Computational Physics - Exercise 5

Joel Schumacher, 309231, joel.schumacher@rwth-aachen.de

June 23, 2017

Yee algorithm

Introduction

In this exercise the time evolution of a one-dimensional electromagnetic wave packet is examined, while propagating through an evacuated cavity ($\epsilon = 1$, $\mu = 1$, $\sigma = \sigma* = 0$) with dissipative boundaries ($\sigma = \sigma* = 1$, thickness: 6λ) . In the middle of the cavity a glass plate is placed ($\epsilon = n^2 = (1.46)^2$).

The wavelength of the wave is set to $\lambda = 1$ and the size of the cavity is $X = 100\lambda$. The size of the glass plate is $g = 2\lambda$. The wave packet is emitted at $x_S = 20\lambda$ and has the functional form (with $f = 1$):

$$J(x_S, t) = \sin(2\pi t) \cdot \exp\left(-\left(\frac{t-30}{10}\right)^2\right)$$

In a variation of this simulation the thickness of the glass plate is increased to take up half of the system: $g = 50\lambda$ and the maximum magnitude of the electric field incident and reflected is measured to approximate the reflection coefficient of glass.

Simulation model

For the simulation the Yee algorithm is utilized, starting from the Maxwell-Equations in 3 dimensions:

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

a set of equations for 1 dimensional systems can be derived (x-directed, z-polarized, transverse mode):

$$\begin{aligned}
\frac{\partial H_y(r,t)}{\partial t} &= \frac{1}{\mu}\left[\frac{\partial E_z(r,t)}{\partial x} - M_y(r,t)\right] \\
\frac{\partial E_z(r,t)}{\partial t} &= \frac{1}{\epsilon}\left[\frac{\partial H_y(r,t)}{\partial x} - J_z(r,t)\right]
\end{aligned}$$

In the Yee algorithm the divergence-freeness (stated by the Gauss law relations) is ensured by the choice of the simulation grid in that every $\vec{E}$ component is surrounded by 4 circulating $\vec{H}$ components and vice versa (in the 3D case).

Therefore the Yee algorithm can solve for both fields simultaneously, by interleaving the simulation grids of $\vec{E}$ and $\vec{H}$.

In the one-dimensional case this simplifies to the simulation grid for E_z and the grid for H_y being displaced by $\frac{\Delta}{2}$, where Δ is the spacing of the simulation grid.

This displacement is also done in time by $\frac{\tau}{2}$, where τ is the time step of the simulation, to implement a leapfrog algorithm.

Furthermore for the numerical solution the finite difference approximation for the first derivative is used, so that the 1 dimensional Maxwell equations become:

$$
\begin{aligned}
\frac{H_y|_{l+1/2}^{n+1} - H_y|_{l+1/2}^{n}}{\tau} &= \frac{1}{\mu_{l+1/2}} \left[\frac{E_z|_{l+1}^{n+1/2} - E_z|_{l}^{n+1/2}}{\Delta} - M_{source_y}|_{l+1/2}^{n+1/2} - \sigma *_{l+1/2} H_y|_{l+1/2}^{n+1/2} \right] \\
\frac{E_z|_{l}^{n+1/2} - E_z|_{l}^{n-1/2}}{\tau} &= \frac{1}{\epsilon_l} \left[\frac{H_y|_{l+1/2}^{n} - H_y|_{l-1/2}^{n}}{\Delta} - J_{source_z}|_{l}^{n} - \sigma_l E_z|_{l}^{n} \right]
\end{aligned}
$$

while Yee's notation is used here in which the lower index denotes the spatial dependence of the quantity and the upper index denotes the temporal dependence.

Rearranging and setting $M_{source} = 0$ yields the update rules:

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

with:

$$
\begin{aligned}
A &= \left(\frac{1 - \frac{\sigma *_{l+1/2} \tau}{2\mu_{l+1/2}}}{1 + \frac{\sigma *_{l+1/2} \tau}{2\mu_{l+1/2}}} \right) \\
B &= \left(\frac{\frac{\tau}{\mu_{l+1/2}}}{1 + \frac{\sigma *_{l+1/2} \tau}{2\mu_{l+1/2}}} \right) \\
C &= \left(\frac{1 - \frac{\sigma_l \tau}{2\epsilon_l}}{1 + \frac{\sigma_l \tau}{2\epsilon_l}} \right) \\
D &= \left(\frac{\frac{\tau}{2\epsilon_l}}{1 + \frac{\sigma_l \tau}{2\epsilon_l}} \right)
\end{aligned}
$$

The parameters chosen for the simulation are $\Delta = \lambda/50$ and $\tau = 0.9\Delta$.

Simulation results

The electric field at different times can be seen in figures 1, 2, 3 and 4. If $\tau = 1.05\Delta$ is chosen as the time step, the simulation does not terminate and the software package used (numpy) throws exceptions regarding the occurence of NaN and INF, which implies severe numerical instability.

Otherwise a partial reflection of the incoming wave at the glass plate can be observed and a rapid absorption of the wave in the boundaries.

Figures with the same parameters, but with a glass plate of thickness $g = 50\lambda$ are 5, 6, 7 and 8.

The maximum magnitude of the electric field on the left side of the glass plate before and after the reflection are:

$$
\begin{aligned}
E_{max,incident} &= \max_{x<X/2} |E(t = 2500)| = 0.01001 \\
E_{max,reflected} &= \max_{x<X/2} |E(t = 5000)| = 0.00188
\end{aligned}
$$

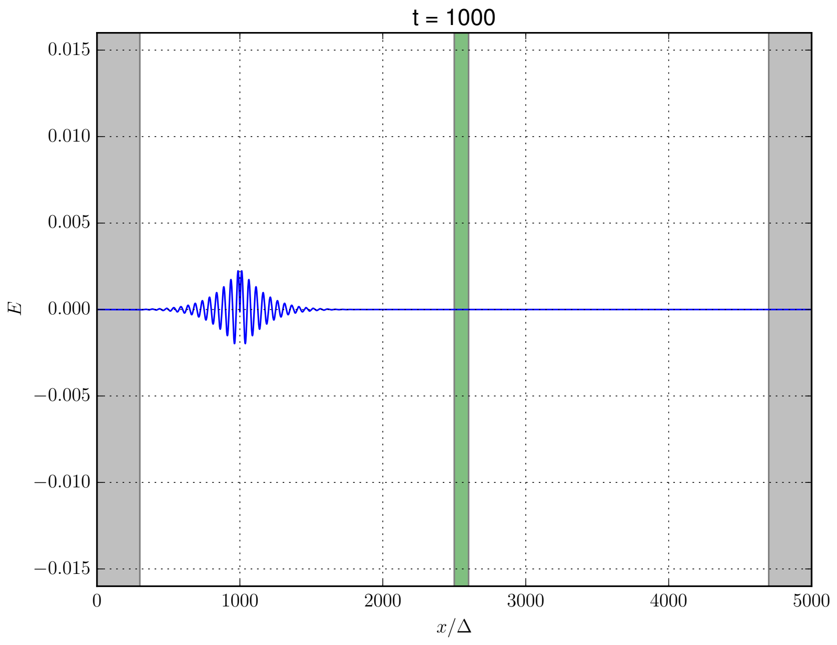

Figure 1: Electric field at time $t = 1000/\tau$, glass plate thickness $g = 2\lambda$. The title of the plot denotes the step number.

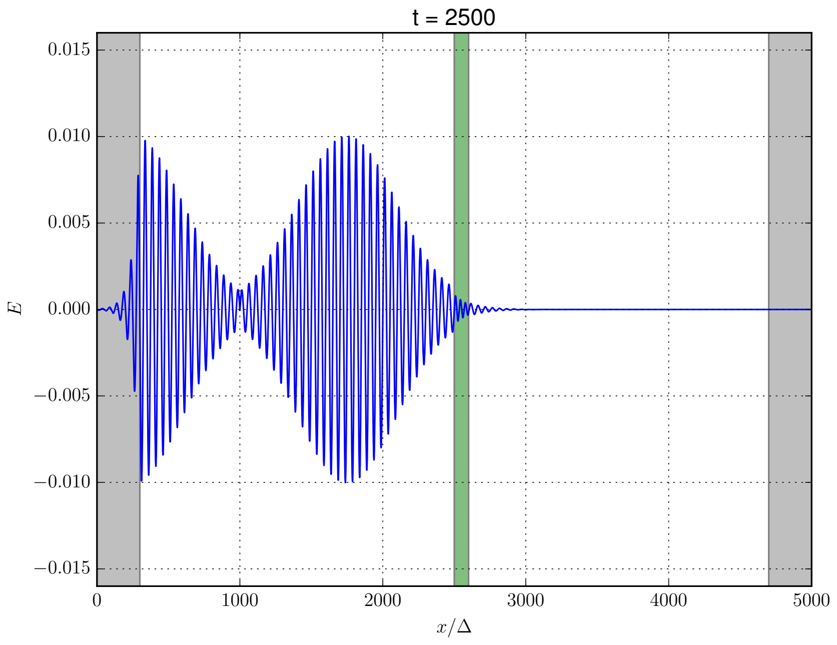

Figure 2: Electric field at time $t = 2500/\tau$, glass plate thickness $g = 2\lambda$. The title of the plot denotes the step number.

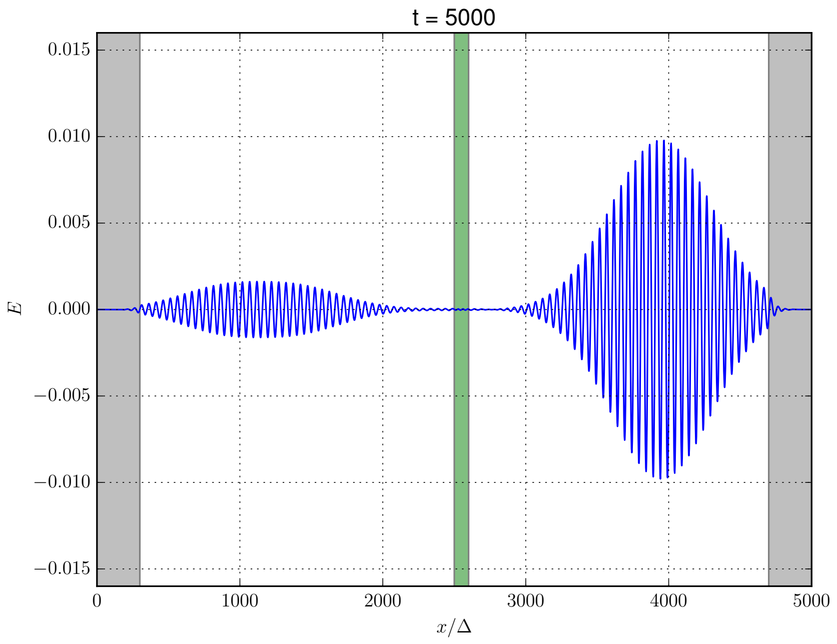

Figure 3: Electric field at time $t = 5000/\tau$, glass plate thickness $g = 2\lambda$. The title of the plot denotes the step number.

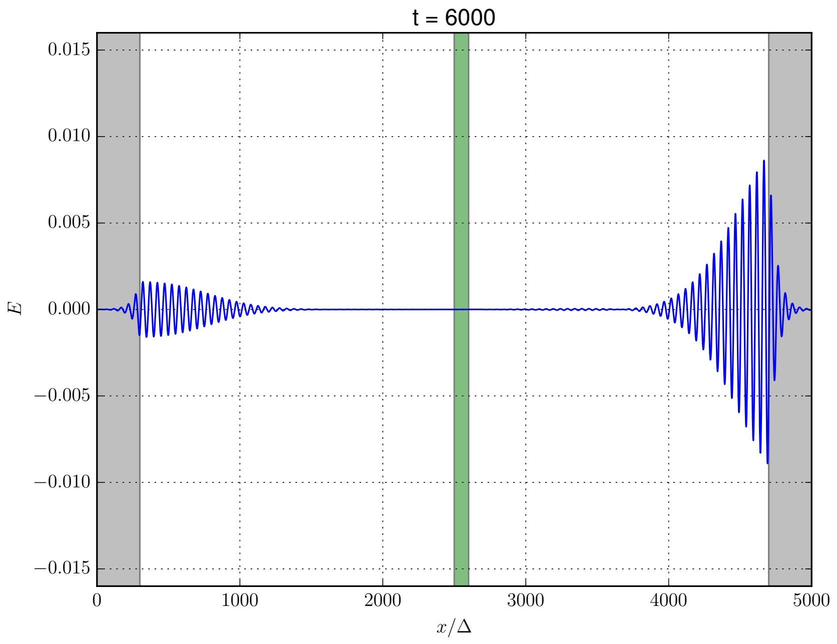

Figure 4: Electric field at time $t = 6000/\tau$, glass plate thickness $g = 2\lambda$. The title of the plot denotes the step number.

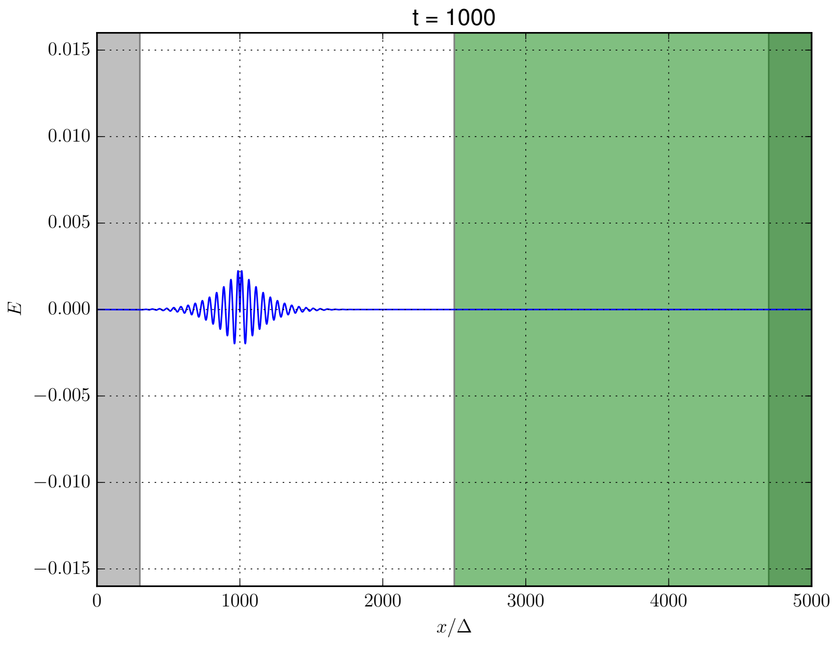

Figure 5: Electric field at time $t = 1000/\tau$, glass plate thickness $g = 50\lambda$. The title of the plot denotes the step number.

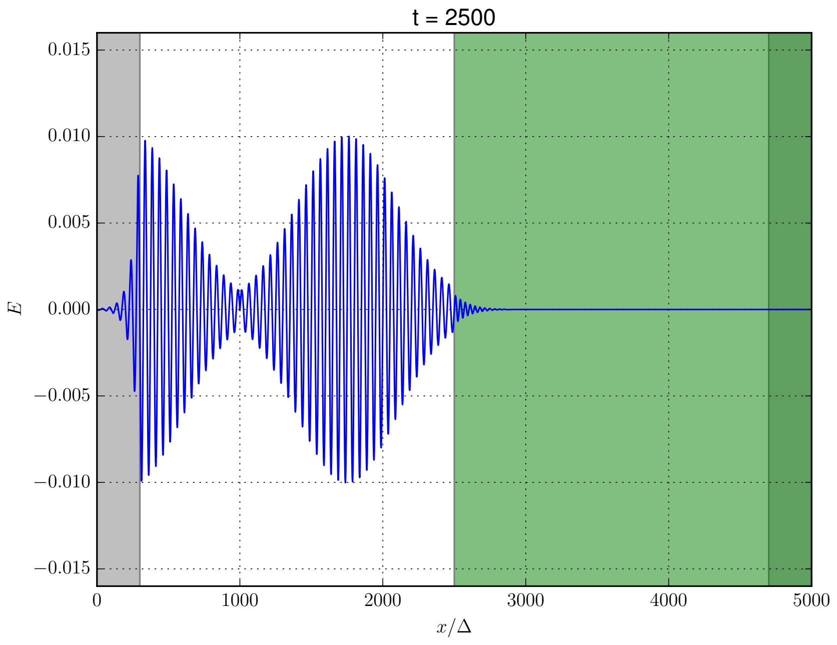

Figure 6: Electric field at time $t = 2500/\tau$, glass plate thickness $g = 50\lambda$. The title of the plot denotes the step number.

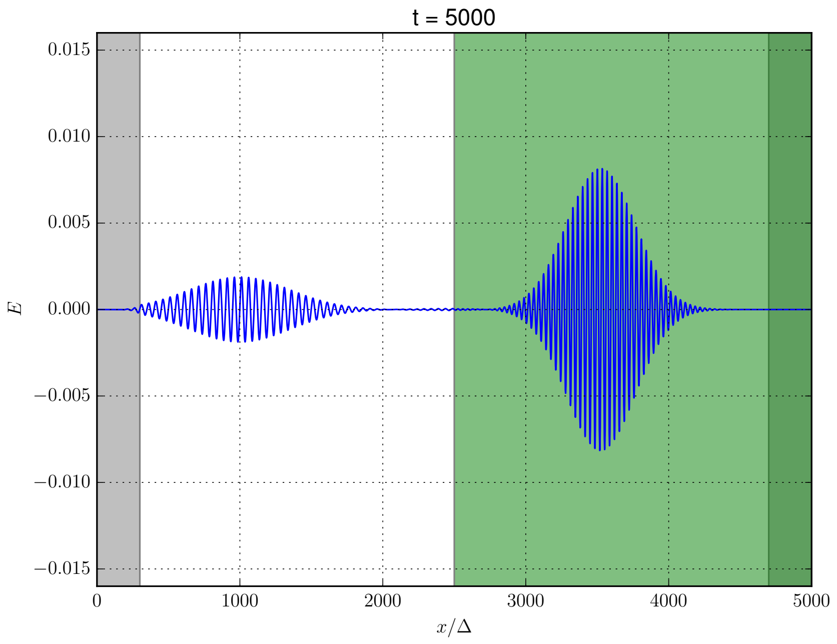

Figure 7: Electric field at time $t = 5000/\tau$, glass plate thickness $g = 50\lambda$. The title of the plot denotes the step number.

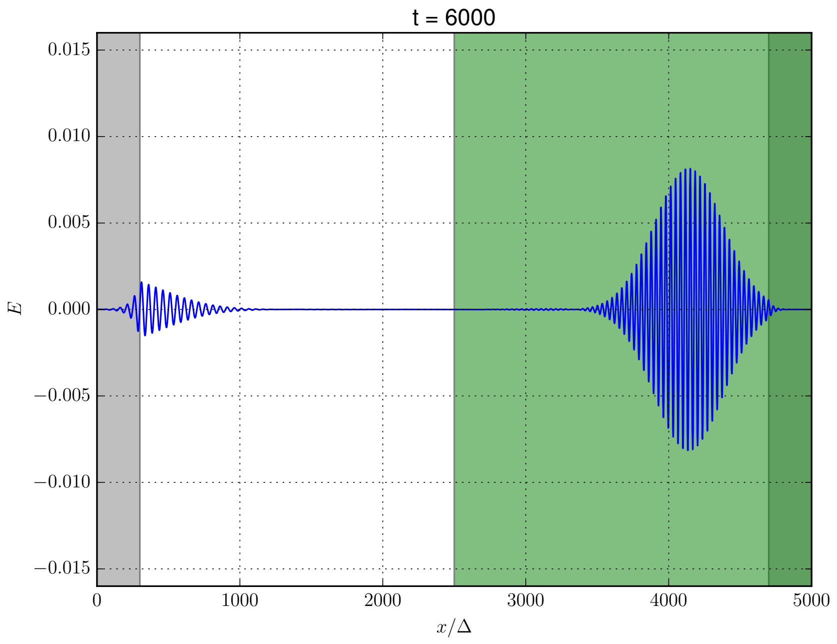

Figure 8: Electric field at time $t = 6000/\tau$, glass plate thickness $g = 50\lambda$. The title of the plot denotes the step number.

Discussion

The simulation gives plausible results. For the thin glass plate the wave is partially reflected, as it would be expected for real light-glass interaction. The conducting boundaries show a rapid decay of the magnitude of the electric field, which is also exactly as expected. In the case of $\tau = 1.05\Delta$ the simulation becomes unstable, because the Courant number $S = \frac{c\tau}{|\Delta|} = \frac{\tau}{|\Delta|}$ exceeds one, which is beyond the upper bound for the stable execution of the algorithm.

Examining a thicker glass plate expected phenomena such as a narrowing of the wave packet and a slow down of the propagation speed can be observed. With the maximum magnitudes of the electric fields left of the glass plate we can estimate the reflection coefficient of glass i.e. a dielectric with $n = 1.46$:

$$R = \frac{E^2_{max,reflected}}{E^2_{max,incident}} = \frac{0.00188^2}{0.01001^2} = 0.03527$$

The theoretical prediction of the reflection coefficient is:

$$R_{theory} = \frac{(n-1.46)^2}{(n+1.46)^2} = 0.03497$$

Which matches sufficiently well with the result from the simulation.

In conclusion the Yee algorithm can correctly solve Maxwell's equations, as shown by physical results regarding multiple aspects of electromagnetic waves, provided parameter sets that ensure numerical stability. It is relatively easy to implement and computationally inexpensive in the 1D case, but scales in $\mathcal{O}(N^d)$ in the d-dimensional case with a grid side length of N, which implies a substantial increase in computational requirements for 3-dimensional simulations.

Appendix

```python
import math

import numpy as np
import matplotlib.pyplot as plt

LAMBDA = 1
NUM_GRID_POINTS_PER_LAMBDA = 50
DELTA = LAMBDA / NUM_GRID_POINTS_PER_LAMBDA # spatial resolution, 0.02
TAU = 0.9 * DELTA # temporal resolution
#TAU = 1.05 * DELTA # temporal resolution
X = 100*LAMBDA # Length of simulation box in spatial units
L = math.floor(X / DELTA) # Length of simulation box in index units, 5000
F = 1
NUM_TIMESTEPS = 10000
GLASS_THICKNESS = 2*LAMBDA
#GLASS_THICKNESS = 50*LAMBDA
N_GLASS = 1.46
SOURCE_POS = 20*LAMBDA
SOURCE_POS_I = math.floor(SOURCE_POS / DELTA) # 1000
BOUNDARY_THICKNESS = 6*LAMBDA

# turn on and off slowly to get a wave packet
def source(t):
    e = (t-30)/10
    return math.sin(2*math.pi*F*t) * math.exp(-e*e)

def sigma(x):
    if (x >= 0 and x <= BOUNDARY_THICKNESS) or (x >= X-BOUNDARY_THICKNESS and x
         <= X):
        return 1
    else:
        return 0
```

```

def eps(x):
    if x >= X/2 and x < X/2 + GLASS_THICKNESS:
        return N_GLASS*N_GLASS
    else:
        return 1

def mu(x):
    return 1

C = np.zeros(L+1)
D = np.zeros(L+1)

for l in range(L+1):
    x = l*DELTA
    temp = sigma(x)*TAU / (2*eps(x))
    C[l] = (1 - temp) / (1 + temp)
    D[l] = TAU / eps(x) / (1 + temp)

A = np.zeros(L)
B = np.zeros(L)

for l in range(L):
    x = (l + 0.5) * DELTA
    temp = sigma(x)*TAU / (2*mu(x))
    A[l] = (1 - temp) / (1 + temp)
    B[l] = TAU / mu(x) / (1 + temp)

E = np.zeros(L+1) # E_z
H = np.zeros(L) # H_y

PLOT_T = [1000, 2500, 5000, 6000]

plt.rc('text', usetex=True)

for t in range(NUM_TIMESTEPS):
    E[1:L-1] = D[1:L-1] * (H[1:L-1] - H[0:L-2]) / DELTA + C[1:L-1] * E[1:L-1]
    E[SOURCE_POS_I] -= D[SOURCE_POS_I] * source(t*TAU)
    H[0:L-1] = B[0:L-1] * (E[1:L] - E[0:L-1]) / DELTA + A[0:L-1] * H[0:L-1]

    if t in PLOT_T:
        E_max = np.max(np.abs(E[:L/2]))
        print("E_max(t=" + str(t) + ") =", E_max)

        ax = plt.gca()
        plt.plot(E)
        plt.title("t = " + str(t))
        yRange = 0.016
        plt.ylim(-yRange, yRange)
        ax.fill_between([0, BOUNDARY_THICKNESS/DELTA], -yRange, yRange,
            facecolor='grey', alpha=0.5)
        ax.fill_between([(X-BOUNDARY_THICKNESS)/DELTA, X/DELTA], -yRange,
            yRange, facecolor='grey', alpha=0.5)
        ax.fill_between([X/2/DELTA, (X/2+GLASS_THICKNESS)/DELTA], -yRange,
            yRange, facecolor='green', alpha=0.5)
        plt.grid(True)
        plt.xlabel("$x / \Delta$")
        plt.ylabel("$E$")
        plt.savefig("t=" + str(t) + ",glass=" + str(math.floor(GLASS_THICKNESS/
            LAMBDA)) + ".pdf")
        #plt.show()
        plt.close()
```