Brussels Faculty of Enginering

Université Libre de Bruxelles

Vrije Universiteit van Brussel

Academic Year 2016-2017

Project Report

Numerical resolution of the Maxwell equations using the Finite-Difference Time-Domain method

Victor Artois, Cédric Hannotier, Mathieu Petitjean

ELEC-H415: Communication Channels

Contents

1 Introduction

This report presents the derivation and implementation of a two dimensional finite-difference time-domain (FDTD) algorithm for the resolution of the Maxwell equations.

First, the Maxwell equations are discretized in order to obtain implementable update equations for the fields. Then advanced boundary conditions are discussed in order to avoid reflections on the borders of the grid. Finally, the C source codes are shown, their functioning is detailed and a validation procedure is carried out.

These codes were used to illustrate the main concepts of electromagnetism presented in the Communication Channels course at the Brussels Faculty of Engineering. The derivation of the equations as well as the codes are based on the works of John Schneider[1].

2 2D FDTD algorithm

The algorithm for the two-dimensional FDTD method is derived under the assumption of a TM^z polarization (Transverse Magnetic, the magnetic field is transverse to the direction of propagation z). The Faraday and Ampere laws can be written:

$$\begin{cases} -\sigma_m \vec{H} - \mu \dfrac{\partial \vec{H}}{\partial t} = \nabla \times \vec{E} \\ \sigma \vec{E} + \varepsilon \dfrac{\partial \vec{E}}{\partial t} = \nabla \times \vec{H} \end{cases} \tag{1}$$

Using the fact that the only non-zero elements are E_z, H_x and H_y, (1) can be rewritten in the set of scalar equations:

$$\begin{cases} -\sigma_m H_x - \mu \dfrac{\partial H_x}{\partial t} = \dfrac{\partial E_z}{\partial y} \\ \sigma_m H_y + \mu \dfrac{\partial H_y}{\partial t} = \dfrac{\partial E_z}{\partial x} \\ \sigma E_z + \varepsilon \dfrac{\partial E_z}{\partial t} = \dfrac{\partial H_y}{\partial x} - \dfrac{\partial H_x}{\partial y} \end{cases} \tag{2}$$

These equations now need to be discretized in order to be solved numerically. First, the fields are discretized in space and time. a corresponds to the spatial step in the x direction, b to the spatial step in the y direction and n corresponds to the temporal step.

$$\begin{aligned} H_x(x,y,t) &= H_x(a\Delta x, b\Delta y, n\Delta t) \equiv H_x^n[a,b] \\ H_y(x,y,t) &= H_y(a\Delta x, b\Delta y, n\Delta t) \equiv H_y^n[a,b] \\ E_z(x,y,t) &= E_z(a\Delta x, b\Delta y, n\Delta t) \equiv E_z^n[a,b] \end{aligned} \tag{3}$$

The derivative of (2) will then be approximated by finite differences. By neglecting second order terms in the step δ, one can find:

$$f'(x_0) = \frac{f\left(x_0 + \frac{\delta}{2}\right) - f\left(x_0 - \frac{\delta}{2}\right)}{\delta} + \mathcal{O}(\delta^2) \simeq \frac{f\left(x_0 + \frac{\delta}{2}\right) - f\left(x_0 - \frac{\delta}{2}\right)}{\delta} \tag{4}$$

[1] Understanding the FDTD method: `http://www.eecs.wsu.edu/~schneidj/ufdtd/`

In order to use (4) to obtain the update equations of the field, it is necessary to include an offset between the electric and magnetic fields. The assumption is done that the electric field nodes are located on integer values of the steps while the magnetic field nodes have offsets of half a step, as shown in Figure 1.

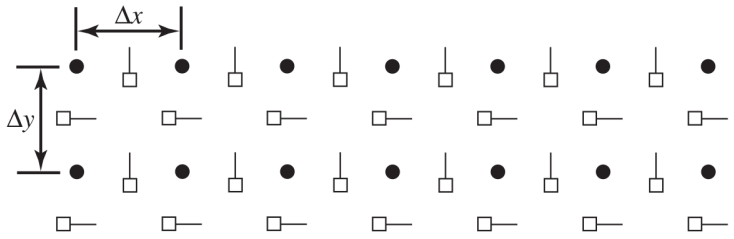

Figure 1: Physical setup of the FDTD grid. The circles represent the electric field nodes E while the squares represent the magnetic field nodes H. An offset of half a spatial step is present[1].

With this particular arrangement of the nodes, the first equation of (2) can be expanded around the point $(x, y, t) = (a\Delta x, (b + \frac{1}{2})\Delta y, n\Delta t)$. Each term of the equation is detailed:

$$\begin{aligned} -\sigma_m H_x^n \left[a, b + \frac{1}{2}\right] &\simeq -\sigma_m \frac{H_x^{n+\frac{1}{2}} \left[a, b + \frac{1}{2}\right] + H_x^{n-\frac{1}{2}} \left[a, b + \frac{1}{2}\right]}{2} \\ -\mu \frac{\partial H_x}{\partial t} &\simeq -\mu \frac{H_x^{n+\frac{1}{2}} \left[a, b + \frac{1}{2}\right] - H_x^{n-\frac{1}{2}} \left[a, b + \frac{1}{2}\right]}{\Delta t} \\ \frac{\partial E_z}{\partial y} &\simeq \frac{E_z^n \left[a, b + 1\right] - E_z^n [a, b]}{\Delta y} \end{aligned} \tag{5}$$

In the approximated equation, the future term (H_x at time step $n+\frac{1}{2}$) can be obtained from the past ones. Once isolated, it is expressed as:

$$H_x^{n+\frac{1}{2}} \left[a, b + \frac{1}{2}\right] = \frac{1 - \frac{\sigma_m \Delta t}{2\mu}}{1 + \frac{\sigma_m \Delta t}{2\mu}} H_x^{n-\frac{1}{2}} \left[a, b + \frac{1}{2}\right] - \frac{1}{1 + \frac{\sigma_m \Delta t}{2\mu}} \frac{\Delta t}{\mu \Delta y} (E_z^n [a, b+1] - E_z^n [a, b]) \tag{6}$$

In (6), the parameters μ and σ_m are the value of the material at the space point $(x, y) = (a\Delta x, b + \frac{1}{2}\Delta y)$. Following the same reasoning, the update equations for H_y and E_z can be found:

$$H_y^{n+\frac{1}{2}} \left[a + \frac{1}{2}, b\right] = \frac{1 - \frac{\sigma_m \Delta t}{2\mu}}{1 + \frac{\sigma_m \Delta t}{2\mu}} H_y^{n-\frac{1}{2}} \left[a + \frac{1}{2}, b\right] + \frac{1}{1 + \frac{\sigma_m \Delta t}{2\mu}} \frac{\Delta t}{\mu \Delta x} (E_z^n [a+1, b] - E_z^n [a, b]) \tag{7}$$

$$E_z^{n+1}[a,b] = \frac{1-\frac{\sigma\Delta t}{2\varepsilon}}{1+\frac{\sigma\Delta t}{2\varepsilon}} E_z^n[a,b] + \frac{1}{1+\frac{\sigma\Delta t}{2\varepsilon}} \left(\frac{\Delta t}{\varepsilon\Delta x} \left\{ H_y^{n+\frac{1}{2}} \left[a+\frac{1}{2},b\right] - H_y^{n+\frac{1}{2}} \left[a-\frac{1}{2},b\right] \right\} - \frac{\Delta t}{\varepsilon\Delta x} \left\{ H_x^{n+\frac{1}{2}} \left[a,b+\frac{1}{2}\right] - H_x^{n+\frac{1}{2}} \left[a,b-\frac{1}{2}\right] \right\} \right) \tag{8}$$

3 Absorbing boundary conditions

When implementing the previous update equation in a program, one needs to investigate the value assigned to the extreme values of the grid, because these cannot be updated through the update equations. For example, when computing $E_z^{n+1}[0,0]$, the value $H_y^{n+\frac{1}{2}}\left[0,-\frac{1}{2}\right]$ is not available.

One basic approach is to impose the electric field to be zero on the boundaries of the grid. Physically, a region where the electric field is always zero is a perfect electric conductor. Any incoming electromagnetic wave on such a conductor will be totally reflected. This can be a wanted behaviour in some simulations, but absorbing boundaries must be implemented in order to simulate an infinite environment.

Since the update equations cannot be used, the following reasoning is based on the wave equation that governs the propagation of the electric field:

$$\left(\frac{\partial^2}{\partial x^2} - \mu\varepsilon\frac{\partial^2}{\partial t^2}\right) E_z = \left(\frac{\partial}{\partial x} - \sqrt{\mu\varepsilon}\frac{\partial}{\partial t}\right)\left(\frac{\partial}{\partial x} + \sqrt{\mu\varepsilon}\frac{\partial}{\partial t}\right) E_z = 0 \tag{9}$$

This equation will be discretized around the point $(x,t) = \left(\frac{\Delta x}{2}, (n+\frac{1}{2})\Delta t\right)$. The electrical field was defined on integer values of spatial indexes, so that $E_z^n\left[\frac{1}{2}\right]$ will be approximated by $(E_z^n[0] + E_z^n[1])/2$. In the following equations, a compact notation has been introduced for readability. I denotes the identity operator, σ_t is a time shift and σ_x is a spatial shift.

$$\left.\frac{\partial E_z}{\partial t}\right|_{\left(\frac{\Delta x}{2},(n+\frac{1}{2})\Delta t\right)} = \frac{\frac{E_z^{n+1}[0]+E_z^{n+1}[1]}{2} - \frac{E_z^n[0]+E_z^n[1]}{2}}{\Delta t} = \left(\frac{I-\sigma_t^{-1}}{\Delta t}\right)\left(\frac{I+\sigma_x^1}{2}\right) E_z^{n+1}[0] \tag{10}$$

$$\left.\frac{\partial E_z}{\partial x}\right|_{\left(\frac{\Delta x}{2},(n+\frac{1}{2})\Delta t\right)} = \frac{\frac{E_z^{n+1}[1]+E_z^n[1]}{2} - \frac{E_z^{n+1}[0]+E_z^n[0]}{2}}{\Delta x} = \left(\frac{\sigma_x^1-I}{\Delta x}\right)\left(\frac{I+\sigma_t^{-1}}{2}\right) E_z^{n+1}[0] \tag{11}$$

The wave equation can thus be rewritten in operator form, and the advection operator $\boldsymbol{A}$ is defined.

$$\left\{\left(\frac{\sigma_x^1-I}{\Delta x}\right)\left(\frac{I+\sigma_t^{-1}}{2}\right) - \sqrt{\mu\varepsilon}\left(\frac{I-\sigma_t^{-1}}{\Delta t}\right)\left(\frac{I+\sigma_x^1}{2}\right)\right\} E_z^{n+1}[0] = \boldsymbol{A}E_z^{n+1}[0] = 0 \tag{12}$$

The update equation provided by (12) is approximate. Better results are obtained when using a second order equation, which means that the advection operator is applied twice. This way, a long but often accurate update equation can be deduced.

$$\boldsymbol{A}\boldsymbol{A}E_z^{n+1} = 0 \tag{13}$$

$$\begin{aligned} E_z^{n+1}[0] = & \frac{-1}{1/S_c^{'} + 2 + S_c^{'}} \{ (1/S_c^{'} - 2 + S_c^{'}) \left[E_z^{n+1}[2] + E_z^{n-1}[0] \right] \\ & + 2(S_c^{'} - 1/S_c^{'}) \left[E_z^{n}[0] + E_z^{n}[2] - E_z^{n+1}[1] - E_z^{n-1}[1] \right] \\ & - 4(1/S_c^{'} + S_c^{'}) E_z^{n}[1] \} - E_z^{n-1}[2] \end{aligned} \tag{14}$$

The update equation (14) is expressed as a function of $S_c^{'} = \Delta t/(\sqrt{\mu\varepsilon}\Delta x) = S_c/\sqrt{\mu_r\varepsilon_r}$, where $S_c = c\Delta t/\Delta x$ is the Courant number. The boundary equation was derived here for the left boundary of the grid but can be similarly found for each boundary. Being a second order equation, two spatial samples are needed as well as samples that are two time steps old. This is to be taken into account during the implementation, where such samples should be stored in memory.

4 Harmonic source

A harmonic source can be simulated by imposing the value of the electrical field where it is located. The harmonic source equation is:

$$E_z(t) = \cos(\omega t) \tag{15}$$

In free space, ω can be rewritten as $\frac{2\pi c}{\lambda}$. To be able to implement the source in a discrete way, (15) has to be written as a function of the spatial and temporal steps. Hence

$$\lambda = N_\lambda \Delta x \qquad\qquad t = n\Delta t \tag{16}$$

N_λ being the *number of points per wavelength*. Using (16), the harmonic source equation becomes

$$E_z^n = \cos\left(\frac{2\pi c\Delta t}{N_\lambda \Delta x} n\right) \tag{17}$$

$$= \cos\left(\frac{2\pi S_c}{N_\lambda} n\right) \tag{18}$$

Equation (18) will be used instead of (17) to make it independent of Δx and Δt.

5 Implementation

5.1 Stability

In order to guarantee the FDTD stability in 2D, a restriction on Δt has to be done:

$$\Delta t \leq \frac{\Delta x}{c\sqrt{2}} \tag{19}$$

Equation (19) can be interpreted as the impossibility of propagating the energy at more than one spatial step by time step.

To ensure FDTD stability whatever the simulation, Δx and Δt will not be explicitly defined. Instead, the ratio between them S_c is used. Hence, $S_c = \frac{1}{\sqrt{2}}$.

5.2 Main loop

The main loop of the code is shown below. All functions are initialized, then the time stepping is performed.

```
#include "fdtd-alloc1.h"
#include "fdtd-macro-tmz.h"
#include "fdtd-proto1.h"
#include "ezinc.h"

int main(){
  Grid *g;                                  // Create grid structure

  ALLOC_1D(g, 1, Grid);                     // allocate memory for Grid

  gridInit(g);                              // initialize all functions
  abcInit(g);
  ezIncInit(g);
  snapshotInit2d(g);

  /* do time stepping */
  for (Time = 0; Time < MaxTime; Time++) {
    updateH2d(g);                           // update magnetic field
    updateE2d(g);                           // update electric field
    Ez(50, SizeY/2) = ezInc(Time);          // Source: impose Ez
    abc(g);                                 // absorbing boundary conditions
    snapshot2d(g);                          // save fields value
  }
  return 0;
}
```

5.3 Fields update

The following code updates the fields of the grid at each iteration, using equations (6) to (8).

```
#include "fdtd-macro-tmz.h"

void updateH2d(Grid *g) {
  int mm, nn;

  /* update Hx for every node */
```

```
  for (mm = 0; mm < SizeX; mm++)
    for (nn = 0; nn < SizeY - 1; nn++)
            Hx(mm, nn) = Chxh(mm, nn) * Hx(mm, nn)
              - Chxe(mm, nn) * (Ez(mm, nn + 1) - Ez(mm, nn));

  /* update Hy for every node */
  for (mm = 0; mm < SizeX - 1; mm++)
    for (nn = 0; nn < SizeY; nn++)
            Hy(mm, nn) = Chyh(mm, nn) * Hy(mm, nn)
              + Chye(mm, nn) * (Ez(mm + 1, nn) - Ez(mm, nn));
  return;
}

void updateE2d(Grid *g) {
  int mm, nn;

  /* update Ez for every node */
  for (mm = 1; mm < SizeX - 1; mm++)
    for (nn = 1; nn < SizeY - 1; nn++)
            Ez(mm, nn) = Ceze(mm, nn) * Ez(mm, nn) +
              Cezh(mm, nn) * ((Hy(mm, nn) - Hy(mm - 1, nn)) -
                              (Hx(mm, nn) - Hx(mm, nn - 1)));
  return;
}
```

The coefficients used in the previous code were initialised as arrays in the function `gridInit`. A shorthand notation is used: for example, `Chxh` is the coefficient multiplying H when updating H_x.

```
#include "fdtd-macro-tmz.h"
#include "fdtd-alloc1.h"
#include <math.h>

#define LOSS 0.1328          // Loss factor
#define PERM 43              // Relative permittivity

void gridInit(Grid *g) {
  double imp0 = 377.0;       // Z_0
  int mm, nn;

  /* terms for the lossy zone */
  double l, L, xLocation, yLocation, xCenter, yCenter, l2, L2;

  Type = tmZGrid;
  SizeX = 201;               // x size of domain
  SizeY = 201;                // y size of domain
  MaxTime = 102;             // duration of simulation
  Cdtds = 1.0 / sqrt(2.0); // Courant number

  /* Allocate memory for the arrays of coefficients */
  ALLOC_2D(g->hx,   SizeX, SizeY - 1, double);
  ALLOC_2D(g->chxh, SizeX, SizeY - 1, double);
  ALLOC_2D(g->chxe, SizeX, SizeY - 1, double);
  ALLOC_2D(g->hy,   SizeX - 1, SizeY, double);
  ALLOC_2D(g->chyh, SizeX - 1, SizeY, double);
  ALLOC_2D(g->chye, SizeX - 1, SizeY, double);
  ALLOC_2D(g->ez,   SizeX, SizeY, double);
  ALLOC_2D(g->ceze, SizeX, SizeY, double);
  ALLOC_2D(g->cezh, SizeX, SizeY, double);

  /* set electric-field update coefficients */
```

```
  for (mm = 0; mm < SizeX; mm++)
    /* loss-less zone */
    for (nn = 0; nn < SizeY; nn++) {
      Ceze(mm, nn) = 1.0;
      Cezh(mm, nn) = Cdtds * imp0;
      /* define lossy zone */
      if (mm >= 30 && mm <= 40 && nn <= 50)
      {
        Ceze(mm,nn) = (1.0 - LOSS) / (1.0 + LOSS);
        Cezh(mm,nn) = imp0 / PERM / (1.0 + LOSS) * Cdtds;
      }
    }
  /* set magnetic-field update coefficients */
  for (mm = 0; mm < SizeX; mm++)
    for (nn = 0; nn < SizeY - 1; nn++) {
      Chxh(mm, nn) = 1.0;
      Chxe(mm, nn) = Cdtds / imp0;
    }
  for (mm = 0; mm < SizeX - 1; mm++)
    for (nn = 0; nn < SizeY; nn++) {
      Chyh(mm, nn) = 1.0;
      Chye(mm, nn) = Cdtds / imp0;
    }
  return;
}
```

5.4 Absorbing boundary conditions

The following code performs the update equation (14) at the boundaries of the grid. First, macros are defined to easily access the stored values of the field and the coefficients are computed in the initialisation function. At each iteration, the new values of the field are computed and then stored in order to perform the next computation.

```
#include <math.h>
#include "fdtd-alloc1.h"
#include "fdtd-macro-tmz.h"

/* Define macros for arrays that store the previous values of the fields */
#define EzLeft(M, Q, N)    ezLeft[(N) * 6 + (Q) * 3 + (M)]
#define EzRight(M, Q, N)   ezRight[(N) * 6 + (Q) * 3 + (M)]
#define EzTop(N, Q, M)     ezTop[(M) * 6 + (Q) * 3 + (N)]
#define EzBottom(N, Q, M)  ezBottom[(M) * 6 + (Q) * 3 + (N)]

/* Instantiate coefficients */
static int initDone = 0;
static double coef0, coef1, coef2;
static double *ezLeft, *ezRight, *ezTop, *ezBottom;

void abcInit(Grid *g) {
  /* Initialise the ABC, must be called before using abc() */
  double temp1, temp2;
  initDone = 1;

  /* allocate memory for ABC arrays */
  ALLOC_1D(ezLeft, SizeY * 6, double);
  ALLOC_1D(ezRight, SizeY * 6, double);
  ALLOC_1D(ezTop, SizeX * 6, double);
  ALLOC_1D(ezBottom, SizeX * 6, double);

  /* calculate ABC coefficients for the update equation */
```

```
  temp1 = sqrt(Cezh(0, 0) * Chye(0, 0));
  temp2 = 1.0 / temp1 + 2.0 + temp1;
  coef0 = -(1.0 / temp1 - 2.0 + temp1) / temp2;
  coef1 = -2.0 * (temp1 - 1.0 / temp1) / temp2;
  coef2 = 4.0 * (temp1 + 1.0 / temp1) / temp2;

  return;
}

void abc(Grid *g) {
  /* Updates the field on the boundaries of the grid g*/
  int mm, nn;

  /* Update of the left side of grid */
  for (nn = 0; nn < SizeY; nn++) {
    Ez(0, nn) = coef0 * (Ez(2, nn) + EzLeft(0, 1, nn))
     + coef1 * (EzLeft(0, 0, nn) + EzLeft(2, 0, nn)
                - Ez(1, nn) - EzLeft(1, 1, nn))
     + coef2 * EzLeft(1, 0, nn) - EzLeft(2, 1, nn);

    /* memorize old fields for next iteration */
    for (mm = 0; mm < 3; mm++) {
      EzLeft(mm, 1, nn) = EzLeft(mm, 0, nn);
      EzLeft(mm, 0, nn) = Ez(mm, nn);
    }
  }

  /* ABC at right side of grid */
  for (nn = 0; nn < SizeY; nn++) {
    Ez(SizeX - 1, nn) = coef0 * (Ez(SizeX - 3, nn) + EzRight(0, 1, nn))
     + coef1 * (EzRight(0, 0, nn) + EzRight(2, 0, nn)
                - Ez(SizeX - 2, nn) - EzRight(1, 1, nn))
     + coef2 * EzRight(1, 0, nn) - EzRight(2, 1, nn);

    /* memorize old fields for next iteration */
    for (mm = 0; mm < 3; mm++) {
      EzRight(mm, 1, nn) = EzRight(mm, 0, nn);
      EzRight(mm, 0, nn) = Ez(SizeX - 1 - mm, nn);
    }
  }

  /* ABC at bottom of grid */
  for (mm = 0; mm < SizeX; mm++) {
    Ez(mm, 0) = coef0 * (Ez(mm, 2) + EzBottom(0, 1, mm))
     + coef1 * (EzBottom(0, 0, mm) + EzBottom(2, 0, mm)
                - Ez(mm, 1) - EzBottom(1, 1, mm))
     + coef2 * EzBottom(1, 0, mm) - EzBottom(2, 1, mm);

    /* memorize old fields for next iteration */
    for (nn = 0; nn < 3; nn++) {
      EzBottom(nn, 1, mm) = EzBottom(nn, 0, mm);
      EzBottom(nn, 0, mm) = Ez(mm, nn);
    }
  }

  /* ABC at top of grid */
  for (mm = 0; mm < SizeX; mm++) {
    Ez(mm, SizeY - 1) = coef0 * (Ez(mm, SizeY - 3) + EzTop(0, 1, mm))
     + coef1 * (EzTop(0, 0, mm) + EzTop(2, 0, mm)
                - Ez(mm, SizeY - 2) - EzTop(1, 1, mm))
     + coef2 * EzTop(1, 0, mm) - EzTop(2, 1, mm);
```

```
    /* memorize old fields for next iteration */
    for (nn = 0; nn < 3; nn++) {
      EzTop(nn, 1, mm) = EzTop(nn, 0, mm);
      EzTop(nn, 0, mm) = Ez(mm, SizeY - 1 - nn);
    }
  }
  return;
}
```

5.5 Harmonic source

The value of the electric field on the points where an harmonic source is present is imposed by the following code, according to Equation 18.

```
#include "ezinc.h"

static double cdtds, ppw = 0;

void ezIncInit(Grid *g){
  /* define the source parameters */
  printf("Enter the points per wavelength for harmonic source: ");
  scanf(" %lf", &ppw);
  cdtds = Cdtds;
  return;
}

double ezInc(double time) {
  /* calculate source function at given time */
  return cos(2 * M_PI * cdtds * time/ppw);
}
```

6 Validation

Once the theoretical update equations are implemented, a validation procedure is needed to confirm the legitimacy of the simulations. The code is validated with a test including a lossy region at the right side of the grid, while the left side is loss-less. In this case, the layer is composed of cerebral tissue (relative permittivity ε_r, conductivity σ) and the source of excitation is an harmonic source at frequency f (this validation will be a first step to introduce the SAR simulation). The values of the parameters are summarised in Table 1. The simulation will be considered valid if the wavelength and the attenuation in the lossy layer match the expected theoretical results.

ε_r	43
σ	$1.3\,\mathrm{S\,m^{-1}}$
f	915 MHz

Table 1: Validation parameters

The wavelength in the lossy layer is defined as

$$\lambda_l = \frac{v_\varphi}{f} \quad \text{where} \quad \begin{cases} v_\varphi & \text{propagation/phase velocity} \\ f & \text{wavelength frequency} \end{cases} \tag{20}$$

Knowing that

$$v_\varphi = \frac{\omega}{\beta} \quad \text{and} \quad \beta = \omega\sqrt{\frac{\mu_0\varepsilon_r\varepsilon_0}{2}\left[\sqrt{1+\left(\frac{\sigma}{\omega\varepsilon_r\varepsilon_0}\right)^2}+1\right]} \tag{21}$$

by coupling (20) and (21), one finds $\lambda_l \approx 48\,\text{mm}$. Being the wavelength of interest, $\Delta x = \frac{\lambda_l}{10} = 4.8\,\text{mm}$ and $\Delta t = S_c\frac{\Delta x}{c} \approx 11.32\,\text{ns}$.

The loss can be characterised by the skin depth $\delta = \frac{1}{\alpha}$, where

$$\alpha = \omega\sqrt{\frac{\mu_0\varepsilon_r\varepsilon_0}{2}\left[\sqrt{1+\left(\frac{\sigma}{\omega\varepsilon_r\varepsilon_0}\right)^2}-1\right]} \tag{22}$$

α representing the rate of decay of the wave amplitude (represented Figure 2). In that way, $\alpha \approx 35.91\,\text{m}^{-1}$ and $\delta \approx 2.79\,\text{cm}$.

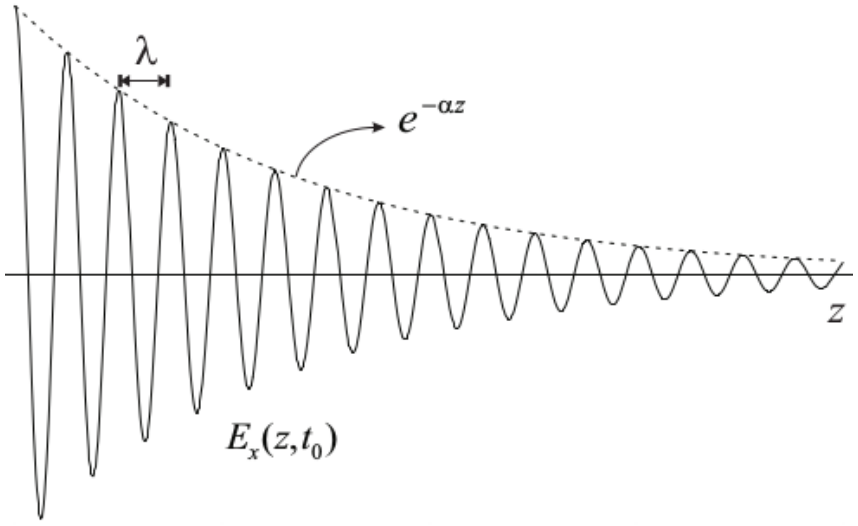

Figure 2: Decay of the wave amplitude in a lossy layer

The different parameters needed to run the simulation are then defined, such as the number of point per wavelength N_λ (source excitation) and the loss parameter used in the code of subsection 5.3 (equals to $\frac{\sigma\Delta t}{2\varepsilon}$ since the magnetic effects are neglected ($\sigma_m = 0$, $\mu_r = 1$)).

$$N_\lambda = \frac{\lambda}{\Delta x} = \frac{c}{f\Delta x} \approx 68.20 \tag{23}$$

$$\text{LOSS} = \frac{\pi}{N_\lambda}S_c\sqrt{\left(1+\frac{N_\lambda^2}{2\pi^2 N_L^2\varepsilon_r\mu_r}\right)^2+1} \approx 19.35\times10^{-3} \tag{24}$$

where

$$N_L = \frac{\delta}{\Delta x} \tag{25}$$

The result of the simulation is shown in Figure 3. The electric field after the lossy layer on a horizontal line $E_z(x, 100, t')$ is shown in Figure 4. On Figure 4, the *Curve*

Fitting MATLAB tool was used to fit the exponential decay. λ_l and α can be deduced: $\lambda_l \approx 48\,\mathrm{mm}$ and $\alpha \approx 35.26\,\mathrm{m}^{-1}$, as expected (neglecting graphical fitting errors).

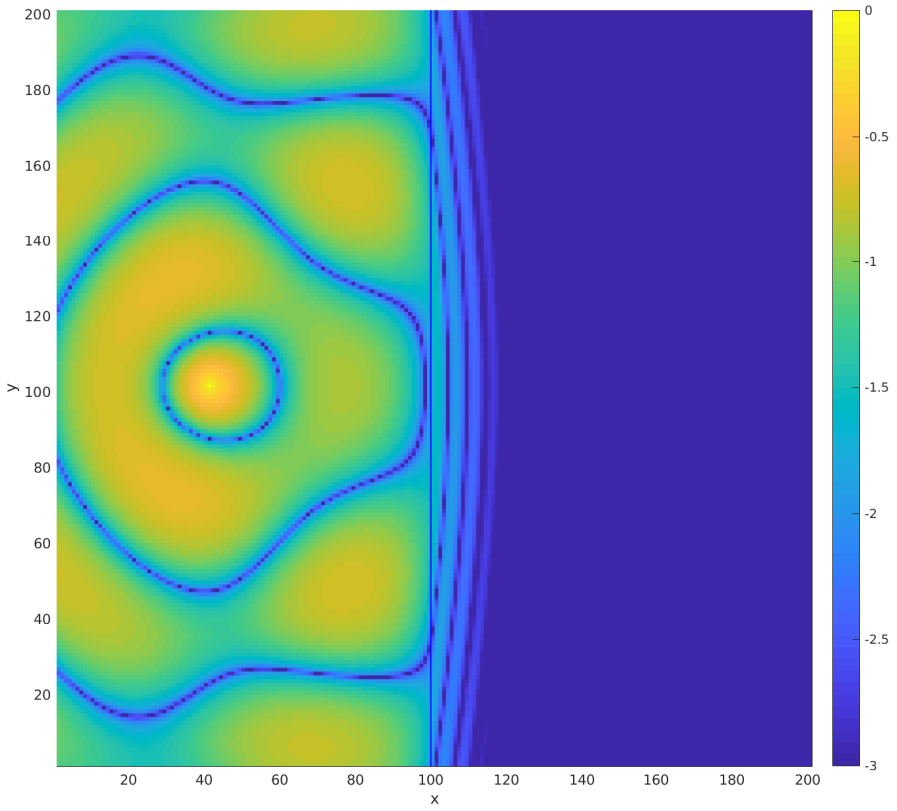

Figure 3: Decay of the wave amplitude in a lossy layer of cerebral tissue

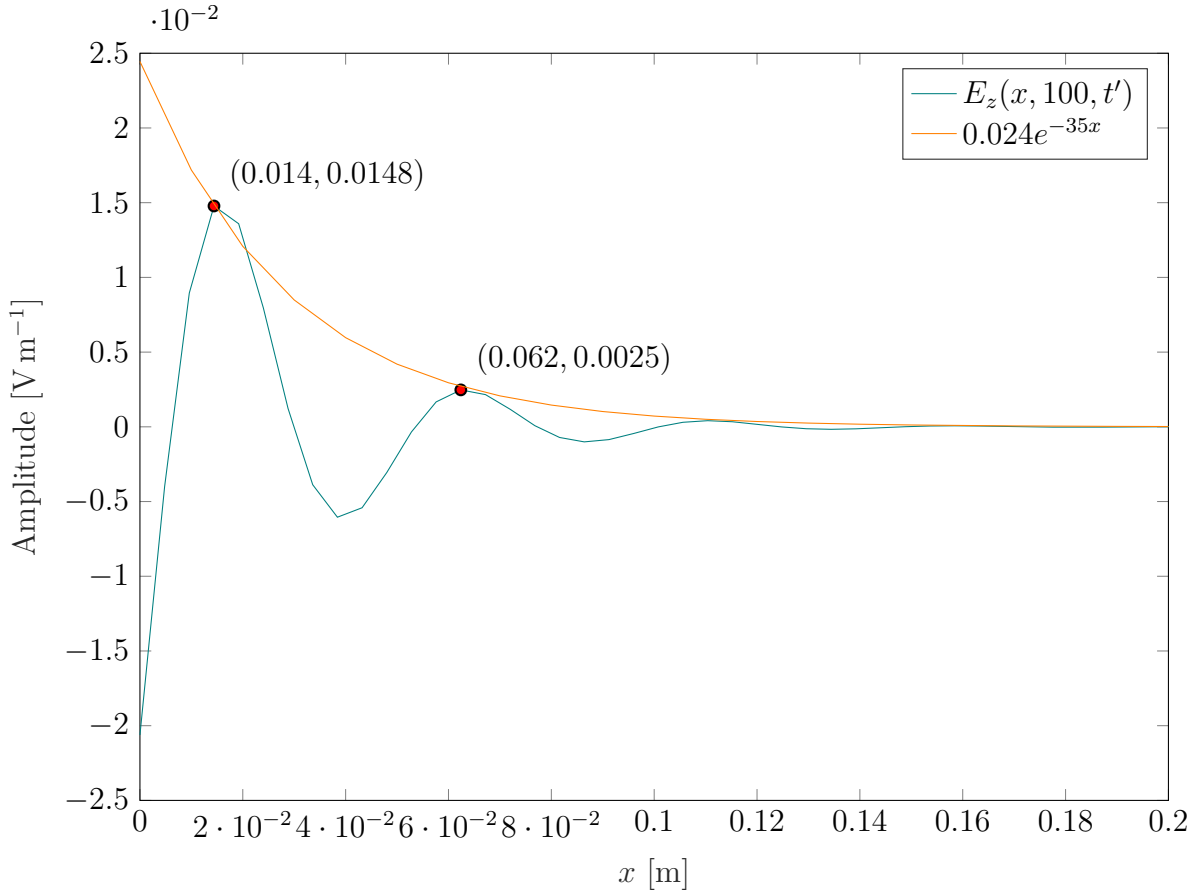

Figure 4: Wave propagation with a lossy layer of cerebral tissue

Hence, the simulation is validated and more advanced simulations can be designed.