TTT4180 Technical Acoustics - Assignment 7

Nicholas Bresina, Department of Electronic Systems, NTNU Trondheim, Norway
nicholdb@stud.ntnu.no

Introduction

This assignment in the Technical Acoustics course (TTT4180) revolves around the transmission-line matrix (TLM) simulation technique used for acoustic waves and studying a variety of acoustic wave propagation scenarios by simulation.

The report is split into two parts, where the first part contains a description of the methods used to implement a TLM simulation in Python. After that, some results generated with said implementation are presented and discussed.

The second part delves deeper into analysis of wave propagation, specifically the sound pressure level with respect to distance, surface reflections, and the effect of a noise screen. This part uses a provided GUI tool called TLMfig that implements TLM in Matlab.

Lastly, the report gives a brief outlook with a discussion of difficulties encountered during this assignment with some additional final conclusions.

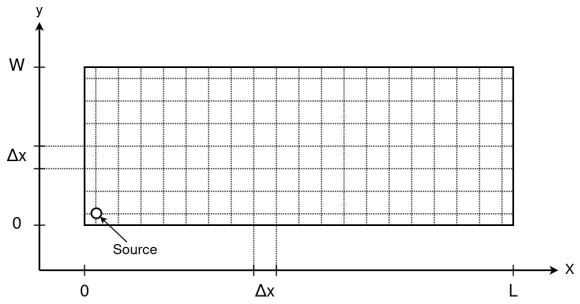


Figure 1: Example for TLM mesh placed on a pipe with dimensions L×W

1 Wave Propagation in a 2D Pipe

To analyze the propagation of a wave in a pipe, a 2D TLM implementation was needed. Python was chosen, as it provides all of the necessary features in libraries such as Numpy [1] for arrays and linear algebra and Matplotlib [5] for plotting the results.

1.1 Methods

1.1.1 TLM

The implementation is strongly based on the article provided with the assignment, that describes TLM [4]. Nevertheless the most important parts will be described in this section.

The main concept is to replace the continuous space with a grid of so-called nodes, as shown in Fig. 1.

Each node can be represented as a pipe system with four branches. And the nodes will be indexed with i, j referencing their position on the grid. The branches are modelled as pipes for the propagation of plane waves. Scattering or outgoing waves are denoted with ${}_kS^n_{i,j}$ and incident waves with ${}_kI^n_{i,j}$ respectively. k is the iteration count and n is the branch index as shown in Fig. 2.

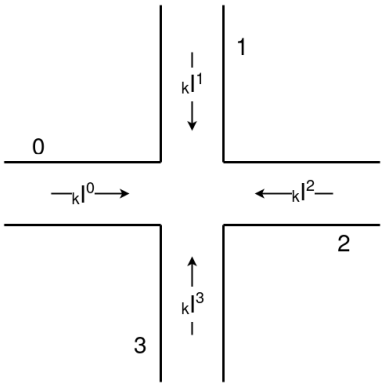


Figure 2: TLM node showing incident and outgoing waves as well as branch indexing

Due to the impedance discontinuity at the node the incident waves of iteration k will be turned into the scattering waves of iteration $k + 1$.

Which can be written as

$$
\begin{aligned}
{}_{k+1}S^0_{i,j} &= \frac{1}{2}\left[-{}_kI^0_{i,j} + {}_kI^1_{i,j} + {}_kI^2_{i,j} + {}_kI^3_{i,j}\right] \\
{}_{k+1}S^1_{i,j} &= \frac{1}{2}\left[{}_kI^0_{i,j} - {}_kI^1_{i,j} + {}_kI^2_{i,j} + {}_kI^3_{i,j}\right] \\
{}_{k+1}S^2_{i,j} &= \frac{1}{2}\left[{}_kI^0_{i,j} + {}_kI^1_{i,j} - {}_kI^2_{i,j} + {}_kI^3_{i,j}\right] \\
{}_{k+1}S^3_{i,j} &= \frac{1}{2}\left[{}_kI^0_{i,j} + {}_kI^1_{i,j} + {}_kI^2_{i,j} - {}_kI^3_{i,j}\right]
\end{aligned}
\tag{1}
$$

This can then be rewritten using the *scattering matrix* as follows

$$
\begin{bmatrix} {}_{k+1}S^0_{i,j} \\ {}_{k+1}S^1_{i,j} \\ {}_{k+1}S^2_{i,j} \\ {}_{k+1}S^3_{i,j} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} {}_kI^0_{i,j} \\ {}_kI^1_{i,j} \\ {}_kI^2_{i,j} \\ {}_kI^3_{i,j} \end{bmatrix} \tag{2}
$$

The factor $\frac{1}{2}$ is due to the fact that the energy is split evenly to four branches. Resulting in a factor of $\frac{1}{4}$ with respect to the energy. Due to $W \propto p^2$ we can establish a factor of $\frac{1}{2}$ for the pressure magnitude.

The incident waves of the next iteration are related to the scattering waves generated by the incident waves of the current iteration. The scattering waves propagate to the branches of the neighboring nodes, which is shown in Fig. 3.

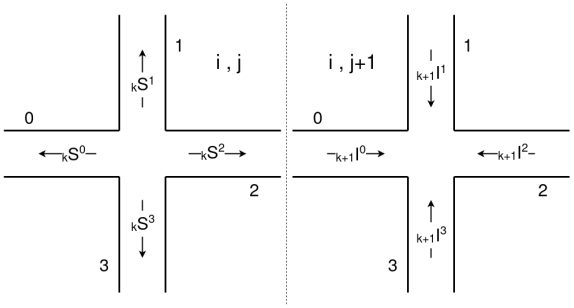


Figure 3: TLM nodes depicting the propagation of the scattering wave of branch 2 to the incident wave of branch 0 for the next iteration

Mathematically this can be written with the following equations

$$
\begin{aligned}
{}_{k+1}I^0_{i,j} &= {}_k\, S^2_{i,j-1} \\
{}_{k+1}I^1_{i,j} &= {}_k\, S^3_{i,j-1} \\
{}_{k+1}I^2_{i,j} &= {}_k\, S^0_{i,j+1} \\
{}_{k+1}I^3_{i,j} &= {}_k\, S^1_{i,j+1}
\end{aligned}
\tag{3}
$$

These relations only count for all the nodes, that are not part of any of the boundaries. At the boundary the next iteration's incident wave is given by the branch's scattering wave and the boundary's reflection coefficient R_m.

$$
{}_{k+1}I^n_{i,j} = R_m \cdot {}_k\, S^n_{i,j} \tag{4}
$$

The pressure can be calculated by the superposition of all the incident waves in the node, as given with

$$
{}_kp_{i,j} = \sum_{i=0}^{3} {}_kI^n_{i,j} \tag{5}
$$

1.1.2 Python Implementation

In the Python code the nodes are represented as an array. Where the size of the array is given by the physical dimensions L×W and the requirement to Δx, shown in Fig. 1, which is

$$
\Delta x \leq \frac{\lambda_{max}}{10} \tag{6}
$$

Where λ_{max} is the wavelength of the maximum frequency that should be possible to represented in the model, which is given by

$$
\lambda_{max} = \frac{c}{f_{max}} \tag{7}
$$

The size of the arrays $M \times N$ is then calculated by

$$
\begin{aligned}
M &= ceil\left(\frac{\mathrm{L}}{\Delta x}\right) \\
N &= ceil\left(\frac{\mathrm{W}}{\Delta x}\right)
\end{aligned}
\tag{8}
$$

Because a wave propagates the distance Δx in one iteration of the simulation, this also gives the timestep with

$$
\Delta t = \frac{\Delta x}{c} \tag{9}
$$

The easiest way to represent the different branches and differentiate between incident and scattering wave is by using a multidimensional array. The first two dimensions are the spatial integer dimensions $M \times N$. The third dimension is used to stack one grid for each branch waves, as shown in Fig. 4. So the resulting array has the shape $(M, N, 4)$. An additional layer is used to store source amplitudes on the grid, which are written as ${}_kG_{i,j}$.

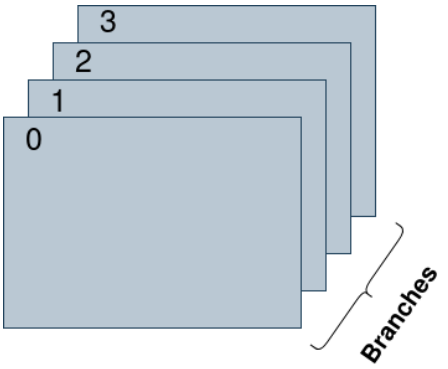


Figure 4: Depiction of the layers used in the implementation

To have efficient computation two of these multi-dimensional arrays are being used, where one corresponds to the current step and the other to the next iteration. With that, the values can easily be updated without generating temporary objects. The full shape of the array is then given with $(M, N, 9, 2)$.

Eq. 2 has to be rewritten to allow incorporating the mentioned source layer.

$$\begin{bmatrix} {}_{k+1}S^0_{i,j} \\ {}_{k+1}S^1_{i,j} \\ {}_{k+1}S^2_{i,j} \\ {}_{k+1}S^3_{i,j} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} {}_kI^0_{i,j} \\ {}_kI^1_{i,j} \\ {}_kI^2_{i,j} \\ {}_kI^3_{i,j} \\ {}_{k+1}G_{i,j} \end{bmatrix} \tag{10}$$

Which will add the sources' amplitudes to the corresponding scattering wave values.

Updating the incident waves can be done just as described in the TLM chapter.

To update the nodes' values for one iteration the following steps are needed:

Algorithm 1 Update steps for one iteration

```
procedure UPDATE_TLM
    for all sources do
        update source amplitude
    for m in M do
        for n in N do
            compute current scattering wave values
    for m in M do
        for n in N do
            update incident waves of next iteration
    update iteration step count
```

1.2 Calculations

The pipe dimensions that are used within this assignment are L = 2m and W = 0.2m. And the maximum considered frequency is 2kHz.

Given $c = 343\text{m/s}$ and using Eqs. (7), (6), and (9) we receive the following values,

$$\begin{aligned} \lambda_{max} &= 0.1715\text{m} \\ \Delta x &= \frac{\lambda_{max}}{10} = 0.01715\text{m} \\ \Delta t &= 50\mu\text{s} \end{aligned} \tag{11}$$

1.2.1 Modes

With the assumption that all boundaries are rigid walls we can calculate the modal frequencies [2] with

$$f_{lm} = \frac{c}{2}\sqrt{\left(\frac{l}{L}\right)^2 + \left(\frac{m}{W}\right)^2} \tag{12}$$

Resulting in the following list of modes:

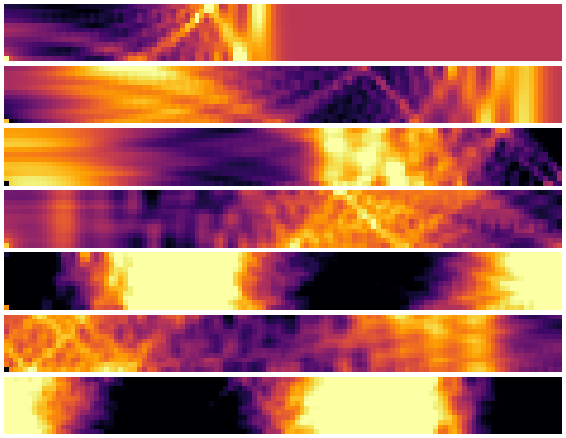
(l,m)	f_{lm} [Hz]
(1,0)	85.75
(2,0)	171.50
(3,0)	257.25
(4,0)	343.00
(5,0)	428.75
(0,1)	857.50
(1,1)	861.78

From this we can conlude that there should not be a mode in y-direction below 857.5Hz and only plane waves should be present.

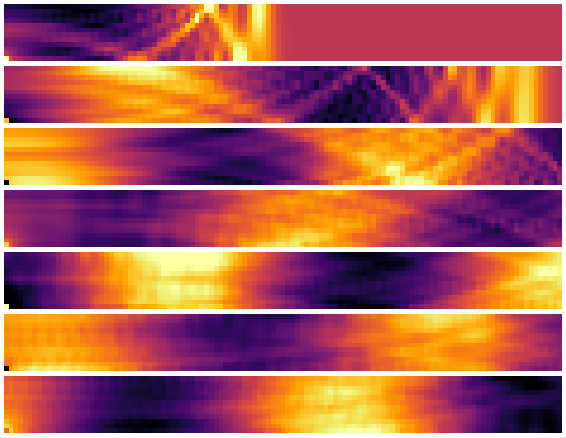
All the modes where $m = 0$ can be considered the resonance frequencies of the closed-closed pipe [3]. Which are given by

$$kL = \pi l \tag{13}$$

1.3 Results



(a) Right boundary with R = 1



(b) Right boundary with R = 0.1

Figure 5: Evolution of the wave in the pipe over time

From Fig. 5 we can observe how the wave propagates when a sine wave source is placed in the bottom left corner with 500Hz. The first image (Fig. 5a) shows the case where the right boundary is a rigid wall. In

that case we can observe that a mode is established with standing nodes. The wave's minima and maxima flip according to the source's amplitude.

In the case that the right boundary has a reflection coefficient of 0.1 (Fig. 5b the behavior is slightly different. Instead of standing nodes, they now move toward the pipe end corresponding to source's excitation.

2 TLMfig

2.1 Propagation in free space

2.1.1 Description

To analyze the behavior of the amplitude of the wave with respect to distance a source and microphones are placed along one line, as shown in Fig. 6.

Figure 6: Simplified example for source and microphone setup to analyze propagation in free space

In the actual simulation the source was placed in the coordinates $(1, 50)$ and the 30 microphones where placed at $(3n + 2, 50)$, where $n \in [0, 30[$. The source was configured to produce a sine wave with an amplitude of one and a number of cells per wavelength (NCPW) of 15.

2.1.2 Results

The signal arrivals at the microphones can be seen in Fig. 7. The microphones with a higher index are further away. This is also visible through the delayed arrivals at the start.

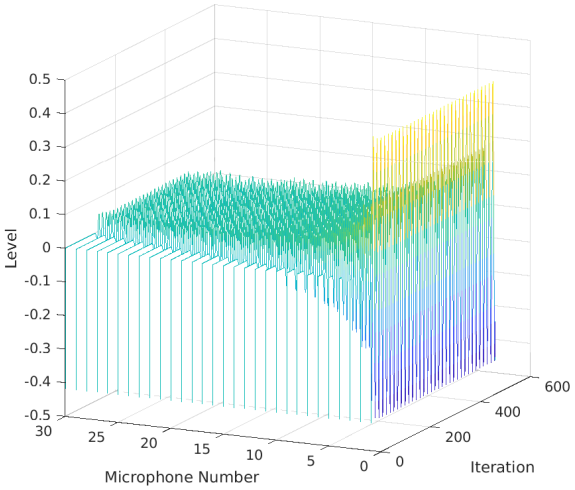


Figure 7: 3D graph of the received signals at the microphones

With these signals we can calculate the sound pressure levels (SPLs) and plot them against the distance, which is depicted in Fig. 8. The calculation was performed by averaging the peak amplitudes and then referencing against $p_0 = 20\mu$Pa.

$$SPL = 20 \log_{10} \left(\frac{p_{peak,mean}}{\sqrt{2} \cdot p_0} \right) \tag{14}$$

In the plot shown in Fig. 8 we can roughly see the expected decay over distance.

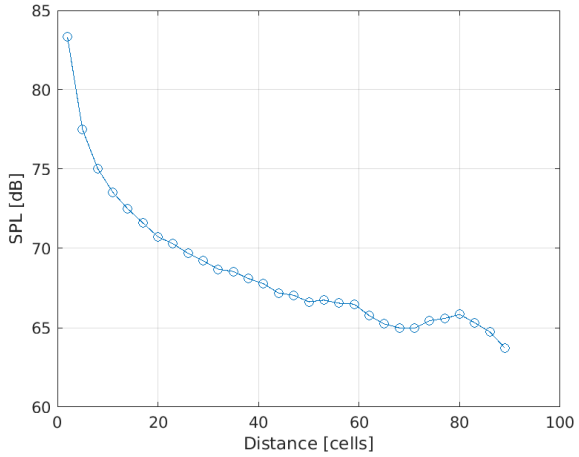


Figure 8: The calculated SPLs for the microphone array against distance

2.2 Reflection from surface

2.2.1 Description

This next simulation is intended to study the behavior of the wave reflected from surfaces with different reflection coefficient. For this a surface was placed at the bottom in TLMfig. The source was placed at $(50, 59)$ and the 20 microphones were placed at around the same height with the following coordinates $(4n + 10, 60)$, where $n \in [0, 20[$. The source was set to generate a gaussian pulse with an amplitude of one.

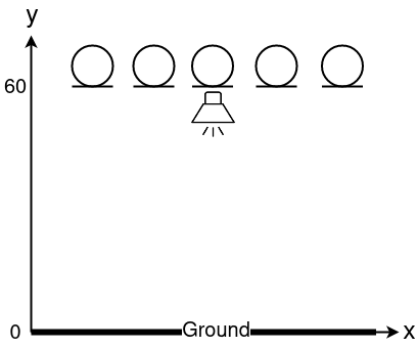
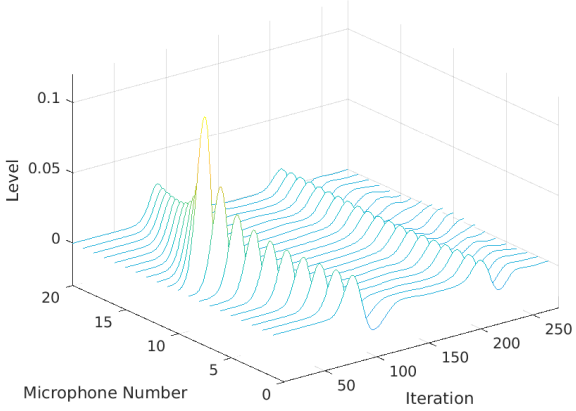


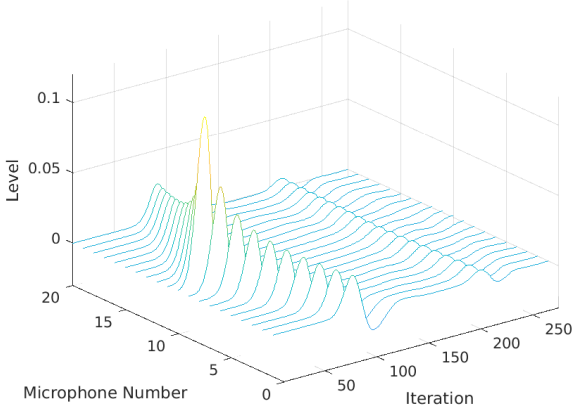
Figure 9: A depiction of the source and microphone arrangement to investigate reflection of surfaces

2.2.2 Results

The signal arrivals are shown in the two plots in Fig. 10. They show the two received pulses for different reflection coefficients, where the first pulse is the direct and the second the reflected one.



(a) Reflection coefficient = 1



(b) Reflection coefficient = 0.5

Figure 10: Signals received at the microphone array with different reflection coefficients

Using this data we can plot the loss on the reflected pulse using the direct pulse as a reference.

$$\text{Loss} = -20\log_{10}\left(\frac{p_{peak,reflected}}{p_{peak,direct}}\right) \qquad (15)$$

This is plotted for all the microphones in the array and displayed in Fig. 11.

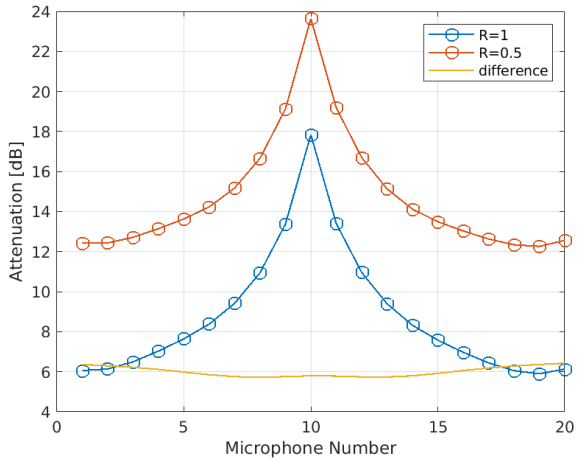


Figure 11: The loss from direct to reflected pulse for the two reflection coefficients

It can be observed that halving the reflection coefficient of the surface results in a loss of $\approx$ 6dB.

2.3 Effect of noise screen

2.3.1 Description

The last simulation is used to analyze the effect of a noise screen. In this case the noise screen was simulated as a perfectly rigid wall and the ground having a reflection coefficient of 0.95. The source was setup to produce a sine wave with a NCPW of 15 and 45 respectively, as well as just noise. Using different NCPW values provides insight in frequency behavior, where a low NCPW corresponds to higher frequencies. The simulation arrangement can be seen in Fig. 12.

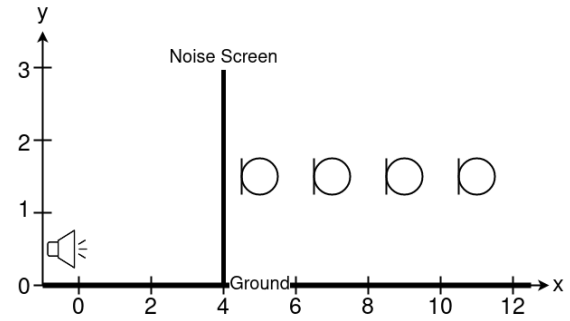


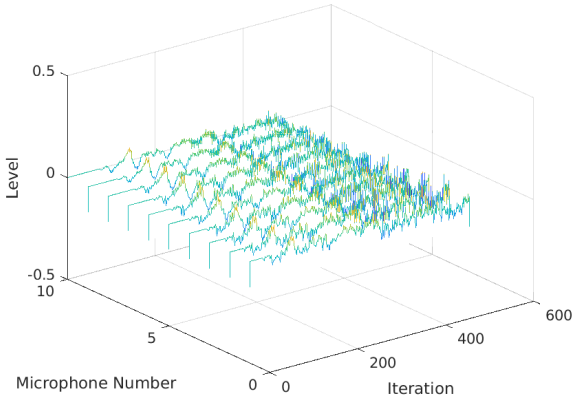
Figure 12: A simplified diagram of source and microphone arrangement for the noise screen simulation

The exact setup is given by the ground being at height of 10 cells, the source at $(10, 12)$, the screen expands from $(18, 10)$ to $(18, 22)$, and the ten microphones have the coordinates $(4n + 20, 16)$ with $n \in [0, 10[$. To analyze the effect of the screen the simulation is run once with and once without screen.

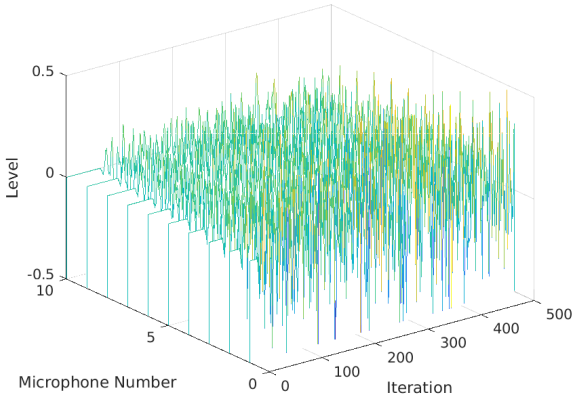
2.3.2 Results

In Fig. 13 an example is given where the provided source type was noise. The data from these two simulations per source can be used to calculate the *insertion loss*. The loss that results from adding the screen is given by

$$\text{Insertion Loss} = 20\log_{10}\left(\frac{p_{rms,direct}}{p_{rms,screen}}\right) \tag{16}$$



(a) with screen



(b) without screen

Figure 13: Example of signals received at the microphone array with and without noise screen using a noise source

The calculation for the insertion loss was performed for all the variation of sound sources and can be seen in Fig. 14. One can observe that the noise screen generally provides a sound reduction. It is the least effective for low frequency sine waves (NCPW = 45). And the effectiveness seems to decrease over distance.

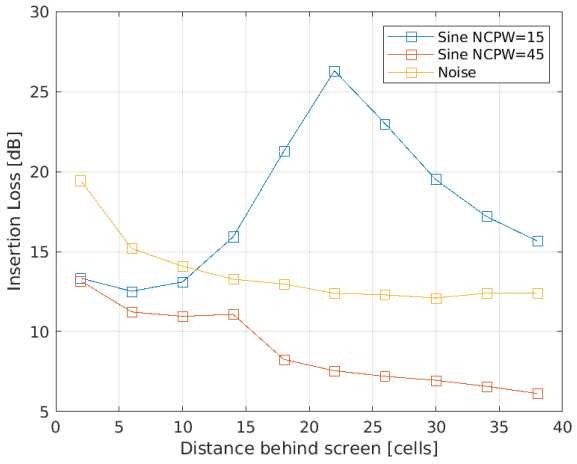
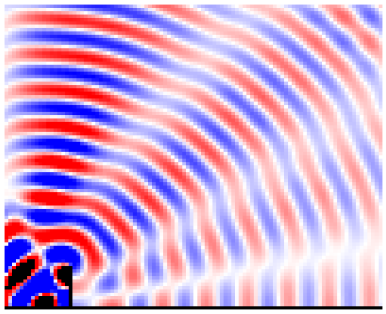
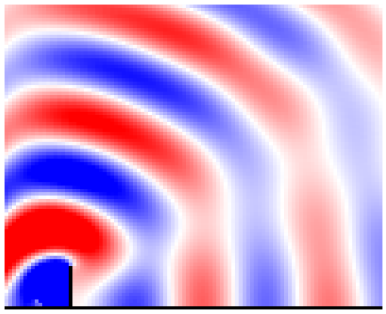


Figure 14: Insertion loss for varying source types resulting from introducing a noise screen

In the case of NCPW = 15 however, we can clearly see that, a strong attenuation takes place at a certain distance after the screen. This strong attenuation can be explained with an interference pattern that is generated after the screen, which is displayed in Fig. 15. We can see that with high frequency (NCPW = 15) there's a beam of a local minimum that will cross the height of the microphone array, resulting in the strong attenuation that's seen in Fig. 14.



(a) NCPW = 15



(b) NCPW = 45

Figure 15: Interference pattern after the noise screen for the two NCPW settings

3 Discussion

3.1 Python

3.1.1 Implementation

The current implementation still seems somewhat faulty, when comparing the results with TLMfig. Debugging faults in the implementation took up more time than expected. Due to that and not being able to completely fix all issues with the code, the frequency response has not been implemeted yet. And is therefore missing in the report.

3.1.2 Matplotlib

Using this library to plot the animation had some pitfalls. The library uses a drawing buffer, which just fills over time. Without manually clearing it before updating the plot, it will slow the animation by a large amount. Once that was introduced, a significant improvement could be observed.

3.2 TLMfig

While using this GUI some shortcomings were discovered. Given that this might only be the case for the author's setup, which consisted of a Linux machine without a dedicated GPU.

3.2.1 Placing Objects

Firstly, there are no coordinates for the cursor while drawing. If this was the case placing objects in the plane would be simplified a lot. A workaround is drawing lines as helpers and erasing them to all but one pixel at the edge of the plane. Nevertheless, this is only a minor issue.

3.2.2 Exporting received signals

Secondly, exporting the received data is not straightforward and this has some overlap with the third observation. Nevertheless possibilites could be to export the data which is plotted into the Matlab workspace or generate a .mat file. There's the option to export the configuration and grabbing the data from there, but that's not exactly a logical step.

3.2.3 Reusing stored configurations

With talking about exporting the configuration we come to the third observation. So assuming a configuration was saved and we reopen TLMfig. Then try to open the stored configuration, it will open in a new TLMfig window. This new window is solely a Matlab figure without the backend logic. Therefore it cannot be used to re-run the stored configuration, making it impossible to reproduce results without re-drawing the setup.

3.2.4 GUI usage

There are some additional unexpected behaviors. E.g. the button label for placing mics updates to 1 when a number is changed in the field, then *not* validated with the enter key, followed by clicking the button to place microphones. It doesn't revert back and stays at the same number, which is 1.

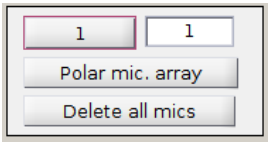


Figure 16: TLMfig button label after bug

Another unusual behavior is observed when using the input box to draw lines. Sometimes one has to actually validate an input with the enter key and other times it's enough to just change the value in the box and click somewhere else in the GUI and a line is drawn. This can lead to drawing way too many lines that were not intended if the user is not aware of this.

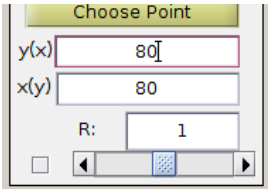


Figure 17: Line drawing input box

3.2.5 Conclusion

All the observed issues are only hindering usability. TLMfig otherwise seems to work as expected and the GUI is largely helpful for simulating wave propagation and analzying propagation behavior in various settings.

4 References

References

[1] The NumPy Community. Numpy v1.21 manual.

[2] Guillaume Dutilleux. Cavities and waveguides. 2020.

[3] Guillaume Dutilleux. Pipes and resonators. 2020.

[4] Y. Kagawa, T. Tsuchiya, B. Fujii, and K. Fujioka. Discrete huygens' model approach to sound wave propagation. *Journal of Sound and Vibration*, 218(3):419–444, 1998.

[5] Matplotlib Development Team. Matplotlib - visualization with python.