Notes on DDA0.1

Augustin Muster

March 18, 2022

Contents

1 DDA method description

1.1 Lattice and polarisabilities

Let us consider a solid of volume V discretized in N dipoles on a cubic lattice (of lattice parameter d) with dielectric tensor $\varepsilon(\boldsymbol{r})$ in a homogoneous medium with dielectric constant $\varepsilon_h$such that

$$V = Nd^3. \tag{1}$$

Each dipole is associated with his position $\boldsymbol{r_n}$, $n = 1, ..., N$and his quaisitatic polarisability tensor defined by

$$\boldsymbol{\alpha}_{n0} = (\epsilon(\boldsymbol{r_n}) - \epsilon_h I)((\epsilon(\boldsymbol{r_n}) - \epsilon_h I) + \boldsymbol{L_n^{-1}}\epsilon_h)^{-1}\boldsymbol{L_n^{-1}}V_n, \tag{2}$$

where $\boldsymbol{I}$ ^is the identy matrix, $V_n = d^3$ is the volume of a discretization unit and $\boldsymbol{L_n}$is the depolarisation tensor defined as

$$[\boldsymbol{L}_n]_{ij} = \delta_{ij}\frac{1}{3} \tag{3}$$

for a cube. A radiative correction of the polarisability has to be taken into account. With this correction, the polarisability tensor $\boldsymbol{\alpha}_n$ now reads7

$$\boldsymbol{\alpha}_n = \left(\boldsymbol{\alpha}_{n0}^{-1} - i\frac{k^3}{6\pi}\right)^{-1}, \tag{4}$$

where k is the wavenumber of the incident wave.

1.2 Scattering

Let us consider that the solid is illuminated by an inciedent field $\boldsymbol{E}_0(\boldsymbol{r})$ the field $\boldsymbol{E}_{exc}(\boldsymbol{r}_n)$ extiting the nth dipole is equal to the incident field plus the contribution of the other dipoles, i.e.

$$\boldsymbol{E}_{exc}(\boldsymbol{r}_n) = \boldsymbol{E}_0(\boldsymbol{r}_n) + k^2 \sum_{m \neq n}^{N} \boldsymbol{G}(\boldsymbol{r}_n, \boldsymbol{r}_m)\boldsymbol{\alpha}_m \boldsymbol{E}_{exc}(\boldsymbol{r}_m), \tag{5}$$

where $\boldsymbol{G}(\boldsymbol{r}_n, \boldsymbol{r}_m)$ is the green tensor which reads

$$\boldsymbol{G}(\boldsymbol{r}_n, \boldsymbol{r}_m) = \frac{\exp(ikR)}{4\pi R}\left(\left(1 + \frac{i}{kR} - \frac{1}{k^2R^2}\right)\boldsymbol{I} - \left(1 + \frac{3i}{kR} - \frac{3}{k^2R^2}\right)\frac{\boldsymbol{R}\otimes\boldsymbol{R}}{R^2}\right), \tag{6}$$

with $\boldsymbol{R} = \boldsymbol{r}_n - \boldsymbol{r}_m$, $R = |\boldsymbol{R}|$. This equation in then just a big system of 3N linear equations that reads

$$\begin{bmatrix} \boldsymbol{I} & \dots & -k^2\boldsymbol{G}(\boldsymbol{r}_1, \boldsymbol{r}_N)\boldsymbol{\alpha} \\ \vdots & \ddots & \vdots \\ -k^2\boldsymbol{G}(\boldsymbol{r}_N, \boldsymbol{r}_1)1 & \dots & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{E}_{exc}(\boldsymbol{r}_1) \\ \vdots \\ \boldsymbol{E}_{exc}(\boldsymbol{r}_N) \end{bmatrix} = \begin{bmatrix} \boldsymbol{E}_0(\boldsymbol{r}_1) \\ \vdots \\ \boldsymbol{E}_0(\boldsymbol{r}_N) \end{bmatrix} \tag{7}$$

that can be solved in order to get the $\boldsymbol{E}_{exc}(\boldsymbol{r}_n)$. Using this, the polarizations of each dipole can be computed using $\boldsymbol{P}_n = \boldsymbol{\alpha}_n \boldsymbol{E}_{exc}(\boldsymbol{r}_n)$

1.3 Cross sections calculations

Then, the extinction, absorbtion and scattering cross section $\sigma_{ext}, \sigma_{abs}, \sigma_{sca}$ (that follows $\sigma_{ext} = \sigma_{abs} + \sigma_{sca}$) can be comuted using (considerning that the incident field is a plane wave $\boldsymbol{E}_0(\boldsymbol{r}) = \boldsymbol{E}_0 \exp(ikr)$))

$$\sigma_{ext} = \frac{k}{\epsilon_h |\boldsymbol{E}_o|^2} \sum_{n=1}^{N} Im(\boldsymbol{E}_0^*(\boldsymbol{r}) \cdot \boldsymbol{P}_n) \tag{8}$$

$$\sigma_{sca} = \frac{k^3}{{\epsilon_h}^2 |\boldsymbol{E}_o|^2} \sum_{n,m=1}^{N} \boldsymbol{P}_n^* \cdot Im(\boldsymbol{G}(\boldsymbol{r}_n, \boldsymbol{r}_m))\boldsymbol{P}_n \tag{9}$$

$$\sigma_{abs} = \frac{k}{{\epsilon_h}^2 |\boldsymbol{E}_o|^2} \sum_{n=1}^{N} Im(\boldsymbol{P}_n \cdot (\boldsymbol{\alpha}_{n0}\boldsymbol{P}_n)^*) \tag{10}$$

2 DDA0.1 documentation

The code is divided in several modules that can be easily inculded in other modules and easily extended. The core functionalities of the DDA are in impolemented in the **DDA.jl.** This code as well as all the others module providing side- but essential functionalities are describet in this section.

2.1 DDA.jl

DDA.jl contains the core functionalyty of the DDA, i.e. the functions to compute the green tensor and solve the linear system of equations. These functions are availeble either in a normal or "dimensionless" version. They are the following.

2.1.1 solve_DDA()

solve_DDA(knorm, r, alpha, input_field::Function ; output="polarisations", solver="LU", verbose=true)

This function generate the DDA equations systems (equation (7)) and solve it in order to output the polarisations $\boldsymbol{P}_n$or the inverse of the linear equation matrix A^{-1}.

Arguments

- **knorm:** norm of the k vector (real scalar).
- **r:** real array containing all the (3d) position vectors of the dipoles.
- **alpha:** complex array containing all the (3x3) polarisability tensor of each dipole.

- **input_field:** pointer to a function that will generate the input field this function haa.
- **output** (facultative): it can takes two values: "polaristions" (the function returns the $\boldsymbol{P}_n$) or "matrix" (the functiuons return the A^{-1}). By default set to "polarisations".
- **solver** (facultative): indication on which solver to use to solve the equations. There is yet only one possibility: "LU" (use the LU decomposition of LAPACK).
- **verbose** (facultative): whether to print everything or not. By default set to "true".

Outputs An array of size (N,3) with the polarisations $\boldsymbol{P}_n$ or the inverse of the (3N*3N) linear equation matrix A^{-1} (dependint on the **output** parameter).

2.1.2 green()

green(r1, r2, knorm)
This function computes the green tensor using equation (6).

Arguments

- **r1:** position of the first dipole (3d vector)
- **r2:** position of the first dipole (3d vector)
- **knorm:** norm of the k vector

Outputs The associated (3x3) green tensor.

2.1.3 solve_DDA_dl()

solve_DDA_dl(knorm, kr, k3alpha, input_field::Function ; output="polarisations", solver="LU", verbose=true)
This function generate the DDA equations systems (equation (7)) with dimensionless inputs and solve it in order to output the polarisations $\boldsymbol{P}_n$or the inverse of the linear equation matrix A^{-1}.

Arguments

- **knorm:** norm of the k vector (real scalar).
- **kr:** real array containing all the (3d) position vectors of the dipoles, multiplied with the norm of the k vector
- **k3alpha:** complex array containing all the (3x3) polarisability tensor of each dipole multiplied with norm of the k vector to the power 3.
- **input_field:** pointer to a function that will generate the input field this function haa.
- **output** (facultative): it can takes two values: "polaristions" (the function returns the $\boldsymbol{P}_n$) or "matrix" (the functiuons return the A^{-1}). By default set to "polarisations".
- **solver** (facultative): indication on which solver to use to solve the equations. There is yet only one possibility: "LU" (use the LU decomposition of LAPACK).
- **verbose** (facultative): whether to print everything or not. By default set to "true".

Outputs An array of size (N,3) with the polarisations $\boldsymbol{P}_n$ or the inverse of the (3N*3N) linear equation matrix A^{-1} (dependint on the **output** parameter).

2.1.4 green_dl()

green_dl(kr1, kr2, knorm)
This function computes the green tensor with dimensionless inputs using equation (6).

Arguments

- **kr1:** position of the first dipole times the norm of the wave vector (3d vector)
- **kr2:** position of the first dipole times the norm of the wave vector (3d vector)
- **knorm:** norm of the k vector (real scalar)

Outputs The associated (3x3) green tensor.

2.2 alpha.jl

The alpha.jl module is a side-functionality module providing functions to compute the polarisability tensors.

2.2.1 depolarisation_tensor()

depolarisation_tensor(lx,ly,lz,Vn)
This function computes the depolarisation tensor as in equation (3) but for an arbitrary parralelipipedic unit.

Arguments

- **lx, ly, lz**: dimensions of the parralelipipedic volume (scalars).
- **Vn:** volume of the parralelipipedic unit (scalar).

Outputs The (3x3) depolarisation tensor

2.2.2 alpha_0()

alpha_0(e,e_m,Ln,Vn)
This function compute the quasistatic polarisability tensor as in equation (2).

Arguments

- **e:** dielectric permetivitty of the medium (complex scalar).
- **e_m:** dielectric permittivity of the embeding medium (complex scalar).
- **Ln:** depolaristion tensor (3x3 matrix).
- **Vn:** volume of the particle (scalar)

Outputs The associated (3x3 complex) quasistatic polarisability tensor.

2.2.3 alpha_radiative()

alpha_radiative(a0,knorm)
This function compute the polarisability tensor with the radiative correction with equation (4), given a quasitatic polarisability tensor.

Arguments

- **a0:** a quasistatic polarisability tensor (3x3 complex)
- **knorm:** the norm of the k vector (scalar)

Outputs The (3x3 complex) polarisability tensor with radisative correction.

2.3 input_fields.jl

The input_fields.jl module is a side-functionality module providing functions to generate the input fields.

2.3.1 plane_wave()

plane_wave(knorm,r,khat=[0,0,1],e0=[1,0,0])
Function to evaluate a plane wave at a given position.

Arguments

- **knorm:** the norm of the k vector (scalar).
- **r:** position where to evaluate the plane wave (3d vector).
- **khat** (faculatitve): direction of the k vector (3d vector). The norm of this vector must be 1. By default set to the direction of the plane wave.
- **e0** (facultative): polarisation (3d complex vector). By default set to [1,0,0] (polarized on the x axis).

Outputs A 3d complex vector representing the plane wave vector at the given position.

2.3.2 plane_wave_dl()

plane_wave_dl(kr,khat=[0,0,1],e0=[1,0,0])
Function to evaluate a plane wave at a given position, but with dimensionless inputs.

Arguments

- **kr:** position where to evaluate the plane wave times the norm of the k vector (3d vector).
- **khat** (faculatitve): direction of the k vector (3d vector). The norm of this vector must be 1. By default set to the direction of the plane wave.
- **e0** (facultative): polarisation (3d complex vector). By default set to [1,0,0] (polarized on the x axis).

Outputs A 3d complex vector representing the plane wave vector at the given position.

2.4 processing.jl

2.4.1 compute_cross_sections()

compute_cross_sections(knorm,p,e_inc,alpha0,r;e0=[1,0,0],explicit_scattering=true,verbose=true)
Compute the extinction, absorbtion and scattering cross sections as done in equations (8), (9) and (10) (only for a plane wave illumination!).

Arguments

- **knorm:** the norm of the k vector (scalar).
- **p:** polarisations (array of 3d complex vectors).
- **alpha0** (facultative): quasistatic polarisability tensors (array of complex 3x3 matrices).
- **r:** real array containing all the (3d) position vectors of the dipoles.
- **e0** (facultative): polarisation (3d complex vector). By default set to [1,0,0] (polarized on the x axis).
- **explicit_scattering** (facultative): wether to compute the scattering cross section explicitely or not. Bay default set to true.
- **verbose** (facultative): whether to print everything or not. By default set to "true".

Outputs an array containing [lambda,Cext, Cabs,Csca]

3 Test and Example.

DDA computations have been made with a silicon sphere with radius of 230nm, with 13000 dipoles, $\lambda = 1000, ..., 2000$nm. The script used to run these computation is called **run_DDA.jl**

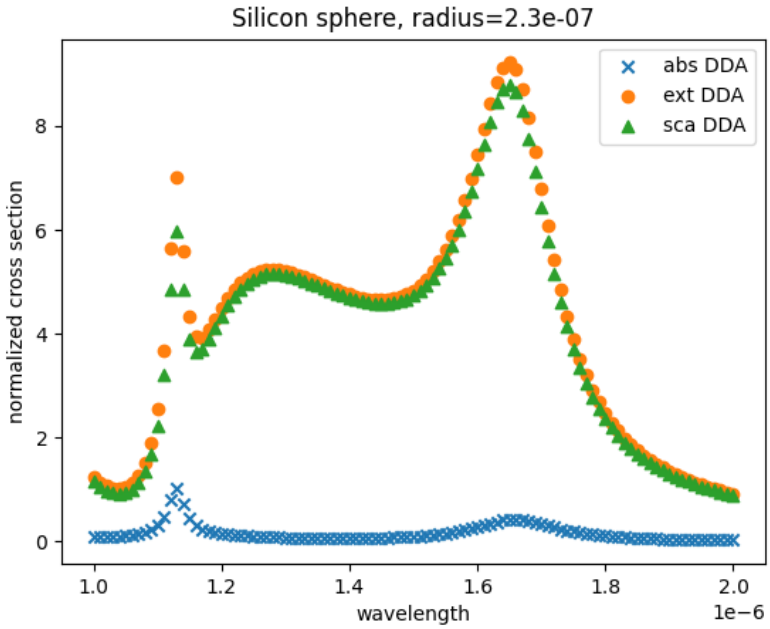

Figure 1: Extinction,scattering and absorbtion cross sections for a silicon sphere with radius of 230nm, computed with the DDA.jl code

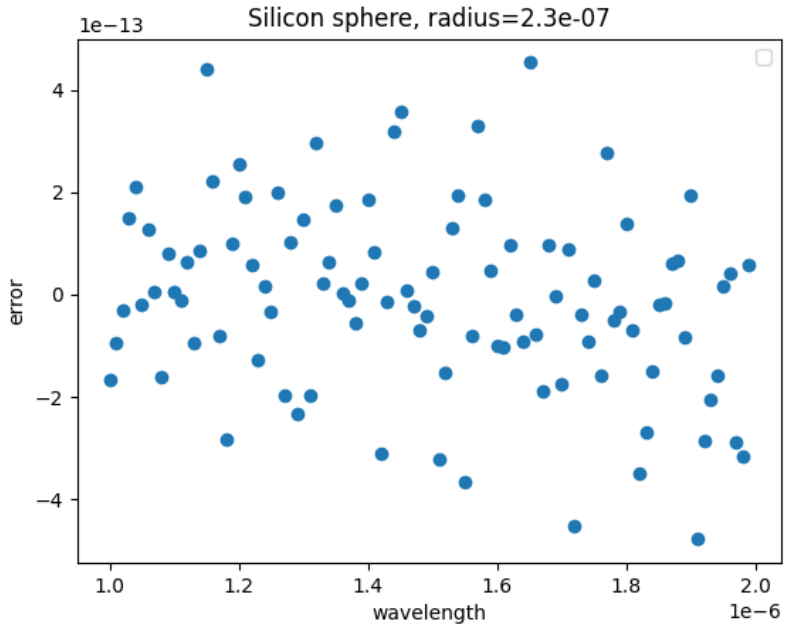

Figure 2: relative error of the extinction for the computation of figure 1.

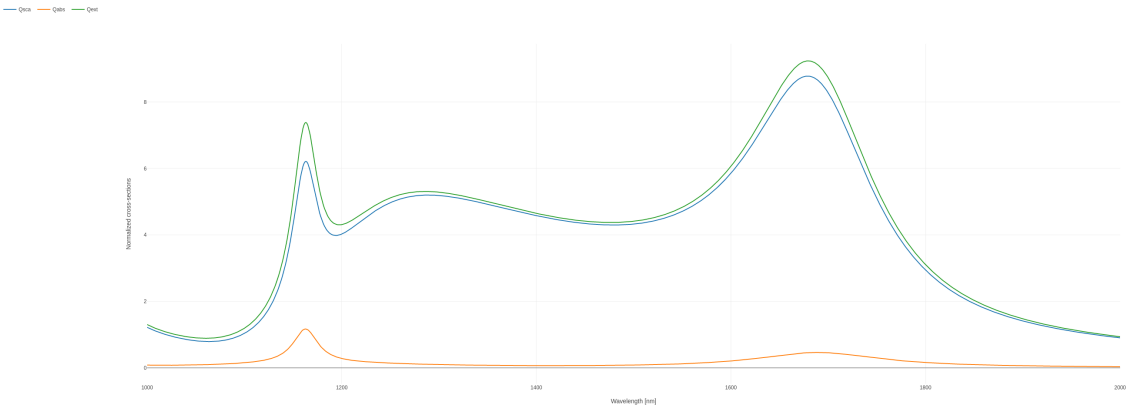

Figure 3: Extinction,scattering and absorbtion cross sections for a silicon sphere with radius of 230nm, computed with Mie theory