PSlab - Photonic Slab Simulation Tool

Release 0.1.1

Maciej Dems

March 1, 2006

Technical University of Łódź, Institute of Physics
Łódź, ul. Wólczańska 219
Email: maciej.dems@p.lodz.pl

You may use and distribute this software according to the General Public Licence terms, but we kindly ask you to give us a recognition by adding the citation [1] to the list of references in any publication you produced with the help of `PSlab`. This is not an obligation, however we believe that you consider our kind request.

Abstract

This document describes the package `PSlab`, wich is the photonics simulation tool based on Plane-wave Admittance Method [1]. It covers the basic structure definition and running photonic simulations and also provides the detailed class description for anyone wanting to use or develop `PSlab`. The fist part of the document is a simple tutorial presenting the `PSlab` basis and giving examples of common tasks. the later chapters cover the program class structure and some more advanced topics.

This manual assumes basic knowledge on physics, photonics and numerical computations as well as the Python language programming. For an informal introduction to Python, see the refs. [2] and [3]

Some more advanced chapters require also the basic skill in C++ programming, however they are required only for people wanting to extend and develop the software.

CONTENTS

CHAPTER

ONE

Introduction

PSlab stands for the "Photonic Slab" and is a tool for modelling of optical properties in various photonic structures. It can be used for simulations of as well some simple two-dimensional devices, like good old edge emitting laser, as more complicated ones including photonic band-gap materials or photonic crystal based lasers.

The main properties of PSlab are

- the computations are fully vectorial, no scalar approximation is performed,
- every structure can be tackled as long as it can be divided into several layers uniform in one direction,
- both Cartesian 2D and 3D geometries are supported,
- 2D periodic boundary conditions are naturally implemened,
- Perfetly Matched Layers (PMLs) as abosorbing boundary conditions can be added at each side to properly analyze the radiating modes.

The computations are performed with the **plane wave admittance method** (PWAM) — a fully vectorial and three-dimensional numerical procedure for solving the Maxwell equations in complicated structures. Contrary to other methods (like FDTD or Method of Lines) PWAM does not require spatial discretization of the computational domain but describes the electromagnetic field as a sum of 2D plane-waves with varying amplitudes along z-axis.

You can find the detailed mathematical desription of PWAM in ref. [1]. This document only presents the basic usage of the PSlab package. It consists of the simple tutorial covering the definition of a structure and running the simulation and more detailde class description and reference.

1.1 Main concepts

Please read the following two sections carefully as it gives the basic idea of what's going on and the definition of the term used. It is probably necessary for understanding of the rest of this manual.

PSlab works on a photonic structure defined by the user. This can be either a fully three-dimensional or two-dimensional one. In the latter case it can be assumed that the device has transational symmetry in one direction in case of Cartesian coordinates or is axisymmetric for cyllindrical coordinates.

Each structure in PSlab consist of the several layers. Regardless of the coordinate system (either Cartesian or cylindrical) the z axis is **always** defined as the one perpendicular to theese layers. There can be many different layers but there are two important requirements:

1. the layer boundaries are planes paraller to each other and perpendicular to z-axis (which is the consequence ot the abovementioned definition) and

2. in each layer the structure is uniform along z-direction (see fig.); in case of non-planar constructs they must be approximated with several thin layers.

The `PSlab` software allows to determine the eigenmodes in the defined structure. This is done either be a searching of the eigenfrequency for an arbitrary wavevector, or by looking for a propagation constant at a fixed frequency[1]. When they are found it is possible to compute the field distibution in the whole structure which can be than saved to a file, ploted with some graphical library, or used in your further calculations.

1.2 Glossary

[1] In `PSlab` the frequency is always refered as the normalized angular frequency $k_0 = \frac{\omega}{c} = \frac{2\pi\nu}{c}$

CHAPTER

TWO

Instalation

The general instalation instructions are in `INSTALL`. You should read it and follow the instructions. The quickest instalations would requere three commands

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

For the successful instalation you will need the following libraries installed in your computer

- The BLAS and LAPACK. These are fortran libraries providing efficient subroutines for linear algebra. You can download them from http://www.netlib.org/blas/ and http://www.netlib.org/lapack/.
- The ARPACK library – a collection of Fortran77 subroutines designed to solve large scale eigenvalue problems http://www.caam.rice.edu/software/ARPACK/.
- FFTW library version 3.x – a fast fourier transform library http://www.fftw.org/.
- Python 2.2 or higher http://www.python.org/.
- Python Numeric (numpy) package (http://numeric.scipy.org/).
- SciPy Python package from http://www.scipy.org.

CHAPTER

THREE

Tutorial

You can communicate with PSlab either by C++ or Python API or by high-level Python interface. The last method is the easiest and most straightforward to use and does not require you to have any knowledge about the internals of the program. However, at the same time it is the most limited one and for some advanced computations, which are described at the end of this chapter you will need to use the Python API.

In this tutorial we assume that you are at least basically familiar with the Python language. If it is not the case you can read the Beginner's Guide [2]. We suggest that you take your time to learn Python at this is a very powerful programming language, making it possible to perform sophisticated simulations and postprocessing.

3.1 Creating a simple geometry

To use PSlab you must first define the structure for simulation. In order to do this you need to load the module `Geometry`. Start Python interpreter (usually by entering command `python` in UN*X shell) and type

```
>>> from pslab import Geometry
PSlab, version 0.1.1, Copyright (C) 2005 by Maciej Dems and Tomasz Czyszanowski
Developed at Laboratory of Computer Physics, Technical University of Lodz

Please note that PSlab comes with ABSOLUTELY NO WARRANTY. This is free
software, and you are welcome to redistribute it under the conditions
of GNU General Public Licence version 2, or any higher version.

>>>
```

The appearance of the banner means that program has loaded successfully. Now you can create the `Geometry3D` object. Its constructor take two parameters, which are either the two-element tuples of the lattice vectors components[1], or the x and y dimensions of the orthogonal (rectangular) unit cell. In the latter case the lattice vectors are assumed to be $[x, 0]$ and $[0, y]$. Because we use the plane-wave expansion here, the boundary condtitions in xy plane are always periodic.

We will run computation for the rectangular cell with all the distances expressed in its lattice constant, so we type

```
>>> geometry = Geometry.Geometry3D(1, 1)
>>>
```

Now, having geometry defined, let's create some layers. Assume we want to run a simulation of photonic crystal slab having refractive index equal to 3.5 with air as the cladding. As the structure have an inverse symmetry in the plane paralel to the layers, we need to define only two layers – half of the slab core and cladding.

[1] The computational cell is a parallelogram with the edges defined by the lattice vectors.

```
>>> core = Geometry.Layer(3.5**2)
>>> cladding = Geometry.Layer(epsilon=1)
>>>
```

The `Layer` class has the constructor which require at least one parameter – the permittivity of the bulk of the layer. You can also specify permeability of the layer giving it as a second parameter. When you skip it the default value 1 is assumed. If you wish you can give the parameters with thier names as in the following example, in which case the order does not need to be preserved.

As every layer must have the finite thickness (specified at the later stage) we want to add also the absorbing PML as the boundary conditions. For this reason we must specify both ϵ and μ and what more they must have form of the diagonal tensor. With PSlab this is possible – you simple give the diagonal tensor components in the list in order $[x, y, z]$

```
>>> sz = 1-2j
>>> PML = Geometry.Layer(epsilon=[sz,sz,1/sz], mu=[sz,sz,1/sz])
>>>
```

To make our structure a photonic crystal let's add an air cylinder to the core

```
>>> core.addObject( Geometry.Cylinder(center=(0,0), radius=0.3, epsilon=1) )
>>>
```

The center parameter is always a tuple with two-dimensional coordinates.

Now we are ready to arrange our layers in a stack. We do this by using method `addLayer` of our `geometry` object (mind the order of the layers — for symmetric position the symmetry plane is **before** the first layer)

```
>>> geometry.addLayer( core(0.3) )
>>> geometry.addLayer( cladding(4.0) )
>>> geometry.addLayer( PML(width=0.5) )
>>>
```

Having our structure defined lets run some simulation. For this purpose we must use the class `Simulation` from the `pslab` module

```
>>> from pslab import Simulation
>>> simulation = Simulation(geometry, size=3)
Creating Plane Wave grid...
  number of distinct layers: 3
  number of inverse vectors: [7,7], total matrix size : 98 (49 planewaves)
  whole mesh is [104x104]

Setting material coefficients for layer 0...

Setting material coefficients for layer 1...
  layer has diagonal Q-matrix :)

Setting material coefficients for layer 2...
  layer has diagonal Q-matrix :)

Creating simple diagonalizer...

Creating admittance-matrix solver...
  the stack consists of 3 layers, interface is after 0 layers
  the structure has symmetry in z direction

>>>
```

where the `size` parameter decides about the number of planewaves used. For three-dimenstional simulation $N = (2 \cdot size + 1)^2$. Now we can for example find the eigenmode around some frequency for the default wavevector (equal to zero).

```
>>> simulation.getMode(2.5)

Searching directly for the mode starting from (2.5+0j)
  searching for the mode with Broyden method starting from (2.5+0j)...
  found mode at (2.53406515-6.0354957e-16j)

>>>
```

It must be noted that in whole program frequencies are given as the $k_0 = \omega/c$ where ω is the angular frequency and c the speed of light.

The short example presented here can be found in `examples/example1.py`. Below there is the full file.

```
#!/usr/bin/python

## Import the program modules
from pslab import Geometry, Simulation

## Declare the geometry. The unit cell is a 1x1 square
geometry = Geometry.Geometry3D(1, 1)

## Fill it with some layers

# PML has anisotropic electric and magnetic tensor
sz = 1-2j
PML = Geometry.Layer(epsilon=[sz,sz,1/sz], mu=[sz,sz,1/sz])

# Cladding is a uniform layer of air
cladding = Geometry.Layer(1)

# Core has isotropic epsilon equal to 12.25 and a circular air rod
core = Geometry.Layer(12.25)
core.addObject( Geometry.Cylinder(center=(0,0), radius=0.3, epsilon=1) )

## Now construct a stack
geometry.addLayer( core(0.3) )
geometry.addLayer( cladding(4.0) )
geometry.addLayer( PML(width=0.5) )

## Define the simulation
simulation = Simulation(geometry, size=3)

## Find some mode around angular frequency 0.5 c/a
mode = simulation.getMode(2.5)

print mode
```

3.2 Non-symmetric structure

In the previous section the structure was assumed to have an inverse symmetry in z direction and the matching interface (MI) was automatically set at the outer boundary of the last layer. As the electric xy component of the electric field at the MI must have non-zero value it was possible only to analyze even TE-like and odd TM-like modes.

In the real problems it is often necessary to consider also the modes with electric field perpendicular to symmetry plane as well as the non-symmetric structure. For this purpose PSlab offers possibility to define the whole structure and set the MI at an arbitrary position.

Consider the structure from the previous example. Assume that in reality the core is not suspended in the air but has a subsrate with $\varepsilon = 2.25$ below.

```
>>> substrate = Geometry.addLayer(2.25)
>>>
```

The PML for any layer can be generate automatically with function `generatePML` which as the arguments take the adjacent layer and PML parameter s_z.

```
>>> PML2 = Geometry.generatePML(substrate, sz)
>>>
```

Now the arrangement of the layers can be cleared (assuming you continue session from the previous section) and new one set up.

```
>>> geometry.clear()
>>> geometry.addLayer( PML(width=0.5) )
>>> geometry.addLayer( cladding(4.0) )
>>> geometry.addLayer( core(0.3) )
>>> geometry.setInterface()
>>> geometry.addLayer( core(0.3) )
>>> geometry.addLayer( substrate(4.0) )
>>> geometry.addLayer( PML2(width=0.5) )
>>>
```

The new method `geometry.setInterface()` gives information about the position of the MI, which is located after the last added layer. Alternatively you can call this method giving the explicit position of MI as the number of layers before it. So in the example above you can call equivalently

```
>>> geometry.setInterface(3)
>>>
```

Please note that the `core` layer is added twice. In such case PSlab will do the time-consuming eigendecomposition for this layer only once thus saving the numerical effort. This property is particuralry useful in case of many periodic layers as e.g. DBR mirrors.

The full code of this section example is to be found in `examples/example2.py` and looks as follow:

```
#!/usr/bin/python

## Import the program modules
from pslab import Geometry, Simulation



## Define the lattice vectors of unit cell
a1 = (1,0)
a2 = (0,1)


## Declare the geometry
geometry = Geometry.Geometry3D(a1, a2)


## Fill it with some layers

# PML has anisotropic electric and magnetic tensor
sz = 1-2j
PML = Geometry.Layer(epsilon=[sz,sz,1/sz], mu=[sz,sz,1/sz])

# Cladding is a uniform layer of air
cladding = Geometry.Layer(1)

# Core has isotropic epsilon equal to 12.25 and a circular air rod
core = Geometry.Layer(12.25)
core.addObject( Geometry.Cylinder(center=(0,0), radius=0.3, epsilon=1) )


# The substrate
substrate = Geometry.Layer(2.25)

# The PML can be generated automatically
PML2 = Geometry.generatePML(substrate, sz)


## Now construct a stack
geometry.addLayer( PML(width=0.5) )
geometry.addLayer( cladding(4.0) )
geometry.addLayer( core(0.3) )
geometry.setInterface()
geometry.addLayer( core(0.3) )
geometry.addLayer( substrate(4.0) )
geometry.addLayer( PML2(width=0.5) )


## Define the simulation
simulation = Simulation(geometry, size=3)


## Find some mode around angular frequency 0.5 c/a
mode = simulation.getMode(2.5)

print mode
```

CHAPTER

FOUR

High level Python interface

The easiest way od using `PSlab` is with the high level Python interface. This is a set of Python classes and functions degingned to aid the definition of your simulation geometry and running the calculations. The following chapter describes in details these classes.

4.1 `pslab.Geometry` — definition of the structure geomety

The `pslab.Geometry` module defines the three groups of classes necessary to define the simulation geometry — geometry itself, the layers and geometrical objects. Below there is the list of classes of all these three groups.

4.1.1 Geometry Types

All the geometry types are derived from the abstract class `Geometry` and provide the following methods:

`addLayer``(`*layer*`(`*width*`))`
: Add a new layer to the geometry setting its width. The *layer*(*widht*) is the recommended way of specyfing the parameters as is the most clear and flexible. You may hovever call the method the following way `geometry.addLayer(`*layer*`,`*width*`)`.

  It is recommenced to add the same layer several times when they repeat in the structure. This can reduce the computation time significantly. Note that this must be exactly the same object. Even if you create two different objects even with the same parameters, all the necessary calculation will be repeated for them separatelly.

`clear``()`
: Clear the stack. That layers are deleted if no other variable references them.

`generateStack``()`
: Generate the list of unique layers, list of indices of layers in the stack and their heights. You usually don't need to call this method individually. This is used when providing the data for the program.

`setInterface``(`[*where*]`)`
: Set the matching interface after the last layer or at the position denoted by *where*.

`interface`
: The position of the matching interface.

`smooth`
: The smoothing parameter for the goemetry. In order to improve the convergence of the solution the material permittivity and permeability profile is covoluted with Gaussian function. The `smooth` is the half-width of the Gaussian function. It's value have to be chosen experimentically — the larger it is the more convergent plane-wave expansion you get but at the same time, the less sharp edges of the objects become.

The following classes can be instantiated:

class `Geometry2D` (*L*)

The simple Cartesian 2D geometry. When you use this class you assume that the electromagnetic field has analytical form along x-axis which is

$$\Psi(x, y, z) = \Psi(y, z) \exp(i\beta x)$$

Ψ being any of the field components. The computational domain of each layer has one dimension of the length given by *L*.

The only possible object in this geometry is the `Rectangle`.

class `Geometry3D` (*A1, A2*[, *symmetry*])

The most general three-dimensional Cartesian geometry. *A1* and *A2* are the 2D vectors defining the computational domain. They should be tuples of two numbers, usually of the form `(a1,0)` and `(0,a2)`. Alternatively you can specify both *A1* and *A2* as single numbers in which case the computational domain is assumed to be rectangle of the size *A1* $\times$ *A2*.

You can define `Cylinder` and `Cuboid` for this geometry.

The optional argument *symmetry* defines whether the structure and electromagnetic field has inversion symmetry in the plane perpendicular to one or both lattice vectors. In the former case set *symmetry* to the simple symmetry definition (see below) and in the latter to the tuple `(`*symmetry1*`,` *symmetry2*`)`, where *symmetry1* and *symmetry2* are symmetry definitions along b_1 and b_2 axes respectively.

In each case symmetry definition is either **'HE'** or **'EH'**. The former means that the H_x and E_y componenst are symmetric and E_x and H_y antisymmetric and the latter is just an opposite.

4.1.2 Layer

Each geometry defined in the `PSlab` consists of several *layers*. They are assumed to be uniform along z-axis and can have any number of *objects* defined in xy-plane.

class `Layer` (*epsilon*[, *mu*])

The layer class defines the single layer. The *epsilon* parameter is a permittivity value of a bulk material of the layer and *mu* is its permeablity (which defaults to 1).

Here and in any geometry objects both *epsilon* and *mu* can be defined either as a single isotropic values or as 3-element sequences with diagonal elements or permibility or permeability diagonal tensors.

The layer has the following methods:

`addObject` (*obj*)

Add a new object to the layer.

`getPWexpansion` (*G, geometry*)

Compute the plane-wave coefficients of the layer. *G* must be a sequence of wavevectors and the *geometry* is one of the abovementioned geometry objects.

In practice you should not call this method yourself. It is used when necessary in the simulation.

`__call__` (*width*)

Return a tuple `(`*layer*`,` *width*`)` whete *layer* is the layer object itself. This method is here to allow you the neat way of adding layers to geometry as `geometry.addLayer(layer(width))`.

4.1.3 Objects

There are several objects which can be used in the `PSlab`. They are all subclasses of either `_GeometryObject2D` or `_GeometryObject3D` which consequently are derived from `_GeometryObject`. All the geometry objects don't have any useful methods other than constructors in which all the necessary informations concerning position, size and material should be given.

There are some common parameters which can be provided to the constructor of any object:

epsilon is the value of dielectric constant of the object. It can be either a single isotropic value or a sequence of three diagonal elements of anisotropic diagonal tensor (non-diagonal anisotropy is not supported by the `PSlab`).

mu the magnetic constant of the object. It is set in simmilar way as *epsilon*, but can be omitted in which case its default value is 1.

smooth the smoothing parameter of the object. The default value is the `smooth` parameter of the geometry. See the desctiption of `smooth` member in section 4.1.1 for more details.

In two-dimensional geometry (`Geometry2D`) there is only one possible object to be defined:

class `Rectangle` (*self, left, right, epsilon*[, *mu*[, *smooth*]])
Rectangle in yz plane in `Geometry2D`. The *left* and *right* are the positions of the sides of the rectangle.

3D geometriy (`Geometry3D`) can contain the following objects:

class `Cylinder` (*center, radius, epsilon*[, *mu*[, *smooth*]])
Cylinder with axis perpendicular to the layers. The *center* must be a sequence of two numbers containg x and y coordinate of the cyulinder center. The meaning of *radius* is obvious ;).

class `Cuboid` (*corner1, corner2, epsilon*[, *mu*[, *smooth*]])
A cuboid with it's sides paraller to x and y axis. *corner1* and *corner2* should both be sequences of two numbers containg x and y coordinates of the oposite corners. If $corner1 = x_1, y_1$ and $corner2 = x_2, y_2$, then all the corners of the cuboid in xy plane are (x_1, y_1), (x_1, y_2), (x_2, y_1), and (x_2, y_2).

4.2 `pslab` — the main module

CHAPTER

FIVE

Low level C++ API classes

CHAPTER

SIX

Utilities

BIBLIOGRAPHY

[1] M. Dems, R. Kotynski, and K. Panajotov, "PlaneWave Admittance Method – a novel approach for determining the electromagnetic modes in photonic structures," *Opt. Express*, vol. 13, pp. 3196-3207, 2005, http://www.opticsexpress.org/abstract.cfm?URI=OPEX-13-9-3196.

[2] Python Tutorial, http://docs.python.org/tut/tut.html.

[3] Python Reference Manual, http://docs.python.org/ref/ref.html.

APPENDIX

A

Extending `PSlab`

A.1 Definig you own geometry objects

INDEX