Appendices

Here, we briefly outline the PyRAMIDS implementation in *Python* and its folder structure, which is designed to offer both ease of use for new users and transparency for experts seeking to understand or extend the underlying mathematical framework. This manual also includes pointers to the required external packages — along with version information on which the code has been tested — and provides example cases to help users get started. These include literature benchmarks, routines that reproduce the figures in this chapter, and validation tests for internal benchmarking purposes.

A Code overview

We provide the functionality described above as a package implemented in *Python* v3.12.7, bundled with benchmarks and examples. The code is organized into multiple folders. The core computations leverage the *NumPy* and *SciPy* libraries, with routines optimized through vectorization for efficiency. Additionally, we employ *Numba JIT* compilation to further enhance performance, with configurable flags allowing users to disable JIT, enable it per run, or use cached compilation. The code is compatible with *NumPy* (v1.26.4), *SciPy* (v1.15.1), and *Numba* (v0.60.0). Matplotlib is used for visualization. We note that using cached compilation is recommended for most users, as it benefits from the speed-up of compilation yet avoids the time penalty of compiling the core code. Instead, for users wishing to perform core code development, it is recommended to disable JIT while debugging. Users can create a *Python* environment and install the required package versions using `pip install -r [package==version]` command. The folder structure is organized as follows:

Library The core of the package contains all routines that implement the underlying mathematical framework and computational engines, as discussed in Part 2.4.

Benchmarks Contains benchmark tests to validate the code by reproducing many examples from the literature. Additionally, it includes internal benchmark scripts for cross-checking the self-consistency of different aspects of the code.

Example A collection of test cases demonstrating the package's capabilities, including scripts to reproduce key results from the main text.

UserSandbox A workspace for users to create and build their examples and routing using templates from the Benchmark and Example folders. We also provide scripts that allow users to run all benchmark and example scripts in a single execution.

B Library folder

The **Library** folder is subdivided into **Core**, **Utility**, and **Use** subdirectories. This modular structure follows a hierarchical organization based on functionality:

Core folder Contains low-level scripts that implement the fundamental mathematical framework discussed in previous sections. These scripts include functions implementing the S-matrix formalism, local density of states (LDOS) calculations, far-field radiation pattern, and slab Green functions. These scripts form the computational backbone of the package. It is not recommended for the user

to call these routines directly in light of the fact that their inputs are specified in the slab-centric coordinate system, and arguments are not type-checked.

Utility folder Provides the helper functions that bridge the **Core** and **Use** modules. These scripts handle argument formatting, translate between global coordinates and slab-centric coordinates, verify the physical relevance of user inputs, and perform data validation. We also provide visualization routines for plotting far-field radiation patterns in different basis (s-p or Cartesian) and Stokes parameters.

Use folder Designed to be called by users of the packages, this subdirectory contains high-level scripts that offer streamlined access to the **Core** functionality scripts. Inputs are specified in the global stack coordinate system. Internally, these scripts leverage the **Utility** wrappers to format inputs before calling the appropriate **Core** functions, ensuring efficiency and minimizing redundancy.

This structured approach improves usability while maintaining computational efficiency and robustness, allowing both expert users and beginners to interact effectively with the package. Moreover, the modular design allows advanced users to modify the core functions without disrupting the framework. Next, we discuss pointers to the role of different routines in each script.

C Core folder

There are four script files with the *slab-centric* coordinate system, namely:

`Core_Smatrix.py` Using the S-matrix formalism as described in section 2.4.3, this script contains routines that compute the optical response of an arbitrary stratified multilayer system for any input $k_{\parallel}$. The implementation routine includes the construction of the interface and layer S-matrices and recursively building the composite S-matrix for a multilayer stack, Redheffer star product for both s- and p-polarizations. Additionally, the routines also calculate the reflection (r) and transmission (t) coefficients in the semi-infinite surrounding media and determine the field coefficients inside a specified layer.

`Core_ImGLDOS.py` implements computing the local density of states (LDOS) for electric, magnetic, and magnetoelectric dipoles in a stratified multilayer stack, following section 2.4.6, defining the integrands as described above, defining the complex contour integration trajectory in equation 2.27, and calling scipy's quadvec integration function to perform adaptive integration.

`Core_Radiationpattern.py` As discussed in section 2.4.7, this script contains routines for computing the far-field radiation pattern of an embedded electromagnetic dipole source in a stratified system using the asymptotic dyadic Green function. It returns the radiation pattern for an arbitrary magnetoelectric $(p_x, p_y, p_z, m_x, m_y, m_z)$ dipole vector (complex-valued, specified in Cartesian coordinates), corresponding to a point dipole placed at $(x, y) = 0$, at arbitrary on-axis location z in the stack. The fields are computed in the s- and p-polarized basis for specified θ, ϕ observation angles, with fluxes derived from the fields, accordingly. Additionally, the script includes a routine for integrating fluxes across all angles in each half-space to obtain the integrated power radiated in each half-space and the total radiated power.

`Core_Greenslab.py` This script contains routines for evaluating the generalized dyadic Green functions in a three-dimensional slab structure, where both the emitter and detector are within the same layer. The implementation follows the derivation as described in Section 2.4.5 exactly.

D Utility folder

This folder contains three scripts:

`Util_argumentchecker.py` This script contains routines for validating function arguments and ensuring the consistency of user inputs (including checks for real or complex values, dipole moment dimensions, and layer specifications). It also reshapes input arrays to align with the `Core` routines.

`Util_argumentwrapper.py` This script provides routines for converting the coordinate system from a user-centric to a slab-centric framework for a given source position z, ensuring compatibility with the routines in the Core folder.

`Util_vectorpolarization.py` This script contains routines for plotting far-field radiation patterns and performing Stokes polarimetry in the back-focal plane, replicating the observations in Fourier microscopy measurements. It enables visualization of angular emission properties and polarization characteristics of dipole sources in stratified systems.

E Use folder

The scripts in this section contain the functions that the package user is intended to interact with. These scripts take as input the *user-centric* coordinate system of the stratified system with the first interface at $z = 0$ and source positions in absolute coordinates, and under the hood, call the utility wrapper routines and the core routines. There are four function files with different application domains:

`Use_Planewave.py` This script interfaces with `Core_Smatrix.py` to compute reflectance, transmittance, and absorptance based on energy balance, as well as layer-resolved absorption separately for s- and p-polarized plane waves incident on a multilayer stack at an arbitrary angle. Additionally, it includes a routine for visualizing the field distribution for any input plane wave complex field on a specified Cartesian grid.

`Use_LDOS.py` This script interfaces with the core routine `Core_ImGLDOS.py` to compute the total local density of states (LDOS) for the canonical dipole orientations (electric and magnetic LDOS for parallel and perpendicular dipoles, magnetoelectric cross term), and for any arbitrary dipole orientation, and at any position within the multilayer stack. Additionally, it includes a routine to examine the LDOS integrand as a function of $k_{\parallel}$ and z, enabling modal analysis and the identification of guided modes in the stack.

`Use_Radiationpattern.py` This script interfaces with the `Core_Radiationpattern.py` to compute the angle-resolved far-field electric field in the s- and p-polarized basis for any dipole orientation, along with the corresponding radiated power in both semi-infinite half-spaces. Additionally, it includes routines to compute radiative LDOS by integrating the radiation

pattern. Routines are provided for arbitrary dipole orientation, as well as for radiative LDOS at the canonical dipole orientations (electric and magnetic LDOS for parallel and perpendicular dipoles, magnetoelectric crossterm).

`Use_Multiplescattering.py` This script provides routines to solve the multiple scattering problem using the theory described in Section 2.4.8, interfacing with `Core_Greenslab.py`. Given a set of coordinates for spherical particles, the static polarizability tensor for each scattering dipole is first determined, incorporating radiation-damping corrections based on the surrounding stack that satisfies the optical theorem. The script then constructs the coupling matrix, where diagonal blocks represent the inverse polarizability, and off-diagonal terms are computed via the Green function $G(r, r')$. It also provides routines to compute the driving field at each scatterer location, either from plane waves or source dipoles, and solves for the induced dipole moments by inverting the coupling matrix multiplied by the driving field. Additionally, routines are included to evaluate the work done on each dipole, enabling the calculation of extinction cross-sections after dividing the work by intensity, and a separate routine integrates the total radiated power in the far field to compute the scattering cross-section.

F Benchmarks folder

There are two subfolders, namely

Literature folder This folder contains a collection of routines to reproduce benchmarked results from various literature sources, namely Refs. [62, 81, 111, 127, 130–132, 135, 158, 162–165].

Internal_Consistency folder This folder contains routines that do not refer to literature but are required for internal consistencies, e.g., between LDOS and far-field integrated power.

In more detail, the internal consistency checks include

`InternalBenchmark_LDOSvsRadpat.py` For any non-absorbing stratified system without guided modes, the radiative LDOS obtained from the radiation pattern integral must be equivalent to the LDOS computed from the imaginary part of the Green function (ImG). This equivalence serves as a fundamental internal consistency check. Given that radiation patterns arise from coherent field superposition, achieving this equality when summing absolute value-squared terms represents a nontrivial validation step. To verify this rigorously, the routine conducts extensive tests with various dipole orientations, including purely electric, purely magnetic, and mixed magnetoelectric dipoles with controlled phase differences. Across all cases, one observes matching between the two LDOS calculations throughout the structure. To our knowledge, this constitutes the *first comprehensive benchmark for magnetoelectric LDOS in stratified media*.

`InternalBenchmark_GreenvsLDOSvsRadpat.py` Similar to the validation routine for non-absorbing stratified systems, this implementation extends the consistency check by comparing LDOS obtained from three independent methods: the radiation pattern integral, the imaginary part of the Green function (ImG

LDOS), and the Green function computed via the angular spectrum representation (Green slab LDOS).

`InternalBenchmark_Green.py` This routine validates the Green slab calculation using the angular spectrum method, showing that in a homogeneous medium, the Green function that is retrieved is identical to the analytical dyadic Green function. For a layered system with a single interface, the routine verifies the correctness of the scattered Green function components against a reference implementation of the single interface case [52] and confirms the expected symmetry properties when the interface is flipped.

`InternalBenchmark_PseudoChiralRotationRadPattern.py` Eq. 2.34 provides insight into the handedness of pseudochiral dipoles. The formula suggests that handedness is determined by maintaining a consistent sign in the imaginary component of the cross-term. When the dipole configuration is rotated, the sign of the imaginary part must be preserved for the handedness to remain the same. For example, if we define ($p_x = 1$, $m_y = 1j$) as left-handed, its rotated counterpart with the same handedness is ($p_y = 1$, $m_x = -1j$) in a homogeneous medium. We see exactly similar behavior in the S_3 radiation pattern for both the dipole orientation cases.

`InternalBenchmark_multiplescattering.py` This routine executes multiple scattering calculations for a set of dipole scatterers in a multilayer stack. A consistency check is to validate the optical theorem in a multiple scattering scenario within a stratified structure without guided modes and for non-absorbing scatterers. In that case, the total integrated far-field radiation must match the extinction.

G Other folders

The "Examples" folder contains (1) various `*.py` routines that demonstrate how to call different functions and utilize the package effectively, collected in the RoutineUse_Examples folder. The folder PaperReproduce_Examples contains all the routines to produce the figures in this chapter. Finally, we provide the folder "UserSandbox" as a folder in which users can create their own problem-specific scripts, for instance, based on the example scripts in the Example folders. The UserSandbox folder also contains scripts to execute all `*.py` files from the Benchmark and Example folders in a single run.