UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN
ICT FOR INTERNET AND MULTIMEDIA

"Numerical Simulation of the Acoustic Response of an Ancient Roman Brass Instrument"

Relatore: Prof. Antonio Rodà

Laureando/a: Aleksandra Matla

ANNO ACCADEMICO 2023 – 2024

Data di laurea 03.07.2024

UNIVERSITÀ DEGLI STUDI DI PADOVA

SCUOLA DI INGEGNERIA
CORSO DI LAUREA MAGISTRALE IN ICT FOR INTERNET AND MULTIMEDIA

Dipartimento di Ingegneria dell'Informazione

TESI DI LAUREA MAGISTRALE IN
ICT FOR INTERNET AND MULTIMEDIA
(Laurea magistrale DM 270/04)

Numerical Simulation of the Acoustic Response of an Ancient Roman Brass Instrument

Supervisor: *Prof. Antonio Rodà*

Student: *Aleksandra Matla*

ANNO ACCADEMICO 2023-2024 03 JULY 2024

Contents

Abstract

In this thesis, I present the development of a computational acoustic model of an ancient Roman tube that can be compared to a trumpet, using the MATLAB environment. The idea is to recreate the sound characteristics and functioning of this historical instrument, which has implications for both archaeological acoustics and the field of digital musicology.

I started by performing a vast research of currently existing methods of modelling the mouthpiece and the movement of the lips, further diving into modelling the instrument itself. Using the digital waveguide and wave equations, I have managed to create a model of the tube simulating the behaviour of a brass instrument.

With the use of the book *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics* by Stefan Bilbao[1] I managed to create a segmented model of the tube using the digital waveguide synthesis methods, as well as scattering matrices in order to properly model the wave reflection and transition from one segment to another.

Later I tested the created MATLAB code with different input bores and bell shapes and generated its behaviour with different reflection parameters and configurations.

Additionally, for every shape, a test with the .wav sound of a mouthpiece was created in order to assess the behaviour of the model with a real-world looking sound wave as input.

At the end, a proposition of a further development of the model is presented, with different conditions and implications noted.

The full code can be found at: `https://gitlab.dei.unipd.it/pitteri.giulio/RomanTube`

This research not only contributes to the understanding of instrument modelling but also offers a framework for virtual reconstruction of lost or inaccessible musical instruments.

Chapter 1

Introduction

1.1 Sound

Sound is a vibration propagated through a medium, i.e., gas, liquid or solid. On the other hand, from the biological point of view, sound is a reception of waves and the perception of them by the brain. But not all the waves are being processed by the brain. Human brain can only perceive waves that are between frequencies of 20 Hz and 20 kHz, or, as we call them, the audible frequency range. When considering the air at normal atmospheric pressure, these frequencies represent sound waves with wavelengths between 17 meters to 1,7 centimetres. Shorter wavelengths are called ultrasounds, whereas longer wavelengths are called infrasounds, but since these are not audible to humans we aren't going to dive into them.

According to the ANSI/ASA standard, sound is defined as an *"oscillation in pressure, stress, particle displacement, particle velocity, etc., propagated in a medium with internal forces (e.g., elastic or viscous), or the superposition of such propagated oscillation"*[2].

In musical instruments, it's the vibrations that create a sound, and in most cases the music is the result of the combined effect of two or more types of vibrations (mechanical, acoustical, or electrical)[3].

1.2 The Instrument

Figure 1.1: A close - up of a trumpet.

The trumpet is a brass instrument, in which the vibration is created by the blow of wind from the lips of a trumpeter. A wave is being created by an embouchure, i.e. by blowing air through slightly separated lips. They produce a standing wave inside the instrument that is a 'buzz' like sound[4]. The first trumpets came in use before 1500 BC and we can recognize them in drawings of ancient Egyptians, Chinese and Scandinavians, although those instruments back then were rather different from the modern ones. They were primarily made from animal horns, emptied inside and opened at the end, without any valves or holes, so the pitch was modulated by the player. Since then, trumpets went through quite a metamorphosis changing different materials: wood, bamboo, bark, clay, human bone; and different shapes: long cylindrical body in between a small circular mouthpiece and a wider sloping horn; ending with the current aspect - a long body bent twice, with three valves and made of brass. Through history, trumpets had different uses, including military ones, where they were used to signal attacks or manoeuvres to the soldiers, religious ceremonies, long distance communications and, last but not least, music[5].

"Lip-driven wind instruments have a very long history, dating back to those made from hollow plant stems, seashells and animal horns. (Carse, 1939; Baines, 1966); even metal trumpets roughly similar to those of the present day existed as long ago as Roman times"[3].

1.3 A Brief History of Physical Modelling

The history of physical modeling is a fascinating journey through time, showcasing the evolution of human understanding and technological advancement in simulating and understanding the physical world.

Physical modeling has been an integral part of engineering and architectural design since ancient times. The use of scaled models can be traced back to civilizations such as the Egyptians and Greeks, who created miniature prototypes to plan and visualize structures like temples and theaters[6].

During the Renaissance, Leonardo da Vinci famously employed physical models to study various hydraulic problems, demonstrating an early understanding of the importance of scale and material properties in simulating real-world phenomena[7].

The 17th century saw significant advancements in physical modeling with scientists like Galileo Galilei, who used models to validate his theories on mechanics and motion. This period marked the beginning of a more systematic approach to physical modeling, integrating mathematical principles with empirical observations[6].

In the 19th century, physical models became crucial in explaining new concepts in biology and physiology. Everett Mendelsohn's work highlights the role of physical models in elucidating physiological concepts during this era[8].

The early 20th century witnessed a surge in the use of physical models for complex engineering projects. Notable examples include the models used in the design of the Boulder Dam and the Conway and Britannia tubular bridges, where models played a pivotal role in testing and refining design concepts before actual construction[6].

With the advent of digital technology, physical modeling has evolved into a blend of traditional techniques and computer simulations. The development of physical modeling synthesis in the late 1980s, particularly the Karplus-Strong algorithm and digital waveguide synthesis by Julius O. Smith III, marked a turning point in the field. These methods allowed for the efficient computation of

waveforms to simulate musical instruments and other sound sources[9]. Today, physical models continue to be used alongside numerical simulations to design, test, and validate engineering projects. They serve as a tangible link between theory and practice, providing insights that are sometimes difficult to obtain through computational methods alone.

1.4 Modelling the Instrument

In order to be able to model a musical instrument while it produces sound, i.e. to simulate a realistic music note produced by this instrument, one must recreate the timbre created in the vibrating air-column[10]. A timbre is sometimes referred to as a tone colour or tone quality. It is one of the reasons we are able to perceive sounds as different, which gives us a unique character of the sound. With timbre, we are able to identify or distinguish different instruments when the same note is played by them. It is created by the excitation of different harmonics or overtones with a sound, in particular by the high-pitched vibrations that are additionally produced on top of the fundamental frequency. The strength and distribution of these sounds is what creates the timbre.

The main problem with the creation of a model of a trumpet is its complexity. The amount of parameters that ought to be controlled create a vast set of configurations of a virtual trumpet. Therefore many literature sources focus mainly on analysing the behaviour of a simple brass instrument and the creation of its model. The main principles of their work can be divided into the same components for sound creation in every instrument that belongs to the brass family. From a simplistic point of view, we can divide a brass instrument into three parts: the lips, the air column and the airflow, that combines the former two.

The lips act as a mechanic resonator and are usually modelled as a mass attached to a spring[11, 12]. It is so because the player puts the lips together and forces them to periodically open and close. This approximation might not be accurate, sine the lips are much more complex in movement and build. We have to take into account the soft tissue, the muscles and the control of them by the player - the tension, the shape in which they are put and their interaction with the rim of the mouthpiece, the teeth, the other lip and the airflow. The movements are also quite different from the model - lips can move in all three dimensions, while the model only assumes a two dimensional movement. However, J.S. Cullen has proved that the horizontal motion of the lips can be neglected, because it has a rather negligible effect on the air flow. The air flow is determined by the total lip opening[13]. The air column then acts just as an acoustic resonator.

1.5 Digital Waveguide

Digital waveguide synthesis is a common tool used for the synthesis of audio, particularly in physical modeling synthesizers. It's based on efficient computational models that simulate the physical media through which acoustic waves propagate, which makes digital waveguides a major part of most modern physical modeling synthesizers.

Digital waveguide synthesis models are based on the principle of sampled acoustic traveling waves. These models replicate the geometry and physical properties of an acoustic system, such as a violin string or a flute pipe, to generate sound digitally[14, 15].

A typical digital waveguide consists of:

- Bidirectional delay lines - representing the path along which the waves travel back and forth,

much like the vibrations along a string,

- Digital filters - that simulate the frequency-dependent losses and mild dispersion that occur in the medium.
- Nonlinear elements - sometimes included to model more complex interactions within the waveguide[14].

The working of a digital waveguide can be divided into four sections:

1. Traveling Waves - the core idea is to use delay lines to simulate the traveling waves that occur in real-world instruments. For example, in a vibrating string, there are waves moving in both directions along the string's length.
2. Reflection and Termination - at the boundaries, such as the ends of a string or the openings of a tube, these waves reflect back, creating standing waves that produce the sound we hear. In a digital waveguide, this reflection is simulated by the delay lines looping back on themselves.
3. Wave Impedance and Scattering - changes in wave impedance along the waveguide cause signal scattering. This is modeled by changes in the delay lines and filters, which alter the characteristics of the traveling waves.
4. Physical Outputs - while traveling waves are great for simulation, they aren't directly measurable in the physical world. To get physical variables like force, pressure, or velocity, the digital waveguide sums the components of the traveling waves[14].

Digital waveguides are a major part of most modern physical modeling synthesizers because they can efficiently model the behavior of various musical instruments. They are also used in virtual reality and gaming to simulate realistic sounds in three-dimensional spaces.

Chapter 2

State of the Art

In order to be able to create a model of a trumpet it's required to understand the build and working of such an instrument. Therefore a review of literature was performed. Since the literature does not focus on a trumpet instrument itself, most of the articles and books were regarding generic brass instruments.

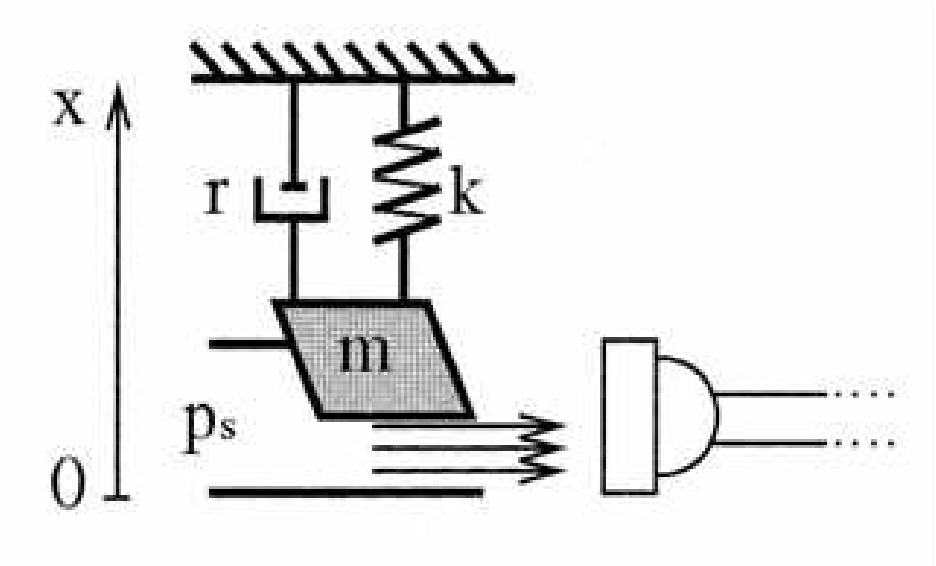


Figure 2.1: Simplified lip model for trumpet-like instruments[1, 16]. In the figure only the upper lip oscillation is considered. The lip is a single mass-spring-damper system with mass m, spring constant k, and damping coefficient r. The bore is approximated as a straight cylindrical tube with area A. We consider a jet of air below the lip.

I started my thesis research by studying the book *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics* by Stefan Bilbao[1]. I started from the beginning, getting familiarized with the sound synthesis and physical modelling, going through additive synthesis, subtractive synthesis, wavetable synthesis, AM and FM synthesis and other methods. Further, I delved into physical models, especially the lumped mass-spring network (see Figure 2.2), which I later used to understand the mechanics of the lips. Starting from easy networks, as can be seen on

Figure 2.1, I later proceeded towards a more elaborated one with a more detailed representation, as presented on Figure 2.3.

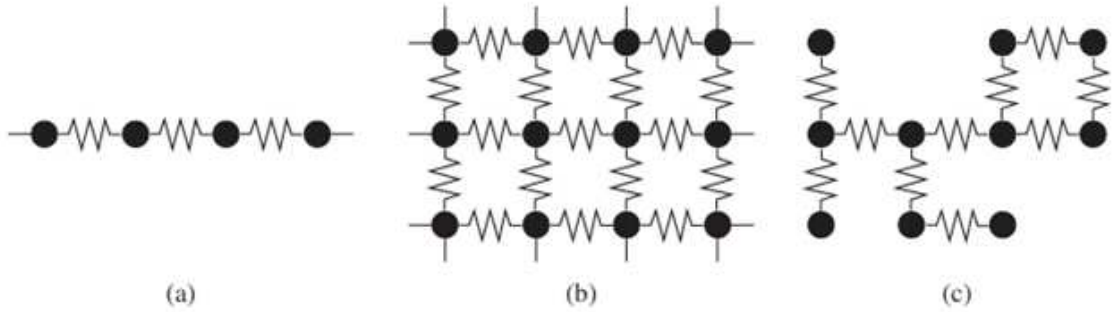


Figure 2.2: Lumped mass-spring networks: (a) in a linear configuration corresponding to a model of lossless string; (b) in a 2D configuration corresponding to a model of a lossless membrane; (c) an unstructured network, without a distributed interpretation[1].

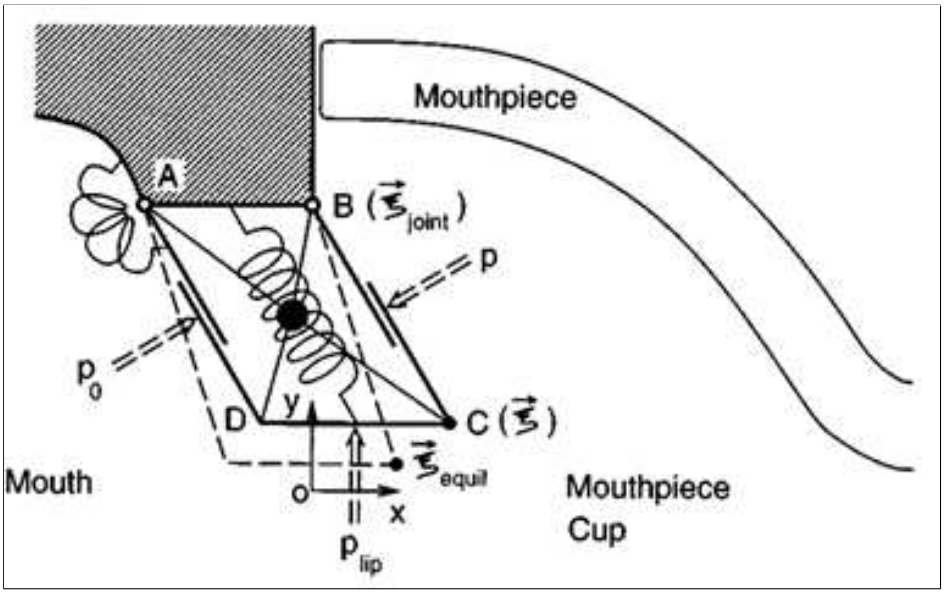


Figure 2.3: Two dimensional lip vibration model. The lips are modelled as a mass on a spring. The model simultaneously executes both swinging and stretching motion[17].

Next, I delved into modal synthesis and digital waveguides, which I later used in modelling the instrument. I proceeded to study physical modelling, starting from abstract synthesis and ending on complex musical systems. Furthermore, I deepened my knowledge regarding the basics of DSP, time series and difference operators.

I then proceeded to study the oscillator chapter, with main focus on the simple harmonic oscillator, finite difference scheme, lumped-mass spring network and sources of oscillation. This chapter introduced programming exercises in MATLAB, which I then proceeded to solve. Some of them required me to better utilize memory space, therefore splitting the proposed code into smaller chunks of code, some of them required me to generate outputs according to given schemes; other ones presented me with the challenge of programming a situation from scratch. Next I skimmed through the chapters

regarding grid functions and finite difference operators in 1D, 1D wave equation, linear bar and string vibration, and nonlinear string vibration. Then I proceeded to focus on the chapter about acoustic tubes. Starting from Webster's equation and simple tubes, I proceeded to understanding complex behaviours and different shapes of tubes. The book then proceeded to explain the vocal tract and speech synthesis, which I also analysed, since some of the mechanics would be similar. I've gone through glottal excitation, formants, wall vibration and loss, scattering methods and finite difference schemes, finally entering the reed and brass instruments chapters. After solving the problems and exercises of these chapter, I moved on into searching for a more specialized research papers connected to modelling a trumpet. The exercises of this chapter focused mainly on the vocal synthesis and vocal tracts, but they did include parts of the previous chapters, so they were a useful learning experience of how to implement all the things I have learned before.

2.1 Modelling and Behaviour of Trumpet and Trombones

While performing a vast research of the papers regarding brass instruments or trumpets, I stumbled upon a PhD Thesis by Janelle Resch titled *"Physical Modelling and the Associated Acoustic Behaviour of Trumpets and Trombones"*[18].

I spent quite a lot of time analysing this thesis, since the MATLAB code was enclosed. First, I gave a quick look at the contents of the thesis, and then I went straight to the MATLAB code, since I knew Janelle did a lot more in this thesis than was unrelated to my work. After a brief analysis of the MATLAB code I realized that without the input data I wouldn't be able to use the code, therefore I contacted her thesis supervisors, Professor Lilia Krivodonova and Professor John Vanderkooy, to obtain Janelle Resch's contact information in order to obtain the data. Thanks to them, I managed to contact Janelle and ask her for the data. Due to some problems on her side, I had to wait a little over a month to receive the data, but in the end I have successfully received it. In the meantime, I started to analyze her thesis meticulously. Her work gave me hope, since the abstract said:

"Accurately modelling the production of realistic musical notes in brass instruments is no easy task. (...) In this thesis, we attempt to accurately model the timbre of musical notes produced on the trumpet and trombone and study the associated acoustic behaviours of both instruments. (...)."

Unfortunately, in the end it didn't provide what I expected it to, but it was still useful to dive into it. It explained a lot of processes and phenomena quite neatly. It talked about the basics of a trumpet and trombone, its build and sound production in them.

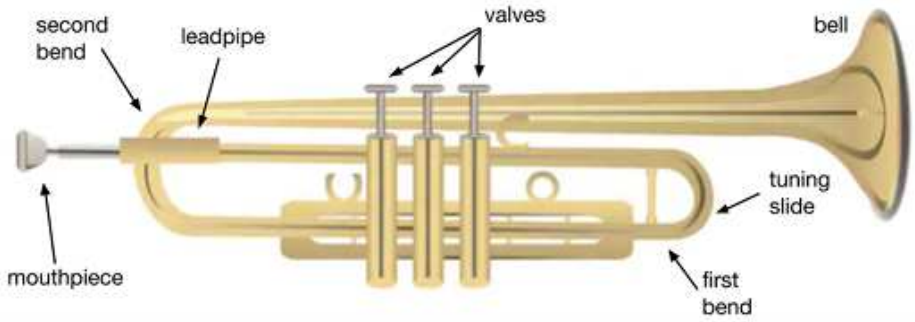


Figure 2.4: An image of a trumpet with labelled the most important parts of it[18].

Then the thesis focused on the experiments they have performed. They have measured the sound pressure with different microphones on the mouthpiece-shank, before the first bend and outside the bell along the central axis of a trumpet and a trombone, as can be seen in Figure 2.5. Next, they have measured the sound pressure of a trumpet and a trombone, by placing a single (for some experiments a couple) accelerometer together with the microphones using beeswax on the outside of the instrument bell near the rim (in the case with two accelerometers, they were placed orthogonal to each other - see Figure 2.6).

Figure 2.5: Microphone placement on the trumpet[18].

Figure 2.6: Accelerometer placement on the bell of the trumpet[18]

They have taken these measures for a total of 33 different settings (different volume, notes and usage of the valves). The results were obtained in form of pressure as a function of time, as shown in Figure 2.7.

As we can see, the output is in milliseconds, therefore it's not quite useful in my work. Nonetheless, I continued to follow the work, as I was hoping the MATLAB code would ease my workload. There was a lot of research done connected to the accelerometers measurements, but since none of it concerned my work, I won't mention it further. Next chapter was connected to mathematical and numerical fundaments. I got acquainted with them briefly, since I doubted that there would be any

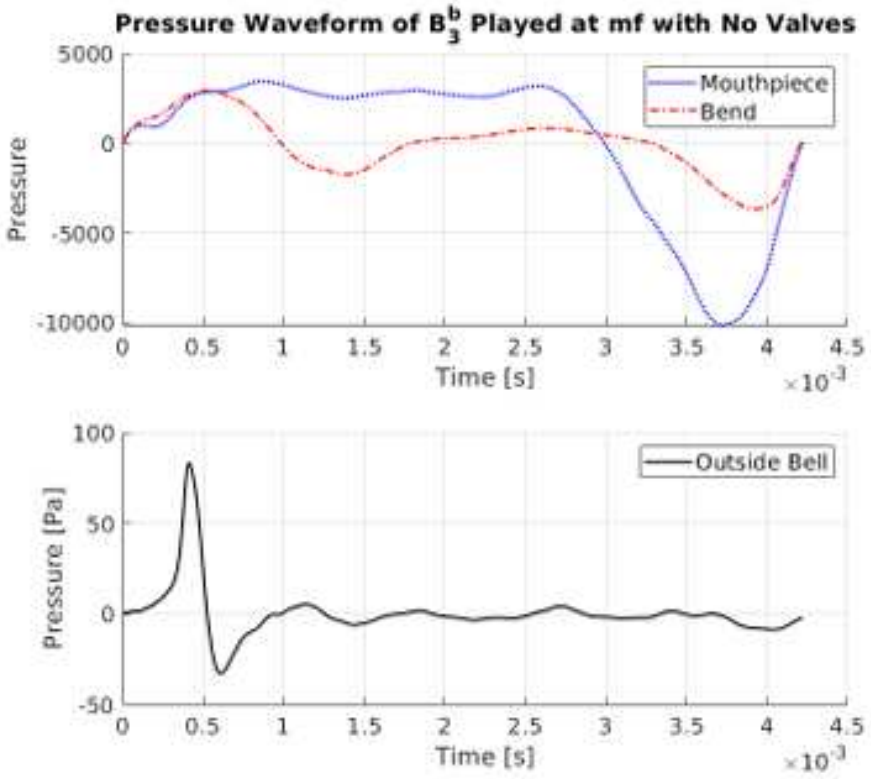


Figure 2.7: Waveforms of pressure vs time of the B3b note played at mezzo forte without any valves compressed[18].

need for equations related to the Galerkin method. Next I skidded through the vibroacoustic and thermoviscous effects and numerical experiments, where it was mentioned that the size and diameter of the bore is usually neglected in the papers. After a thorough examination, it was proven that the radius of the trumpet bore, that is located 24 cm away from the radius of the mouthpiece-shank part is 1,77 times bigger, which actually has a lot of impact on the wave propagation inside the instrument, as can be seen in Figure 2.8.

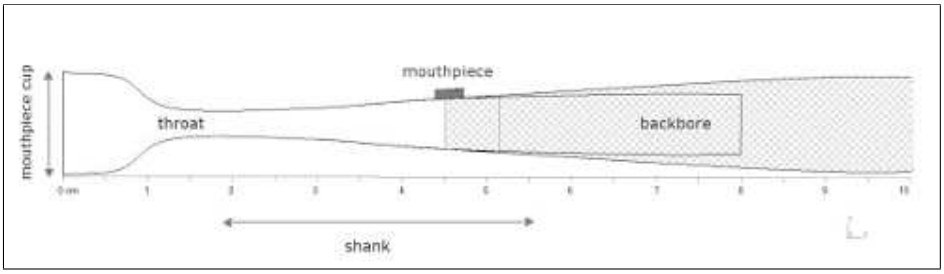


Figure 2.8: A diagram of the throat - mouthpiece - bore part of the trumpet (first 10 cm of the trumpet).

For most of these topics a measurement and a simulation was made, which I will not discuss, since it's not relevant for my thesis.

After getting acquainted with the work, I put all my efforts into running the MATLAB code at-

tached. After I was given the data, I started cleaning up the code and adjusting it. It required to change some lines of code, change variables put in proper references (i.e. following the code instructions: *Read in pressure wanted, e.g.: let x be column wanted*). Unfortunately, after cleaning up the code I found out I did not have all the data required to run it. Nevertheless, I found a way, without consulting the author, to obtain this data from the plot figures. Using a very clever online tool called WebPlotDigitizer I was able to obtain the needed data. Now, with the data at hand I was able to run the code. Unfortunately, after analysing the exact structure of the code and the way it works, I realized that the code didn't actually produce sound, it only merely analyzes the input data, performs filter operation etc. and outputs plots. I had hopes that it might actually produce some sound as it was stated in the abstract, but sadly I have been disappointed.

I moved on to look for other papers that mentioned modelling a brass instrument or a trumpet, preferably in MATLAB, but also mathematical solutions were acceptable, since basing my work on them I would be able to create the model myself.

2.2 Lip Model

The *Trumpet sound simulation using a two-dimensional lip vibration model* by Seiji Adachi and Masa-ski Sato[17] introduced an interesting lip model. It started with comparing a three different lip vibration configuration models comprising of p_0 as pressure in the upstream region of the valve (blowing pressure), p as the pressure downstream region of the valve (mouthpiece pressure), in configurations as follows: an inwardly striking valve, which closes in incremented blowing pressure p_0; outwardly striking valve, which opens further as p_0 increases; a retracting valve that moves laterally to the direction of the flow. These configurations are presented in Figure 2.9.

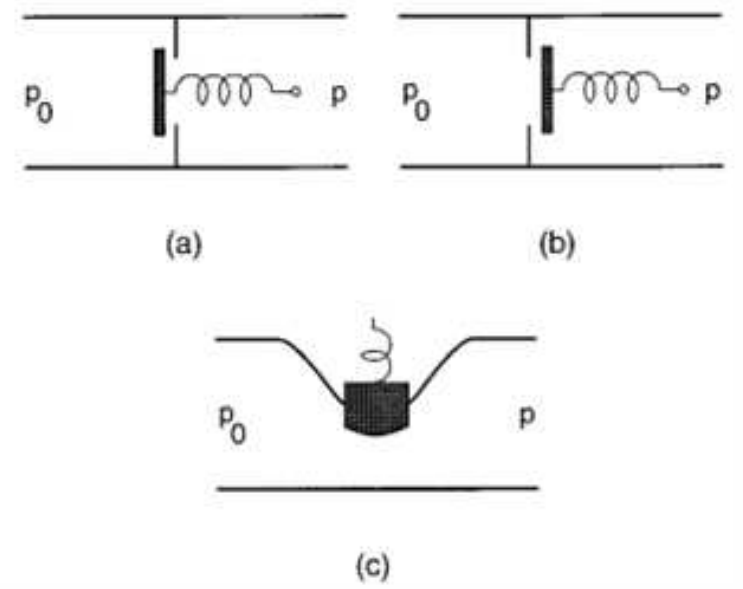


Figure 2.9: Three different configurations of a pressure-controlled valve in an acoustic tube - (a) an inwardly striking valve; (b) an outwardly striking valve and (c) a retracting valve that moves laterally to the direction of the flow[17].

The model was further expanded into a more complex two-dimensional scheme shown in Figure 2.10.

The lips were assumed to work like a simple mechanic oscillator that consists of one mass, stiffness and damping. The mass m is assumed to be localised at the centre of the lip with two springs - one for swinging motion and one for the stretching motion. The lip motion equations are given in Equations 2.1, 2.2, 2.3 and 2.4.

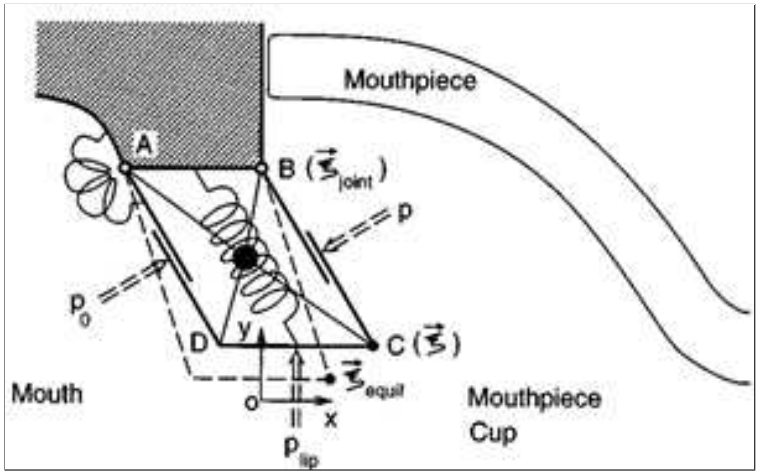


Figure 2.10: A two-dimensional lip vibration model with the assumption of upper and lower lips being symmetrical[17].

$$\frac{1}{2}m\frac{d^2\xi}{dt^2} = -\frac{1}{2}\sqrt{\frac{m}{k}}\frac{d\xi}{dt} + F_{store} + F_d + F_{Bernoulli} \tag{2.1}$$

where:

- m - mass,
- ξ - two-dimensional vector that assigns the position of the tip of the lip,
- k - stiffness factor,
- Q - quality factor,

and

$$F_{restore} = -\frac{1}{2}k(\xi - \xi_{equil}) \tag{2.2}$$

$$F_{ap} = \frac{\delta p}{\delta x}P(\xi - \xi_{joint})^{\perp} \tag{2.3}$$

$$F_{Bernoulli} = b\ d\ p_{tip}\ e_y \tag{2.4}$$

where:

- ξ_{equil} - position of the lips at rest,

- b - lip width,
- e_y - unit vector along the y axis.

Further on, the paper suggested an algorithmic time-domain simulation, that went as follows:

1. Suppose the variables ξ, p, p_{lip}, U_{acoust}, U_{lip}, and S_{lip} are all known at all times earlier than the present, solve the equation of the lip motion (2.1), and find the new ξ at the time one step ahead.
2. Calculate the new S_{lip} and U_{lip} defined by Eqs. 2.5 and 2.6, respectively.
3. With the new S_{lip}, U_{lip}, and past data of p and $U = U_{acoust} + U_{lip}$, solve the flow equation [i.e., sum of Eqs. 2.7 and 2.8] and the feedback equation 2.9 simultaneously, and obtain the new p and U_{acoust}.
4. Solve Eq. 2.8 to obtain the new p_{lip}.
5. Update time by one step, and then return to 2.1.

$$S_{lip} = \max\{2b\xi_y, 0\}, \tag{2.5}$$

$$U_{lip} = \left\{(b(\xi - \xi_{joint}) \times \frac{d\xi}{dt}\right\} \cdot e_z = b\left\{(\xi_X - \xi_{joint_x})\frac{d\xi_y}{dt} - (\xi_y - \xi_{joint_y})\frac{d\xi_x}{dt})\right\} \tag{2.6}$$

$$p_0 - p_{lip} = \frac{1}{2}\rho\left(\frac{U_{acoust}}{S_{lip}}\right)^2 + \frac{\rho d}{S_{lip}}\frac{\partial U_{acoust}}{\partial t} \tag{2.7}$$

$$p_{lip} - p = -\rho U_{acoust}^2\left(\frac{1}{S_{cup}S_{lip}} - \frac{1}{S_{cup}^2}\right) \tag{2.8}$$

$$p(t) = Z_c U(t) + \int_0^\infty ds\ r(s)\{Z_c U(t-s) + p(t-s)\} \tag{2.9}$$

where:

- ξ - a 2D vector stating the position of C,
- k - stiffness factor,
- Q - quality factor,
- $F_{bernoulli}$ - external force generated by the Bernoulli pressure or the pressure at the lip opening p_{lip},
- e_z - unit vector parallel to the axis of swinging motion,
- $r(t)$ - reflection function, $Z_c = \rho c/S_{cup}$.

With this step by step solution I decided to put my efforts into creating a code of my own based on the step by step solution shown before. I started by analyzing each equation and trying to solve it. Unfortunately, I got stuck on Equation 2.1. Lacking knowledge of finite-differential equations forced me to step back and dive more into this topic. I started with basic DSP and moved forward to differential equations with the use of[19]. With the knowledge I now possessed I tried once again to solve Equation 2.1. Unfortunately, I wasn't able to find the boundary conditions, and even with help from different sources I was still unable to solve it.

2.3 Trombone Model

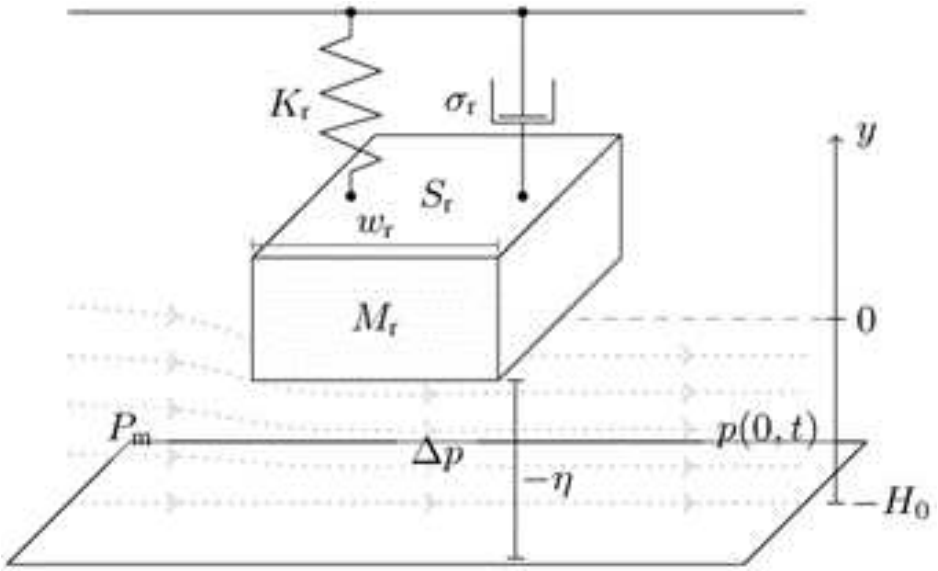


Figure 2.11: The lip mechanism according to[20].

"A Physical Model of the Trombone Using Dynamic Grids for Finite-Difference Schemes" by Silvin Willemsen, Stefan Bilbao, Michele Ducceschi and Stefania Serafin[20] was showing a complete simulation of a trombone using FDTD method. It introduced in details the digital waveguide approach to modelling a tube. It started by explaining a 1D model approximation of wave propagation in an acoustic tube, by using a system of first order PDEs, that were to describe the wave propagation by Equations 2.10 and 2.11.

$$\frac{S}{\rho_0 c^2}\partial_t p = -\partial_x(Sv) \tag{2.10}$$

$$\rho_0 \partial_t v = -\partial_x p \tag{2.11}$$

where:

- $p = p(x,t)$ - acoustic pressure,
- $v = v(x,t)$ - particle velocity,

- $S(x)$ - cross-sectional area,
- ρ_0 - density of air,
- c - speed of sound in air.

According to the paper, in order for the system to be excited, a lip reed could be modelled as a mass-spring damper system, that includes two nonlinearities due to flow and the collision of the lip against the mouthpiece (Figure 2.11). The model is seen as a movement of the upper lip where the lower lip is static.

Pseudocode 1 Pseudocode showing the calculations used in the code[20].

while *application is running* **do**

Retrieve new parameters	(L^n, ω_{r}^n and P_{m}^n)
Update L_p^n and L_q^n	(Eqs. (29), (43) & (28))
Calc. $\mathcal{N}^n$ and N^n	(Eqs. (24) and (17))
Calc. α^n	(Eq. (25))
if $N^n \neq N^{n-1}$ **then**	
Add or remove point	(Eq. (32) or (36))
Update M^n and M_q^n	(Eq. (20))
end	
Calc. $p_{M^n+1}^n$ and q_{-1}^n	(Eqs. (27))
Calc. $\mathbf{v}^{n+1/2}$ and $\mathbf{w}^{n+1/2}$	(Eqs. (21b) and (21d))
Calc. $y^{n+3/2}$ w/o collision	(Eqs. (18))
Calc $g^{n+1/2}$	(Eq. (19))
Calc. $y^{n+3/2}$ with collision	(Eqs. (18))
Calc. $U_{\mathrm{B}}^{n+1/2}$ and $U_{\mathrm{r}}^{n+1/2}$	(Eqs. (18c) and (18d))
Calc. $\mathbf{p}^{n+1}$ and $\mathbf{q}^{n+1}$	(Eqs. (37))
Retrieve output	
Update system states	($\mathbf{p}^{n-1} = \mathbf{p}^n$, $\mathbf{p}^n = \mathbf{p}^{n+1}$) (same for $\mathbf{v}^{n-1/2} = \ldots$, $y^{n-1/2}$, $y^{n+1/2}$, and ψ^n)
Update N^{n-1}	($N^{n-1} = N^n$)
Increment n	

end

Furthermore, they have assumed that the volume flow velocity is conserved. Based on that, they managed to define the total air volume entering the system as per Equations 2.12, 2.13 and 2.14.

$$S(0)v(0,t) = U_B(t) + U_r(t) \tag{2.12}$$

$$U_B = w_r[-\eta]_+ sgn(\Delta p)\sqrt{\frac{2|\Delta p|}{\rho_0}} \tag{2.13}$$

$$U_r = S_r \frac{dy}{dt} \tag{2.14}$$

where:

- ω - effective lip-reed width,
- $Kr = Kr(t)$ - stiffness,
- $\eta = \eta(t)\triangle - y - H_0$ - inverted distance between the lips,
- y - displacement from the equilibrium.

Next, they started to define the dynamic grid, which I sidestepped, since I had no need for it in a trumpet - while it was necessary for them to add, since the basics of trombone pitch change are due to the difference in tube length, but the trumpet stays the same length all the time. They then proceeded to explain the implementation of the code they have made in order to create a working model of a trumpet. Unfortunately, the code was done in C++ using the JUCE framework. The code works as reported in Pseudocode 1.

They have also made an graphical user interface (GUI) where the geometry of the tube is being plotted along with the paths showing the pressure states in blue and the velocity in green (see Figure 2.12). The real-time application is controlled by the usage and movement of the mouse in the bottom panel.

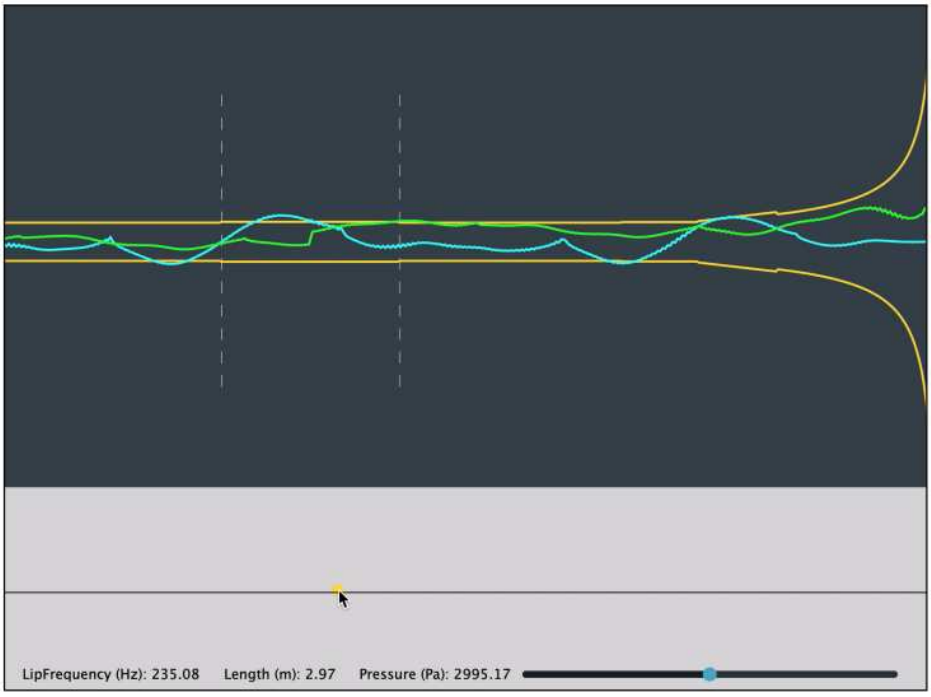


Figure 2.12: Screenshot of the GUI showing the geometry of the tube (in orange), the state pressure (in blue) and the velocity (green). The start and end of the slide are marked as an dashed line[20].

I have tried to port the code into the MATLAB environment, but unfortunately translating it from C++ to MATLAB code was quite hard. I chose another approach, which consisted of creating the code from scratch using the steps shown in Pseudocode 1. Unfortunately, computational obstacles stopped me in doing so.

2.4 Matlab Modelling of Articulated Brass Instruments

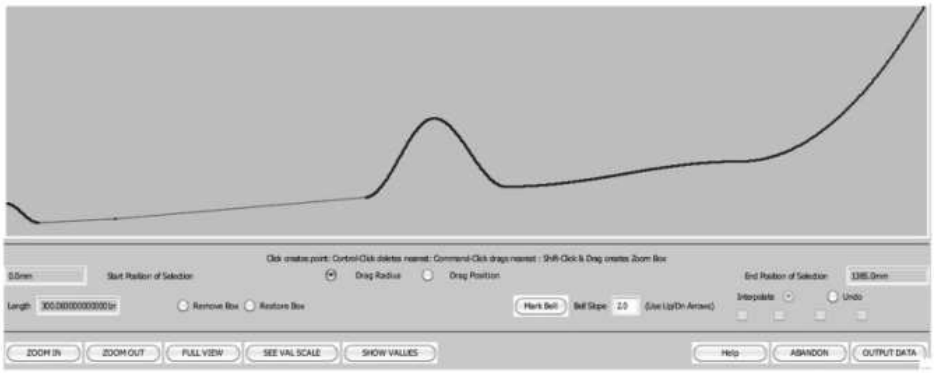


Figure 2.13: The interface of Sound Loom using the brass environment: instrument interface[21].

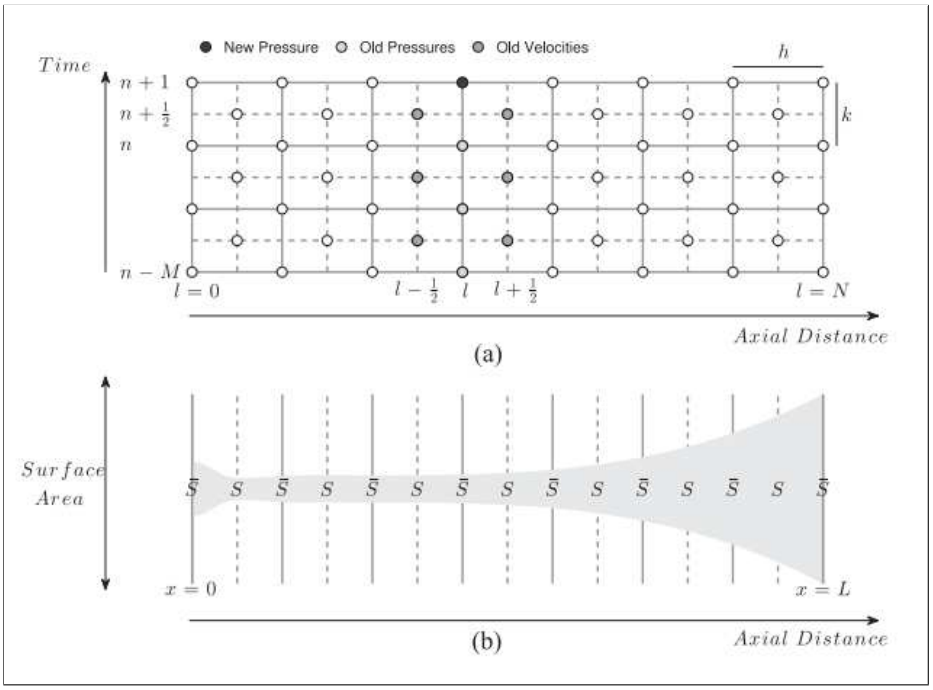


Figure 2.14: (a) Grid arrangement and representation of the lossy pressure. (b) Bore profile on interleaved spacial grid[21].

I then found a paper titled *"An Environment for Physical Modelling of Articulated Brass Instruments"* by Reginald Langford Harrison, Stefan Bilbao, James Perry and Trevor Wishart[21]. They have described the lip dynamics of a brass player as an outward striking reed, that can be modelled as a damped mass-spring system (Equation 2.15) driven by the pressure difference between

the player's mouth and the instrument's mouthpiece[17] with the assumption of Bernoulli-type nonlinear flow.

$$\ddot{y} + \sigma\dot{y} + 4\pi^2 f_{lip}^2 y = S_r \Delta p/\mu \tag{2.15}$$

where:

- y - is the lip displacement from its equilibrium position,
- σ and f_{lip} - the lips damping and natural resonance frequency,
- μ - lip mass,
- S_r - surface area of the lip.

Next, they have used the finite-difference time-domain methods, in order to make the simulation more flexible for the time-varying systems. They have created an FDTD scheme in terms of time step and grid spacing[22]. The arrangement of the discrete pressure and velocity fields and the sampling of the instrument bore is shown on Figure 2.14.

The authors did create a working model of a tube (Figure 2.13) in MATLAB using the finite difference time domain methods and Webster's equations, but unfortunately the code could not be shared with me by them yet. Nevertheless, I have tried to reproduce their work. The authors created an FDTD scheme that consisted of the definition of grids and bore surface areas. Following this they created the finite-difference approximations. That gave me an idea on how to create my own code in order to be able to model the tube.

Chapter 3

The Code

3.1 Digital Waveguide

I started by borrowing the code from prof. Bilbao called: "*1D Wave Equation: Finite Difference Digital Waveguide Synthesis*". In the code, a simple digital waveguide is being presented.

```
% matlab script waveeq1ddw.m
% digital waveguide method for the 1D wave equation
% fixed boundary conditions
% raised cosine initial conditions

%%%%%% begin global parameters

SR = 44100;                          % sample rate (Hz)
f0 = 441;                            % fundamental frequency (Hz)
TF = 1;                              % duration of simulation (s)
ctr = 0.7; wid = 0.1;                % center location/width of
   excitation
u0 = 1; v0 = 0;                      % maximum initial displacement/
   velocity
rp = 0.3;                            % position of readout (0-1)

%%%%%% end global parameters

% begin derived parameters

k = 1/SR;                            % time step
NF = floor(SR*TF);                   % duration of simulation (
   samples)
N = floor(0.5*SR/f0);                % length of delay lines
rp_int = 1+floor(N*rp);              % rounded grid index for readout
rp_frac = 1+rp*N-rp_int;             % fractional part of readout
   location
```

```
24
25 % initialize delay lines and output
26
27 wleft = zeros(N,1); wright = zeros(N,1);
28 out = zeros(NF,1);
29
30 % create raised cosine and integral
31
32 xax = ([1:N]'-1/2)/N;
33 ind = sign(max(-(xax-ctr-wid/2).*(xax-ctr+wid/2),0));
34 rc = 0.5*ind.*(1+cos(2*pi*(xax-ctr)/wid));
35 rcint = zeros(N,1);
36 for qq=2:N
37     rcint(qq) = rcint(qq-1)+rc(qq)/N;
38 end
39
40 % set initial conditions
41
42 wleft = 0.5*(u0*rc+v0*rcint/(2*f0));
43 wright = 0.5*(u0*rc-v0*rcint/(2*f0));
44
45 %%%%%% start main loop
46
47 for n=3:NF
48     temp1 = wright(N); temp2 = wleft(1);
49     wright(2:N) = wright(1:N-1); wleft(1:N-1) = wleft(2:N);
50     wright(1) = -temp2; wleft(N) = -temp1;
51     % readout
52     out(n) = (1-rp_frac)*(wleft(rp_int)+wright(rp_int))...
53         +rp_frac*(wleft(rp_int+1)+wright(rp_int+1));
54 end
55
56 %%%%%% end main loop
57
58 % plot output waveform
59
60 plot([0:NF-1]*k, out, 'k');
61 xlabel('t'); ylabel('u'); title('1D Wave Equation: Digital
      Waveguide Synthesis Output');
62 axis tight
63
64 % play sound
65
66 soundsc(out,SR);
```

Code snippet 3.1: "1D Wave Equation: Finite Difference Digital Waveguide Synthesis" by Prof. Stefan Bilbao.

I further developed the code into a working one. I started by understanding the code and the basic working principles of waveguides. I delved into the book by Stefan Bilbao[1], especially focusing on the chapter called *"Acoustic tubes"*. With help from my supervisors, I managed to understand the working of a digital waveguide. With that, I was able to manipulate the variables and learn even more.

A waveguide is a structure that guides waves inside it by restricting their dispersion into one direction. Without such a restriction, usually the waves would disperse into all directions. When it comes to acoustics, waveguides focus mostly on sound waves.

Looking at the system presented by professor Bilbao, we may notice the waveguide is build with fixed boundary conditions and raised cosine initial conditions. With the delay lines being updated in the main loop, we can actually see the working of the waveguide.

First, the delay lines are initialized and a raised cosine integral is created. Then, with each step, the delay lines are being updated. With the loop executing for as long as the duration time set for the simulation, and with the readout location set to 0.3 of the total tube length, we can see the behaviour of the wave inside the waveguide.

3.2 Reflection Parameters

I then proceeded to modify the base code in a way that would allow me more freedom of choice for the parameters and better understanding of a step-by-step synthesis. Firstly, I added reflection parameters *db* (diffusion of the backward wave) and *df* (diffusion of the forward wave), which allow me to set if the wave is being reflected, reflected and inversed, or if it is going to disperse. I implemented these parameters in the main loop. Then, I changed the cosine initial wave into a constant array of value 1, so I could see how the system works. As we can see in the code, we are inputting ones on the first position of the forward wave (wright) (assuming that the tube begins from the left side and ends on the right side, the position in the wave is numbered from left to right, meaning that the first position of the wave is on the left side of the tube (at the beginning) while the N-th position is on the right (at the end)).

```
%%%%%% Main Loop: Simulate Wave Propagation %%%%%%
for n = 1:NF
    temp1 = wright(N);
    temp2 = wleft(1);
    wright(2:N) = wright(1:N-1);
    wright(1) = 1;
    wleft(1:N-1) = wleft(2:N);
    wright(1) = wright(1)+df*temp2;
    wleft(N) = db*temp1;

    % Readout
    out(n) = (1 - rp_frac) * (wleft(rp_int) + wright(rp_int)) ...
        + rp_frac * (wleft(rp_int+1) + wright(rp_int+1));
end
```

Code snippet 3.2: Creation of a wave inside a waveguide with reflection parameters on both ends.

By running this code and using the *plot* function, we can see the sound propagation output at the end of the tube in time. I carried out simulations of the tube with following states:

1. open at both ends (*df = 0; db = 0*) (Figure 3.1),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.2),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figure 3.3),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.4).

As it can be seen on the plots marked next to it's state, the model was working. The simulations were run on a 2 seconds timespan.

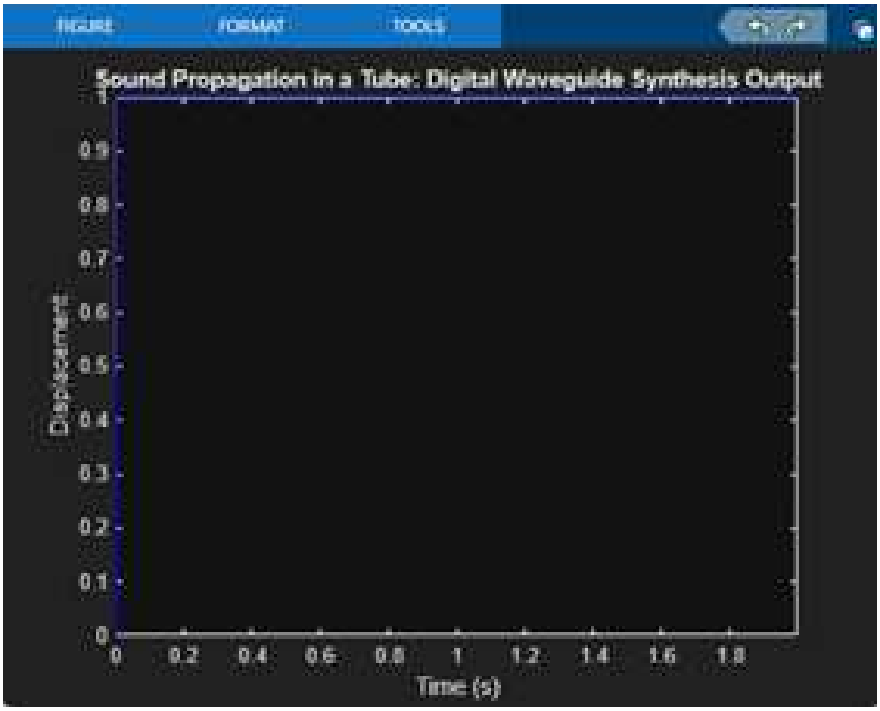


Figure 3.1: Sound propagation in a tube - displacement values in time with both ends open *(df = 0; db = 0).*

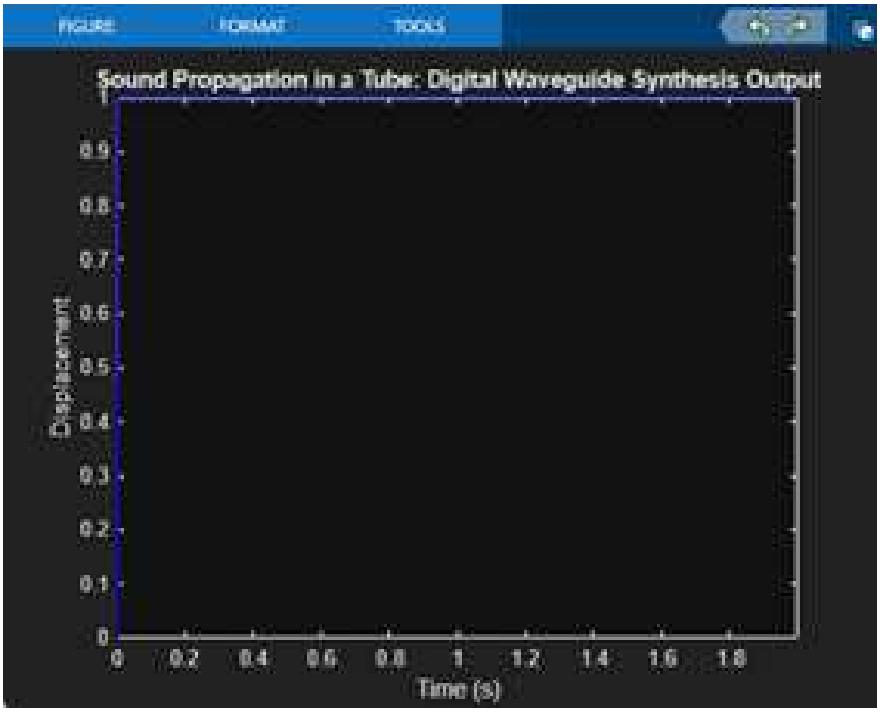


Figure 3.2: Sound propagation in a tube - displacement values in time with left end closed and right end open *(df = 1; db = 0)*.

Then I inversed the closed ends that so *df = 0* and *db = 1*.

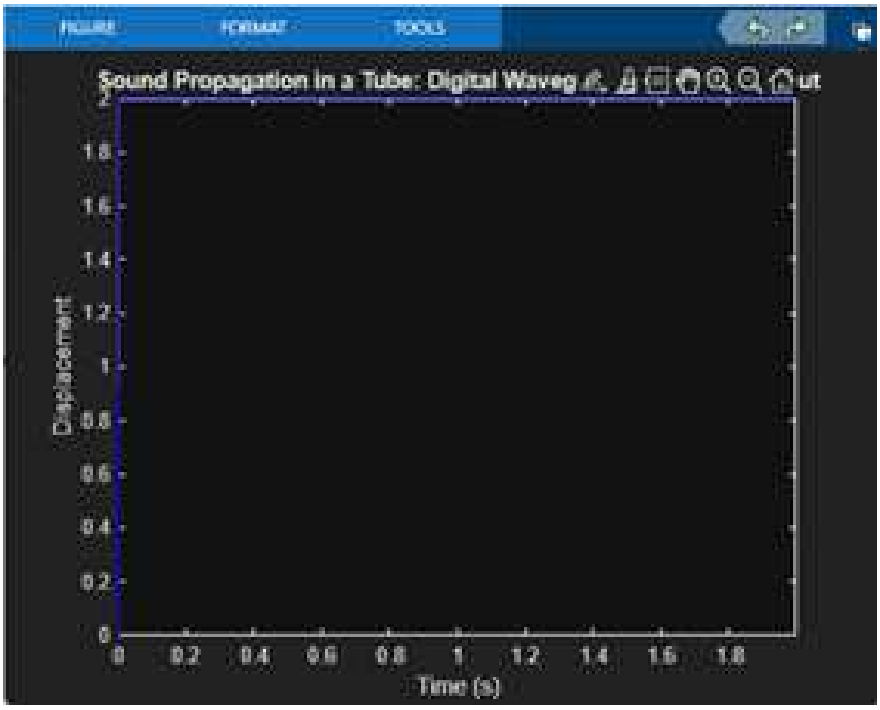


Figure 3.3: Sound propagation in a tube - displacement values in time with left end open and right end closed *(df = 0; db = 1)*.

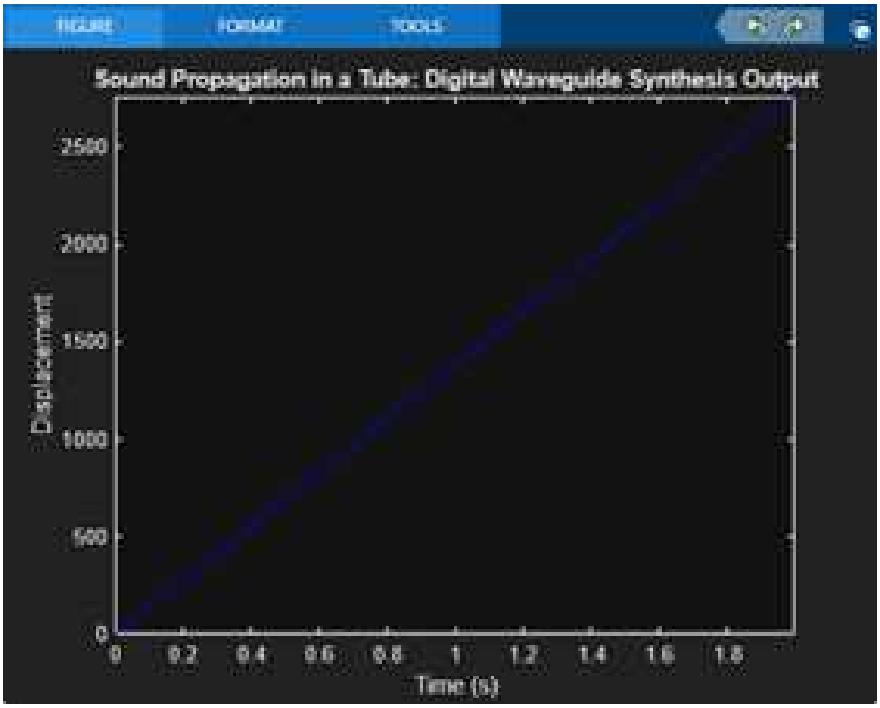


Figure 3.4: Sound propagation in a tube - displacement values in time with both ends closed *(df = 1; db = 1)*.

I had overcome the lack of the mouthpiece model element by creating a .wav file of the mouthpiece sound. The input .wav file can be seen on Figure 3.5.

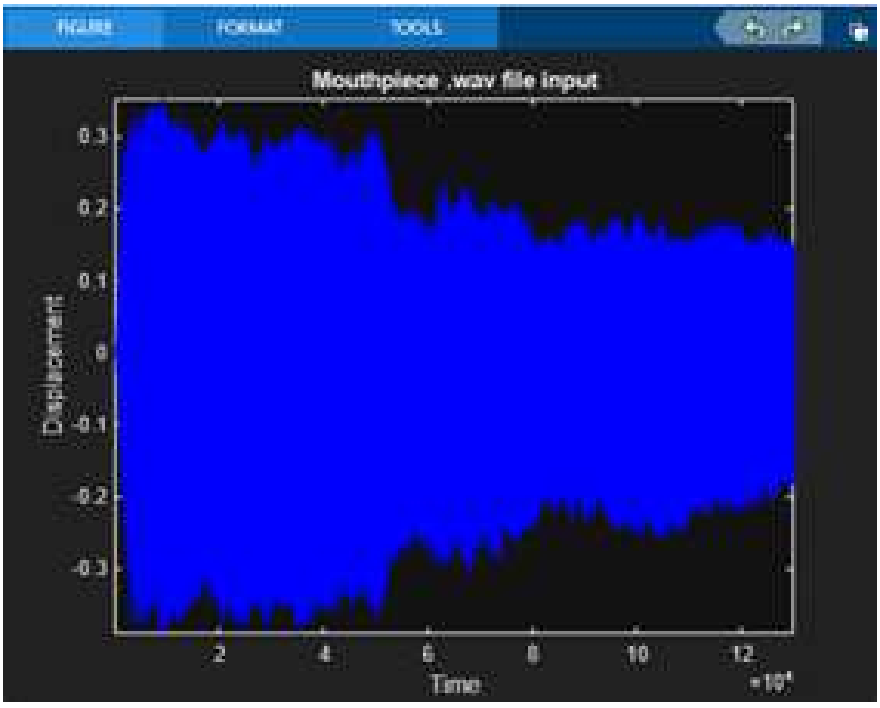


Figure 3.5: Mouthpiece input .wav file as a displacement in time plot.

3.3 The Mouthpiece Sound

Next, I introduced the mouthpiece in the code as input data and changed the input for the forward wave from a constant array to the values of the .wav sound .

```
1   wright(1) = input_data(mod(n-1,cut_value)+1);
```

Code snippet 3.3: Changed input of the right wave from ones to the values of the .wav file.

I ran the simulation again with different end closure configurations:

1. open at both ends *(df = 0; db = 0)* (Figure 3.6),
2. open at the right end, closed on the left *(df = 1; db = 0)* (Figure 3.7),
3. open on the left end, closed on the right *(df = 0; db = 1)* (Figure 3.8),
4. closed on both ends *(df = 1; db = 1)* (Figure 3.9).

Each simulation has been run for 3 seconds. The output can be seen on the figures marked below according to it's state.

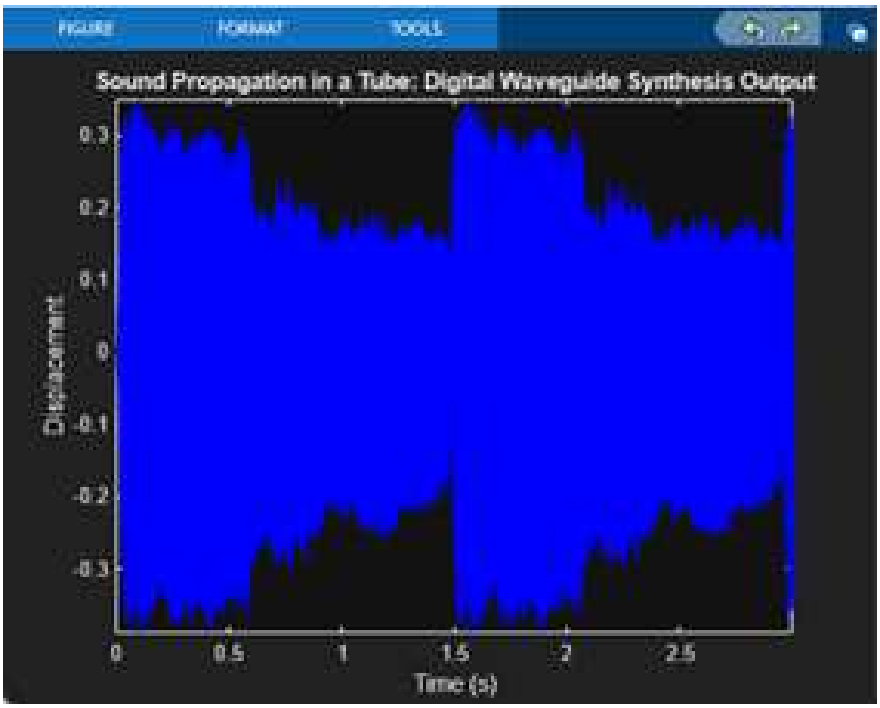


Figure 3.6: Mouthpiece sound propagation in a tube - displacement in time with both ends open *(db = 0; df = 0)*.

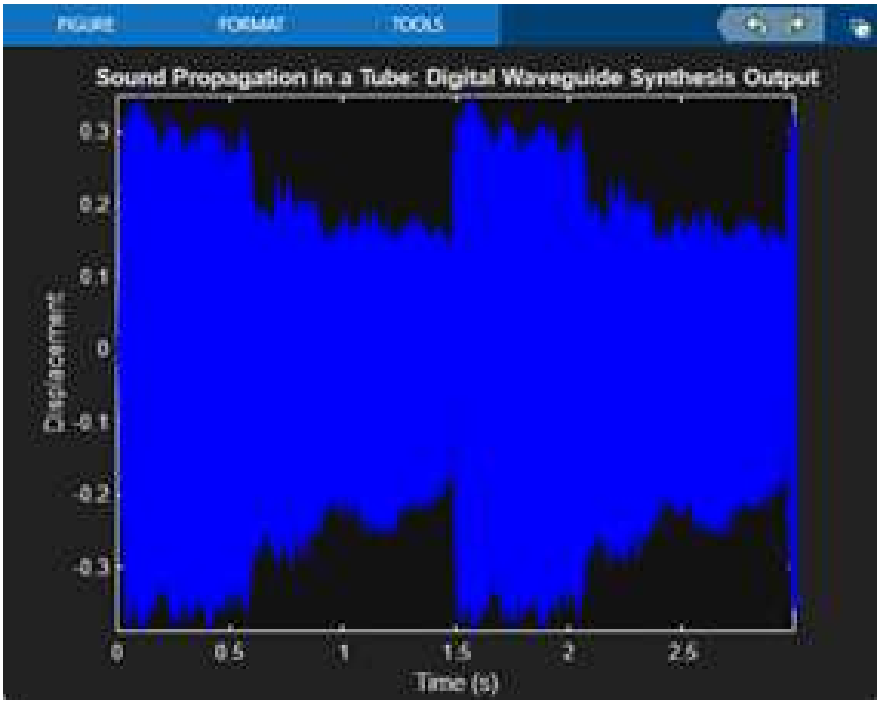


Figure 3.7: Mouthpiece sound propagation in a tube - displacement in time with left end closed and right end open *(df = 1; db = 0)*.

Then I inverted the closed ends that so *df = 0* and *db = 1*.

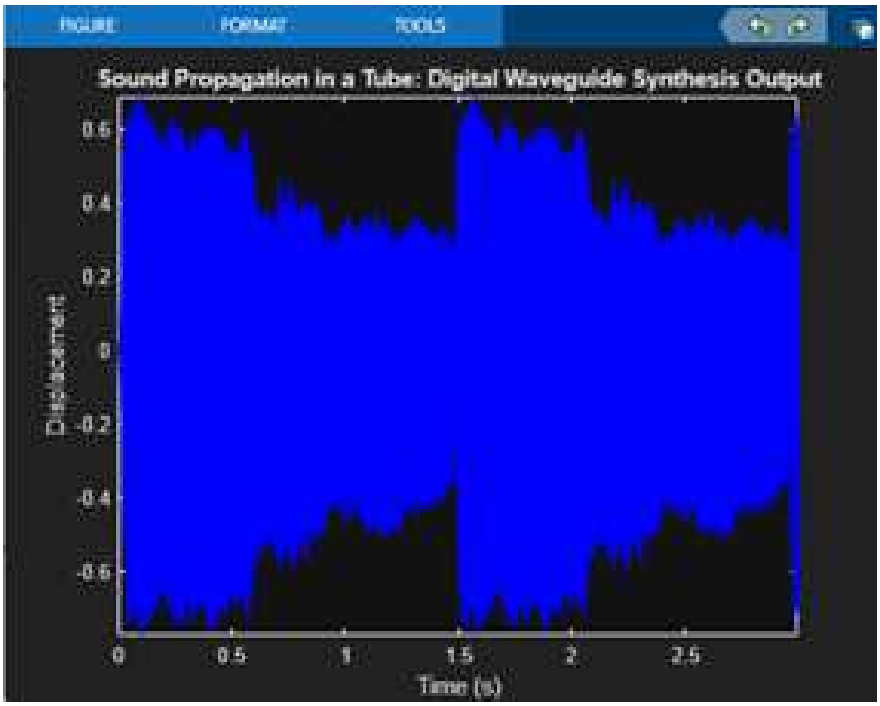


Figure 3.8: Mouthpiece sound propagation in a tube - displacement in time with left end open and right end close *(df = 0; db = 1)*.

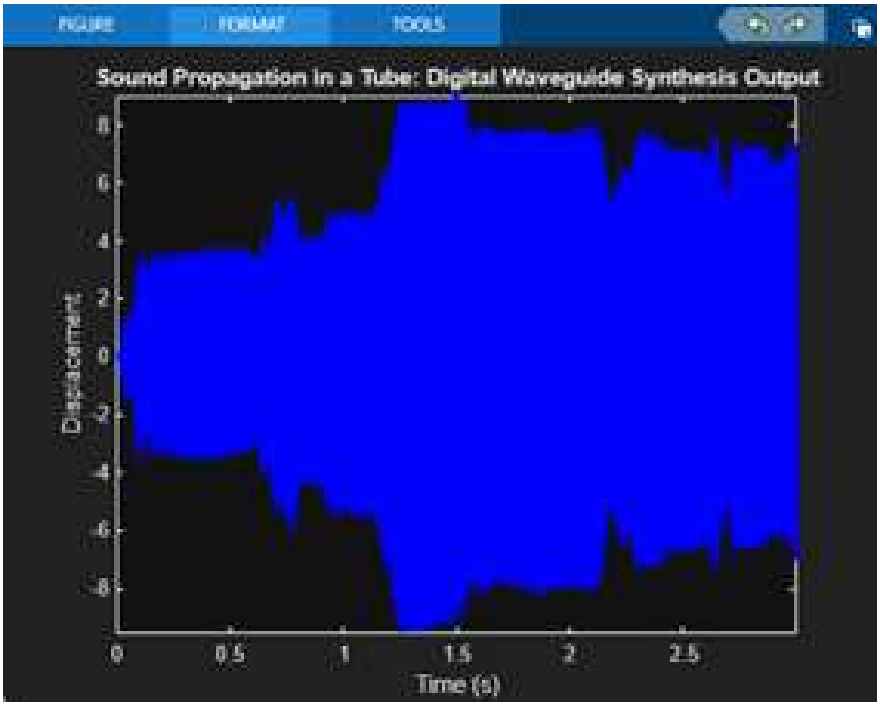


Figure 3.9: Mouthpiece sound propagation in a tube - displacement in time with both ends closed *(df = 1; db = 1)*.

I have also added an *if* loop, that would display step by step the wave propagation inside the tube, creating a "movie" of the wave propagation. Figure 3.10 shows snippets of the standing wave that was created in the tube with both ends closed *(df = 1; db = 0)*. The X axis shows the tube length, the plot is of the mouthpiece .wav file input.

As we can see in Figure 3.10, a standing wave is slowly appearing in time.

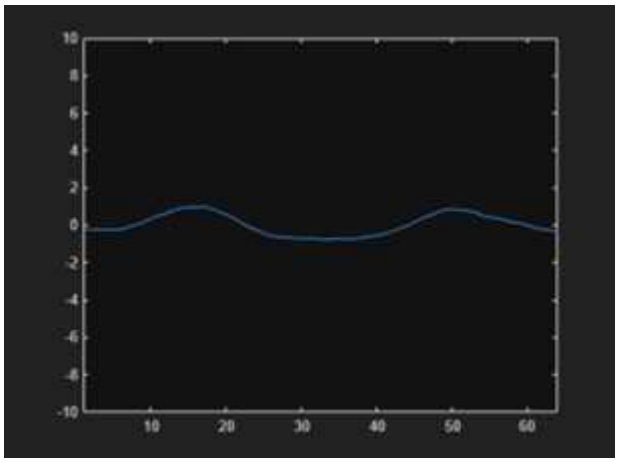
Since I now had a working segment of the tube, I moved forward creating two segments, firstly both with the same diameter and next with different diameters. In order to do that, I needed to figure out how both the forward and backwards waves propagate through the interface between two segments.

3.4 Segmentation

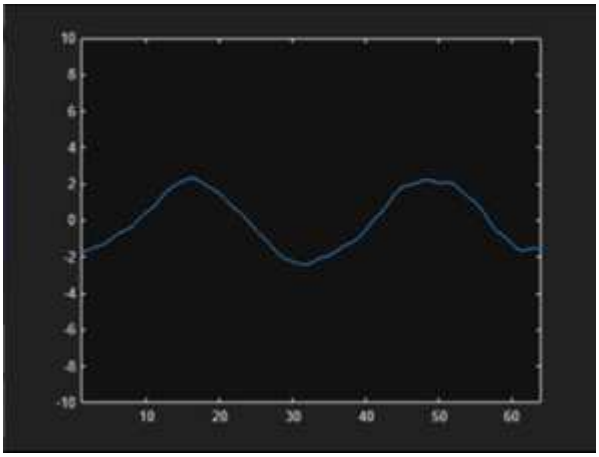
3.4.1 Segments

Once again, I turned into professor's Bilbao book "*Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*"[1], especially looking at chapter nº 9.2. Although the chapter is focused around vocal tract and speech synthesis, professor Bilbao does explain that the vocal tract synthesis is quite similar to a tube synthesis together with its digital waveguide.

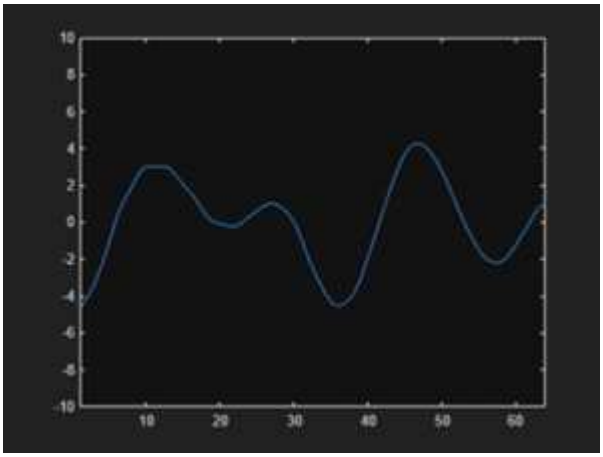
In the beginning, I wanted to create just two segments in order to get acquainted with the creation of segments and the usage of the scattering methods. I did however had to familiarize myself with the whole methodology to do so. A typical scattering structure may be described as a tube profile divided into approximated segments in the shape of a cylinder with length h, such that $N = 1/h$,



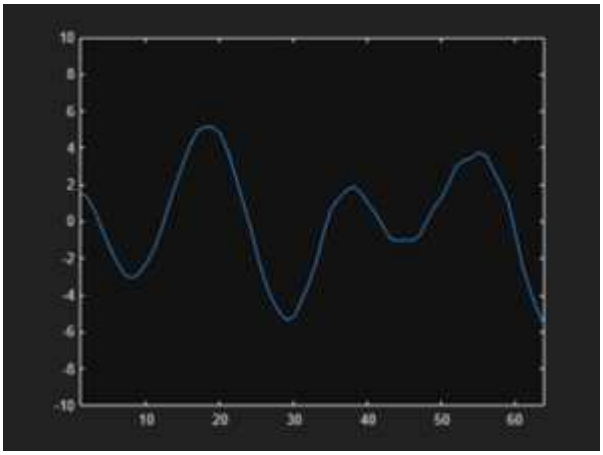
(a) after 0.033 s



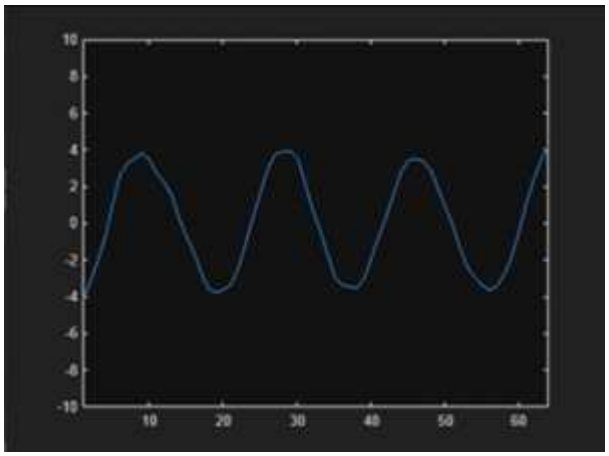
(b) after 0.433 s



(c) after 1.066 s



(d) after 1.9 s



(e) after 2.666 s

Figure 3.10: Sound propagation in a tube - a simulation of 3 s.

with the assumption that the tube length equals 1 (see Figure 3.11). The left and right ends of the l-th tube are located at $x = l * h$ and $x = (l + 1) * h$, for $l = 0, ..., N - 1$[1].

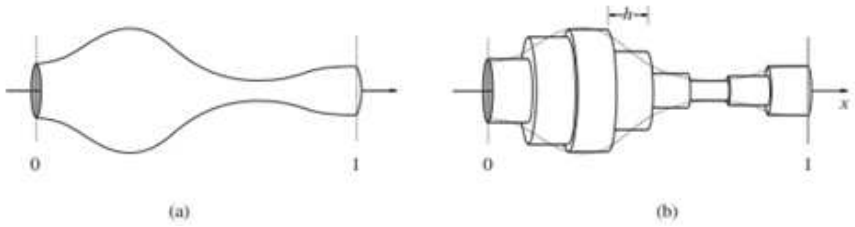


Figure 3.11: (a) an acoustic tube before and (b) after division into a cylindrical-shaped segments (with the length of each segment being h = 1/N)[1].

Each segment is described by a constant surface area of $[S]_{l+1/2}$ which is an approximation of the surface area of the centre of a segment. Such approximation can lead us to reduce the system into a 1D wave equation, with its solution provided by Equation 3.1

$$p(x,t) = p^{(+)} + p^{(-)} \tag{3.1}$$

where the subscripts (+) and (-) indicate the solution travelling to the right (forward) and left (backward) respectively[1].

3.4.2 Junctions Between Segments

We can therefore label the solutions on the junction (l) between the segments (n) as in Equation 3.2.

$$p^{(+),n}_{l,left}, p^{(-),n}_{l,left}, p^{(+),n}_{l,right}, p^{(-),n}_{l,right} \tag{3.2}$$

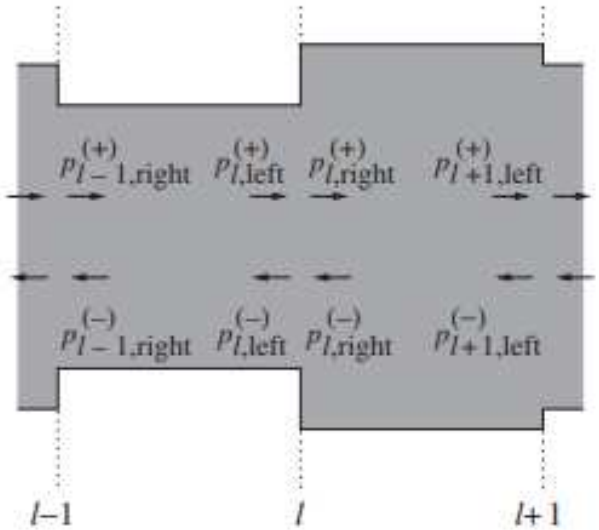


Figure 3.12: A sequence of cylindrical tube sections with mark of the position of the solution[1].

Considering Figure 3.12, we can write the wave propagation as in Equation 3.3.

$$p_{l,left}^{(+),n+1} = p_{l-1,right}^{(+),n} \qquad p_{l,right}^{(-),n+1} = p_{l+1,left}^{(+),n} \tag{3.3}$$

When applying the scattering operation, we can derive Equations 3.4 and 3.5

$$p_{l,left}^{(+),n} + p_{l,left}^{(-),n} = p_{l,right}^{(+),n} + p_{l,right}^{(-),n} \tag{3.4}$$

$$[S]_{l-\frac{1}{2}} \left(p_{l,left}^{(+),n} - p_{l,left}^{(-),n} \right) = [S]_{l+\frac{1}{2}} \left(p_{l,right}^{(+),n} - p_{l,right}^{(-),n} \right) \tag{3.5}$$

which can then be rewritten in a scattering form using a 2x2 matrix multiplication. In order for it to work, we have to use the reflection parameter r_l, which can be derived from Equations 3.6 and 3.7.

$$r_l = \frac{[S]_{l+\frac{1}{2}} - [S]_{l-\frac{1}{2}}}{[S]_{l+\frac{1}{2}} + [S]_{l-\frac{1}{2}}} \tag{3.6}$$

$$\begin{bmatrix} p_{l,left}^{(-),n} \\ p_{l,right}^{(+),n} \end{bmatrix} = \begin{bmatrix} -r_l & 1+r_l \\ 1-r_l & r_l \end{bmatrix} \begin{bmatrix} p_{l,left}^{(+),n} \\ p_{l,right}^{(-),n} \end{bmatrix} \tag{3.7}$$

3.4.3 Two - Segment Code

The first problem I encountered was the wave propagation evaluation at the interface between the first and the second segment. As part of the wave would be bouncing back, but part of it would be going forward to the next segment, I had to derive the reflection and transmission parameters. As I was just implementing 2 segments of the same diameter, I have assigned values instead of making the program derive it from the equations. Having the same diameter the value or r_l equals to zero, where:

- [M11] *-r_l = 0,*
- [M12] *1 + r_l = 1,*
- [M21] *1 - r_l = 1,*
- [M22] *r_l = 0.*

In the brackets, we can see the denominations of the variables used in the code. Applying this to the code, we get:

```
wright = zeros(N,1);
wleft = zeros(N,1);
df = 1;
M11 = 0;
```

```
5
6  wright2 = zeros(N,1);
7  wleft2 = zeros(N,1);
8  M12 = 1+M11;
9  db2 = 1;
10
11 M21 = 2-M12;
12 M22 = -M11;
13
14 %%%%%% Main Loop: Simulate Wave Propagation %%%%%%
15 for n = 1:NF
16     temp1 = wright(N);
17     temp2 = wleft(1);
18
19     temp11 = wright2(N);
20     temp22 = wleft2(1);
21
22     wright(2:N) = wright(1:N-1);
23     wright(1) = 1;
24     wleft(1:N-1) = wleft(2:N);
25     wright(1) = wright(1)+df*temp2;
26     wleft(N) = M11 * temp1 + M12 * temp22;
27
28     wright2(2:N) = wright2(1:N-1);
29     wright2(1) = M21 * temp1;
30     wleft2(1:N-1) = wleft2(2:N);
31     wright2(1) = wright2(1) + M22 * temp22;
32     wleft2(N) = db2 * temp11;
```

Code snippet 3.4: Modified code including the implementation of the scattering matrix for two segments of a tube.

When we run the code, we can notice on Figure 3.13 that the connection between two segments works fine.

3.4.4 Multi - Segment Code

With this method working, I was able to build a more complex system of multiple segments. It did require expanding the code a lot, since the difference between the first and second segment, the middle segments and the second-to-last and last segments were too big to be able to put it in one code. It required me to use classes, methods and functions, that were still hard for me to move around. Fortunately, after studying, I was able to get used to their implementation in the code and I was able to use them.

I had to divide the code into three methodologies:

1. The first segment and it's connection to the second segment, where I had to implement a reflection parameter *df* instead of using the scattering matrix (code snippet 3.7),

2. the connection between the middle segments, where the scattering matrix was the main way of deriving the parameters (code snippet 3.8),
3. the second-to-last and last segments, where the reflection parameter *db* was implemented (code snippet 3.9).

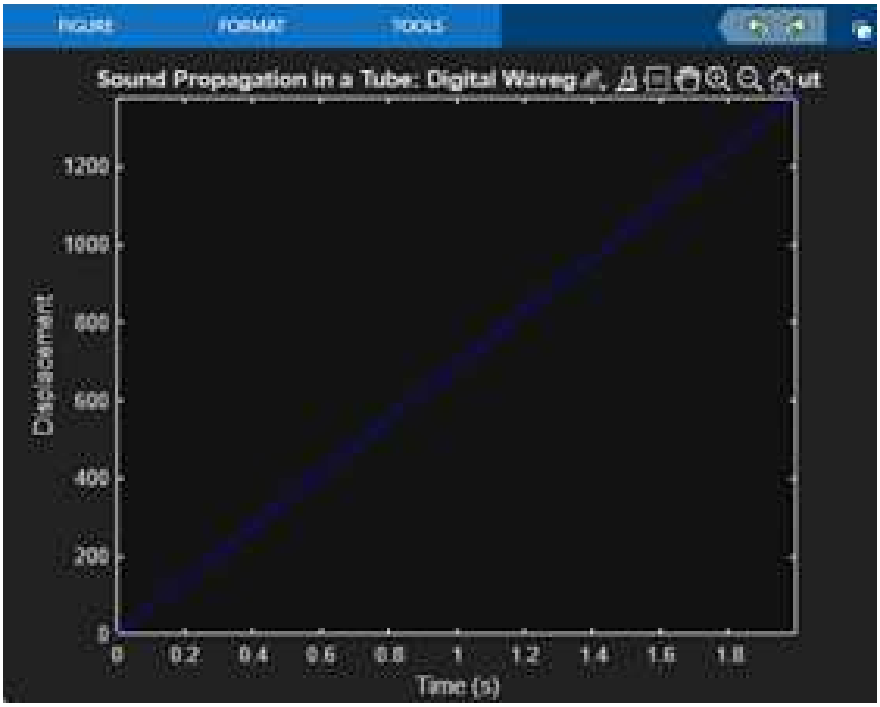


Figure 3.13: Sound propagation of constant array in tube with two segments of the same diameter marking the working of the system.

I set *df* and *db* at the beginning, imported a .txt file with saved profiles of shapes I would like to implement, and initialized a tube object:

```
classdef Tube
    properties
        wright
        wleft
        M11
        M12
        M21
        M22
        temp1
        temp2
    end
    methods
        function obj = Tube(N, rl, Wl, Wr)
            obj.wleft = Wl;
```

```
            obj.wright = Wr;
            obj.temp1 = obj.wright(N);
            obj.temp2 = obj.wleft(1);
            obj.M11 = - rl;
            obj.M12 = 1 - rl;
            obj.M21 = 1 + rl;
            obj.M22 = rl;
        end
    end
end
```

Code snippet 3.5: Definition od a class used in the code listed later.

Next, I initialized each segment object:

```
%%%%% Initialize Each Segment Object %%%%%%
for i = 1:numSegments
    Sn = data(i, 1);
    Sn1 = data(i, 2);
    rl = (Sn1-Sn)/(Sn1+Sn);
    Wr = zeros(N,1);
    Wl = zeros(N,1);
    tube{i} = Tube(N, rl, Wr, Wl); % Pass N, rl, Wr, Wl to the
   Tube constructor
end
```

Code snippet 3.6: Initialization of each segment object passed to the Tube constructor.

and created the main loop taking under consideration the three methodologies listed above.

```
%%%%%% Main Loop: Simulate Wave Propagation %%%%%%

for i = 1:NF
    for l = 1:numSegments
        if l == 1
            tube{l}.temp1 = tube{l}.wright(N);
            tube{l}.temp2 = tube{l}.wleft(1);
            tube{l}.wright(2:N) = tube{l}.wright(1:N-1);
            %tube{l}.wright(1) = input_data(mod(i-1,cut_value)+1);
            tube{l}.wright(1) = 1;
            tube{l}.wleft(1:N-1) = tube{l}.wleft(2:N);
            tube{l}.wright(1) = tube{l}.wright(1) + df * tube{l}.
   temp2;
            tube{l}.wleft(N) = tube{l}.M11 * tube{l}.temp1 + tube{
   l}.M12 * tube{l+1}.temp2;
```

Code snippet 3.7: First methodology listed above - the fist tube segment using *df* of the left end and scatterng matrix on the right.

```
        elseif l == numSegments
            tube{l}.temp1 = tube{l}.wright(N);
            tube{l}.temp2 = tube{l}.wleft(1);
            tube{l}.wright(2:N) = tube{l}.wright(1:N-1);
            tube{l}.wright(1) = tube{l-1}.M21 * tube{l-1}.temp1;
            tube{l}.wleft(1:N-1) = tube{l}.wleft(2:N);
            tube{l}.wright(1) = tube{l}.wright(1) + tube{l-1}.M22
  * tube{l}.temp2;
            tube{l}.wleft(N) = db * tube{l}.temp1;
```

Code snippet 3.8: Second methodology listed above - with the usage of the scattering matrix on both ends.

```
        else
            tube{l}.temp1 = tube{l}.wright(N);
            tube{l}.temp2 = tube{l}.wleft(1);
            tube{l}.wright(2:N) = tube{l}.wright(1:N-1);
            tube{l}.wright(1) = tube{l-1}.M21 * tube{l-1}.temp1;
            tube{l}.wleft(1:N-1) = tube{l}.wleft(2:N);
            tube{l}.wright(1) = tube{l}.wright(1) + tube{l-1}.M22
  * tube{l}.temp2;
            tube{l}.wleft(N) = tube{l}.M11 * tube{l}.temp1 + tube{
  l}.M12 * tube{l+1}.temp2;
        end
    end
```

Code snippet 3.9: Third methodology listed above - with the usage of scattering matrix on the left end and the *db* parameter on the right end.

As we can see, each time the code iterates, it refers to a cell referring to a segment. For each one of these cells all the operations, like creation of a forward and backward wave taking into account the reflection parameters from the scattering matrix, are being performed. Each time the operation on a segment is done, the output is being saved and in the end a plot of the sound propagation is being shown.

3.5 Shaping

Now, with a working digital waveguide, I started shaping it in order to see the wave propagation and how the shapes affect the output of the waves. I have therefore created a list of various shapes to test. I've started with an easy straight tube with one end shaped like a cone. Then I moved to create a more brass-like bell with different bore length, After testing that, I began to check the working of more imaginary-like shapes.

3.5.1 Short Bore and Cone-Like Bell Shape

The first shape I have uploaded, was a short bore and cone-like bell shape (Figure 3.14). The input data had to be created as two columns, with the first column signalling the first segment, while the second column signalling the next segment. You can see there are the same numbers in the n row

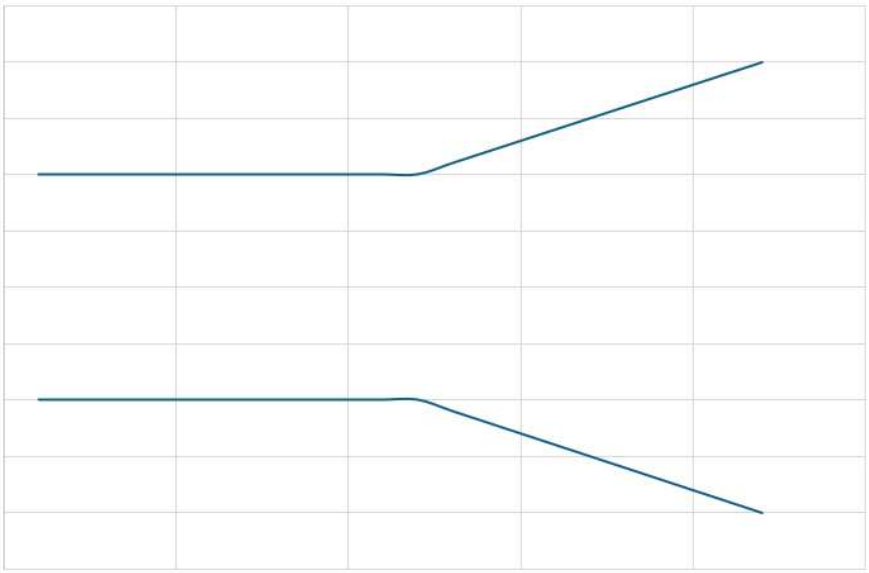


Figure 3.14: Shape of a short bore and cone-like bell input data.

in the second column as there are in the $n + 1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

left	right
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1
1	1.1
1.1	1.2
1.2	1.3
1.3	1.4
1.4	1.5
1.5	1.6
1.6	1.7
1.7	1.8
1.8	1.9
1.9	2

Table 3.1: Table of input data of a short bore and cone-like bell shape.

The output of testing this type of shape with input of zeros and with different configurations are

as follows (the simulations were run for 3 seconds):

1. open at both ends (*df = 0; db = 0*) (Figure 3.15),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.16),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figure 3.17),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.18).

Additionally, I have made a test with the input.wav file of a mouthpiece sound with the left end completely closed and the right end half closed $df = 1; db = 0.5$ (Figure 3.19).

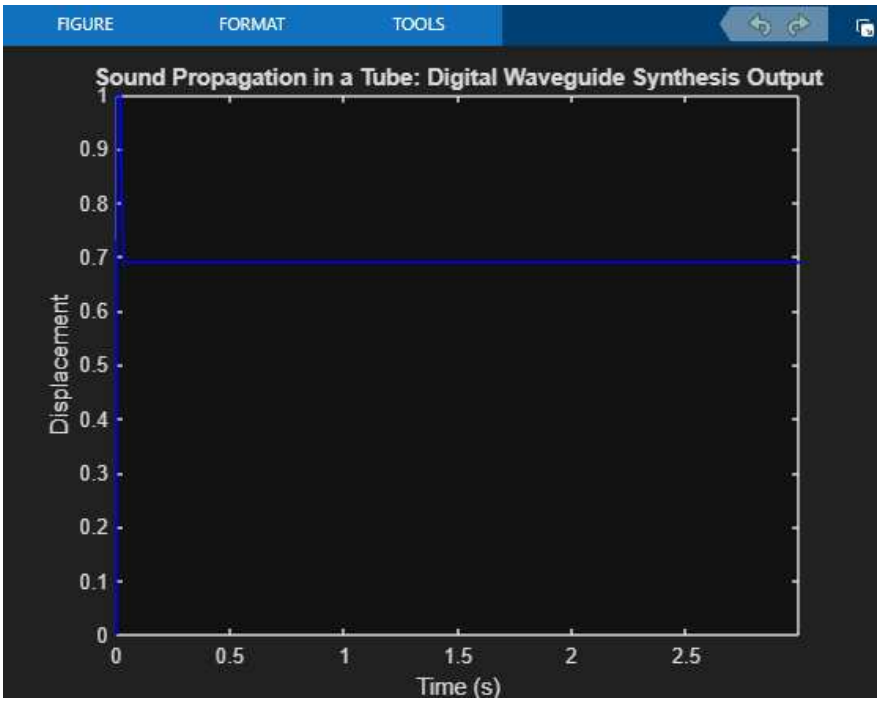


Figure 3.15: The short bore and cone-like bell shape with both ends open $df = 0; db = 0$ with input of zeros.

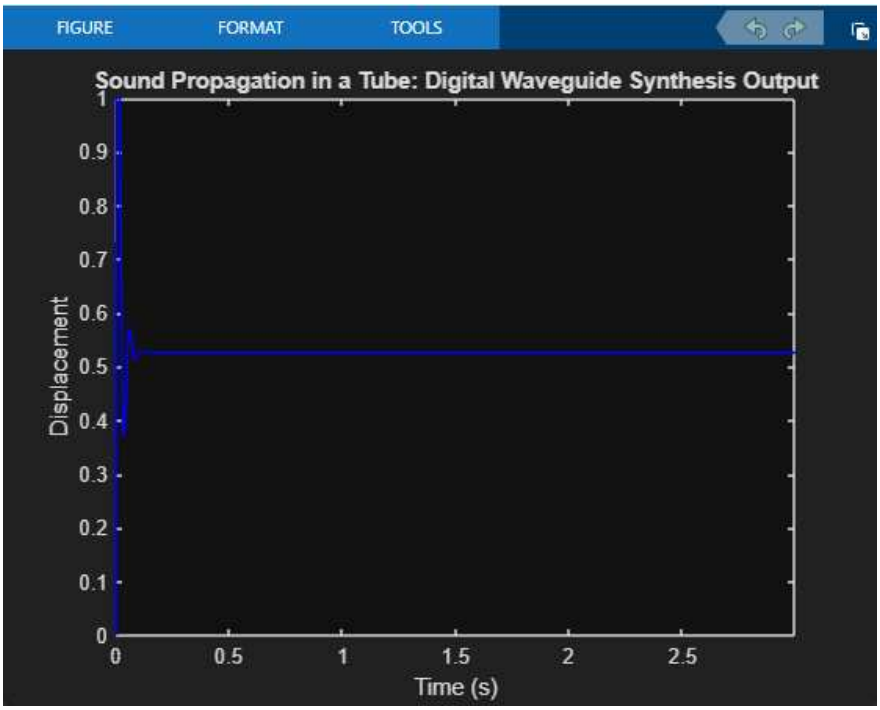


Figure 3.16: The short bore and cone-like bell shape with left end closed and right open $df = 1; db = 0$ with input of zeros.

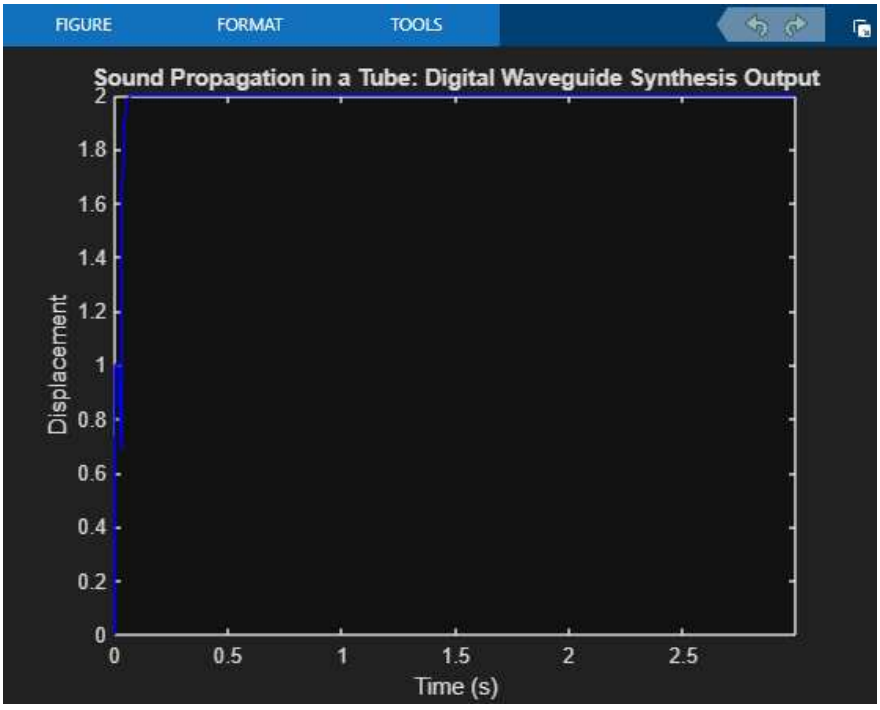


Figure 3.17: The short bore and cone-like bell shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

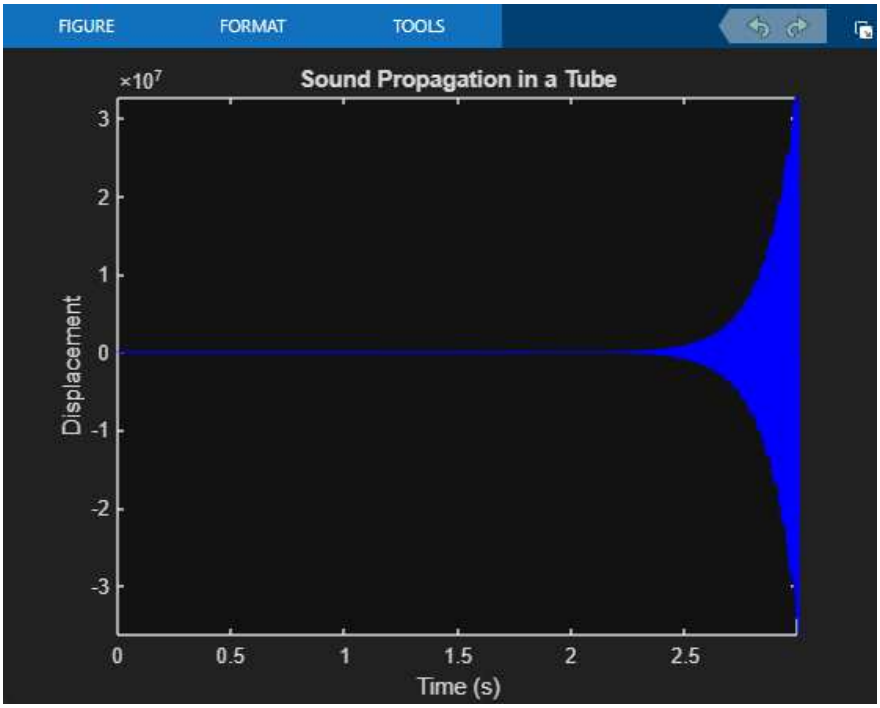


Figure 3.18: The short bore and cone-like bell shape with both ends closed $df = 1; db = 1$ with input of zeros.

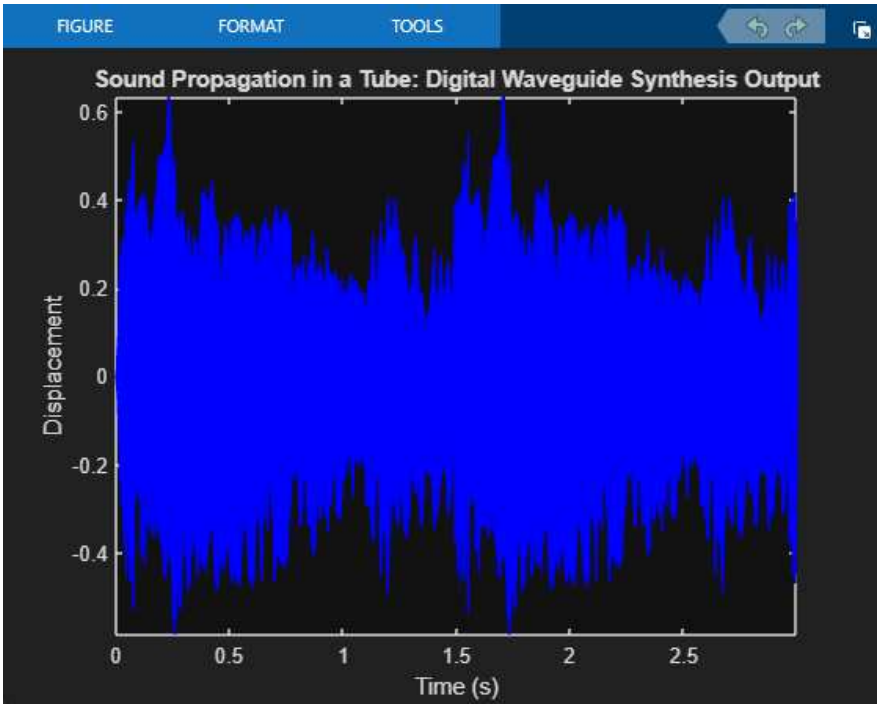


Figure 3.19: The short bore and cone-like bell shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

3.5.2 Short Bore and Brass Bell Shape

The same process was done for another shape - short bore and brass bell (Figure 3.20).

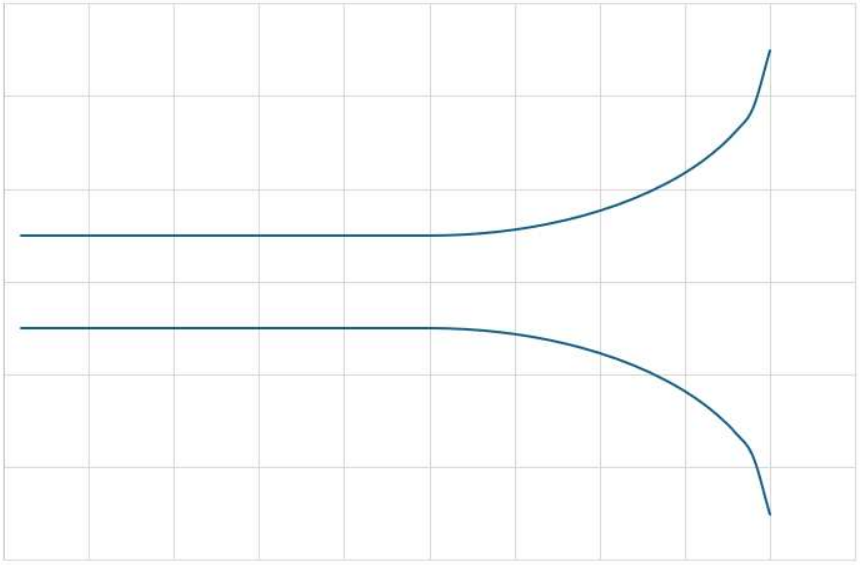


Figure 3.20: Shape of a short bore and brass bell input data.

left	right
5	5
5	5
...	...
5	5
5	5.025
5.025	5.100
5.100	5.226
5.226	5.404
5.404	5.635
5.635	5.921
5.921	6.265
6.265	6.670

6.670	7.139
7.139	7.680
7.680	8.297
8.297	9.000
9.000	9.801
9.801	10.717
10.717	11.771
11.771	13.000
13.000	14.464
14.464	16.282
16.282	18.755
18.755	25.000

Table 3.2: Table of input data of a short bore and brass bell shape.

In the Table 3.2 the first column signalling the first segment, while the second column signals the next segment. You can see there are the same numbers in the n row in the second column as there are in the $n + 1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

The output of testing this type of shape with input of zeros and with different configurations are as follows (the simulations were run for 3 seconds):

1. open at both ends (*df = 0; db = 0*) (Figure 3.21),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.22),

3. open on the left end, closed on the right (*df = 0; db = 1*) (Figure 3.24),
4. closed on both ends (*df = 1; db = 1*) (Figure 3.25).

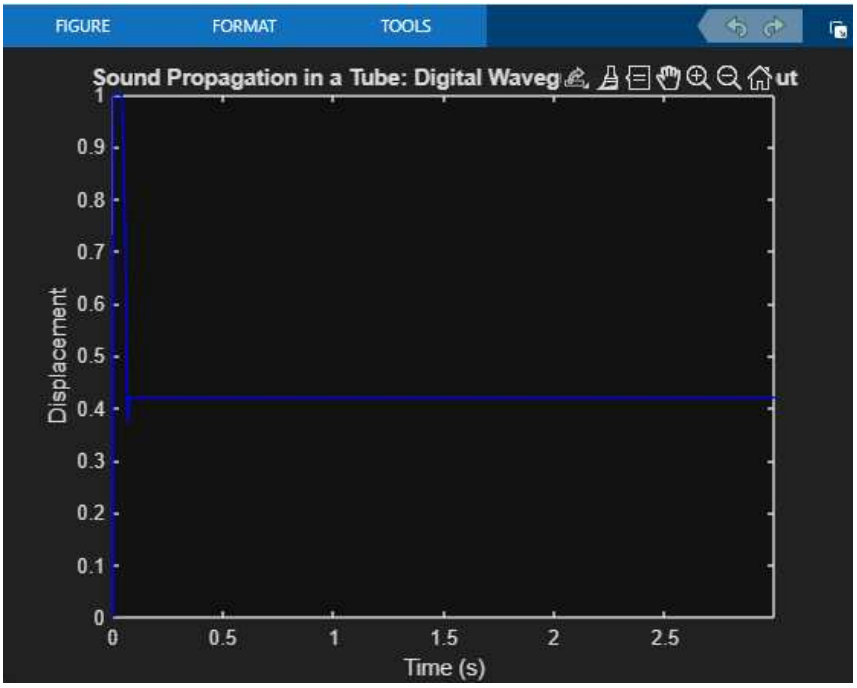


Figure 3.21: The short bore and brass bell shape with both ends open $df = 0; db = 0$ with input of zeros.

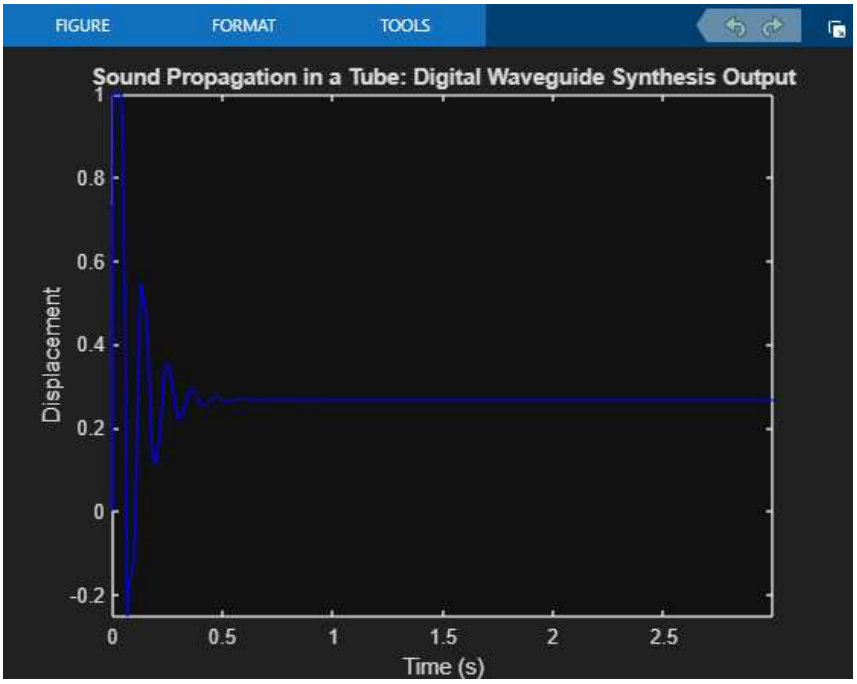


Figure 3.22: The short bore and brass bell shape with left end closed and right open $df = 1; db = 0$ with input of zeros.

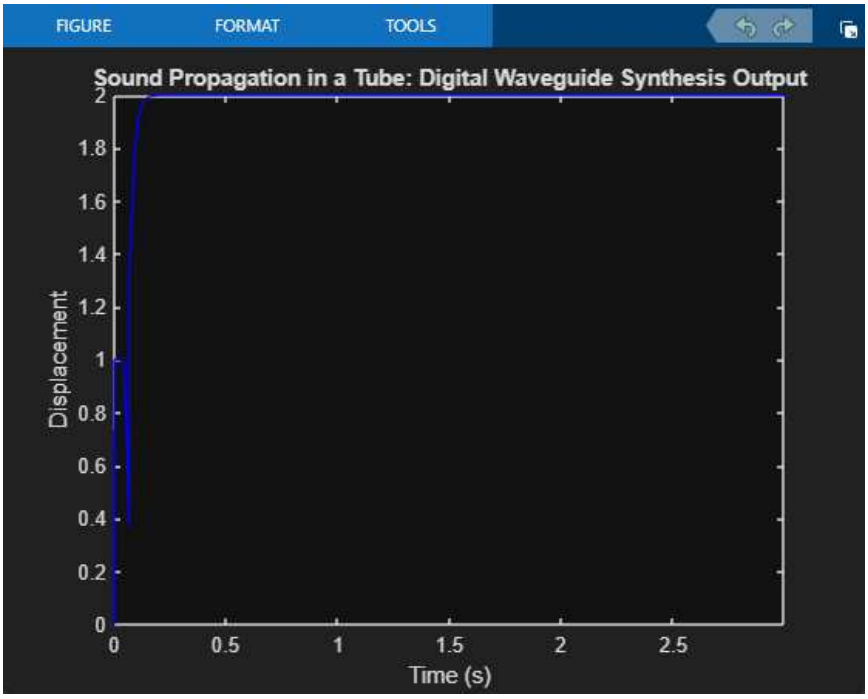


Figure 3.23: The short bore and brass bell shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

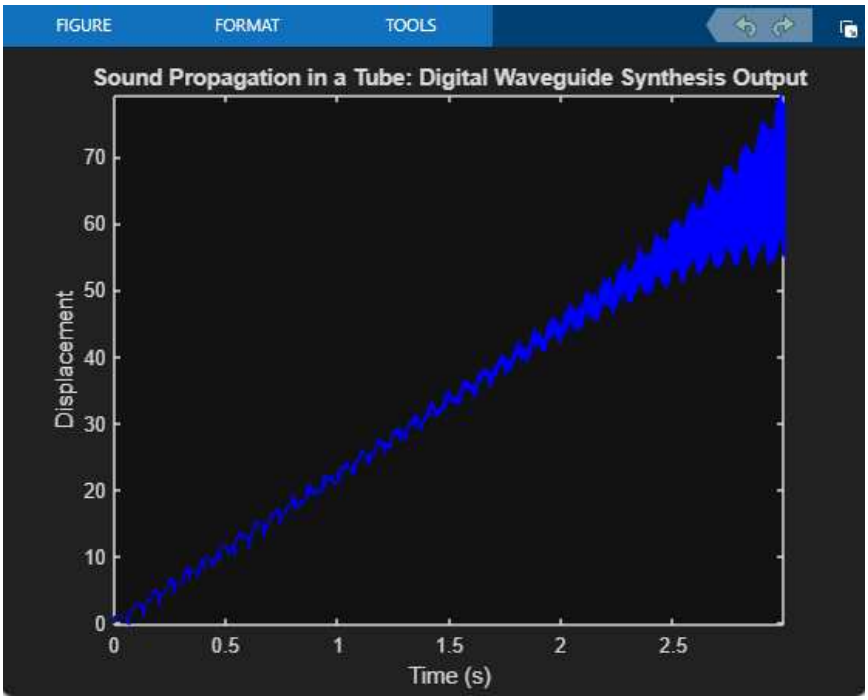


Figure 3.24: The short bore and brass bell shape with both ends closed $df = 1; db = 1$ with input of zeros.

Next I tested it with the sound of the .wav file and left end closed and right side partially open

$df = 1; db = 0.5$, see Figure 3.25.

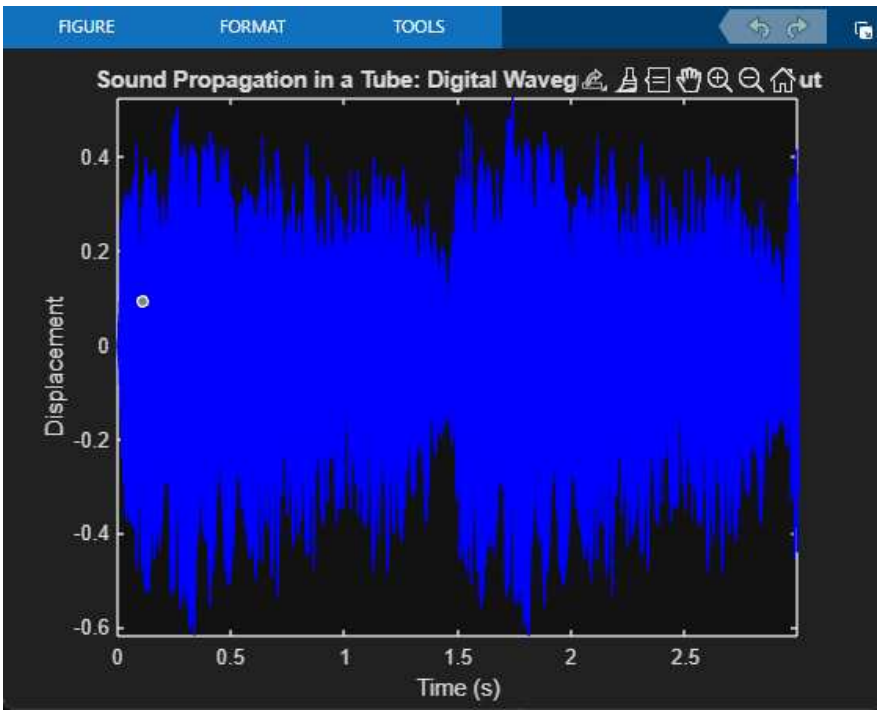


Figure 3.25: The short bore and brass bell shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

3.5.3 Long Bore an Brass Bell Shape

The same process as before was done for another shape - long bore and brass bell (Figure 3.26).

Figure 3.26: Shape of a long bore and brass bell input data.

The first column signalling the first segment, while the second column signals the next segment. You can see there are the same numbers in the n row in the second column as there are in the $n+1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

left	right
5	5
5	5
...	...
5	5.025
5.025	5.100
5.100	5.226
5.226	5.404
5.404	5.635
5.635	5.921
5.921	6.265
6.265	6.670
6.670	7.139
7.139	7.680
7.680	8.297
8.297	9.000
9.000	9.801
9.801	10.717
10.717	11.771
11.771	13.000
13.000	14.464
14.464	16.282
16.282	18.755
18.755	25.000

Table 3.3: Table of input data of a long bore and brass bell shape.

The output of testing this type of shape with input of zeros and with different configurations are as follows (the simulations were run for 3 seconds):

1. open at both ends (*df = 0; db = 0*) (Figure 3.27),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.28),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figure 3.29),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.30).

Next I tested it with the sound of the .wav file and left end closed and right side partially open $df = 1; db = 0.5$ (Figure 3.31).

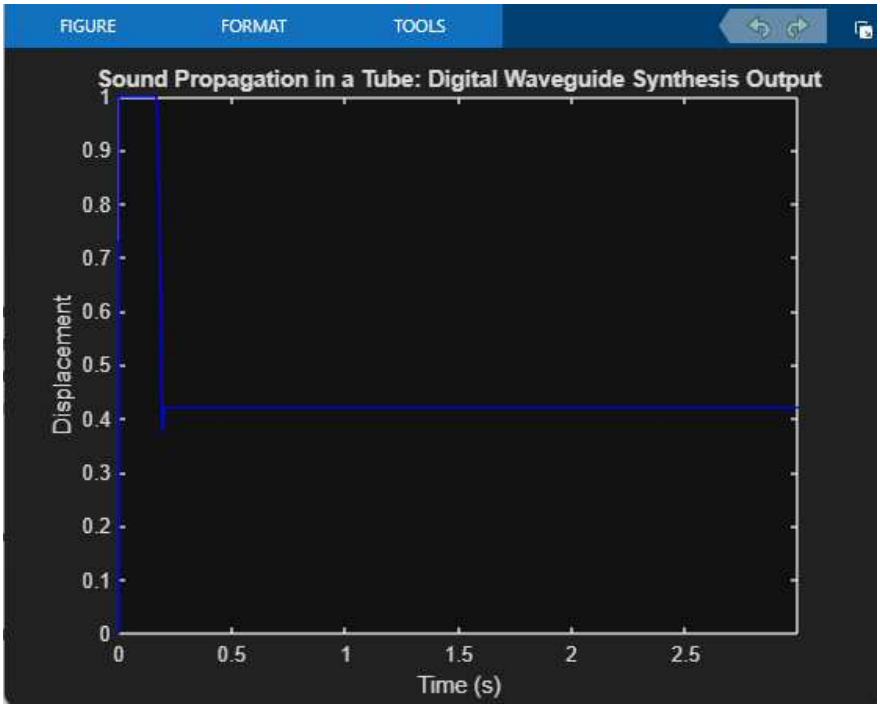


Figure 3.27: The long bore and brass bell shape with both ends open $df = 0; db = 0$ with input of zeros.

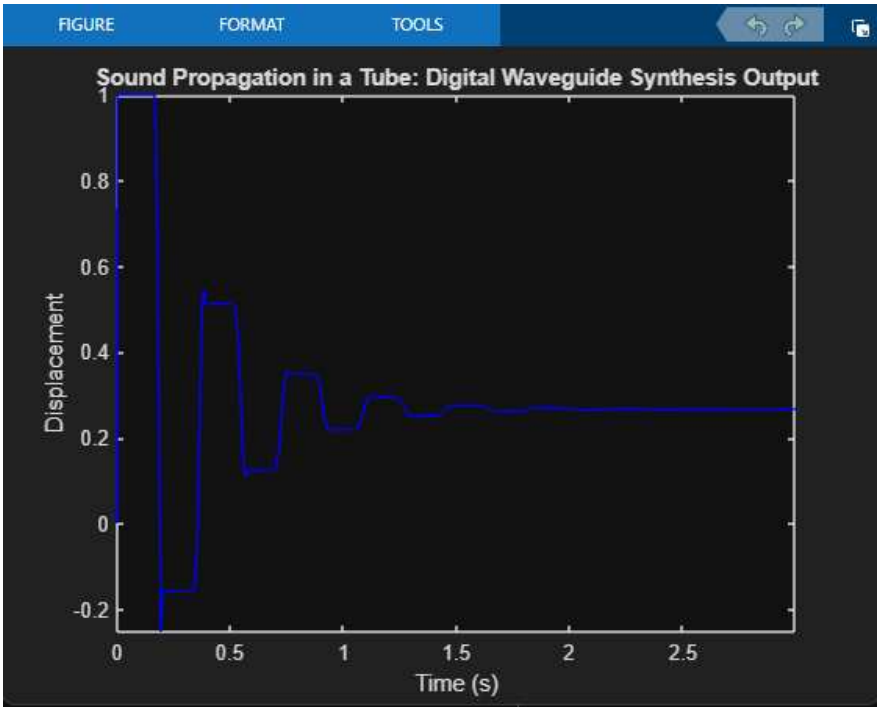


Figure 3.28: The long bore and brass bell shape with left end closed and right open $df = 1; db = 0$ with input of zeros.

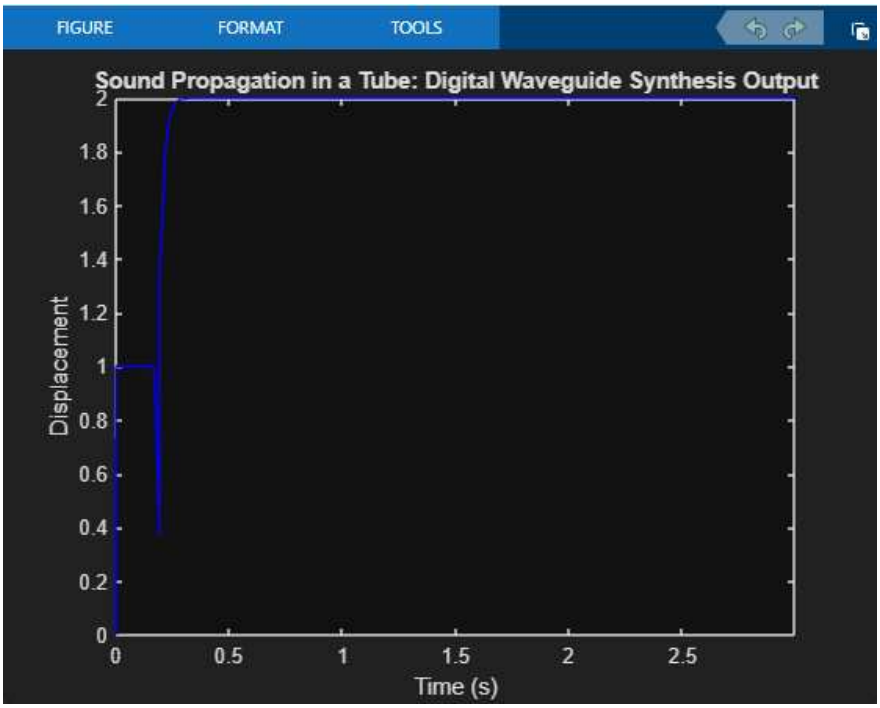


Figure 3.29: The long bore and brass bell shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

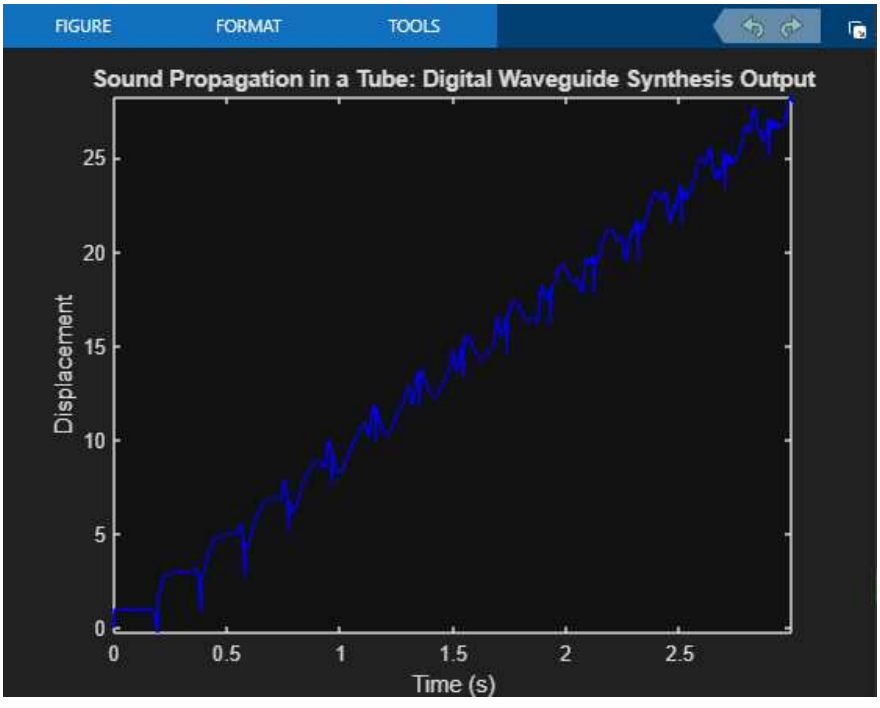


Figure 3.30: The long bore and brass bell shape with both ends closed $df = 1; db = 1$ with input of zeros.

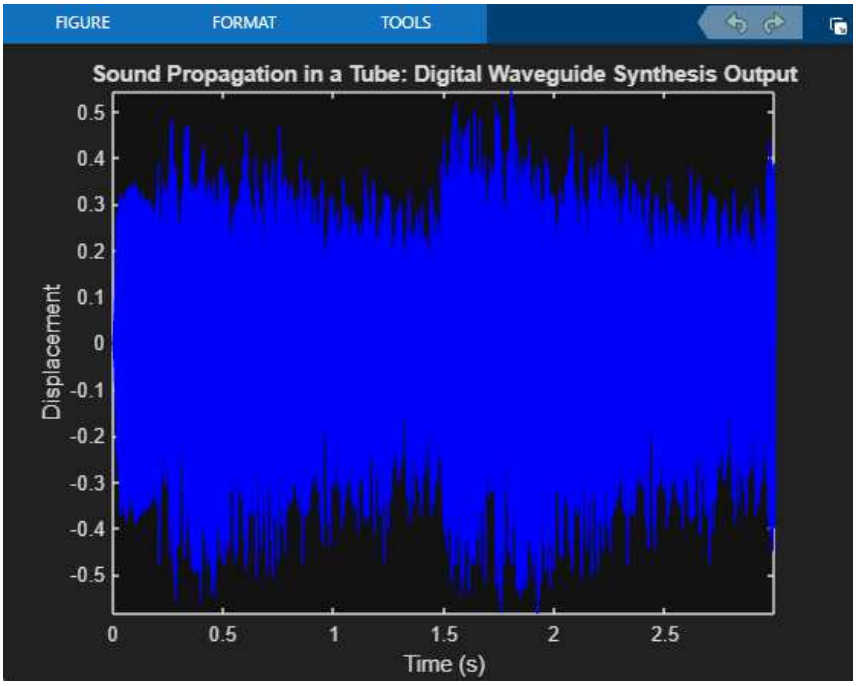


Figure 3.31: The long bore and brass bell shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

3.5.4 Round Shape

The same process as before was done for another shape - a round one (Figure 3.32).

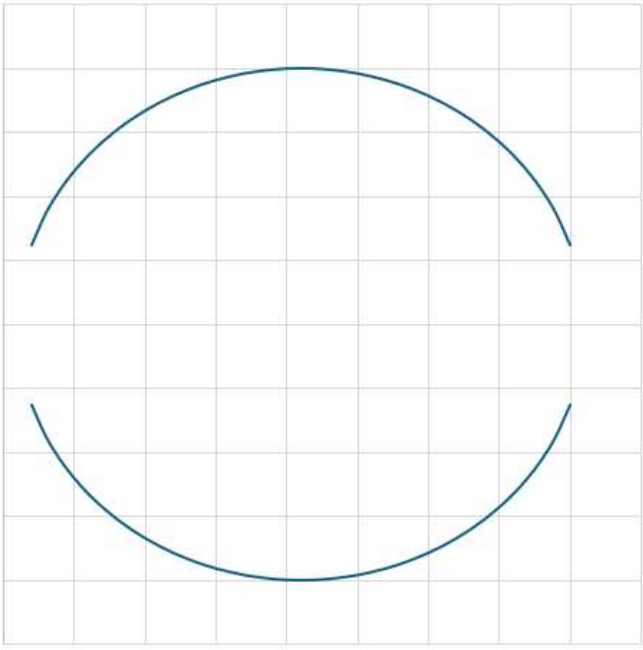


Figure 3.32: A round shape input data.

left	right
-6.25	-8.72
-8.72	-10.54
-10.54	-12
-12	-13.23
-13.23	-14.28
-14.28	-15.2
-15.2	-16
-16	-16.7
-16.7	-17.32
-17.32	-17.86
-17.86	-18.33
-18.33	-18.74
-18.74	-19.08
-19.08	-19.36
-19.36	-19.6
-19.6	-19.77
-19.77	-19.9
-19.9	-19.98
-19.98	-20

-20	-19.98
-19.98	-19.9
-19.9	-19.77
-19.77	-19.6
-19.6	-19.36
-19.36	-19.08
-19.08	-18.74
-18.74	-18.33
-18.33	-17.86
-17.86	-17.32
-17.32	-16.7
-16.7	-16
-16	-15.2
-15.2	-14.28
-14.28	-13.23
-13.23	-12
-12	-10.54
-10.54	-8.72
-8.72	-6.25

Table 3.4: Table of input data of a small round shape.

The first column signalling the first segment, while the second column signals the next segment. You can see there are the same numbers in the n row in the second column as there are in the $n+1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

The output of testing this type of shape with input of zeros and with different configurations are as follows (the simulations were run for 1 second):

1. open at both ends (*df = 0; db = 0*) (Figure 3.33),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.34),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figure 3.35),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.36 and Figure 3.37).

Next I tested it with the sound of the .wav file and left end closed and right side partially open $df = 1; db = 0.5$ (Figures 3.38 and 3.39).

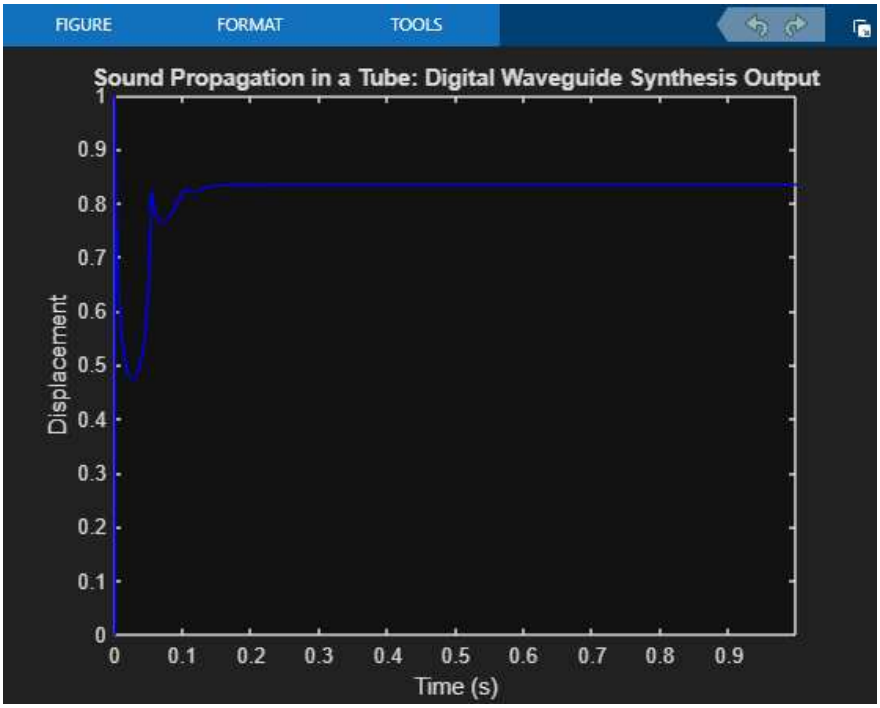


Figure 3.33: The round shape with both ends open $df = 0; db = 0$ with input of zeros.

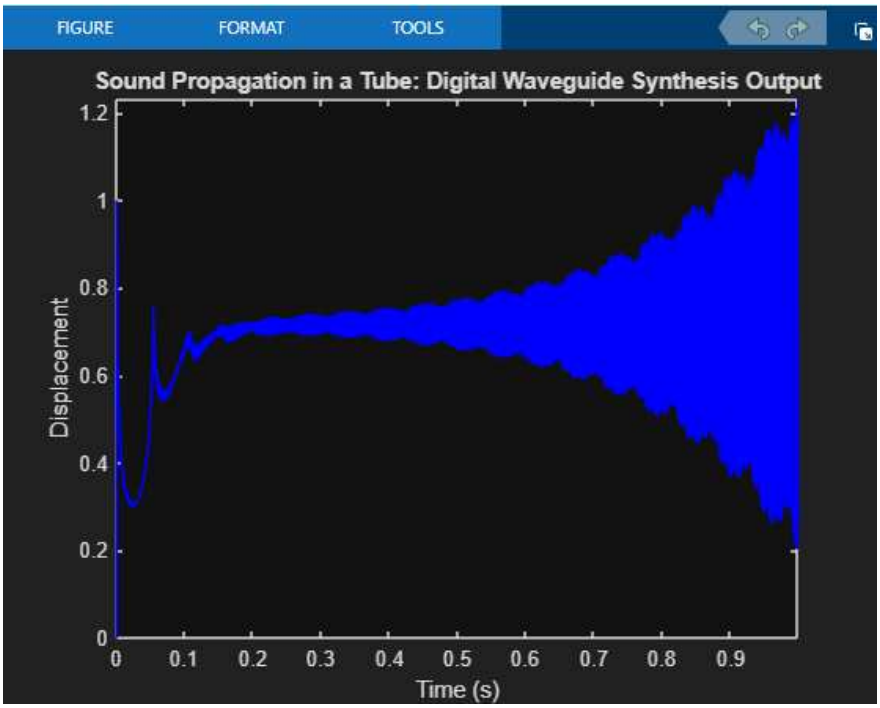


Figure 3.34: The round shape with left end closed and right open $df = 1; db = 0$ with input of zeros.

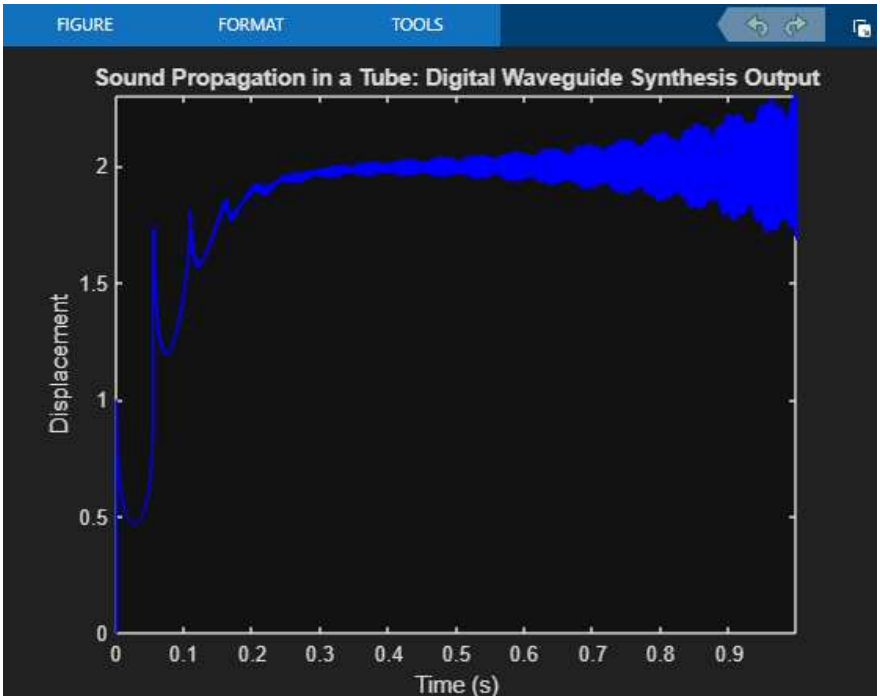


Figure 3.35: The round shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

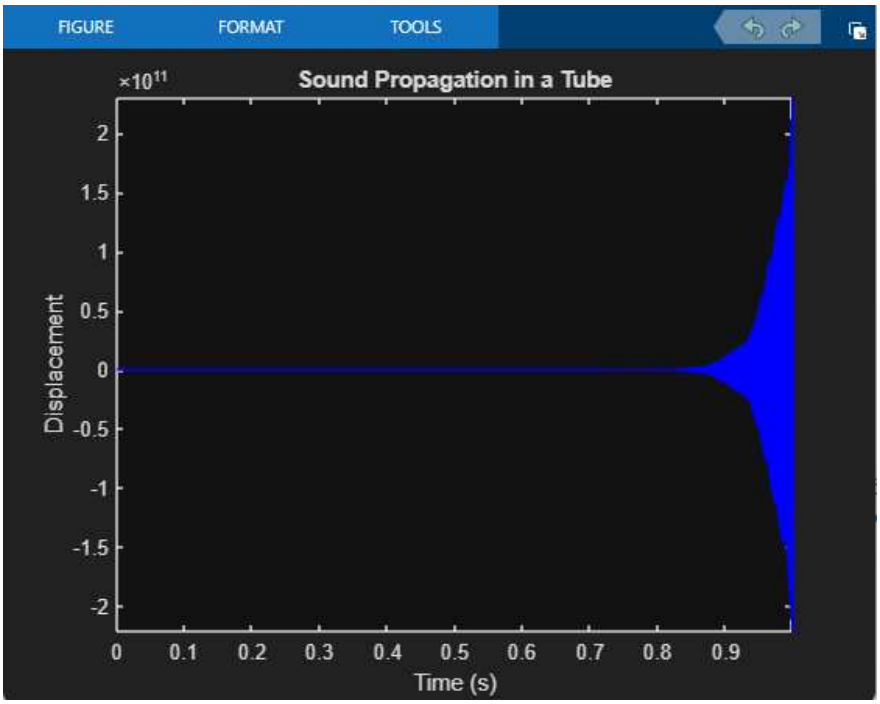


Figure 3.36: The round shape with both ends closed $df = 1; db = 1$ with input of zeros.

When we run the simulation for 0.1 second, the output is shown on Figure 3.37.

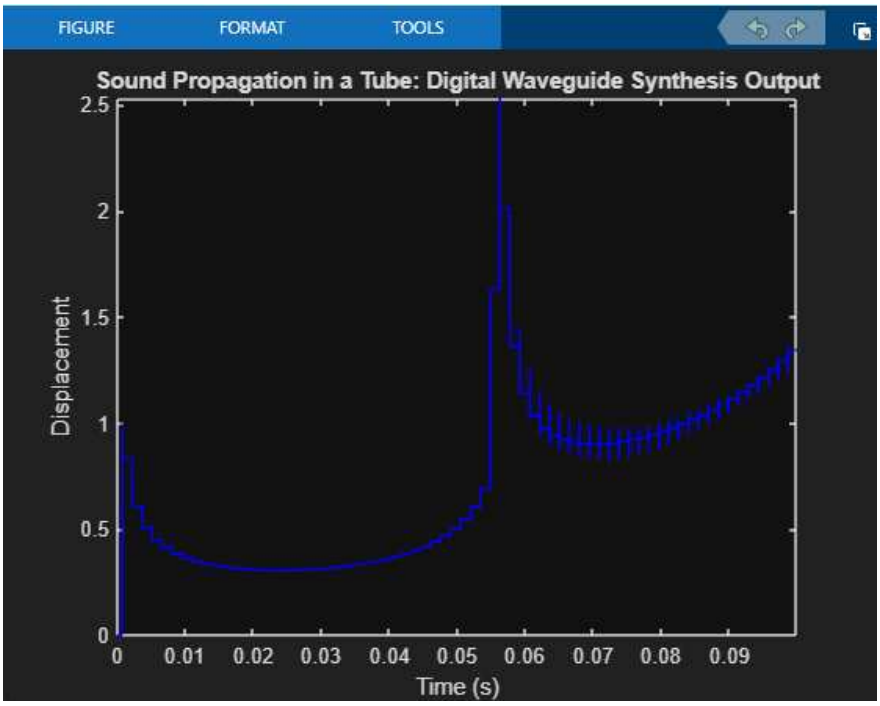


Figure 3.37: The round shape with both ends closed $df = 1; db = 1$ with input of zeros and simulation time of 0.1 s.

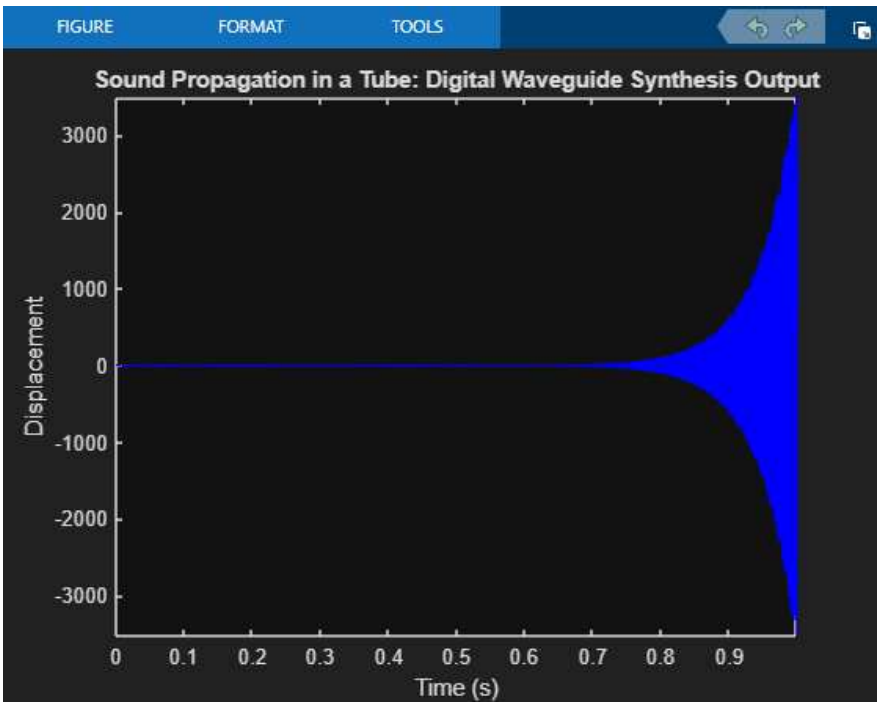


Figure 3.38: The round shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

When we run the simulation for 0.1 second, the output is shown on Figure 3.39.

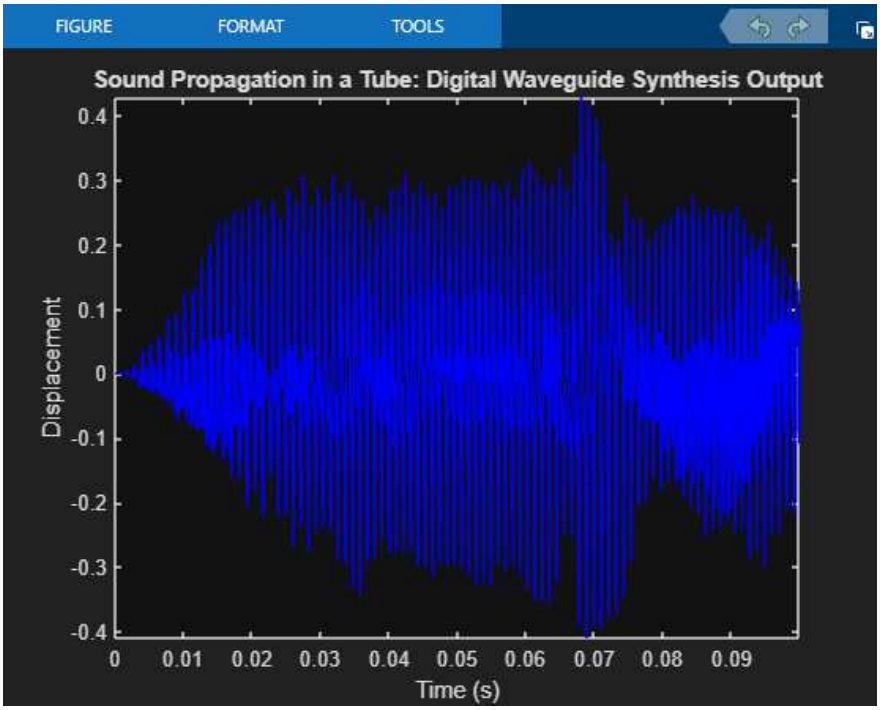


Figure 3.39: The round shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$ with the simulation time of 0.1 s.

3.5.5 Half-Triangular Shape

The same process as before was done for another shape - a half-triangular one (Figure 3.40).

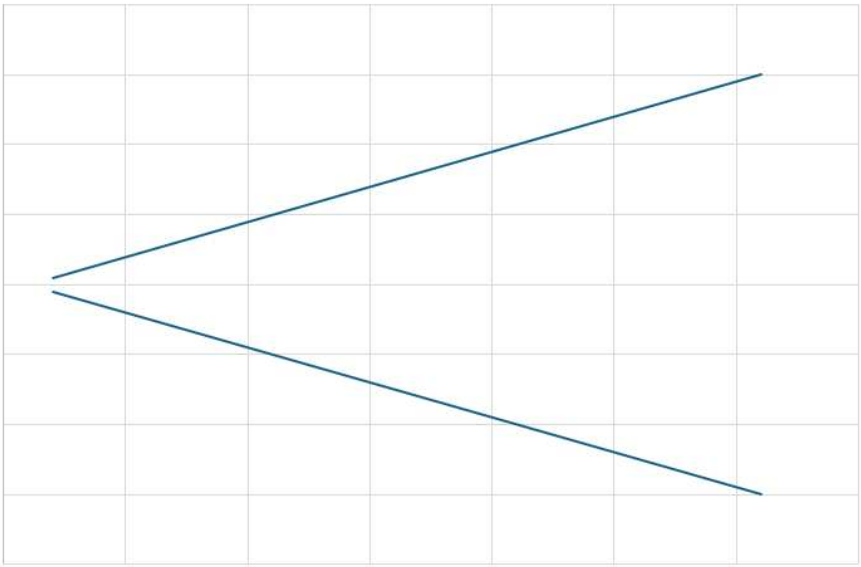


Figure 3.40: A half-triangular shape input data.

left	right
0.5	1
1	1.5
1.5	2
2	2.5
2.5	3
3	3.5
3.5	4
4	4.5
4.5	5
5	5.5
5.5	6
6	6.5
6.5	7
7	7.5
7.5	8
8	8.5
8.5	9
9	9.5
9.5	10
10	10.5
10.5	11
11	11.5
11.5	12
12	12.5
12.5	13
13	13.5
13.5	14
14	14.5
14.5	15

Table 3.5: Table of input data of a half-triangular shape.

The first column signalling the first segment, while the second column signals the next segment. You can see there are the same numbers in the n row in the second column as there are in the $n+1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

The output of testing this type of shape with input of zeros and with different configurations are as follows (with the simulation running time of 1 s):

1. open at both ends (*df = 0; db = 0*) (Figure 3.41),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.42),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figures 3.43 and 3.44),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.45).

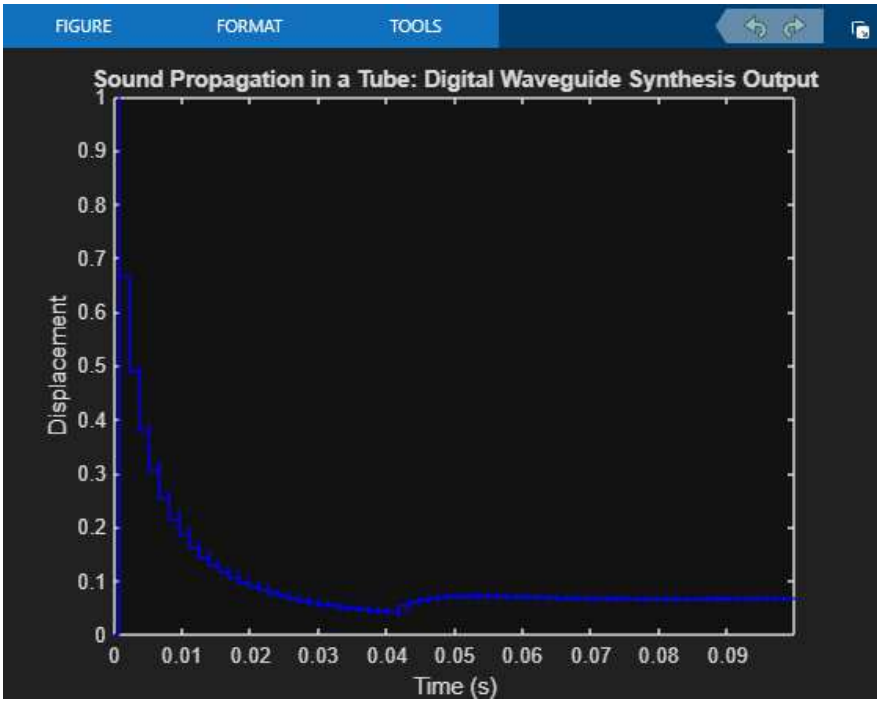


Figure 3.41: The half-triangular shape with both ends open $df = 0; db = 0$ with input of zeros and the simulation time of 0.1 s.

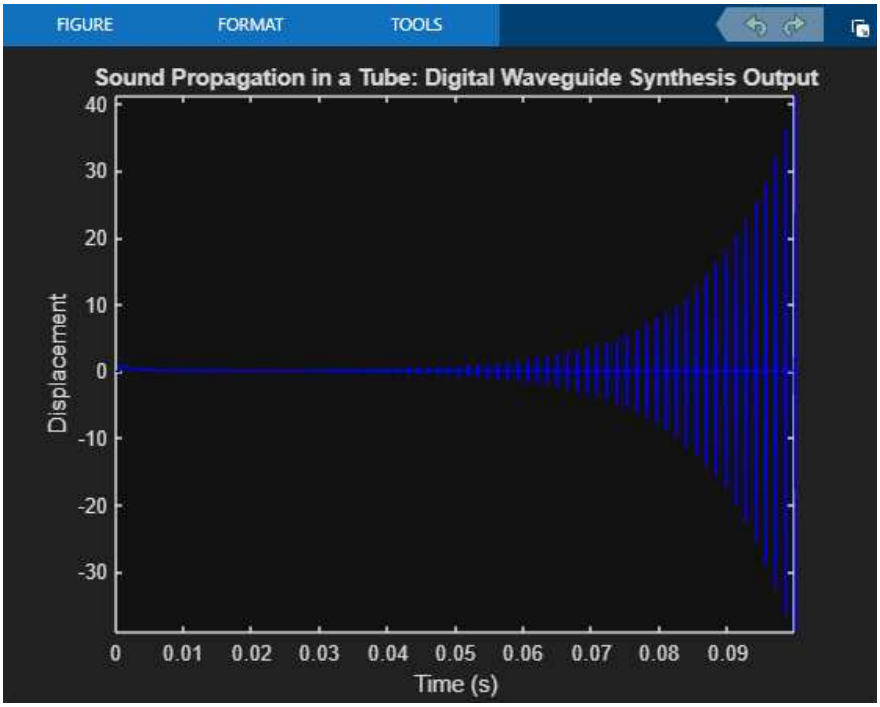


Figure 3.42: The half-triangular shape with left end closed and right open $df = 1; db = 0$ with input of zeros and the simulation time of 0.1 s.

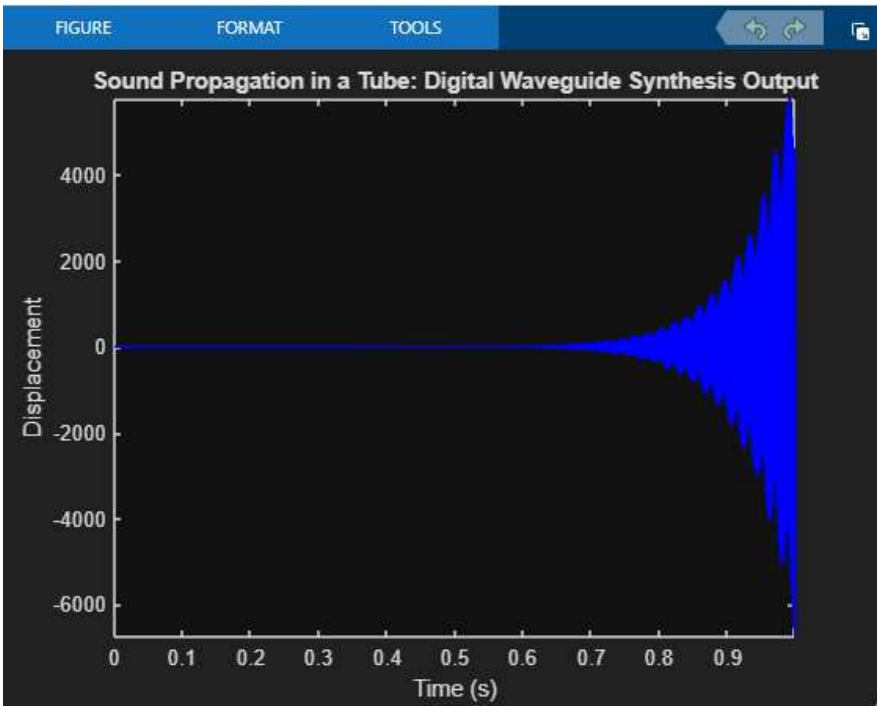


Figure 3.43: The half-triangular shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

When we run the simulation for 0.1 second, the output is shown on Figure 3.44.

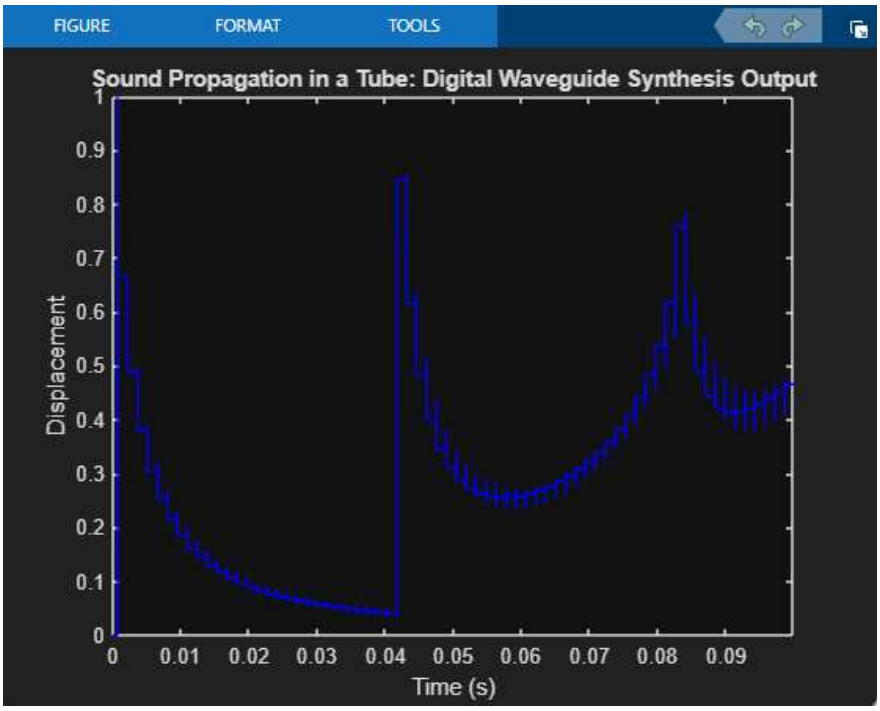


Figure 3.44: The half-triangular shape with left end open and right end closed $df = 0; db = 1$ with input of zeros and simulation time of 0.1 s.

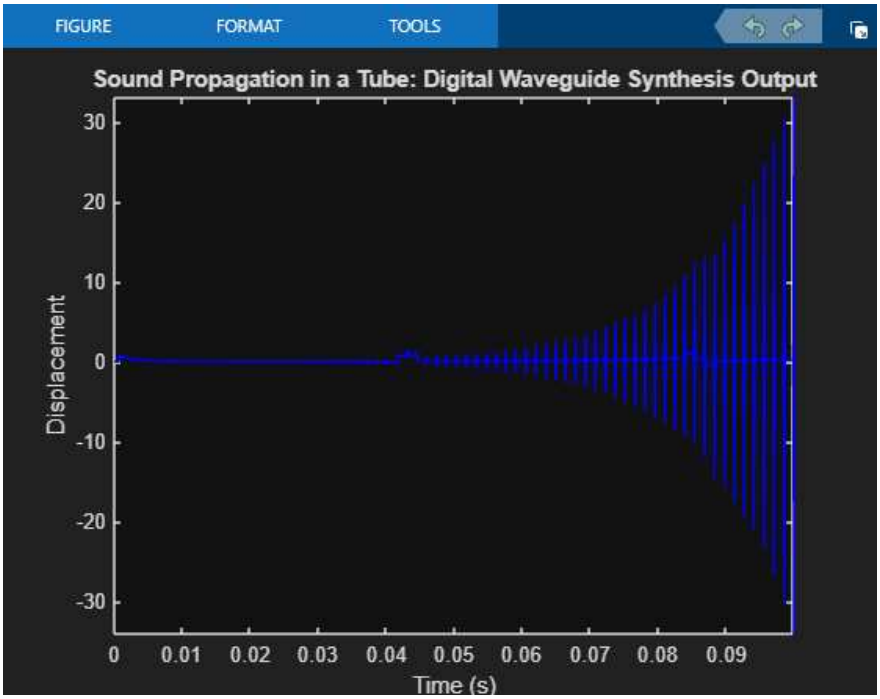


Figure 3.45: The half-triangular shape with both ends closed $df = 1; db = 1$ with input of zeros with the simulation time of 0.1 s.

Next I tested it with the sound of the .wav file and left end closed and right side partially open $df = 1; db = 0.5$ (Figures 3.46 and 3.47).

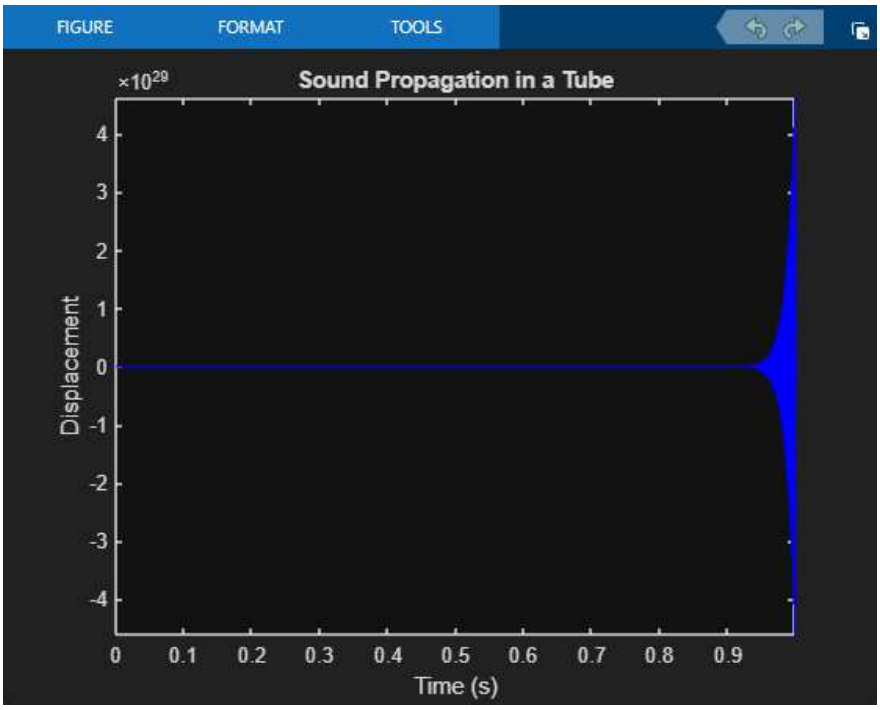


Figure 3.46: The half-triangular shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

When we run the simulation for 0.1 second, the output is shown on Figure 3.47.

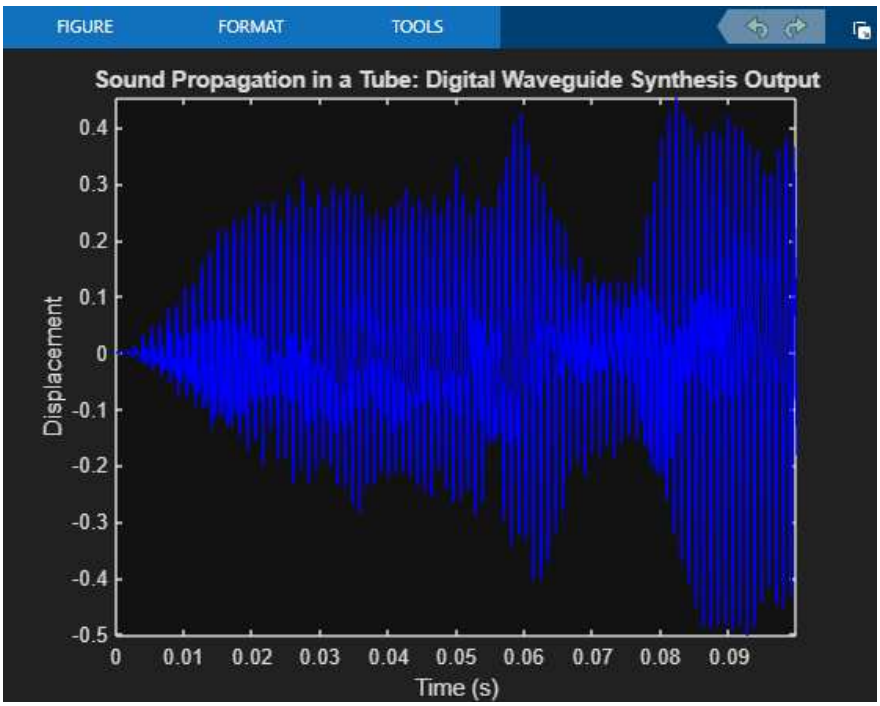


Figure 3.47: The half-triangular shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$ and simulation time of 0.1 s.

3.5.6 Small Triangular Shape

The same process as before was done for another shape - small triangular (Figure 3.48).

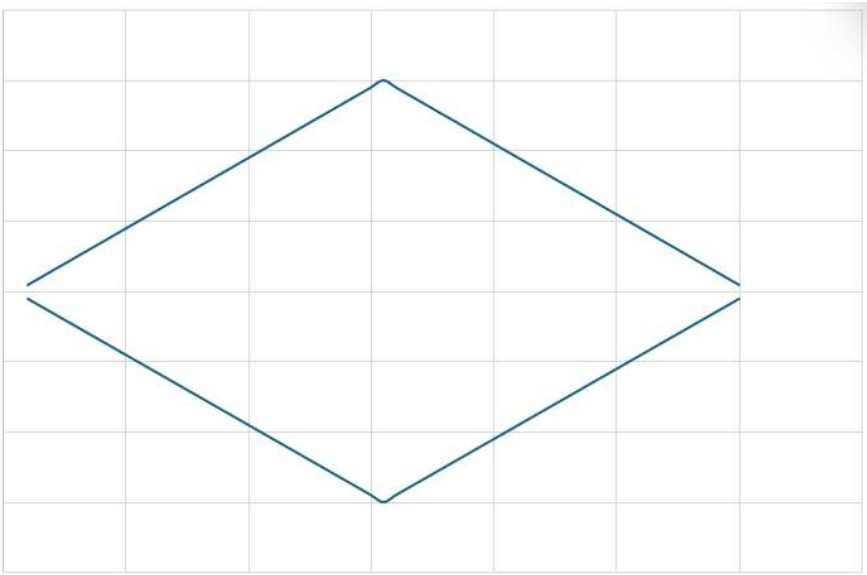


Figure 3.48: A small triangular shape input data.

left	right		
0.1	0.2	3	2.9
0.2	0.3	2.9	2.8
0.3	0.4	2.8	2.7
0.4	0.5	2.7	2.6
0.5	0.6	2.6	2.5
0.6	0.7	2.5	2.4
0.7	0.8	2.4	2.3
0.8	0.9	2.3	2.2
0.9	1	2.2	2.1
1	1.1	2.1	2
1.1	1.2	2	1.9
1.2	1.3	1.9	1.8
1.3	1.4	1.8	1.7
1.4	1.5	1.7	1.6
1.5	1.6	1.6	1.5
1.6	1.7	1.5	1.4
1.7	1.8	1.4	1.3
1.8	1.9	1.3	1.2
1.9	2	1.2	1.1
2	2.1	1.1	1
2.1	2.2	1	0.9
2.2	2.3	0.9	0.8
2.3	2.4	0.8	0.7
2.4	2.5	0.7	0.6
2.5	2.6	0.6	0.5
2.6	2.7	0.5	0.4
2.7	2.8	0.4	0.3
2.8	2.9	0.3	0.2
2.9	3	0.2	0.1

Table 3.6: Table of input data of a small triangular shape.

The first column signalling the first segment, while the second column signals the next segment. You can see there are the same numbers in the n row in the second column as there are in the $n+1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

The output of testing this type of shape with input of zeros and with different configurations are as follows (the simulations were run for 1 second):

1. open at both ends (*df = 0; db = 0*) (Figures 3.49 and 3.50),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figures 3.51 and 3.52),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figures 3.53 and 3.54),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.55).

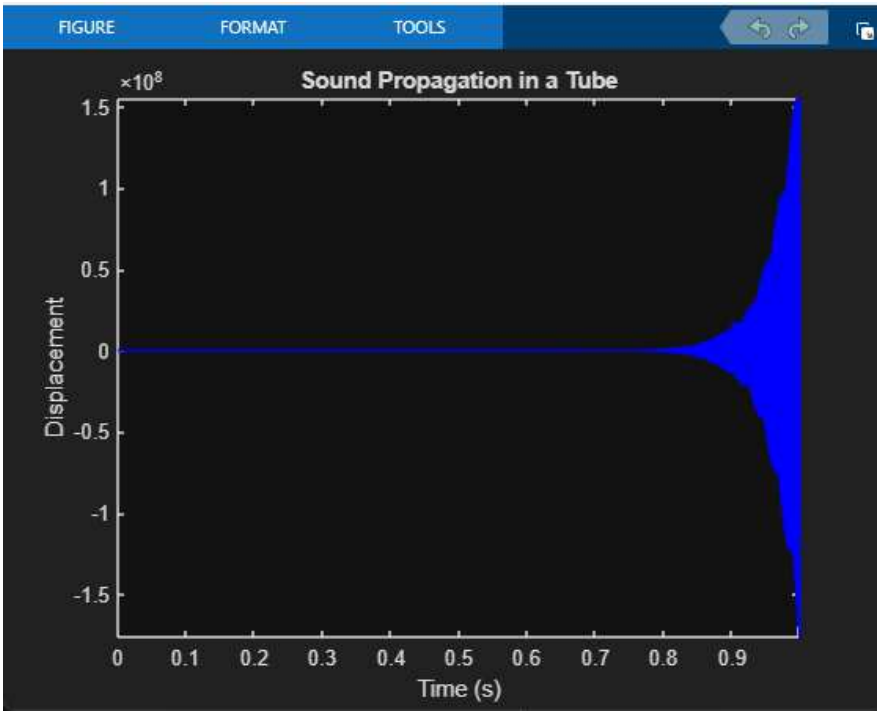


Figure 3.49: The small triangular shape with both ends open $df = 0; db = 0$ with input of zeros.

When we run the simulation for 0.1 second, the output is shown on Figure 3.50.

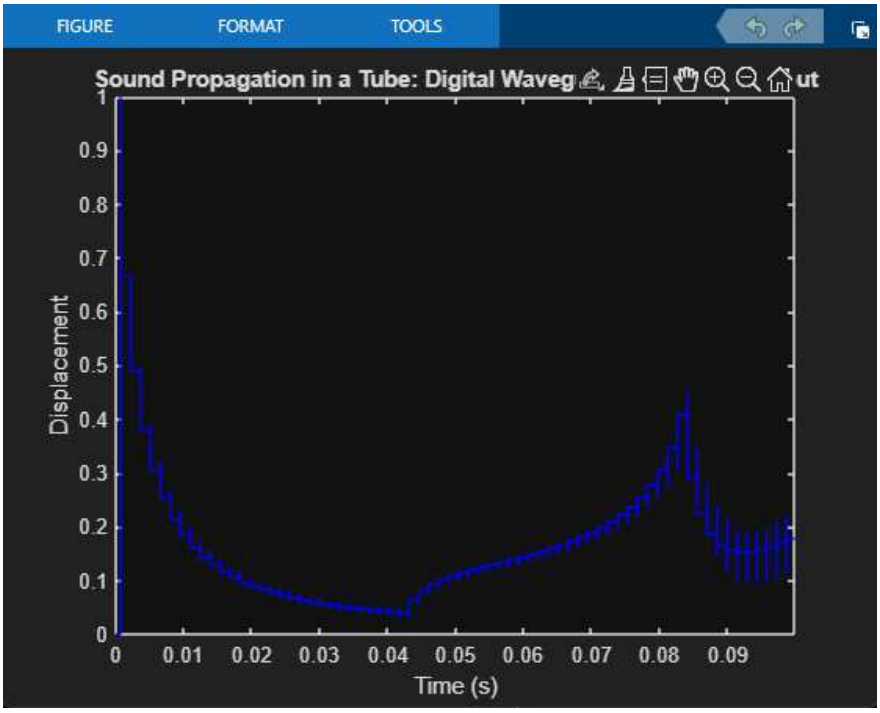


Figure 3.50: The small triangular shape with both ends open $df = 0; db = 0$ with input of zeros with the simulation running for 0.1 s.

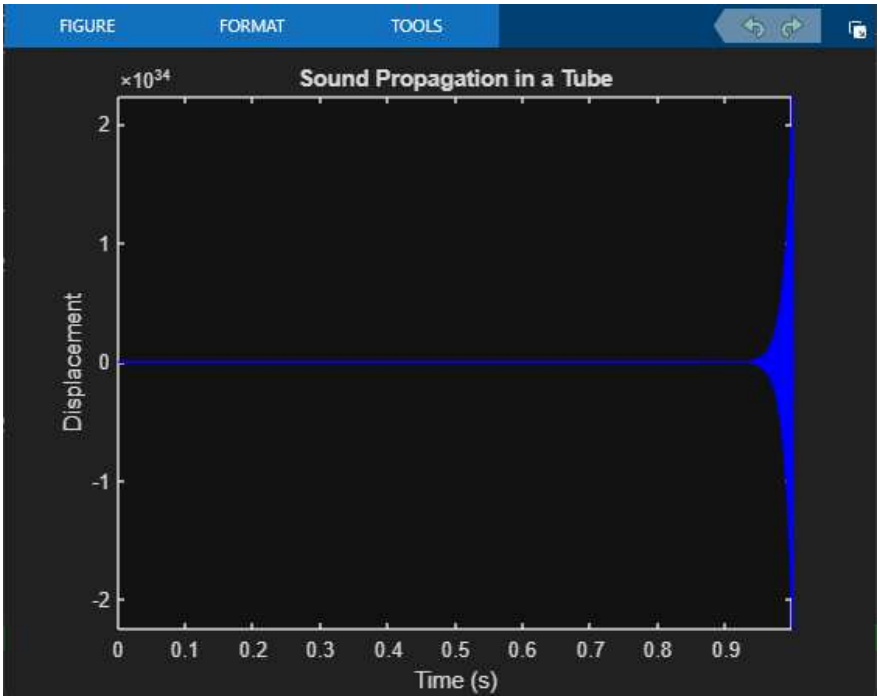


Figure 3.51: The small triangular shape with left end closed and right open $df = 1; db = 0$ with input of zeros.

When we run the simulation for 0.1 second, the output is shown on Figure 3.52.

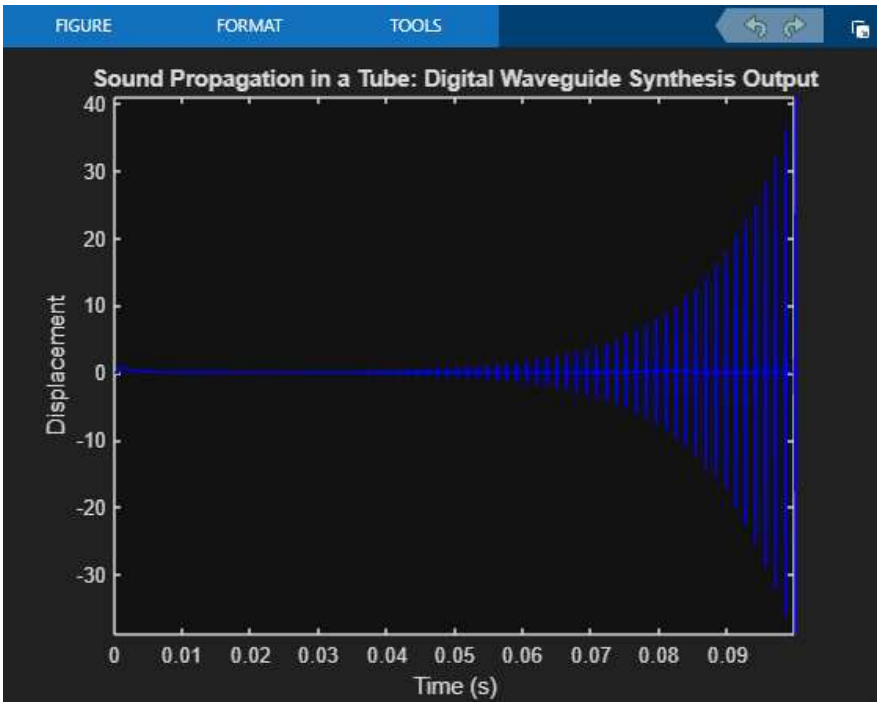


Figure 3.52: The small triangular shape with left end closed and right open $df = 1; db = 0$ with input of zeros and th simulation running for 0.1 s.

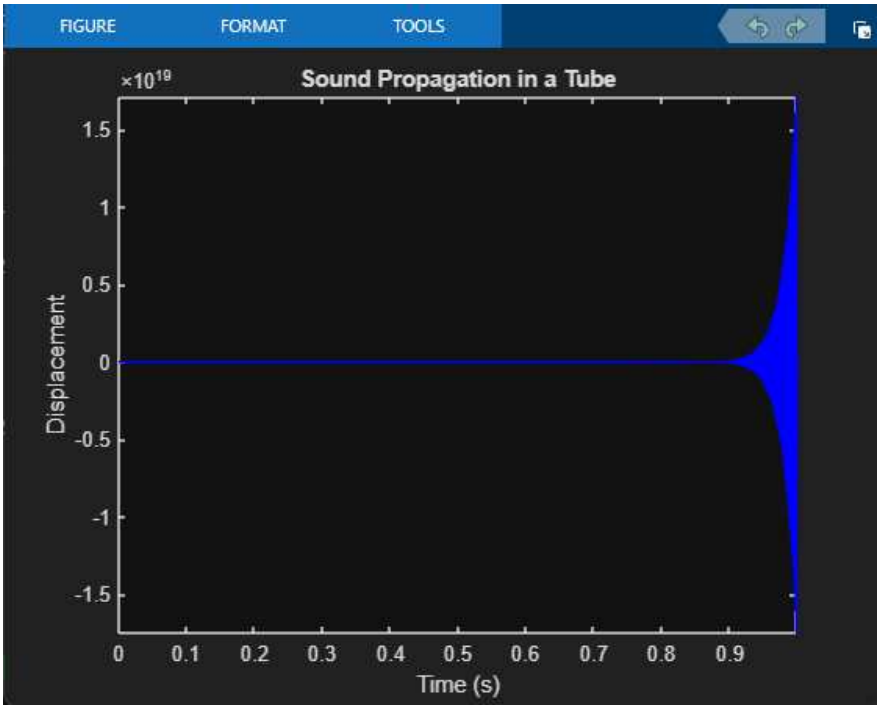


Figure 3.53: The small triangular shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

When we run the simulation for 0.1 second, the output is shown on Figue 3.54.

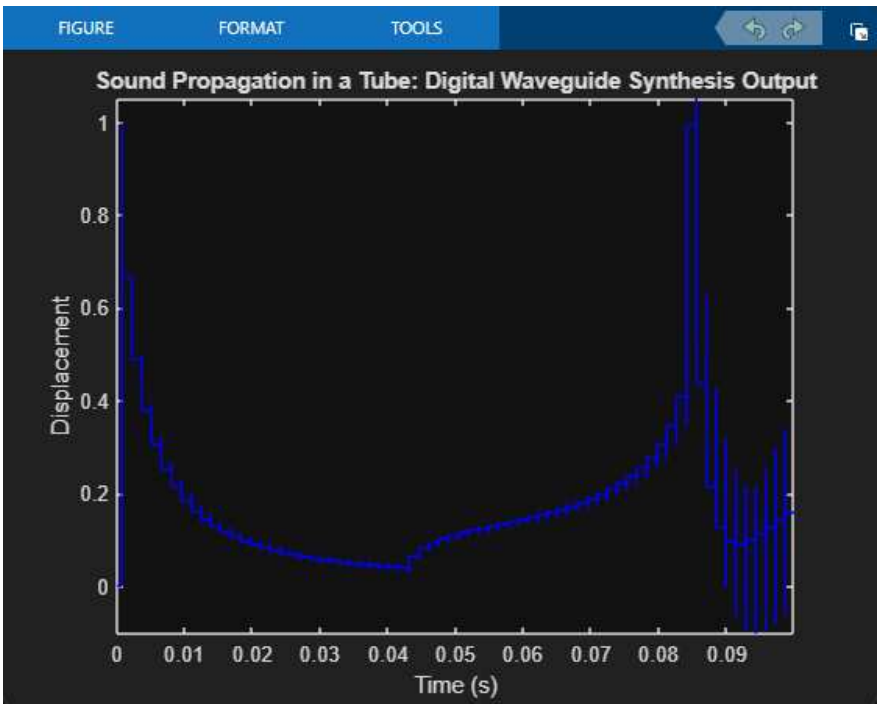


Figure 3.54: The small triangular shape with left end open and right end closed $df = 0; db = 1$ with input of zeros and the simulation run for 0.1 s.

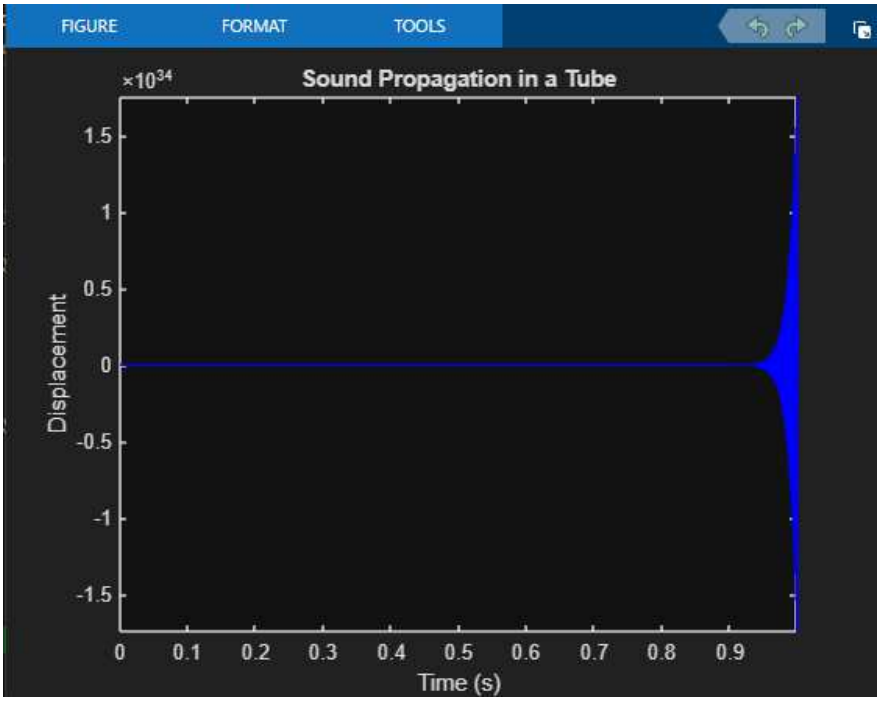


Figure 3.55: The small triangular shape with both ends closed $df = 1; db = 1$ with input of zeros.

Next I tested it with the sound of the .wav file and left end closed and right side partially open $df = 1; db = 0.5$ (Figure 3.56).

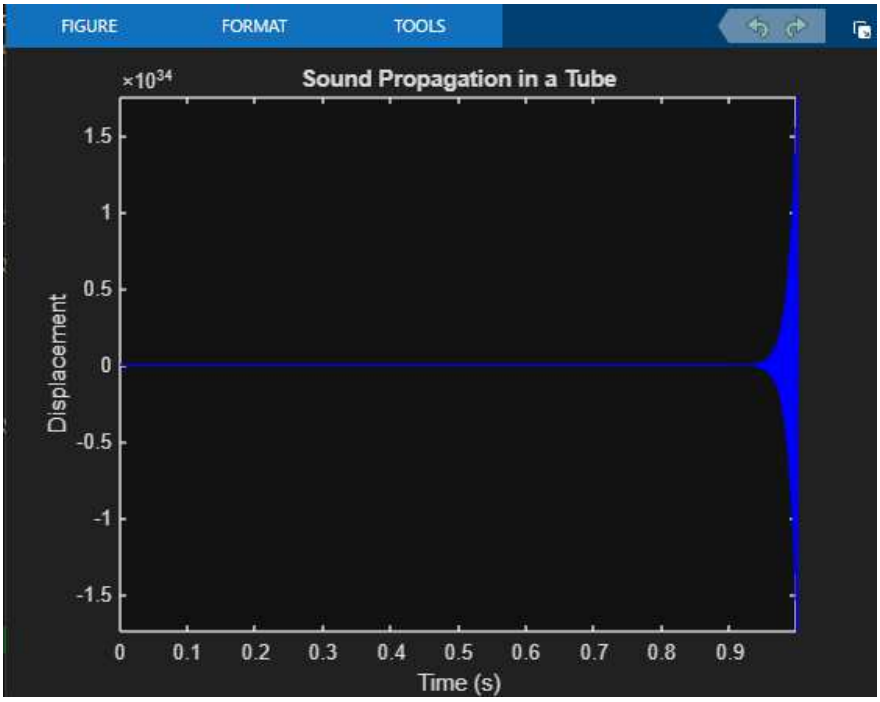


Figure 3.56: The small triangular shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

3.5.7 Big Triangular Shape

The same process as before was done for another shape - a big triangular one (Figure 3.57).

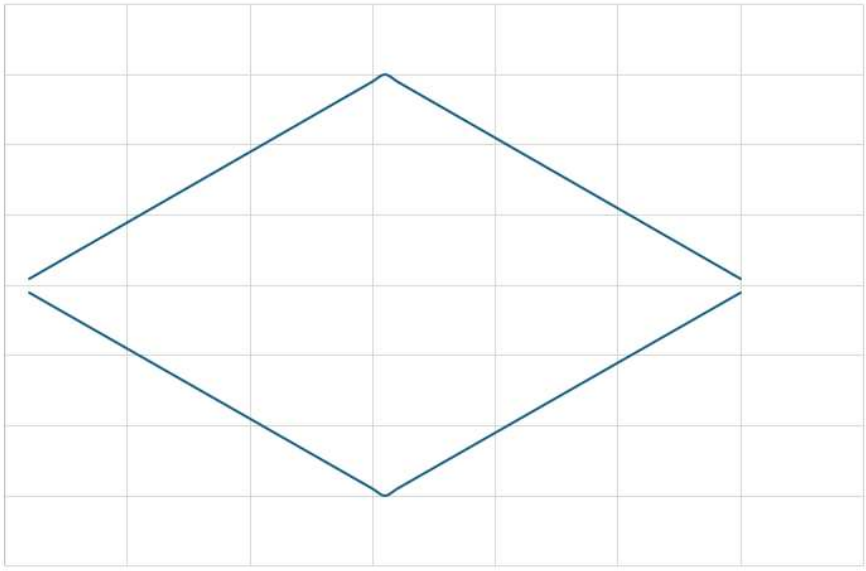


Figure 3.57: A big triangular shape input data.

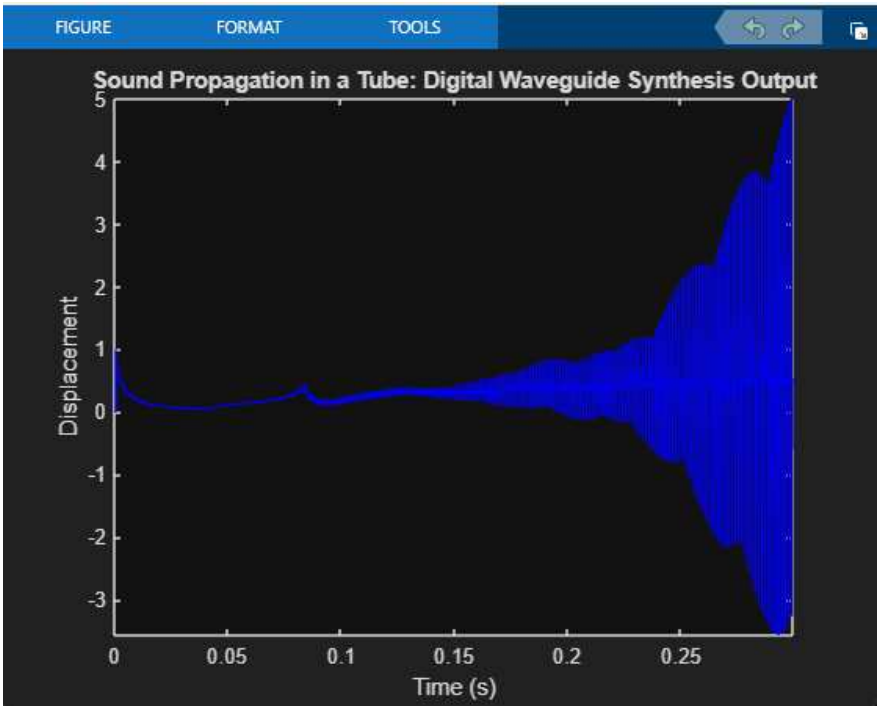


Figure 3.58: The big triangular shape with both ends open $df = 0; db = 0$ with input of zeros and simulation time of 0.3 s.

left	right		
1	2	30	29
2	3	29	28
3	4	28	27
4	5	27	26
5	6	26	25
6	7	25	24
7	8	24	23
8	9	23	22
9	10	22	21
10	11	21	20
11	12	20	19
12	13	19	18
...	...	...	...
19	20	12	11
20	21	11	10
21	22	10	9
22	23	9	8
23	24	8	7
24	25	7	6
25	26	6	5
26	27	5	4
27	28	4	3
28	29	3	2
29	30	2	1

Table 3.7: Table of input data of a big triangular shape.

The first column signalling the first segment, while the second column signals the next segment. You can see there are the same numbers in the n row in the second column as there are in the $n+1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

The output of testing this type of shape with input of zeros and with different configurations are as follows (with the simulation running time of 1 s):

1. open at both ends (*df = 0; db = 0*) (Figure 3.58),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.60),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figures 3.61 and 3.62),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.63).

Next I tested it with the sound of the .wav file and left end closed and right side partially open $df = 1; db = 0.5$ (Figure 3.64).

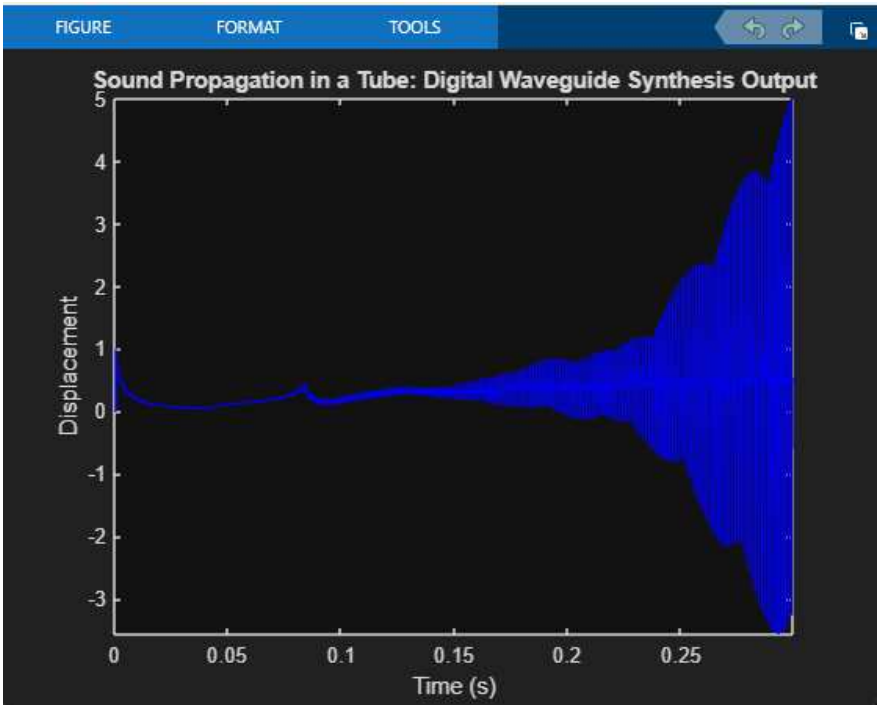


Figure 3.59: The big triangular shape with both ends open $df = 0; db = 0$ with input of zeros and simulation time of 0.3 s.

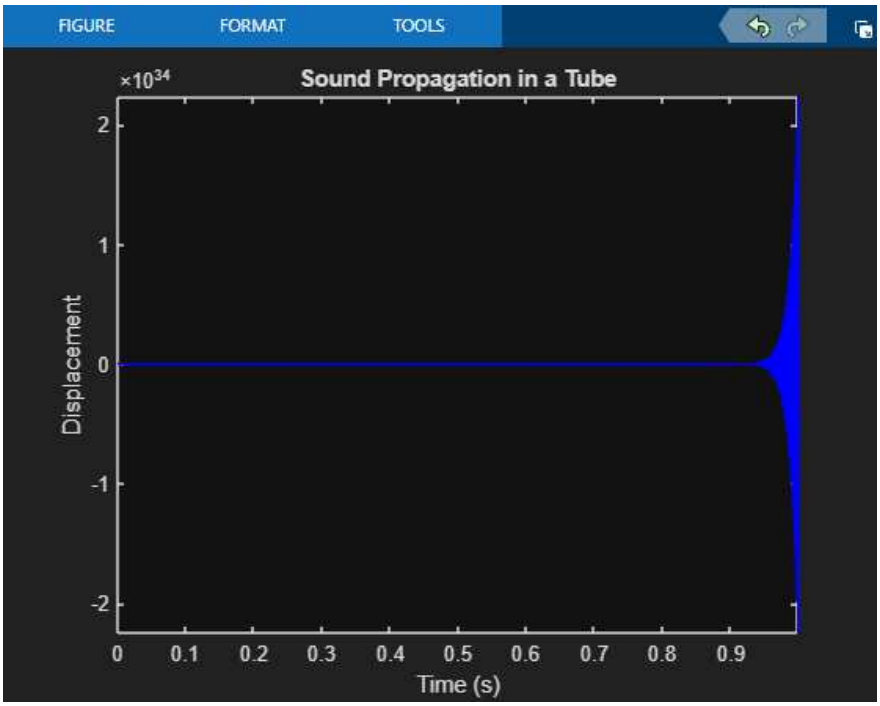


Figure 3.60: The big triangular shape with left end closed and right open $df = 1; db = 0$ with input of zeros.

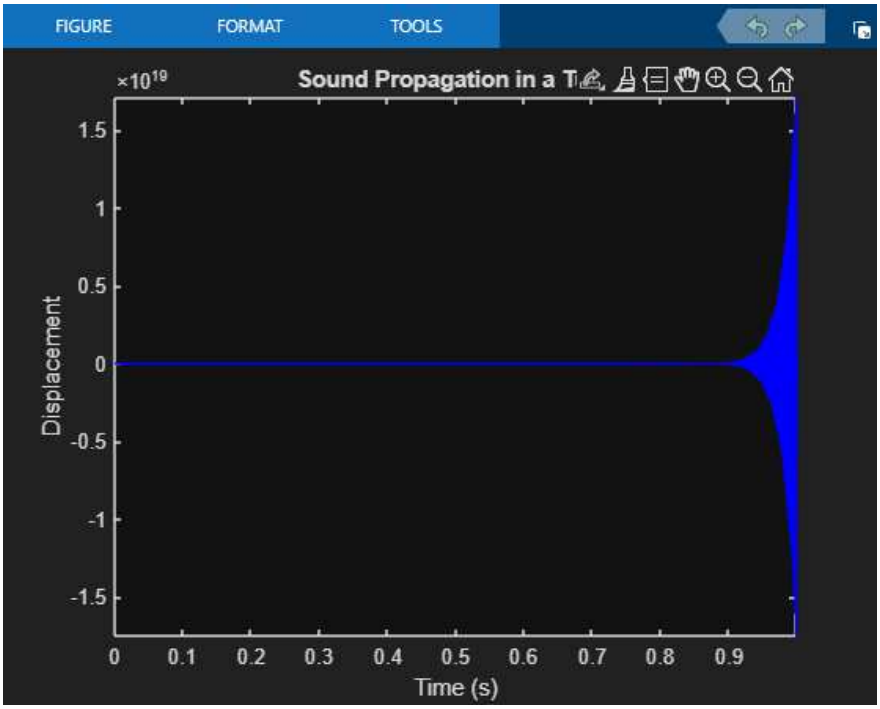


Figure 3.61: The big triangular shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

When we run the simulation for 0.1 second, the output is as in Figure 3.62.

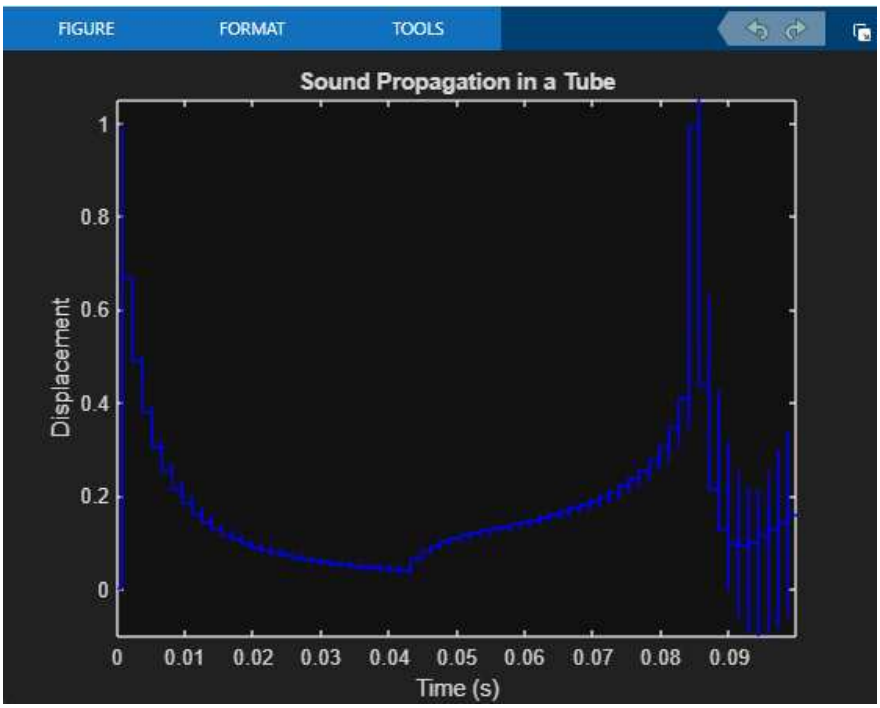


Figure 3.62: The big triangular shape with left end open and right end closed $df = 0; db = 1$ with input of zeros and the simulation time of 0.1 s.

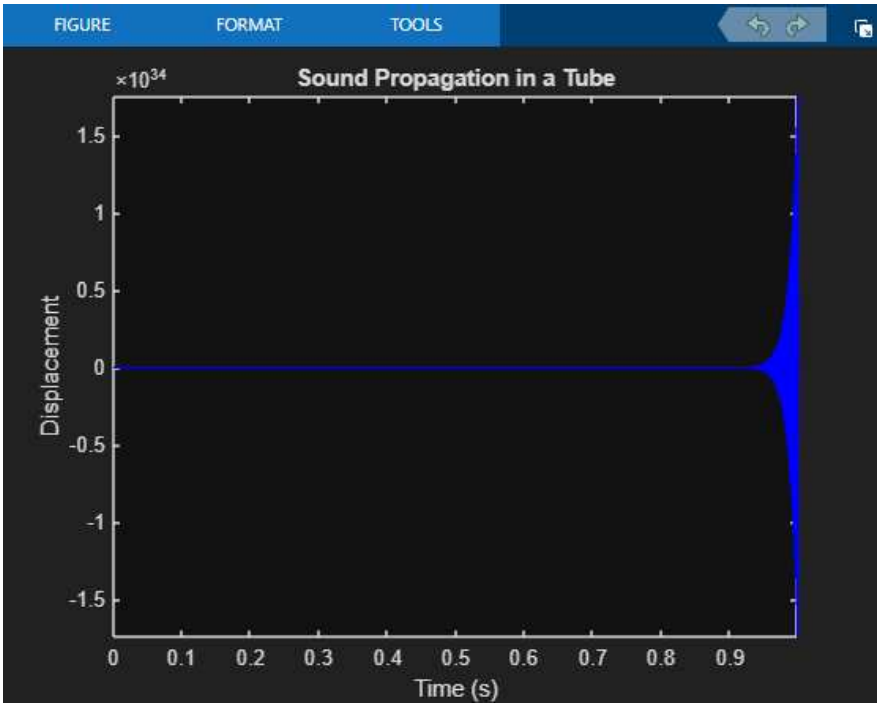


Figure 3.63: The big triangular shape with both ends closed $df = 1; db = 1$ with input of zeros.

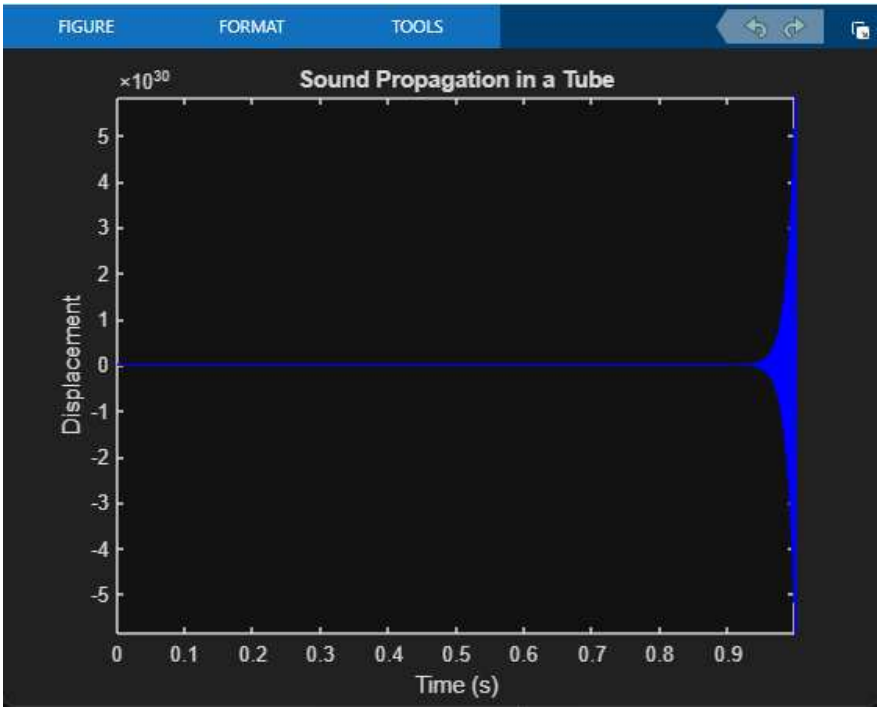


Figure 3.64: The big triangular shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

3.5.8 Round With a Long Tube Shape

The same process as before was done for another shape - a round with a long tube one (Figure 3.65).

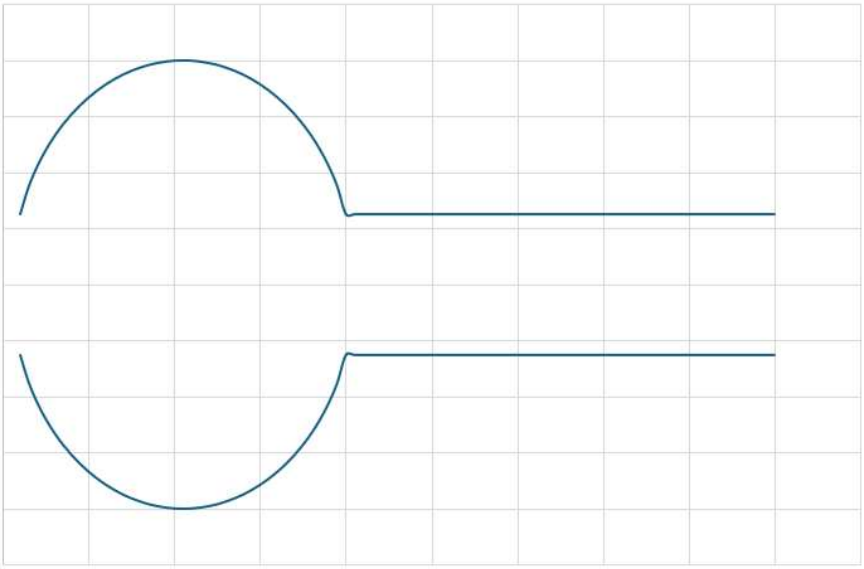


Figure 3.65: A round with a long tube shape input data.

left	right
-6.25	-8.72
-8.72	-10.54
-10.54	-12
-12	-13.23
-13.23	-14.28
-14.28	-15.2
-15.2	-16
-16	-16.7
-16.7	-17.32
-17.32	-17.86
-17.86	-18.33
-18.33	-18.74
-18.74	-19.08
-19.08	-19.36
-19.36	-19.6
-19.6	-19.77
-19.77	-19.9
-19.9	-19.98
-19.98	-20
-20	-19.98
-19.98	-19.9
-19.9	-19.77
-19.77	-19.6
-19.6	-19.36
-19.36	-19.08
-19.08	-18.74
-18.74	-18.33
-18.33	-17.86
-17.86	-17.32
-17.32	-16.7
-16.7	-16
-16	-15.2
-15.2	-14.28
-14.28	-13.23
-13.23	-12
-12	-10.54
-10.54	-8.72
-8.72	-6.25
-6.25	-6.25
-6.25	-6.25
...	...
-6.25	-6.25

Table 3.8: Table of input data of a round with a long tube shape.

The first column signalling the first segment, while the second column signals the next segment. You can see there are the same numbers in the n row in the second column as there are in the $n+1$ row in the first column. This is because the row is showing the situation at the junction of two segments on each junction.

The output of testing this type of shape with input of zeros and with different configurations are as follows (with the simulation running time of 1 s):

1. open at both ends (*df = 0; db = 0*) (Figure 3.66),
2. open at the right end, closed on the left (*df = 1; db = 0*) (Figure 3.67),
3. open on the left end, closed on the right (*df = 0; db = 1*) (Figure 3.68),
4. closed on both ends (*df = 1; db = 1)* (Figure 3.69).

Next I tested it with the sound of the .wav file and left end closed and right side partially open $df = 1; db = 0.5$ (Figure 3.70).

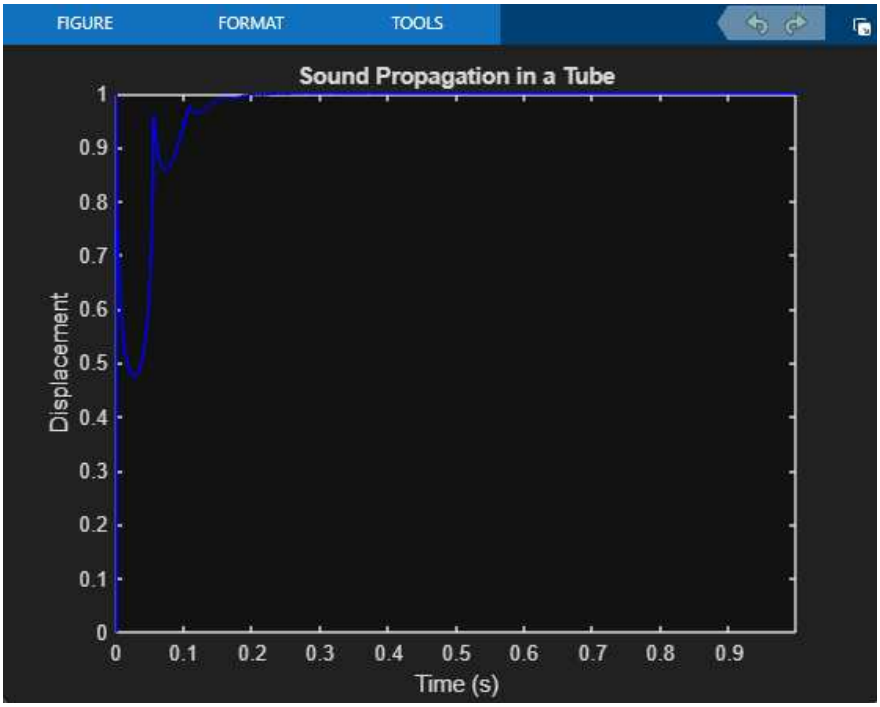


Figure 3.66: The round with a long tube shape with both ends open $df = 0; db = 0$ with input of zeros.

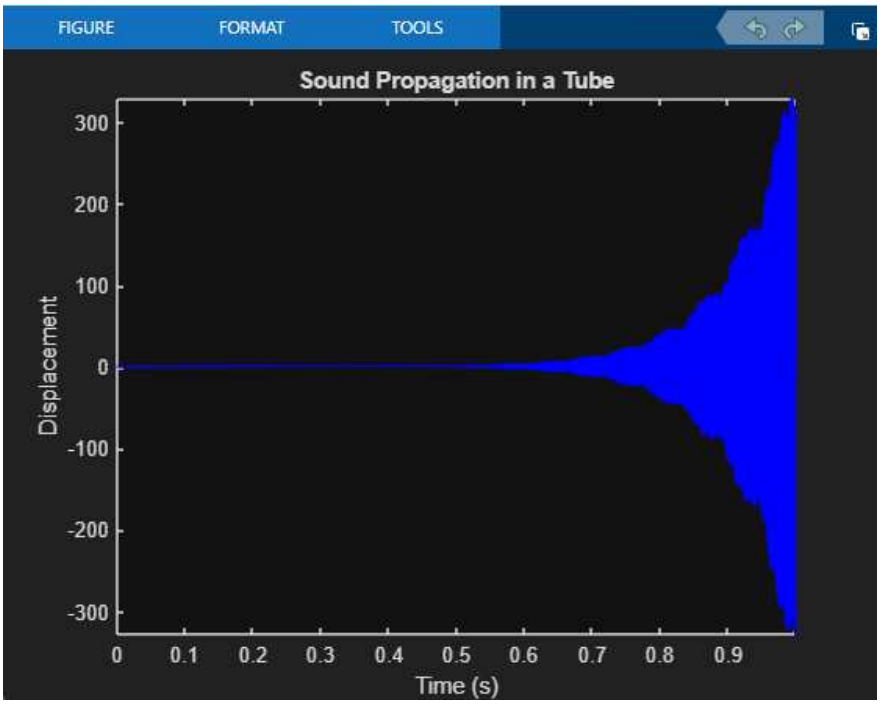


Figure 3.67: The round with a long tube shape with left end closed and right open $df = 1; db = 0$ with input of zeros.

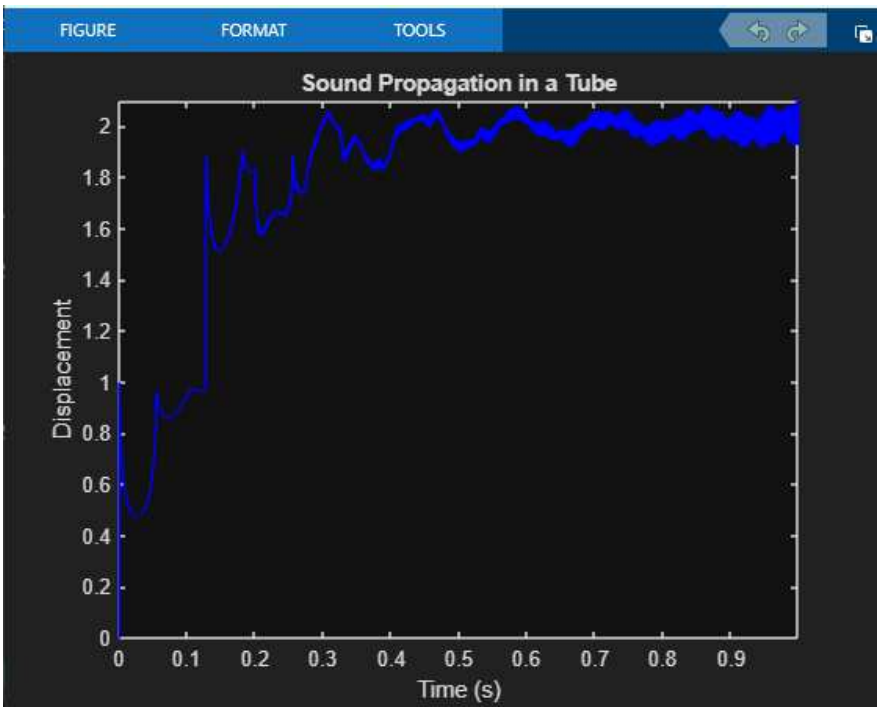


Figure 3.68: The round with a long tube shape with left end open and right end closed $df = 0; db = 1$ with input of zeros.

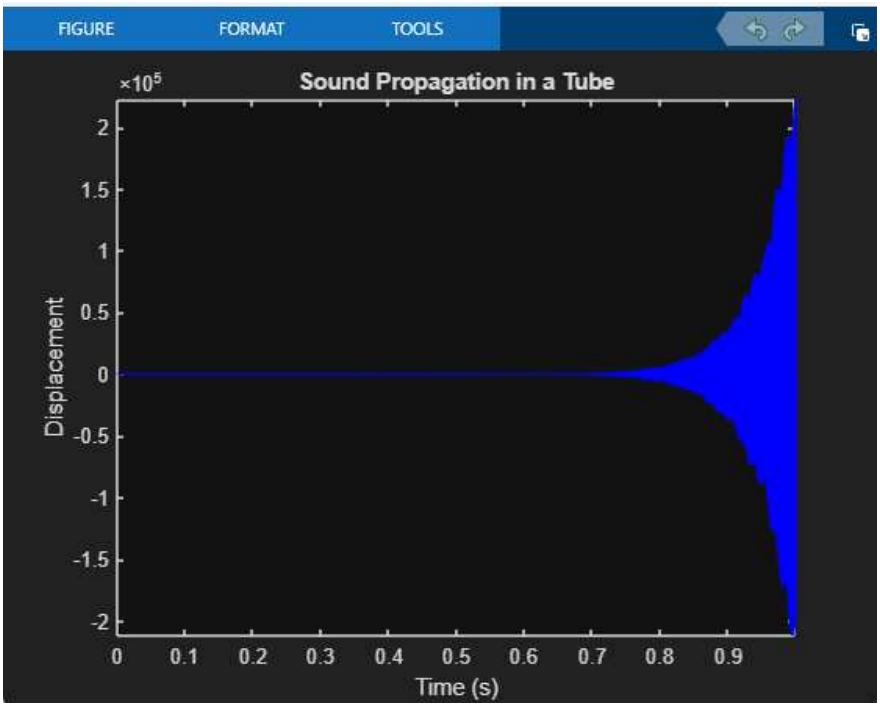


Figure 3.69: The round with a long tube shape with both ends closed $df = 1; db = 1$ with input of zeros.

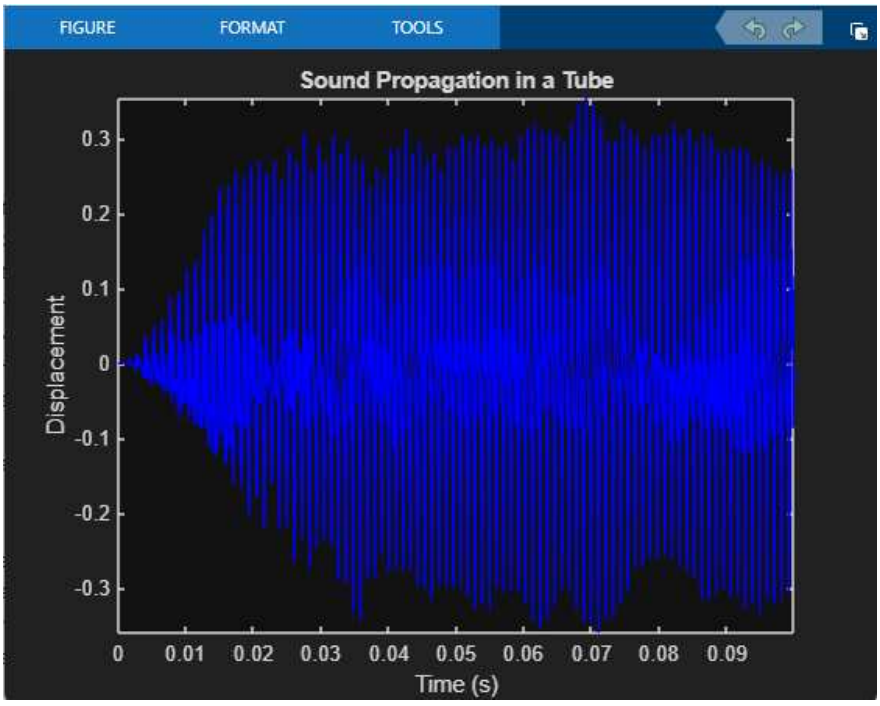


Figure 3.70: The round with a long tube shape with input as .wav file and left end closed and right end partially open $df = 1; db = 0.5$.

Chapter 4

Conclusions

The code produced models a working digital waveguide with the possibility to implement any intended shape. Based on that, we can develop any brass instrument further. With the proper conditions, implementations and few additions we can create a working model of a trumpet or trombone.

4.1 Results

As we can see, the shapes have a major impact on the output of the tube. Comparing the cone-like and the brass bell we can see that it has an impact on the number of reflections of the wave, with the brass bell shape multiplying them. We can also see some odd behaviours with both ends closed with the brass bell, that are not like any other shape.

Comparing the brass bells with different bore lengths, we can notice a funny, almost 'step-like', behaviour of the wave, where the length of the bore is so big, that the forward wave doesn't manage to merge with the backward wave, therefore creating a 'step-like' output.

The round shape and the round one with a long bore added on the right tend to create particular outputs. With both ends closed the wave reinforcement is so big, that the scale of the displacement had to be changed to 10^{11}.

With the half-triangular shape, we can see the output behaving step-like, with a strong similarity to a megaphone output. A very interesting thing is happening in Figure 3.45, where we can see a small displacement from the beginning, at around 0.04 s and around 0.85 s. We can also see the magnitude of the amplification of the output data in Figure 3.46 with the displacement reaching the 10^{29} magnitude.

There is not much difference between the size of the triangular shape, with the exclusion of Figure 3.58, where the output shape is quite odd and more round-shape-like. As for the half-triangular and differently sized triangular shapes, there aren't any major differences.

In the round with a long bore on the right side shape an interference at the interface between the circle and a the bore shapes can be observed in the output. We can also see the step-like behaviour which is characteristic of the circle shape.

4.2 Further Developments

Based on the works of [21] and [20] we can further develop the code by adding a mouthpiece part, in which the sound is going to be created, and the mouthpiece sound will be passed to the tube model. Furthermore, we can develop a graphic interface, with a proper wave visualisation and output visualisation in real time. Using a proper mouthpiece model, as developed in [11, 13, 16, 17, 20, 21] we can easily create a whole working model of a trumpet that, with the proper shape, can be implemented into any brass instrument.

We can also further develop the gridding and the delay lines in the model, so that the sound generation would be more flexible.

Additionally, after creating the mouthpiece model, the reflection between the mouthpiece and the lips could be studied more. According to [23] the studies show that the lips have a reflection magnitude of less than one, absorbing up to 6% of the incident amplitude at low frequencies. But as the authors have noticed, this matter should be studied deeper.

Furthermore, the reflection parameters between the instrument and the air around the instrument ought to be studied further, as to develop a more suitable parameter that reflects real life situations. Since this parameter might be dynamic according to the conditions around, this should be taken into account.

Same goes for the parameters of the conditions around the instrument, since they might have a major impact on the sound that is heard. According to [24]:

"[...] musician, instrument and room form a closed feedback loop that continuously shapes the generated sound."

After the authors studied a set of 11 trumpet players that were recorded while performing several pieces with multiple auralized versions of real spaces, the results indicate a moderate effect on temporal and energy room acoustic parameters on performance level and timbre. Additionally, in an environment with considerably stronger energy, players tended to decrease the overall tempo of the performance. The results revealed that listeners are able to consistently perceive level and timbre variations induced by stage acoustic conditions.

List of Figures

List of Tables

Code snippets

3 The Code

Bibliography

[1] S. Bilbao. *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*. Chichester: John Wiley & Sons Ltd, 2009. 1, 7, 8, 23, 29, 31,

[2] Acoustical Society of America. *ANSI/ASA S1.13-2020: Measurement Of Sound Pressure Levels In Air*. Melville, New York, 2020. 3

[3] Neville H Fletcher and Thomas D Rossing. *The physics of musical instruments*. Springer Science & Business Media, 2012. 3, 4

[4] www.orsymphony.org. Brass family of instruments: What instruments are in the brass family? Technical report, [Accessed 01 12 2023]. 4

[5] P. Y. Martin. A brief history of the trumpet: The origins and evolution. Technical report, https://hellomusictheory.com/learn/history-of-the-trumpet/, [Accessed December 2023]. 4

[6] Bill Addis. Physical models: Their historical and current use in civil and building engineering design. *Berlin: Wilhelm Ernst & Sohn*, 2021. 4

[7] Michael S. Bruno. *Physical Models*. Springer International Publishing, 2019. 4

[8] Jeffrey Schank and Charles Twardy. *Mathematical Models*. The Cambridge History of Science. Cambridge University Press, 2009. 4

[9] D Vicinanza. Astra project on the grid. Technical report, https://web.archive.org/web/20140224034559/http://www.astraproject.org /index.html, 2007. 5

[10] M. L. E. &. J. L. Rocamora. Wind instruments synthesis toolbox for generation of music audio signals with labeled partials. *In Proceedings of 2009 Brazilian Symposium on Computer Music, no. 2, pp. 2-4*, 2009. 5

[11] C. V. a. X. Rodet. Trumpet and trumpet player: Model and simulation in a musical context. *ICMC'01*, 2001. 5, 74

[12] M. Campbell. Brass instruments as we know them today. *Acta Acustica united with Acustica, vol. 90, no. 4, pp. 600– 610*, 2004. 5

[13] J. Cullen. *A study of brass instrument acoustics using an artificial reed mechanism, laser doppler anemometry and other techniques*. PhD thesis, The University of Edinburgh, 2000. 5, 74

[14] Julius Orion Smith III. A basic introduction to digital waveguide synthesis (for the technically inclined). *Center for Computer Research in Music and Acoustics (CCRMA) Department of Music, Stanford University, Stanford, California 94305 USA*, 2006. 5, 6

[15] P. R Cook. Real sound synthesis for interactive applications. *CRC Press*, 2002. 5

[16] X. R. a. C. Vergez. Nonlinear dynamics in physical models: From basic models to true musical-instrument. *Computer Music Journal, vol. 23, no. 3, pp. 35-49*, 1997. 7, 74

[17] M. a. S. Seiji Adachi. Trumpet sound simulation using a two-dimensional lip vibration model. *The Journal of the Acoustical Society of America, vol. 99, no. 2, pp. 1200-1209*, 1996. 8, 12, 13, 19, 74,

[18] J. Resch. *Physical Modelling and the Associated acoustic Behaviour of Trumpets and Trombones.* PhD thesis, University of Waterloo, 2019. 9, 10, 11,

[19] https://web.ece.ucsb.edu/ yoga/courses/Signals.html; [last accessed on november 2023]. 15

[20] M. Ducceschi S. Willemsen, S. Bilbao and S. Serafin. A physical model of the trombone using dynamic grids for finite-difference schemes. *in Proceedings of the 24th International Conference on Digital Audio Effects (DAFx20in21)*, 2021. 15, 16, 17, 74,

[21] S. B. J. P. T. W. Reginald Langford Harrison. An environment for physical modeling of articulated brass instruments. *Computer Music Journal, vol. Winter, pp. 80-95*, 2015. 18, 74,

[22] R. K. Friedrichs Courant and H. Lewy. On the partial differential equations of mathematical physics. *IBM Journal of Research and Development 11(2):215 - 234*, 1967. 19

[23] J. A. Kempa and R. A. Smithb. Measuring the effect of the reflection of sound from the lips in brass musical instruments. *Proceedings of the Acoustics 2012 Nantes Conference*, 2012. 74

[24] Sebastia V. AMENGUAL GARI; Malte KOB; Tapio LOKKI. Analysis of trumpet performance adjustments due to room acoustics. *Proceedings of the International Symposium on Room Acoustics, Amsterdam, Netherlands*, 2019. 74

Index