SPlaC package v1.0
guide and supplementary information

Eric Le Ru and Pablo Etchegoin

December 3, 2008

Contents

This guide is part of SPlaC v1.0, the SERS and Plasmonics Codes package for Matlab. More information can be found on the website:

http://www.victoria.ac.nz/raman/book/codes.aspx

The first section contains general information on the package and is essentially a repeat of the content of the readme file. The rest of this guide contains some supplementary information on specific aspects of the implementation, in particular in the relation to Mie theory. Note that many of the concepts and formulae used in these codes are discussed in the book: E. C. Le Ru and P. G. Etchegoin, *Principles of Surface-Enhanced Raman Spectroscopy and Related Plasmonic Effects* (Elsevier, Amsterdam, 2009). These will not be repeated here.

Note finally that the codes can be used independently of the book or even of this guide. These are only required for a detailed understanding of the codes and possibly for adapting/rewriting them to specific needs.

1 Introduction

1.1 License

1.2 Description and overview

This package contains a suite of Matlab codes to carry out various EM calculations relevant to SERS and plasmonics. Most of the underlying theory and relevant formulae are described in the aforementioned book (see license above). The relevant equations and sections from the book are referenced when possible in the "inline comments"to the codes. However, the codes can also be used independently of the book (most of the theory can also be found in standard EM textbooks).

This package includes:

- 30 self-running MatLab codes (functions) to reproduce most of the "theoretical" figures of the book
  (in directory `BookFigures`).
- Analytical expressions for the optical properties of silver (Ag) and gold (Au) in the visible/NIR region
  (in directory `General`).
- Codes to evaluate reflection/refraction at multilayer planar interfaces (in directory `Plane`).
  These include local-field calculations and example scripts for Surface Plasmon Resonances (SPRs) in the Otto and Kretsmann configurations.
- Ellipsoids in the electrostatics approximation
  (in directory `Ellipsoids`).
  These include local-field calculations relevant to SERS (average and maximum enhancement factors) for prolate and oblate spheroids, and for general ellipsoids.
- Mie theory codes for plane wave excitation
  (in directory `Mie/PWE`).
  These include radiation profile and local field calculations along with plotting tools for visualization.
- Extension of Mie theory to dipolar emission
  (in directory `Mie/DIP`).
  These include radiation profile and radiative and non-radiative enhancements calculations, along with plotting tools for visualization.
- Extension of Mie theory to spherical multilayers (e.g. coated sphere)
  (in directory `Mie/MUL`).
  These codes reuse many of the codes in `Mie/PWE` and `Mie/DIP`.
- Scripts to obtain useful symbolic expressions related to Mie theory
  (in directory `Mie/SYM`).
  The Matlab "symbolic toolbox" is necessary to run these.

1.3 Installation

- Unzip the `SPlaC.zip` file, keeping the subdirectory structure for clarity.
- Then set your Matlab current directory to `SPlaC` and run the `InitPath.m` function once in Matlab to add all the SPlaC subdirectories to your Matlab path. All the SPlaC functions and scripts are then accessible from the command line. This allows all codes to run and communicate with each other irrespective of the current directory.

Note that you must run `InitPath.m` each time you restart Matlab. To avoid this step, you may add all SPlaC folders to your Matlab path permanently or edit the `startup.m` file to do that (check Matlab help for details).

1.4 Disclaimer

These codes have been developed and tested with Matlab 7.5.0 (R2007b) on a PC running Microsoft Windows XP Pro SP2. Slight changes may be necessary to run them on older (or newer!) version of Matlab.

Although every efforts have been made to get rid of bugs (programming bugs, or even incorrect physical formulae), some may (must, we could say) still be present. We hope the users will help us identify them and we will try to update the codes when necessary.

Note also in this context that these codes do not implement a strict error checking of the user inputs, i.e. following the famous English proverb "garbage in, garbage out" (also known as "GIGO" in computer science), if the parameters are incorrect during a function call, errors will occur.

The authors do not accept any responsibility for improper use of the codes, accidental errors that might still be present in them, or improper interpretation of their limitations and/or results derived therefrom. It is the responsibility of the user to check the validity of the inputs/outputs, their physical interpretation, and their suitability for his/her specific problem.

1.5 Feedback

We would like to hear from the users of these codes to improve them. This includes simple issues of layouts and organization of the information or plain errors. Please feel free to send us any feedback (good or bad), bug reports, questions, comments, or suggestions to eric.leru@vuw.ac.nz and/or pablo.etchegoin@vuw.ac.nz.

1.6 Getting started

The easiest way (perhaps) to start realizing the extent of the codes, and what can be done with them, is to try to reproduce the figures from the book in the directory "BookFigures". From this latter directory, each individual m-file (a MatLab function) reproduces a specific figure shown in the book and its accompanying calculations.

The reason for trying to gain familiarity with the codes by reproducing figures is twofold. First of all, the codes run by themselves and, therefore, the inexperienced user is spared the work of having to decide which parameters he/she needs to start the calculation. In addition, the figures and the results can (and should) be used in conjunction with the book itself. In that manner, a lengthy explanation of the physical meaning of the calculation (which would be impossible to include in the code itself) is readily available.

Moreover, the reader can also adapt the figure to his/her specific needs, or "play with the parameters" to get a deeper understanding.

Once the user is familiar with the meaning of the calculation, the codes producing the figures can be inspected. The most important programming lines are accompanied by a suitable comment or remark. We believe any user with a minimum of experience in MatLab can easily follow the logic of the calculation. The main exception to this are the lines of code used for plotting and visualization, which are not explained in as much detail, but can be understood using the Matlab help. For example, the MatLab script `MakeFig6_11.m`, to reproduce Figure 11 of Chapter 6, contains under close inspection a call to the routine: `PweSolveSingleSphere`, which solves the Mie scattering problem for a single sphere. This code therefore provides a hands-on example of how to call this routine. In this way, a potential user can learn without much effort and through examples how the different functions can be pieced together to produce a specific outcome. This can then be potentially be used for different needs or applications beyond the ones shown in the book.

In the case of the codes for Mie theory, two functions, `PweFullMonty` and `DipFullMonty` are also provided to give an illustration of the possibilities of the codes by producing a number of representative figures. These are a good starting point to browse and understand the codes. These can be called simply by running the scripts `PweFullMonty` for Mie theory for plane wave excitation, and `DipFullMonty` for Mie theory for dipole emission.

Finally, the codes can be browsed in HTML format in the SPlaC online help. Thanks to the cross-linking of all functions and scripts, this provides a relatively easy and efficient way to navigate the codes. The SPlaC online help can be found at:

http://www.victoria.ac.nz/raman/book/codes/helpstart.html

from where it can also be downloaded as a zip file for local browsing.

2 A few Matlab technicalities, tips, and tricks

We discuss here a few specific issues related to Matlab programming. Users are encouraged to use the Matlab help for further details and SPlaC handling of potential numerical problems.

2.1 Vectors and matrices

2.1.1 General remarks

Matlab has been designed to work with arrays, typically vectors (row or column) and matrices. Organizing data into arrays and manipulating these arrays rather than their elements individually result in a much more concise

code and much faster execution times. We have therefore tried to design the codes with this in mind. This however requires a bit of "getting used to".

To help with the readability of the codes, the dimension of every variable is specified in the codes as "inline comments". For example, [1 x N] means a row vector with N elements (1 row, N columns). [L x 1] denotes a column vector with L elements (L rows, 1 column). [L x N] is a matrix with L rows and N columns. Many of the codes compute quantities that are wavelength-dependent. As a rule of thumb, the wavelengths are always specified in rows, i.e. variable `lambda` is a column vector [L x 1].

Note also that the codes make use of *structures* to group variables and pass or return them between functions. Check Matlab help for more information on structures.

2.1.2 Carrying sums using matrix operations

Let us consider a sum:

$$S(\lambda) = \sum_{n=1}^{N} f_n(\lambda). \tag{1}$$

This sum needs to be evaluated for L values of λ. One then uses the matrix F [L x N] containing $f_n(\lambda)$, and the command: `S=sum(F,2)`. This results in a column vector S [L x 1] with the desired sums.

If we now consider a sum:

$$S(\lambda) = \sum_{n=1}^{N} a_n f_n(\lambda), \tag{2}$$

this can be concisely (and rapidly) carried out as a matrix-vector product. Defining A as the row vector [1 x N] with elements a_n, we then simply have: `S=F*transpose(A)` resulting in a column vector S [L x 1] with the desired sums. Note that `*` denotes here the matrix product (as opposed to `.*` denoting the element-by-element product on two arrays of same dimension.

Similar considerations can be extended to the approximation of integrals as sums (using the simplest Simpson's approximation). If we have an integral:

$$I = \int_{\theta_1}^{\theta_2} f(\lambda, \theta) d\theta, \tag{3}$$

we define T as a row vector [1 x T] of θ's evenly spaced by a step `dtheta`, and F the matrix [L x T] containing $f(\lambda, \theta)$. We then have `I=sum(F,2) * dtheta`. For an integral:

$$J = \int_{\theta_1}^{\theta_2} f(\lambda, \theta) g(\theta) d\theta, \tag{4}$$

we define G a row vector [1 x T] with $g(\theta)$ and we have J=F*transpose(G) * dtheta. This is implemented for example as follows:

```
nNbTheta = 361; % large number of theta for small steps
dtheta = (theta2-theta1)/(nNbTheta-1); % step
theta = linspace(theta1,theta2,nNbTheta); % row [1 x T]
% assume F is [L x T] and G is [1 x T]
I = dtheta * sum(F,2);
J = dtheta * F * transpose(G);
```

2.2 Spherical Bessel functions and numerical issues

2.2.1 Computation of spherical Bessel functions

Although the spherical Bessel functions are not directly implemented in Matlab, it is straightforward to compute them from the standard Bessel functions, for which full support is provided in Matlab in the form of the functions `besselj` (for J_n), `bessely` (for Y_n), and `besselh` (for $H_n^{(1)} = J_n + iY_n$). As briefly mentioned in Sec. H.6.2, the use of `besselh` may lead to loss of precision. This is particularly acute when real and imaginary part differ by large orders of magnitude. To illustrate this, let us consider $H_{15}^{(1)}(1)$, one can check that:

```
besselh(15,1) = 1.1862e-001 -9.2570e+014i
besselj(15,1) = 2.2975e-017
bessely(15,1) = -9.2570e+014
```

Although the `besselh` result is correct when viewed as a complex number within double floating-point precision (i.e. $\approx$ 1e-16 relative error), it is obvious that the operation `real(besselh)`, which should be equivalent to `besselj` gives a wrong result by many order of magnitudes, because of loss of precision/term cancellation.

To avoid this issue, one may use `besselj + i*bessely` instead of besselh as suggested in Sec. H.6.2. This is satisfactory in most cases, except when loss of precision/term cancellation occurs when carrying the sum `besselj + i*bessely`. Unfortunately, this occurrence is also possible, as in the following example based on the calculation of $H_1^{(1)}(23i)$:

```
besselh(1,23i)                  = -1.7347e-011 - 1.0622e-027i
besselj(1,23i)                  = 4.8816e-008 + 7.9722e+008i
bessely(1,23i)                  =  -7.9722e+008 +4.8833e-008i
besselj(1,23i)+i*bessely(1,23i) = -1.7347e-011 -1.1921e-007i
```

This time, the imaginary part in the final expression is incorrect because of a cancellation problem in the sum.

Therefore, there is unfortunately, to our knowledge, no easy magic solution to this problem. In fact such cancellation problems may also arise in the evaluation of most of the sums that are computed within Mie theory. It is intrinsic to numerical problems involving numbers of widely different

magnitudes. Fortunately, such cancellations problem have in many cases no significant consequences on the final results, i.e. they occur in terms whose magnitude is anyway negligible compared to others and therefore do not affect the final results. In other cases, they may be related to real physical aspects (for example, the evaluated formula could have been simplified analytically). Despite this, it is necessary to identify these problems during the computation, and let the user investigate further whether or not they are an issue.

2.2.2 Check for numerical problems in the Mie theory codes of SPlaC

Perhaps, the better compromise is therefore to use `besselh`, since its result is correct within double floating-point precision in the absence of further manipulations. The only solution is then to check for potential term cancellation problems in *every* sums that are computed. This is carried out in the codes by carrying the sums by calling the functions `GenCheckSum2Mat`, `GenCheckSumMatVec`, and `GenCheckSumReal`. These functions (see SPlaC help for details) return the result of the sum, and in the process check for any cancellation issue by calling the auxiliary function `GenCheckSumNumPb`. For example, after a sum of the type $S = \sum a_n$, one must call the function `GenCheckSumNumPb(S,max(abs(an)))` (the arguments may be vectors or matrices of the same size). It will check that:
`abs(S)>1e-12*abs(max(abs(an)))`, which typically ensures that any term cancellations could be handled within the double floating-point precision. If not, a warning is issued and it is up to the user to investigate whether it is a real problem or not. Note that such checks must also be done when taking the real and imaginary parts of complex numbers (these operations are equivalent to sums of the number and its complex conjugate) for example using `GenCheckSumReal`. For sums carried out as the product of a matrix and vector as described earlier, one can call the function `GenCheckSumMatVec` to both do the sum and check for numerical problems.

In practice, it is better to keep these error-checking functions, but they do slow down *significantly* the calculations. For more intensive computations, and once such errors have been ruled out on representative examples, it is possible to speed up the computations by temporarily turning off the error checking routines by defining the global variable as follows:

```
global noCheckSum;
noCheckSum=true;
```

To reactivate the error checking, either define `noCheckSum=false`, or simply use `clear all` or `clear global noCheckSum` to clear the global variable.

Finally, it is worth mentioning a common situation where loss-of-precision warnings will always occur: that of non-absorbing spheres for the calcula-

tion of absorption or non-radiative properties such $Q_{\rm ext}$, $Q_{\rm abs}$, $M_{\rm Tot}$, or $M_{\rm NR}$. These are not an issue since these quantities are easily derived for non-absorbing materials as: $Q_{\rm ext} = Q_{\rm Sca}$, $Q_{\rm abs} = 0$, $M_{\rm Tot} = M_{\rm Rad}$, and $M_{\rm NR} = 0$.

3 Mie theory codes: plane wave excitation (PWE)

Appendix H of the book provides the theoretical basis for our Mie theory implementation. Substantial simplifications can be obtained for the special cases of plane wave excitation (PWE) and dipolar emission. Some of these were discussed in App. H and Ref. [1] provides a detailed description of the PWE case. This section and the next one summarize the most important formulae that were not included in App. H but are used to actually implement numerically the Mie calculations.

Note that the function `PweFullMonty` illustrates many of the possibilities of these codes for plane wave excitation. The script `PweScriptFullMonty` can be used to define the relevant parameters and call the function `PweFullMonty`. It will produce six examples of figures summarizing the results of the Mie computations.

3.1 Angle-dependent functions for PWE

For plane wave excitation, we only need $P_n^m(\cos\theta)$ for $|m| = 1$. We could use the Matlab function `legendre` to compute all P_n^m for $m = 0..n$ but this would not be very efficient when only $m = 1$ is needed. It is therefore better to compute the P_n^1 directly for n =1..nNMax using a recurrence relation. We here use the ones provided in Ref. [1]. To this end, we define as in Ref. [1]:

$$\pi_n = -\frac{P_n^1(\cos\theta)}{\sin\theta} \quad \text{and} \quad \tau_n = -\frac{dP_n^1(\cos\theta)}{d\theta}. \tag{5}$$

Note that Ref. [1] does not use the Condon-Shortley phase in their definition of $P_n^m(\cos\theta)$, i.e. there is a factor $(-1)^m$ difference with our definition (see Eq. H.19 of the book). The negative signs in the definition above (not present in Ref. [1]) arise because of this different convention, but the function π_n and τ_n are the same as those used in Ref. [1].

We then have the recurrence relations [1]:

$$\pi_n = \frac{2n-1}{n-1}\cos\theta\pi_{n-1} - \frac{n}{n-1}\pi_{n-2}, \tag{6}$$

$$\tau_n = n\cos\theta\pi_n - (n+1)\pi_{n-1}, \tag{7}$$

with the initial conditions:

$$\pi_0 = 0 \quad \text{and} \quad \pi_1 = 1. \tag{8}$$

Note that these relations are only valid for $0 \leq \theta \leq \pi$ (always the case here).

The function `PwePinTaun(nNmax,theta)` can be used to compute π_n and τ_n for several angle (`theta` given as a column vector). As an example, the script `PweScriptPolarPinTaun` reproduces the polar plots of π_n and τ_n shown in Ref. [1]. Also, the script `SymPinTaun` can be used to obtain the symbolic expressions of these two functions for the first few n's.

The auxiliary angle functions $T^{(i)}_{n,m}(\theta)$ (Eq. H.36) can then be deduced easily for $|m| = 1$ from these two functions:

$$T^1_{n,1}(\theta) = T^1_{n,-1}(\theta) = -\mu_n \pi_n(\theta), \tag{9}$$

$$T^2_{n,1}(\theta) = -T^2_{n,-1}(\theta) = -\mu_n n(n+1) \sin\theta \, \pi_n(\theta), \tag{10}$$

$$T^3_{n,1}(\theta) = -T^3_{n,-1}(\theta) = -\mu_n \tau_n(\theta), \tag{11}$$

where

$$\mu_n = \sqrt{\frac{2n+1}{4\pi}} \frac{1}{n(n+1)}. \tag{12}$$

Note also that for $\theta = 0$, we have:

$$\pi_n(0) = \tau_n(0) = \frac{n(n+1)}{2}. \tag{13}$$

3.2 Radial arguments of VSHs

3.2.1 Computation

The Riccati-Bessel functions and their derivatives can be computed using functions `GenRBall`, `GenRBpsi2`, and `GenRBxi2` (see SPlaC help for details).

The radial (r-dependent) arguments of the VSHs can be computed from the three auxiliary functions Z^0_n, Z^1_n, and Z^2_n, defined in Eq. H.35 (repeated below). Rather than using the functions written to calculate the Riccati-Bessel functions, it is faster to write a dedicated function for this task: `GenZnAll`. One then needs to compute once the spherical Bessel function $z_n(\rho)$ ($j_n(\rho)$ or $h^{(1)}_n(\rho)$), from which we deduce:

$$Z^0_n(\rho) = z_n(\rho), \tag{14}$$

$$Z^1_n(\rho) = z_n(\rho)/\rho, \tag{15}$$

$$Z^2_n(\rho) = [\rho z_n(\rho)]'/\rho = Z^0_{n-1}(\rho) - n Z^1_n(\rho), \tag{16}$$

where we have used $z'_n(\rho) = z_{n-1}(\rho) - (n+1) z_n(\rho)/\rho$.

The script `SymZnAll` provides analytical expressions for these functions for small n, along with Taylor expansions for small ρ.

3.2.2 Values at origin

Of interest is the behavior for $\rho = 0$ (only when the regular function j_n is used). All $Z_n^i(0)$ are then zero for $n \geq 1$, except:

$$Z_{n=1}^1(0) = \frac{1}{3} \quad \text{and} \quad Z_{n=1}^2(0) = \frac{2}{3}. \tag{17}$$

In terms of VSH, we then have $\mathbf{M}_{n,m}^{(1)}(0) = \mathbf{0}$ for all $n \geq 1$, $\mathbf{N}_{n,m}^{(1)}(0) = \mathbf{0}$ for all $n \geq 2$,

$$\mathbf{N}_{1,1}(0) - \mathbf{N}_{1,-1}(0) = -\frac{1}{\sqrt{3\pi}}\mathbf{e}_x, \tag{18}$$

and

$$\mathbf{N}_{1,0}(0) = \frac{1}{\sqrt{6\pi}}\mathbf{e}_z. \tag{19}$$

3.2.3 Far-field asymptotic expansions

Finally, when the Hankel function of the first kind, $h_n^{(1)}$ is used (for example for outgoing spherical waves), the asymptotic forms at infinity are to dominant order:

$$Z_n^0(\rho) \approx (-i)^{n+1}\frac{e^{i\rho}}{\rho}, \tag{20}$$

$$Z_n^1(\rho) \approx (-i)^{n+1}\frac{e^{i\rho}}{\rho^2}, \tag{21}$$

$$Z_n^2(\rho) \approx (-i)^{n}\frac{e^{i\rho}}{\rho}. \tag{22}$$

3.3 VSH expansions for PWE

3.3.1 Grouping of $|m| = 1$ terms

For plane wave excitation, we can choose without loss of generality a plane wave propagating along (Oz) and polarized along x. We then have only $|m| = 1$ terms (see Sec. H.4.1). Moreover, for the incident field $a_{n,1} = a_{n,-1}$ and this property is transfered to other VSH expansions for the same problem, for example for the scattered field $c_{n,1} = c_{n,-1}$. Similarly, $b_{n,1} = -b_{n,-1}$ and therefore $d_{n,1} = -d_{n,-1}$.

Starting from Eq. (H.33) and (H.34), the VSHs sums can therefore be grouped and expressed in terms of the angle-dependent functions π_n and τ_n using:

$$\begin{cases} \mathbf{M}_{n,1}(k,\mathbf{r}) \cdot \mathbf{e}_r = \mathbf{M}_{n,-1}(k,\mathbf{r}) \cdot \mathbf{e}_r &= 0 \\ (\mathbf{M}_{n,1}(k,\mathbf{r}) + \mathbf{M}_{n,-1}(k,\mathbf{r})) \cdot \mathbf{e}_\theta &= -2iZ_n^0(kr)\mu_n\pi_n(\theta)\cos\phi \\ (\mathbf{M}_{n,1}(k,\mathbf{r}) + \mathbf{M}_{n,-1}(k,\mathbf{r})) \cdot \mathbf{e}_\phi &= 2iZ_n^0(kr)\mu_n\tau_n(\theta)\sin\phi \end{cases} \tag{23}$$

$$\left\{\begin{array}{lcl}(\mathbf{N}_{n,1}(k,\mathbf{r})-\mathbf{N}_{n,-1}(k,\mathbf{r}))\cdot\mathbf{e}_r &=& -2Z_n^1(kr)\mu_n n(n+1)\sin\theta\pi_n(\theta)\cos\phi\\ (\mathbf{N}_{n,1}(k,\mathbf{r})-\mathbf{N}_{n,-1}(k,\mathbf{r}))\cdot\mathbf{e}_\theta &=& -2Z_n^2(kr)\mu_n\tau_n(\theta)\cos\phi\\ (\mathbf{N}_{n,1}(k,\mathbf{r})-\mathbf{N}_{n,-1}(k,\mathbf{r}))\cdot\mathbf{e}_\phi &=& 2Z_n^2(kr)\mu_n\pi_n(\theta)\sin\phi\end{array}\right. \tag{24}$$

We now consider a general expansion of the field in VSHs as:

$$\mathbf{E}(\mathbf{r}) = E_0\sum_{n,m} c_{n,m}\mathbf{M}_{n,m}^{(i)}(k_M,\mathbf{r}) + d_{n,m}\mathbf{N}_{n,m}^{(i)}(k_M,\mathbf{r}). \tag{25}$$

For PWE, i.e. $|m|=1$ only and $c_{n,1}=c_{n,-1}$ and $d_{n,1}=-d_{n,-1}$, we then We then have:

$$\begin{aligned}\mathbf{E}(\mathbf{r}) &= E_0\sum_n c_{n,1}\left(\mathbf{M}_{n,1}^{(3)}(k_M,\mathbf{r})+\mathbf{M}_{n,-1}^{(3)}(k_M,\mathbf{r})\right)\\ &\quad +d_{n,1}\left(\mathbf{N}_{n,1}^{(3)}(k_M,\mathbf{r})-\mathbf{N}_{n,-1}^{(3)}(k_M,\mathbf{r})\right).\end{aligned} \tag{26}$$

The three components of the field can then be conveniently rewritten as:

$$\mathbf{E}\cdot\mathbf{e}_r = -2\sin\theta\cos\phi E_0\sum_n d_{n,1}Z_n^1(kr)\mu_n n(n+1)\pi_n(\theta), \tag{27}$$

$$\mathbf{E}\cdot\mathbf{e}_\theta = -2E_0\cos\phi\sum_n \mu_n\left(ic_{n,1}Z_n^0(kr)\pi_n(\theta)+d_{n,1}Z_n^2(kr)\tau_n(\theta)\right), \tag{28}$$

$$\mathbf{E}\cdot\mathbf{e}_\phi = 2E_0\sin\phi\sum_n \mu_n\left(ic_{n,1}Z_n^0(kr)\tau_n(\theta)+d_{n,1}Z_n^2(kr)\pi_n(\theta)\right). \tag{29}$$

We are therefore left with sums over one index only (n).

3.3.2 Representation of the field in our PWE implementation

Moreover, it is convenient to separate from these expressions the $\phi-$dependence, which is very simple for PWE. This can be expressed in a single expression for $\mathbf{E}$ as:

$$\mathbf{E}/E_0 = \cos\phi E_{cr}\mathbf{e}_r + \cos\phi E_{ct}\mathbf{e}_\theta + \sin\phi E_{sf}\mathbf{e}_\phi, \tag{30}$$

where E_{cr}, E_{ct}, and E_{sf} do not depend on ϕ and can be obtained simply from the above sums.

To fully characterize the field numerically, it is better to compute E_{cr}, E_{ct}, and E_{sf} for various r, θ, and possibly λ, and the ϕ-dependence (if needed) then results from the above expression. This is implemented in SPlaC in two functions:

- `PweEgenThetaAllPhi:`
  which can be used to calculate the $\lambda-$ and $\theta-$ dependence of $\mathbf{E}$ at a fixed distance (r_0) from the origin. It returns E_{cr}, E_{ct}, E_{sf} as [L x T] matrices.

- `PweEgenRThetaAllPhi:`
  which can be used to calculate the $r-$ and $\theta-$ dependence of $\mathbf{E}$ at a *fixed wavelength* λ_0. It returns E_{cr}, E_{ct}, E_{sf} as [R x T] matrices.

3.3.3 Equivalence with the treatment of Ref. [1]

In Ref. [1], this grouping of the VSHs for $m = \pm 1$ for PWE was anticipated and more adapted versions of the VSHs were defined (these definitions are however less common for some extensions of Mie theory). In Ref. [1], the definitions of the expansion coefficients and susceptibilities also differ from our treatment. The final results, i.e. the value of the fields, should however be the same for both methods.

Using the expansion of the incident plane wave and considering a single sphere, we have for the scattered field: $c_{n,1} = \Gamma_n K_n$ and $d_{n,1} = \Delta_n K_n$ (see Eq. H.74 and H.75). The scattered field therefore has the following components (in the spherical coordinate frame):

$$\mathbf{E}_{\mathrm{Sca}} \cdot \mathbf{e}_r = -2E_0 \sin\theta \cos\phi \sum_n \mu_n K_n \Delta_n Z_n^1(kr) n(n+1) \pi_n(\theta), \tag{31}$$

$$\mathbf{E}_{\mathrm{Sca}} \cdot \mathbf{e}_\theta = -2E_0 \cos\phi \sum_n \mu_n K_n \left(i\Gamma_n Z_n^0(kr) \pi_n(\theta) + \Delta_n Z_n^2(kr) \tau_n(\theta) \right), \tag{32}$$

$$\mathbf{E}_{\mathrm{Sca}} \cdot \mathbf{e}_\phi = 2E_0 \sin\phi \sum_n \mu_n K_n \left(i\Gamma_n Z_n^0(kr) \tau_n(\theta) + \Delta_n Z_n^2(kr) \pi_n(\theta) \right). \tag{33}$$

This is (fortunately) consistent with the expressions of Ref. [1] (Eqs. 4.45 and 4.50 in Ref. [1]) thanks to the following correspondences (B. indicates the notations of Bohren and Huffman in Ref. [1]):

$$\begin{aligned} a_n(\mathrm{B.}) &\equiv -\Delta_n \\ b_n(\mathrm{B.}) &\equiv -\Gamma_n \\ E_n(\mathrm{B.}) &\equiv -2i\mu_n K_n E_0 \\ i a_n E_n(\mathrm{B.}) &\equiv -2\mu_n K_n E_0 \Delta_n \\ b_n E_n(\mathrm{B.}) &\equiv 2i\mu_n K_n E_0 \Gamma_n \end{aligned} \tag{34}$$

The equivalence can also be viewed directly from the VSHs expressions using the following correspondences (the signs arise partly from the differ-

ence in the inclusion of the Condon-Shortley phase between the two treatments):

$$\begin{aligned} 2\mu_{n,m}\mathbf{M}_{enm}(\text{B.}) &\equiv (-1)^m\mathbf{M}_{n,m} + \mathbf{M}_{n,-m}, \\ 2i\mu_{n,m}\mathbf{M}_{onm}(\text{B.}) &\equiv (-1)^m\mathbf{M}_{n,m} - \mathbf{M}_{n,-m}, \end{aligned} \tag{35}$$

where

$$\mu_{n,m} = \sqrt{\frac{2n+1}{4\pi n(n+1)}\frac{(n-m)!}{(n+m)!}}. \tag{36}$$

The same expressions apply for $\mathbf{N}$.

3.4 Enhancement factors for PWE

3.4.1 Local field intensity EFs

The field at any point can always be written in the form:

$$\mathbf{E}(r,\theta,\phi) = \cos\phi E_{cr}(r,\theta)\mathbf{e}_r + \cos\phi E_{ct}(r,\theta)\mathbf{e}_\theta + \sin\phi E_{sf}(r,\theta)\mathbf{e}_\phi. \tag{37}$$

It is therefore easy to derive the parallel and perpendicular local field intensity EFs (LFIEF, see Eq. 5.14 and the following discussion on p. 275):

$$M^{\perp}_{\text{Loc}}(r,\theta,\phi) = |E_{cr}(r,\theta)|^2\cos^2\phi, \tag{38}$$

$$M^{//}_{\text{Loc}}(r,\theta,\phi) = |E_{ct}|^2\cos^2\phi + |E_{sf}|^2\sin^2\phi. \tag{39}$$

These expressions are convenient to compute surface averages, since the averaging corresponding to the ϕ-dependence can be easily done analytically and only the θ-averaging is left to compute numerically. For example, the surface averages on a spherical surface ($r = r_0$) can be obtained as:

$$\langle M^{\perp}_{\text{Loc}}\rangle(r=r_0) = \frac{1}{2}\langle|E_{cr}(r_0,\theta)|^2\rangle, \tag{40}$$

$$\langle M^{//}_{\text{Loc}}\rangle(r=r_0) = \frac{1}{2}\langle|E_{ct}(r_0,\theta)|^2 + |E_{sf}(r_0,\theta)|^2,\rangle \tag{41}$$

and the total average LFIEF is simply the sum:

$$\langle M_{\text{Loc}}\rangle(r=r_0) = \frac{1}{2}\langle|E_{cr}(r_0,\theta)|^2 + |E_{ct}(r_0,\theta)|^2 + |E_{sf}(r_0,\theta)|^2\rangle. \tag{42}$$

These averages are computed in the function `PweEFaverages`, which is called by `PweEsurf` and `PweEmap`.

3.4.2 SERS EFs

The approximate SERS EF (Eq. 5.15) is:

$$F^0_{E4}(r,\theta,\phi) = \left[\left(|E_{cr}(r,\theta)|^2 + |E_{ct}(r,\theta)|^2\right)\cos^2\phi + |E_{sf}(r,\theta)|^2\sin^2\phi\right]^2. \tag{43}$$

From this expression, the surface average can then be shown to be:

$$\begin{aligned}\langle F^0_{E4}\rangle(r=r_0) &= \frac{1}{8}\langle 3\left(|E_{cr}|^2 + |E_{ct}|^2\right)^2 &(44)\\ &\quad +3|E_{sf}|^4 + 2\left(|E_{cr}|^2 + |E_{ct}|^2\right)|E_{sf}|^2\rangle. &(45)\end{aligned}$$

One may also separate the perpendicular and parallel contributions to F^0_{E4}. The corresponding averages are then:

$$\langle F^{0-\perp}_{E4}\rangle(r=r_0) = \frac{3}{8}\langle |E_{cr}|^4\rangle. \tag{46}$$

and

$$\langle F^{0-//}_{E4}\rangle(r=r_0) = \frac{1}{8}\langle 3\left(|E_{ct}|^4 + |E_{sf}|^4\right) + +2|E_{ct}|^2|E_{sf}|^2\rangle. \tag{47}$$

These are also computed in the function `PweEFaverages`, which is called by `PweEsurf` and `PweEmap`.

3.5 Radiation (scattering) profile for PWE

The scattered field in the far-field can be obtained from the asymptotic expressions of the VSHs in Eqs. (H.39) and (H.40). For PWE, we can also directly use the asymptotic form of the Z^i_n given earlier. We then have:

$$\mathbf{E}\cdot\mathbf{e}_r \approx 0 + O(\frac{1}{r}), \tag{48}$$

$$\mathbf{E}\cdot\mathbf{e}_\theta \approx -2E_0\cos\phi\frac{e^{ikr}}{kr}\sum_n \mu_n(-i)^n\left(c_{n,1}\pi_n(\theta) + d_{n,1}\tau_n(\theta)\right), \tag{49}$$

$$\mathbf{E}\cdot\mathbf{e}_\phi \approx 2E_0\sin\phi\frac{e^{ikr}}{kr}\sum_n \mu_n(-i)^n\left(c_{n,1}\tau_n(\theta) + d_{n,1}\pi_n(\theta)\right). \tag{50}$$

Following Ref. [1] (Sec. 4.4.4 of Ref. [1]), we can define the functions of θ for the scattering amplitude matrix:

$$S_1(\theta) = -2\sum_n \mu_n(-i)^{n+1}\left(c_{n,1}\tau_n(\theta) + d_{n,1}\pi_n(\theta)\right), \tag{51}$$

and

$$S_2(\theta) = -2\sum_n \mu_n(-i)^{n+1}\left(c_{n,1}\pi_n(\theta) + d_{n,1}\tau_n(\theta)\right). \tag{52}$$

We then have:

$$\mathbf{E}\cdot\mathbf{e}_\theta \approx E_0\cos\phi\frac{e^{ikr}}{-ikr}S_2(\theta), \tag{53}$$

$$\mathbf{E}\cdot\mathbf{e}_\phi \approx -E_0\sin\phi\frac{e^{ikr}}{-ikr}S_1(\theta). \tag{54}$$

S_1 and S_2 are scattering amplitudes from which all polarization properties of the scattered field in the far-field can be derived [1]. These can be plotted as normalized differential scattering cross-sections as (for S_1 for example):

$$\frac{dQ_{\rm Sca}^{S1}}{d\Omega} = \frac{d\sigma_{\rm Sca}}{d\Omega}/(\pi a^2) = \frac{|S_1(\theta)|^2}{\pi x^2}. \tag{55}$$

Note that

$$Q_{\rm Sca} = \pi\int_0^\pi\left[\frac{dQ_{\rm Sca}^{S1}}{d\Omega}+\frac{dQ_{\rm Sca}^{S2}}{d\Omega}\right]\sin\theta d\theta. \tag{56}$$

The function `PweFarField` can be used to compute S_1 and S_2, while the function `PwePlotEfarSca` produces the plots.

4 Mie theory codes: dipole emission

4.1 General remarks

As explained in Sec. H.5.1, the dipole is considered, without loss of generality, to be on the z axis at a distance d outside the sphere with $\mathbf{p} = p_x\mathbf{e}_x + p_z\mathbf{e}_z$ (note that $\mathbf{e}_x = \mathbf{e}_\theta$ on the z axis).

Most results can also be inferred from the solution for the two cases of a parallel dipole (along $\mathbf{e}_x$) and a perpendicular dipole (along $\mathbf{e}_z$). The incident field expansion can then be carried out as explained in Sec. H.5.1. For a parallel dipole, the only non-zero coefficients are then $a_{n,1} = a_{n,-1}$, $b_{n,1} = -b_{n,-1}$. This is similar to the situation for PWE and many of the functions written for PWE can therefore be used (they are located in the folder `Mie/PWE`). For a perpendicular dipole, the only non-zero coefficients are $a_{n,0}$. Specific functions have therefore been written for this particular case (they are located in the folder `Mie/DIP`).

As for the PWE case, the function `DipFullMonty` provides an illustration of the various possibilities of the codes and can be called by running the script `DipScriptFullMonty`.

4.2 Angle-dependent functions for dipole emission

For dipole emission, we need to introduce new angle-dependent functions for the VSHs with $m = 0$. These are defined as:

$$p_n(\theta) = P_n(\cos\theta) \quad \text{and} \quad t_n = \frac{dP_n(\cos\theta)}{d\theta}, \tag{57}$$

where P_n are the Legendre polynomials. We could use the Matlab function `legendre` to compute all P_n^m for $m = 0..n$ but this would not be very efficient when only $m = 0$ is needed. It is therefore better to compute p_n and t_n directly using the following recurrence relations:

$$p_n = \frac{2n-1}{n}\cos\theta p_{n-1} - \frac{n-1}{n}p_{n-2}, \tag{58}$$

$$t_n = \cos\theta t_{n-1} - n\sin\theta p_{n-1}, \tag{59}$$

with the initial conditions:

$$p_0 = 1, \quad p_1 = \cos\theta, \quad t_0 = 0, \quad t_1 = -\sin\theta. \tag{60}$$

The function `DipPinTaunPnTn(nNmax,theta)` can be used to compute π_n, τ_n (both defined for PWE and needed for a parallel dipole), and p_n and t_n (needed for $m = 0$) for a number of `theta` (as a column vector).

Also, the script `SymPnTn` can be used to obtain the symbolic expressions of p_n and t_n for the first few n's.

Finally, note that for $\theta = 0$, we have:

$$p_n(0) = 1 \quad \text{and} \quad t_n(0) = 0. \tag{61}$$

4.3 VSH expansions for dipole emission

4.3.1 General expression for $m = 0$

For a general dipole emission, we always take without loss of generality the dipole on the z-axis and aligned in the $(x\mathrm{O}z)$ plane. The coefficient of the expansions are then only non-zero for $|m| = 1$ (same as PWE) and for $m = 0$ for $\mathbf{N}_{n,0}$ VSHs only. The angle-dependent functions for PWE can be used for $|m| = 1$. We need to study those relevant to $\mathbf{N}_{n,0}$. We have:

$$\begin{cases} \mathbf{N}_{n,0}(k,\mathbf{r})\cdot\mathbf{e}_r &= n(n+1)\mu_{n,0}Z_n^1(kr)p_n(\theta) \\ \mathbf{N}_{n,0}(k,\mathbf{r})\cdot\mathbf{e}_\theta &= \mu_{n,0}Z_n^2(kr)t_n(\theta) \\ \mathbf{M}_{n,0}(k,\mathbf{r})\cdot\mathbf{e}_\phi &= 0 \end{cases} \tag{62}$$

where

$$\mu_{n,0} = \sqrt{\frac{2n+1}{4\pi n(n+1)}}, \tag{63}$$

and p_n and t_n have been defined in the previous section.

4.3.2 Representation of the field in our implementation of dipole emission

As for PWE, it is convenient to isolate from the field expressions the $\phi-$dependence, which is irrelevant for $m = 0$ and very simple for $|m| = 1$. The field expression for a general dipole $\mathbf{p} = p_x\mathbf{e}_x + p_z\mathbf{e}_z$ is determined by the three set of coefficients $c_{n,1}$, $d_{n,1}$, and $d_{n,0}$ and takes the form:

$$\mathbf{E}/E_{p0} = (p'_x \cos\phi E_{cr} + p'_z E_{m0r})\mathbf{e}_r + (p'_x \cos\phi E_{ct} + p'_z E_{m0t})\mathbf{e}_\theta + p'_x \sin\phi E_{sf}\mathbf{e}_\phi, \tag{64}$$

where $p = \sqrt{|p_x|^2 + |p_z|^2}$, $p'_x = p_x/p$, and $p'_z = p_z/p$. E_{cr}, E_{ct}, E_{sf}, E_{m0r}, and E_{m0t} do not depend on ϕ and can be expressed as:

$$E_{m0r} = \sum_n d_{n,0} n(n+1)\mu_{n,0} Z_n^1(kr) p_n(\theta), \tag{65}$$

$$E_{m0t} = \sum_n d_{n,0}\mu_{n,0} Z_n^2(kr) t_n(\theta), \tag{66}$$

while the expressions for E_{cr}, E_{ct}, and E_{sf} have already been given in the PWE section.

Moreover, to fully characterize the field numerically, it is better to compute E_{cr}, E_{ct}, E_{sf}, E_{m0r}, and E_{m0t} for various r, θ, and possibly λ. Results for any dipole orientation, including the ϕ-dependence then results from the above expression. This is implemented in the function `DipEgenThetaAllPhi`, which can be used to calculate the $\lambda-$ and $\theta-$ dependence of $\mathbf{E}$ at a fixed distance from the origin (r_0). It returns E_{cr}, E_{ct}, E_{sf}, E_{m0r}, and E_{m0t} as [L x T] matrices. This function is similar to its counterpart for PWE, `PweEgenThetaAllPhi`.

5 Mie theory codes: spherical multilayer systems

The Mie theory codes can also be used to compute the properties of a general multilayer system with spherical symmetry, for example, a nano-shell. The function `MulPweFullMonty` provides an illustration of this for plane wave excitation and can be called by running the script `MulPweScriptFullMonty`. Similarly, the function `MulDipFullMonty` provides an illustration for dipole emission and can be called by running the script `MulDipScriptFullMonty`. Apart from a few specific functions found in directory `Mie/MUL`, in particular `MulSuscepGDAB` for the Mie susceptibilities, the multilayer codes use many of the functions already written for plane wave excitation and dipole emission.

References

[1] C. F. Bohren and D. R. Huffman, *Absorption and Scattering of Light by Small Particles* (Wiley, New York, 1983).