User's manual for FDTD code
Optical Science Lab
Revision 1.0

Joseph L. Hollmann

October 1, 2013

1 Introduction

This will provide an introduction to the two dimensional FDTD simulation, *execute_FDTD*. It is important to note that this manual assumes a basic knowledge of MATLAB programming.

2 Getting Started

In order to use *execute_FDTD*, you will need to ensure you have the functions listed in Table **??**.

3 Calling the program

Call the FDTD program as follows:
[Ewave,media] = execute_FDTD(filename,media,Ewave,timesteps,options);

Function	Description	Release
execute_FDTD	calls FDTD and displays results	
sourceval	creates and updates source	
FDTD_2D	Solves discreteized Maxwell's equations using CPML BC's	

Table 1: Minimum programs you will need to use *execute_FDTD*.

Variable	Description
Filename	Full name (including path) of output file
media	Structure that stores information about medium. (see section 3.1)
Ewave	Stores field, source and detector (optional) information. (see section 3.2)
timesteps	number of timesteps simulation should run through – integer
showresults	is a 1 to graph results every 20 timesteps or 0 to hide graphs
options	structure specifies simulation options for the user (see section 3.3)

Table 2: The inputs for the function *execute_FDTD*.

The inputs *media*, *Ewave* and *options* are structures that are described in Sections 3.1, 3.2, and 3.3, respectively. A description of each input is provided in Table 2.

3.1 media

This structure contains all the information necessary to describe the medium our electromagnetic wave is propagating through. The structure contains the following fields: (note everything is described in meters)

- .depth – The depth of the medium
- .width – The width of the medium
- .delSpatial – The FDTD simulation area is discretized into pixel. This is the square pixel's size along one dimension.
- .delTemporal – The FDTD simulation also occurs in discrete timesteps. This is the size of the timestep.
- .ior_matrix – Simulation space.
  Note that:

  size(media.ior_matrix,1) = round(media.depth/media.delSpatial);
  size(media.ior_matrix,2) = round(media.width/media.delSpatial);

Note: all these fields are called as follows:
media.XXXX
where XXXX is the field described above. There may be other fields in the media structure but they are not necessary for the FDTD code. If you wish to know their purpose, please refer to your medium–creation function to determine there function.

3.2 Ewave

This structure contains all the information necessary to describe our Electromagnetic wave, its source and a possible detector. It contains several fields

- .lambda0 – wavelength of the electric field to be simulated. Necessary for source generation.
- .source – contains information about the source
  - .dpos – depth position of the 1–pixel thick source.
  - .xpos – location of the source's center
  - .type – describes the type of source being emitted. Can be either 'plane' for a plane wave or 'gaussian' for a Gaussian wave. See table **note:** → here ← for a description of the two options. Also, note that the 'gaussian' option may require a few more fields/
  - .rampup – describes the length of time it takes for the source to reach full strength. Note that the signal ramps up to full strength (1 V/M) as a error function
  - .rampdwn – describes the length of time it takes required for the source to turn off. Note that the signal ramps down to off as a error function
  - . pulswidth – describes the length of time the source is a full strength. If this is greater than the simulation length, the source will never turn off.
  - .delay – describes time after simulation has started before source is turned on. I don't think this has ever been used.

Note: an example call is as follows
Ewave.source.dpos

- .detector (optional – collects time–varying field signal at a specified depth)
  - .delay – describes time to wait before collecting data. Ideal for reducing memory consumption if the field needs to propagate some distance before relevant data may be collected
  - .dpos – depth location of the detector

  Note: an example call is as follows
  Ewave.detector.dpos

3.3 options

This structure is a 'catch all' that allows the user maximum versatility with the FDTD simulation without having to edit the actual code. The options structure can contain information relating to the boundary conditions, visualization and interaction with modules (explained **note:** → here ←).

3.3.1 boundary conditions

The default boundary conditions are the convolutional perfectly matched layers (CPML) with 10 layers. [**?**] Due to historical reasons, the code also allows the user to specify Mur boundary conditions [**?**]. You may specify the BC's as follows

3.3.2 modules

Early on in code development, we found every user wanted a slightly different output. Instead of developing a solution for each user, we developed a way to allow the user to interact with the program during execution without having to edit the simulation code. The module is called before the FDTD code is executed. This is typically where necessary constants are initialized and stored. The code is then called at every iteration of the FDTD code.

Say we have a module named 'filename'. It is called as follows:
options.callfcn.file = 'filename';

The module will tell execute_FDTD what variables it needs and control the interaction from then on. See **note:** → here ← for more details.

4 Outputs

After execution, *execute_FDTD* will automatically store the structures in *media* and *Ewave*.

4.1 Ewave

In addition to the input fields, the structure *Ewave* will also contain the following:

- .Ez_field – real electric field (z–direction) for the final time-step of the simulation
- .Hx_field – real magnetic field (x–direction) for the final time-step of the simulation
- .Hy_field – real magnetic field (y–direction) for the final time-step of the simulation

If you also specified the necessary information for a detector, you will also find the signal at the detector–depth versus time in
Ewave.detector.timesig