time

Inner product of any time series f^n and g^n defined over domain $\mathcal{D}$ and the discrete counterpart to (3.1) is

$$\langle f^n, g^n\rangle_{\mathcal{D}} = \sum_{l\in\mathcal{D}} h f_l^n g_l^n \tag{3.2}$$

where the multiplication by h is the discrete counterpart of dx the continuous definition.

3.1.1 Summation by Parts

for

3.2 Matrices

For several purposes, such as implementation in `MATLAB` and several analysis techniques described shortly, is useful to write an FD scheme in *matrix form*.

In this document, matrices and vectors are written using bold symbols. Many notations exist, blabla $\bar{a}$ $\vec{a}$ A matrix uses a capital letter whereas vectors are decapitalised. The dimensions of a matrix are denoted using "$row \times column$". A 6×14 matrix, for example, thus has 6 rows and 14 columns. Along those lines, a *row vector* is a matrix with 1 row and more than 1 column and a *column vector* is a matrix with 1 column and more than 1 row. If a matrix has only 1 row and 1 column, it can be used as a *scalar*.

Matrix Multiplication

In order for matrix multiplication (including matrix-vector multiplication) to be valid, the number of columns of the first matrix needs to be equal to the number of rows in the second matrix. The result will then be a matrix with a number of rows equal to that of the first matrix and a number of columns equal to that of the second matrix. See Figure 3.1 for reference.

As an example, consider the $L \times M$ matrix $\mathbf{A}$ and a $M \times N$ matrix $\mathbf{B}$. The multiplication $\mathbf{AB}$ is defined as the number of columns of matrix $\mathbf{A}$ (M) is equal to the number of rows of matrix $\mathbf{B}$ (also M). The result, $\mathbf{C}$, is a $L \times N$ matrix. The multiplication $\mathbf{BA}$ is undefined as the number of columns of the first matrix does not match the number of rows in the second matrix.

Multiplying two matrices written in their dimensions as

$$\overbrace{(L \times M)}^{\mathbf{A}} \cdot \overbrace{(M \times N)}^{\mathbf{B}} = \overbrace{(L \times N)}^{\mathbf{C}} \tag{3.3}$$

In a FDTD context

Matrix multiplication when working with FDTD methods usually involves multiplying a square matrix (with equal rows and columns) onto a column vector (see Figure 3.1a). Consider a square matrix with $(N+1) \times (N+1)$ elements $\mathbf{A}$ and a $(N+1) \times 1$ column vector $\mathbf{u}$. Multiplying these results in a $(N+1) \times 1$ column vector $\mathbf{w}$:

$$\mathbf{Au} = \mathbf{w}. \tag{3.4}$$

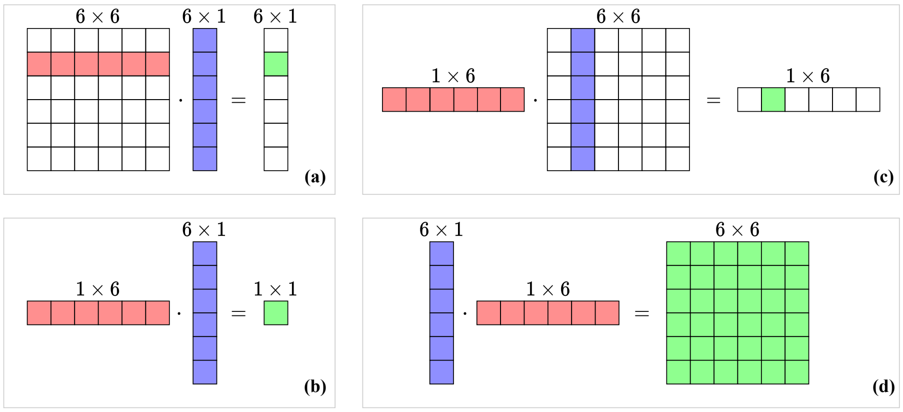


Fig. 3.1: Visualisation of valid matrix multiplications. The "inner" dimensions (columns of the left matrix and rows of the right) must match and result in a matrix with a size of "outer" dimensions (rows of the left matrix and columns of the right).

Expanding this operation results in

$$\underbrace{\begin{bmatrix} a_{00} & a_{01} & \dots & a_{0N} \\ a_{10} & a_{11} & \dots & a_{2N} \\ \vdots & \vdots & & \vdots \\ a_{N0} & a_{N1} & \dots & a_{NN} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{bmatrix}}_{\mathbf{u}} = \underbrace{\begin{bmatrix} a_{00}u_0 + a_{01}u_1 + \ldots + a_{0N}u_N \\ a_{10}u_0 + a_{11}u_1 + \ldots + a_{1N}u_N \\ \vdots \\ a_{N0}u_0 + a_{N1}u_1 + \ldots + a_{NN}u_N \end{bmatrix}}_{\mathbf{w}} \tag{3.5}$$

where the subscripts for a indicate the indices for the row and column.

Operators in Matrix Form

FD operators approximating first-order spatial derivatives can be written in matrix form and applied to a column vector $\mathbf{u}$ containing the state of the system. The size of these matrices depend on the number of grid points the system is described for and the boundary conditions.

Not assuming any size for now, the FD operators in (2.6) can be written in matrix form according to

$$\mathbf{D}_{x+} = \frac{1}{h}\begin{bmatrix} \ddots & \ddots & & & \mathbf{0} & \\ & -1 & 1 & & & \\ & & -1 & 1 & & \\ & & & -1 & 1 & \\ & & & & -1 & \ddots \\ & \mathbf{0} & & & & \ddots \end{bmatrix} \quad \mathbf{D}_{x-} = \frac{1}{h}\begin{bmatrix} \ddots & & & & \mathbf{0} & \\ \ddots & 1 & & & & \\ & -1 & 1 & & & \\ & & -1 & 1 & & \\ & & & -1 & 1 & \\ & \mathbf{0} & & & \ddots & \ddots \end{bmatrix}$$

$$\mathbf{D}_{x\cdot} = \frac{1}{2h}\begin{bmatrix} \ddots & \ddots & & & \mathbf{0} & \\ \ddots & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & -1 & 0 & 1 & \\ & & & -1 & 0 & \ddots \\ & \mathbf{0} & & & \ddots & \ddots \end{bmatrix}$$

Averaging operators μ_{x+}, μ_{x-} and $\mu_{x\cdot}$ are defined in a similar way:

$$\mathbf{M}_{x+} = \frac{1}{2}\begin{bmatrix} \ddots & \ddots & & & \mathbf{0} & \\ & 1 & 1 & & & \\ & & 1 & 1 & & \\ & & & 1 & 1 & \\ & & & & 1 & \ddots \\ & \mathbf{0} & & & & \ddots \end{bmatrix} \quad \mathbf{M}_{x-} = \frac{1}{2}\begin{bmatrix} \ddots & & & & \mathbf{0} & \\ \ddots & 1 & & & & \\ & 1 & 1 & & & \\ & & 1 & 1 & & \\ & & & 1 & 1 & \\ & \mathbf{0} & & & \ddots & \ddots \end{bmatrix}$$

$$\mathbf{M}_{x\cdot} = \frac{1}{2}\begin{bmatrix} \ddots & \ddots & & & \mathbf{0} & \\ \ddots & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & 1 & 0 & 1 & \\ & & & 1 & 0 & \ddots \\ & \mathbf{0} & & & \ddots & \ddots \end{bmatrix}$$

It is important to notice that only spatial operators are written in this matrix form and then applied to state vectors at different time steps ($n+1$, n and $n-1$).

Finally, the identity matrix is a matrix with only 1s on the diagonal and 0s elsewhere as

$$\mathbf{I} = \begin{bmatrix} \ddots & & & & \mathbf{0} & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & \mathbf{0} & & & & \ddots \end{bmatrix} \tag{3.6}$$

Schemes and Update Equations in Matrix Form

With these building blocks, Eq. (2.40) can be written in matrix form.

If Dirichlet boundary conditions are used (Eq. (2.45a)) the end points of the system do not have to be included in the calculation. The values of the grid function u_l^n for $l \in \{1, \dots, N-1\}$ can be stored in a state vector according to $\mathbf{u}^n = [u_1^n, \dots, u_{N-1}^n]^T$ where T denotes the transpose operation. Furthermore, $(N-1) \times (N-1)$ matrices $\mathbf{D}_{x+}$ and $\mathbf{D}_{x-}$ can be multiplied to get a same-sized matrix $\mathbf{D}_{xx}$:

check whether to not put this earlier

$$\mathbf{D}_{xx} = \mathbf{D}_{x+}\mathbf{D}_{x-} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \mathbf{0} \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ \mathbf{0} & & & 1 & -2 \end{bmatrix}. \tag{3.7}$$

If instead, Neumann boundary conditions are used (Eq. (2.45a)), the values of u_l^n for the full range $l \in \{0, \dots, N\}$ will be stored as $\mathbf{u}^n = [u_0^n, \dots, u_N^n]^T$ and the $(N+1) \times (N+1)$ matrix $\mathbf{D}_{xx}$ will be

$$\mathbf{D}_{xx} = \frac{1}{h^2}\begin{bmatrix} -2 & 2 & & & \mathbf{0} \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ \mathbf{0} & & & 2 & -2 \end{bmatrix}, \tag{3.8}$$

where the 2s correspond to the multiplication by 2 with u_1^n and u_{N-1}^n in Eqs. (2.46) and (2.47) respectively.

Regardless of the boundary conditions, the FD scheme in (2.40) can be written in matrix form as

$$\frac{1}{k^2}\left(\mathbf{u}^{n+1} - 2\mathbf{u} + \mathbf{u}^{n-1}\right) = c^2\mathbf{D}_{xx}\mathbf{u}^n, \tag{3.9}$$

and rewritten to a matrix form of the update equation analogous to Eq. (2.42)

$$\mathbf{u}^{n+1} = (2\mathbf{I} + c^2k^2\mathbf{D}_{xx})\mathbf{u}^n - \mathbf{u}^{n-1}. \tag{3.10}$$

The identity matrix is necessary here for correct matrix addition.

3.2.1 Note on square matrices

If a matrix is square it has the potential to have an inverse. A square matrix $\mathbf{A}$ is invertable if there exists a matrix $\mathbf{B}$ such that

$$\mathbf{AB} = \mathbf{BA} = \mathbf{I}. \tag{3.11}$$

This matrix $\mathbf{B}$ is then called the *inverse* of $\mathbf{A}$ and written as $\mathbf{A}^{-1}$. That a matrix is square does not mean that it has an inverse, and is called *singular*.

3.2.2 System of Linear Equations

A number of unknowns described by the same number of equations can be solved using a

$$\begin{aligned}\mathbf{Au} &= \mathbf{w}\\ \mathbf{u} &= \mathbf{A}^{-1}\mathbf{w}\end{aligned}$$

3.2.3 Eigenvalue Problems

A square matrix $\mathbf{A}$ is characterised by its *eigenvalues* and corresponding *eigenvectors*. In a FDTD context, these are usually associated with the modes of a system, where the eigenvalues relate to the modal frequencies and the eigenvectors the the modal shapes. Section 3.5 will provide more information on this.

ask stefan

To find these characteristic values for a $p \times p$ matrix $\mathbf{A}$ an equation of the following form must be solved

$$\mathbf{A}\boldsymbol{\phi} = \lambda\boldsymbol{\phi}. \tag{3.12}$$

This is called is an *eigenvalue problem* and has p solutions (corresponding to the dimensions of $\mathbf{A}$). These are the p^{th} eigenvector $\boldsymbol{\phi}_p$ and the corresponding eigenvalue λ_p calculated using

$$\lambda_p = \text{eig}_p(\mathbf{A}), \tag{3.13}$$

where $\text{eig}_p(\cdot)$ denotes the p^{th} eigenvalue of. Instead of delving too deep into eigenvalue problems and the process of how to solve them, an easy way to obtain the solutions using MATLAB is provided here:

Check code here!

```
[~, lambda, phi] = eig(A, 'vector');
```

The eigenvalues are given in a $p \times 1$ column vector `lambda` and the p^{th} eigenvector appears in the p^{th} column of $p \times p$ matrix `phi`. Note that the outcome is not sorted! To do this, do

```
[lambda, order] = sort(lambda);
phi = phi(:, order);
```

3.3 von Neumann Analysis

Much literature gives the stability condition using an "it can be shown that" argument (fx. [2]). Here, I would like to take the opportunity to

Finding stability conditions for models

The following trigonometric identities will come in handy when performing the analyses [18, p. 71]:

$$\sin(x) = \frac{e^{jx} - e^{-jx}}{2j} \quad \Rightarrow \quad \sin^2(x) = \frac{e^{j2x} + e^{-j2x}}{-4} + \frac{1}{2}, \tag{3.14a}$$

$$\cos(x) = \frac{e^{jx} + e^{-jx}}{2} \quad \Rightarrow \quad \cos^2(x) = \frac{e^{j2x} + e^{-j2x}}{4} + \frac{1}{2}. \tag{3.14b}$$

One can analyse a FD scheme by

FULL DOC SWEEP:check 'a' or 'an' FD scheme

$$u_l^n = z^n e^{jl\beta h} \tag{3.15}$$

where β is a real wavenumber. Important to remember is that without a shift in space (fx. $l+1$) or time (fx. $n-1$) $l = 0$ or $n = 0$ respectively:

$$u_l^n = z^0 e^{j0\beta h} = 1 \tag{3.16a}$$
$$u_{l+1}^n = z^0 e^{j1\beta h} = e^{j\beta h} \tag{3.16b}$$
$$u_{l-1}^n = z^0 e^{j(-1)\beta h} = e^{-j\beta h} \tag{3.16c}$$
$$u_{l+2}^n = z^0 e^{j2\beta h} = e^{j\beta h} \tag{3.16d}$$
$$u_{l-2}^n = z^0 e^{j(-2)\beta h} = e^{-j2\beta h} \tag{3.16e}$$
$$u_l^{n+1} = z^1 e^{j0\beta h} = z \tag{3.16f}$$
$$u_l^{n-1} = z^{-1} e^{j0\beta h} = z^{-1} \tag{3.16g}$$

Talk about solution of

3.4 Energy Analysis

The Hamiltonian or $\mathfrak{H}$

Multiplying scheme by $(\delta_{t\cdot} u_l^n)$

Debugging physical models... One can plot the energy of the system and in a lossless system, the rate of change of the total energy should be 0, i.e.,

$$\delta_{t+}\mathfrak{h} = 0 \quad \implies \quad \mathfrak{h}^n = \mathfrak{h}^0. \tag{3.17}$$

.

Although the energy of a lossless system should be unchanged according to Eq. (3.17), but in a finite precision simulation, ultra slight fluctuations of the energy should be visible due to rounding errors.

Plotting energy should be within *machine precision*, which mostly is in the range of 10^{-15}

Product Identities

Some useful identities used in this work are

$$(\delta_{t\cdot}u_l^n)(\delta_{tt}u_l^n) = \delta_{t+}\left(\frac{1}{2}(\delta_{t-}u_l^n)^2\right), \tag{3.18a}$$

$$(\delta_{t\cdot}u_l^n)u_l^n = \delta_{t+}\left(\frac{1}{2}u_l^n e_{t-}u_l^n\right), \tag{3.18b}$$

$$(\delta_{t+}u_l^n)(\mu_{t+}u_l^n) = \delta_{t+}\left(\frac{1}{2}(u_l^n)^2\right), \tag{3.18c}$$

$$(\delta_{t\cdot}u_l^n)(\mu_{t\cdot}u_l^n) = \delta_{t\cdot}\left(\frac{1}{2}(u_l^n)^2\right). \tag{3.18d}$$

Again, these can be used for spatial derivatives as well by substituting the 't' subscripts for 'x'. Also to

When an operator is applied to a product of two grid functions, the discrete counter part of the product rule needs to be used according to

$$\delta_{t+}(u_l^n w_l^n) = (\delta_{t+}u_l^n)(\mu_{t+}w_l^n) + (\mu_{t+}u_l^n)(\delta_{t+}w_l^n). \tag{3.19}$$

3.4.1 Stability Analysis

One can perform stability analysis using the energy analysis techniques presented here. To arrive at a condition the energy must be *positive definite*

3.5 Modal Analysis

Modes are the resonant frequencies of a system. The amount of modes that a discrete system contains depends on the amount of moving points. A mass-spring system thus has one resonating mode, but a FD scheme of the 1D wave equation with $N = 30$ and Dirichlet boundary conditions will have 29 modes.

This section will show how to obtain the modes of an FD scheme implementing the 1D wave equation.

We start by using the the matrix form of FD scheme (2.40) from Eq. (3.9)

$$\frac{1}{k^2}\left(\mathbf{u}^{n+1}-2\mathbf{u}^n+\mathbf{u}^{n-1}\right)=c^2\mathbf{D}_{xx}\mathbf{u}.$$

Following [3] we assume a solution of the form $\mathbf{u}=z^n\boldsymbol{\phi}$. Substituting this into the above equation yields the characteristic equation

more explanation, perhaps refer to von neumann analysis in 3.3

$$(z-2+z^{-1})\boldsymbol{\phi}=c^2k^2\mathbf{D}_{xx}\boldsymbol{\phi}. \tag{3.20}$$

This is an eigenvalue problem (see Section 3.2.3) where the p^{th} solution $\boldsymbol{\phi}_p$ may be interpreted as the modal shape of mode p. The modal frequencies are the solutions to the following equations:

$$\begin{aligned} z_p-2+z_p^{-1}&=c^2k^2\text{eig}_p(\mathbf{D}_{xx}),\\ z_p+(-2-c^2k^2\text{eig}_p(\mathbf{D}_{xx}))+z_p^{-1}&=0. \end{aligned} \tag{3.21}$$

If the CFL condition for the scheme is satisfied, the roots will lie on the unit circle. Furthermore we can substitute a test solution $z_p=e^{j\omega_pk}$ with angular frequency of the p^{th} mode ω_p and solve for these:

$$\begin{aligned} e^{j\omega_pk}+e^{-j\omega_pk}-2-c^2k^2\text{eig}_p(\mathbf{D}_{xx})&=0,\\ \frac{e^{j\omega_pk}+e^{-j\omega_pk}}{-4}+\frac{1}{2}+\frac{c^2k^2}{4}\text{eig}_p(\mathbf{D}_{xx})&=0, \end{aligned}$$

and using Eq. (3.14a) we get

$$\begin{aligned} \sin^2(\omega_pk/2)+\frac{c^2k^2}{4}\text{eig}_p(\mathbf{D}_{xx})&=0,\\ \sin(\omega_pk/2)&=\frac{ck}{2}\sqrt{-\text{eig}_p(\mathbf{D}_{xx})},\\ \omega_p&=\frac{2}{k}\sin^{-1}\left(\frac{ck}{2}\sqrt{-\text{eig}_p(\mathbf{D}_{xx})}\right). \end{aligned} \tag{3.22}$$

3.5.1 One-Step Form

For more complicated systems, where the coefficients of z and z^{-1} in the characteristic equation are not identical for example, it is useful to rewrite the update in *one-step form*. Although the eigenfrequency calculation needs to be done on a larger matrix, it allows for a more general and direct way to calculate the modal frequencies and damping coefficients per mode.

If matrix $\mathbf{A}$ has an inverse, any scheme of the form

$$\mathbf{A}\mathbf{u}^{n+1}=\mathbf{B}\mathbf{u}^n+\mathbf{C}\mathbf{u}^{n-1}, \tag{3.23}$$

can be rewritten to

$$\underbrace{\begin{bmatrix}\mathbf{u}^{n+1}\\ \mathbf{u}^{n}\end{bmatrix}}_{\mathbf{w}^{n+1}} = \underbrace{\begin{bmatrix}\mathbf{A}^{-1}\mathbf{B} & \mathbf{A}^{-1}\mathbf{C}\\ \mathbf{I} & \mathbf{0}\end{bmatrix}}_{\mathbf{Q}}\underbrace{\begin{bmatrix}\mathbf{u}^{n}\\ \mathbf{u}^{n-1}\end{bmatrix}}_{\mathbf{w}^{n}} \tag{3.24}$$

which relates the unknown state of the system to the known state through matrix $\mathbf{Q}$ which encompases the scheme. The sizes of the identity matrix $\mathbf{I}$ and zero matrix $\mathbf{0}$ are the same size as $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ and are essentially used to make the matrix square and allow for eigenvalue calculation.

Again we can assume solutions of the form $\mathbf{w} = z^n\boldsymbol{\phi}$ (where ϕ is less-trivially connected to the modal shapes) and get

$$z\boldsymbol{\phi} = \mathbf{Q}\boldsymbol{\phi}, \tag{3.25}$$

which can be solved for the pth eigenvalue as

$$z_p = \text{eig}_p(\mathbf{Q}). \tag{3.26}$$

As the scheme could exhibit damping, the test solution needs to include this and will be $z_p = e^{s_p k}$ with complex frequency $s_p = j\omega_p + \sigma_p$ and damping of the pth eigenfrequency σ_p. Substituting the test solution and solving for s_p yields

$$\begin{aligned} e^{s_p k} &= \text{eig}_p(\mathbf{Q}),\\ s_p &= \frac{1}{k}\ln\left(\text{eig}_p(\mathbf{Q})\right). \end{aligned} \tag{3.27}$$

Solutions for the frequency and damping for the pth eigenvalue can then be obtained through

$$\omega_p = \Im(s_p) \quad \text{and} \quad \sigma_p = \Re(s_p), \tag{3.28}$$

where $\Im(\cdot)$ and $\Re(\cdot)$ denote the "imaginary part of" and "real part of" respectively.

As the elements of $\mathbf{Q}$ are real-valued, the solutions s_p in Eq. (3.27) come in complex conjugates. For analysis, only the $\Im(s_p) \geq 0$ should be considered as these correspond to non-negative frequencies.

3.6 Dispersion analysis

Part II

Resonators

Resonators

Though the physical models described in the previous part are also considered resonators, they are *ideal* cases. In other words, you would not be able to find these “in the wild" as they do not includes effects such as losses or dispersion in the case of the 1D wave equation.

Scaling

Scaling, or non-dimensionalisation can be useful to reduce the parameter space

Domain $x \in [0, L]$ is scaled to $x' = x/L$ such that $x' \in [0, 1]$.

The 1D wave equation in (2.37) can be rewritten to

$$\partial_t^2 u = \gamma^2 \partial_{x'x'} u \tag{3.29}$$

where scaled wavespeed $\gamma = c/L$ has units of frequency.

Although this parameter reduction might be useful for resonators in isolation, when they are connected

Moreover, for the parameters to make

As, later on, different resonators in isolation will be connected, the overview is better kept when all parameters are written out

- Bars and Stiff Strings
- 2D Models
- Brass

Chapter 4

Stiff string

Stiff string and stuff Used in many of the publications in Part IX VI

4.1 Continuous time

Consider the transverse displacement of a lossless stiff string of length L described by $u = u(x,t)$ defined for $x \in \mathcal{D}$ with domain $\mathcal{D} = [0, L]$ and time $t \geq 0$. The PDE describing its motion is

should I even include the lossless one? It's just so that we can slowly build up to the damped model...

$$\rho A \partial_t^2 u = T \partial_x^2 u - EI \partial_x^4 u \tag{4.1}$$

parameterised by material density ρ (in kg/m^3), cross-sectional area $A = \pi r^2$ (in m^2), radius r (in m) tension T (in N), Young's modulus E (in Pa) and area moment of inertia $I = \pi r^4/4$ (in m^4). If either E or I is 0, Eq (4.1) reduces to the 1D wave equation in (2.37) where $c = \sqrt{T/\rho A}$. If instead $T = 0$, Eq. (4.1) reduces to the *ideal bar* equation.

Dispersion Analysis

The 4th-order spatial derivative models *frequency dispersion*, a phenomenon that causes different frequencies to travel at different speeds. As opposed to the undesired numerical dispersion

4.1.1 Adding Losses

Before moving on to the discretisation of Eq. (4.1), losses can be added to the system. This is done by simply adding terms to (4.1) according to

First appeared in [1]

$$\rho A \partial_t^2 u = T \partial_x^2 u - EI \partial_x^4 u - 2\sigma_0 \rho A \partial_t u + 2\sigma_1 \rho A \partial_t \partial_x^2 u \tag{4.2}$$

where the loss coefficients σ_0 (in s^{-1}) and σ_1 (in m^2/s) describe the frequency dependent and frequency independent losses respectively.

A more compact way to write Eq. (4.2), and as is also found often in the literature [3] is to divide both sides by ρA to get

etc.

$$\partial_t^2 u = c^2\partial_x^2 u - \kappa^2\partial_x^4 u - 2\sigma_0\partial_t u + 2\sigma_1\partial_t\partial_x^2 u \tag{4.3}$$

where $c = \sqrt{T/\rho A}$ is the wave speed (in m/s) as in the 1D wave equation in (2.37) and $\kappa = \sqrt{EI/\rho A}$ is a *stiffness coefficient* (in m^2/s).

check wavespeed or wave speed (entire document)

Intuition

Although Eq. (4.2) might look daunting at first, the principle of Newton's second law remains the same.

Something about the 4th spatial derivative and the loss terms here...

Boundary Conditions

The boundary conditions found in Eq. (2.38) can be extended to

$$u = \partial_x u = 0 \quad \text{(clamped)} \tag{4.4a}$$
$$u = \partial_x^2 u = 0 \quad \text{(simply supported)} \tag{4.4b}$$
$$\partial_x^2 u = \partial_x^3 u = 0 \quad \text{(free)} \tag{4.4c}$$

at $x = 0, L$.

insert figure showing virtual grid points

4.2 Discrete Time

Equation (4.2) can be discretised as

$$\rho A\delta_{tt}u_l^n = T\delta_{xx}u_l^n - EI\delta_{xxxx}u_l^n - 2\sigma_0\rho A\delta_{t\cdot}u_l^n + 2\sigma_1\rho A\delta_{t-}\delta_{xx}u_l^n \tag{4.5}$$

The δ_{xxxx} operator is the the second-order spatial difference in Eq. (2.8) applied to itself.

$$\delta_{xxxx} = \delta_{xx}\delta_{xx} = \frac{1}{h^4}\left(e_{x+}^2 - 4e_{x+} + 6 - 4e_{x-} + e_{x-}^2\right) \tag{4.6}$$

4.2.1 Combining operators

Operators can be combined by simply multiplying their definitions. Recalling the definitions for δ_{t-} in Eq. (2.3b) and δ_{xx} Eq. (2.8) their combination results in

$$\begin{aligned}\delta_{t-}\delta_{xx} &= \frac{1}{k}\left(1 - e_{t-}\right)\frac{1}{h^2}\left(e_{x+} - 2 + e_{x-}\right),\\ &= \frac{1}{kh^2}\left(e_{x+} - 2 + e_{x-} - e_{t-}(e_{x+} - 2 + e_{x-})\right).\end{aligned}$$

A multiplication of two (different)shift operators applied to a grid function simply means to apply each shift individually. The $\delta_{t-}\delta_{xx}$ operator applied to u_l^n thus yields

$$\delta_{t-}\delta_{xx}u_l^n = \frac{1}{hk^2}\left(u_{l+1}^n - 2u_l^n + u_{l-1}^n - u_{l+1}^{n-1} + 2u_l^{n-1} - u_{l-1}^{n-1}\right). \tag{4.7}$$

4.3 Modal analysis

To perform a modal analysis on the FD scheme of the damped stiff string in (4.5)

The matrix form of the δ_{xxxx} operator in (4.6) with simply supported boundary conditions can be obtained by multiplying two $\mathbf{D}_{xx}$ matrices according to

$$\mathbf{D}_{xx}\mathbf{D}_{xx} = \mathbf{D}_{xxxx} = \frac{1}{h^4}\begin{bmatrix} 5 & -4 & 1 & & & \mathbf{0} & \\ -4 & 6 & \ddots & \ddots & & & \\ 1 & \ddots & \ddots & -4 & 1 & & \\ & \ddots & -4 & 6 & -4 & \ddots & \\ & & 1 & -4 & \ddots & \ddots & 1 \\ & & & \ddots & \ddots & 6 & -4 \\ & \mathbf{0} & & & 1 & -4 & 5 \end{bmatrix}. \tag{4.8}$$

Chapter 5

Brass

5.1 Second-order system

Acoustic potential...

5.2 First-order system

This will be the first appearance of a first-order system.

Chapter 6

2D Systems

In this work, it is mainly used to model a simplified body in papers...

State variable $u = u(x, y, t)$ where $t \geq 0$ and $(x, y) \in \mathcal{D}$ where $\mathcal{D}$ is 2-dimensional. The state variable can then be discretised according to $u(x, y, t) \approx u_{l,m}^n$ with space $x = lh$ and $y = mh$ and time $t = nk$ and $k = 1/f_s$. For simplicity the grid spacing in both the x and y directions are set to be the same but could be different.

In continuous time the operators:

$$\Delta = \partial_x^2 + \partial_y^2 \tag{6.1}$$

The same shift operators as defined in Chapter 2 can be applied to grid function $u_{l,m}^n$. Additional ones are

$$e_{y+}u_{l,m}^n = u_{l,m+1}^n, \quad \text{and} \quad e_{y-}u_{l,m}^n = u_{l,m-1}^n. \tag{6.2}$$

6.1 2D Wave Equation

The 2D wave equation be used to model an ideal membrane such as done in

$$\partial_t^2 u = c^2 \Delta u \tag{6.3}$$

where $c = \sqrt{T/\rho H}$ is the wavespeed (in m/s) T is the tension per unit lengthapplied to the boundary (in N/m), material density

check whether this is right..

6.2 Thin plate

Used in [A], [B], [D] and [E] biharmonic operator, Laplacian in (**??**) applied to itself.

$$\rho H \partial_t^2 u = -D \Delta\Delta u \tag{6.4}$$

where $D = EH^3/12(1-\nu^2)$

6.3 Stiff membrane

Combination between Eqs. (6.5) and (6.4)

$$\rho H \partial_t^2 u = T\Delta u \tag{6.5}$$

[F]

Part III

Exciters

Exciters

Now that a plethora of resonators have been introduced in part II, different mechanisms to excite them will be introduced here. First, different examples of

- Simple pluck ((half) raised-cos)
- Hammer
- Bow Models
- Lip reed

Chapter 7

Unmodelled Excitations

Different title here?

7.1 Initial conditions

Hammer

Full raised cosine

Pluck

- Cut-off raised cosine
- Triangle (for string)

7.2 Signals

7.2.1 Pulse train

For brass

7.2.2 Noise

Noise input

Chapter 8

Modelled Excitations

8.1 Hammer but maybe not as I didn't use it..

Hammer modelling

8.1.1 Mass-spring Systems Revisited: Adding Damping

Damping can be added to Eq. (2.28)

$$M\ddot{u} = -Ku - R\dot{u} \tag{8.1}$$

with damping coefficient R (in kg/s). To this system we can add an external force:

$$M\ddot{u} = -Ku - R\dot{u} + F \tag{8.2}$$

where

8.2 The Bow

The bow...

Helmholtz motion..

See fx. `https://www.youtube.com/watch?v=6JeyiM0YNo4`

Characteristic triangular motion (wave shapes?)

8.2.1 Static Friction Models

In static bow-string-interaction models, the friction force is defined as a function of the relative velocity between the bow and the string only. The first mathematical description of friction was proposed by Coulomb in 1773 [**?**]

to which static friction, or *stiction*, was added by Morin in 1833 [?] and viscous friction, or velocity-dependent friction, by Reynolds in 1886 [?]. In 1902, Stribeck found a smooth transition between the static and the coulomb part of the friction curve now referred to as the Stribeck effect [?]. The latter is still the standard for static friction models today.

8.2.2 Dynamic Friction Models

As opposed to less complex bow models, such as the hyperbolic [source] and exponential [source] models, the elasto-plastic bow model assumes that the friction between the bow and the string is caused by a large quantity of bristles, each of which contributes to the total amount of friction.

8.3 Lip-reed

Lip-reed model

8.3.1 Coupling to Tube

Part IV

Interactions

The models described in part II already sound quite convincing on their own. However, these are just individual components that can be combined to approximate a fully functional (virtual) instrument. The following chapters will describe different ways of interaction between individual systems. Chapter 9 describes ways to connect different systems and Chapter 10 describes collision interactions between models.

ε

Newtons third law (action reaction)

Subscripts are needed

Somewhere in this Chapter have a section about fretting and how to generate different pitches using only one string

Interpolation and spreading operators

Using $l_\mathrm{c} = \lfloor x_\mathrm{c}/h \rfloor$ and $\alpha_\mathrm{c} = x_\mathrm{c}/h - l_\mathrm{c}$ is the fractional part the location of interest.

$$I_0(x_\mathrm{c}) = \begin{cases} 1, & \text{if } l = l_\mathrm{c}, \\ 0, & \text{otherwise} \end{cases} \tag{8.3}$$

$$I_1(x_\mathrm{c}) = \begin{cases} (1-\alpha_\mathrm{c}), & \text{if } l = l_\mathrm{c} \\ \alpha_\mathrm{c}, & \text{if } l = l_\mathrm{c}+1 \\ 0 & \text{otherwise} \end{cases} \tag{8.4}$$

$$I_3(x_\mathrm{c}) = \begin{cases} ... \end{cases} \tag{8.5}$$

The following identity is very useful when solving interactions between components:

$$\langle f, J_p(x_\mathrm{c}) \rangle_\mathcal{D} = I_p(x_\mathrm{c}) f \tag{8.6}$$

Chapter 9

Connections

Something about connections

Pointlike $\delta(x - x_\text{c})$ or distributed E_c

9.1 Rigid connection

The simplest connection is Forces should be equal and opposite.

If component a is located 'above' component b, and their relative displacement is defined as $\eta = a - b$, then a positive η is going to have a negative effect on a and a positive effect on b and vice-versa. This is important for the signs when adding the force terms to the schemes.

9.2 Spring-like connections

9.2.1 Connection with rigid barrier (scaled)

Consider the (scaled) 1D wave equation with an additional force term F^n

$$\delta_{tt}u_l^n = \gamma^2\delta_{xx}u_l^n + J(x_\text{c})F^n \tag{9.1}$$

where

$$F^n = -\omega_0^2\mu_{t\cdot}\eta^n - \omega_1^4(\eta^n)^2\mu_{t\cdot}\eta^n - 2\sigma_\times\delta_{t\cdot}\eta^n \tag{9.2}$$

and

$$\eta^n = I(x_\text{c})u_l^n. \tag{9.3}$$

To obtain F^n, an inner product of scheme (9.1) needs to be taken with $J(x_\text{c})$ over domain $\mathcal{D}$ which, using identity (8.6) yields

$$\delta_{tt}I(x_\text{c})u_l^n = \gamma^2 I(x_\text{c})\delta_{xx}u_l^n + \underbrace{I(x_\text{c})J(x_\text{c})}_{\|J(x_\text{c})\|_\mathcal{D}^2}F^n. \tag{9.4}$$

As u is connected to a rigid barrier according to (9.3), a shortcut can be taken and Eqs. (9.2) and (9.3) can be directly substituted into Eq. (9.4) to get

$$\delta_{tt}\eta^n = \gamma^2 I(x_\text{c})\delta_{xx}u_l^n + \|J(x_\text{c})\|_\mathcal{D}^2 \left(-\omega_0^2\mu_{t\cdot}\eta^n - \omega_1^4(\eta^n)^2\mu_{t\cdot}\eta^n - 2\sigma_\times\delta_{t\cdot}\eta^n\right). \tag{9.5}$$

and solved for η^{n+1}:

$$\begin{aligned}&\Big(1 + \|J(x_\text{c})\|_\mathcal{D}^2 k^2[\omega_0^2/2 + \omega_1^4(\eta^n)^2/2 + \sigma_\times/k]\Big)\eta^{n+1}\\ &\quad = 2\eta^n - \Big(1 + \|J(x_\text{c})\|_\mathcal{D}^2 k^2[\omega_0^2/2 + \omega_1^4(\eta^n)^2/2 - \sigma_\times/k]\Big)\eta^{n-1}\\ &\qquad + \gamma^2 k^2 I(x_\text{c})\delta_{xx}u_l^n\end{aligned} \tag{9.6}$$

This can then be used to calculate F^n in (9.2) and can in turn be used to calculate u_l^{n+1} in (9.1).

9.2.2 String-plate connection

In this example, let's consider a string connected to a plate using a nonlinear damped spring. This could be interpreted as a simplified form of how guitar string would be connected to the body.

Continuous

The systems in isolation are as in (4.2) and (6.4), but with an added force term:

$$\partial_t^2 u = c^2\partial_x^2 u - \kappa_\text{s}^2\partial_x^4 u - 2\sigma_{0,\text{s}}\partial_t u + 2\sigma_{1,\text{s}}\partial_t\partial_x^2 u - \delta(x - x_\text{c})\frac{f}{\rho_\text{s}A} \tag{9.7a}$$

$$\partial_t^2 w = -\kappa_\text{p}^2\Delta\Delta w - 2\sigma_{0,\text{p}}\partial_t w + 2\sigma_{1,\text{p}}\partial_t\partial_x^2 w + \delta(x - x_\text{c}, y - y_\text{c})\frac{f}{\rho_\text{p}H} \tag{9.7b}$$

where

$$f = f(t) = K_1\eta + K_3\eta^3 + R\dot{\eta} \tag{9.8}$$

and

$$\eta = \eta(t) = u(x_\text{c}, t) - w(x_\text{c}, y_\text{c}, t) \tag{9.9}$$

Discrete

System (9.7) can then be discretised as

$$\delta_{tt}u_l^n = c^2\delta_{xx}u_l^n - \kappa_\text{s}^2\delta_{xxxx}u_l^n - 2\sigma_{0,\text{s}}\delta_{t\cdot}u_l^n + 2\sigma_{1,\text{s}}\delta_{t-}\delta_{xx}u_l^n - J_\text{s}(x_\text{c})\frac{f^n}{\rho_\text{s}A}, \tag{9.10}$$

$$\delta_{tt}w_l^n = -\kappa_\text{p}^2\delta_\Delta\delta_\Delta w_l^n - 2\sigma_{0,\text{p}}\delta_{t\cdot}w_l^n + 2\sigma_{1,\text{p}}\delta_{t-}\delta_{xx}w_l^n + J_\text{p}(x_\text{c}, y_\text{c})\frac{f^n}{\rho_\text{p}H}, \tag{9.11}$$

where

$$f^n = K_1 \mu_{tt}\eta^n + K_3(\eta^n)^2\mu_{t\cdot}\eta^n + R\delta_{t\cdot}\eta^n, \tag{9.12}$$

and

$$\eta^n = I(x_\mathrm{c})u_l^n - I(x_\mathrm{c}, y_\mathrm{c})w_l^n. \tag{9.13}$$

Expansion

System (9.10) can be expanded at the connection location x_c by taking an inner product of the schemes with their respective spreading operators.

9.2.3 Solving for f

9.2.4 Non-dimensional

The scaled system can be written as:

$$\partial_t^2 u = \gamma^2\partial_x^2 u - \kappa_\mathrm{s}^2\partial_x^4 u - 2\sigma_{0,\mathrm{s}}\partial_t u + 2\sigma_{1,\mathrm{s}}\partial_t\partial_x^2 u - \delta(x - x_\mathrm{c})F \tag{9.14}$$

$$\partial_t w = -\kappa_\mathrm{p}^2\Delta\Delta w - 2\sigma_{0,\mathrm{p}}\partial_t w + 2\sigma_{1,\mathrm{p}}\partial_t\partial_x^2 w + \delta(x - x_\mathrm{c}, y - y_\mathrm{c})F \tag{9.15}$$

where

$$F = F(t) = \omega_1^2\eta + \omega_3^4\eta^3 + \sigma_\mathrm{c}\dot{\eta} \tag{9.16}$$

and

$$\eta = \eta(t) = u(x_\mathrm{c}, t) - w(x_\mathrm{c}, y_\mathrm{c}, t) \tag{9.17}$$

$$I(x_\mathrm{c})\delta_{tt}u_l^n = c^2\left(I(x_\mathrm{c})\delta_{xx}u_l^n\right) + I(x_\mathrm{c})J(x_\mathrm{c})F \tag{9.18}$$

Chapter 10

Collisions

Something about collisions

10.1 Classic models

Note that when using Eq. (7.37) in [3]

10.2 Michele's tricks

Part V

Dynamic Grids

Chapter 11

Dynamic Grids

Often in math, you should view the definition not as a starting point, but as a target. Contrary to the structure of textbooks, mathematicians do not start by making definitions and then listing a lot of theorems, and proving them, and showing some examples. The process of discovering math typically goes the other way around. They start by chewing on specific problems, and then generalising those problems, then coming up with constructs that might be helpful in those general cases,and only then you write down a new definition (or extend an old one). - Grant Sanderson (AKA 3Blue1Brown) https://youtu.be/O85OWBJ2ayo?t=359

11.1 Background and Motivation

Simulating musical instruments using physical modelling – as mentioned in Part I– allows for manipulations of the instrument that are impossible in the physical world. Examples of this are changes in material density or stiffness, cross-sectional area (1D), thickness (2D) and size. Apart from being potentially sonically interesting, there are examples in the physical world where certain aspects of the instrument are manipulated in real-time.

check if is still true

Tension in a string is changed when tuning it

Some artists even use this in their performances [11, 13]

The hammered dulcimer is another example where the strings are tensioned over a bridge where one can play the string at one side of the bridge, while pushing down on the same string on the other side [10].

1D:

- Trombone
- Slide whistle
- Guitar strings

  - Fretting finger pitch bend
  - Above the nut [13]
  - Tuning pegs directly [11]
- Hammered dulcimer [10]
- Erhu?

2D:

- Timpani
- Bodhrán: https://youtu.be/b9HyB5yNS1A?t=146
- Talking drum (hourglass drum): https://youtu.be/B4oQJZ2TEVI?t=9
- Flex-a-tone (could also be 1D tbh..): https://www.youtube.com/watch?v=HEW1aG8XJQ

A more relevant example is that of the trombone, where the size of the instrument is changed in order to play different pitches. Modelling this using FDTD methods would require

In his thesis, Harrison points out that in order to model the trombone, grid points need to be introduced

Something about time-dependent variable coefficient Stokes flow: https://arxiv.org/abs/10

Time-varying propagation speed in waveguides: https://quod.lib.umich.edu/cgi/p/pod/idx/fractional-delay-application-time-varying-propagation-speed.pdf?c=icmc;idno=bbp2372.

Special boundary conditions (look at!): Modeling of Complex Geometries and Boundary Conditions in Finite Difference/Finite Volume Time Domain Room Acoustics Simulation (`https://www.researchgate.net/publication/260701231_Modeling_of_Complex_Geometries_and_Boundary_Conditions_in_Finite_DifferenceFinite_Volume_Time_Domain_Room_Acoustics_Simulation`)

11.2 Method

Iterations have been:

- Interpolated boundary conditions
- Linear interpolation

These sections are taken from the JASA appendix

In this appendix, some iterations done over the course of this project will be shown in more detail. In the following, the 1D wave equation with a wave speed of $c = 1470$ m/s, a length of $L = 1$ m, Dirichlet boundary conditions and a sample rate of $f_s = 44100$ Hz is considered, and – through Eq. (**??**) – satisfies the CFL condition with equality. These values result in $N = 30$, or a grid of 31 points including the boundaries. Then, the wave speed is decreased to $c \approx 1422.6$ m/s, i.e., the wave speed that results in $N = 31$ and satisfies the stability condition with equality again.

11.2.1 Full-Grid Interpolation

One way to go from one grid to another is performing a full-grid interpolation [3, Chap. 5]. If the number of points changes according to Eq. (**??**), i.e., if $N^n \neq N^{n-1}$ the full state of the system $(u_l^n, u_l^{n-1}\ \forall l)$ can be interpolated to the new state. See Figure 11.1.

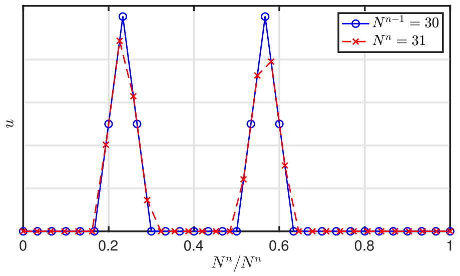


Fig. 11.1: Upsampling u (with an arbitrary state) using (linear) full-grid interpolation with $N^{n-1} = 30$ and $N^n = 31$. The horizontal axis is normalised with respect to N^n.

An issue that arises using this method is that the Courant number λ will slightly deviate from the CFL condition as c changes. Using Eq. (**??**) with L/ck approaching 31 (from below), the minimum value of $\lambda \approx 30/31 \approx 0.9677$. This, employing Eq. (**??**), has a maximum frequency output of $f_{\text{max}} \approx 18,475$ Hz. The Courant number will deviate more for higher values of c and thus lower values for N – for instance, if N approaches 11 (from below), $\lambda \approx 10/11 \approx 0.9091$ and $f_{\text{max}} \approx 16,018$ Hz.

Another problem with full-grid interpolation, is that it has a low-passing effect on the system state, and thus on the output sound. Furthermore, this state-interpolation causes artefacts or 'clicks' in the output sound as the method causes sudden variations in the states.

All the aforementioned issues could be solved by using a (much) higher sample rate and thus more grid points, but this would render this method impossible to work in real time.

11.2.2 Adding and removing Points at the Boundary

To solve the issues exhibited by a full-grid interpolation, points can be added and removed at a single location and leave most points unaffected by the parameter changes. A good candidate for a location to do this is at a fixed

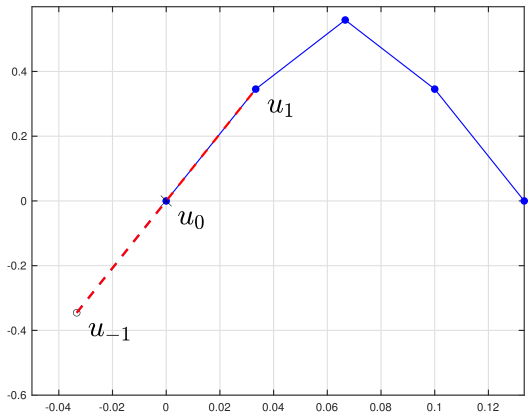


Fig. 11.2: The simply supported boundary condition: both the state and the curvature at the boundary – at $l = 0$ – should be 0.

(Dirichlet) boundary. The state u at this location is always 0 so points can be added smoothly.

As c decreases, h can be calculated according to Eq. (**??**) and decreases as well.

This has a physical analogy with tuning a guitar string. Material enters and exits the neck (playable part of the string) at the nut, which in discrete time means grid points appearing and disappearing at one boundary.

To yield smooth changes between grid configurations, an interpolated boundary has been developed, the possibility of which has been briefly mentioned in [3, p. 145]. The Dirichlet condition in Eq. (2.38a) can be extended to be the simply supported boundary condition:

$$u(x,t) = \frac{\partial^2}{\partial x^2}u(x,t) = 0 \quad \text{where} \quad x = 0, L, \tag{11.1}$$

or, when discretised,

$$u_l^n = \delta_{xx}u_l^n = 0, \quad \text{where} \quad l = 0, N. \tag{11.2}$$

This means that on top of that the state of the boundary should be 0, the curvature around it should also be 0. One can again solve for the virtual grid points at the boundary locations, yielding

$$u_{-1}^n = -u_1^n \quad \text{and} \quad u_{N+1}^n = -u_{N-1}^n. \tag{11.3}$$

This is visualised in Figure 11.2.

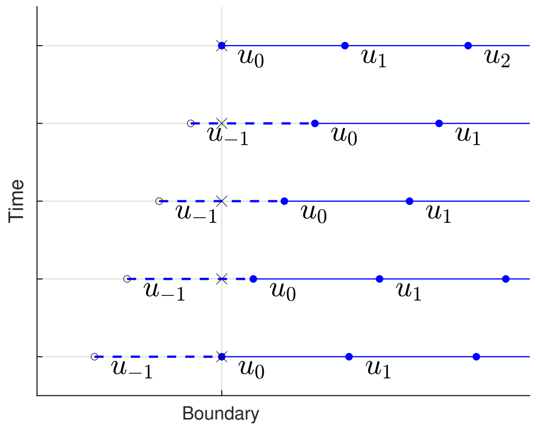


Fig. 11.3: The grid changing over time

If the flooring operation in Eq. (**??**) is removed this introduces a fractional number of grid points.

The by-product of using a fractional N this is that the CFL condition in (2.44) can now always be satisfied with equality no matter what the wave speed is.

An issue with this method is that removing points is much harder than adding.

their interactions change through a change in the grid spacing and wave speed. This interaction, though, is defined by λ which

11.2.3 Cubic interpolation

11.2.4 Sinc interpolation

11.2.5 Displacement correction

The displacement correction can be interpreted as a spring force pulling u_M^n and w_0^n to the average displacement.

$$\begin{aligned} u_M^{n+1} &= 2u_M^n + \lambda^2(u_{M-1}^n - 2u_M^n + u_{M+1}^n) - K\left(u_M^n - \frac{u_M^n + w_0^n}{2}\right) \\ w_0^{n+1} &= 2u_M^n + \lambda^2(w_{-1}^n - 2w_0^n + w_1^n) - K\left(w_0^n - \frac{u_M^n + w_0^n}{2}\right) \end{aligned} \tag{11.4}$$

$$\begin{aligned} u_M^{n+1} &= 2u_M^n + \lambda^2(u_{M-1}^n - 2u_M^n + u_{M+1}^n) + \frac{K}{2}(w_0^n - u_M^n) \\ w_0^{n+1} &= 2u_M^n + \lambda^2(w_{-1}^n - 2w_0^n + w_1^n) - \frac{K}{2}(w_0^n - u_M^n) \end{aligned} \tag{11.5}$$

with $K = K(\alpha)$

$$K = (1-\alpha)^\epsilon. \tag{11.6}$$

11.3 Analysis and Experiments

11.3.1 Interpolation technique

11.3.2 Interpolation range

11.3.3 Location

... where to add and remove points

Using the whole range, we can still add/remove points at the sides.

11.4 Discussion and Conclusion

Part VI

Real-Time Implementation and Control

Real-Time Implementation and Control

It’s all fun and dandy with all these physical models, but what use are they if you can’t control them in real time?!

Chapter 12

Real-Time Implementation

JUCE Give overall structure of code

Implementation of the physical models using FDTD methods

As mentioned in Chapter 1, FDTD methods are used for high-quality and accurate simulations, rather than for real-time applications. This is due to their lack of simplifications.

Usually, `MATLAB` is used for simulating

Here, an interactive application is considered real-time when

Control of the application generates or manipulates audio with no noticeable latency.

Also the application needs to be controlled continuously

Helps to (informally) evaluate the models by interacting with it in a natural way (rather than static parameters)

12.1 MATLAB vs. C++

It is usually a good idea to prototype a physical modelling application in MATLAB for several reasons:

- Easier to debug
  - Plotting functionality
  - No need for memory handling
- Instability due to programming errors

12.1.1 Speed

Here is where the power of C++

12.1.2 Syntax

Indexing in Matlab is 1-based, meaning that the index of a vector starts at 1. If `u` is a vector with 10 elements, the first element is retrieved as `u(1)` and the last as `u(10)`. C++, on the other hand, is 0-based and retrieving the first and last element of a size-10 vector happens through `u[0]` and `u[9]` respectively.

12.2 Do's and don'ts in Real-Time FD schemes

Some of the things I learned (the hard way)...

- Create a limiter
- Structure your application into classes
- Use pointer switches
- Comment your code (hehe)

Create a limiter

Programming errors happen. To save your speakers, headphones or – most importantly – your ears, create a limiter.

```
double limit (double val)
{
    if (val < -1)
    {
        val = -1;
        return val;
    }
    else if (val > 1)
    {
        val = 1;
        return val;
    }
    return val;
}
```

```
for  i = 0:lengthSound
    uNext = ...
end
```

Use pointer switches

One of the most important things in working with FD schemes for real-time audio applications is to

The maximum number of copy-operations this takes is $2(N+1)$ if the boundaries also need updated in the case of free or Neumann boundary conditions. For a string with simply supported or clamped boundary conditions this can be reduced to $2N$ or $2(N-1)$ respectively, but does not maka big difference in computational time.

A pointer switch, however, only needs 4 copy-operations per iteration as shown in Algorithm 12.1

```
for n = 1:lengthSound
    ...
    uPrev = u;
    u = uNext;
end
```

In C++ this is done using

```
double updateStates()
{
    double* uTmp = u[2];
    u[2] = u[1];
    u[1] = u[0];
    u[0] = uTmp;
}
```

Algorithm 12.1: Implementation of a pointer switch also shown in Figure 12.1b. A temporary pointer is assigned to where the $\mathbf{u}^{n-1}$ pointer is currently pointing at to be able to assign that location in memory to the $\mathbf{u}^{n+1}$ pointer in the end.

Grouping terms and Precalculating coefficients

Generally in implementations of FD schemes the most computationally expensive part of the algorithm is the calculation of the scheme itself. This is due to the rate at which it needs to be updated which usually is 44100 Hz. Graphics can be updated at rates orders of magnitude lower than the audio (say 10-20 Hz) and still be considered smooth enough.

Recall the update equation for the 1D wave update equation in Eq. (2.42)

$$u_l^{n+1} = (2-2\lambda^2)u_l^n - u_l^{n-1} + \lambda^2\left(u_{l+1}^n + u_{l-1}^n\right).$$

Grouping the terms like this allows for the coefficients multiplied onto the grid function at different temporal and spatial indices to be precomputed. This can significantly decrease the amount of operations per sample.

The undamped stiff string FD scheme

$$\delta_{tt}u_l^n = c^2\delta_{xx}u_l^n - \kappa^2\delta_{xxxx}u_l^n, \tag{12.1}$$

can be expanded to an update equation as

$$\begin{aligned} u_l^{n+1} &= 2u_l^n - u_l^{n-1} + \lambda^2\left(u_{l+1}^n - 2u_l^n + u_{l-1}^n\right)\\ &\quad - \mu^2\left(u_{l+2}^n - 4u_{l+1}^n + 6u_l^n + -4u_{l-1}^n + u_{l-2}^n\right) \end{aligned} \tag{12.2}$$

where $\lambda = ck/h$ and $\mu = \kappa k/h^2$.

For implementation purposes there is a better way to write this scheme that reduces the amount of computations. This is done by collecting the terms based on the grid function and pre-calculating the coefficients multiplied onto these. As the schemes are spatially symmetric, "neighbouring points" relative to u_l^n can also be grouped to get

$$u_l^{n+1} = \underbrace{\left(2-2\lambda^2-6\mu^2\right)}_{\texttt{B1}}u_l^n + \underbrace{\left(\lambda^2+4\mu^2\right)}_{\texttt{B2}}\left(u_{l+1}^n + u_{l-1}^n\right)\underbrace{-\mu^2}_{\texttt{B3}}\left(u_{l+2}^n + u_{l-2}^n\right)\underbrace{-}_{\texttt{C1}}u_l^{n-1}. \tag{12.3}$$

These coefficients can then be pre-calculated and do not have to be

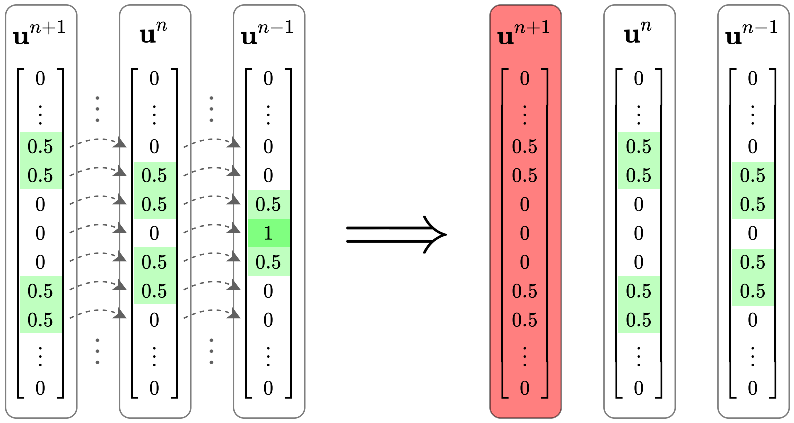
```
//// Constructor ////
B1 = 2.0 - 2.0 * lambdaSq - 6.0 * muSq; // u_l^n
B2 = lambdaSq + 4.0 * muSq;             // u_{l+-1}^n
B3 = -muSq;                             // u_{l+-2}^n
C1 = -1.0;                              // u_l^{n-1}

//// Function where scheme is calculated ////
for (int l = 2; l < N-1; ++l) // clamped boundaries
{
    u[0][l] = B1 * u[1][l]
            + B2 * (u[1][l+1] + u[1][l-1])
            + B3 * (u[1][l+2] + u[1][l-2])
            + C1 * u[2][l];
}
```

Algorithm 12.2: Precalculation

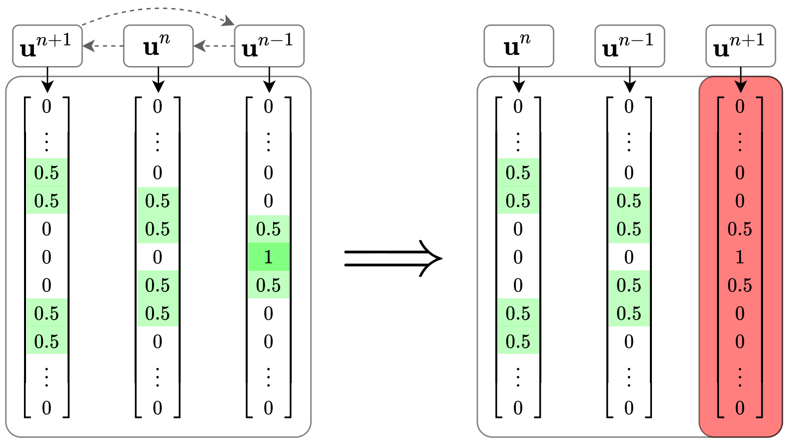
12.3 Graphics

```
void Simple1DWave::paint (juce::Graphics& g)
{
    // clear the background
    g.fillAll (getLookAndFeel().findColour
    (juce::ResizableWindow::backgroundColourId));

    Path stringState = visualiseState (g, 100);

    // choose your favourite colour
    g.setColour (Colours::cyan);

    // visualScaling depends on your excitation
    g.strokePath (visualiseState (g, 100), PathStrokeType(2.0f));

}

Path Simple1DWave::visualiseState (Graphics& g, double visualScaling)
{
    // String-boundaries are in the vertical middle of the component
    double stringBoundaries = getHeight() / 2.0;

    // initialise path
    Path stringPath;

    // start path
    stringPath.startNewSubPath (0, -u[1][0] * visualScaling +
    stringBoundaries);

    double spacing = getWidth() / static_cast<double>(N);
    double x = spacing;

    for (int l = 1; l <= N; l++)
    {
        // Needs to be -u, because a positive u would visually go down
        float newY = -u[1][l] * visualScaling + stringBoundaries;
        if (isnan(x) || isinf(abs(x)) || isnan(u[1][l]) ||
    isinf(abs(u[1][l])))
            std::cout << "Wait" << std::endl;

        // if we get NAN values, make sure that we don't get an
        // exception
        if (isnan(newY))
            newY = 0;

        stringPath.lineTo (x, newY);
        x += spacing;
    }

    return stringPath;
}
```

```
void Simple1DWave::excite()
{
    //// Raised cosine excitation ////

    // width (in grid points) of the excitation
    double width = 10;
    double excitationLoc = 0.2;
    // make sure we're not going out of bounds at the left boundary
    int start = std::max (floor((N+1) * excitationLoc) - floor(width *
    0.5), 1.0);

    for (int l = 0; l < width; ++l)
    {
        // make sure we're not going out of bounds
        // at the right boundary (this does 'cut off'
        // the raised cosine)

        if (l+start > N - 1)
            break;

        u[1][l+start] += 0.5 * (1 - cos(2.0 * double_Pi * l /
    (width-1.0)));
        u[2][l+start] += 0.5 * (1 - cos(2.0 * double_Pi * l /
    (width-1.0)));
    }
}
```



(a) Copying values: $2(N+1)$ operations per iteration.



(b) Pointer switch: 4 operations per iteration.

Fig. 12.1: Updating the state vectors by (a) copying all values individually, or (b) performing a pointer switch. Non-zero values are highlighted in green for clarity. The values of the red vector will be overwritten by the update of the scheme in the next iteration so these values will no longer be used.

Chapter 13

Control

13.1 Sensel Morph

150 Hz

13.2 Phantom OMNI

Part VII

Complete Instruments

Complete Instruments

This part will give several examples of full instrument models that have been developed during the PhD. Chapter 14 shows a three instrument-inspired case-studies using a large-scale modular environment, Chapter 15 describes the implementation of the tromba marina and Chapter 16 that of the trombone.

Chapter 14

Large Scale Modular Physical models

In the paper "Real-Time Control of Large-Scale Modular Physical Models using the Sensel Morph" [A] we presented a modular physical modelling environment using three instruments as case studies.

14.1 Bowed Sitar

- Stiff String
- Thin Plate
- Pluck
- Bow
- Non-linear spring connections

14.2 Dulcimer

- Stiff String
- Thin Plate
- Hammer (simple)
- Non-linear spring connections

14.3 Hurdy Gurdy

- Stiff String
- Thin Plate
- Pluck
- Bow
- Non-linear spring connections

Chapter 15

Tromba Marina

15.1 Introduction

15.2 Physical Model

Using linear (partial) differential operator $\mathcal{L}$

$$\mathcal{L}q = 0 \tag{15.1}$$

where $q(\boldsymbol{x}, t)$

15.2.1 Continuous

$$\mathcal{L}_\text{s} = \rho_\text{s}A\partial_t^2 - T\partial_\chi^2 + E_\text{s}I\partial_\chi^4 + 2\rho_\text{s}A\sigma_{0\text{s}}\partial_t - 2\rho_\text{s}A\sigma_{1\text{s}}\partial_t\partial_\chi^2\,. \tag{15.2}$$

Complete system

Test

15.2.2 Discrete

15.3 Real-Time Implementation

15.3.1 Control using Sensel Morph

15.3.2 VR Application

Chapter 16

Trombone

Published in [H]

16.1 Introduction

Interesting read: https://newt.phys.unsw.edu.au/jw/brassacoustics.html

16.2 Physical Model

Most has been described in Chapter 5

16.2.1 Continuous

Just to save the conversation with Stefan about Webster's equation:

Using operators ∂_t and ∂_x denoting partial derivatives with respect to time t and spatial coordinate x, respectively, a system of first-order PDEs describing the wave propagation in an acoustic tube can then be written as [A]

$$\frac{S}{\rho_0 c^2}\partial_t p = -\partial_x(Sv) \tag{16.1a}$$

$$\rho_0 \partial_t v = -\partial_x p \tag{16.1b}$$

with acoustic pressure $p = p(x,t)$ (in N/m^2), particle velocity $v = v(x,t)$ (in m/s) and (circular) cross-sectional area $S(x)$ (in m^2). Furthermore, ρ_0 is the density of air (in kg/m^3) and c is the speed of sound in air (in m/s). System (16.1) can be condensed into a second-order equation in p alone, often referred to as Webster's equation [?]. Interesting! In NSS it is the acoustic potential right? Can you go from that to a second-order PDE in p? There is a time-derivative hidden there somewhere right? (Just wondering :))Yes, the form in p alone is the one you usually see. You get it by differentiating the first equation, giving you a $\dot{v}$ on the RHS, and then you can substitute the second equation in...I used the velocity potential one because it has direct energy balance properties. Right. So Webster's eq. in p and Ψ are identical (will exhibit identical behaviour), except for the unit of the state variable..?yes that's right...using the velocity potential allows you to do all the energy analysis easily, in terms of physical impedances. But the scheme you get to in the end is the same, just one derivative down. Alright cool! Thanks for the explanation :) For simplicity, effects of viscothermal losses have been neglected in (16.1). For a full time domain model of such effects in an acoustic tube, see, e.g. [**?**].

16.2.2 Discrete

16.3 Real-Time Implementation

Unity??

16.4 Discussion

more for your info, don't think I want to include this: To combat the drift, experiments have been done involving different ways of connecting the left and right tube. One involved alternating between applying the connection to the pressures and the velocity. Here, rather than adding points to the left and right system in alternating fashion, points were added to pressures p and q and velocities v and w in an alternating fashion. Another experiment involved a "staggered" version of the connection where (fx.) for one system (either left or right), a virtual grid point of the velocity was created from known values according to (**??**), rather than both from pressures. This, however, showed unstable behaviour. No conclusory statements can be made about these experiments at this point. $\leftarrow$ which is exactly why I don't want to include this section

As the geometry varies it matters a lot where points are added and removed as this might influence the way that the method is implemented. speculative section coming up The middle of the slide crook was chosen, both because it would be reasonable for the air on the tube to "go away from" or "go towards" that point as the slide is extended or contracted, and because the geometry does not vary there. Experiments with adding / removing grid points where the geometry varies have been left for future work. even more speculative.. $\rightarrow$ It could be argued that it makes more sense to add points at the ends of the inner slides as "tube material" is also added there. This would mean that the system should be split in three parts: "inner slide", "outer slide" and "rest", and would complicate things even more.

check whether all references are used

References

[1] J. Bensa, S. Bilbao, R. Kronland-Martinet, and J. O. Smith, "The simulation of piano string vibration: From physical models to finite difference schemes and digital waveguides," *Journal of the Acoustical Society of America (JASA)*, vol. 114, no. 2, pp. 1095—-1107, 2003.

[2] S. Bilbao, "A modular percussion synthesis environment," in *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx-09)*, 2009.

[3] ——, *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. John Wiley & Sons, 2009.

[4] S. Bilbao, C. Desvages, M. Ducceschi, B. Hamilton, R. Harrison-Harsley, A. Torin, and C. Webb, "Physical modeling, algorithms, and sound synthesis: The ness project," *Computer Music Journal*, vol. 43, no. 2-3, pp. 15–30, 2019.

[5] S. Bilbao, J. Perry, P. Graham, A. Gray, K. Kavoussanakis, G. Delap, T. Mudd, G. Sassoon, T. Wishart, and S. Young, "Large-scale physical modeling synthesis, parallel computing, and musical experimentation: The ness project in practice," *Computer Music Journal*, vol. 43, no. 2-3, pp. 31–47, 2019.

[6] R. Courant, K. Friedrichs, and H. Lewy, "Über die partiellen differenzengleichungen de mathematischen physik," *Mathematische Annalen*, vol. 100, pp. 32–74, 1928.

[7] C. Desvages and S. Bilbao, "Two-polarisation physical model of bowed strings with nonlinear contact and friction forces, and application to gesture-based sound synthesis," *Applied Sciences*, vol. 6, no. 5, 2016.

[8] C. G. M. Desvages, "Physical modelling of the bowed string and applications to sound synthesis," Ph.D. dissertation, The University of Edinburgh, 2018.

[9] N. H. Fletcher and T. D. Rossing, *The Physics of Musical Instruments*. Springer, 1998.

[10] T. L. Glenn, "Amazing hammered dulcimer musician - joshua messick," 2014. [Online]. Available: https://youtu.be/veuGTnzgNRU?t=215

[11] J. Gomm, "Passionflower," 2011. [Online]. Available: https://www.youtube.com/watch?v=nY7GnAq6Znw

[12] B. Hamilton, "Finite difference and finite volume methods for wave-based modelling of room acoustics," Ph.D. dissertation, The University of Edinburgh, 2016.

[13] J. Mayer, "Gravity (Live in L.A.)," 2008. [Online]. Available: https://youtu.be/dBFW8OvciIU?t=284

[14] F. Mittelbach, *The LATEX companion*, 2nd ed. Addison-Wesley, 2005.

[15] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114—117, 1965.

[16] F. Orduña-Bustamente, "Auralization in space and in rooms of arbitrary d dimensions," in *Proceedings of the 14th Sound and Music Computing Conference*, 2017, pp. 250–253.

[17] J. Villeneuve and J. Leonard, "Mass-interaction physical models for sound and multi-sensory creation: Starting anew," in *Proceedings of the 16th Sound and Music Computing Conference*, 2019.

[18] R. Zucker, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. National Bureau of Standards Applied Mathematics Series 55, 1972, ch. 4: Elementary Transcendental Functions, pp. 65–226, Tenth Printing.

Part VIII
Appendix

Appendix A

List of Symbols

The list of symbols found below contains often-used symbols in the thesis in the context that they are normally used. Depending on the context they might carry a different meaning (y being displacement of the lip-reed in Chapter 16 but the vertical spatial coordinate for 2D systems in fx. Chapter 6). Some might also be accompanied by a subscript in the main document

Symbol	Description	Unit
α	Fractional part of	
A	Cross-sectional area of string	m^2
c	Wave speed	m/s
$\frac{d^n}{dt^n}$	n^{th} order derivative with respect to t	-
∂_t^n	n^{th} order partial derivative with respect to t	-
δ_{t+}, δ_{t-}, $\delta_{t\cdot}$	Forward, backward and centred difference in time operator	-
δ_{x+}, δ_{x-}, $\delta_{x\cdot}$	Forward, backward and centred difference in space operator	-
δ_{tt}	Second order difference in time operator	-
δ_{xx}	Second order difference in space operator	-
δ_{xxxx}	Fourth order difference in space operator	-
μ_{t+}, μ_{t-}, $\mu_{t\cdot}$	Forward, backward and centred average in time operator	-
μ_{x+}, μ_{x-}, $\mu_{x\cdot}$	Forward, backward and centred average in space operator	-
μ_{tt}	Second order average in time operator	-
μ_{xx}	Second order average in space operator	-

Symbol	Description	Unit
E	Young's Modulus	Pa (kg·m^{-1}·s^{-2})
f	Force or frequency	N or Hz
f_s	Sample rate	Hz
F	Scaled force	depends on system
h	Grid spacing	m
H	Membrane / Plate thickness	m
I	Area moment of inertia	m^4
l	Spatial index to grid function	-
L	Length	m
k	Time step ($= 1/f_s$)	s
K	Spring coefficient	N/m
κ	Stiffness coefficient	m^2/s (1D) m^4·s^{-2} (2D)
n	Sample index to grid function	-
N	Number of points string	-
$\mathbb{N}^0$	Set of non-negative integers	-
t	Time	s
T	Tension	N (1D) N/m (2D)
u	State variable	m
x	Spatial dimension (horizontal for 2D systems)	m
y	Vertical spatial dimension	m
γ	Scaled wave speed	s^{-1}
λ	Courant number for 1D wave eq. ($= ck/h$)	-
μ	Stiffness free parameter	-
ν	Poisson's ratio	-
η	Relative displacement spring	m
ρ	Material density	kg·m^{-3}
$\lfloor\cdot\rfloor$	Flooring operation	-
$\lceil\cdot\rceil$	Ceiling operation	-

Subscripts

c	Connection
p	Plate

Symbol	Description	Unit
s	String	

Appendix B

List of Abbreviations

Abbreviation	Definition
1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
DoF	Degrees of freedom
Eq.	Equation
Eqs.	Equations
FD	Finite-difference
FDTD	Finite-difference time-domain
ODE	Ordinary differential equation
PDE	Partial differential equation

Appendix C

Code Snippets

C.1 Mass-Spring System (Section 2.3)

```
%% Initialise variables
fs = 44100;              % sample rate [Hz]
k = 1 / fs;              % time step [s]
lengthSound = fs;        % length of the simulation (1 second) [samples]

f0 = 440;                % fundamental frequency [Hz]
omega0 = 2 * pi * f0;    % angular (fundamental) frequency [Hz]
M = 1;                   % mass [kg]
K = omega0^2 * M;        % spring constant [N/m]

%% initial conditions (u0 = 1, d/dt u0 = 0)
u = 1;
uPrev = 1;

% initialise output vector
out = zeros(lengthSound, 1);

%% Simulation loop
for n = 1:lengthSound

    % Update equation Eq. (2.35)
    uNext = (2 - K * k^2 / M) * u - uPrev;

    out(n) = u;

    % Update system states
    uPrev = u;
    u = uNext;
end
```

C.2 1D Wave Equation (Section 2.4)

```matlab
%% Initialise variables
fs = 44100;          % Sample rate [Hz]
k = 1 / fs;          % Time step [s]
lengthSound = fs;    % Length of the simulation (1 second) [samples]

c = 300;             % Wave speed [m/s]
L = 1;               % Length [m]
h = c * k;           % Grid spacing [m] (from CFL condition)
N = floor(L/h);      % Number of intervals between grid points
h = L / N;           % Recalculation of grid spacing based on integer N

lambdaSq = c^2 * k^2 / h^2; % Courant number squared

% Boundary conditions ([D]irichlet or [N]eumann)
bcLeft = "D";
bcRight = "D";

%% Initialise state vectors (one more grid point than the number of
    intervals)
uNext = zeros(N+1, 1);
u = zeros(N+1, 1);

%% Initial conditions (raised cosine)
loc = round(0.8 * N);        % Center location
halfWidth = round(N/10);     % Half-width of raised cosine
width = 2 * halfWidth;       % Full width
rcX = 0:width;               % x-locations for raised cosine

rc = 0.5 - 0.5 * cos(2 * pi * rcX / width); % raised cosine
u(loc-halfWidth : loc+halfWidth) = rc; % initialise current state

% Set initial velocity to zero
uPrev = u;

% Range of calculation
range = 2:N;

% Output location
outLoc = round(0.3 * N);

%% Simulation loop
for n = 1:lengthSound

    % Update equation Eq. (2.42)
    uNext(range) = (2 - 2 * lambdaSq) * u(range) ...
        + lambdaSq * (u(range+1) + u(range-1)) - uPrev(range);

    % boundary updates Eq. (2.46)
    if bcLeft == "N"
        uNext(1) = (2 - 2 * lambdaSq) * u(1) - uPrev(1) ...
        + 2 * lambdaSq * u(2);
    end

    % Eq. (2.47)
    if bcRight == "N"
        uNext(N+1) = (2 - 2 * lambdaSq) * u(N+1) - uPrev(N+1) ...
        + 2 * lambdaSq * u(N);
    end

    out(n) = u(outLoc);

    % Update system states
    uPrev = u;
    u = uNext;
end
```

Part IX

Papers

Paper Errata

Here, some errors in the published papers will be listed:

Real-Time Tromba [D]

- The minus sign in Eq. (28) (and thus Eqs. (31) and (35)) should be a plus sign.
- $\sigma_{1,\text{s}}$ in Eq. (21) should obviously be $\sigma_{1,\text{p}}$
- the unit of the spatial Dirac delta function δ should be m^{-1}

DigiDrum [F]

- σ_0 and σ_1 should be multiplied by ρH in order for the stability condition to hold.
- stability condition is wrong. Should be:

$$h \geq \sqrt{c^2k^2 + 4\sigma_1 k + \sqrt{(c^2k^2 + 4\sigma_1 k)^2 + 16\kappa^2 k^2}} \tag{1}$$

- Unit for membrane tension is N/m.

Dynamic grids [G]

- Reference in intro for 'recently gained popularity' should go to [4]
  Note: not really an error, but should be changed before resubmission

Paper A

Real-Time Control of Large-Scale Modular Physical Models using the Sensel Morph

Silvin Willemsen, Nikolaj Andersson, Stefania Serafin and Stefan Bilbao

The paper has been published in the *Proceedings of the 16th Sound and Music Computing (SMC) Conference*, pp. 275–280, 2019.

changed the template here, should check if it's ok like this

Abstract

Here is an abstract.

A.1 Introduction

Here is an introduction [14].

A.2 Conclusion

Here is the conclusion.

Paper B

Physical Models and Real-Time Control with the Sensel Morph

Paper C

Real-Time Implementation of an Elasto-Plastic Friction Model applied to Stiff Strings using Finite Difference Schemes

Paper D

Real-time Implementation of a Physical Model of the Tromba Marina

Paper E

Resurrecting the Tromba Marina: A Bowed Virtual Reality Instrument using Haptic Feedback and Accurate Physical Modelling

Paper F

DigiDrum: A Haptic-based Virtual Reality Musical Instrument and a Case Study

Paper G

Dynamic Grids for Finite-Difference Schemes in Musical Instrument Simulations

Paper H

A Physical Model of the Trombone using Dynamic Grids for Finite-Difference Schemes