EXERCISE 6: MAXWELL'S EQUATION

Computational Physics

Authors
George Farah, Simon Suchan
Email: george.farah@rwth-aachen.de, simon.suchan@rwth-aachen.de
Matricle No: 421409, 397059

Contents

1 Introduction **3**

2 Simulation Model and Method **3**

3 Simulation Results **5**

3.1 Exercise one 5

3.2 Exercise Two 5

4 Conclusion and Discussion **5**

5 Appendix **8**

1 Introduction

In this exercise sheet, Maxwell's equation $\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$ in one dimension that represents the light transmission and reflection will be simulated.

At first, We will see the light transmission and reflection in 1d with absorbing boundary conditions and a thin layer of glass in the middle. Afterward with the same boundary conditions and source, we will see the behavior of light with a thicker glass layer that covers half the simulation box.
In both cases, Yee algorithm will be used for the simulation.

2 Simulation Model and Method

The 1d Maxwell eqaution that represents light transmission is the following:

$$\begin{aligned}
\frac{\partial H_y(x,t)}{\partial t} &= \frac{1}{\mu(x)}[\frac{\partial E_z(x,t)}{\partial x} - \sigma^*(x)H_y(x,t)] \\
\frac{\partial E_z(x,t)}{\partial t} &= \frac{1}{\epsilon(x)}[\frac{\partial_y(x,t)}{\partial x} - J_{source_y}(x,t) - \sigma(x)E_z(x,y)]
\end{aligned} \tag{1}$$

Where $\epsilon(x)$ represents the material constants, $\mu(x)$, $\sigma(x)$ and $\sigma^*(x)$ represent the boundary conditions for the E and H respectively, and they are assumed to equal. And $\mu(x) = 1$ is assumed everywhere.

J_{source} is the starting wave packet: $J_S(i_s, t) = sin(2\pi t f)\exp(-((t-30)/10)^2)$

For the simulation parameters, we use the following: wavelength $\lambda = 1$, number of grid point 50, spatial resolution $\Delta = \frac{\lambda}{50} = 0.02$, temporal resolution $\tau_1 = 0.9\Delta$ $\tau_2 = 1.05\Delta$, length of simulation box $X = 100\lambda = L\Delta \rightarrow L = 5000$, number of time steps is $m = 10000$, and the current source starts at $x_s = 20 \rightarrow i_s = 1000$

For the absorbing boundary conditions and according to the simulation parameters, and with $\sigma(x) = \sigma^*(x)$:

$$\sigma(x) = \begin{cases} 1, & \text{if } 0 \leq x \leq 6, \\ 0, & \text{if } 6 < x < 94, \\ 1, & \text{if } 94 \leq x \leq 100 \end{cases} \tag{2}$$

For the glass layer, we have two cases, first a glass layer at the middel of the system with thickness 2 and index of refraction $n = 1.46$:

$$\epsilon(x) = \begin{cases} 1, & \text{if } 0 \leq x < 50, \\ n^2, & \text{if } 50 \leq x < 52, \\ 1, & \text{if } 52 \leq x \leq 100 \end{cases} \tag{3}$$

In the second case we have a glass layer with a thickness that covers half the simulation box:

$$\epsilon_2(x) = \begin{cases} 1, & \text{if } 0 \leq x < 50, \\ n^2, & \text{if } 50 \leq x \leq 100, \end{cases} \tag{4}$$

For the simulation a "Yee Grid" is going to be used, where the points of E and H are set along a 2d grid that represents the temporal and spatial dimensions, at half steps the magentic field H lies, and at full time steps the electric field. Then at each time step m, all spatial steps for H and E will be updated, for the H spatial steps, we start from 1 (since they lie at a half step after the E field) and stop at L. While for E field we start at 0 and finish at L-1 (since they lie a half step before).
As shown in the exercise description, the update rule of magnetic and electric fields while lying on the Yee grid:

$$\begin{aligned} H_y|_{l+\frac{1}{2}}^{m+1} &= A_{l+1/2} H_y|_{l+1/2}^{m} + \left(\frac{E_z|_{l+1}^{m+1/2} - E_z|_{l}^{m+1/2}}{\Delta} \right) \\ E_z|_{1}^{m+1/2} &= C_l E_z|_{l}^{n-1/2} + D_l \left(\frac{H_y|_{l+1/2}^{m} - H_y|_{l-1/2}^{m}}{\Delta} - J_{\text{source}}|_{l}^{m} \right) \end{aligned} \tag{5}$$

Where

$$\begin{aligned} A_{l+1/2} &= \frac{1 - \left(\frac{\tau\sigma_{l+1/2}}{2}\right)}{1 + \left(\frac{\tau\sigma_{l+1/2}}{2}\right)} \\ B_{l+1/2} &= \frac{\tau}{1 + \frac{\sigma_{l+1/2}\tau}{2}} \\ C_l &= \frac{1 - \frac{\sigma_l \tau}{2\epsilon_l}}{1 + \frac{\sigma_l \tau}{2\epsilon_l}} \\ D_l &= \frac{\frac{\tau}{\epsilon_l}}{1 + \frac{\sigma_l \tau}{2\epsilon_l}} \end{aligned}$$

As mentioned earlier at each time step we will calclculate all the spatial dimensions steps, to do so we vectorize the arrays for E,H,A,B,C,D so the algorithm runs faster, meaning at each time step we will calculate the following vectors:

$$\begin{aligned} H[0:L-1] &= A[0:L-1] * H_{prev}[0:L-1] + B[0:L-1](E[1:L] - E[0:L-1])/\Delta \\ E[1:L] &= C[1:L] * E_{prev}[1:] + D[1:L] * (H[1:L] - H[0:L-1])/\Delta \\ E[i_s] &+= D[i_s] * J_{source}(n * \tau) \end{aligned} \tag{6}$$

With starting conditions: $E[0:L] = 0, H[0:L-1] = 0$ and the values at each spatial steps for the vectors A,B,C,D are predetermined according to the upper equations of boundaries and glass coefficients.
Lastly from the elctrical field amplitued in the second case where the glass covers half the area we want to calculate the reflection coefficient of the glass:

$$R = \frac{|E^2_{\text{max reflected}}|^2}{|E_{\text{max incident}}|^2} \tag{7}$$

For all exercises programming language Python has been used with its libraries numpy and matplotlib for plots.

3 Simulation Results

3.1 Exercise one

Figures using thin glass in the middle (see equation (3)) and border conditions for both E and H as in (2) and $\tau = 0.9\Delta$
Using the other condition for $\tau = 1.05\Delta$ the algorithm is not stable and does not deliver any meaningful output because the time step in this case is larger than the spatial resolution which does not allow us to use the Yee approach where at each time step we can include all the possible spatial positions.

3.2 Exercise Two

Figures using thin glass in the middle (see equation (4)) and border conditions for both E and H as in (2) and $\tau = 0.9\Delta$
Now using equation (7) to caluclate the reflection coefficient, we use the simulated electric field amplitude to get the the maximum reflection at the area $x < 2500$ and the maximum incident at $x \geq 2500$ to get R=0.252

4 Conclusion and Discussion

We have seen in this exercise how to simulate light according to Maxwell equation in 1d, we have seen how the behavior works for thin and thick glass material. The simulation has been done according to the Yee algorithm that spreads the spatial and temporal dimensions on a grid where the E field lies at complete steps while the magenteic field lies on half steps. We have seen also that for this algorithm to be stable the temporal resolution steps needs to be on a smaller scale than that of the spatial resolution, and that is what we call the Courant condition. At the end we have calculated the reflection coeffieients and got a value of 0.25

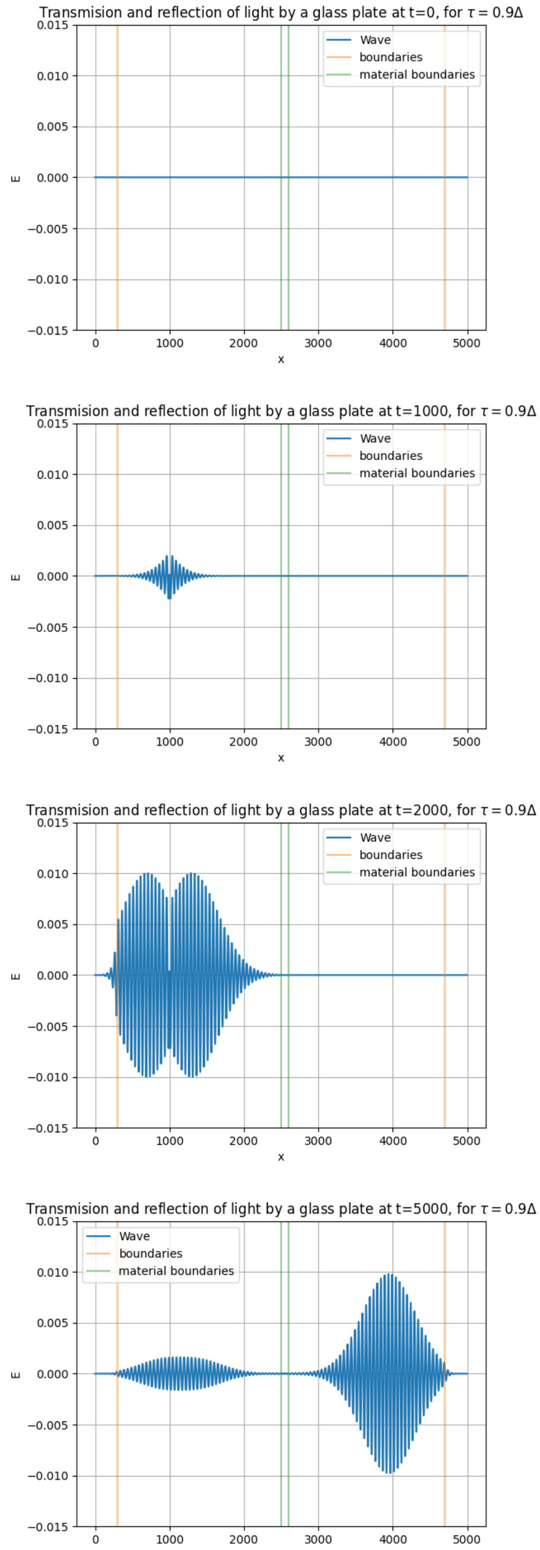

Figure 1: Four Figures for light transmission and reflection at different times.

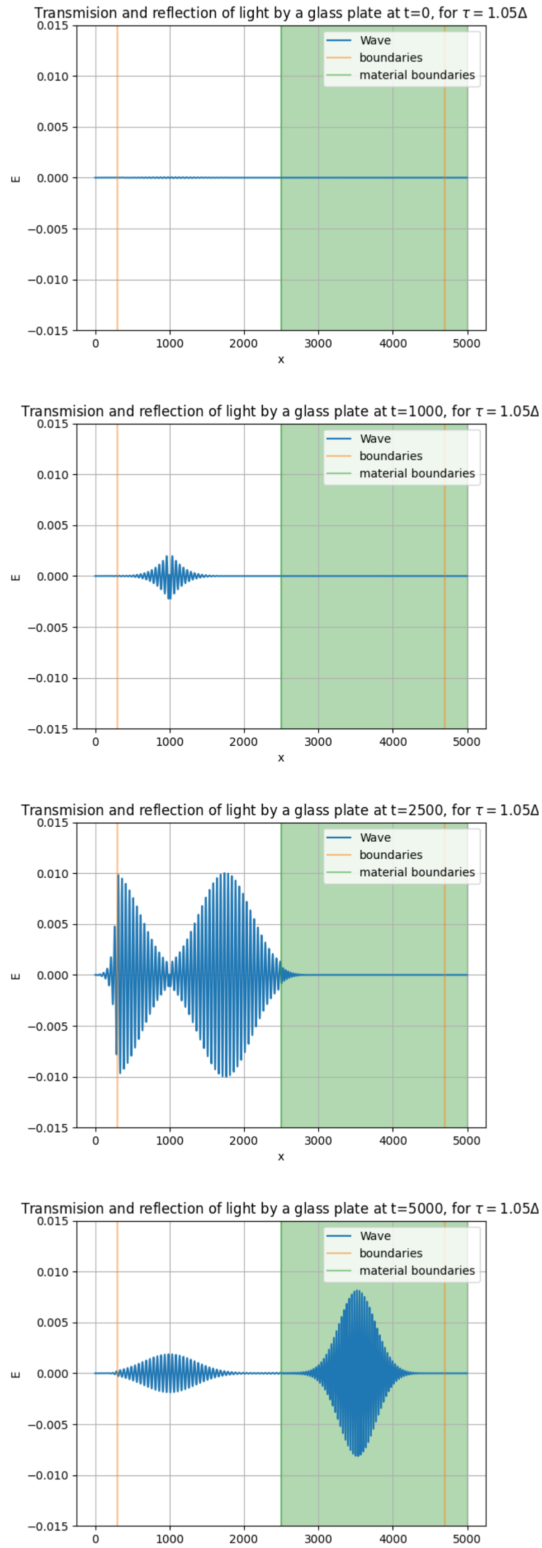

Figure 2: Four Figures for light transmission and reflection at different times using a thick glass that covers half the spatial area.

5 Appendix

Here is the code for the first three exercise in python with the plots:

```
"""
Exercise 6:
Yee Algorithm for simulating Maxwell equation

"""
#%%
import numpy as np
import matplotlib.pyplot as plt

#%%

L = 5000
delta = 0.02
tau1 = 0.9 * delta
tau2 = 1.05 * delta
freq = 2 * np.pi
m = 10000 ## time steps
jsource = 1000


Ex = np.zeros(L)
Hz = np.zeros(L)
Ex_prev = np.zeros(L)
Hz_prev = np.zeros(L)

def Source_Function(t:float):
    """Wave Packet assuming f=1

    Parameters
    ----------
    t : float
        time

    Returns
    -------
    float
        source function
    """
    return np.sin(2*np.pi*t) * np.exp(-((t-30)/10)**2)


def sigma(x):
```

```
    if x >=0 and x<=6:
        return 1
    elif x>6 and x<L*delta-6:
        return 0
    else:
        return 1

def epsilon(x):
    if x >= 0 and x < L*delta/2:
        return 1
    elif L*delta/2 <= x and x < (L*delta/2)+2:
        return 1.46 ** 2
    else:
        return 1

def epsilon_thicc(x):
    if x >= 0 and x < L*delta/2:
        return 1
    elif L*delta/2 <= x and x < (L*delta/2)+2:
        return 1.46 ** 2
    else:
        return 1

def A(sigma_array, tau):
    nominator = 1 - (sigma_array * tau)/(2)
    denominator = 1 + (sigma_array * tau)/(2)
    return nominator / denominator

def B(sigma_array, tau):
    nominator = tau
    denominator = 1 + (sigma_array*tau)/(2)
    return nominator / denominator

def C(sigma_array, epsilon_array, tau):
    nominator = 1 - (sigma_array * tau)/(2*epsilon_array)
    denominator = 1 + (sigma_array * tau)/(2*epsilon_array)
    return nominator / denominator

def D(sigma_array, epsilon_array, tau):
    nominator = tau/epsilon_array
    denominator = 1 + (sigma_array * tau)/(2*epsilon_array)
    return nominator / denominator

## defining the simulation
x = np.linspace(0,100,5000)
```

```python
sigma_array = np.zeros(L)
for i in range(len(sigma_array)):
    sigma_array[i] = sigma(x[i])

epsilon_array = np.zeros(L)
for i in range(len(epsilon_array)):
    epsilon_array[i] = epsilon(x[i])

taus = tau1

A_array = A(sigma_array,tau1)
B_array = B(sigma_array, tau1)
C_array = C(sigma_array, epsilon_array, taus)
D_array = D(sigma_array, epsilon_array, taus)


#%%
## In the grid n is T in full time step
## And j is halp space steps (l in the script)
for n in range(m):
    #Update magnetic field boundaries
    Hz[L-1] = Hz_prev[L-2]
    #Update magnetic field
    #for j in range(L-1):
    Hz[0:L-1] = A_array[0:L-1] * Hz_prev[0:L-1] +
        B_array[0:L-1] * (Ex[1:L] - Ex[0:L-1])/delta
    Hz_prev = Hz
    #Magnetic field source
    Hz[jsource-1] -= Source_Function(n)
    Hz_prev[jsource-1] = Hz[jsource-1]
    #Update electric field boundaries
    Ex[0] = Ex_prev[1]
    #Update electric field
    #for j in range(1,L):
    Ex[1:L] = C_array[1:]  * Ex_prev[1:] + D_array[1:L] *(
        (Hz[1:L]-Hz[0:L-1])/delta -  Source_Function(n))
    Ex_prev = Ex
    Ex[jsource] +=
        D(sigma_array[jsource],epsilon_array[jsource] , taus)
        * Source_Function((n)*taus)
    Ex_prev[jsource] = Ex[jsource]

    if  n == 0 or n == 1000 or n == 2000 or n == 5000:
    #   plt.figure(figsize=(20,10))
        plt.plot(Ex,label="Wave")
```

```
        plt.grid()
        plt.title(fr"Transmision and reflection of light by a glass plate at t={n}, for $ \tau = 1.05 \Delta$")
        plt.plot(sigma_array-0.5, label="boundaries",alpha=0.5)
        plt.plot(epsilon_array-2, label="material boundaries",alpha=0.5)
        plt.ylim(-0.015,0.015)
        plt.legend()
        plt.xlabel("x")
        plt.ylabel("E")
        plt.savefig(f"Ex_tau105_t{n}.png")
        plt.show()
        plt.close()

# %%
### Simulating the thick condition

epsilon_array_thicc = np.zeros(L)

def epsilon_thicc(x):
    if x >= 0 and x < L*delta/2:
        return 1
    elif L*delta/2 <= x :
        return 1.46 ** 2
    else:
        return 1

for i in range(len(epsilon_array_thicc)):
    epsilon_array_thicc[i] = epsilon_thicc(x[i])
#%%
taus = tau1

A_array = A(sigma_array,tau1)
B_array = B(sigma_array, tau1)
C_array = C(sigma_array, epsilon_array_thicc, taus)
D_array = D(sigma_array, epsilon_array_thicc, taus)
E_array = []

#%%
for n in range(m):
    #Update magnetic field boundaries
    Hz[L-1] = Hz_prev[L-2]
    #Update magnetic field
    #for j in range(L-1):
    Hz[0:L-1] = A_array[0:L-1] * Hz_prev[0:L-1] +
```

```
            B_array[0:L-1] * (Ex[1:L] - Ex[0:L-1])/delta
        Hz_prev = Hz
        #Magnetic field source
        Hz[jsource-1] -= Source_Function(n)
        Hz_prev[jsource-1] = Hz[jsource-1]
        #Update electric field boundaries
        Ex[0] = Ex_prev[1]
        #Update electric field
        #for j in range(1,L):
        Ex[1:L] = C_array[1:]  * Ex_prev[1:] + D_array[1:L] *(
            (Hz[1:L]-Hz[0:L-1])/delta -  Source_Function(n))
        Ex_prev = Ex
        Ex[jsource] +=
            D(sigma_array[jsource],epsilon_array[jsource] , taus)
            * Source_Function((n)*taus)
        Ex_prev[jsource] = Ex[jsource]
        E_array.append(Ex)

        if  n == 0 or n == 100 or n == 1000 or n == 2500 or n ==
            5000:
        #   plt.figure(figsize=(20,10))
            plt.plot(Ex,label="Wave")
            plt.grid()
            plt.title(fr"Transmision and reflection of light by a
                glass plate at t={n}, for $ \tau = 1.05 \Delta$")
            plt.plot(sigma_array-0.5, label="boundaries",alpha=0.5)
            plt.plot(epsilon_array_thicc-2, label="material
                boundaries",alpha=0.5)
            plt.ylim(-0.015,0.015)
            plt.fill_between(np.arange(2500,5000), -0.015,
                0.015,alpha=0.3, color="green")
            plt.legend()
            plt.xlabel("x")
            plt.ylabel("E")
            plt.savefig(f"Ex_tau09_t{n}_thicc.png")
            plt.show()
            plt.close()
# %%
## caculating R
E_array = np.array(E_array)
E_power = np.abs(E_array)**2
R = np.max(E_power[-1][0:2500])/np.max(E_power[-1][2500:])
```