A-FMM User Guide

Marco Passoni

Contents

Chapter 1

Algorithm

Aperiodic-Fourier Modal Method is a frequency domain method based on the Scattering Matrix approach.

This method can be used to solve Maxwell equations on any structure, provided it can be represented as a multilayer. Inside each layer, the dielectric constant along the stacking direction (let's call it z) has to be uniform. Variation of the dielectric constant in the plane perpendicular to the stacking direction (the xy) plane is instead permitted.

The basic working principle is as follows. Firstly, the fields in each layer (for now assumed infinite in the z direction) are expressed as a sum of forward and backward propagating plane waves (expansion on a Fourier basis). Secondly, the boundary conditions are applied to each interface, linking together the fields in each layer. The output of the code is the scattering matrix S, a matrix that connects the Fourier expansion of the fields outgoing from the structure to the one of the incoming fields:

$$\begin{bmatrix} u_N \\ d_0 \end{bmatrix} = S \begin{bmatrix} u_0 \\ d_N \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} u_0 \\ d_N \end{bmatrix} \tag{1.1}$$

where u_i and d_i are the coefficient of the Fourier expansion in the i^{th} layer for the forward and backward propagating waves.

If a total of M elements of the Fourier expansion are retained, each u_i and d_i is a $2M$ vector (the factor 2 account for the two different polarization states) and the Scattering Matrix S is a $4M \times 4M$ matrix.

Once the scattering matrix of a system is known, information such as reflection, transmission and diffraction in the open orders can be extracted. Since the scattering matrix gives the coupling coefficient between the input and output

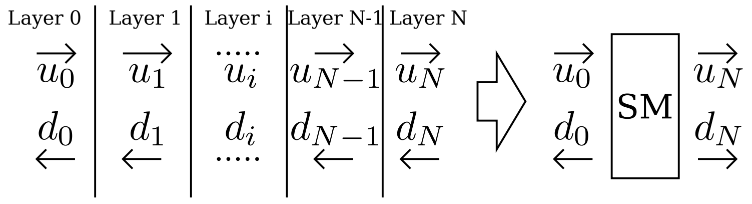

Figure 1.1: scattering matrix

Fourier modes, getting such information is just a matter of taking the correct element in the scattering matrix and normalizing it with the ratio of the Poynting vectors between the selected modes.

The A-FMM has its roots in the classical modal method for crossed gratings, firstly devised by L. Li [1] in 1997. The method was then updated by J.P. Hugonin and P. Lalanne [2] with the inclusion of a coordinate transformation. The role of the coordinate transformation is to map the entire $\mathbf{R}$ space ($\mathbf{R}^2$ for 3D calculation) to the finite computational cell. In this way the computational cell can contain an arbitrary dielectric constant distribution, which can still be treated with Fourier Methods but without crosstalk between adjacent cells.

The procedure involved in the A-FMM can be summarized as:

1. Dividing the system in N layers.
2. Solving the Maxwell equations in each layer, assuming homogeneity in the staking (z) direction, expanding the filed on a Fourier Basis.
3. Using the solution found in 2, calculating the propagation matrix in each layer and the scattering matrix of each interface.
4. Using the recursion algorithm for the scattering matrix, calculating the scattering matrix of the full structure.
5. Obtaining the desired quantities by post-processing of the scattering matrix.

A brief summary of every step will now be reported. Firstly, it is necessary to expand the fields in each layer as a sum over pseudo-Fourier basis :

$$\Phi(x,y,z) = \sum_{n,m} = \Phi_{n,m}(z) \exp\left\{ i(k_x + \frac{2\pi n}{L_x})x + i(k_y + \frac{2\pi m}{L_y})y \right\} \tag{1.2}$$

where Φ is a generic component of the electromagnetic fields, k_x, k_y are the components of the wavector along the x, y direction. The values L_x and L_y are the dimension of the computational cell. Due to the coordinate transformation they lack a physical meaning, but their values are still important in the definition of the basis for the pseudo-Fourier expansion, and remain as computational parameters.

Once the expansion is established, it can be inserted in the Maxwell equations to get the solution inside the single layer. After this substitution and a little manipulation it is possible to write:

$$-\frac{i}{k_0}\frac{\partial}{\partial z}\begin{bmatrix} E_x \\ E_y \end{bmatrix} = \begin{bmatrix} \bar{K_x}\varepsilon^{-1}\bar{K_y} & 1 - \bar{K_x}\varepsilon^{-1}\bar{K_x} \\ \bar{K_y}\varepsilon^{-1}\bar{K_y} - 1 & -\bar{K_y}\varepsilon^{-1}\bar{K_x} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \end{bmatrix} = F \begin{bmatrix} H_x \\ H_y \end{bmatrix} \tag{1.3a}$$

$$-\frac{i}{k_0}\frac{\partial}{\partial z}\begin{bmatrix} H_x \\ H_y \end{bmatrix} = \begin{bmatrix} -\bar{K_x}\bar{K_y} & \bar{K_x}^2 - \varepsilon_{y,x} \\ \varepsilon_{x,y} - \bar{K_y}^2 & \bar{K_x}\bar{K_y} \end{bmatrix} \begin{bmatrix} E_x \\ E_y \end{bmatrix} = G \begin{bmatrix} E_x \\ E_y \end{bmatrix} \tag{1.3b}$$

where E_x, E_y, H_x, H_y are the vectors containing the Fourier expansion coefficients of the respective field, and $k_0 = \omega/c$ is the vacuum wavevector of the

[1] Li, Lifeng. "New formulation of the Fourier modal method for crossed surface-relief gratings." JOSA A 14.10 (1997): 2758-2767.

[2] Hugonin, J. P., et al. "Fourier modal methods for modeling optical dielectric waveguides." Optical and Quantum Electronics 37.1-3 (2005): 107-119.

radiation. The matrices $\varepsilon, \varepsilon_{x,y}$ and $\varepsilon_{y,x}$ are built with the Fourier transform of the dielectric function in the computational cell — see ref. [?] and App. **??** for further details. The matrix $\bar{K_x}$ is built as:

$$\bar{K_x} = F_x K_x, \tag{1.4}$$

where K_x is simply the matrix containing the wavevectors of the Fourier expansion ($K_x(g, g') = (k_x + g_x)\delta_{gg'}$) and F_x is built from the Fourier coefficient of the derivative of the function used for the coordinate transformation — see App **??** for details. The matrix F_y is built in an analogous way.

A further summary of Eq. 1.3a and 1.3b is desirable. Focusing on solutions of Ψ which have a z dependence in the form $e^{i\gamma z}$ it is possible to write:

$$FG \begin{bmatrix} E_x \\ E_y \end{bmatrix} = \frac{\gamma^2}{k_0^2} \begin{bmatrix} E_x \\ E_y \end{bmatrix}, \tag{1.5}$$

which takes the form of an eigenvalue problem concerning only the electric field components. The eigenvalues γ^2/k_0^2 are related to the propagation constant along z of the eigenmodes of the layer, while the eigenvectors give the electric field profile of th modes. Since for every eigenvalue two solutions for γ are possible — forward and backward propagating — only the solution satisfying $Re(\gamma) + Im(\gamma) > 0$ will be considered and labeled with subscript q.

Defining the vector $E = [E_x E_y]^T$ and the matrix $\tilde{E}$ containing as columns the eigenvectors of Eq. 1.5, the corresponding matrix $\tilde{H}$ for the magnetic filed can be expressed as:

$$\tilde{H} = \frac{k_0}{\gamma} \tilde{E} \tag{1.6}$$

which contains the Fourier coefficient regarding the magnetic filed of the eigenmodes.

At this point it is possible, after the definition of the vectors u and d, containing, respectively, the amplitude of the forward and backward propagating modes inside the layer, to write the Fourier coefficients of the field at an arbitrary value z as a function of u and d:

$$\begin{bmatrix} E \\ H \end{bmatrix}_z = \begin{bmatrix} \tilde{E} & \tilde{E} \\ \tilde{H} & -\tilde{H} \end{bmatrix} \begin{bmatrix} u \\ d \end{bmatrix}_z \tag{1.7}$$

Currently a method of connecting the vectors u and d at different z values (both in the same layer or in different ones) is still lacking. In the following the scattering matrix for both cases will be built.

The easiest case is when the two values of z are in the same layer. In this condition, the values of u and d at z and z' are simply linked with a propagation scattering matrix:

$$\begin{bmatrix} u' \\ d \end{bmatrix} = \begin{bmatrix} \exp[i\Gamma(z - z') & 0 \\ 0 & \exp[i\Gamma(z - z')] \end{bmatrix} \begin{bmatrix} u \\ d' \end{bmatrix} \tag{1.8}$$

where Γ is simply the matrix with the propagation constants of the modes γ on the diagonal.

When the two z values lie in different layers the calculations are a bit more complex. However, is it possible to demonstrate that the scattering matrix of a single interface between two layers can be written as (normal variables refer to the first layer, the primed to the second):

$$\begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} = \begin{bmatrix} T_+ - T_- T_+^{-1} T_- & T_- T_+^{-1} \\ -T_+^{-1} T_- & T_+^{-1} \end{bmatrix}, \tag{1.9}$$

where $T_\pm$ is defined as:

$$T_\pm = \frac{1}{2}\left(\tilde{E'}^{-1}\tilde{E} \pm \tilde{H'}^{-1}\tilde{H}\right). \tag{1.10}$$

Now that both propagation and interface scattering matrices are defined, in order to connect the u and d vectors at different z values all that is needed is a recursion algorithm to join together two matrices, allowing the scattering matrix of the complete system to be built up piece by piece. In details, the recursion algorithm joining to scattering matrices S and s in the total scattering matrix S^T is:

$$S^T_{11} = s_{11}(1 - S_{12}s_{21})^{-1}S_{11} \tag{1.11a}$$

$$S^T_{12} = s_{12} + s_{11}(1 - S_{12}s_{21})^{-1}S_{12}s_{22} \tag{1.11b}$$

$$S^T_{21} = S_{21} + S_{22}(1 - s_{21}S_{12})^{-1}s_{21}S_{11} \tag{1.11c}$$

$$S^T_{22} = S_{22}(1 - s_{21}S_{12})^{-1}s_{22} \tag{1.11d}$$

It is worth noting that this relation, together with the fact that no diverging exponential appears in the propagation matrix 1.8, ensures the numerical stability of the S-matrix, at variance with other schemes (i.e. T-matrix).

Once the scattering matrix of the entire system is known, informations such a coupling strength between one input mode and one output mode are easily calculated. As an example, the fraction of power scattered from a mode m to a mode n can be calculated as:

$$\frac{P_n}{P_m} = |S(n,m)|^2 \frac{\mathcal{P}_n}{\mathcal{P}_m} \tag{1.12}$$

where $S(n,m)$ is the relevant S-matrix element and the quantities $\mathcal{P}_i$ are the z-component of the Poyting vector referring to the i mode. This component can be calculated as:

$$\mathcal{P} = \frac{1}{2} Re\left[E_x^T H_y^* - E_y^T H_x^*\right] \tag{1.13}$$

where E_x, E_y, H_x and H_y are the vectors of the Fourier components of the mode under consideration.

Speaking of the Poynting vector, it is also possible to calculate the total Poynting vector as a function of z. This is a little more complex than before (where only the Poyting vector relative to a single mode was calculated) since now every mode (forward and backward propagating), together with their interference, has to be considered. In the end, the total Poyting vector $\mathcal{P}_z^{TOT}$ can be expressed as:

$$\mathcal{P}_z^{TOT} = \frac{1}{2} Re\left\{\begin{bmatrix} u & d \end{bmatrix}_z \begin{bmatrix} \tilde{\mathcal{P}} & -\tilde{\mathcal{P}} \\ \tilde{\mathcal{P}} & -\tilde{\mathcal{P}} \end{bmatrix} \begin{bmatrix} u^* \\ d^* \end{bmatrix}_z\right\} \tag{1.14}$$

where $\tilde{\mathcal{P}}$ is a matrix whose elements are defined as:

$$\tilde{\mathcal{P}}_{nn'} = \sum_k \left[\tilde{E}_x(k,n)\tilde{H}_y^*(k,n') - \tilde{E}_y(k,n)\tilde{H}_x^*(k,n')\right] \tag{1.15}$$

However, the practical use of Eq. 1.14 to get the total Poyting vector requires the calculation of u_z and d_z, namely the vectors containing the coefficients of the forward and backward propagating modes at z. In order to do that, two S-matrices have to be built. The first S is the matrix linking the beginning of the structure to the desired z coordinate, while the second S' links this z coordinate to the end of the structure. Once those S-matrices are known, the vectors u_z and d_z can be calculated from the input vector u and d' as:

$$u_z = \left(1 - S_{12}S'_{21}\right)^{-1} \left(S_{11}u + S_{12}d'\right) \tag{1.16a}$$

$$d_z = \left(1 - S'_{21}S_{12}\right)^{-1} \left(S'_{21}u + S'_{22}d'\right) \tag{1.16b}$$

One last, but very important, application of the scattering matrix is the calculation of the photonic bands in periodic structures. This can be done by solving a generalized eigenvalue problem derived from the S-matrix of the unit cell. The eigenvalue problem is obtained imposed Bloch boundary conditions on the fields on the two side of the structure:

$$\begin{bmatrix} u' \\ d' \end{bmatrix} = e^{ik\Lambda} \begin{bmatrix} u \\ d \end{bmatrix} \tag{1.17}$$

where k and Λ are the Bloch vector and the period along the z direction, respectively. Inserting the Bloch boundary condition in the definition of the scattering matrix (Eq. 1.1) easily yields:

$$\begin{bmatrix} S_{11} & 0 \\ S_{21} & -1 \end{bmatrix} \begin{bmatrix} u \\ d \end{bmatrix} = e^{ik\Lambda} \begin{bmatrix} 1 & -S_{12} \\ 0 & -S_{22} \end{bmatrix} \begin{bmatrix} u \\ d \end{bmatrix} \tag{1.18}$$

which can be solved to obtain the Bloch vector k for given frequency.

The Aperiodic Fourier Modal Method is the main instrument for the research on slow light in silicon grating waveguides, both for calculation of photonic bands and transmission through finite size structures.

Chapter 2

The software

2.1 examples

2.1.1 Mode of a slab waveguide

File: `slab_modes.py`
This example is designed to illustrate the basic usage of the `creator` and `layer` classes. This script calculates the effective indexes as a function of energy of a planar slab waveguide. The script works as follows:

1. Some parameters (truncation order, energy vector) are defined;
2. A creator instance `cr` is initialized;
3. The structure is defined calling the slab method of `cr` — a number of methods for the creator class are built in the code;
4. A layer instance in created;
5. The transformation of coordinate is applied;
6. Loop over energies is started, inside the loop:
   (a) The `mode` method of the `layer` class is used to calculate the modes;
   (b) Effective indexes are retrieved from the `gamma` vector of the `layer` instance and sorted;
   (c) Results are printed;

2.1.2 Mode of a ridge waveguide

File: `ridge_modes.py`
This example has the same structure as the previous one, but the analyzed structure is a ridge waveguide, thus requiring a 2D cell.

2.1.3 Reflection and transmission of a 1D grating. Wavelenght and angle

File: `1D_grating.py`
This example is designed to illustrate the basic usage of the stack class and

how to calculate coupling between different input and output modes (such as reflection and transmission). The scripts is organized as follows:

1. Definition of the relevant parameters (truncation order, vectors for energies and angle sweeps);
2. Definition of the layers used in the structure;
3. Creation of and instance if the `stack` class, defining the full structure and containing all the methods for solving it:
   (a) A list of layers composing the structure is contructed;
   (b) A list of the thicknesses of each layer is constructed (length the same as list of layer). The thickness values of the first and last layers are useful only for plotting purposes, and have no effect whatsoever on the calculation;
   (c) A stack object is created using the these two lists as inputs;
   (d) The method `count_interface` of the `stack` object is called. This is a necessary step right after a stack definition;
   (e) The method `plot_stack` is called to plot the reconstructed epsilon of the structure, output in a pdf file;
4. The loops over energies and angle are started:
   (a) The scattering matrix is constructed calling the `solve` method of the `stack` class. Inputs in this case are energy (mandatory) and parallel wavevector (optional, assumed 0 if missing).
   (b) Reflection and transmission between relevant modes are then extracted using the appropriate methods (`get_T` and `get_R`).
   (c) Results are printed to screen.

Some notes on this script have to be made. First, note than to set the angle of incidence it is necessary to specify the parallel $x(y)$ wavevector. This wavevector is expressed in unit of $1/L_x(1/L_y)$ and the index of refraction has to be inserted manually. Second, an observation on the reading of the results with `get_R`. When the first an last layer are not patterned, the eigenstates of those layer are simple plane waves (in number 2*NPW, due to polarization). The first NPW modes are polarized (E field) along x, the second NPW along y. Inside each bloch, the modes are ordered following the dictionary G (which can be printed de-commenting the commented lines in the example). The 0,0 order is at the center of each block, thus at NPW/2 and 3/2*NPW (which one is TE and TM depends on which wavector is inserted in the calculation). Furthermore, it is important to use the option `ordered='no'`, otherwise the modes are ordered in decreasing effective index, resulting in a lack of consistence when a sweep in angle is made.

2.1.4 Reflection and transmission of a 1D grating. Wavelenght and angle

File: `2D_grating.py`
This is the 2D version of the previous example. It requires much more computational effort, but is otherwise totally identical.

2.1.5 PML implementation test: reflection and transmission in waveguide

File: `pml_test.py`
This script illustrate the use of the complex coordinate transformation which acts as a PML around the simulation regions, allowing the treatment of structure in which the scattering losses are consistent. It is also the prototype of the calculation of transmission and reflection from a waveguide structure. The script is built following the benchmark of Hugonin *et al.*[1] The script is The script is structures as follows:

1. A series of layer instances are defined (see Ex.2.1.1), one for each different segment of the structure.
2. A stack instance is created, using as input a list of layers and one of thicknesses.
3. The complex coordinate transformation is applied in the x direction.
4. The scattering matrix is calculated (`solve`)
5. Reflection and transmission are obtained (see Ex. 2.1.3).
6. Fields inside the structure are plotted.

A few observations have to be made. This this time the function `get_R` (and `get_T`) are called with the option `ordered='yes'` (whcih is also the default option). IN this way the modes in the reference layer are ordered in decreasing effective index. Thus the 1 mode is the fundamental one one (fundamental TE in this specific case), 2 second one (fundamental TM in this case) and so on. The same syntax is used in the `plot_E` command.

[1] Hugonin, Jean Paul, and Philippe Lalanne. "Perfectly matched layers as nonlinear coordinate transforms: a generalized formalization." JOSA A 22.9 (2005): 1844-1849.

Chapter 3

Reference Guide

In this chapter a detailed explanation of the module is provided. For each submodule a summary of the classes and function involved can be found, together with details on the most common functions.

3.1 Module composition

The A-FMM module in turn composed by different submodules:

- creator module: Contain the creator class, used to define the dielectric costant of the single layer.
- layer module: Contain the layer class, used to solve the Maxwell equation inside each single layer. It requires a creator instance to be initialized.
- scattering module: Contain the S_matrix class. All method for scattering matrix creation and manipulation are here implemented.
- satck module: Contain the stack class, used to calculate the Scattering Matrix and related quantities of the hole structure. It is initialize from a collection of layer instances.
- sub_sm module: Contains auxiliary function that are called in more than one module. It is non loaded by default when loading A_FMM.

3.2 S_matrix class

3.2.1 Initialization

This class contains the definition of the scattering matrix object and all method for scattering matrix manipulation. The scattering matrix is an object relating the amplitudes of the incoming end outgoing modes of a structure. It presets itself as following:

$$\begin{bmatrix} u' \\ d \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} u \\ d' \end{bmatrix} \tag{3.1}$$

following the convention of figure 3.1. An instance is initialized as:

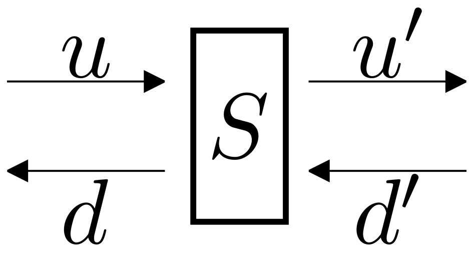

Figure 3.1: Convention adopted for the scattering matrix formalism.

```
S=S_matrix(N)
```

where `N` is the dimension of the four $N \times N$ matrices composing the scattering matrix. Each of this matrix is saved as an `numpy.ndarray` as an attribute of the S_matrix object. For example to the $(1, 2)$ element of the S_{21} matrix is accessed as `S.S21[1,2]`.

3.2.2 Method

`add(s)`

Takes as input an other instance of the S_matrix class s, which is then joined to the right to the scattering matrix that calls the method.

`add_left(s)`

Same as `add` but s matrix is joined to the left.

`add_uniform(lay,d)`

Takes as input:
- `lay`: An instance of the layer class;
- `d`: A float representing the thickness of the layer.

This method creates the scattering matrix for propagation trough a layer `lay` of thickness `d`, and then joins it to the right of the scattering matrix that calls the method.

`add_uniform_left(lay,d)`

Same as `add_uniform` but joins to the left.

3.3 Layer Class

3.3.1 Initialization

This layer class is contained in the layer module. It is responsible for definition of the single layer inside the scattering matrix approach. It also contains the methods for adding the coordinate transform an plotting of the mode fields. The different layer are:

```
lay=layer(Nx,Ny,creator,Nyx=1.0)
```

which is the standard implementation of the class. A `creator` object is needed for the creation, and all the calculation for the Fourier transform are done analytically. The parameter for the initialization are:

- `Nx,Ny`: Trucation order respectively in x and y direction. Actual number of plane waves is $(2N_x + 1)(2N_y + 1)$.
- `creator`: The creator instance describing the dielectric function.
- `Nyx`: Ratio between the cell period in y and x direction (default is 1.0).

```
lay=layer_num(Nx,Ny,func,args=(),Nyx=1.0,NX=1024,NY=1024)
```

Implementation in which the Fourier transforms for the layer are handled numerically. It is more versatile than the standard implementation, albeit should be less precise. The profile of the dielectric is obtained from the function `func`. The objects needed are:

- `Nx,Ny`: Integers, trucation order respectively in x and y direction. Actual number of plane waves is $(2N_x + 1)(2N_y + 1)$.
- `func`: Function which defines the dielectric constant. It has to be in the form `func=func(x,y,..)`. After x and y additional parameters can be defined in the function and passed in `args`. Domain of the function in -0.5:0.5 in x and -0.5*Nyx:0.5*Nyx in y.
- `Nyx`: Ratio between the cell period in y and x direction (default is 1.0).
- `args`: Tuple containing eventual additional parameters for `func`.
- `NX,NY`: Integers, number of point in each direction used for numerical integration in the Fourier transforms.

```
lay=layer_uniform(Nx,Ny,eps,Nyx=1.0)
```

Implementation of the uniform layer. It is implemented on is own because in this way its definition and solution are much faster. The parameters are:

- `Nx,Ny`: Trucation order respectively in x and y direction. Actual number of plane waves is $(2N_x + 1)(2N_y + 1)$.
- `eps`: The dielectric constant of the uniform layer. Can be complex.
- `Nyx`: Ratio between the cell period in y and x direction (default is 1.0).

When initializing the layer class all the matrices involve in the eigenvalue problem for the layer are created.

3.3.2 Fundamental Methods

`trasform(ex=0,ey=0)`

Add the real coordinate transform.

- `ex,ey`: Respectively the width of the untransformed region in x and y direction. If not specified transformation in not applied in that direction.

`trasform_complex(ex=0,ey=0)`

Add the complex coordinate transform simulating PML boundary conditions.

- `ex,ey`: Respectively the width of the untransformed region in x and y direction. If not specified transformation in not applied in that direction.

`mode(k0,kx=0.0,ky=0.0,v=1)`

Solve the eigenvalue problem of the layer.

- `k0`: Energy of the mode
- `kx,ky`: Respectively the lateral wavevector in the x and y direction (in unit of inverse of the period).
- `v`: If equal to 0 only eigenvalue are calculated. Useful when interested only in propagation constant of the mode of the single layer. Default is 1.

create three new attributes of the class layer:

- `W`: Vector of the eigenvalues (effective indexes of modes squared).
- `V`: Matrix of electric eigenvectors of the modes. `V[:,i]` is the vector to the i^{th} mode.
- `VH`: Matrix of magnetic eigenvectors of the modes. Same convention as `V`.

`eps_plot(pdf=None,N=200,s=1.0)`

Plot dielectric function profile reconstructed from fourier transform.

- `pdf`: Name of the pdf file used to save the figure (string, without the .pdf).
- `N`: Width in pixel of the cell. Default 200.
- `s`: Number of fundamental cells plotted. Default 1.

`plot_E(pdf,i,N=100,s=1,func=np.abs)`

Plot electric field profile of selected mode.

- `pdf` Instances of the PdfPages class. Specify pdf file where to save field.
- `i` Number of mode to plot. Mode are ordered in decreasing effective index. Numeration start at 1.
- `N` Width in pixel of the cell. Default 100.
- `s` Number of fundamental cells plotted. Default 1.
- `func` Since field is complex, function to apply to field before plotting. Default is abs. Useful can be real, imag, angle.

`plot_H(pdf,i,N=100,s=1,func=np.abs)`

Plot magnetic field profile of selected mode. Same inputs as `plot_E`.

`plot_field(pdf,i,N=100,s=1,func=np.abs)`

Plot both electric and magnetic field profile of selected mode. Same inputs as `plot_E`.

`get_field(x,y,i,func=np.abs)`

Return field of selected mode at selected point.

- `x,y` Coordinate of the point in which to calculate fields.
- `i` Number of mode to plot. Mode are ordered in decreasing effective index. Numeration start at 1.
- `func` Since field is complex, function to apply to field before plotting. Default is abs. Useful can be real, imag, angle.

Return 1-dim array of length 4 containing in order: Ex,Ey,Hx,Hy

`get_P_norm()`

Calculate z component of Poynting vector, used in the scattering matrix algorithm to normalize correctly reflection ad transmission. The values of Poynting vector are then stored in the new attribute `P_norm`.

`T_interface(lay)`

Takes in input a different instance of the layer class. Return the transfer matrix representing the interface between the layer that calls the method and the layer given ad input. The obtained matrix is an instance of `numpy.ndarray`.

`T_prop(d)`

Takes in input a float value `d` representing the thickness of the layer. Return the transfer matrix of the propagation trough a thickness d of the layer that calls the method. Could generate numerical instabilities. The obtained matrix is an instance of `numpy.ndarray`.

`interface(lay)`

Takes in input a different instance of the layer class. Return the scattering matrix representing the interface between the layer that calls the method and the layer given ad input. The obtained matrix is an instance of `S_matrix`.

3.4 Stack Class

3.4.1 Initialization

The stack class is contained in the stack module. It is the fundamental class in the programs. It represent the full structure of the multilayer under analysis, and contain all the method for the scattering matrix calculation and the extraction of all the relevant information. It is initialized as follows:

```
st=stack(mat,d)
```

where `mat` and `d` a a list of layers (each element is a list of the layer class) and of thicknesses (each element is a float), respectively. If `mat` and `d` have different length an error is raised. It is important that each layer has the same parameters, such as number of plane waves and computational cell aspect ratio. However application of the coordinate transformation to each layer is not needed, since it can be applied collectively with a method of the stack class.

Right afterward the initialization, the method `count_interface()` (with no argument) has to be called. This creates a number of auxiliary variable which are needed in order to perform all the calculations.

3.4.2 Fundamental Methods

Here are listed the fundamental methods of the class.

`count_interface()`

Has to be called right after initialization. Define lists of all different layers and interfaces in the structure.

`transform(ex=ex,ey=ey)`

Apply the real coordinate transform to all layers in the stack instance. Inputs are the same of `layer.trasform()`.

`transform_complex(ex=ex,ey=ey)`

The same as `transform`, but with the complex coordinate transformation.

`solve(k0,kx=0.0,ky=0.0)`

Methods for creating the scattering matrix of the structure. Inputs are:

- `k0`: Energy of the mode
- `kx,ky`: Respectively the lateral wavevector in the x and y direction (in unit of inverse of the period).

The methods creates an instance of the `S_matrix` class, which can than be accesses as `stack.S`. This version of solve is optimized for cpu usage. It avoid multiple calculation of the same layer and interface when identical layers are repeated in the structure. When a great number of different layer are present, can lead to a massive memory occupation.

`solve_serial(k0,kx=0.0,ky=0.0)`

Same as `solve`, but solve each layer and interface in the order in which they appear. It is designed to limit memory usage, at the price of doing more calculation than necessary when repeating layers are present.

`plot_stack(nome='cross_section_X',N=100,dz=0.01,y=0.0,func=np.abs)`

Create a pdf containing the xz profile of the reconstructed epsilon. Inputs are:

- `nome`: name of the pdf file
- `N`: Number of points per cell in the x direction
- `dz`: Step in the z direction
- `y`: y coordinate at which the slice is taken
- `func`: function applied to epsilon before plotting.

`plot_stack_y(nome='cross_section_Y',N=100,dz=0.01,x=0.0,func=np.abs)`

Same as `plot_stack`, but output the yz slice. Change in input parameters is straightforward.

`get_R(self,i,j,ordered='yes')`

Get reflection coefficient from the i^{th} to the j^{th} mode of the first layer. If `ordered='yes'`, modes are ordered in decreasing effective index. If `ordered='no'` no special ordering in applied. If layer is uniform, modes are plane waves and are ordered along the G dictionary (accessible as `layer.G`). Do not use the unordered version if first layer is not uniform.

`get_T(self,i,j,ordered='yes')`

Get reflection coefficient from the i^{th} mode of the first layer to the j^{th} mode of the last layer. Same considerations as `get_R` apply.

3.4.3 Manipulation methods

The methods listed here are used to manipulate existing stack instances. They can be double, flipped and joined together to create more complex structures. All these methods have to be called after a solve one, when the scattering matrix is already defined (with the same parameters for each structure).

```
double()
```

Double the structure on which is called. The first and last layer has to be the same, and the thicknesses of those layer is assumed to be 0.

```
flip()
```

Flip the structure on which is called.

```
join(st2)
```

Add the structure `st2` (another `stack` instance) to the right of the structure on which is called. Only requirement is that the last layer of the first is the same as the first layer of the second. The particular layer chosen is irrelevant, since they are sacrificed in the joining process.