Semi-Analytic-Stacking-Algorithm

Sep 10, 2019

Contents

This Software allows you to calculate the optical behavior of stacked materials. It requires you to already know the Jonas-Matrices of the complex layers in your Stack and works out their interactions. Calculations are based on Lx4x4 composite Jonas matrices, called S-matrices, and the Starproduct between them. L represents the wavelengths were you wish to calculate the behavior.

$$S = \begin{pmatrix} T_f & R_f \\ R_b & T_b \end{pmatrix}$$

T_f : Transmission Jonas matrix for light coming from the front

R_b : Reflection for the back

CHAPTER 1

Usage

The exact usage is described in example_usage.py. In general you have to define multiple Layer-Objects:

```
l1 = MetaLayer(s_mat, cladding, substrate)
l2 = NonMetaLayer(n_vec, cladding, substrate)
```

These can be Meta-Layers where you need to provide a Lx4x4 S-matrix or Non-Meta-Layers where you need to provide a vector of refractive indices's at the desired wavelengths. Then you pass the layers to a stack object and build your result:

```
s = Stack([l1,l2,...], wavelengths, cladding, substrate)
result = s.build()
```

In the case of layers cladding and substrate represent the environment in which s_mat or n_vec were measured. For Stack its what materials are blow/on-top of the Stack.

1.1 The Stack

class stack.**Layer**

Parrent class of Meta- and NonMetaLayer, contains information about which symmetry opperations will be applied.

class stack.**MetaLayer**(*s_mat*, *cladding*, *substrate*)

Class to describe a Meta-Surface in the Stack.

Parameters

- **s_mat** (*L x 4 x 4 numpy Array*) – the Lx4x4 S-Matrix of the Meta-Layer, externally simulated/measured
- **cladding** (*vector*) – containing the refraction indices of the cladding.
- **substrate** (*vector*) – containing the refraction indices of the substrate.

class stack.**NonMetaLayer**(**n_vec*, *height*)
Class to describe a homogenous isotropic or anisotropic Layer.

Parameters

- **height** (*height in (µm)*) –
- **n_vec** (*one or two vactors containing the diffraction indeces.*) – If only one vector is given homogenous behavior will be assumed.

class stack.**Stack**(*layer_list*, *wav_vec*, *cladding*, *substrate*)
Class to describe the whole Stack, contains information about the layers, cladding, substrate and further options.

Parameters

- **layer_list** (*list of Layer objects*) –
- **wav_vec** (*vector*) – The target wavelengths where the Meta-Surfaces were simulated/ measured
- **cladding** (*vector*) – The refrectiv indeces of the cladding.
- **substrate** (*vector*) – The refractiv indeces of the substrate. The first material to be hit by light.

build()
Builds all the propagation and interface matrices and multiplies them.

Returns **s_mat** – S-matrix describing the behavior of the whole stack. The dimension is HxLx4x4 when a height vector was given

Return type Lx4x4 or HxLx4x4 numpy array

build_geo(*order*)
A version of build using star_product_cascaded_geo(), change this doc_str

Returns **s_mat** – S-matrix describing the behavior of the whole stack. The dimension is HxLx4x4 when a height vector was given

Return type Lx4x4 or HxLx4x4 numpy array

create_interface(*l_2*, *l_1*)
Creates the interface S-Matrix for the transmission between two Layers

Parameters

- **l_1** (*NonMetaLayer or MetaLayer Objects*) –
- **l_2** (*NonMetaLayer or MetaLayer Objects*) –

Returns **s_mat** – interface S-Matrix

Return type L x 4 x 4 numpy array

create_interface_rot(*l_2*, *l_1*)
Creates the interface S-Matrix for the transmission between two Layers in case of rotation, uses create_interface

Parameters

- **l_1** (*NonMetaLayer or MetaLayer Objects*) –
- **l_2** (*NonMetaLayer or MetaLayer Objects*) –

Returns **s_mat**

Return type Lx4x4 S-Matrix

create_propagator(*layer*)
Creates the propagator S-Matrix

Parameters **layer** (NonMetaLayer *or* MetaLayer *object*) –

Returns **s_mat** – propagation S-Matrix

Return type H x L x 4 x 4 numpy array

order(*order*)
Returns the nth order S-Matrix of the starproduct developt via the geometric series.

Parameters **order** (*int*) –

Returns **s_out** – S-Matrix of the order'th series developt

Return type H x L x 4 x 4 numpy Array

order_up_to(*order*)
Builds a list of S-matrices up to the target order.

Parameters **order** (*int*) –

Returns **s_list**

Return type list of HxLx4x4 numpy Arrays

1.2 Operations on S-Matrices

Symmetry operations can be applied directly to the S-matrices.

smat_oparations.**flip_smat**(*s_mat*)
Flip a given S-Matrix

Parameters **s_mat** (*L x 4 x 4 numpy Array*) – S-Matrix

Returns **s_out** – flipped S-Matrix

Return type L x 4 x 4 numpy Array

smat_oparations.**mirror_smat**(*s_mat*)
Mirror a given S-Matrix

Parameters **s_mat** (*L x 4 x 4 numpy Array*) – S-Matrix

Returns **s_out** – mirrored S-Matrix

Return type L x 4 x 4 numpy Array

smat_oparations.**phase_shift**(*smat*, *ang*)
Shifting the phase of a given S-Matrix by a given angle

Parameters

- **s_mat** (*L x 4 x 4 numpy Array*) – S-Matrix
- **ang** (*float*) – rotaion angle in rad

Returns **s_out** – shifted S-Matrix

Return type L x 4 x 4 numpy Array

smat_oparations.**rot_smat**(*s_mat*, *ang*)
Rotate a given S-Matrix by a given angle

Parameters

- **s_mat** (*L x 4 x 4 numpy Array*) – S-Matrix
- **ang** (*float*) – rotaion angle in rad

Returns s_out – rotated S-Matrix

Return type L x 4 x 4 numpy Array

1.3 The Starproduct

The Starproduct between two S-matrices is defined as follows:

$$\begin{pmatrix} r_1 & u_1 \\ w_1 & s_1 \end{pmatrix} \star \begin{pmatrix} r_2 & u_2 \\ w_2 & s_2 \end{pmatrix} = \begin{pmatrix} r_2(1-u_1w_2)^{-1}r_1 & u_2 + r_2u_1(a-w_2u_1)^{-1}s_2 \\ w_1 + s_1w_2(1-u_1w_2)^{-1}r_1 & s_1(1-w_2\cdot u_1)^{-1}s_2 \end{pmatrix}$$

. The following functions just apply this definition once analytically and once by usage of a geometric matrix series. Thats possible, because

$$(1-u_1w_2)^{-1}$$

could be easily written down in a series.

`star_product.`**`star_product_analyt`**(*SIN_1*, *SIN_2*)

Calculate Lifeng Li's starproduct for two S-matrices SIN_1 and SIN_2, such that S = S1 * S2. The starproduct between two arbitrary S-matrices was precalculated analytically with Mathematica.

Parameters

- **SIN_1** (*HxLx4x4 numpy array*) – H is height_vec_len, the dimension of the height vector given to the layer object. (Most of the time equal to 1) L is wav_vec_len the number of measured wavelengths
- **SIN_2** (*HxLx4x4 numpy array*) – H is height_vec_len, the dimension of the height vector given to the layer object. (Most of the time equal to 1) L is wav_vec_len the number of measured wavelengths

Returns s_out

Return type HxLx4x4 numpy array

`star_product.`**`star_product_cascaded`**(*smat_list*)

Iteratively calculates the starproduct (Li, 1996) of N S-matrices, where N >= 2. The iteration goes the through the starproduct pair-wise, so that: S = ((((((S1 * S2) * S3) * S4) * …) * Sn-1) * Sn).

Parameters **smat_list** (*list*) – A list containing N HxLx4x4 S-matrices

Returns smat

Return type HxLx4x4 numpy array

`star_product.`**`star_product_cascaded_geo`**(*smat_list*, *order*)

A version of star_product_cascaded unsing star_product_geometric.

Parameters

- **smat_list** (*list*) – A list containing N HxLx4x4 S-matrices
- **order** (*int*) –

Returns smat

Return type An L-by-4-by-4 S-matrix.

`star_product.`**`star_product_geometric`**(*SIN_1*, *SIN_2*, *order*)

A version of star_product where the [I - a @ b]**-1 term is developed as a geometric series to the nth order.

Parameters

- **`SIN_1`** (*`HxLx4x4 numpy array`*) – H is height_vec_len, the dimension of the height vector given to the layer object. (Most of the time equal to 1) L is wav_vec_len the number of measured wavelengths
- **`SIN_2`** (*`HxLx4x4 numpy array`*) – H is height_vec_len, the dimension of the height vector given to the layer object. (Most of the time equal to 1) L is wav_vec_len the number of measured wavelengths
- **`order`** (*`int`*) –

Returns s_out

Return type HxLx4x4 numpy array

CHAPTER 2

References

[1] **J. Sperrhake, M. Decker, M. Falkner, S. Fasold, T. Kaiser, I. Staude, T. Pertsch,** "Analyzing the polarization response of a chiral metasurface stack by semi-analytic modeling", Optics Express 1246, 2019

[2] **C. Menzel, J. Sperrhake, T. Pertsch,** "Efficient treatment of stacked metasurfaces for optimizing and enhancing the range of accessible optical functionalities", Physical Review A 93, 2016

[3] **J. Sperrhake, T. Kaiser, M. Falkner, S. Fasold, T. Pertsch,** "Interaction of reflection paths of light in metasurfaces stacks",

Python Module Index

S

Index