Implementation of the Finite-Difference Time-Domain Method

Tymoteusz Wrzeszcz, Joshua Jordaan, Karolina Paciorek
Group number 1, Seminar B

June 25, 2020

Abbe School of Photonics
Friedrich-Schiller-Universität Jena

Contents

1 The Task

The task was to implement FDTD method in 1D and 3D case. This scheme is based on the rigorous solution of Maxwell's equations. Electric and magnetic fields are calculated for every time step in specified time interval in a loop. Resulting equations are derived directly from curl equation, and depending on dimensionality of the problem, some equations are kicked out or all field components are calculated. In 1D (no dependence on two chosen coordinates), divergence equation is always fullfiled since fields are transversely polarized in the direction of field change. 3D requires prior knowledge of solution to determine if fields are free of divergence. This imposes requirement of source implementation as a current density which is modelled in a way that it will produce desired field. Equations for H and E fields are solved on the so-called Yee grid, which is based on spatial and temporal translation of H field with respect to E field.

2 1D Implementation

For this case invariance in z and y is assumed, and all dynamics happen in x direction. Therefore two sets of equations can be solved, E_z and H_y or E_y and H_z. In the following solution first set was solved.

$$\begin{aligned} \frac{\partial E_z(x,t)}{\partial x} &= \mu_0 \frac{\partial H_y(x,t)}{\partial t} \\ \frac{\partial H_y(x,t)}{\partial x} &= \epsilon_0 \epsilon_r(x) \frac{\partial E_z(x,t)}{\partial t} + j_z(x,t) \end{aligned} \tag{1}$$

As a current source, pulsed source was used. It consists of delta-shaped spatial profile, and gaussian temporal envelope. The equation reads as:

$$j_z(t,x) = j_0 \delta(x - x_0) \exp -\frac{(t - t_0)^2}{\tau^2} \exp -2\pi i f n \Delta t \tag{2}$$

where $j_0 = 1\,\mathrm{A/m^2}$, $\tau = 1\,\mathrm{fs}$, $t_0 = 3\tau$, f is the source frequency, x_0 is the position in computational domain, and $n\Delta t$ is the next time step.
Perfect electric conducting boundaries were used as boundary conditions. Their presence will be shown in the implementation of equations as field values near borders require boundary values. Due to Yee grid discretization, indices of H field are fractional, and for computational purpose they have to be changed to integers. However, from the physical point of view, Yee grid implies half step shift of H field with respect to E field, and therefore H field has to be interpolated.

```
x = np.linspace(-x_span/2, x_span/2, Nx)
dt = dx / (2 * c)
t = np.arange(0, time_span, dt)
j_spat = np.zeros(Nx)
j_spat[Nx // 2 + int(source_position / dx)] = 1
time_part = np.exp(2 * np.pi * source_frequency * t / complex(0, 1)) * np.exp(-(t -
    3 * source_pulse_length)**2 / (source_pulse_length**2))
Ez = np.empty((len(t), Nx), dtype = 'complex')
Hy = np.empty((len(t), Nx), dtype = 'complex')
Ez[0, :] = 0
Hy[0, :] = 0
```

```
n = 0
while (n < len(t) - 1):
    Ez[n+1, 1:Nx] = Ez[n, 1:Nx] + (dt / (eps0 * eps_rel[1:Nx] * dx)) * (Hy[n, 1:Nx]
        - Hy[n, 0:Nx-1]) - ((dt * j_spat[1:Nx] * time_part[n]) / (eps0 *
        eps_rel[1:Nx]))
    Ez[n+1, 0] = Ez[n, 0] + (dt / (eps0 * eps_rel[0] * dx)) * Hy[n, 0] - ((dt *
        j_spat[0] * time_part[n]) / (eps0 * eps_rel[0]))
    Hy[n+1, 0:Nx-1] = Hy[n, 0:Nx-1] + (dt / (mu0 * dx)) * (Ez[n+1, 1:Nx] - Ez[n+1,
        0:Nx-1])
    Hy[n+1, -1] = Hy[n, -1] + (dt / (mu0 * dx)) * (-Ez[n+1, -1])
    n+=1
for i in np.arange(1, Nx):
    Hy[:, i] = 0.5 * (Hy[:, i - 1] + Hy[:, i])
return Ez, Hy, x, t
```

Firstly temporal and spatial part of the current source is created. Position of delta peak in spatial part can be set to multiplicity of step size Δx. Then initial fields at time $t = 0$ are set to 0. Fields are calculated in the *while* loop in the specified time interval. Electric field E_z in next time step is calculated first. With knowledge of its value, H_y in next time step can be obtained. Two equations in this loop corresponds only to PEC boundary conditions, where components "outside" boundaries are missing. That is why they are not taken into account during calculations of boundary values. At the end H_y field is interpolated to kick out half step shift with respect to E_z.

3 Convergence tests

Due to the 3D FDTD method requiring much longer computation times and the fact both methods use the same central difference discretization scheme, here we assume accuracy and convergence results obtained in the 1D case will also apply in the 3D.
Accuracy and convergence were tested with varying spatial step size Δx. As to ensure stability of the FDTD method $\Delta t \leq \Delta x/(2c)$ then time step Δt, and spatial step Δx are coupled, so changing Δx, changes also Δt. Root mean square error was used to calculate relative error. The equation reads as:

$$RMSE = \sqrt{\frac{\sum(v_{ref} - v)^2}{N x_{ref}}} \tag{3}$$

Two versions of this computation were implemented. First one takes field profile at last time step for each Δx. As a reference, field distribution for smallest Δx was used. For every Δx other than the reference one, fields had to be interpolated to reference grid. Second implementation takes only middle field values at position $x = 0$ for every time step. As a reference, again field values for smallest Δx were used (smallest Δt, more time steps). Because of this, fields had to be interpolated in time to reference grid.
Step size Δx is varied on the logarithmic scale from 3 nm to 50 nm with 40 steps. Additionally resulting data (log-log representation) was fitted using power law function which reads as ax^b.

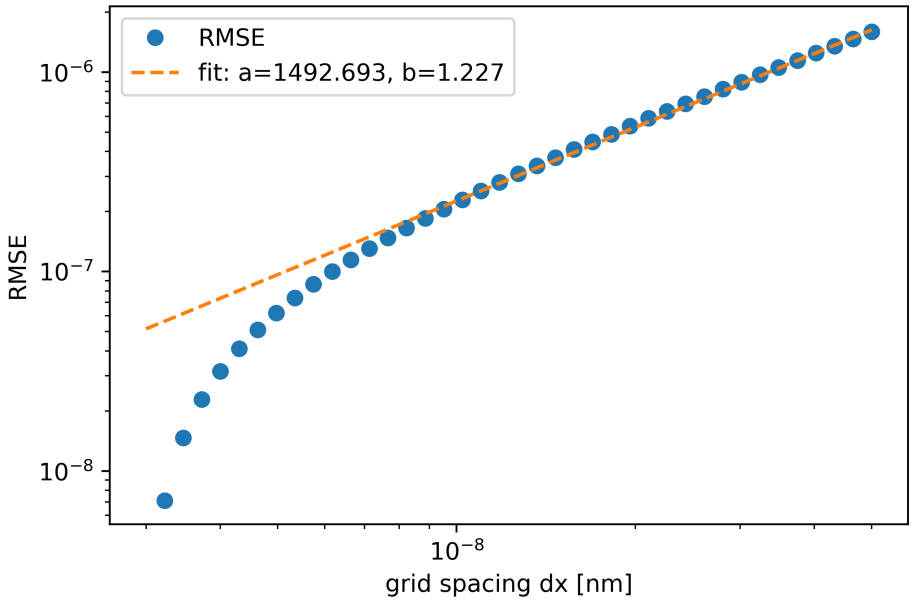


Figure 1: Convergence in dependence of Δx. Error calculated using field distributions for last time step.

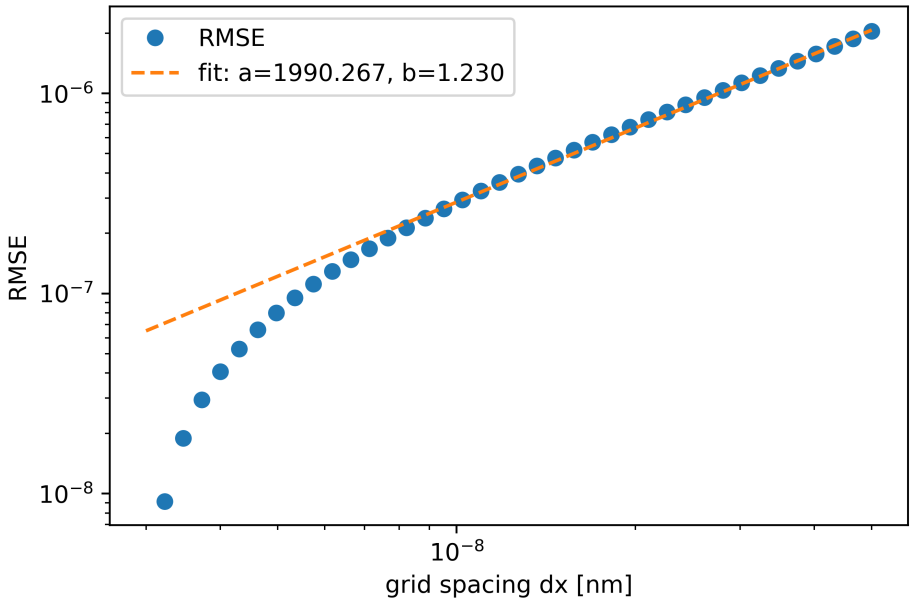


Figure 2: Convergence in dependence of Δx. Error calculated using field values at the center of the grid for each time step.

As it can be seen, Figure 1 and Figure 2 are almost the same. Expected quadratic convergence is clearly visible for small steps sizes Δx. This is an effect of chosen finite-difference algorithm used for derivatives approximation. Otherwise error scales linearly with increasing Δx. Step size $\Delta x = 15\,\text{nm}$ suggested for testing the implementation is therefore too large. Since the

computational time is not much longer for $\Delta x = 3\,\text{nm}$, which is the smallest step size used here, it should be used for simulations. The error is then roughly 50 times smaller.

4 1D results

Source parameters used for testing: $f = 500\,\text{THz}$, $\tau = 1\,\text{fs}$, $t_0 = 3\tau$, with source located at the center of the computation domain.
Computational parameters: spatial window size $W = 18\,\mu\text{m}$, step size $\Delta x = 15\,\text{nm}$, simulation time span $T = 60\,\text{fs}$, time step $\Delta t = \frac{\Delta x}{2c} = 0.025\,\text{fs}$. Relative permittivity distribution can be arbitrary. Firstly simulation was done using only dispersion-free medium with $\epsilon = 1$, and then another material with $\epsilon = 4$ was added at position $x = 4.5\,\mu\text{m}$.

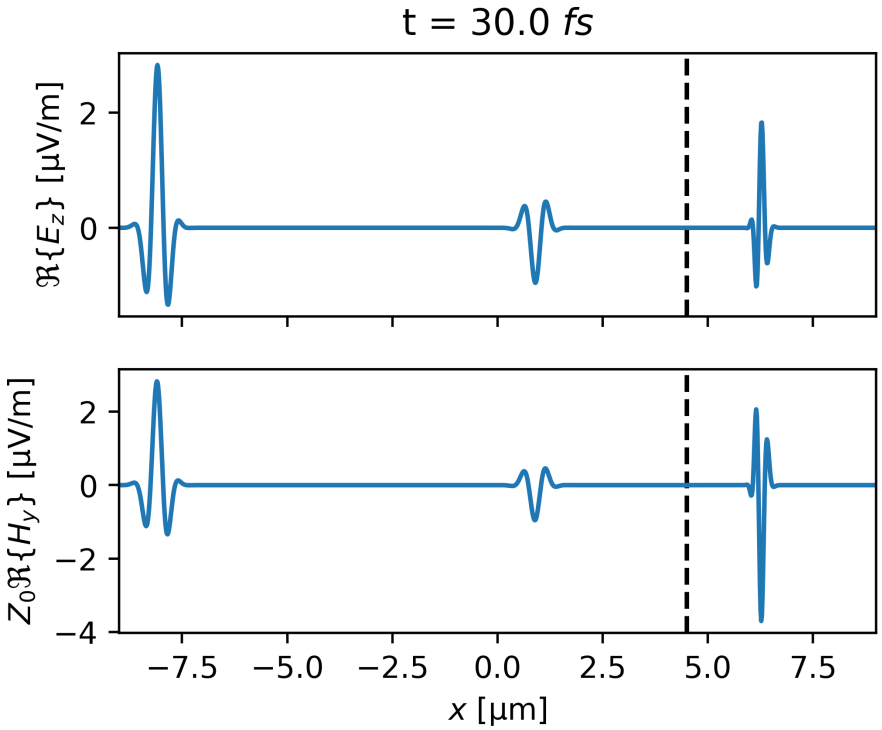
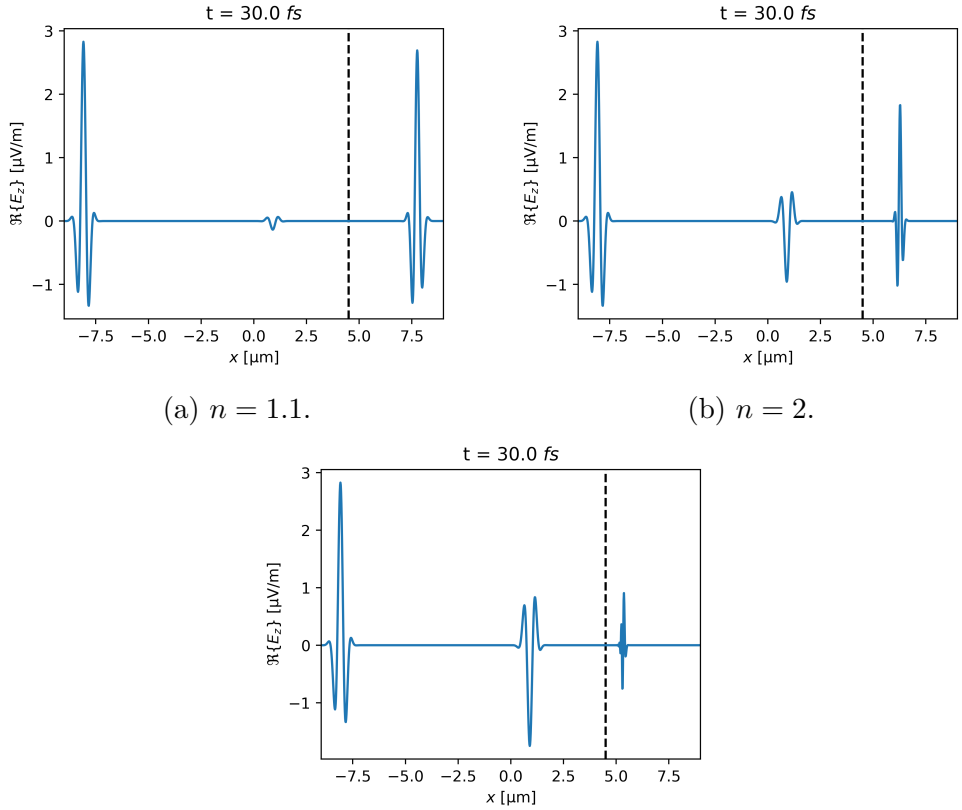


Figure 3: Distribution of E_z and H_y after 30 fs from the start of simulation.

Initially two pulses are generated in both left and right side from the origin $x = 0$. Field distribution in the center on Figure 3 corresponds to the initial field that was reflected from the interface between both media, and therefore its amplitude has decreased. There is also a transmitted part of this pulse which propagates in second material. It can be seen that in the higher refractive index material, pulse spatial width is smaller. Field profile on the left is just the initial distribution that was reflected from the PEC boundary without any changes.

When refractive index of added material changes, some interesting effects can be seen. As it is shown on Figure 4 pulse propagates slower in higher refractive index material (position inside material is closer to the interface). As mentioned before, also spatial width of pulse decreases with higher refractive index. Additionally with higher refractive index more light is reflected than transmitted.



(a) $n = 1.1$. (b) $n = 2$.

(c) $n = 4$.

Figure 4: Distribution of Ez for different refractive indices of the added material.

Figure 5 shows simulation in dispersion-free medium with $\epsilon = 1$. Source pulse is made wider to show interference that happens between two pulses reflected from boundaries. When both pulses start to overlap, their amplitude grows significantly.

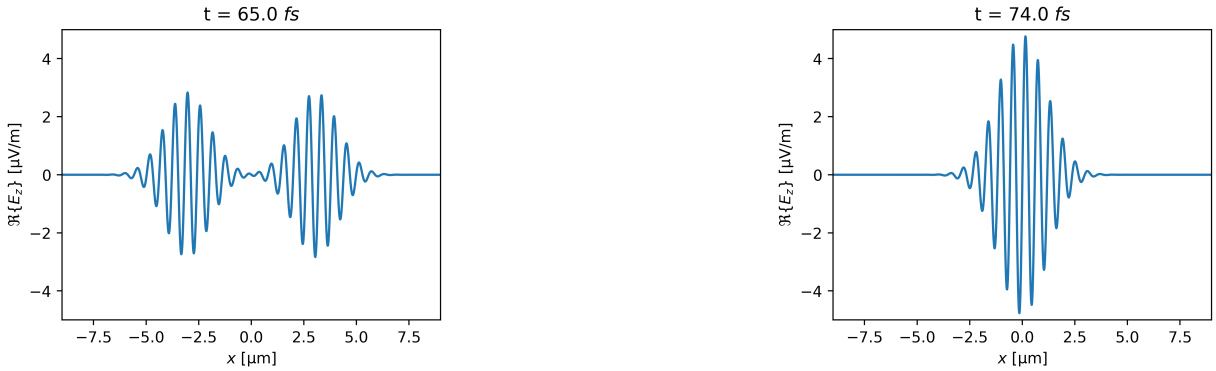


Figure 5: Distribution of Ez for different time steps. source pulse length $\tau = 5\,\text{fs}$, simulation time span $T = 90\,\text{fs}$.

5 3D Implementation

In 3D the FDTD method is a discretization on the 3D Yee Grid of the following Maxwell's Equations

$$\frac{\partial \boldsymbol{E}(\boldsymbol{r},t)}{\partial t} = \frac{1}{\epsilon_0 \epsilon(\boldsymbol{r})} \nabla \times \boldsymbol{H}(\boldsymbol{r},t) - \boldsymbol{j}(\boldsymbol{r},t) \tag{4}$$

$$\frac{\partial \boldsymbol{H}(\boldsymbol{r},t)}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}(\boldsymbol{r},t). \tag{5}$$

Performing the discretization using a central difference scheme in both space and time leads to the following explicit update functions for the H anf E fields where n is the temporal index and i, j, k the usual spacial indices:

$$E_x\big|_{i,j,k}^{n+1} = E_x\big|_{i,j,k}^{n} + \frac{\Delta t}{\epsilon_0 \epsilon_x\big|_{i,j,k}} \left(\frac{H_z\big|_{i,j,k}^{n} - H_z\big|_{i,j-1,k}^{n}}{\Delta y} - \frac{H_y\big|_{i,j,k}^{n} - H_y\big|_{i,j,k-1}^{n}}{\Delta z} - j_x\big|_{i,j,k}^{n} \right) \tag{6}$$

$$E_y\big|_{i,j,k}^{n+1} = E_y\big|_{i,j,k}^{n} + \frac{\Delta t}{\epsilon_0 \epsilon_y\big|_{i,j,k}} \left(\frac{H_x\big|_{i,j,k}^{n} - H_x\big|_{i,j,k-1}^{n}}{\Delta z} - \frac{H_z\big|_{i,j,k}^{n} - H_z\big|_{i-1,j,k}^{n}}{\Delta x} - j_y\big|_{i,j,k}^{n} \right) \tag{7}$$

$$E_z\big|_{i,j,k}^{n+1} = E_z\big|_{i,j,k}^{n} + \frac{\Delta t}{\epsilon_0 \epsilon_z\big|_{i,j,k}} \left(\frac{H_y\big|_{i,j,k}^{n} - H_y\big|_{i-1,j,k}^{n}}{\Delta z} - \frac{H_x\big|_{i,j,k}^{n} - H_x\big|_{i,j-1,k}^{n}}{\Delta x} - j_z\big|_{i,j,k}^{n} \right) \tag{8}$$

$$H_x\big|_{i,j,k}^{n+1} = H_x\big|_{i,j,k}^{n} + \frac{\Delta t}{\mu_0} \left(\frac{E_y\big|_{i,j,k+1}^{n+1} - E_y\big|_{i,j,k}^{n+1}}{\Delta z} - \frac{E_z\big|_{i,j+1,k}^{n+1} - E_z\big|_{i,j,k}^{n+1}}{\Delta y} \right) \tag{9}$$

$$H_y\big|_{i,j,k}^{n+1} = H_y\big|_{i,j,k}^{n} + \frac{\Delta t}{\mu_0} \left(\frac{E_z\big|_{i+1,j,k}^{n+1} - E_z\big|_{i,j,k}^{n+1}}{\Delta x} - \frac{E_x\big|_{i,j+1,k}^{n+1} - E_x\big|_{i,j,k}^{n+1}}{\Delta z} \right) \tag{10}$$

$$H_z\big|_{i,j,k}^{n+1} = H_z\big|_{i,j,k}^{n} + \frac{\Delta t}{\mu_0} \left(\frac{E_x\big|_{i,j+1,k}^{n+1} - E_x\big|_{i,j,k}^{n+1}}{\Delta y} - \frac{E_y\big|_{i+1,j,k}^{n+1} - E_y\big|_{i,j,k}^{n+1}}{\Delta x} \right). \tag{11}$$

$$\tag{12}$$

Here it is important to note that due to the Yee Grid structure that tangential E-fields and normal H-fields are stored at integer Yee Grid indices and have N grid points where N is the size along the appropriate dimension of grid that $\epsilon(\boldsymbol{r})$ and $\boldsymbol{j}(\boldsymbol{r},t)$ are defined on, and the tangential H-fields and normal E-fields are stored at fractional indices and have $N-1$ points. We also implement PEC boundaries such that the tangential E-fields and the normal H-fields are set to zero and are not updated. Furthermore due to the fields in the Yee Grid being staggered in space and time, to report them in the original coordinate system of $\epsilon(\boldsymbol{r})$ and $\boldsymbol{j}(\boldsymbol{r},t)$ they need to be interpolated, where the we use the fact that at the PEC boudaries the normal components of the E-field and the tangential components of the H-field are duplicated exactly.
The main strategy used when implementing the 3D FDTD scheme, due to the update functions complexity, was to iterate over the arrays in the most straightforward way possible using inefficient for loops without vectorization and then heavily rely on the just-in-time compiled functionality of the *numba* python package to speed things up. The update functions of the E and H fields as well as the source are given below.

```
#Setting up field arrays
(Nx,Ny,Nz) = eps_rel.shape
Ex = np.zeros((Nx-1,Ny,Nz),dtype=complex)
Ey = np.zeros((Nx,Ny-1,Nz),dtype=complex)
Ez = np.zeros((Nx,Ny,Nz-1),dtype=complex)
Hx = np.zeros((Nx,Ny-1,Nz-1),dtype=complex)
Hy = np.zeros((Nx-1,Ny,Nz-1),dtype=complex)
Hz = np.zeros((Nx-1,Ny-1,Nz),dtype=complex)

@jit(nopython=True) #just in time compile to speed up for loops
#iterating through field arrays being careful of indices to avoid PEC boundary
    update and grid size
def Update_E(Ex,Ey,Ez,Hx,Hy,Hz,jx,jy,jz,eps_rel,dr,dt,Nx,Ny,Nz,eps0):
    Ex = Ex.copy(); Ey = Ey.copy(); Ez = Ez.copy();
    for i in range(Nx-2):
        for j in range(1,Ny-2):
            for k in range(1,Nz-2):
            #quantites that need to be interpolated are done so at every step
                inv_eps = (0.5/eps_rel[i,j,k] + 0.5/eps_rel[i+1,j,k])/eps0
                jx_intp = 0.5*(jx[i,j,k]+jx[i+1,j,k])
                Ex[i,j,k] = Ex[i,j,k] + inv_eps*dt*((Hz[i,j,k] - Hz[i,j-1,k])/dr -
                    (Hy[i,j,k] - Hy[i,j,k-1])/dr - jx_intp)

    for i in range(1,Nx-2):
        for j in range(Ny-2):
            for k in range(1,Nz-2):
                inv_eps = (0.5/eps_rel[i,j,k] + 0.5/eps_rel[i,j+1,k])/eps0
                jy_intp = 0.5*(jy[i,j,k]+jy[i,j+1,k])
                Ey[i,j,k] = Ey[i,j,k] + inv_eps*dt*((Hx[i,j,k] - Hx[i,j,k-1])/dr -
                    (Hz[i,j,k] - Hz[i-1,j,k])/dr - jy_intp)

    for i in range(1,Nx-2):
        for j in range(1,Ny-2):
            for k in range(Nz-2):
                inv_eps = (0.5/eps_rel[i,j,k] + 0.5/eps_rel[i,j,k+1])/eps0
                jz_intp = 0.5*(jz[i,j,k]+jz[i,j,k+1])
                Ez[i,j,k] = Ez[i,j,k] + inv_eps*dt*((Hy[i,j,k] - Hy[i-1,j,k])/dr -
                    (Hx[i,j,k] - Hx[i,j-1,k])/dr - jz_intp)

    return Ex,Ey,Ez

@jit(nopython=True)
def Update_H(Ex,Ey,Ez,Hx,Hy,Hz,dr,dt,Nx,Ny,Nz,mu0):
    Hx = Hx.copy(); Hy = Hy.copy(); Hz = Hz.copy();
    for i in range(1,Nx-2):
        for j in range(Ny-2):
            for k in range(Nz-2):
                Hx[i,j,k] = Hx[i,j,k] + (dt/mu0)*(Ey[i,j,k+1] - Ey[i,j,k] -
                    Ez[i,j+1,k] + Ez[i,j,k])/dr

    for i in range(Nx-2):
```

```
        for j in range(1,Ny-2):
            for k in range(Nz-2):
                Hy[i,j,k] = Hy[i,j,k] + (dt/mu0)*(Ez[i+1,j,k] - Ez[i,j,k] -
                    Ex[i,j,k+1] + Ex[i,j,k])/dr

    for i in range(Nx-2):
        for j in range(Ny-2):
            for k in range(1,Nz-2):
                Hz[i,j,k] = Hz[i,j,k] + (dt/mu0)*(Ex[i,j+1,k] - Ex[i,j,k] -
                    Ey[i+1,j,k] + Ey[i,j,k])/dr

    return Hx,Hy,Hz

@jit(nopython=True,parallel=True)
def Update_source(jx,jy,jz,freq,tau,n,dt):
    jx = jx.copy(); jy = jy.copy(); jz = jz.copy();
    tt = (n+0.5)*dt
    t0 = 3*tau
    jz = np.exp(-2j*np.pi*freq*tt)*np.exp(-(tt-t0)**2/tau**2)*jz
    return jx,jy,jz
```

These functions are then used in the main loop where over the time range specified with time steps of size $\Delta t = \Delta x/(2c)$. The order of operations is that the source is updated, then the E-field is updated, and then the H-field. At the desired output steps the field it interpolated and saved as an output. The main loop was also compiled with *numba*:

```
# constants
c = 2.99792458e8 # speed of light [m/s]
mu0 = 4*np.pi*1e-7 # vacuum permeability [Vs/(Am)]
eps0 = 1/(mu0*c**2) # vacuum permittivity [As/(Vm)]

#Set up time/output arrays
dt = dr/(2*c)
tstep = np.arange(0,time_span+dt,dt)
t = np.array([])
F = np.array([],dtype=complex)

@jit(nopython=True,parallel=True)
def main_loop(Ex,Ey,Ez,Hx,Hy,Hz,jx,jy,jz,eps_rel,dr,dt,Nx,Ny,Nz,eps0,mu0,F,t):
    for n in range(tstep.size):
        if n == 0:
            Exn = Ex; Eyn = Ey; Ezn = Ez;
            Hxn = Hx; Hyn = Hy; Hzn = Hz;

        #Defining previous H to use in interpolate
        Hxp = Hxn.copy(); Hyp = Hyn.copy(); Hzp = Hzn.copy();

        #Updating fields and iterating
        jxn,jyn,jzn = Update_source(jx,jy,jz,freq,tau,n,dt)
        Exn,Eyn,Ezn =
            Update_E(Exn,Eyn,Ezn,Hxn,Hyn,Hzn,jxn,jyn,jzn,eps_rel,dr,dt,Nx,Ny,Nz,eps0)
```

```
        Hxn,Hyn,Hzn = Update_H(Exn,Eyn,Ezn,Hxn,Hyn,Hzn,dr,dt,Nx,Ny,Nz,mu0)

        #Savig output field when needed
        if n%output_step == 0:
            Outf = np.zeros((Nx,Ny,Nz),dtype=np.complex128)
            Outf = Interpol_out(Outf,field_component,Exn,Eyn,Ezn,Hxn,Hyn,
                Hzn,Hxp,Hyp,Hzp,Nx,Ny,Nz)
            F = np.append(F.flatten(),Outf[:,:,z_ind].flatten())
            t = np.append(t,tstep[n])
            del Outf

    F = np.reshape(F,(-1,Nx,Ny))
    return F, t
```

The output interpolation function was defined as:

```
@jit(nopython=True,parallel=True)
def Interpol_out(Outf,field_component,Ex,Ey,Ez,Hx,Hy,Hz,Hxp,Hyp,Hzp,Nx,Ny,Nz):
    fc = field_component
    if fc == 'ex':
    #first interpolating manually at boundary and then looping over other values
        Outf[0,:,:] = Ex[0,:,:]
        Outf[-1,:,:] = Ex[-1,:,:]
        for i in range(1,Nx-1):
            Outf[i,:,:] = 0.5*(Ex[i-1,:,:] + Ex[i,:,:])
    if fc == 'ey':
        Outf[:,0,:] = Ey[:,0,:]
        Outf[:,-1,:] = Ey[:,-1,:]
        for j in range(1,Ny-1):
            Outf[:,j,:] = 0.5*(Ey[:,j-1,:] + Ey[:,j,:])
    if fc == 'ez':
        Outf[:,:,0] = Ez[:,:,0]
        Outf[:,:,-1] = Ez[:,:,-1]
        for k in range(1,Nz-1):
            Outf[:,:,k] = 0.5*(Ez[:,:,k-1] + Ez[:,:,k-1])
    if fc == 'hx':
        Outf[:,0,0] = (1/8)*(4*Hx[:,0,0]+4*Hxp[:,0,0])
        Outf[:,-1,-1] = (1/8)*(4*Hx[:,-1,-1]+4*Hxp[:,-1,-1])
        for j in range(1,Ny-1):
            for k in range(1,Nz-1):
                Outf[:,j,k] = (1/8)*(Hx[:,j-1,k-1] + Hx[:,j-1,k] + Hx[:,j,k-1] +
                    Hx[:,j,k] + Hxp[:,j-1,k-1] + Hxp[:,j-1,k] + Hxp[:,j,k-1] +
                    Hxp[:,j,k])
    if fc == 'hy':
        Outf[0,:,0] = (1/8)*(4*Hy[0,:,0]+4*Hyp[0,:,0])
        Outf[-1,:,-1] = (1/8)*(4*Hy[-1,:,-1]+4*Hyp[-1,:,-1])
        for i in range(1,Nx-1):
            for k in range(1,Nz-1):
                Outf[i,:,k] = (1/8)*(Hy[i-1,:,k-1] + Hy[i-1,:,k] + Hy[i,:,k-1] +
                    Hy[i,:,k] + Hyp[i-1,:,k-1] + Hyp[i-1,:,k] + Hyp[i,:,k-1] +
                    Hyp[i,:,k])
```

```
    if fc == 'hz':
        Outf[0,0,:] = (1/8)*(4*Hz[0,0,:]+4*Hzp[0,0,:])
        Outf[-1,-1,:] = (1/8)*(4*Hz[-1,-1,:]+4*Hzp[-1,-1,:])
        for i in range(1,Nx-1):
            for j in range(1,Ny-1):
                Outf[i,j,:] = (1/8)*(Hz[i-1,j-1,:] + Hz[i,j-1,:] + Hz[i-1,j,:] +
                    Hz[i,j,:] + Hzp[i-1,j-1,:] + Hzp[i,j-1,:] + Hzp[i-1,j,:] +
                    Hzp[i,j,:])

    return Outf
```

6 3D Results

For the test case the source distribution modelled was given as

$$\boldsymbol{j}(\boldsymbol{r},t) = j_0 \exp(-2\pi f t) \exp\left(-\frac{(t-t_0)^2}{\tau^2}\right) \exp\left(-\frac{x^2+y^2}{w^2}\right) \boldsymbol{e}_z, \tag{13}$$

where $j_0 = 1\,\mathrm{A/m^2}$, $f = 500\,\mathrm{THz}$, $\tau = 1\,\mathrm{fs}$, $t_0 = 3\tau$ and $w = 2\Delta x$. Which was modelled over a grid with size in points of 199X201X5, a step size in all dimensions of 30 nm and total time span of 10 fs. The results of this test case are shown below in Figure 6.

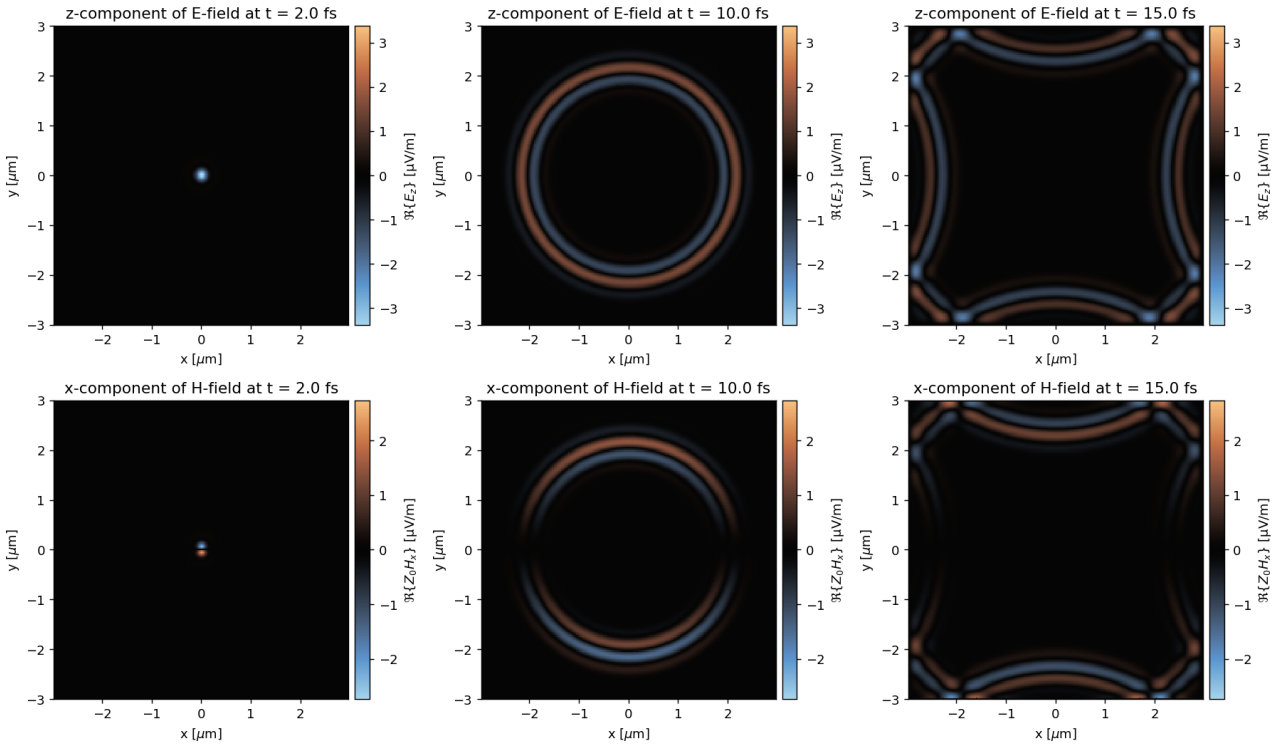


Figure 6: Evolution of the E_z and H_x field components at various time slices. The time span has been extended by 5 fs to view the interaction with the PEC boundary.

From Figure 6 we can see that the oscillating line current in the z direction emits a field which is precisely described by a dipole radiation pattern. Characteristically the z-component of the

created electric field has a constant magnitude for a particular z-slice of the field, and the magnetic field x-component is maximal orthogonal to the y-axis, zero parallel to the y-axis and undergoes a sign change on either side which corresponds to a circularly polarised H-field related to the dipole oscillation by the right hand rule. We can also see that the PEC boundaries are behaving as expected by providing 100% reflection and the π phase change.
For the sake of interest the field propagation through a disk of high refractive index material ($\epsilon = 10$) was also investigated. The results of which are in Figure 7. Here all of the same parameters as before were used accept for the alteration of the ϵ distribution to account for the disk material. From the plots the focusing effect of the disk is clear.

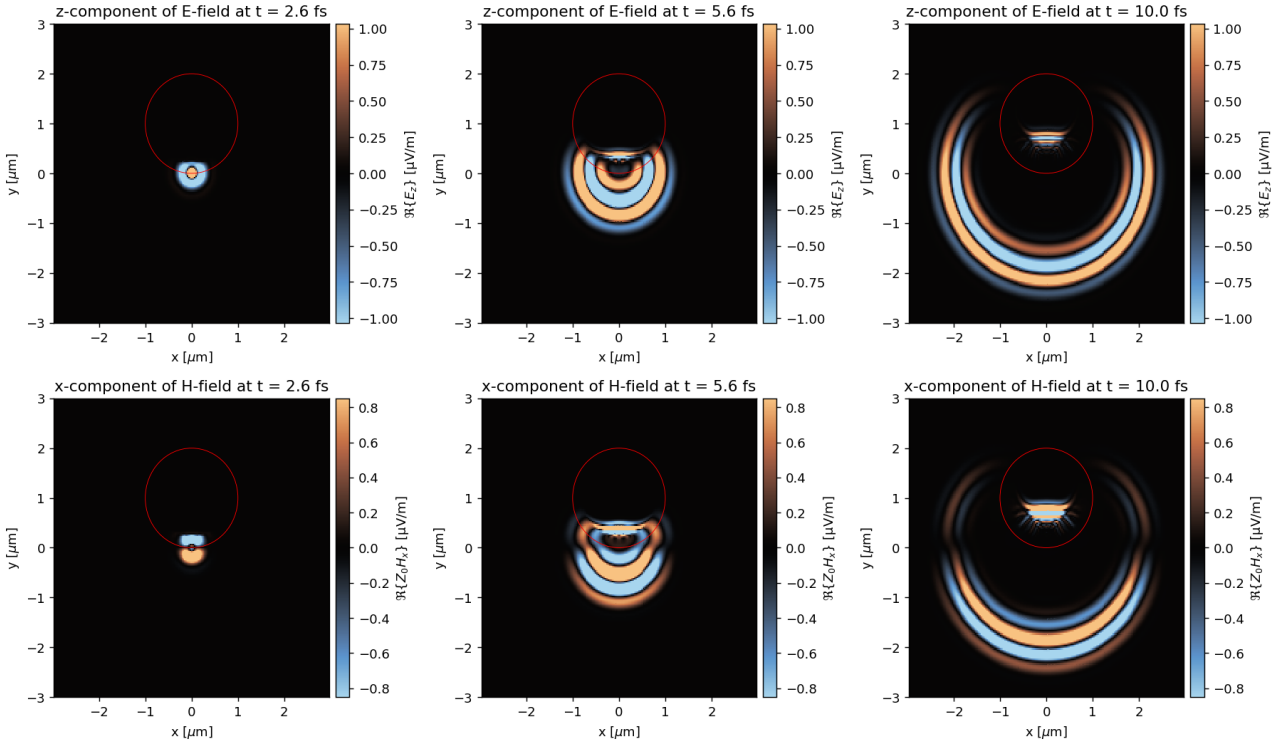


Figure 7: Evolution of the E_z and H_x field components as they propagate through disk of $\epsilon = 10$ material. Disk outline shown in red.