Computational Physics Exercise 6: Numerical Solutions of the Maxwell Equations Using the Yee-Algorithm

Fynn Janssen 411556[1] and Tamilarasan Ketheeswaran 411069[2]

April 2023

Contents

1 Introduction **2**

2 Simulation Model and Method **2**

3 Results **7**

3.1 The Thin Glass Plate 7

3.2 The Thick Glass Plate 9

4 Discussion **11**

[1] fynn.janssen@rwth-aachen.de

[2] tamilarasan.ketheswaran@rwth-aachen.de

1 Introduction

In this exercise, we use the 1D-Yee algorithm to simulate the transmission and reflection of light at a glass plate. First, we investigate the system that contains a thin glass plate. In this instance, we check what happens if the Courant condition is not satisfied anymore. For the second task, we change the thin glass plate to one which covers the entire right plane. Here, we want to calculate the reflection coefficient of the glass. In both cases, the wave packet is created on the left side of the glass and both systems have fixed boundary conditions that absorb the EM-waves reflectionless.

The code is written in the programming language `python`, and the packages `numpy`[1] as well as `matplotlib` are used. Additionally, to obtain a faster code we used the libraries `njit, vectorize`, and `float64` from `numba`.

2 Simulation Model and Method

First, we describe the system we want to simulate.

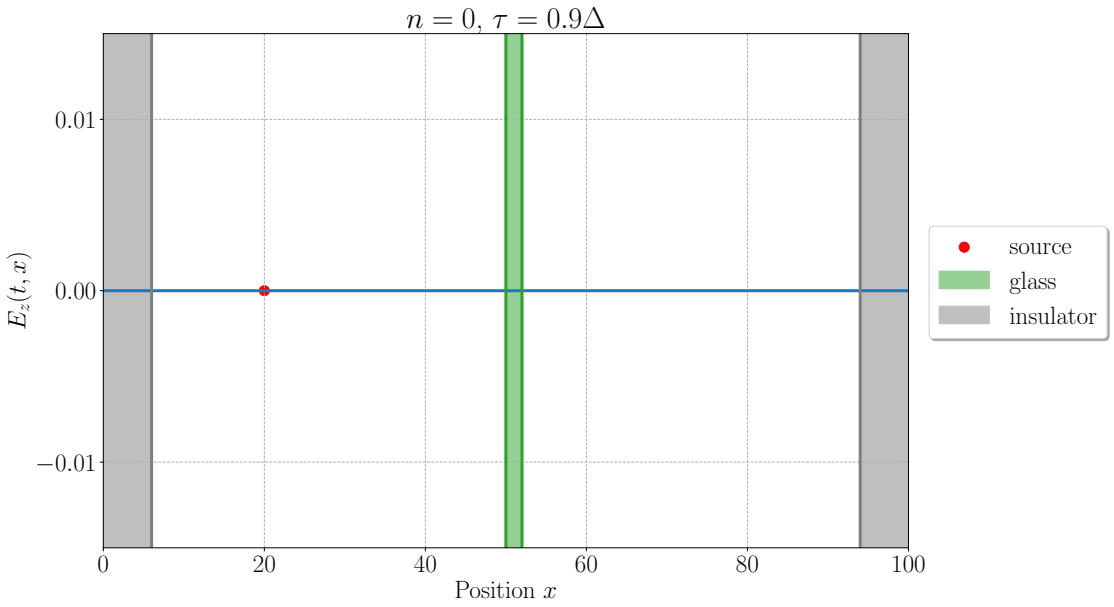

Figure 1: sketch of the overall system. The glass plate is depicted in green in the middle, and at the system's boundaries there are two insulators (grey). The source at x_s is represented by the red point.

In fig.(1), the system containing the thin glass plate is sketched. In principle, we have a source (in red) at x_s that creates a wave packet, according to

$$J_{source,z}(x,t) = \begin{cases} \sin(\omega t) \cdot e^{-((t-30)/10)^2} & \text{if } x = x_s \\ 0 & \text{if } x \neq x_s \end{cases}. \tag{1}$$

At the edges of the system, we have two insulators in grey, characterized by the electrical conductivity σ and the magnetic loss σ^*, which we introduce later more precisely.[2] Further, we have fixed boundary conditions

$$E_z(x = 0, t) = E_z(x = X, t) = 0, \tag{2}$$

[1] abbreviated as `np`

[2] note that the term "insulator" does not imply perfect absorption. In our simulation there will be reflections, small enough to call it an insulator

where X is the length of the system and, therefore, the right boundary. This implies that any wave reaching $x = 0$ or $x = X$ is reflectionless absorbed.

In the center of the system, we have a glass plate (in green), characterized by the electrical permittivity ϵ and magnetic permeability μ, has a different refraction index n_d than the space in white, which represents vacuum[3].

In general, such a system can be solely described by the Maxwell equations. Assuming, we have no free charges and the materials are linear, isotropic, nondispersive, and lossless, the Maxwell equations simplify into

$$\begin{aligned}
\frac{\partial \vec{H}(\vec{r},t)}{\partial t} &= \frac{1}{\mu}\left[-\vec{\nabla}\times\vec{E}(\vec{r},t) - \vec{M}(\vec{r},t)\right] \\
\frac{\partial \vec{E}(\vec{r},t)}{\partial t} &= \frac{1}{\epsilon}\left[\vec{\nabla}\times\vec{H}(\vec{r},t) - \vec{J}(\vec{r},t)\right] \\
\epsilon\vec{\nabla}\cdot\vec{E}(\vec{r},t) &= 0 \\
\mu\vec{\nabla}\cdot\vec{H}(\vec{r},t) &= 0
\end{aligned}$$

From electrodynamics, we know that for a wave traveling solely in the x-direction, the $\vec{E}$ and $\vec{H}$ fields are orthogonal to the direction of propagation and orthogonal to each other. We orient our coordinate system such that the $\vec{E}$ field is aligned along the z-axis, and the $\vec{H}$ field along the y-axis. From the system being symmetric in y, z-direction, the Maxwell equations simplify into

$$\frac{\partial H_y(x,t)}{\partial t} = \frac{1}{\mu(x)}\left[\frac{\partial E_z(x,t)}{\partial x} - M_y(x,t)\right] \tag{3}$$

$$\frac{\partial E_z(x,t)}{\partial t} = \frac{1}{\epsilon(x)}\left[\frac{\partial H_y(x,t)}{\partial x} - J_z(x,t)\right] \tag{4}$$

with

$$M_y(x,t) = \sigma(x)^* \cdot H_y(x,t) \quad \text{and} \quad J_z(x,t) = J_{source,z}(x,t) + \sigma(x)E_z(x,t).$$

Inserting $M_y(x,t)$ and $J_z(x,t)$ into eq.(4), the Maxwell equations we want to solve are

$$\frac{\partial H_y(x,t)}{\partial t} = \frac{1}{\mu(x)}\left[\frac{\partial E_z(x,t)}{\partial x} - \sigma(x)^* H_y(x,t)\right] \tag{5}$$

$$\frac{\partial E_z(x,t)}{\partial t} = \frac{1}{\epsilon(x)}\left[\frac{\partial H_y(x,t)}{\partial x} - J_{source,z}(x,t) - \sigma(x)E_z(x,t)\right]. \tag{6}$$

To implement the Yee-algorithm, a Yee grid along the x-axis in our system from left to right boundary must be defined. The Yee grid that is used is explained later more thoroughly. In this instance the E_z and H_y fields are discretized in time and position. The corresponding update rules for the respective fields are given by

$$\begin{aligned}
H_y\Big|_{l+1/2}^{n+1} &= A_{l+1/2}H_y\Big|_{l+1/2}^{n} + B_{l+1/2}\left[\frac{E_z\Big|_{l+1}^{n+1/2} - E_z\Big|_{l}^{n+1/2}}{\Delta}\right] \\
E_z\Big|_{l}^{n+1/2} &= C_l E_z\Big|_{l}^{n-1/2} + D_l\left[\frac{H_y\Big|_{l+1/2}^{n} - H_y\Big|_{l-1/2}^{n}}{\Delta} - J_{source}\Big|_{l}^{n}\right]
\end{aligned} \tag{7}$$

[3] $n_{vacuum} = 1$

with $H_y\Big|_{l+1/2}^{n+1}$, the H_y field at timestep $n+1$ at position $l+1/2$. $E_z\Big|_{l}^{n+1/2}$, the E_z field at timestep $n+1/2$ at position l.
$J_{source}\Big|_{l}^{n}$ is the source that creates the wavepacket at time n and position l.

A, B, C, D, are functions dependent on the $\epsilon, \mu, \sigma, \sigma^*$, defined as the following

$$A_{l+\frac{1}{2}} = \frac{1 - \frac{\sigma^*_{l+\frac{1}{2}}\tau}{2\mu_{l+\frac{1}{2}}}}{1 + \frac{\sigma^*_{l+\frac{1}{2}}\tau}{2\mu_{l+\frac{1}{2}}}} \qquad B_{l+\frac{1}{2}} = \frac{\frac{\tau}{\mu_{l+\frac{1}{2}}}}{1 + \frac{\sigma^*_{l+\frac{1}{2}}\tau}{2\mu_{l+\frac{1}{2}}}} \tag{8}$$

$$C_l = \frac{1 - \frac{\sigma_l\tau}{2\epsilon_l}}{1 + \frac{\sigma_l\tau}{2\epsilon_l}} \qquad D_l = \frac{\frac{\tau}{\epsilon_l}}{1 + \frac{\sigma_l\tau}{2\epsilon_l}}. \tag{9}$$

Δ is the spatial resolution we choose, defined as $\Delta = \frac{\lambda}{\text{number of grid points per } \lambda}$, with λ being the wavelength.

With all the necessary background information given, we want to discuss the implementation of this task in Python. Before that, we want to mention that this algorithm is not based on saving all the values of fields at each time, but rewriting the old values at an earlier time. According to the Yee algorithm, a box of size X with spatial resolution Δ is discretized in $L = \frac{X}{\Delta}$ equidistant steps. First, we initialize an array that goes from 0 to L in equal steps. This is done by

$$l = \texttt{np.arange(0,L+1)} = 0, ..., L.$$

The E field is calculated at each point in l, where the position is given by $x = l\Delta$. Therefore, we need to store $L+1$ elements in the E field array. We initialize this array using

$$\texttt{E} = \texttt{np.zeros(L+1)}.$$

On the other hand, the H field is calculated at each intermediate position between two electrical field points. Hence, the position at which the H components are calculated are given by half-integer steps $x = (l+1/2)\Delta$. In order to obtain only values for H inside the considered box $0 \leq x \leq X$, l goes from 0 to $L-1$, such that we have L components and not $L+1$. Otherwise, $l = L$ would be at the position $x = L\Delta + \frac{1}{2}\Delta > X$ and, thus, outside the box. This array can be initialized using

$$\texttt{H} = \texttt{np.zeros(L)}.$$

During the whole simulation the two arrays (`H` and `E`), are updated after each time step being passed. While the electrical field is updated at each time $t = (n+1/2)\tau$, the magnetic field is updated at $t = n\tau$. This separation of the time for the E and H is characteristic for the Yee algorithm.[4] The n is an index that is iterated with a `for-loop` defining the time that has elapsed.

As a small summary of the discretization and the Yee grid, we present the following visualization in fig.(2). The different discretizations for the two fields are necessary since we investigate a system of coupled differential equations. This implies that it is not possible to compute both fields at exactly the same time which explains the half-integer steps.

As already mentioned, to solve the differential equations we solve eq.(7) iteratively. During each step in the `for-loop` the arrays are updated. Keeping in mind that the source is only non-zero at x_s

[4] Also we do not save all the values of the E and H fields at all times as that would be memory-consuming and take too much time

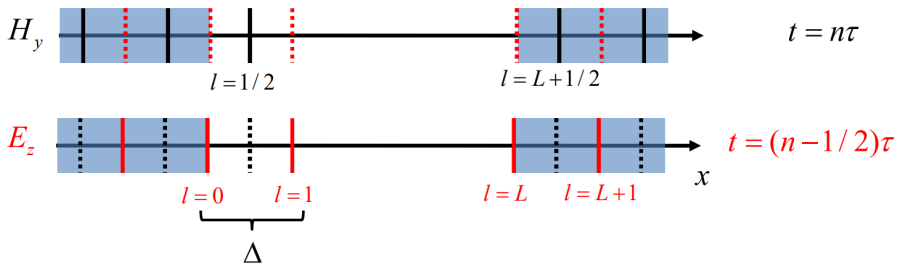

Figure 2: Visualization of the one dimensional Yee-grid for the `E` and `H` fields.
[Source: Computational Physics – Lecture 12: How to solve Maxwell's equations numerically? II, page 14]

which corresponds to the index $i_s = x_s/\Delta$, we introduce the Kronecker-δ_{l,i_s} and obtain the following update rules

$$\begin{aligned} H_{l+1/2}^{n+1} &= \frac{B_{l+1/2}}{\Delta}\left(E_{l+1}^{n+1/2} - E_l^{n+1/2}\right) + A_{l+1/2} H_{l+1/2}^n \\ E_l^{n+1/2} &= \frac{D_l}{\Delta}\left(H_{l+1/2}^n - H_{l-1/2}^n\right) + C_l E_l^{n-1/2} - \delta_{l,i_s} D_{i_s} J_{source}(i_s, n\tau). \end{aligned} \tag{10}$$

In the Python code, we implement the update rules (compare eq.(10)) for the `E` and `H` fields as follows

$$\texttt{E[1:-1]} = \frac{\texttt{D}_1\texttt{[1:-1]*(H[1:]-H[:-1])}}{\Delta} + \texttt{C}_1\texttt{[1:-1]*E[1:-1]}$$

$$\texttt{E[i}_s\texttt{]} \quad -= \texttt{D}_1\texttt{[i}_s\texttt{]*J}_{\texttt{source}}\texttt{(n,}\tau\texttt{)}$$

$$\texttt{H} \qquad = \frac{\texttt{B}_1\texttt{[:-1]*(E[1:]-E[:-1])}}{\Delta} + \texttt{A}_1\texttt{[:-1]*H.}$$

The E and H fields are updated for each position on the Yee-grid simultaneously. Further, we only update `E[1:-1]` as $E(0) = E(L) = 0$ at all times as boundary condition. This also enables the cutting of the arrays to compute e.g.[5]

$$H_{l+1/2}^n - H_{l-1/2}^n \tag{11}$$

for all l simultaneously with

$$\texttt{H[1:]-H[:-1].} \tag{12}$$

Furthermore, $\texttt{J}_{\texttt{source}}\texttt{(n,}\tau\texttt{)}$ is a `function` that calculates the current that is created according to eq.(1) for a given τ and n.[6] The functions used for updating the fields, the A, B, C, D, and $\epsilon, \mu, \sigma, \sigma^*$, are defined in separate `python-functions` to maintain clarity.

In the second part of this exercise, the glass is chosen to extend until the right boundary of the box. Here we want to calculate the reflection coefficient of the glass given by

$$R := \frac{|E_{\text{reflected}}^{\text{maximum}}|^2}{|E_{\text{incident}}^{\text{maximum}}|^2}. \tag{13}$$

Here, $E_{\text{reflected}}^{\text{maximum}}$ is the maximum amplitude of the field of the reflected wave and $E_{\text{incident}}^{\text{maximum}}$ of the incident wave. To find appropriate values for the maximum one has to keep in mind that interference with other reflected waves[7] causes the amplitude to change. To obtain reasonable results we determine the

[5] same for E
[6] τ and n are the only two measures defining the time t
[7] in particular the one that travels to the left in the beginning

maximum using `np.max()`, for both the reflected and incident waves, at multiple times. To this end, the first window is chosen as $t_{\text{in}} \in \{1700\tau, ..., 2000\tau\}$ and the second from $t_{\text{ref}} = \{4700\tau, ..., 4950\tau\}$.[8] The first window is to find the maximum amplitude of the incident wave and the second for the reflected one.

Furthermore, to only consider the reflected and incident waves[9] we only investigate the spatial region in the interval $l = 1000, ..., 2000$, which corresponds to the positions $x = 20, ..., 40$. In this spatial interval, we calculate the maximum value of $|E|^2$ for both, the incident and reflected waves. The calculated maxima of $|E|^2$ for the different times in the intervals t_{in} and t_{ref} are saved as two arrays. From these arrays, we can calculate the mean[10], $\overline{|E^{\text{maximum}}|^2}$ and infer from there the reflection coefficient R of the glass

$$R_{result} = \frac{\overline{|E^{\text{maximum}}_{\text{reflected}}|^2}}{\overline{|E^{\text{maximum}}_{\text{incident}}|^2}}. \tag{14}$$

This method will average out the interferences with other reflected unwanted waves.

Furthermore, we can compare our result to the theoretical prediction. An electromagnetic wave that hits a medium perpendicular has the reflection coefficient

$$R_{theory} = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2, \tag{15}$$

where n_1 and n_2 are the refractive indices of the two media. In our case, the wave propagates from vacuum $n_1 = 1$ into glass $n_2 = 1.46$. Hence, we find the reflection coefficient

$$R_{theory} = 0.03497. \tag{16}$$

[8] the steps in the time windows are $\Delta t = \tau$

[9] and not the transmitted wave

[10] using `np.mean()`

3 Results

In this section, we present the results for two different systems.

3.1 The Thin Glass Plate

The first system was already discussed in the previous section and fig.(1), containing two insulators at the boundaries and a thin glass plate in the center. Mathematically, the insulators are described by the electrical conductivity σ and the magnetic loss σ^*

$$\sigma(x) = \sigma^*(x) = \begin{cases} 1 & \text{if} \quad 0 \leq x \leq 6\lambda \\ 0 & \text{if} \quad 6\lambda < x < L\Delta - 6\lambda \\ 1 & \text{if} \quad L\Delta - 6\lambda \leq x \leq L\Delta. \end{cases} \tag{17}$$

Further, the glass plate is described by the electric permittivity

$$\epsilon(x) = \begin{cases} 1 & \text{if} \quad 0 \leq x < L\Delta/2 \\ n_d^2 & \text{if} \quad L\Delta/2 \leq x < L\Delta/2 + 2\lambda \\ 1 & \text{if} \quad L\Delta/2 + 2\lambda \leq x \leq L\Delta, \end{cases} \tag{18}$$

where n_d is the refractive index of the glass with $n_d = 1.46$.[11] Moreover, the system has a constant magnetic permeability

$$\mu(x) = 1. \tag{19}$$

In fig.(3), the results for the different times (a) $n = 2500$ ($t \approx 45$), (b) $n = 3500$ ($t \approx 63$), (c) $n = 3510$ ($t \approx 63$), (d) $n = 4500$ ($t \approx 81$), and (e) $n = 20000$ ($t \approx 360$), and $\tau = 1.05\Delta = 0.021$ for (f) $n = 500$ ($t \approx 9$) are presented and the parameters of the system are $\lambda = 1$, $\Delta = \lambda/50 = 0.02$, $\tau = 0.9\Delta = 0.018$, $X = L\Delta = 100\lambda = 100$, and $L = 5000$.

[11] In the code, both, the σ and ϵ can simply be implemented via `if`-statements

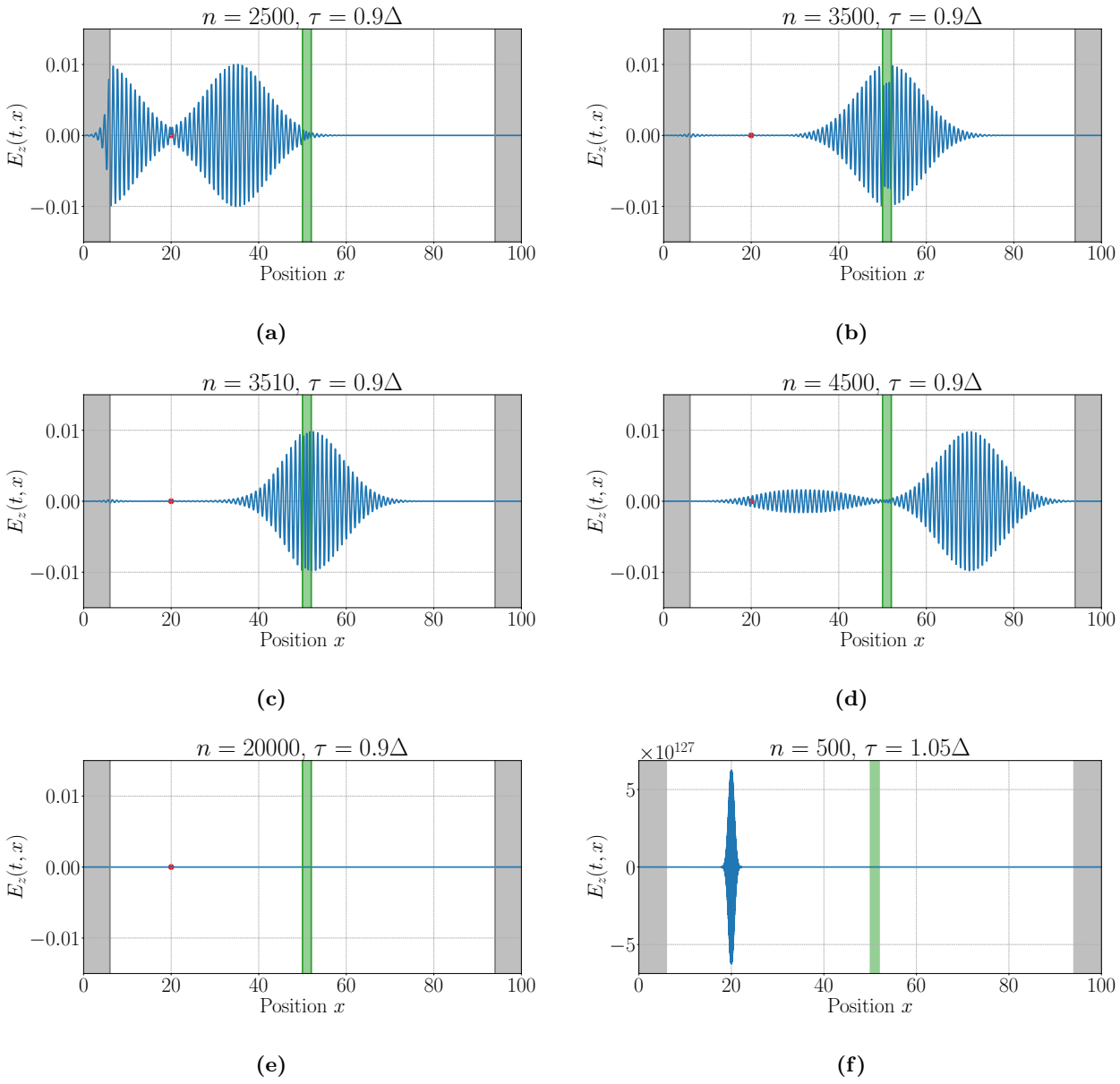

Figure 3: Numerical solutions of the Maxwell equations using the Yee algorithm in one dimension for the system with the thin glass plate. The figures show the z-component of the electric field. The plots contain the glass plate in green and the insulator insulators in grey. The discretizations are $\lambda = 1$, $\Delta = \lambda/50 = 0.02$, $\tau = 0.9\Delta = 0.018$, $X = L\Delta = 100\lambda = 100$, and $L = 5000$. The plots show the solution of the algorithm at different times: (a) $n = 2500$ ($t \approx 45$), (b) $n = 3500$ ($t \approx 63$), (c) $n = 3510$ ($t \approx 63$), (d) $n = 4500$ ($t \approx 81$), and (e) $n = 20000$ ($t \approx 360$), and $\tau = 1.05\Delta = 0.021$ for (f) $n = 500$ ($t \approx 9$).

3.2 The Thick Glass Plate

The second system that we study is very similar. The electrical conductivity σ, the magnetic loss σ^*, and the magnetic permeability μ are the same as before. However, now, we consider a thick glass plate that extends from the middle up to the end of the simulated spatial interval. This is described by the electric permittivity[12]

$$\epsilon(x) = \begin{cases} 1 & \text{if} \quad 0 \leq x < L\Delta/2 \\ n_d^2 & \text{if} \quad L\Delta/2 \leq x < L\Delta, \end{cases} \tag{20}$$

where we have the same refractive index $n_d = 1.46$. This system, at its initial time $t = 0$, can be seen in fig.(4a). The obtained simulation results are presented in fig.(4) for the times t and time descretizations τ: $\tau = 0.9\Delta = 0.018$ for (a) $n = 0$ ($t \approx 0$) (b) $n = 2500$ ($t \approx 45$), (c) $n = 3500$ ($t \approx 63$), (d) $n = 4500$ ($t \approx 81$), as well as (e) $n = 20000$ ($t \approx 360$), and $\tau = 1.05\Delta = 0.021$ for (f) $n = 500$ ($t \approx 9$). The other parameters are identical to the previous case $\lambda = 1$, $\Delta = \lambda/50 = 0.02$, $X = L\Delta = 100\lambda = 100$, and $L = 5000$.

Furthermore, according to eq.(14) and the described method, we determined the following result for the reflection coefficient R

$$R = 0.03537 \tag{21}$$

[12] In the code this can simply be implemented via `if`-statements

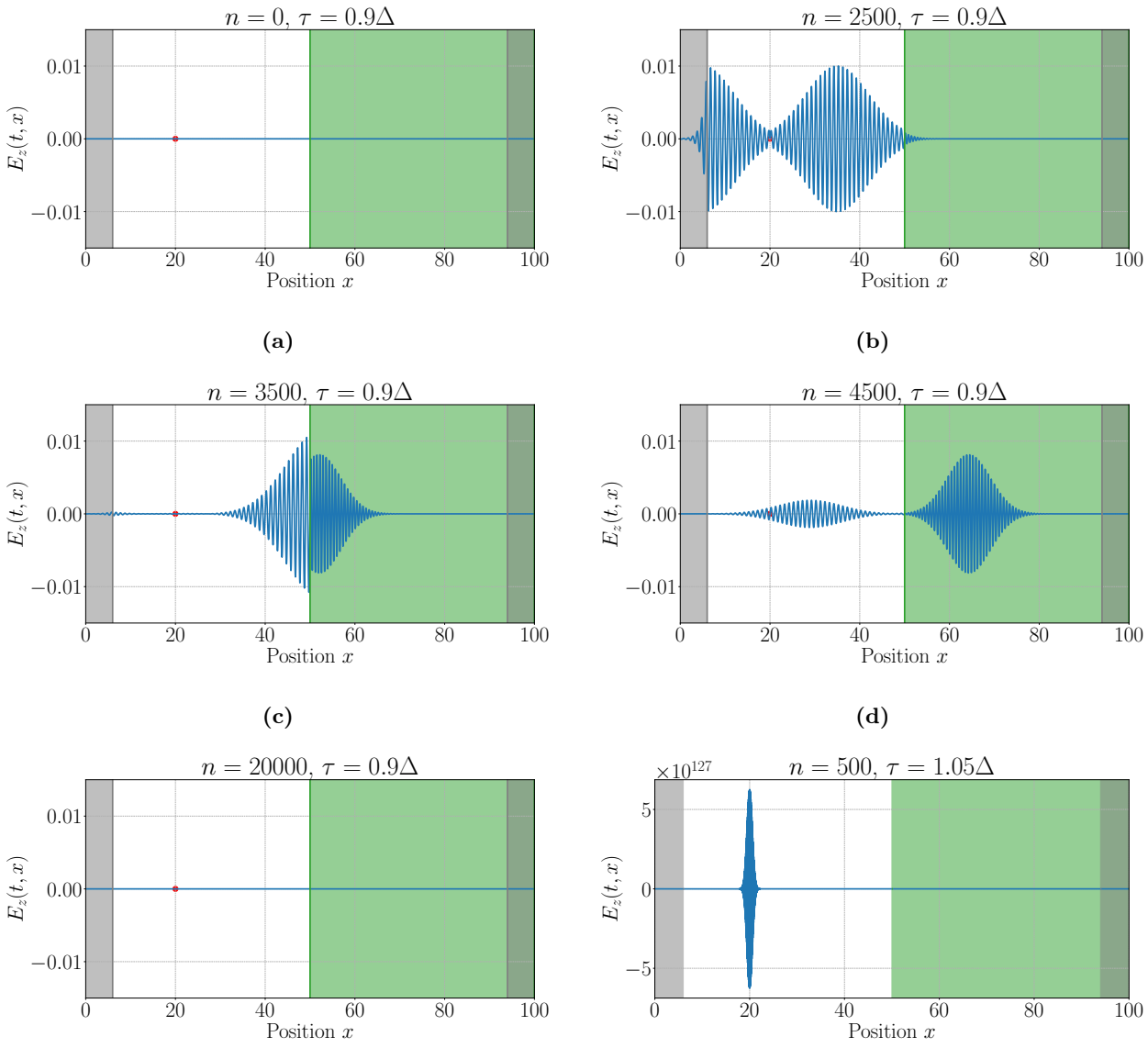

Figure 4: Numerical solutions of the Maxwell equations using the Yee algorithm in one dimension for the system with the thick glass plate. The figures show the z-component of the electric field. The plots contain the glass plate in green and the insulator insulators in grey. The discretizations are $\lambda = 1$, $\Delta = \lambda/50 = 0.02$, $X = L\Delta = 100\lambda = 100$, and $L = 5000$. The plots show the solution of the algorithm at different times and time discretizations: (a) $n = 0$ ($t \approx 0$) (b) $n = 2500$ ($t \approx 45$), (c) $n = 3500$ ($t \approx 63$), (d) $n = 4500$ ($t \approx 81$), and (e) $n = 20000$ ($t \approx 360$), and $\tau = 1.05\Delta = 0.021$ for (f) $n = 500$ ($t \approx 9$).

4 Discussion

First of all, in fig.(3) we can see the results for the system with the thin glass plate at five different times, whereas the initial value $n = 0$ is presented in fig.(1). First, we discuss the results for the discretization $\tau = 0.9\Delta$ in figs. ((a)-(e)) and after that we discuss the result for $\tau = 1.05\Delta$ in fig. (f). In fig.(3a), we can see that the source (compare eq.(1)) at x_s creates two wavepackets going in opposite directions. The wavepacket that is going to the left is mostly absorbed by the insulator (denoted by the grey regions) that is described by σ and σ^* (compare eq.(17)), which can also be seen in the figures for $n = 3500, 3510, 4500, 20000$. The wavepacket propagating to the right passes through the glass plate, shown in fig.(3b). Here, we can see that the amplitude inside the glass (denoted by the green region) is smaller than outside the glass, and further, we notice that the wavelength decreases inside the glass, which was expected. The smaller amplitude inside the glass results from interference with reflected waves. In fig.(3c), we picked a slightly different time given by $n = 3510$. Here, we can see that the amplitude inside the glass is not smaller than outside, which supports the reasoning we made for $n = 3500$ (compare fig.(3b)).
At $n = 4500$, we can see three wavepackets. One wavepacket is propagating in positive x direction and positioned at the right side of the glass, resulting from the transmitted wave. The other two wavepackets are both propagating in negative x direction and they are positioned at the left side of the glass.[13] These originate from the reflection of the incoming wave on the two sides of the glass. Due to the thin glass, the wavepackets overlap which makes them hard to distinguish, resulting in a seemingly longish wavepacket. At last, we present a very late simulation time corresponding to $n = 20000$, where we can see that all waves got absorbed at the boundaries. These results match our physical expectations. Hence, the choice of the discretization $\tau = 0.9\Delta < \Delta$, which satisfies the Courant condition, is valid and yields reasonable results. For a quantitative discussion, we have to investigate physical quantities and compare them to theoretical predictions. This was done for the case of the thick glass plate, where we compute the reflection coefficient.

In contrast to the results for $\tau < \Delta$, we present one simulation result for $\tau > \Delta$, presented in fig.(3f). We find that even at early times, the amplitude of the wave is diverging. At $n = 500$, the amplitude is at 10^{127}. This shows that the results are numerically unstable if the Courant condition $\tau \leq \Delta$ is not met.

The results for the system with the thick glass are presented in fig.(4). Similar to the previous discussion, we first discuss the results for the discretization $\tau = 0.9\Delta$ in figs. ((a)-(e)) and after that we discuss the result for $\tau = 1.05\Delta$ in fig. (f). The system mathematically described by eq.(17) and eq.(20) is presented in fig.(4a) at the initial time corresponding to $n = 0$. Now, we again discuss the results for $n = 2500, 3500, 4500, 20000$. For $t = 2500$, we do not really notice any differences compared to the thin glass, since only the tail of the wavepacket, has reached the glass. For $n = 3500$, we can see that the wavepacket going to the right partially transmitted into the glass, with a smaller amplitude. The larger amplitude of the wave outside the glass is a result of the interference between the incoming wave that is still outside the glass and the already reflected wave. The reflected and transmitted waves are clearly visible in fig.(4d) for $n = 4500$. Here, the difference in the wavelength is clearly noticeable, and, other than the system with the thin glass plate, we only have one reflected wave. Finally, for $n = 20000$, all wavepackets are absorbed by the grey areas.

Furthermore, we present the result for $\tau = 1.05\Delta$ in fig.(4f). We find a similar result compared to the system with the thin glass plate. The amplitude is of order 10^{127}, which implies that the algorithm breaks down if the Courant condition is not met.

[13] Theoretically we have another wave packet originating from the wave that was reflected by the insulator at the beginning. It being substantially smaller than the other waves, we can neglect this wave.

At last, we discuss the results for the reflection coefficient R. If we compare the theoretical prediction (see eq.(16)) and the result of our calculation (see eq.(14)), we find

$$|R_{theory} - R| = 4 \cdot 10^{-4}. \tag{22}$$

Hence, the results of the simulation yield a value that is very close to the theoretical prediction. The difference between theoretical prediction and simulation results is of order 10^{-4}. Due to numerical uncertainties, this is a reasonable result. Therefore, we conclude that the Yee-algorithm yields reasonable physical results.

```
import numpy as np
import matplotlib.pyplot as plt

#numba for faster code compilation
from numba import njit, vectorize, float64

#for fancier plots
import os
os.environ["PATH"] += os.pathsep + '/Library/TeX/texbin'

plt.rcParams["text.usetex"] = True
plt.rcParams["font.family"] = "times new roman"


pi      = np.pi


#parameters
lamb = 1
grid = 50
Delta = lamb/grid

tau1 = 0.9*Delta
tau2 = 1.05*Delta

X = 100*lamb
L = 5000
f = 1/lamb
w = 2*pi*f
m = 10000

#left and right insulator boundary
leftinsu = 6*lamb
rightinsu = (L*Delta -6*lamb)


#function to calculate sigma and sigma*
@vectorize([float64(float64)])
def sigma(x):
    if leftinsu < x< rightinsu:
        return 0
    else:
        return 1

#function to calculate epsilon
#takes in position as well as the end of the right glass
@vectorize([float64(float64,float64)])
def epsilon(x,rightglass):
    n_d = 1.46
    leftglass = (L*Delta/2)
    if leftglass <= x < rightglass:
        return n_d**2
    else:
        return 1

#function defining the mu
@vectorize([float64(float64)])
def mu(x):
    return 1

#position of source
x_s = 20*lamb
```

```
i_s = x_s/Delta
#source function, takes in time t = n\tau
@njit
def J_s(t):
    return np.sin(w*t)*np.exp(-((t-30)/10)**2)

#positions
x = np.linspace(0,X,L+1)


#the four coefficient introduced in the lecture
@njit
def A_l12(l,tau):
    x = (l+1/2)*Delta
    num = 1-(sigma(x)*tau/(2*mu(x)))
    den = 1+(sigma(x)*tau/(2*mu(x)))
    return num/den
@njit
def C_l(l,tau,rightglass):
    x = l*Delta
    num = 1-(sigma(x)*tau/(2*epsilon(x,rightglass)))
    den = 1+(sigma(x)*tau/(2*epsilon(x,rightglass)))
    return num/den

@njit
def B_l12(l,tau):
    x = (l+1/2)*Delta
    num = (tau/mu(x))
    den = 1+(sigma(x)*tau/(2*mu(x)))
    return num/den

@njit
def D_l(l,tau,rightglass):
    x = l*Delta
    num = (tau/epsilon(x,rightglass))
    den = 1+(sigma(x)*tau/(2*epsilon(x,rightglass)))
    return num/den


#update function for H
@njit
def H_n1_l12(l,n, tau, E,H):
    Bl12 = B_l12(l,tau)
    Al12 = A_l12(l,tau)
    return  Bl12[:-1]*(E[1:]-E[:-1])/Delta + Al12[:-1]*H

#update function for E
@njit
def E_n12_l(l,n, tau, E,H,rightglass):
    Dl = D_l(l,tau,rightglass)
    Cl = C_l(l,tau,rightglass)

    E[1:-1] = Dl[1:-1]*(H[1:]-H[:-1])/Delta + Cl[1:-1]*E[1:-1]
    E[int(i_s)] -=  Dl[int(i_s)]*J_s(n*tau1)
    return E

#function that calculates the fields for a given n_max, tau
#glass takes in arguements 1 for "thin" and 0 for "thick"
def maxwell(n_max,tau, glass):

    #defines the boundaries of the glass depending on the chosen glass
    leftglass = (L*Delta/2)
```

```
    if glass == 1:
        rightglass = (L*Delta/2 + 2*lamb)
    else:
        rightglass = L*Delta

    #defining the grid and the initial field values
    l = np.arange(0,L+1)
    H = np.zeros(L)
    E = np.zeros(L+1)

    #array only needed if the reflection coefficients are calculated
    E_incoming = []
    E_reflected = []
    #iteration over time
    for n in range(n_max):
        #updating the fields
        E = E_n12_l(l,n,tau,E,H,rightglass)
        H = H_n1_l12(l,n,tau,E,H)
        #finds maxima in specific time window (only for "thick" glas)
        #needed to calc R
        if glass == 0 and n_max >5000:
            if 1700 < n < 2000:
                E_incoming += [np.max((E[1000:2000])**2)]
            elif 4700 < n <4950:
                E_reflected += [np.max((E[1000:2000])**2)]
    #prints the Reflection coefficient
    if glass == 0 and n_max >5000:
        R = np.mean(E_reflected)/np.mean(E_incoming)
        print("Reflection Coefficient: "+ str(R))
    return E, H

#function needed to set certain plotting arguments
def setup(title, xlabel, ylabel):
    plt.figure(figsize=(16, 9))
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.grid(linestyle="--")
    plt.tight_layout()

#function generating the plots for given values n_max, tau, glass
def maxwell_plots(n_max, tau, glass, set_legend):
    E, H    = maxwell(n_max, tau, glass)
    x       = np.linspace(0, X, L+1)

    leftglass = (L*Delta/2)

    if set_legend:
        plt.rcParams["font.size"]   = "25"
    else:
        plt.rcParams["font.size"]   = "42"

    if tau == tau1:
        savetau = 0.90
    else:
        savetau = 1.05

    if glass == 1.:
        rightglass = (L*Delta/2 + 2*lamb)
    else:
        rightglass = X

```

```
    setup(rf"$n={n_max}$, $\tau={savetau} \Delta$", "Position $x$", "$E_z(t,x)$")

    lw = 3
    plt.plot(x, E, c="tab:blue", lw = lw)
    plt.scatter(x_s, 0, marker="o", color="red", label="source", s=100)

    plt.vlines(leftglass, -0.2, 0.2, color="tab:green", lw=lw)
    plt.vlines(rightglass, -0.2, 0.2, color="tab:green", lw=lw)
    plt.vlines(leftinsu, -0.2, 0.2, color="grey", lw=lw)
    plt.vlines(rightinsu, -0.2, 0.2, color="grey", lw=lw)

    plt.axvspan(leftglass, rightglass, alpha=0.5, color="tab:green", label="glass")
    plt.axvspan(0, leftinsu, alpha=0.5, color="grey", label="insulator")
    plt.axvspan(rightinsu, X, alpha=0.5, color="grey")

    plt.locator_params(axis='y', nbins=5)
    plt.xlim(0, 100)

    if tau == tau1:
        plt.ylim(-0.015, 0.015)

    if set_legend:
        plt.legend(bbox_to_anchor=(1.01, 0.65), fancybox=True, shadow=True, fontsize
    =25)

    plt.tight_layout()
    if glass == 1.:
        plt.savefig(f"Plots/maxwell_tau{savetau}_thinglass_nmax{n_max}.pdf")
    else:
        plt.savefig(f"Plots/maxwell_tau{savetau}_thickglass_nmax{n_max}.pdf")
    plt.show()


#values at which we are interested to generate plot
n_max0 = 0
n_max1 = 2500
n_max2 = 3500
n_max3 = 3510
n_max4 = 4500
n_max5 = 20000


#generation of plots
maxwell_plots(n_max0, tau1, 0., False)
maxwell_plots(n_max1, tau1, 0., False)
maxwell_plots(n_max2, tau1, 0., False)
maxwell_plots(n_max3, tau1, 0., False)
maxwell_plots(n_max4, tau1, 0., False)
maxwell_plots(n_max5, tau1, 0., False)

maxwell_plots(500, tau2, 0., False)

maxwell_plots(500, tau2, 1., False)

maxwell_plots(n_max0, tau1, 1., True)
maxwell_plots(n_max1, tau1, 1., False)
maxwell_plots(n_max2, tau1, 1., False)
maxwell_plots(n_max3, tau1, 1., False)
maxwell_plots(n_max4, tau1, 1., False)
maxwell_plots(n_max5, tau1, 1., False)
```