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 provides a brief description of methods used for an own implementation of TLM simulation in Python. After, some results are presented and discussed, which were generated from said implementation.

The second part contains further analyisis of sound pressure with respect to distance, surface reflections, and the effect of a noise screen on the sound pressure level. Instead of using the own implementation, this part uses a given Matlab implementation called TLMfig.

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

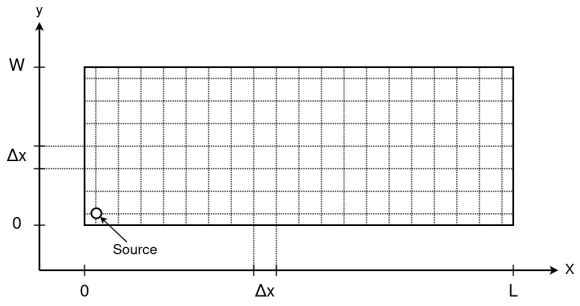


Figure 1: Example for TLM mesh placed on a pipe with dimensions L×W

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 [2] 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 [5]. Nevertheless the most significant parts will be described in this section.

The main idea 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. Each branch will contain a superposition of an incident and scattering pressure waves. These are denoted as ${}_kI^n_{i,j}$ and ${}_kS^n_{i,j}$ respectively, where k is the iteration count and n is the branch index as shown in Fig. 2.

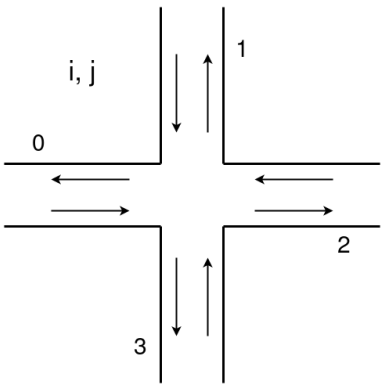


Figure 2: TLM node showing incident and outgoing waves as well as branch indexing

The incident waves of iteration k will be turned into the scattering waves of iteration $k + 1$. This can be written as

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

Which can 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 we're considering the propagation of pressure waves. An incident wave will evenly distribute the energy over the four branches, resulting in a factor of $\frac{1}{4}$ with respect to the energy. Given

$$
W_p = \frac{p^2_{rms}}{4K}, \tag{3}
$$

which shows the square relationship between energy and pressure. We can establish that a factor $\frac{1}{4}$ in energy to be a factor of $\frac{1}{2}$ with respect to pressure.

The incident waves of the next iteration are the scattering waves of the respective branches of the neighboring branches, which is shown in Fig. 3.

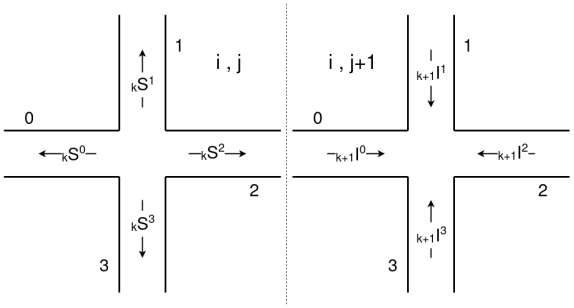


Figure 3: TLM nodes depicting the propagation of the scattering wave of branch 2 to the incident wave of branch 0 for the next iteration

Mathematically this can be written with the following equations

$$
\begin{aligned}
{}_{k+1}I^0_{i,j} &=_k\, S^2_{i,j-1}\\
{}_{k+1}I^1_{i,j} &=_k\, S^3_{i,j-1}\\
{}_{k+1}I^2_{i,j} &=_k\, S^0_{i,j+1}\\
{}_{k+1}I^3_{i,j} &=_k\, S^1_{i,j+1}
\end{aligned}
\tag{4}
$$

Now 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 current scattering wave and the boundary's reflection coefficient R_m.

$$
{}_{k+1}I^n_{i,j} = R_m \cdot_k\, S^n_{i,j} \tag{5}
$$

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

$${}_kp_{i,j} = \sum_{i=0}^{3} {}_kI^n_{i,j} - \sum_{i=0}^{3} {}_kS^n_{i,j} \tag{6}$$

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

$$
\Delta x \leq \frac{\lambda_{max}}{10} \tag{7}
$$

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

$$
\lambda_{max} = \frac{c}{f_{max}} \tag{8}
$$

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

$$
\begin{aligned}
M &= ceil\left(\frac{\mathrm{L}}{\Delta x}\right)\\
N &= ceil\left(\frac{\mathrm{W}}{\Delta x}\right)
\end{aligned}
\tag{9}
$$

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

$$
\Delta t = \frac{\Delta x}{c} \tag{10}
$$

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 per branch for each incident and scattering waves, as shown in Fig. 4. An additional layer is added in the third dimension to represent source amplitudes placed on the grid, which are written as ${}_kG_{i,j}$. So the resulting array has the shape $(M, N, 9)$.

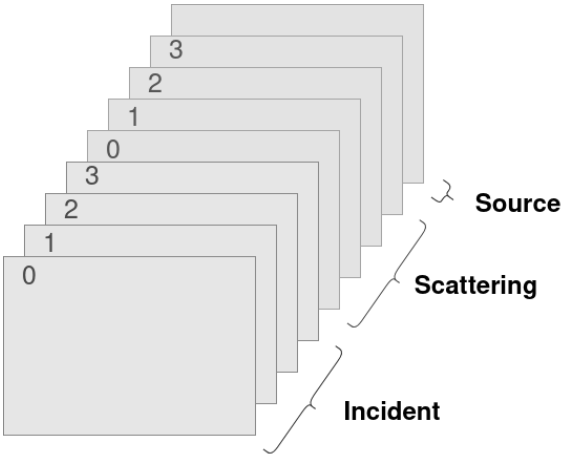


Figure 4: Depiction of the layers used in the implementation

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.

$$\begin{bmatrix} {}_{k+1}S^0_{i,j} \\ {}_{k+1}S^1_{i,j} \\ {}_{k+1}S^2_{i,j} \\ {}_{k+1}S^3_{i,j} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \text{-1} & 1 & 1 & 1 & 1 \\ 1 & \text{-1} & 1 & 1 & 1 \\ 1 & 1 & \text{-1} & 1 & 1 \\ 1 & 1 & 1 & \text{-1} & 1 \end{bmatrix} \begin{bmatrix} {}_kI^0_{i,j} \\ {}_kI^1_{i,j} \\ {}_kI^2_{i,j} \\ {}_kI^3_{i,j} \\ {}_{k+1}G_{i,j} \end{bmatrix} \tag{11}$$

Which will add the sources' amplitudes to the corresponding scattering waves.

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

Using a feature of NumPy that allows matrix multiplication for multidimensional arrays, it's possible to compute all scattering values using the *scattering matrix* for a whole $(M, 1)$ or $(1, N)$ vector at once. Updating the incident waves can also be achieved with one iteration in each the x-direction and the y-direction, updating the horizontal and vertical propagation respectively. This approach allows reducing the run time complexity to be $\mathcal{O}(2N)$ in the worst case.

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 i in M do                ▷ iterate over x-direction
        compute next scattering values
        compute horizontal incident propagation
    for i in N do                ▷ iterate over y-direction
        compute vertical incident propagation
    update iteration step count
```

And calculating the pressure values can be done by summing up the incident and scattering layers.

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 $c = 343$m/s and using Eqs. (8), (7), and (10) we receive the following values,

$$\begin{aligned} \lambda_{max} &= 0.1715\text{m} \\ \Delta x &= \frac{\lambda_{max}}{10} = 0.01715\text{m} \\ \Delta t &= 50\mu\text{s} \end{aligned} \tag{12}$$

1.2.1 Modes

With the assumption that all boundaries are rigid walls we can calculate the modal frequencies [3] with

$$f_{lm} = \frac{c}{2}\sqrt{\left(\frac{l}{L}\right)^2 + \left(\frac{m}{W}\right)^2} \tag{13}$$

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.

All the modes where $m = 0$ can be considered the resonance frequencies of the closed-closed pipe [4]. Which are given by

$$kL = \pi l \tag{14}$$

1.3 Results

1.4 Discussion

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. 5.

Figure 5: 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 31 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. 6. The microphones with a higher index are further away. This is also visible through the delayed arrivals at the start.

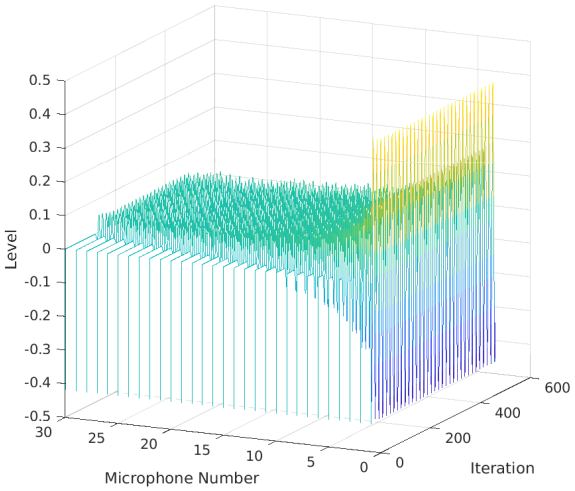


Figure 6: 3D graph of the received signals at the microphones

With these signals we can calculate the sound pressure levels (SPLs) and plot them against the distance, which is depicted in Fig. 7. The calculation was performed by averaging the peak amplitudes and then referencing against $p_0 = 20\mu$Pa.

$$SPL = 20 \log_{10} \left(\frac{p_{peak,mean}}{\sqrt{2} \cdot p_0} \right) \tag{15}$$

In the plot shown in Fig. 7 we can roughly see the expected decay over distance.

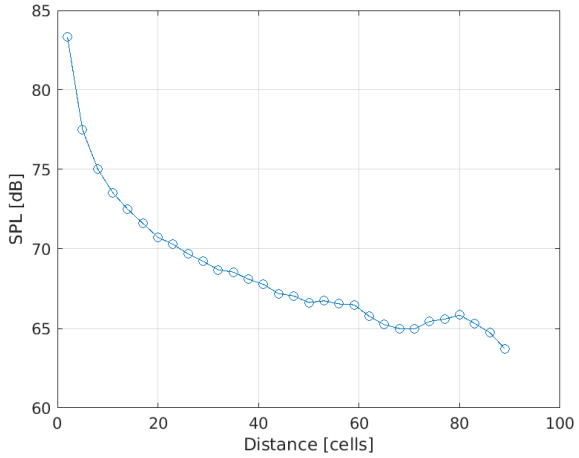


Figure 7: The calculated SPLs for the microphone array against distance

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 21 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.

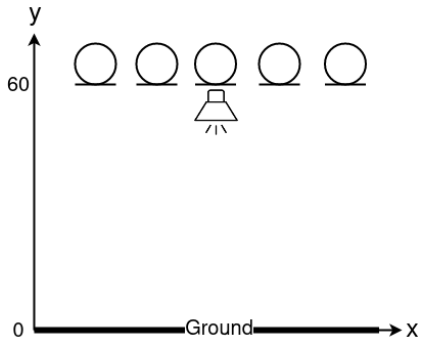
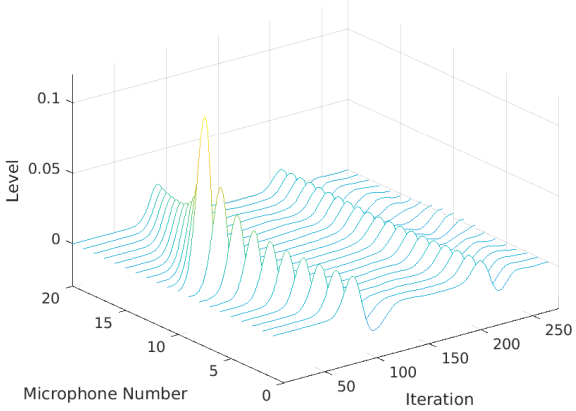


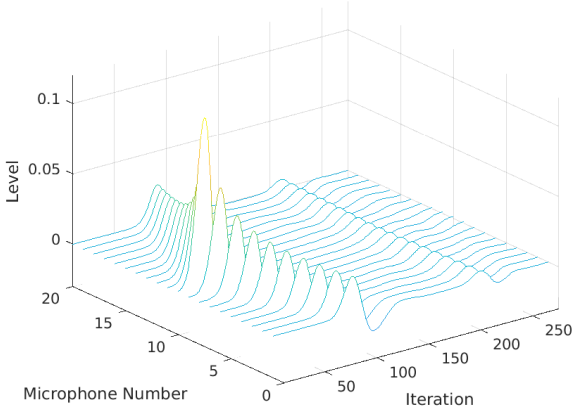
Figure 8: A depiction of the source and microphone arrangement to investigate reflection of surfaces

2.2.2 Results

The signal arrivals are depicted in the two plots in Fig. 9. 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 9: 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.

$$\text{Loss} = -20\log_{10}\left(\frac{p_{peak,reflected}}{p_{peak,direct}}\right) \tag{16}$$

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

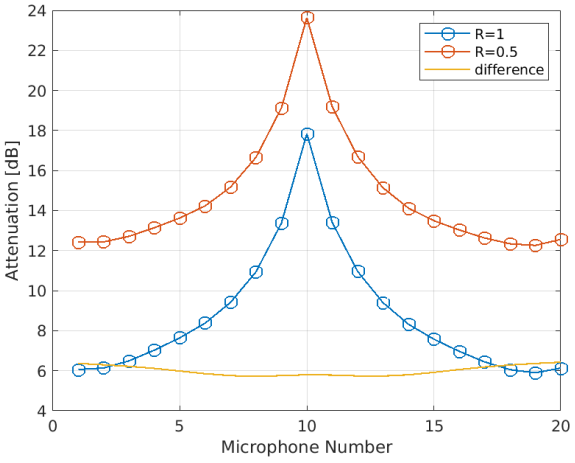


Figure 10: The loss from direct to reflected pulse for the two reflection coefficients

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

The last simulation is intended to analyze the effect of a noise screen. In this case the noise screen was simulated as a perfectly rigid wall. The simulation configuration can be seen in Fig. 10. The ground has a reflection coefficient of 0.95 and the source is set to noise with varying number of cells per wavelength to achieve low- and high-frequency noise.

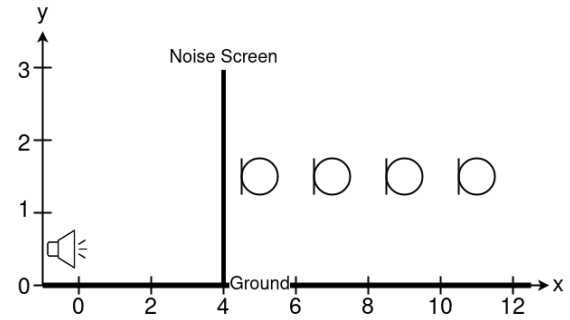


Figure 11: An exemplary setup of source and microphone for the noise screen simulation, with distances given in meters

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 eleven 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.

2.4 Discussion

3 Conclusion

3.1 Python

3.1.1 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, which is only a minor issue though. 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.

3.2.2 Exporting received signals

Secondly, exporting the received data is not straightforward and this has some overlap with the third problem. 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 behaviors that are unexpected. 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.

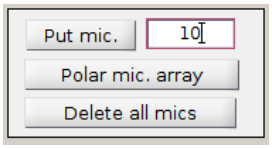


Figure 12: TLMfig button label before bug

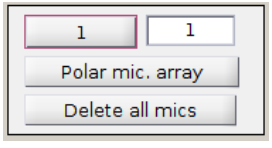


Figure 13: TLMfig button label after bug

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.

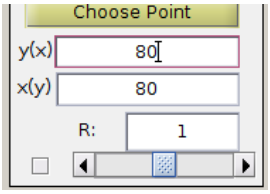


Figure 14: Line drawing input box

3.2.5 Final Words

Even with all the beforementioned issues while using TLMfig, the GUI is largely helpful for simulating wave propagation and analzying the behavior as reflections for example.

4 References

References

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

[2] Matplotlib development team. Matplotlib - visualization with python.

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

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

[5] 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.