User Guide v-1

Finite Difference Time Domain Simulator User Guide

SemiVi LLC

Zollikon, Switzerland.

October 11, 2025

Contents

Chapter 1

Introduction

The FDTD simulation package is a software to perform FDTD simulations on a rectangular or a cubic grid with a non-uniform mesh along any of the three axes. A 2D or 3D structure created using SemiVi tensor structure and mesh generator or an equivalent config file is taken as an input for simulations. Electric fields and magnetic fluxes calculated during the simulation can be stored on a 2D grid to visualize as a movie, or can be post-processed to obtain phase, average, or time evolution of the field at a node.

1.1 Features

FDTD simulator supports materials with the following properties.

- Constant real and imaginary permittivity,
- Wavelength-dependent real and imaginary permittivity,
- Dispersive materials which exhibit dielectric response of the following models, 1. Drude model, 2. Debye model, 3. Lorentz model, or 4. Kerr model.
- Constant but non-unity real permeability.

Following types of electromagnetic wave sources can be used in the simulator.

- Plane-wave source with various beam shapes such as, uniform beam, Gaussian beam, or mode-beam.

- Dipole source.

The simulator supports domain boundary models, such as 1. RBC, 2. RBC, 3. CPML, and 4. RBC for oblique incidence. In the case of RBC with oblique incidence of any of the waves, Sine-Cosine method is internally used for FDTD simulations.

A Scattering Matrix (ScMat) calculation interface using FDTD simulations has been implemented in the simulator. Using this interface, ScMat elements can be calculated between user-defined ports for each of the user-provided wavelengths. Extraction of broad-band ScMat parameters at all the frequencies starting from DC to user-provided frequency can be performed using the Broad-band ScMat interface. Results of the simulation are stored in touchstone file formats.

FDTD simulator also comes with built-in Graphics Processing Unit (GPU) acceleration which uses graphics cards to perform faster FDTD simulations.

1.2 Installation

SemiVi currently supports software installation on various Linux distributions. The software installer is available in Debian package (*.deb file) and in RPM format (*.rpm file).

Note, the following package needs to be installed manually by you before installing the circuit solver from the installer package.

- Intel math kernel libraries (released in 2020 or later): They include distributions of open-mp, pardiso, etc. specific for Intel processors. Installation of mkl libraries is necessary, *only if you have downloaded mkl version of the OptoSolver*. The OptoSolver sources mkl functions from the above installation. These functions can offer speed-up in the calculations on Intel processors. The mkl package can be downloaded from Intel website. *If the OptoSolver without mkl-acceleration is downloaded, then installation of the above package is not necessary.*

- Nvidia Cuda (v11.6 or above) – only if GPU accelerated of FDTD solver licenses are purchased. It is recommended to install proprietary Nvidia drivers together with Cuda.

Once all the above packages are installed, download the installer on the local machine. The installer file named `optosolver_amd64.deb` will appear in the `Downloads` directory. Go to the directory using `cd` command. Use the following command to install the `optosolver` from the installer.

```
>> sudo apt install ./optosolver_amd64.deb
```

Alternately, one may use `dpkg` to install the software and use `apt` to install missing dependencies as follows.

```
>> sudo dpkg -i ./optosolver_amd64.deb
>> sudo apt install -f
```

You need to have `root` access to install the software on your machine.

1.3 Licensing

Two types of licenses can be purchased for SemiVi FDTD solver.

Node-locked licenses enable unlimited number of *simultaneous* executions of the FDTD solver on the client machine. The node-locked license limits the usage of the FDTD solver only to the machine on which the license is activated.

With *server licenses*, the FDTD solver can be run on any of the machines in the client organization on which the server license is activated. However, only the specified number of *simultaneous* executions are possibie at a time.

1.3.1 Purchasing the licenses

The clients can place order for any of the above licenses on SemiVi website (https://www.semivi.ch/sales) or by contacting our sales-person.

We will process the request and send the license files by email. The license files need to be activated on the desired machines using the license key which is emailed separately using the following command.

1.3.2 Installation of SemiVi-activator

The license file must be activated on the desired computer before use. For that purpose, download the installer `semivila_amd64.deb` file on the local machine and install it as follows.

```
>> sudo apt install ./semivila_amd64.deb
```

1.3.3 License activation

To activate the license file, please run the following command.

```
>> semivila -a File.lic <Server|NodeLocked>License.lic\\
```

Replace `File.lic` with the your license file, and use appropriate name for the activated file. You will be prompted to input the 16 digit license key. A successful activate of the license file will generate the activated license file. Copy the activated license file to the `/opt/semivi/licenses/` folder and rename it to `ServerLicense.lic` or `NodeLockedLicense.lic` for server and node-locked licenses respectively. If you have more than one license files, please delete the older expired license files. If you wish to keep more than one active license files, you can also name the license files as `<i>NodeLockedLicense.lic` where `<i>` could be from 0 to 49. For ex. `49NodeLockedLicense.lic` or `49ServerLicense.lic`. The program will read the license files and lock the first available license. All the target users must have read rights on the license file.

User-guides of all the software provided by SemiVi are stored at the location `/opt/semivi/userguides/`.

Tutorials of all the software provided by SemiVi are stored at the location `/opt/semivi/tutorials/optosolver`.

Chapter 2

Theory of FDTD Solver

Response of a material system in the absence of electric charges to an electromagnetic excitation is described by Maxwell's equations (with $\rho = 0$).

$$\nabla \cdot \vec{D} = 0 \tag{2.1a}$$

$$\nabla \cdot \vec{B} = 0 \tag{2.1b}$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \tag{2.1c}$$

$$\nabla \times \vec{H} = \frac{\partial \vec{D}}{\partial t} \tag{2.1d}$$

Here, $\vec{D} = \epsilon \vec{E}$ is a displacement vector, E is an electric field, $\vec{H} = \vec{B}/\mu$ is the magnetic flux, and $\vec{B}$ is the magnetic field.

Out of Maxwell's equations, Eq. 2.1a and Eq.2.1b are always satisfied by FDTD algorithm. Eq. 2.1c links spatial variation of electric field vector ($\vec{E}$) with temporal variation of magnetic flux vector ($\vec{H}$) and Eq. 2.1d links spatial variation of $\vec{H}$ with temporal variation of $\vec{E}$. In FDTD algorithm, these two equations (Eq. 2.1c and Eq. 2.1d) are alternately solved on a special rectangular (cubic, in 3D) grid called Yee grid at every half-time step. That is, Eq. 2.1c is solved at time-step q while Eq. 2.1d is solved at time-step $q + \frac{1}{2}$.

2.1 FDTD Algorithm

Curl of $\vec{E}$ on the Left Hand Side (LHS) of Eq. 2.1c can be expanded and its x, y, z components can be equated to those of the Right Hand Side (RHS) which gives,

$$\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} = -\mu \frac{\partial H_x}{\partial t}, \tag{2.2a}$$

$$\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} = -\mu \frac{\partial H_y}{\partial t}, \tag{2.2b}$$

$$\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = -\mu \frac{\partial H_z}{\partial t}, \tag{2.2c}$$

$$\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} = \sigma E_x + \epsilon \frac{\partial E_x}{\partial t}, \tag{2.2d}$$

$$\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} = \sigma E_y + \epsilon \frac{\partial E_y}{\partial t}, \tag{2.2e}$$

$$\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} = \sigma E_z + \epsilon \frac{\partial E_z}{\partial t}. \tag{2.2f}$$

In FDTD algorithm, the above equations are discretized on a special Yee grid. On this grid, various components of $\vec{E}$ and $\vec{H}$ are discretized as shown in Fig. **??**. Note, how these components are defined at the points which are shifted by half the grid spacing from the node of the grid. All these components are stored at the same $[m, n, p]$ location on the computer memory. Using the notations, update equation for E_x at $t = (q+1)\Delta_t$ is,

$$\begin{aligned} E_x^{q+1}[m,n,p] = {} & \frac{1 - \frac{\sigma \Delta_t}{2\epsilon}}{1 + \frac{\sigma \Delta_t}{2\epsilon}} E_x^q[m,n,p] + \\ & \frac{1}{1 + \frac{\sigma \Delta_t}{2\epsilon}} \Big(\frac{\Delta_t}{\epsilon \Delta_y} \{ H_z^{q+1/2}[m,n,p] - H_z^{q+1/2}[m,n-1,p] \} - \\ & \frac{\Delta_t}{\epsilon \Delta_z} \{ H_y^{q+1/2}[m,n,p] - H_y^{q+1/2}[m,n,p-1] \} \Big), \end{aligned} \tag{2.3}$$

Similar update equations can be written for E_y and E_z. All the components of $\vec{E}$ are computed at $t = (q+1)\Delta_t$ from $\vec{E}$ at $t = (q-1)\Delta_t$

and $\vec{H}$ at $t = (q+1)\Delta_t$. Note, that we have used integral location numbers where the quantities are stored on computer memory.

Update equation for Hx at $t = (q+1/2)\Delta_t$ is given by,

$$H_x^{q+1/2}[m,n,p] = H_x^{q-1/2}[m,n,p] + (\frac{\Delta_t}{\mu\Delta_z}\{E_y^q[m,n,p] - E_y^q[m,n,p-1]\} - \frac{\Delta_t}{\mu\Delta_y}\{E_z^q[m,n,p] - E_z^q[m,n-1,p]\}), \quad (2.4)$$

Similar update equations can be written for H_y and H_z. All the components of $\vec{H}$ are computed at $t = (q+1/2)\Delta_t$ from $\vec{H}$ at $t = (q-1/2)\Delta_t$ and $\vec{E}$ at $t = q\Delta_t$.

In the above Eq. 2.3 and Eq. 2.4, Δ_t is temporal time-step which is kept fixed throughout the simulations. Δ_x, Δ_y, and Δ_z are grid spacing in x-, y- and z- directions. They can be fixed or variable throughout the grid. But, the grid must be a rectangular (cubic, in 3D) grid. Also, ϵ is permittivity, μ is permeability, and σ is electric conductivity of the material at location $[m,n,p]$.

Two types of waves can propagate in 2D structures, Transverse Magnetic (TM) and Transverse Electric (TE). In TM propagation, E_z, H_x, and Hy are non-zero while other components are zero. In TE propagation, E_x, E_y, and H_z components are non-zero while others are zero. Update equations in these two distinct cases can be derived by neglecting the 'zero' components in Eq. 2.2.

If `WavelengthDepIndex` is set in FDTD solver settings, then ϵ and σ are calculated from wavelength dependent refractive index of the material, using following equations,

$$\epsilon = (n+ik)^2 = n^2 - k^2 + ink = \epsilon' + i\epsilon'' \quad (2.5)$$

where n and k are real and imaginary parts of refractive index at a given wavelength (λ). Eq. 2.5 gives real and imaginary permittivity from complex refractive index. Conductivity is calculated from imaginary permittivity using the following relation,

$$\sigma = 2 \cdot \epsilon'' \cdot \epsilon_0\omega = 2 \cdot \epsilon'' \cdot \epsilon_0 \cdot (\frac{2 \cdot \pi \cdot C}{\lambda}) \quad (2.6)$$

.

2.1.1 Calculating time-step

To achieve stability of FDTD simulations, the time-step (Δ_t) has an upper limit.

For 2D simulations,

$$\Delta_t \leq \min_{m\in[0,M),n\in[0,N)} \frac{1}{c_{m,n}\left(\sqrt{\frac{1}{\Delta_{x,m}^2}+\frac{1}{\Delta_{y,n}^2}}\right)} \tag{2.7}$$

Here, $\Delta_{x,m}$ and $\Delta_{y,n}$ are the grid spacing at m^{th} and n^{th} point, and $c_{m,n}$ is the speed of light at grid point $[m,n]$.

For 3D simulations,

$$\Delta_t \leq \min_{m\in[0,M),n\in[0,N),p\in[0,P)} \frac{c}{c_{m,n,p}\left(\sqrt{\frac{1}{\Delta_{x,m}^2}+\frac{1}{\Delta_{y,n}^2}+\frac{1}{\Delta_{z,p}^2}}\right)} \tag{2.8}$$

Here, $\Delta_{z,p}$ is the grid spacing at p^{th} point, and $c_{m,n,p}$ is the speed of light at grid point $[m,n,p]$.

In FDTD simulator, Δ_t is set to the upper limit given above.

2.2 Boundary conditions

Three types of boundary conditions can be defined at any of the six faces of cuboidal FDTD simulation domain in 3D (or four edges of rectangular domain in 2D). They are described below.

2.2.1 RBC

This is default Boundary Condition (BC) at all the faces of cuboidal domain. In update equations, it is implicitly assumed that $\vec{E}$ and $\vec{H}$ are zero outside the simulation domain. This is same as introducing a 'hard-wall' at the boundary. As a result, all the electromagnetic waves are reflected at the boundary.

2.2.2 RBC

A RBC is set at both 'min' and 'max' faces (such as 'Xmin' and 'Xmax) of the cuboid simultaneously.

When all the light sources are emitting waves tangential to the faces with RBC, $\vec{E}$ and $\vec{H}$ at the 'min' face are set to their values at 'max' face. For example, if faces normal to x direction are set as periodic, then $\vec{E}$ and $\vec{H}$ are updated as follows.

$$\vec{E}[0,n,p] = \vec{E}[M-1,n,p] \; \forall \; n \in [0,N), \; p \in [0,P) \tag{2.9a}$$

$$\vec{H}[0,n,p] = \vec{H}[M-1,n,p] \; \forall \; n \in [0,N), \; p \in [0,P) \tag{2.9b}$$

Similar equations may be written if RBC are set at the faces normal to y- or z- direction.

When any of the light source is emitting waves in an oblique direction, such that its component normal to the faces with RBC is non-zero, the above equations are invalid. In that case, FDTD simulation switches to sine-cosine method.

2.2.3 CPML

A CPML is set at any 'max' or 'min' face separately. It introduces a special region in the first n layers nearest to the boundary, where n is a user-defined number. During an $\vec{E}$ update in this special region, an additional displacement coming from convolutional functions ($\Psi^q_{Eu,v,w} \forall u,v,w \in x,y,z, u \neq v \neq w$ where v is the normal to the CPML boundary) are added to it to cancel out the reflections coming from the finite domain boundary at the surface.

If CPML is applied at 'Ymin', Eq. 2.2d changes to,

$$\epsilon \frac{\partial E_x}{\partial t} = \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} + \Psi^q_{E_x,y,z} - \Psi^q_{E_x,z,y} \tag{2.10}$$

The convolutional function is defined as follows,

$$\Psi^q_{E_u,v,w} = C_w \cdot \frac{\partial H^q_v}{\partial w} + b_w \cdot \Psi^{q-1}_{E_u,v,w} \tag{2.11}$$

Here, C_w and b_w are functions of attenuation coefficient. Note, that $\Psi^q_{E_u,v,w}$ depends on $\vec{H}$ as well as $\Psi^{q-1}_{E_u,v,w}$. Value of the function at the last time-step is stored at each of the grid points in the special region. It is updated at every time-step as per Eq. 2.11.

2.3 Dispersive medium

Many materials especially metals exhibit complex refractive index which shows strong dispersion near specific frequencies, called 'pole frequencies'. Assumption of a constant refractive index is invalid for such materials. They are treated using Auxiliary Differential Equation (ADE) in FDTD simulations. Drude model provides a good approximation of the dispersion of refractive index in metals. Implementation of Drude model in FDTD using ADE method is described below. The Electric susceptibility for a Drude model is given by,

$$\chi_e(\omega) = \frac{\omega_p^2}{i\omega(i\omega + g)} \tag{2.12}$$

where ω_p is the plasma frequency and g is the damping factor.

Polarization current in the material is given by,

$$\vec{J_p} = i\omega\epsilon_0\chi_e\vec{E} = \epsilon_0\frac{\omega_p^2}{i\omega + g}\vec{E} \tag{2.13}$$

Rearranging the terms and expressing in time-domain ($i\omega \rightarrow \frac{\partial}{\partial t}$),

$$g\vec{J_p} + \frac{\partial\vec{J_p}}{\partial t} = \epsilon_0\omega_p^2\vec{E} \tag{2.14}$$

The above equation can then be used to update $\vec{J_p}$ at a given time-step from its value at previous time-step. This polarization current is added to the update equation for $\vec{E}$. In this way, ADE is used to model dispersive media with 'Drude pole' in FDTD algorithm. Note, that $\vec{J_p}$ must be stored at each node of the dispersive material.

Other poles, such as 'Debye pole', 'Lorentz pole' are transformed to the time domain in a similar manner.

2.4 Sources

2.4.1 Plane-wave source

A source is initialized as an axis-aligned rectangular window in 3D or an axis-aligned segment in 2D structures. The source has its own

'auxiliary' grid which is same as a cross-section of the FDTD grid confined in the source window. $\vec{E}$ and $\vec{H}$ generated by the source (on the auxiliary grid) are *added* to the existing $\vec{E}$ and $\vec{H}$ on FDTD grid. In this way, the source behaves as a 'soft source'.

To generate source $\vec{E}$ and $\vec{H}$ on the auxiliary grid, a point source with a temporally sinusoidal excitation is employed at an end of the grid. At every step, $\vec{E}$ and $\vec{H}$ are advanced using FDTD method on the auxiliary grid *independently*. This procedure yields source $\vec{E}$ and $\vec{H}$ throughout the auxiliary grid with the same properties as the main grid. They are then added to the existing $\vec{E}$ and $\vec{H}$ on the main grid at the corresponding grid-points in the source window.

The above procedure outlines field update for a 'uniform' beam-shape in the source window. To generate sources with different beam-shapes, such as a Gaussian-beam, $\vec{E}$ and $\vec{H}$ are multiplied with a spatial Gaussian function. Similarly, to generate a source with the beam-shape of a mode, $\vec{E}$ and $\vec{H}$ are multiplied with the magnitude of mode electric field. In this treatment, phase information of Gaussian-beam or mode-beam is ignored.

2.4.2 Dipole source

Existence of an electric dipole at any node in the grid introduces a polarization current in the direction of that of the dipole. It is subtracted from $\vec{E}$ at the given node. To avoid pumping energy into the grid indefinitely, polarization current is assumed to be temporally Gaussian or of the shape of Ricker wave.

Chapter 3

Configuration File Structure

Optosolver software reads various inputs from a FDTD solver configuration file and performs FDTD simulations for a given time. The FDTD solver is executed by using the following command –

```
>> OptoSolver fdtdsolver fdtdSiWG_dev.cfg
```

In the above command, the word after `Optosolver` is the name of the program to be executed (in this case – `fdtdsolver`). The program name is followed by the configuration file name (in this case – `fdtdSiWG_dev.cfg`). A sample configuration file of the FDTD solver is provided below.

```
File:
{
  Device = "fdtdSiWG_str.cfg";
  Out = "SiWG";
}

Solver:
{
```

```
  MaximumTime = 15E-14;
  GridType = "TM";
  Settings = ["WavelengthDepIndex"];
}

Source*left:
{
  Type = "PlaneWave";
  BeamShape = "ModeBeam";
  Position: ([-0.6, -0.2, 0.], [-0.6, 0.2, 0.]);
  Theta = 90;
  Phi = 0;
  Wavelength = 0.3;
  EffectiveIndex = 3.5;
  NRise = 5;
  Intensity = 1000;
  Delay = 1;
  Flags = ["UsePowerMethodModeSolver"];
}

Boundary*xbdr:
{
  Axis = ["Xmin", "Xmax"];
  Model = "CPML";
  PMLLayers = 20.;
}

Boundary*ybdr:
{
  Axis = [ "Ymin", "Ymax", "Zmin", "Zmax"];
  Model = "CPML";
  PMLLayers = 20.;
}

Movie*XYEField:
{
  Size = "640x480";
  Plane = "XY";
```

```
  Intercept = 0;
  Quantities = ["ElectricFieldZ"];
}

PointSensor*X0pt:
{
  Position = [0.05, 0.025, 0.];
  Quantities = ["AbsElectricField", "AbsMagneticFlux"];
}

TimeAverage*AvgMid:
{
  TimeBegin = 2E-15;
  TimeEnd = 4E-15;
  Quantities = ["AbsElectricField", "AbsMagneticFlux"];
  Position = ([-0.35, -0.35, 0.], [0.35, 0.35, 0.]);
  TimeSteps = 1;
}

PhaseCalculator*Ph:
{
  Time = 2E-15;
  Quantities = ["ElectricFieldZ", "MagneticFluxZ"];
  Position = ([-0.35, -0.35, 0.], [0.35, 0.35, 0.]);
}

Detector*det:
{
  StepX = 5;
  StepY = 5;
  StepZ = 5;

  Tolerance = 1E0;
  StartTime = 2E-15;
  Position = ([-0.35, -0.35, -0.26], [0.35, 0.35, 0.26]);
  Quantities = ["ElectricFieldZ"];
}
```

The above config file is composed of various sections which define the solver settings, sources, domain boundaries, detectors, the quantities to be visualized over time, etc. In the config file, the string before '*' gives the section type, whereas the string after '*' specifies the section name. Various keywords in each of the section and their functionality is shortly described below.

3.1 File Section

The keyword `Device` provides the file name from which the device structure is created. Internally, the file is processed differently according to its extension.

- If the file extension is "str.cfg", the file is processed as an input file for the tensor mesh generation.
- If the file extension is "str.h5", The file is read as hdf5 file generated by the structure and tensor mesh generator.

The keyword `Out` sets the prefix to the output file name. Prefix is followed by the type of data the file stores, e.g. PointSensor, TimeAverage, etc. The file name also contains unique identifiers of the file. In this case, the output files will be called 'SiWG_<UID>_fdtd.xdmf' and 'SiWG_<UID>_fdtd.h5'. Here, <UID> is a unique set of strings specific for the file stored.

If the mode source is used, the mode solver is used to calculate modes in the cross-section containing the source. In this case, the program also stores the modal quantities in the files named 'SiWG_Xloc_<xid>_mode.xdmf' and 'SiWG_Xloc_<xid>_mode.h5'. Here, <xid> represents index of X coordinate at which the mode in the YZ plane is calculated.

3.2 Solver Section

This section lists the information needed for the FDTD solver. Maximum time for which FDTD time stepping is performed is set by the keyword 'MaximumTime'. When the simulation domain is 2D,

'GridType' specifies whether the grid is 'Transverse Electric (TE)' or 'Transverse Magnetic (TM)'.

Various flags are provided as a list of comma-separated strings with the keyword 'Settings'. In the above file, the keyword 'WavelengthDepIndex' is listed. Hence, the solver uses wavelength dependent dielectric permittivity instead of a constant one.

Exactly one 'Solver' section must be present in the config file.

3.3 Source Section

Multiple sources with different names can be instantiated in a config file.

In the given config file, a source named 'left' is instantiated. The source section lists various properties of the source. This source is of type 'PlaneWave'. That is, it emits a sinusoidal wave in the direction specified by polar angle 'Theta' and azimuthal angle 'Phi' (in degrees). Wavelength of the wave (in μm) is set by the keyword 'Wavelength'. The source begins emitting the plane wave at time-step 'Delay' and the wave reaches its full amplitude in 'NRise' time-steps. Amplitude of the plane wave is set by the keyword 'Intensity' (W/m^2).

Spatial window from which the plane wave is emitted into the FDTD simulation is set by 'Position'. Position is a '(...,)' list of two points (specified as list of three floating point numbers). In a 3D structure, the two points form diagonally opposite points of an 'axis-aligned' rectangle. An axis-aligned rectangle has all of its sides parallel to one of the three axes. In a 2D structure, the two coordinates define an axis-aligned segment.

Spatial shape of the plane wave in the plane of incidence (keyword 'BeamShape') is set to 'ModeBeam'. This means, the mode solver calculates mode at the cross-section of the source at the source wavelength near the index given by 'EffectiveIndex'. Modal distribution of electric field is used to determine spatial shape of the beam. Alternatively, 'UniformBeam' or 'GaussianBeam' can also be used as 'BeamShape'.

Various flags can be provided as a list of comma-separated strings with the keyword 'Flags'. In the above file, the keyword 'UsePowerMethodModeSolver' is listed as a flag. Hence, a faster 'power method' is used for mode calculations in mode solver.

3.4 Boundary Section

Multiple 'Boundary' sections with different names can be instantiated in a config file.

In the given file, a boundary section modeling 'PML' boundary type has been instantiated. This model is applicable along the planes at the boundary perpendicular to the axes given in the list with the keyword 'Axis'. In this case, since the keyword 'Y' is specified, the model is applicable at the layers in XZ plane at both maximum and minimum y boundaries. Separate boundary models can be applied at +Y or -Y axis by using the keywords 'Ymax' and 'Ymin', respectively. Boundaries perpendicular to Z axis are modeled in the same manner.

For 'CPML' type boundary, the number of absorbing PML layers at the boundaries are specified with the keyword 'PMLLayers'.

Separate boundary sections must be used for instantiating different boundary models. Following boundary models are recognized by the Beam Propagation Method (BPM) solver – 1. Perfectly Matching Layer (PML) 2. Absorbing Boundary Condition (ABC) 3. RBC 4. RBC. By default, boundaries at all the six faces of the simulation domain are assumed to be reflecting.

3.5 Data Postprocessors

Various data postprocessors and visualizers are available in the FDTD solver.

3.5.1 Movie section

FDTD solver can store a given quantity on a 2D grid or a 2D cross-section of a 3D grid at each time-step in hdf5 format and a script file to visualize the data as a movie in xdmf format. The files are named, '<out>_<secname>_fdtd.h5' and '<out>_<secname>_fdtd.xdmf', respectively. Here <secname> is the name of the `Movie` section.

In 2D grid, the data on the entire simulation domain is stored. In 3D grid, the data in the plane parallel to the plane given by the keyword `Plane` at the intercept given by `Intercept` is stored. In

this case, data in 'XY' plane at the Z-intercept of 0.0 is stored. The quantities to be stored are listed by the keyword `Quantities` as a list of strings. In this case, 'ElectricFieldZ' is stored.

In this case, the files named 'SiWG_XYEField_fdtd.h5' and 'SiWG_XYEField_fdtd.xdmf' will be stored at the end of simulation. The data can be visualized in paraview using the following command.

```
>> paraview SiWG_XYEField_fdtd.xdmf
```

3.5.2 PointSensor section

FDTD solver stores the value of the quantities at each time step at a single point on the FDTD grid nearest to the point whose coordinates are given by `Position`. The quantities to be stored are listed by the keyword `Quantities` as a list of strings. In this case, 'ElectricFieldZ' is stored. The data is stored in 'csv' format in the file named '<out>_<secname>_fdtd.csv', where <secname> is name of `PointSensor` section.

3.5.3 TimeAverage section

FDTD solver stores average value of the given quantities in the cuboid box defined by diagonally opposite points with the coordinates given by `Position`. Time averaging is performed by averaging at each n^{th} time-step, where n is `TimeSteps`, starting from `TimeBegin`, ending at `TimeEnd`. The quantities to be stored are listed by the keyword `Quantities` as a list of strings. The data is stored in an hdf5 file named '<out>_TimeAvg_<secname>_fdtd.h5'. Here <secname> is the name of the `TimeAverage` section. A xdmf file named '<out>_TimeAvg_<secname>_fdtd.xdmf' is also written for visualizing the data with paraview.

3.5.4 PhaseCalculator section

The `PhaseCalculator` object of FDTD solver calculates phase and amplitude of the given quantities at a time-step just after time given by `Time` in the cuboid box defined by diagonally opposite points with the coordinates given by `Position`. The quantities to be stored are

listed by the keyword `Quantities` as a list of strings. The data is stored in an hdf5 file named '<out>_Phase_<secname>_fdtd.h5'. Here <secname> is the name of the `PhaseCalculator` section. A xdmf file named '<out>_Phase_<secname>_fdtd.xdmf' is also written for visualizing the data with paraview.

3.6 Detector section

The `Detector` object of FDTD solver checks if time variation of *all* the given quantities has stabilized below the tolerance given by `Tolerance`. Once this condition is reached, time-stepping loop ends. Testing is performed after time specified by `StartTime` in the cuboid window (or rectangular window in 2D simulations). The quantities whose tolerance is tested are listed by the keyword `Quantities` as a list of strings. To accelerate this step, one can state that tolerance is tested only on the grid formed by every n^{th} x-, y-, and z- coordinate, where n is specified by `StepX`, `StepY`, or `StepZ`, respectively.

3.7 Running FDTD Simulations

In this section, the above config file is used to perform mode calculations on a lateral 2D cross section of a waveguide shown in Fig. 3.3. The waveguide is aligned along the lateral direction (X-axis). It is invariant along Z dimension (normal to the plane of the figure). The waveguide structure can be created using the command

```
>> OptoSolver str fdtdSiWG_str.cfg.
```

It is not necessary to generate the structure before simulating it. The config file for generating the structure ('fdtdSiWG_str.cfg') can be specified as `Device` in `File` section of the mode solver config file 'fdtdSiWG_dev.cfg'. The solver internally generates the structure and passes it to the FDTD solver. The structure config file must also be present in the same folder. Once the config file is set, the FDTD calculations can be performed using the following command

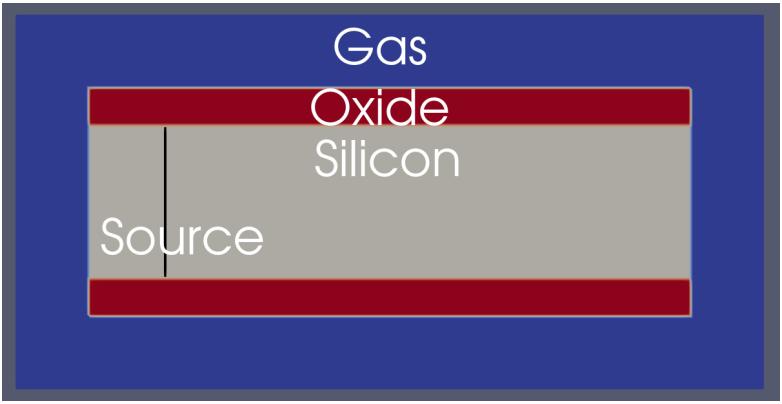


Figure 3.1: Structure of the lateral cross-section of the simulated waveguide along X axis. The waveguide is invariant in Z direction normal to the plane of the figure.

```
>> OptoSolver fdtdsolver fdtdSiWG_dev.cfg
```

3.8 Visualizing Results

The FDTD calculations will generate various hdf5 (extension *.h5) and corresponding xdmf files (extension *.xdmf), depending on which data visualizer/postprocessor object among those listed in Section 3.5. The xdmf files can be opened in paraview for data visualization. Also, the `PointSensor` object saves data in a 'csv' file which can be viewed with any 'csv' viewer.

3.8.1 Average calculator

Average calculator stores time-averaged spatial distribution of the specified quantities in the hdf5 file 'SiWG_TimeAvg_AvgMid_fdtd.h5'. An xdmf script file 'SiWG_TimeAvg_AvgMid_fdtd.xdmf' is also generated to visualize the data in paraview as follows.

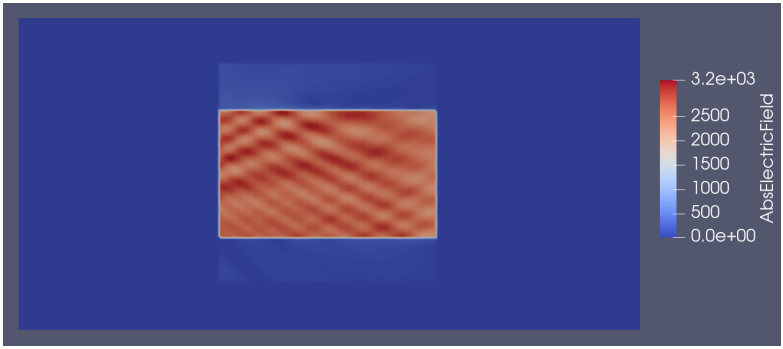


Figure 3.2: Time-averaged magnitude of Electric field

```
>> paraview SiWG_TimeAvg_AvgMid_fdtd.xdmf
```

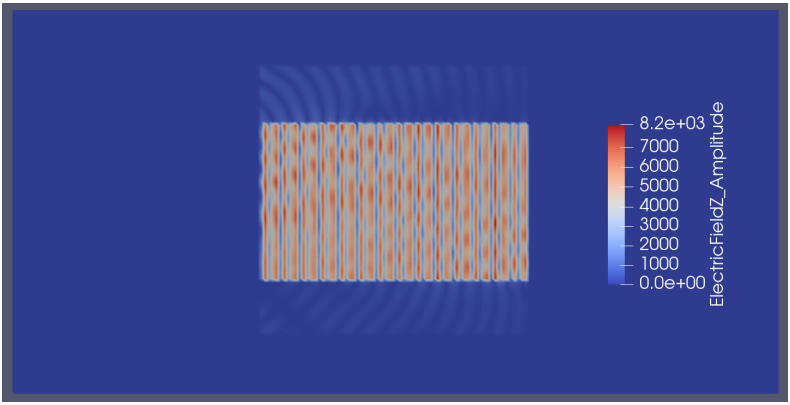
It opens paraview, in which one can plot spatial distribution of time-averaged magnitude of electric field ($|\vec{E}|$) by clicking `Apply` and then selecting `AbsElectricField` from the drop-down list on the top panel. As shown in Fig. 3.2, since the waveguide exhibits negligible loss, $|\vec{E}|$ is spatially uniform in Silicon region of the waveguide. The emitted plane wave travels only a short distance during the simulation time. Therefore, $|\vec{E}|$ is zero in the region ahead of the wavefront. Small spatial variations in $|\vec{E}|$ are visible in the figure. They arise from small internal reflections at oxide-Silicon surface.

3.8.2 Phase calculator

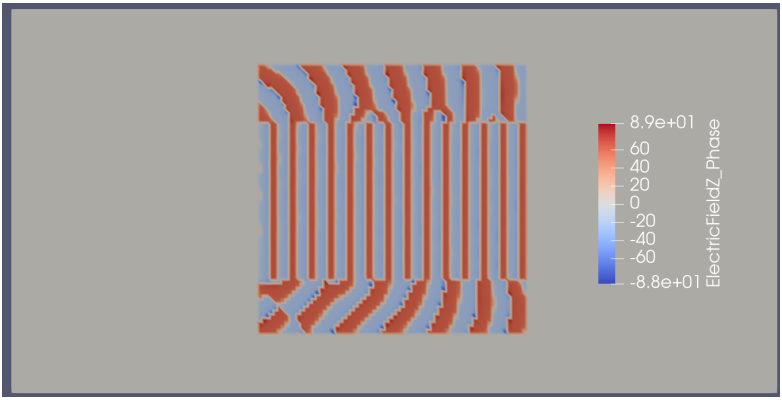
Phase calculator stores time-averaged spatial distribution of the specified quantities in the hdf5 file 'SiWG_Phase_Ph_fdtd.h5'. An xdmf script file 'SiWG_Phase_Ph_fdtd.xdmf' is also generated to visualize the data in paraview as follows.

```
>> paraview SiWG_Phase_Ph_fdtd.xdmf
```

Fig. 3.3(a) plots E_z at a specific time point. The snapshot shows sinusoidal variation of E_z between the source and the wavefront. Fig. 3.3(b) plots spatial variation of the phase of E_z. It varies linearly



(a) Magnitude of Electric Field



(b) Phase of Electric Field

Figure 3.3: Magnitude and Phase of Electric Field in Z direction.

between −90 deg to 90 deg and back to −90 deg in each sinusoid of the waveform.

3.8.3 Movie

Phase calculator stores snapshots of the specified quantities in the XY plane at each time-step in an hdf5 file named 'SiWG_XYEField_fdtd.h5'. An xdmf script file 'SiWG_XYEField_fdtd.xdmf' is also generated to visualize the data in paraview as follows.

```
>> paraview SiWG_XYEField_fdtd.xdmf
```

The movie can be played by first clicking `Apply` and then selecting `ElectricFieldZ` from the drop-down list on the top panel. After that, the `play` symbol on the top panel can be clicked to view the stored snapshots as a movie in paraview

3.8.4 Point sensor

Point sensor stores time-variation of the specified quantities at a specific location in a csv file 'SiWG_PointData_Xid_95_Yid_53_Zid_0.csv'. Data in this file can be imported in any csv plotting tool such as gnuplot, qtiplot, or excel.

Chapter 4

Configuring FDTD Solver

Example configuration files provided in Chapters 3, **??**, and **??** list typical configurations of FDTD solver for FDTD simulations, ScMat calculations, and broad-band ScMat calculations, respectively.

In the chapter, a list of all the available configurations in the FDTD solver and their usage information is provided. For clarity, the keywords are listed section-wise. The mandatory keywords are marked 'mandatory'. Optional keywords are provided with the default input values.

4.1 Keywords in File Section

Following keywords must be listed in `File` section of FDTD solver config file.

- `Device`: (Mandatory) Specify either a config file for structure generation or a saved mesh file in hdf5 format.

- `Out`: (Mandatory) Specify prefix of the output csv, xdmf and hdf5 files.

4.2 Keywords in Solver Section

Following keywords may be listed in `Solver` section of FDTD solver config file.

- `MaximumTime`: (Mandatory) Maximum time in Seconds till FDTD time-stepping is performed.
- `GridType`: (Mandatory) In 2D, type of Grid used for FDTD simulations. If the grid is transverse magnetic ('TM'), E_z, H_x, H_y are computed, where as if the grid is transverse electric ('TE'), E_x, E_y, H_z are computed. This keyword also sets mode polarization, if mode source is used for calculating beam shape.
- `Settings`: Additional settings applied to the grid are listed as strings.
    1. `DisableTFSFOnSideFaces`: Boundary box of Total Field Scattered Field (TFSF) simulations is disabled on the side faces
    2. `WavelengthDepIndex`: Real and imaginary dielectric constants are determined from the table of wavelength dependent complex refractive indices specified in the material config file. Linear interpolation is used if the wavelength is not in the list. If the wavelength is outside of the table range, then the dielectric constants are set to smallest/largest frequency.
    3. `ActivateAllDispersiveMedia` : Dispersive material model parameters are provided in each of the material parameter file. For each region, if the material is dispersive, the dispersive model is automatically activated, if this flag is set.
    4. `CalculateAdjustedFields` : Electric and magnetic fields are calculated on the 'Yee' grid. Thus, the two are spatially shifted by half spacing. If this flag is set, magnetic fields are interpolated to be on the same location as electric fields. Note, that this operation is computationally expensive, therefore, must be used only when absolutely necessary.

4.3 Keywords in Source section

The source can be of type 1. `PlaneWave`, 2. `Dipole`, or 3. `PEC`. Input parameters depend on the type of source selected.

4.3.1 Plane-wave source

- `Position`: (Mandatory) Defines a window from which the light source is emitted into the simulation domain. It is a '(...,)' list of two points (each point is specified as list of three floating point numbers). For example,

  ```
  Position: ([-0.58, -0.25, 0.0], [-0.58, 0.25, 0.])
  ```

  In a 3D structure, the two points form diagonally opposite points of an 'axis-aligned' rectangle. An axis-aligned rectangle has all of its sides parallel to one of the three axes. In a 2D structure, the two coordinates define an axis-aligned segment.

- `BeamShape`: Spatial shape of the input beam in the source window defined above. It can be – 1. `Uniform`, 2. `ModeBeam`, or 3. `GaussianBeam`. By default, a uniform beam is used.

- `Theta`: (Mandatory) Polar angle of wave direction of the plane wave source.

- `Phi`: (Mandatory) Azimuthal angle of wave direction of the plane wave source.

- `Intensity`: (Mandatory) Intensity of the source (W/m^2).

- `NRise`: Number of time-steps till the source begins to emit at full intensity. It must be a positive integer. Default value is 0.

- `Delay`: Number of time-steps after which the source begins to emit. It can take negative values. Default value is 0.

- `Wavelength:` (Mandatory) Wavelength of the source is specified in the units of μm.

- `EffectiveIndex:` (Mandatory) Effective refractive index of the waveguide around which the waveguide modes are searched at the source locations. It is applicable *only if* `ModeBeam` is used.

- `PowerMethodTol:` Applicable only when `ModeBeam` is used and power method is used for mode calculations. When difference between the eigenvalues between successive iterations is less than the tolerance then the calculations are terminated. Default value is 10^{-8}.

- `PowerMethodMaxIter:` Applicable only when `ModeBeam` is used and power method is used for mode calculations. Maximum iterations for mode calculations using the power method. Default value is 50.

- `Flags:` Various flags are set by providing a list of appropriate keywords as comma-separated strings. Following flags can be used.

    1. `UsePowerMethodModeSolver`: A faster power method is used for mode calculations.
    2. `ModeInBoundingBox`: Only cross-section of the structure in the source window is used for mode calculations, instead of the entire cross-section of the structure in the plane of the source.
    3. `VectorialModeEquation`: Vectorial equation is used for mode calculations instead of scalar equation.

- `BeamCenter`: Mandatory only when `GaussianBeam` is used. Center of Gaussian beam.

- `BeamRadius`: Mandatory only when `GaussianBeam` is used. Radius of Gaussian beam at the center.

4.3.2 Dipole source

A dipole source is instantiated.

- `DipoleCenter`: (Mandatory) Location of dipole source.

- `Theta`: (Mandatory) Polar angle of wave direction of the dipole.
- `Phi`: (Mandatory) Azimuthal angle of wave direction of the dipole.
- `Intensity`: (Mandatory) Intensity of the dipole source.
- `Delay`: Number of time-steps after which the source begins to emit. It can take negative values. Default value is 0.
- `Spread`: Standard deviation (in number of time-steps) of temporally Gaussian or Ricker-Wave dipole source.

4.3.3 PEC source

A PEC is instantiated. The PEC is characterized by the index of the region in which the PEC must be instantiated.

4.4 Dispersive section

Metallic materials exhibit complex refractive index which shows strong dispersion near specific frequencies. In FDTD solver, ADE method is used to model highly dispersive materials such as Aluminum. Following keywords are used to instantiate dispersive materials.

- `Region`: Name of the region in which the model is active. Each region needs to have a separate section.
- `Model`: One of the following models can be selected for dispersive material. 1. `Drude`, 2. `Debye`, 3. `Lorentz`, 4. `Kerr`,. Note, that input parameters may be different for different models.
- `NumberPoles`: Number of poles to be used for modeling the selected model.
- `DeltaEpsi`: A list of prefactors multiplied to each of the poles. It is required for `Debye` and `Lorentz` models.
- `PoleFreq`: A list of pole frequencies in Hz. It is required for `Drude` and `Lorentz` models.

- `DampFact`: A list of damping factors. It is required for `Drude` and `Lorentz` models.

- `TauPole`: A list of time constants. It is required for `Debye` model.

4.4.1 Kerr model

Materials which show Kerr effect may also have Lorentz poles or Debye poles. Therefore `Kerr` model requires a special set of keywords which are listed below.

- `LorentzPoleFreq`: A list of Lorentz-type pole frequencies in Hz.

- `LorentzDamping`: A list of Lorentz-type damping factors

- `LorentzDeltaEpsi`: A list of prefactors multiplied to each of the Lorentz poles.

- `DebyeTauPole`: A list of time constants in Debye model.

- `DebyeDeltaEpsi`: A list of prefactors multiplied to each of the Debye poles.

- `KerrXi3`: A list of prefactors of instantaneous Kerr effect.

- `InstantaneousKerrFactor`: A list of numbers ($\in [0., 1.]$) which determine proportion of instantaneous Kerr effect to Kerr-Raman effect.

- `OpticalPhononFreqency`: A list of optical phonon frequencies (in Hz) for Raman effect.

- `RamanBandwidth`: A list of Raman band-width numbers (in Hz) for Raman effect.

Each of the list must have the same number of entries as `NumberPoles`. If any one of the Lorentz (Debye) parameter list is not defined, then Lorentz (Debye) pole will be deactivated.

4.5 Keywords in Boundary section

In the config file, a Boundary section is instantiated by the name 'Boundary*<name>', where <name> represents the boundary name. Multiple boundary definitions with different names can be instantiated in a config file. Following keywords can be defined in `Boundary` section.

- `Axis`: Current boundary definition is applied to the list of boundaries specified by `Axis`. Available sets of boundary names are –

  1. `Xmin`: YZ boundary at the smallest x-coordinate.
  2. `Xmax`: YZ boundary at the largest x-coordinate.
  3. `Ymin`: XZ boundary at the smallest y-coordinate.
  4. `Ymax`: XZ boundary at the largest y-coordinate.
  5. `Zmin`: XY boundary at the smallest z-coordinate.
  6. `Zmax`: XY boundary at the largest z-coordinate.

  The `Axis` are listed in the following format. `Axis = ["Ymin", "Ymax"];`

- `Model` Boundary type to be applied to the current boundary condition. Following boundary models are recognized by the FDTD solver – 1. CPML 2. RBC 3. RBC. Separate boundary sections must be used for instantiating different boundary models.

4.6 Movie section

FDTD solver stores given set of quantities on a 2D grid or a 2D cross-section of a 3D grid at each time-step in hdf5 format. An xdmf file is also written to visualize the data as a movie in paraview. The files are named, '<out>_<secname>_fdtd.h5' and '<out>_<secname>_fdtd.xdmf', respectively. Here <secname> is the name of the `Movie` section. Following keywords can be listed in Movie section.

- `Plane`: Plane of 2D cross-section of a 3D grid. Available Values are – 1. `XY` 2. `XZ` 3. `YZ`. In case of 2D simulations, the 2D grid itself is used.

- `Intercept`: Intercept of 2D cross-section of the 3D grid.
- `Quantities`: Comma separated list of quantities to be stored.

4.7 PointSensor section

When `PointSensor` object is defined, FDTD solver stores the value of the quantities at each time step at a single point on the FDTD grid nearest to the user specified point. They are stored in 'csv' format in the file named '<out>_<secname>_fdtd.csv', where <secname> is name of `PointSensor` section. Following keywords can be listed in this section.

- `Position`: Coordinate of the point as a list of three floating points.
- `Quantities`: Comma separated list of quantities whose values at a single point are to be stored.

4.8 TimeAverage section

When a `TimeAverage` object is defined, FDTD solver calculates average of the required quantities in a bounding box given by the user. The data is stored in an hdf5 file named '<out>_TimeAvg_<secname>_fdtd.h5'. Here <secname> is the name of the `TimeAverage` section. A xdmf file named '<out>_TimeAvg_<secname>_fdtd.xdmf' is also written for visualizing the data with paraview. Following keywords can be listed.

- `Position`: List of coordinates of two points which define an axis-aligned cuboid (in 3D), or an axis-aligned rectangle (in 2D).
- `TimeBegin`: Time averaging begins at this time (in Seconds).
- `TimeEnd`: Time averaging ends when time exceeds this time (in Seconds).
- `TimeSteps`: To reduce computational burden, every n^{th} step is added to the average. n is specified as an integer here.
- `Quantities`: Comma separated list of quantities whose average is to be calculated.

4.9 PhaseCalculator section

When `PhaseCalculator` object is defined, FDTD solver calculates phase and amplitude of the required quantities at a user-provided time. The data is stored in an hdf5 file named '<out>_Phase_<secname>_fdtd.h5'. A xdmf file named '<out>_Phase_<secname>_fdtd.xdmf' is also written for visualizing the data with paraview.

- `Position`: List of coordinates of two points which define an axis-aligned cuboid (in 3D), or an axis-aligned rectangle (in 2D).
- `Time`: Time (in Seconds) at which phase and amplitude of the given quantities is to be calculated.
- `Quantities`: Comma separated list of quantities whose average is to be calculated.

4.10 Detector section

The `Detector` object of FDTD solver checks if time variation of *all* the given quantities has stabilized below the tolerance given by `Tolerance`. If yes, FDTD simulations are stopped. Following keywords can be defined in this section.

- `Position`: List of coordinates of two points which define an axis-aligned cuboid (in 3D), or an axis-aligned rectangle (in 2D) window. The detector works in this window only.
- `StartTime`: Time (in Seconds) from which detection of stabilized quantities begins.
- `Quantities`: Comma separated list of quantities which are to be checked if stabilized or not.
- `Tolerance`: Average variation in the quantities must be less than the tolerance.
- `StepX`: Specifies n_x. Every n_x^{th} grid point along X axis is checked.
- `StepY`: Specifies n_y. Every n_y^{th} grid point along Y axis is checked.
- `StepZ`: Specifies n_z. Every n_z^{th} grid point along X axis is checked.

Chapter 5

Python Interface

The Opto-solver package provides a python module to perform FDTD, BPM, and mode simulations using a python script. The package also provides commands to retrieve simulation results. Together with `tensormesher` python interface, it enables users to construct a device, simulate it, and post-process the results using a python script. This would come handy for structure optimization for specific applications.

This chapter describes python interface of the FDTD solver.

> **Note:** Whenever possible, please use config file to setup the FDTD solver object. Providing config file ensures that all the data are input in the correct order.

5.1 Import modules

Python modules of the Opto-solver and the tensor-mesher package are imported using the following script.

```
import numpy as np
import cutensormesher as m
import cuoptosolver as s
```

Note, that if you have downloaded hardware-accelerated version of the optosolver, then you must import the modules with prefix `cu` as shown above. Else, import `tensormesher` and `optosolver`. Do not mix them.

5.2 Constructors

Three constructors are provided for the FDTD-solver, as follows.

- `s.fdtdsolver(Device=dev)` constructs the solver object by taking `m.device` object `dev` provided by the tensor-mesher as an input.
- `s.fdtdsolver(CmdFile="file.cfg")` constructs the solver object by parsing config file of the FDTD-solver. Note, this is exactly the same file as described in Chapter 3. It also imports device structure or mesh from the `File`section.
- `s.fdtdsolver(CmdFile="file.cfg",Device=dev)` constructs the solver object by parsing config file of the FDTD-solver. Note, this is exactly the same file as described in Chapter 3, except that device given as an argument is used in the solver..

5.3 Setup the solver

The commands given below are called on the `fdtdsolver` object. They setup the solver, e.g. add sources, specify boundary conditions, etc.

Note: While modifying the solver object, it is strongly recommended to use the following commands in the same order in which they are listed below. For example, set global parameters, then set boundary conditions, and then add sources.

5.3.1 setGlobalParameters

The command `setGlobalParameters()` sets solver various parameters on a global scope. It takes the following arguments.

1. `NumericParams` : A python dictionary mapping parameter name to its numeric value. The following numeric parameters can be supplied.
    - `MaximumTime` : Maximum simulation time. If a `Detector` is set, then simulations may finish earlier.

2. `StringParams` : A python dictionary mapping parameter name to a string. The following string parameters can be supplied.

    - `GridType` : Possible alternatives - "TE" for TE grid or "TM" for TM grid.

3. `Settings` : A python list of strings is suplied. Following strings are recoginized.

    - `DisableTFSFOnSideFaces`: Boundary box of TFSF simulations is disabled on the side faces
    - `WavelengthDepIndex`: Real and imaginary dielectric constants are determined from the table of wavelength dependent complex refractive indices.
    - `ActivateAllDispersiveMedia` : Dispersive material model parameters are provided in each of the material parameter file. For each region, if the material is dispersive, the dispersive model is automatically activated, if this flag is set.
    - `CalculateAdjustedFields` : Electric and magnetic fields are calculated on the 'Yee' grid. Thus, the two are spatially shifted by half spacing. If this flag is set, magnetic fields are interpolated to be on the same location as electric fields. Note, that this operation is computationally expensive, therefore, must be used only when absolutely necessary.

5.3.2 setDomainBoundary

The command `setDomainBoundary(...)` takes the following arguments -

1. `Model`: type of domain boundary to be set. Possible alternatives are - CPML, RBC, and RBC.

2. `Axes`: axes at which the above specified boundary is to be set are provided as a list of strings. Possible alternatives are - "Xmin", "Xmax", "Ymin", "Ymax", "Zmin", "Zmax".

3. `NumericParams`: A python dictionary mapping parameter name to its numeric value . The following numeric parameters can be supplied.
    - PMLLayers: number of PML layers at each boundary face.
    - amax : maximum "a" parameter of the "CPML" model.
    - kappamax: maximum κ parameter of the "CPML" model.
    - sigmamax: maximum σ parameter of the "CPML" model.
4. `Flags`: An empty list. This parameter is currently unused.

5.3.3 resetDomainBoundaryToReflective

`resetDomainBoundaryToReflective()` command resets all the domain boundaries to "reflective" domain boundaries.

5.3.4 addSource

The command `addSource(...)` takes the following arguments-

1. `Name` : Name of the source
2. `NumericParams` : A python dictionary mapping parameter name to its numeric value . The following numeric parameters can be supplied.
    - Theta: Polar angle (deg)
    - Phi: Azimuthal angle (deg)
    - Wavelength: source wavelength (μm)
    - EffectiveIndex: initial index of the mode-beam source
    - NRise: rise time in number of periods
    - Intensity: source intensity (W/m^2)
    - Delay: delay in number of periods
    - BeamRadius: if "GaussianBeam" source is used, specify beam radius.
    - Spread: if "Dipole" source is used, specify standard deviation of temporally Gaussian shape.

3. `StringParams`: A python dictionaly mapping parameter name to its string value. The following string parameters can be supplied.

    - Type: Possible alternatives are "PlaneWave", "Diople", "PEC".
    - BeamShape: Possible alternatives are "ModeBeam", "UniformBeam", "GaussianBeam".
    - Region: if "PEC" is used, then specify the region name in which PEC is active.

4. `NumericListParams`: A python dictionary mapping parameter name to a list of numeric values. The following string parameters can be supplied.

    - WindowMin: Botton-left point of the source window as a list of three numbers.
    - WindowMax: Top-right point of the source window.
    - DipoleCenter: Location of the dipole, for "Dipole" source.
    - WaveDir: Wave propagation direction for "PlaneWave" source
    - DipoleDir: Dipole direction for "Dipole" source.

5. `StringListParams`: A python dictionary mapping parameter name to a list of strings. The following string parameters can be supplied.

    - Flags: Various flags can be listed e.g. "UsePowerMethodModeSolver", "ModeInBoundingBox", "VectorialModeEquation".

5.3.5 removeSource

The command `removeSource(...)` takes name of the source as an input parameter and deletes the source from the solver.

5.3.6 addDispersiveMedium

The command `addDispersiveMedium(...)` takes the following arguments.

1. `Name` : Name for reference.
2. `NumericParams` : A python dictionary mapping parameter name to its numeric value. The following numeric parameters can be supplied.
   - `NumberOfPoles` : number of poles in the specified model.
3. `StringParams` : A python dictionaly mapping parameter name to its string value. The following string parameters can be supplied.
   - Model: Possible alternatives are "Debye", "Drude", "Lorentz", and "Kerr".
   - Region: Name of the region in which this model is active.
4. `NumericListParams` : A python dictionary mapping parameter name to a list of numeric values. The following parameters can be supplied.
   - TauPole: A list of pole τ(sec).
   - DeltaEpsi: A list containing prefactor each of the poles.
   - PoleFreq: A list of pole frequencies (Hz). It is internally converted to rad/sec by multiplying with 2π.
   - DampFact: A list of damping factors of the poles (Hz). It is *not* converted to rad/sec.

   If `Kerr` model is used, the following parameters for each of the poles must also be provided instead.
   - LorentzPoleFreq
   - LorentzDamping
   - LorentzDeltaEpsi
   - DebyeTauPole

- DebyeDeltaEpsi
- KerrXi3
- InstantaneousKerrFactor
- OpticalPhononFreqency
- RamanBandwidth

5. `StringListParams`: Currently not used.

5.3.7 removeDispersiveMedium

The command `removeDispersiveMedium()` takes reference name of the dispersive medium and deletes it from the solver.

5.3.8 addVisualiser

The command `addVisualiser()` adds various visualizations (e.g. point sensor, movie) to the solver. It takes the following arguments.

1. `Name` : Name of the visualization instance.

2. `Type` : Visualization type. Possible alternatives are "PointSensor", "Movie", "TimeAverage", "PhaseCalculator", "IntensityCalculator", and "Detector".

3. `NumericParams` : A python dictionary mapping parameter name to its numeric value. Different numeric parameters are supplied for different visualizers as given below.

   - "PointSensor" : None.
   - "Movie" : `Intercept` of the plane.
   - "TimeAverage" : The parameters `TimeBegin`, and `TimeEnd` specify start- and end-time of the averaging, while `TimeStep` specify n where averaging is done every n^{th} step.
   - "PhaseCalculator": `Time` at which phase is to be stored.
   - "IntensityCalculator" : `Time` at which intensity is to be calculated and stored.

- "Detector" : `StartTime` start time of detection. `StepX`, `StepY`, and `StepZ` specify n where probe every n^{th} point along each direction. `TimeStep` specify n where averaging is done every n^{th} step. `Tolerance` specifies minimum fractional difference between the current and previous values of the field below which the simulation will be terminated.

5.3.9 removeVisualiser

The command `removeVisualiser()` takes the name of the visualiser and deletes it from the solver.

5.4 Solve FDTD System

The following commands begin the simulations.

5.4.1 portToGPU

The command `portToGPU(...)` ports the simulations to the first available GPU (with GPU id = 0). If you wish to port to another GPU, please give the GPU id as an argument. This command must be run before solving the system on the GPU.

5.4.2 solveSystem

The command `solveSystem(...)` begins FDTD simulations on the *CPU*. Make sure you *did not* run `portToGPU` command before running this command.

5.4.3 solveSystemGPU

The command `solveSystemGPU(...)` begins FDTD simulations on the *GPU*. Make sure you *have* run `portToGPU` command before running this command.

5.5 Retrieve visualizer data

Once the simulation is finished, the following commands help in accessing the data stored by the visualizers. Note, that in all the commands, the visualizer is identified by its name.

5.5.1 getPointSensorDatasetName

The command `getPointSensorDatasetName(...)` takes `Name` of the point sensor object as an input and returns the names of all the datasets whose data is recorded at the given point.

5.5.2 getTimeAverageDatasetName

The command `getTimeAverageDatasetName(...)` takes `Name` of the time-average object as an input and returns the names of all the datasets for which time-average is calculated.

5.5.3 getPhaseCalculatorDatasetName

The command `getPhaseCalculatorDatasetName(...)` takes `Name` of the phase calculator object as an input and returns the names of all the datasets whose phase is calculated.

5.5.4 getIntensityCalculatorDatasetName

The command `getIntensityCalculatorDatasetName(...)` takes `Name` of the phase calculator object as an input and returns the names of all the datasets whose phase is calculated.

5.5.5 getPointSensorData

The command `getPointSensorData(...)` takes `Name` of the point sensor object as an input and returns time varying values of the stored datasets as a 2D "NumPy" array. Leading dimension is a time dimension and trailing dimension stores dataset values for each of the datasets and time.

5.5.6 getTimeAverageData

The command `getTimeAverageData(...)` takes `Name` of the time-average object as an input and returns time-averaged values of the stored datasets at each vertex as a 2D "NumPy" array. Leading dimension has number of dataset entries and trailing dimension stores dataset values at each vertex for each of the datasets.

5.5.7 getPhaseCalculatorAmplitude and getPhaseCalculatorPhase

The commands `getPhaseCalculatorAmplitude(...)` and `getPhaseCalculatorPh` take `Name` of the phase-calculator object as an input. They returns values of amplitude and of phase of the stored datasets at each vertex as a 2D "NumPy" array. Leading dimension has number of dataset entries and trailing dimension stores amplitude/phase dataset values at each vertex for each of the datasets.

5.5.8 getIntensityCalculatorData

The command `getIntensityCalculatorData(...)` takes `Name` of the intensity-calculator object as an input and returns values of the stored datasets at each vertex as a 2D "NumPy" array. Leading dimension has number of dataset entries and trailing dimension stores dataset values at each vertex for each of the datasets. Names of the stored dataset are given by `getIntensityCalculatorDatasetName` function.

Appendix A

Notation and Acronyms

Acronyms

ABC	Absorbing Boundary Condition
ADE	Auxiliary Differential Equation
BC	Boundary Condition
BPM	Beam Propagation Method
MOSFET	Metal-Oxide Semiconductor Field Effect Transistor
CPML	Convolutional Perfectly Matching Layer
FDTD	Finite Difference Time Domain
GPU	Graphics Processing Unit
LHS	Left Hand Side
RBC	Periodic Boundary Condition

PEC	Perfect Electric Conductor
PML	Perfectly Matching Layer
RBC	Reflecting Boundary Condition
RHS	Right Hand Side
ScMat	Scattering Matrix
TE	Transverse Electric
TFSF	Total Field Scattered Field
TM	Transverse Magnetic

Bibliography

Curriculum Vitae

Saurabh Sant
Birth: 11th November 1989 in Mumbai, India
Citizenship: India
Languages: English (fluent), German (A1), Marathi (native), Hindi (fluent)

Education:

- February 2014 – April 2018
  Ph. D. in Electrical Engineering at ETH Zürich
  under the supervision of Prof. Andreas Schenk

- September 2012 – December 2013
  Master of Science in Electrical Engineering at ETH Zürich
  Masters Thesis under the supervision of Dr. Heike Riel and Prof. Andreas Schenk on "Template-based fabrication of InGaAs nanowires for TFET devices"

- August 2007 – May 2011
  Bachelor of Technology in Electrical Engineering at IIT Bombay, Mumbai, India.

Work experience:

- June 2017 – August 2017
  Internship on "TCAD Modeling of Photonic and Plasmonic LASERs-On-Chip"
  IBM Research GmbH, Rüschlikon

- July 2011 – August 2012
  Research Assistant on "Mobility Modeling of Unstrained and Strained Ge Channel pMetal-Oxide Semiconductor Field Effect Transistor (MOSFET)" Department of Electrical Engineering, IIT Bombay, Mumbai, India.