ALiS: A Light Simulator

A library for FDTD simulation

Yongsop Hwang

Written by Yongsop Hwang (yongsop.hwang@unisa.edu.au)
Reviewed by Ho-Seok Ee (hsee@kongju.ac.kr)

This material is free to be distributed.
Document version: 0.02
Last updated: 14th February, 2025

Contents

Introduction

Welcome to ALiS in Wonderland!

ALiS (**A** **Li**ght **S**imulator) is a numerical simulation package of C language library based on the Finite-Difference Time-Domain (FDTD) method [1], developed by Ho-Seok Ee at Kongju National University, South Korea.

This document is an instruction manual of ALiS to guide users from installation to applications. There exists a manual written by Ee in Korean [2] which is recommended to be considered as a primary instruction material. The objective of this manual is to provide guidance to non-Korean speaking users as well as to supplement the primary manual with additional information. Naturally, this material has some contents duplicated and translated from the ALiS manual in Korean [2].

Note that this document is incomplete and being continuously updated. ALiS version described in the document is 1.2.0.

Advantages

ALiS has great advantages in the two following aspects compared to other FDTD simulation tools.

- Non-uniform grid
- Dispersive materials

ALiS supports a non-uniform grid mesh structure so that a higher resolution can be implemented in the region of interest while setting a lower resolution for a less important background region to significantly improve the calculation speed.

ALiS provides a comprehensive model to implement dispersive materials including simple conductivity, Drude model, Lorentz model, and the critical-points model. The dispersion curves of most existing materials can be models using one of the models for a desired wavelength range. A material library is also provided for frequently used materials in UV, visible, and near-infrared ranges. Additionally, users can fit the dispersion curve of the material of their choice using one of the provided models.

Additionally, ALiS supports a parallel simulation in a multicore CPU by OpenMP which is included in the compiler.

Validation

The validity of ALiS has been tested in more than a hundred publications for various structures and devices which include metalenses, nanowires, plasmonic waveguides, photonic crystals, cavities, and nanolasers.

Here are some examples.

- Ho-Seok Ee, Kyung-Deok Song, Sun-Kyung Kim, and Hong-Gyu Park. Finite-difference time-domain algorithm for quantifying light absorption in silicon nanowires. *Israel Journal of Chemistry*, 52(11-12):1027–1036, 2012
- Yongsop Hwang, Min-Soo Hwang, Won Woo Lee, Won Il Park, and Hong-Gyu Park. Metal-coated silicon nanowire plasmonic waveguides. *Applied Physics Express*, 6(4): 042502, 2013
- Yongsop Hwang and Hong-Gyu Park. Geometric dependence of metal-coated silicon nanowire plasmonic waveguides. *Journal of Optics*, 16(2):025001, 2014
- Yoon-Ho Kim, Soon-Hong Kwon, Ho-Seok Ee, Yongsop Hwang, You-Shin No, and Hong-Gyu Park. Dependence of Q factor on surface roughness in a plasmonic cavity. *Journal of the Optical Society of Korea*, 20(1):188–191, 2016
- Taesu Ryu, Moohyuk Kim, Yongsop Hwang, Myung-Ki Kim, and Jin-Kyu Yang. High-efficiency SOI-based metalenses at telecommunication wavelengths. *Nanophotonics*, 11(21):4697–4704, 2022

Citation

ALiS can be cited in a published material as:

> Ho-Seok Ee, *A Light Simulator based on the FDTD method*, GitHub repository, https://github.com/hseelab/alis, (Version 1.2.0), 2022.

This user manual can be cited as:

> Yongsop Hwang, *ALiS: A Light Simulator - A library for FDTD simulation*, https://github.com/scigg/alis_manual, 2025.

Chapter 1

Installation

The ALiS package is freely available on GitHub (https://github.com/hseelab/alis).

1.1 Under a user account in a server

ALiS can be downloaded to a user directory in a Linux system by the following command.

```
$ git clone https://github.com/hseelab/alis.git
```

Once you download the package, you will have the following items in your directory.

```
$ ls
Makefile   bin   doc   examples   src
```

Here are some of the required libraries.

- hdf5
- szip

You may need to load these libraries if they are already installed in a shared Linux server (*e.g.* NCI Gadi), or you need to install them in your home directory. To load you may need to do as follows:

```
$ module load hdf5
$ module load szip
```

The loaded modules can be seen by the following command (in NCI Gadi).

```
$ module list
Currently Loaded Modulefiles:
```

```
1) pbs   2) hdf5/1.10.7   3) szip/2.1.1
```

Then do the common configuring and making process as follows:

```
$ ./configure
$ make
make install
```

For convenience, add the path to the binary directory of ALiS in your environment file. For instance, you add the following line in your `.bashrc`.

```
export PATH="/home/username/alis/bin:$PATH"
```

Then run the following commands to make sure the path is properly added.

```
$ source .bashrc
$ echo $PATH
```

Now, run ALiS to check the installation is complete.

```
$ alis
ALiS: A Light Simulator
Usage: alis FILE [OPTIONS]
```

Congratulations! Now you are almost ready to run some exciting simulations for completely free.

1.2 For all users (Ubuntu)

ALiS can also be installed for all users. First, move to the directory `/usr/local/src` and do the followings.

```
$ sudo git clone https://github.com/hseelab/alis.git
$ cd alis
$ sudo make
```

If you are asked to install the hdf5 library, then install it using the following commands.

```
$ sudo apt-get update
$ sudo apt-get install libhdf5-serial-dev
```

Then the `<hdf5.h>` will be able to be found under `/usr/include/hdf5/serial`. You may also find that `<png.h>` is required.

```
$ sudo apt-get install libpng-dev
```

If all the required libraries are installed, then run `make` again.

```
$ sudo make
```

Now add the path for all users by editing `/etc/environment`.

```
$ sudo vi environment
```

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:
/usr/local/src/alis/bin"
```

Now activate the new environment and check the path.

```
$ source /etc/environment
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
/bin:/usr/games:/usr/local/games:/usr/local/src/alis/bin
```

Run ALiS with a user account to check the installation is complete.

```
$ alis
ALiS: A Light Simulator
Usage:  alis FILE [OPTIONS]
```

Congratulations! Now you are almost ready to run some exciting simulations for completely free.

1.3 Test run: Dipole band

Well, alright. It is probably too early to celebrate when it has not been tested yet. Let's test with some example files. You can find example files under the directory `examples`.

```
$ cd examples
$ ls
Bowtie_Nanoantenna.c   Dipole_PML.c   Metasurface_Farfield.c
...
Dipole_Band.c   Light_Emitting_Diode.c
```

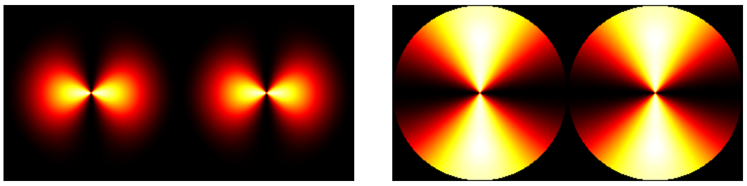
You can choose one of the examples and test if ALiS runs correctly. For example, with the `Dipole_Band.c` example, you can do as follows.

```
$ alis Dipole_Band.c
$ ./Dipole_band
[00:00:06/00:00:06] [6400/6400: 100.0%]
Performing Near-to-Far-Field Transformation.
[400:00:10/00:00:10] [450264/50264: 100.0%]
Total radiated power: 1.003440
Intensity to (0,0) / (3/(8*PI)): 1.006136
Radiated power to hemisphere * 2: 1.002368
```

You need to load modules as follows if you have logged in again before running ALiS.

```
$ module load hdf5
$ module load szip
```

You will find a new directory `Dipole_Band` has been created. Some text and image data are generated there.



(a) P-pol **(b)** S-pol

Figure 1.1 Images of (a) P- and (b) S- polarized light.

Chapter 2

World

The `world` in ALiS is a **structure pointer** pointing a `struct world` structure saving simulation geometry and results. Each simulation requires one `world`. To create `world` which is a simulation domain, three **structures** of `dom`, `res`, and `sur` must be created.

2.1 Domain

The calculation domain is defined by the `dom` structure which can be created as follows.

- dom Dom = {{$\mathtt{x}_{\mathtt{max}}$}, {$\mathtt{y}_{\mathtt{max}}$}, {$\mathtt{z}_{\mathtt{max}}$}};
- dom Dom = {{$\mathtt{x}_{\mathtt{min}}$, $\mathtt{x}_{\mathtt{max}}$}, {$\mathtt{y}_{\mathtt{min}}$, $\mathtt{y}_{\mathtt{max}}$}, {{$\mathtt{z}_{\mathtt{min}}$, $\mathtt{z}_{\mathtt{max}}$}};

$\mathtt{x}_{\mathtt{min}} = -\mathtt{x}_{\mathtt{max}}$ when $\mathtt{x}_{\mathtt{min}}$ is omitted as in the first example, which is the same for the y- and z-directions. For a 2D simulation, setting the domain in the y-direction be zero makes the simulation faster than setting other directions be zero.

- dom Dom = {{$\mathtt{x}_{\mathtt{min}}$, $\mathtt{x}_{\mathtt{max}}$}, {0}, {{$\mathtt{z}_{\mathtt{min}}$, $\mathtt{z}_{\mathtt{max}}$}};

For the Total-Field Scattered-Field (TFSF) calculation, two domains need to be declared in the same simulation: one for the total field and the other for the scattered field.

```
dom DTF = {{300}, {300}, {300}};
dom DSF = {{400}, {400}, {400}};
```

In the example above, `DTF` and `DSF` are the domains for the total field and the scattered field, respectively. Please note that the domain of the total field must be a subset of the domain of the scattered field, due to the nature of the TFSF simulation method.

2.2 Resolution

The calculation domain is defined by the `res` structure which can be created as follows.

- `res Res = {dt, {dx}};`
- `res Res = {dt, {dx}, {eV}};`
- `res Res = {dt, {dx, dy, dz}, {eV},`
  `{{`M_x`, `x_{min}`, `x_{max}`}, {`M_y`, `y_{min}`, `y_{max}`}, {`M_z`, `z_{min}`, `z_{max}`}}};`

The variables have the following meanings.

1. `dt`: Simulation time step.
2. `dx`, `dy`, `dz`: Simulation spatial grids. `dy` and `dz` can be omitted when `dx=dy=dz`.
3. `eV`: The wavelength of a photon with 1 eV of energy, λ_{eV}.
4. M_x, x_{min}, x_{max}: Magnification, minimum, and maximum boundaries for setting up a non-uniform grid.

The variable `eV` determines the unit length a of the simulation. The energy of a photon is given by $E = hf = hc/\lambda$ where the Planck constant h is $6.62607015 \times 10^{-34}$ J $\cdot$ s $\simeq$ $4.135667696 \times 10^{-15}$ eV·s and the speed of light c is 299792458 m/s. Therefore, the wavelength of a photon with 1 eV can be obtained by setting the energy E to be 1 eV, which leads to λ_{eV} $[a] = hc/1$. Please note λ_{eV} is in the simulation unit with the unit length of a. It can be seen, for instance, that l_0 can be set to be 1 nm by setting `eV`$(=\lambda_{eV})$ to be 1240. Alternatively, the simulation unit length can be set to be 1 μm by setting `eV`=1.24. This can be omitted when all the materials in the simulation are non-dispersive, due to the scale invariance of Maxwell's equations (see §A.1).

A non-uniform grid can be set by M_x, x_{min}, x_{max}, which are the magnification, minimum, and maximum boundaries in the x-axis, respectively. If these variables are set, then the spatial grid size of `dx` will be applied in the domain in $x_{min} \leq x \leq x_{max}$ whereas a coarse mesh with a magnified grid size of M_x`*dx` will be applied where $x < x_{min}$ or $x > x_{max}$. Note that the minimum and maximum boundaries must be negative and positive, respectively, i.e., $x_{min} < 0$ and $x_{max} > 0$. Similarly, a non-uniform grid can be set for y- and z-directions, too. Set a non-uniform grid and enjoy the fast simulation!

2.3 Boundary conditions: Surface

The boundary condition is defined by the `sur` structure which can be created as follows.

- `sur Sur = {{`x_{min}`, `x_{max}`}, {`y_{min}`, `y_{max}`}, {`z_{min}`, `z_{max}`}};`
- `sur Sur = {{PBC, `$2\pi/k_x$`}, {PBC, `$2\pi/k_y$`}, {`z_{min}`, `z_{max}`}};`
- `sur Sur = {{`x_{min}`, `x_{max}`}, {`y_{min}`, `y_{max}`}, {`z_{min}`, `z_{max}`}, {`n_{PML}`}};`
- `sur Sur = {{`x_{min}`, `x_{max}`}, {`y_{min}`, `y_{max}`}, {`z_{min}`, `z_{max}`}, {`n_{PML}`, `σ_{max}`,`
  κ_{max}`, `α_{max}`, `m_σ`, `m_α`}};`

The following surfaces can be set for xyz_{min} and xyz_{max}.

1. `PEC`: Perfect electric conductor.
2. `PML`: Perfectly matched layer.

3. SYM: Symmetric boundary condition.
4. PBC: Periodic boundary condition.
5. BBC: Bloch boundary condition.
6. HBC: Hybrid boundary condition.

PEC is a boundary condition with 100% reflection.

PML is an absorbing layer to mimic an infinetly extended space [8, 9]. A convolution PML (CPML) [10] is implemented in ALiS. The simulation domain is automatically expanded according to the PML thickness, hence the dom structure does not require a manual modification for PML. PML can be further specified by the following variables.

1. $\mathtt{n}_{\mathtt{PML}}$: The thickness of the PML layer in the unit of FDTD spatial grids. The default value is 12.
2. $\sigma_{\mathtt{max}}$, $\kappa_{\mathtt{max}}$, $\alpha_{\mathtt{max}}$, $\mathtt{m}_{\sigma}$, $\mathtt{m}_{\alpha}$: Variables for CPML. The default values are 1, 1, 0, 3.5, and 1, respectively.

SYM is a symmetric boundary condition. It can be applied only for $\mathtt{xyz}_{\mathtt{min}}$. When SYM is applied, the minimum boundaries set in dom are overridden by $\mathtt{x}_{\mathtt{min}}$=$-\mathtt{x}_{\mathtt{max}}$ and so on. The simulation will be run only in the positive domain of the SYM applied coordinate.

PBC is a periodic boundary condition. If PBC is set in a min boundary without specifying the max boundary, then the simple periodic boundary condition will be applied. If the max boundary is specified by $2\pi/k$, then a Bloch boundary condition with the Bloch vector of k will be applied. In this case, the simulation time will be doubled as both the electric and magnetic fields are forced to be complex numbers.

BBC is a Bloch boundary condition, which is basically identical to PBC except for the Bloch boundary condition is forced to be applied even when the max boundary is omitted. Both the electric and magnetic fields are complex numbers.

HBC is a hybrid boundary condition.

The following code is an example of setting dom, res, and sur structures.

```
float kx, ky, k=1.2;

dom Dom = {{250}, {0, 250*sqrtf(3)}, {500}};
res Res = {5, {10, 5*sqrtf(3), 10}, {1240}};
sur Sur = {{BBC, 500*3/kx}, {HBC, 500*sqrtf(3)/ky}, {SYM, PML}};
```

2.4 Creating a world

A world can be created by createWorld() function.

- world World = createWorld(Dom, Res, Sur, id_string);

The id_string is a unique string identifier of a simulation which is automatically added as a prefix to all results files generated by the simulation.

The created **structure pointer** `World` has the following member variables which can be accessed by the arrow (->) operator following the C language syntax.

- `dt, dx, dy, dz`: The time step and the spatial grid sizes.
- `xMin, xMax, yMin, yMax, zMin, zMax`: The simulation domain boundaries excluding PML.
- `xMIN, xMAX, yMIN, yMAX, zMIN, zMAX`: The simulation domain boundaries including PML.
- `iMin, iMax, jMin, jMax, kMin, kMax`: The simulation domain boundaries excluding PML in the FDTD grid unit.
- `iMIN, iMAX, jMIN, jMAX, kMIN, kMAX`: The simulation domain boundaries including PML in the FDTD grid unit.
- `E->x[i][j][k], E->y[i][j][k], E->z[i][j][k]`: The E-field components at `(i,j,k)`.
- `H->x[i][j][k], H->y[i][j][k], H->z[i][j][k]`: The H-field components at `(i,j,k)`.

A `world` can also be duplicated by `cloneWorld()` function. The duplicated world has the same `dom`, `res`, and `sur` as well as all the objects in the original world.

- `world New_World = cloneWorld(Old_World);`

A `world` can be rebooted by `rebootWorld()` function which eliminates its sources and sets all the fields to 0.

- `void rebootWorld(World);`

A `world` can be completely erased from the memory by the following functions.

- `void deleteWorld(World);`
- `void deleteCloneWorld(World);`

Chapter 3

Materials

A material is defined by a **structure** `matter`.

3.1 Non-dispersive materials

Non-dispersive materials are created by a macro function `n()`. For instance, a material with a refractive index of 1.4 can be created by `n(1.4)`. Some frequently used materials are predefined.

- `PM`: Perfect metal. $n = \infty$.
- `Air`, `Vacuum`: Air and vacuum. $n = 1$.
- `n(`n`)`: A dielectric with the refrative index of n.

3.2 Dispersive materials

ALiS provides simple conductivity, the Drude model, the Lorentz model, and the critical-points model to fit the dispersion curves of materials. The mathematical formulae of the models are described in Appendix A.2. Frequently used materials as listed below are predefined in `alis_m.h` file.

- Ag, Al, Au, Cr, Cu, Ni, Ti, Pd, Pt, Si, and Ge.

These materials are fitted in the wavelength interval of 400 - 2000 nm based on the data published in CRC Handbook of Chemistry and Physics [11], and in the paper by Johnson and Chrsity [12]. Alternatively, the fitting results in the wavelength interval of 250 - 700 nm can be imported by `#define NUV` prior to `#include <alis.h>`

```
#define NUV
#include <alis.h>
```

Chapter 4

Geometry: Object

Geometry in ALiS is defined by `object`.

4.1 Shapes

Supported shapes for `object` are as follows.

1. `Box, Ball, Diamond, RodX, RodY, RodZ, BiconeX, BiconeY, BiconeZ`
2. `DRodX, DRodY, DRodZ, HRodXY, HRodXZ, HRodYZ, HRodYX, HRodZX, HRodZY`

The position and size of an object can be determined either (1) by its centre and the radius,

- `object Object = {shape, {{x, y, z}, {r}}};`
- `object Object = {shape, {{x, y, z}, {`r_x`, `r_y`, `r_z`}}};`

or (2) by its boundary values.

- `object Object = {shape, {{`x_{min}`, `x_{max}`}, {`y_{min}`, `y_{max}`}, {`z_{min}`, `z_{max}`}};`

4.2 Operations on objects

You can make a new object or a group of new objects by applying operations such as: `Intersection`, `Combination`, `Difference`, `Translation`, `Rotation`, `CLatticeX`, `CLatticeY`, `CLatticeZ`, `Lattice`, and `DLattice`.

- `object New = {Intersection, {n}, objects {O1, O2, ...}};`
- `object New = {Combination, {n}, objects {O1, O2, ...}};`
- `object New = {Difference, {n}, objects {O1, O2, ...}};`
- `object New = {Translation, {x, y, z}, objects {Old}};`
- `object New = {Rotation, {`θ_x`, `θ_y`, `θ_z`}, objects {Old}};`
- `object New = {CLatticeX, {{n}, {y, z}}, objects {Old}};`

- `object New = {CLatticeY, {{n}, {x, z}}, objects {Old}};`
- `object New = {CLatticeZ, {{n}, {x, y}}, objects {Old}};`
- object New = {Lattice, {{a_x, n_x}, {a_y, n_y}, {a_z, n_z}}, objects {Old}};
- object New = {DLattice, {{a_x, n_x}, {a_y, n_y}, {a_z, n_z}}, objects {Old}};

`New`, `Old`, `O1`, `O2`, and `n` are the new object, old object, first object, second object, and number of objects, respectively. The angles for `Rotation` are in degrees (°). In cases of `Lattice` and `DLattice`, only a basic shape object can be put in place of `Old`, while a composite object made by an operation can be `Old` for the other operations.

Example 1: A triangular lattice photonic crystal slab with a single-cell defect

```
object PhC =
  {Difference, {2}, objects {
    {Box, {{-INF, INF}, {-INF, INF}, {-100, 100}}},
    {Difference, {2}, objects {
      {DLattice, {{500, 9}, {500*sqrtf(3), 5}}, objects {
        {RodZ, {{-175, 175}, {-175, 175}, {-INF, INF}}},
      }},
      {RodZ, {{175, 175}, {-175, 175}, {-INF, INF}}},
    }},
  }};
```

An array of `Object` can be created and used to composite many `objects` as in the following example.

Example 2: A bull's eye with pitch p and depth d.

```
object Ring[15], Disk[15][2];
for (int n=0; n<15; n++) {
  Disk[n][0] = (object) {RodZ,
    {{-p*(n+0.25), p*(n+0.25)}, {-p*(n+0.25), p*(n+0.25)}, {0, d}}};
  Disk[n][1] = (object) {RodZ,
    {{-p*(n-0.25), p*(n-0.25)}, {-p*(n-0.25), p*(n-0.25)}, {0, d}}};
  Ring[n] = (object) {Difference, {2}, Disk[n]};
}
object BullsEye = {Combination, {15}, Ring};
```

The following example is wrong.

Example 3: Wrong example!

```
object Ring[15];
for (int n=0; n<15; n++) {
  Ring[n] = (object)
    {Difference, {2}, objects {
      {RodZ,
        {{-p*(n+0.25), p*(n+0.25)}, {-p*(n+0.25), p*(n+0.25)}, {0, d}}},
      {RodZ,
        {{-p*(n-0.25), p*(n-0.25)}, {-p*(n-0.25), p*(n-0.25)}, {0, d}}}
  }};
}
object BullsEye = {Combination, {15}, Ring};
```

4.3 Putting objects into the world

- `void putObjects(World, M1, O1, M2, O2, ..., Background_Matter)`

where `M1`, `M2` are the first and the second materials and `O1`, `O2` are the first and the second objects.

- `void putObjectArray(World, Matter_Array[], Object_Array[])`

4.4 Resetting the geometry

All information of `matter` and `object` can be removed by the following function.

- `void removeObjects(World)`

Chapter 5

Sources

Different types of electromagnetic sources can be set in ALiS. Once a source is set, it will be automatically launched in the `World` when the updating functions described in §6.1 are executed. The `World` will have the following member variables in addition to those described in §2.4.

- `T`: The time step for the period of the last launched source
- `N`: The time step to complete the launch of any `Pulse` type source.

5.1 Point dipole

A point diple can be placed at (x, y, z) by the following function.

- `void pointDipole(World, field, x, y, z, waveform,`
  `λ, Δλ, amplitude, phase)`

`field` is the field component of the dipole source which can be one of the followings: `Ex`, `Ey`, `Ez`, `Hx`, `Hy`, and `Hz`. `waveform` can be set to be `Pulse` for a pulsed wave or `Sine` for a continuous wave. λ is the central wavelength of the source. $\Delta\lambda$ is the linewidth for `Pulse` while the ramp-up time step of the source for `Sine`. The ramp-up time step is the time step for the source to reach the `amplitude`. `amplitude` and `phase` are optional with the default values of 1 and 0, respectively. The unit for `phase` is degree.

5.2 Planewave

A function to excite a planewave is defined as follows.

- `void planewave(World, Dom, polarization, theta, phi, waveform,`
  `λ, Δλ, amplitude, phase)`

For `Dom`, either the entire simulation domain can be used or the domain of the total field can be put when the Total-Field Scattered-Field (TFSF) method is used. For polarization, `Ppol` or `Spol` can be put. `theta` and `phi` are the polar and the azimuthal angles in degrees, respectively. The following options are supported for `waveform`: `Sine`, `Pulse`, and `Band`.

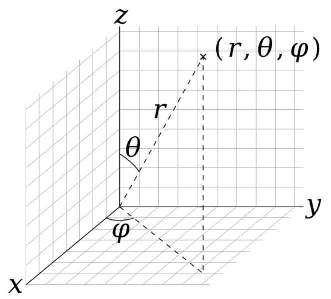


Figure 5.1 The coordinate system for the planewave in ALiS. The rotation of the polar angle θ is applied first for the rotation about the y-axis ($\phi = 0$) then the rotation of the azimuthal angle ϕ follows. The image is from Wikipedia.

The coordinate system to define a planewave is shown in Figure 5.1. The planewave is placed at the positive z-axis boundary of the entire domain parallel to the xy-plane in a 3D simulation. `Ppol` and `SPol` are initially polarized in x- and y-directions, respectively, when both rotational angles are 0. The rotation of the polar angle θ is applied first for the rotation about y-axis ($\phi = 0$) then the rotation of the azimuthal angle ϕ follows. Please refer to Appendix A.5 for further details regarding the rotation of planewave.

If the Total-Field Scattered-Field (TFSF) is applied, then the planewave is at the positive z-axis boundary of the total field domain. Therefore, it is generally advised to construct the simulation geometry considering the propagation of the beam in -z-direction.

A circularly polarized planewave can be excited by launching a x- and a y-polarized waves with $\pi/2$ relative phase. For example, a pulsed circularly polarized planewave can be excited as follows:

```
int pol = atoi(argv[1]); // polarization
planewave(W, Dom, Ppol, 0, 0, Pulse, 600, 10, 1, 0); // Ex
planewave(W, Dom, Ppol, 0, 90, Pulse, 600, 10, 1, pol*90); // Ey
```

The variable `pol` is an argument which can be set to be -1 or $+1$ to select different circular polarization. In this example, -1 and $+1$ are LCP and RCP waves propagating in -z-direction, respectively, according to the circular polarization convention defined using a set of helical bases, $\mathbf{e}_\pm = (\mathbf{e}_x \mp \mathbf{e}_y)/\sqrt{2}$, as in Ref. 13. The generation of circularly polarized waves can be seen from the E-field amplitude as a function of time shown in Figure 5.2.

Please note the phase option is currently not supported for the `waveform` of `Band`.

5.3 Focused beam

A focused beam with a focal point at (x, y, z) can be set by the following function.

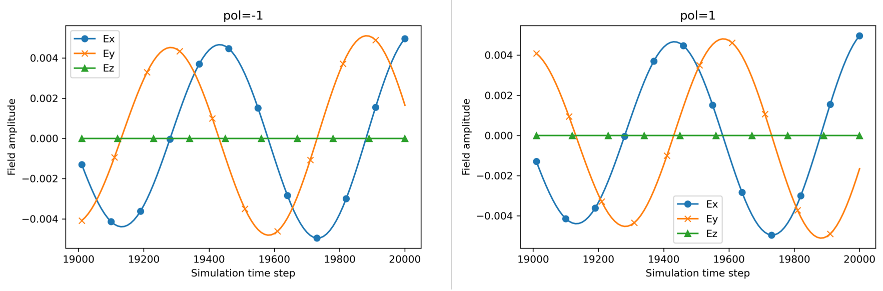


Figure 5.2 A circularly polarized source can be excited by setting the relative phase of -90 or 90 degrees between E_x and E_y sources.

- `void focusedBeam(World, Dom, field, x, y, z, theta, waveform,` `λ, Δλ, spotsize, amplitude, phase)`

Please note that the principal axis of a focused beam must be set to be z-axis and the beam must propagate in $\pm z$-directions. `field` can be `Ex` or `Ey`. `theta` can be 0 or 180 which corresponds to the propagation in $-z$- or $+z$-direction, respectively. `spotsize` is the radius of the beam on the focal plane, *i.e.*, the beam waist, in the unit of the wavelength. The rest are as explained in Section 5.1.

The TFSF method can be used for `focusedBeam`.

5.4 Resetting sources

The following function removes all sources in the world.

- `void removeSources(World)`

Chapter 6

Propagation

The electromagnetic fields need to be propagated in a discrete time step to progress the simulation in the FDTD method. This chapter explains how you can make the fields propagate.

6.1 Updating fields

To run the simulation for the time `t = N * dt`, the *H*- and *E*-fields require to be updated for `N` times. To do this, you execute `updateH(World)` and `updateE(world)` functions `N` times in a for loop as follows. For a technical reason, it is recommended to update *H*-field first and then *E*-field.

```
for (int n=1; timer(string, n, N); n++)
{
updateH(W);
/* Output statements ...  */
updateE(W);
/* Output statements ...  */
}
```

`n`, `N`, `W` are the current time step, the total time steps, and the simulation world, respectively. `timer` function works identically to a conventional for loop for `n<=N` with an additional functionally to display the current simulation progress in percentage and estimated run time as `stdout`. `World->ID` can be used in `string` in `timer` to indicate the current simulation.

Sometimes, you may want to print the simulation progress and the estimated run time instead of displaying them in real time, which is often the case when you use a shared server. Then the for loop can be run in the same manner as printing the current values. This can be realized as below.

```
for (int n=1; n<=N; n++){
   updateH(W);
   updateE(W);
   if (n%(10*W->T) == 0){
      float pgr = 100*n/N;
      printf("Current time step: %d\n", n);
      printf("Simulation progress: %.2f%\n", pgr);
   }
}
```

Chapter 7

Outputs

7.1 Fields

Values from fields can be extracted as outputs. Possible output `field`'s are listed below.

1. Refractive index: `RI, LogRI, ContourRI`
2. Basic fields: `Ex, Ey, Ez, Hx, Hy, Hz, Jx, Jy, Jz, iEx, iEy, iEz, iHx, iHy, iHz`
3. Energy density: `EE, UE, HH, UH, U, LogEE, LogUE, LogHH, LogUH, LogU`
4. Energy flux: `Sx, Sy, Sz, ExHy, ExHz, EyHz, EyHx, EzHx, EzHy, JE, LogJE`
5. Optical force: `Fx, Fy, Fz, Txx, Txy, Txz, Tyx, Tyy, Tyz, Tzx, Tzy, Tzz`
6. Charge density: `DivE`
7. Scattered fields: `ScEx, ScEy, ScEz, ScEE`

You may recognize that the refractive index can be used as an output just like other fields. Outputing the refractive index is useful to check the geometry. The scattered fields can be used as outputs in a TFSF simulation. Please note that all outputs in the scattered-field domain are calculated from scattered fields. The scattered fields outputs `ScEx, ScEy, ScEz, ScEE` are useful to extract the scattered fields in the total-field domain.

7.2 Simple output functions

The output functions listed below take `field` as their input and return the corresponding values.

- `float get(World, field, x, y, z)`
- `float worldMax(World, field)`
- `float worldSum(World, field)`
- `float poyntingX(World, x)`
- `float poyntingY(World, y)`
- `float poyntingZ(World, z)`

- `float poyntingIn(World,` x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}`)`
- `float poyntingOut(World,` x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}`)`
- `float objectAbsorption(World, Object)`

The following writing functions allow you to write the extracted values using the output functions listed above into a file.

- `void writeTxt(World, filename, output_string)`
- `void writeRow(World, filename,` f_1, f_2, f_3, $\cdots$`)`
- `void writeSpectrum(World,` n_{total}, λ_{min}, λ_{max},
  `filename,` f_1, f_2, f_3, $\cdots$`)`

The writing function `writeSpectrum` returns the data in three columns: wavelength, amplitude, and phase. The range of phase in the third column is $[-1, 1]$ as normalized by π. For the $x-$component of complex E-field, for instance, which can be written as

$$E_x = |E_x|e^{i\phi_x}, \tag{7.1}$$

the second column is $|E_x|$ normalized by the initial amplitude and the third column is ϕ_x/π. Differently from the other two writing functions, `writeSpectrum` takes only **basic fields** as the output fields. Other fields such as `EE` will return incorrect values.

Example: Output functions.

```
writeRow(World, "-time", World->t, get(World, Hz, 10, 10, 20));
writeSpectrum(World, 65536, 200, 2000, "-freq", get(World, Hz, 10, 10, 20)
    );
```

`min` and `max` values of the other two coordinates can be specified in `poyntingX`, `poyntingY`, `poyntingZ` functions.

- `float poyntingX(World, x,` y_{min}, y_{max}, z_{min}, z_{max}`)`
- `float poyntingY(World, y,` x_{min}, x_{max}, z_{min}, z_{max}`)`
- `float poyntingZ(World, z,` x_{min}, x_{max}, y_{min}, y_{max}`)`

7.3 Slice output functions

`slice` is a **structure pointer** to provide various output features, which can be understood as a sliced cross-section of the `world`. 2D and 1D `slice` can be generated by setting the size of the other one or two dimensions to be zero.

- `slice Slice = createSlice(World,` x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}`);`
- `slice Slice = createSliceX(World, y, z);`
- `slice Slice = createSliceY(World, x, z);`
- `slice Slice = createSliceZ(World, x, y);`
- `slice Slice = createSliceXY(World, z);`
- `slice Slice = createSliceYZ(World, x);`
- `slice Slice = createSliceZX(World, y);`

You can obtain a maximum and a sum of `field` in a `slice` using the following functions.

- `float sliceMax(World, field, Slice)`
- `float sliceSum(World, field, Slice)`

The following functions save the `slice` to a file.

- `void sliceSnap(World, field, Slice, n, filetype, filename)`
- `void sliceTimeAvg(World, field, Slice, n, filetype, filename)`
- `void sliceFreqDom(World, field, Slice, n, λ, filetype, filename)`

If `%%` is used in `filename`, then the output will be written in a file whose name is `field_name-slice_name`. The supported `filetype`'s are hdf5, plain text, and png bitmap image, which can be selected by `h5`, `txt`, and `png(colormap,` $f_{\texttt{norm}}$`)`, respectively. For PNG format, $f_{\texttt{norm}}$ is a normalization factor, and the following colormaps are available.

1. For field amplitude: `dkbr`, `dkrb`
2. For field intensity: `gray`, `red`, `blue`, `hot`, `hsv`, `jet`

Example: Slice output functions.

```
slice XY = createSliceXY(World, 0);
...
/* main loop */
  ...
  sliceTimeAvg(World, Ez, XY, World->T, h5, "/%%/");
  sliceSnap(World, Ez, XY, World->T/15, png(dkbr,1), "/%%/");
```

7.4 Far-field output functions

ALiS supports the transformation from near field to far field. Three steps are required to make a far-field output: (1) Create a `phaser`, (2) Update the `phaser` during the progress of the simulation, and (3) Calculate the far field from the `phaser`.

A `phaser` can be created as follows.

- `phaser Phaser = createPhaser(World, λ,`
  $x_{\texttt{min}}$, $x_{\texttt{max}}$, $y_{\texttt{min}}$, $y_{\texttt{max}}$, $z_{\texttt{min}}$, $z_{\texttt{max}}$`);`

where λ is the wavelength for the phaser calculation, and $\texttt{xyz}_{\text{min, max}}$ define the size of the phaser box. If the boundary values of the phaser box can be omitted, then it will be set to be the entire simulation domain. Currently, phaser-related functions are not supported in the non-uniform grid, hence the phaser box must be inside the domain of a uniform-grid.

To update the phaser during the progress of the simulation, the following function is used.

- `void updatePhaser(World, Phaser)`

Far-field properties are calculated from the phaser using the following functions.

- `float complex farField(Phaser, polarization, theta, phi)`
- `float farFieldI(Phaser, polarization, theta, phi)`
- `float farFieldFlux(Phaser, polarization, theta, phi,` `angle`$_{\mathtt{min}}$`, angle`$_{\mathtt{max}}$`,` Δ`angle)`
- `void farFieldTheta(World, Phaser, phi, filename)`
- `void farFieldPhi(World, Phaser, theta, filename)`
- `void farFieldProfile(World, Phaser, map, N`$_{\mathtt{x}}$`, N`$_{\mathtt{y}}$`,` `filetype, filename)`

The available `map`'s for the `farFieldProfile` function are as follows.

1. Far-field maps: `AzimuthalMap`, `NorthernAzimuthalMap` `SouthernAzimuthalMap`, and `CylindericalMap`

7.5 Shell command

In addition, a shell command can be used in `exec()` to further manipulate the data files generated by the writing functions.

- `void exec(command_string)`

Example: A shell command displaying the date and current time.

```
#include <alis.h>

int main(int argc, char **argv)
{
    exec("date");
}
```

Chapter 8

Run

8.1 Execution

As we have briefly seen in §1.3, execution of an ALiS code has two steps: Compile and run.

Once you have your C code completed, then you compile the code by the following command.

```
$ alis YourALiSCode.c
```

Then you will find an executable file created in the same directory. The next step is to run the executable.

```
$ ./YourALiSCode
```

Then you will see your code running.

8.2 Parameter sweep

Most of the time, you would want to run a series of simulations for a parameter sweep.

Let's take the nanoblock scattering as an example. Under the directory `examples`, you will be able to find `Nanoblock_Scattering.c` which is as follows.

Example: Nanoblock scattering.

```
#include <alis.h>


int main(int argc, char **argv)
{
  dom DTF = {{300}, {300}, {300}}; // Domain of total-field area
  dom DSF = {{400}, {400}, {400}}; // Domain of scattered-field area
  res Res = {5, {10}, {1240}};
  sur Sur = {{PML}, {SYM, PML}, {PML}};
  world W = createWorld(DSF, Res, Sur, "%s", argv[0]);

  object Nanoblock = {Box, {{-200, 200}, {-200, 200}, {-200, 200}}};
  object Substrate = {Box, {{-INF, INF}, {-INF, INF}, {-INF,-200}}};
  putObjects(W, Au, Nanoblock, n(1.5), Substrate, Air);
  planewave(W, DTF, Ppol, 55, 0, Sine, 500, 20);

  float abs=0, sca=0;
  slice XY = createSliceXY(W, 0);
  slice XZ = createSliceXZ(W, 0);
  slice YZ = createSliceYZ(W, 0);

  for (int n=1, N=40*W->T; timer(n, N); n++) {
    updateH(W);
    abs += poyntingIn (W,-250, 250,-250, 250,-250, 250);
    sca += poyntingOut(W,-350, 350,-350, 350,-350, 350);
    updateE(W);
    abs += poyntingIn (W,-250, 250,-250, 250,-250, 250);
    sca += poyntingOut(W,-350, 350,-350, 350,-350, 350);

    if (n%W->T == 0) {
      writeRow(W, "/Cross-section", W->t, abs/W->T, sca/W->T, (abs+sca)/W
    ->T);
      abs = sca = 0;
    }
    if (N-n < W->T) {
      sliceSnap(W, ScEE, XY, 10, png(hot,2), "/%%/");
      sliceSnap(W, ScEE, XZ, 10, png(hot,2), "/%%/");
      sliceSnap(W, ScEE, YZ, 10, png(hot,2), "/%%/");
    }
  }
}
```

You can see that there is a gold nanoblock cube at the centre of the simulation domain and the code is calculating absorption and scattering cross-sections using the TFSF method. It can also be seen that the characteristic length is 1 nm since `eV` in `res` is given to be 1240 as explained in §2.2. Therefore, the size of the nanoblock is 400 nm on each side.

Now, imagine you want to change the length of each side of the nanoblock from 300 to 400 with a 20 nm step. In this case, you want to get the size of the nanoblock as an argument. Let's define a variable for the half-length of the nanoblock cube.

```
  int hL = atoi(argv[1]); // Nanoblock half-length (nm)
```

Then use this variable `hL` to define the dimension of the Nanoblock.

```
  object Nanoblock = {Box, {{-hL, hL}, {-hL, hL}, {-hL, hL}}};
```

As we do not override the simulation results on the same file names and directories, it is generally a good idea to change the name of the `world` for different parameters.

```
  world W = createWorld(DSF, Res, Sur, "%s_hL%d", argv[0], hL);
```

Then the first part of the modified code will look like this.

```
#include <alis.h>


int main(int argc, char **argv)
{
    int hL = atoi(argv[1]); // Nanoblock half-length (nm)

    dom DTF = {{300}, {300}, {300}}; // Domain of total-field area
    dom DSF = {{400}, {400}, {400}}; // Domain of scattered-field area
    res Res = {5, {10}, {1240}};
    sur Sur = {{PML}, {SYM, PML}, {PML}};
    world W = createWorld(DSF, Res, Sur, "%s_hL%d", argv[0], hL);

    object Nanoblock = {Box, {{-hL, hL}, {-hL, hL}, {-hL, hL}}};
    object Substrate = {Box, {{-INF, INF}, {-INF, INF}, {-INF,-200}}};
    putObjects(W, Au, Nanoblock, n(1.5), Substrate, Air);
    planewave(W, DTF, Ppol, 55, 0, Sine, 500, 20);
    ...
```

Let's make an executable shell command file `Nanoblock_Scattering.sh` to run the parameter sweep.

```
#!/bin/bash

for hL in $(seq -w 150 10 200); do
   ./Nanoblock_Scattering ${hL}
done
```

This shell command file sweeps the half-length of the nanoblock from 150 to 200 nm with a 10 nm step. Now run the shell command file in the command line.

```
$ ./Nanoblock_Scattering.sh
```

Then you will see the running timer indicating the simulation progress and after some minutes, you will see the simulation results under the directory named with the suffixes `_hL150`, `_hL160`, . . . , `_hL200`.

Appendices

Appendix A

Further details of ALiS

A.1 Natural unit system

Maxwell's equations are scale invariant [14]. It is convenient to use scale-invariant units so that we can avoid the headache of handling ridiculously small or large numbers which may cause overflow or underflow.

ALiS uses a scale-invariant natural unit system where the speed of light and the impedance in vacuum, $c \equiv 299792458$ m/s and $Z_0 \equiv \pi \times 119.9169832\ \Omega$, respectively, are the basis units which implies both c and Z_0 are 1 in ALiS. Accordingly, the electric permittivity ($\epsilon_0 \equiv Z_0^{-1}c^{-1}$) and the magnetic susceptibility ($\mu_0 \equiv Z_0 c^{-1}$) in vacuum are also 1. Therefore, the units for velocity, electric permittivity, magnetic susceptibility, and resistance are c, ϵ_0, μ_0, and Z_0, respectively.

Other physical quantities can be represented by other units satisfying the given relations with the basis units. If you choose a length a as your unit length, then other physical quantities will have units according to the unit length as shown in Table A.1.

Table A.1 Examples of unit conversion when the unit length is set to be a or 1 nm.

Physical quantity	Unit in a	Unit when $a = 1$ nm
Length, L	a	nm
Wavenumber, k	$2\pi/a$	2π nm$^{-1} \simeq 6.28$ nm^{-1}
Time, t	a/c	c^{-1} nm $\simeq 3.34 \times 10^{-18}$ s
Frequency, f	c/a	c nm$^{-1} \simeq 3.00 \times 17$ Hz
Angular frequency, ω	$2\pi c/a$	$2\pi c$ nm$^{-1} \simeq 1.88 \times 10^{18}$ s^{-1}

Let us assume you want to use nm for the unit of length. First, you write all the length values in nm in your C code. Then the units of other physical quantities such as wavenumber, time, frequency, and angular frequency are as in the last column in the Table. To obtain the actual values, you can multiply the output values by the values in the last column.

It is convenient to remember that the wavelength λ and the period T have the same unit as well as the wavenumber k and the angular frequency ω also do in this normalzied unit system.

Other physical quantities such as power, intensity, energy, force, E-field, H-field, and electric current can be interpreted in different units. In case the unit of power is set to be μW and the characteristic length $a = 1$ nm, then the units of other physical quantities are as in Table A.2.

Table A.2 Units when the characteristic length $a = 1$ nm and the unit of power is μW.

Physical quantity	Unit
Power	μW
Intensity	μW/nm^2 =W/μm^2
Energy	c^{-1} nm $\cdot$ μW $= 3.34 \times 10^{-24}$ J
Force	c^{-1} nm $\cdot$ μW $= 3.34 \times 10^{-15}$ N
E-field	$(Z_0\Omega^{-1})^{1/2}$ mV/nm $= 19.4$ V/μm
H-field	$(Z_0\Omega^{-1})^{1/2}$ mA/nm $= 51.5$ A/mm
Electric current	$(Z_0\Omega^{-1})^{1/2}$ mA $= 51.5$ μA

A.2 Dispersion models

ALiS supports dispersion models of conductivity, Drude, Lorentz, and critical-points for simulations with dispersive materials such as metals and semiconductors.

$$\epsilon(\omega) = \epsilon_\infty + \sum_p \chi_p(\omega) \tag{A.1}$$

The equation of the dispersion model is given in Eq. (A.1). The poles and the corresponding formulae for the $\chi_p(\omega)$ of each model are shown in Table A.3.

Example: Materials defined in `alis_m.h`

```
matter Ag = {{4.0323},{{9.1864,0.021}}};
matter Au = {{2.1452},{{8.4146,0.0255},
     {2.6203,0.2147,0.2082,0.3059},{3.1543,1.1662,4.1353,2.5005}}};
```

Silver and gold are shown as examples of dispersive materials. It can be seen that silver is defined using the Drude model, where $\epsilon_\infty = 4.0323$, $\omega_p = 9.1864$, and $\gamma_p = 0.021$. Gold has four sets of parameters where the last two sets are for the critical-points model.

You can find an interactive Python notebook code (`DFFit.ipnyb`) for fitting dispersive materials under `doc` directory.

Table A.3 Dispersion models

Model	`Pole`	$\chi_p(\omega)$
Conductivity	$\{0,\ \sigma_p\}$	$-\frac{\sigma_p}{i\omega}$
Drude model	$\{\omega_p,\ \gamma_p\}$	$-\frac{\omega_p^2}{\omega^2+i\gamma_p\omega}$
Lorentz model	$\{\omega_p,\ \gamma_p,\ f_p\}$	$\frac{f_p\omega_p^2}{\omega_p^2-\omega^2-i\gamma_p\omega}$
Critical-points model	$\{\omega_p,\ \gamma_p,\ f_p,\ g_p\}$	$\frac{\omega_p(f_p+ig_p)}{\omega_p+\omega+i\gamma_p}+\frac{\omega_p(f_p-ig_p)}{\omega_p-\omega-i\gamma_p}$

A.3 Grid specifications

The grid specifications in ALiS are as follows.

- `E->x[i][j][k]`$\equiv E_x(i-1/2,j,k)$
- `E->y[i][j][k]`$\equiv E_y(i,j-1/2,k)$
- `E->z[i][j][k]`$\equiv E_z(i,j,k-1/2)$
- `H->x[i][j][k]`$\equiv H_x(i,j-1/2,k-1/2)$
- `H->y[i][j][k]`$\equiv H_y(i-1/2,j,k-1/2)$
- `H->z[i][j][k]`$\equiv H_z(i-1/2,j-1/2,k)$

A.4 Mathematics in ALiS

A.4.1 Mathematical constants

The following constants can be used in ALiS.

- `PI` $= 3.14159$

A.4.2 Trigonometric functions

`sin()`, `cos()`, `tan()` can be used to take an angle in radians as their argument. If you run the following code, for example,

```
#include <alis.h>

int main(int argc, char **argv)
{
    // maths test
    float q = 30; // angle in degrees
    sin_q = sin(q*PI/180);
    cos_q = cos(q*PI/180);
    tan_q = tan(q*PI/180);
    printf("sin(theta) = %.2f\n", sin_q);
    printf("cos(theta) = %.2f\n", cos_q);
    printf("tan(theta) = %.2f\n", tan_q);
    printf("\n");
}
```

the result will be as follows.

```
sin(theta) = 0.50
cos(theta) = 0.87
tan(theta) = 0.58
```

A.5 Notes on the planewave source

Table A.4 shows some examples of polarization of the planewave source depending on the angles. Some examples of the planewave source excited in free space are shown in Figure A.1. The `waveform` of `Band` was used for the simulation in the example.

`polarization`	`theta` (°)	`phi` (°)	E_x	E_y	E_z
`Ppol`	0	0	1	0	0
	0	45	$1/\sqrt{2}$	$1/\sqrt{2}$	0
	0	90	0	1	0
	45	0	~ 0	0	a
	90	0	0	0	a
	90	45	0	0	a
	90	90	0	0	a
`Spol`	0	0	0	1	0
	0	45	$1/\sqrt{2}$	$1/\sqrt{2}$	0
	0	90	1	0	0
	45	0	0	$1/\sqrt{2}$	0
	45	45	a	a	0
	90	0	0	1	0
	90	45	a	a	0
	90	90	1	0	0

Table A.4 Some examples of polarization of the planewave depending on the rotational angles. A constant a means that the magnitude of the field component has a clear non-zero value, but the relative amplitude is not confirmed.

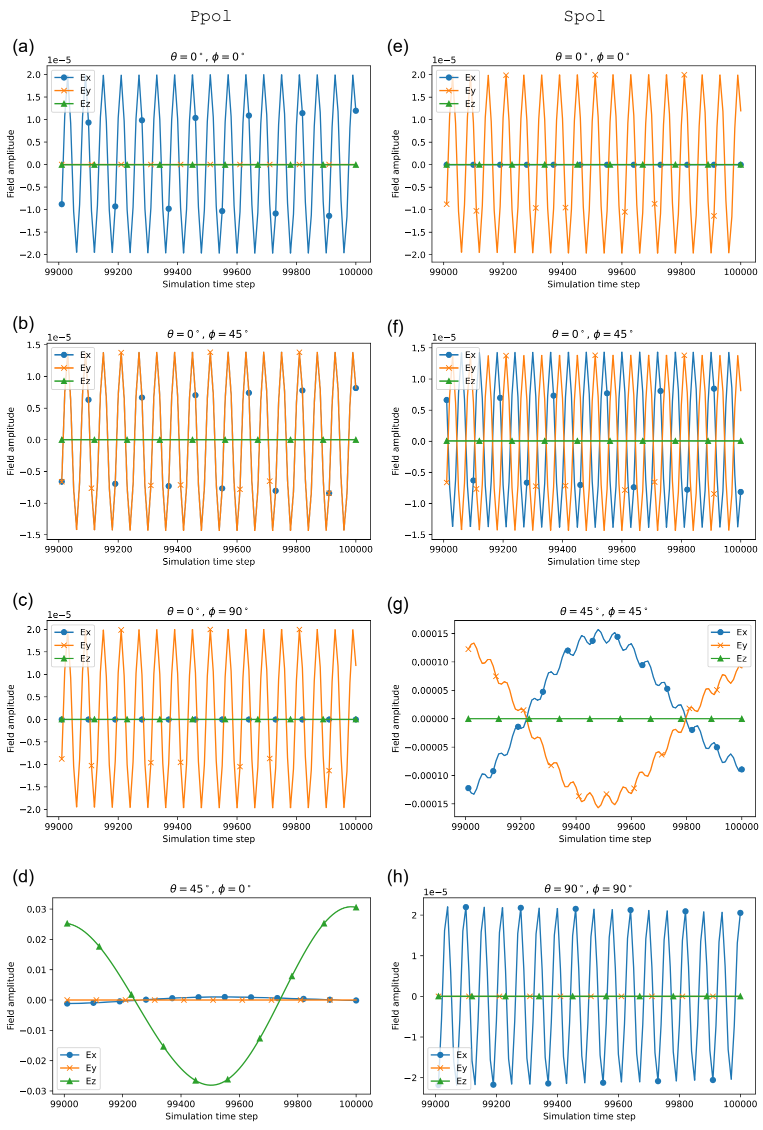


Figure A.1 Some examples of planewave sources excited to propagate in free space are shown. The field amplitude as a function of time is displayed for E_x, E_y, and E_z components. Please note the relative phase between E_x and E_y-fields are different for `Ppol` and `Spol`.

Bibliography

[1] Allen Taflove, Ardavan Oskooi, and Steven G Johnson. *Advances in FDTD computational electrodynamics: Photonics and nanotechnology.* Artech house, 2013.

[2] Ho-Seok Ee. *ALiS Manual.* Unpublished, 2017.

[3] Ho-Seok Ee, Kyung-Deok Song, Sun-Kyung Kim, and Hong-Gyu Park. Finite-difference time-domain algorithm for quantifying light absorption in silicon nanowires. *Israel Journal of Chemistry*, 52(11-12):1027–1036, 2012.

[4] Yongsop Hwang, Min-Soo Hwang, Won Woo Lee, Won Il Park, and Hong-Gyu Park. Metal-coated silicon nanowire plasmonic waveguides. *Applied Physics Express*, 6(4): 042502, 2013.

[5] Yongsop Hwang and Hong-Gyu Park. Geometric dependence of metal-coated silicon nanowire plasmonic waveguides. *Journal of Optics*, 16(2):025001, 2014.

[6] Yoon-Ho Kim, Soon-Hong Kwon, Ho-Seok Ee, Yongsop Hwang, You-Shin No, and Hong-Gyu Park. Dependence of Q factor on surface roughness in a plasmonic cavity. *Journal of the Optical Society of Korea*, 20(1):188–191, 2016.

[7] Taesu Ryu, Moohyuk Kim, Yongsop Hwang, Myung-Ki Kim, and Jin-Kyu Yang. High-efficiency SOI-based metalenses at telecommunication wavelengths. *Nanophotonics*, 11(21):4697–4704, 2022.

[8] Jean-Pierre Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of computational physics*, 114(2):185–200, 1994.

[9] Steven G Johnson. Notes on perfectly matched layers (PMLs). *arXiv preprint arXiv:2108.05348*, 2021.

[10] J Alan Roden and Stephen D Gedney. Convolution PML (CPML): An efficient FDTD implementation of the CFS–PML for arbitrary media. *Microwave and optical technology letters*, 27(5):334–339, 2000.

[11] David R Lide. *CRC handbook of chemistry and physics*, volume 85. CRC press, 2004.

[12] Peter B Johnson and R W Christy. Optical constants of the noble metals. *Physical review B*, 6(12):4370, 1972.

[13] Kirill Koshelev, Ivan Toftul, Yongsop Hwang, and Yuri S Kivshar. Scattering matrix for chiral harmonic generation and frequency mixing in nonlinear metasurfaces. *Journal of Optics*, 2023.

[14] John D Joannopoulos, Steven G Johnson, Joshua N Winn, and Robert D Meade. *Photonic Crystals: Molding the flow of light*. Princeton University Press, 2008.