Computer Physics Communications 233 (2018) 167–192

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

pyGDM—A python toolkit for full-field electro-dynamical simulations and evolutionary optimization of nanostructures☆

Peter R. Wiecha

CEMES-CNRS, Université de Toulouse, CNRS, UPS, Toulouse, France

ARTICLE INFO

Article history:
Received 14 February 2018
Received in revised form 8 June 2018
Accepted 12 June 2018
Available online 10 July 2018

Keywords:
Electrodynamical simulations
Green dyadic method
Coupled dipoles approximation
Nano-optics
Photonic nanostructures
Nano plasmonics

ABSTRACT

pyGDM is a python toolkit for electro-dynamical simulations in nano-optics based on the Green Dyadic Method (GDM). In contrast to most other coupled-dipole codes, pyGDM uses a generalized propagator, which allows to cost-efficiently solve large monochromatic problems such as polarization-resolved calculations or raster-scan simulations with a focused beam or a quantum-emitter probe. A further peculiarity of this software is the possibility to very easily solve 3D problems including a dielectric or metallic substrate. Furthermore, pyGDM includes tools to easily derive several physical quantities such as far-field patterns, extinction and scattering cross-section, the electric and magnetic near-field in the vicinity of the structure, the decay rate of quantum emitters and the LDOS or the heat deposited inside a nanoparticle. Finally, pyGDM provides a toolkit for efficient evolutionary optimization of nanoparticle geometries in order to maximize (or minimize) optical properties such as a scattering at selected resonance wavelengths.

Program summary
Program Title: pyGDM
Program Files doi: http://dx.doi.org/10.17632/8wjcccv73j.1
Licensing provisions: GPLv3
Programming language: python, fortran
Nature of problem: Full-field electrodynamical simulations of photonic nanostructures. This includes problems like optical scattering, the calculation of the near-field distribution or the interaction of quantum emitters with nanostructures. The program includes a module for automated evolutionary optimization of nanostructure geometries with respect to a specific optical response.
Solution method: The optical response of photonic nanostructures is calculated using field susceptibilities ("Green Dyadic Method") via a volume discretization. The approach is formally very similar to the coupled dipole approximation.
Additional comments including restrictions and unusual features: Only 3D nanostructures. The volume discretization is limited to about 10000 meshpoints.

Full-field electro-dynamical simulations are used in nano-optics to predict the optical response of small (often sub-wavelength) particles by solving the Maxwell's equations [1]. Examples are either the scattering or the confinement of an external electro-magnetic field by dielectric [2] or metallic [3] nano-structures, the appearance of localized surface plasmons [4] or the interaction of nano-structures with quantum emitters placed in their vicinity [5]. Nano-optics governs manifold effects and applications. Examples are phase control or polarization conversion either at the single particle level [6,7] or from metasurfaces [8], shaping of the directionality of scattering [9], thermoplasmonic heat generation with sub-micrometer heat localization [10] or nonlinear nano-optics [11,12]. It is of great importance to be able to calculate optical effects occurring in sub-wavelength small structures in order to predict or to interpret experimental findings.

In this paper, we present the python toolkit "pyGDM" for full-field electro-dynamical simulations of nano-structures. Below we list the key-features and aims of pyGDM which we will explain in detail in the following.

- Easy to use. Easy to install: Fully relying on freely available open-source python libraries (numpy, scipy, matplotlib).
- Fast: Performance-critical parts are implemented in fortran and are parallelized with openmp. Efficient and parallelized

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

E-mail address: peter.wiecha@cemes.fr.

https://doi.org/10.1016/j.cpc.2018.06.017
0010-4655/

scipy libraries are used whenever possible. Spectra can be calculated very rapidly via an MPI-parallelized routine.
- Electro-dynamical simulations including a substrate.
- Different illumination sources such as plane wave, focused beam or dipolar emitter.
- Efficient calculation of large problems such as raster-scan simulations.
- Provide tools to rapidly post-process the simulations and derive physical quantities such as
  - optical near-field inside and around nanostructures.
  - extinction, absorption and scattering cross-sections.
  - polarization- and spatially resolved far-field scattering.
  - heat generation, temperature distribution around nano-objects.
  - photonic local density of states (LDOS).
  - modification of the decay-rate of dipolar emitters in the presence of a nanostructure.
- Evolutionary optimization of the nano-particle geometry with regard to specific optical properties.
- Easy to use visualization tools including animations of the electro-magnetic fields.

We will start with a brief introduction to the Green Dyadic Method (GDM), the numerical discretization scheme and the renormalization of the Green's dyad. We will also compare the GDM to other frequently used numerical techniques. In the second part, we will explain in more detail the main features and tools provided by pyGDM. We start by explaining the general structure of pyGDM and the ingredients to set up a simulation. Then we describe how the main simulation routines work. This part is followed by descriptions of the pyGDM-tools for simulating different optical effects, post-processing, data-analysis and visualization. Subsequently we will illustrate the capabilities of pyGDM by some example simulations and benchmarks. In particular, we will compare pyGDM-simulations to Mie theory. Finally, we will give an overview on the evolutionary optimization submodule of pyGDM, accompanied by several examples. In the Appendix we provide details concerning more technical tools and aspects of pyGDM as well as instructions for the compilation, installation and use of pyGDM.

1. The Green dyadic method

In the following we will give a brief introduction to the basic concepts of the Green dyadic method, implemented in pyGDM. Before we begin with this short overview, we want to note that the GDM is a frequency domain technique, solving Maxwell's equations for monochromatic fields (oscillating at fixed frequency ω).

Note. We use CGS (centimeter, gram, second) units in pyGDM which results in simpler terms for most of the equations. This is first of all helpful for the derivation of the main equations and has no impact on the simulation results. The post-processing routines return values that conform with SI units such as cross-sections (units of nm^2), powers in Watt or unit-less values (e.g. relative field intensities such as $|\mathbf{E}|^2/|\mathbf{E}_0|^2$).

1.1. From Maxwell's equations to Lippmann–Schwinger equation

All electromagnetic phenomena can be entirely described by the four Maxwell's equations, which in the frequency domain is written as follows (CGS units):

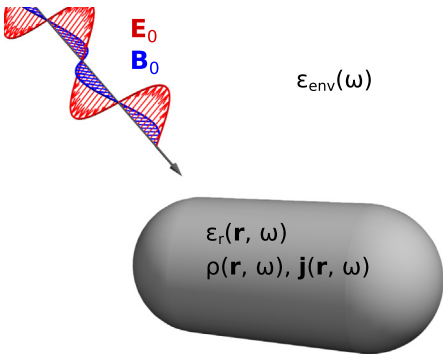

Fig. 1. Electromagnetic wave impinging on a nanostructure of arbitrary shape, placed in a homogeneous environment.

$$\nabla \cdot \mathbf{E}(\mathbf{r},\omega) = \frac{4\pi}{\epsilon_{\text{env}}}\rho(\mathbf{r},\omega) \tag{1a}$$

$$\nabla \times \mathbf{E}(\mathbf{r},\omega) = \mathrm{i}k_0\mathbf{B}(\mathbf{r},\omega) \tag{1b}$$

$$\nabla \cdot \mathbf{B}(\mathbf{r},\omega) = 0 \tag{1c}$$

$$\nabla \times \mathbf{B}(\mathbf{r},\omega) = -ik_0\epsilon_{\text{env}}\mathbf{E}(\mathbf{r},\omega) + \frac{4\pi}{c}\mathbf{j}(\mathbf{r},\omega) \tag{1d}$$

where the charge density ρ and the current density $\mathbf{j}$ are associated with an arbitrary nanostructure, placed in an environment of permittivity ϵ_{env} (*cf.* Fig. 1). $k_0 = \omega/c$ is the wavenumber of light in vacuum, c the speed of light and the symbol $\times$ is the rotational. ϵ_r and μ_r are the relative dielectric permittivity and magnetic permeability of the nanostructure, respectively. For dispersive media, ϵ_r and μ_r are functions of the frequency ω. They are defined as the ratios of the material's permittivity and permeability relative to the vacuum values ϵ_0 and μ_0. They can be related to the electric and magnetic susceptibilities as $\chi_{\text{e}} = (\epsilon_r - \epsilon_{\text{env}})/4\pi$ and $\chi_{\text{m}} = (\mu_r - \mu_{\text{env}})/4\pi$, respectively. In general, $\chi_{\text{e}}(\mathbf{r},\omega)$ and $\chi_{\text{m}}(\mathbf{r},\omega)$ are functions of frequency and space. In pyGDM we assume non-magnetic media, hence $\mu_r = \mu_{\text{env}} = 1$.

It is possible to derive a wave-equation for the electric field from Maxwell's equations (see e.g. Ref. [13], chapter 9 or Ref. [14]):

$$(\Delta + k^2)\mathbf{E}(\mathbf{r},\omega) = -\frac{4\pi}{\epsilon_{\text{env}}}\left(k^2 + \nabla\nabla\right)\mathbf{P}(\mathbf{r},\omega) \tag{2}$$

where ∇ and Δ are the nabla- and Laplace operators, respectively, $\mathbf{P} = \boldsymbol{\chi}_{\text{e}} \cdot \mathbf{E}$ is the electric polarization and k the wavenumber in the environment medium with $k = \sqrt{\epsilon_{\text{env}}}\, k_0$.

Note. The dielectric function is in general a tensor of rank 2. In pyGDM, an isotropic susceptibility $\chi_{\text{e,iso}}$ is assumed, hence the susceptibility tensor $\boldsymbol{\chi}_{\text{e}}$ is defined as

$$\boldsymbol{\chi}_{\text{e}}(\mathbf{r},\omega) = \begin{bmatrix} \chi_{\text{e,iso}}(\mathbf{r},\omega) & 0 & 0 \\ 0 & \chi_{\text{e,iso}}(\mathbf{r},\omega) & 0 \\ 0 & 0 & \chi_{\text{e,iso}}(\mathbf{r},\omega) \end{bmatrix}. \tag{3}$$

In future versions of pyGDM anisotropic polarizabilities might be supported.

From the wave-equation (2) one can derive a vectorial Lippmann–Schwinger equation for the electric field (see e.g. Ref. [14]):

$$\mathbf{E}(\mathbf{r},\omega) = \mathbf{E}_0(\mathbf{r},\omega) + \int \mathbf{G}^{\text{EE}}_{\text{tot}}(\mathbf{r},\mathbf{r}',\omega) \cdot \boldsymbol{\chi}_{\text{e}} \cdot \mathbf{E}(\mathbf{r}',\omega)\mathrm{d}\mathbf{r}' \tag{4}$$

which relates in a self-consistent manner the incident (or "zero order", "fundamental") electric field $\mathbf{E}_0$ with the total field $\mathbf{E}$ inside the structure of susceptibility $\boldsymbol{\chi}_{\text{e}}$. The integral in Eq. (4) runs over

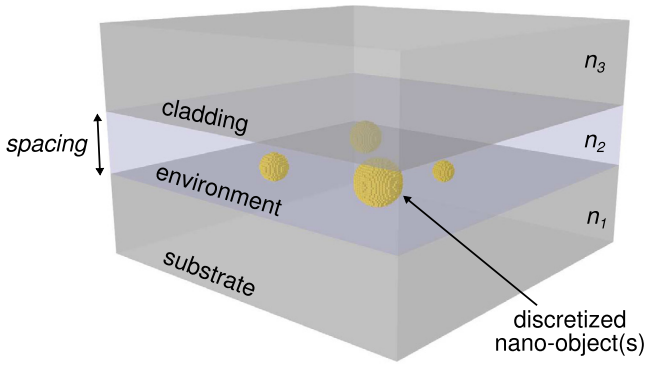

Fig. 2. Geometry of the reference system described by the Green's dyad used in pyGDM: The discretized nano-structure is placed in the environment layer with (complex) refractive index n_2 and of thickness *spacing*. It is sandwiched between a substrate (n_1) and a cladding layer (n_3).

the volume of the structure. $\mathbf{G}_{\text{tot}}^{\text{EE}}$ is the Green's dyad, describing the environment in which the structure is placed (see also Appendix B). The Green's dyadic tensors $\mathbf{G}$ are also called field susceptibilities and were originally introduced by G.S. Agarwal. [15]. For an object in vacuum $\mathbf{G}_{\text{tot}}^{\text{EE}} = \mathbf{G}_0^{\text{EE}}$, which is written as [14,16]

$$\mathbf{G}_0^{\text{EE}}(\mathbf{r}, \mathbf{r}', \omega) = \frac{1}{\epsilon_{\text{env}}}\left(k^2\,\mathbf{I} + \nabla\nabla\right) G_0(\mathbf{r}, \mathbf{r}', \omega)$$
$$= \frac{\mathrm{e}^{ikR}}{\epsilon_{\text{env}}}\left(-k^2\mathbf{T}_1(\mathbf{R}) - ik\mathbf{T}_2(\mathbf{R}) + \mathbf{T}_3(\mathbf{R})\right). \tag{5}$$

$\mathbf{I}$ is the Cartesian unitary tensor, ∇ the nabla operator acting along $\mathbf{r}$ and G_0 the scalar Green's function (see Eq. (16)). The superscript "EE" indicates that the Green's function accounts for an electric–electric interaction. Furthermore we used the abbreviations $\mathbf{R} = \mathbf{r} - \mathbf{r}'$ and

$$\mathbf{T}_1(\mathbf{R}) = \frac{\mathbf{RR} - \mathbf{I}R^2}{R^3} \tag{6}$$
$$\mathbf{T}_2(\mathbf{R}) = \frac{3\mathbf{RR} - \mathbf{I}R^2}{R^4} \tag{7}$$
$$\mathbf{T}_3(\mathbf{R}) = \frac{3\mathbf{RR} - \mathbf{I}R^2}{R^5}. \tag{8}$$

$\mathbf{RR}$ is the tensorial product of $\mathbf{R}$ with itself and R represents its modulus. $\mathbf{T}_1$ describes far-field effects while $\mathbf{T}_2$ and $\mathbf{T}_3$ account for the near-field.

In pyGDM an additional non-retarded Green's dyad is used which allows to include a substrate and a cladding layer (see Fig. 2):

$$\mathbf{G}_{\text{tot}}^{\text{EE}} = \mathbf{G}_0^{\text{EE}} + \mathbf{G}_{\text{3-layer}}\,. \tag{9}$$

Such dyadic function $\mathbf{G}_{\text{3-layer}}$ for a layered reference system can be derived in an asymptotic form using the image charges method (see also Appendix B). The derivation of a retarded Green's dyad for multi-layered systems is explained in detail e.g. in Refs. [17,18]. For a derivation of the Lippmann–Schwinger equation in SI units, see e.g. Ref. [19].

1.2. Volume discretization

For arbitrarily shaped objects, the integral in the Lippmann–Schwinger equation (4) can generally not be solved analytically. In the following we describe a numerical approach which requires the discretization of the integral into a sum over finite size volume elements (see also Ref. [14]). For reasons of clarity the dependency on the frequency ω will be omitted in the following. We discretize the nano-object using N cubic volume elements centered at positions $\mathbf{r}_i$, as illustrated in Fig. 3. The cube side lengths d and thus $V_{\text{cell}} = d^3$ are constant on the mesh.

$$\mathbf{E}(\mathbf{r}_i, \omega) = \mathbf{E}_0(\mathbf{r}_i, \omega) + \sum_{j=1}^{N} \mathbf{G}_{\text{tot}}^{\text{EE}}(\mathbf{r}_i, \mathbf{r}_j, \omega) \cdot \boldsymbol{\chi}_{\text{e}}(\mathbf{r}_j, \omega) \cdot \mathbf{E}(\mathbf{r}_j, \omega) V_{\text{cell}}. \tag{10}$$

We can rewrite Eq. (10) as follows

$$\mathbf{E}_0(\mathbf{r}_i) = \mathbf{E}(\mathbf{r}_i) - \sum_{j=1}^{N} \mathbf{G}_{\text{tot}}^{\text{EE}}(\mathbf{r}_i, \mathbf{r}_j) \cdot \boldsymbol{\chi}_{\text{e}}(\mathbf{r}_j) \cdot \mathbf{E}(\mathbf{r}_j) V_{\text{cell}}$$
$$= \sum_{j=1}^{N} \left(\delta_{ij}\mathbf{I} - \boldsymbol{\chi}_{\text{e}}(\mathbf{r}_j) \cdot V_{\text{cell}}\, \mathbf{G}_{\text{tot}}^{\text{EE}}(\mathbf{r}_i, \mathbf{r}_j)\right) \cdot \mathbf{E}(\mathbf{r}_j) \tag{11}$$

where δ_{ij} is the Kronecker symbol.

Let us now define two $3N$-dimensional vectors containing the ensemble of all electric field vectors in the discretized nano-object

$$\mathbf{E}_{0,\text{obj.}} = \Big(E_{0,x}(\mathbf{r}_1), E_{0,y}(\mathbf{r}_1), E_{0,z}(\mathbf{r}_1), E_{0,x}(\mathbf{r}_2), \ldots, \ldots, E_{0,z}(\mathbf{r}_N)\Big)$$

$$\mathbf{E}_{\text{obj.}} = \Big(E_x(\mathbf{r}_1), E_y(\mathbf{r}_1), E_z(\mathbf{r}_1), E_x(\mathbf{r}_2), \ldots, \ldots, E_z(\mathbf{r}_N)\Big).$$

Together with the $3N \times 3N$ matrix $\mathbf{M}$ composed of 3×3 sub-matrices

$$\mathbf{M}_{ij} = \delta_{ij}\mathbf{I} - \boldsymbol{\chi}_{\text{e}}(\mathbf{r}_j) \cdot V_{\text{cell}} \mathbf{G}_{\text{tot}}^{\text{EE}}(\mathbf{r}_i, \mathbf{r}_j) \tag{12}$$

we obtain a coupled system of $3N$ linear equations

$$\mathbf{E}_{0,\text{obj.}} = \mathbf{M} \cdot \mathbf{E}_{\text{obj.}}\,. \tag{13}$$

If we inverse the matrix $\mathbf{M}$ defined by Eq. (12), we can calculate the field $\mathbf{E}_{\text{obj.}}$ inside the structure for all possible incident fields $\mathbf{E}_{0,\text{obj.}}$ (at frequency ω) by means of a simple matrix–vector multiplication:

$$\mathbf{E}_{\text{obj.}} = \boldsymbol{\kappa} \cdot \mathbf{E}_{0,\text{obj.}}\,, \tag{14}$$

where we used the symbol $\boldsymbol{\kappa}$ for the inverse matrix

$$\boldsymbol{\kappa}(\omega) = \mathbf{M}^{-1}(\omega)\,. \tag{15}$$

$\boldsymbol{\kappa}$ is called the *generalized field propagator*, as introduced by Martin et al. [20].

Note. In our notation, $\boldsymbol{\kappa}$ represents the full $3N \times 3N$ matrix, describing the response of the entire nanostructure. This matrix is composed of 3×3 sub-tensors $\mathbf{K}(\mathbf{r}_i, \mathbf{r}_j)$ for the couples of ith and jth meshpoint.

Note. After Eq. (10), we can use the Green's dyad of the reference system with the field inside the particle in order to calculate the total electric field at any point $\mathbf{r}_i$ outside the nanostructure.

1.3. Renormalization of the Green's dyad

When integrating the polarization distribution in Eq. (4) over the volume of the nanostructure, we integrate scalar Green's functions of the form

$$G_0(\mathbf{r}, \mathbf{r}') = \frac{\mathrm{e}^{ik\,|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r} - \mathbf{r}'|}. \tag{16}$$

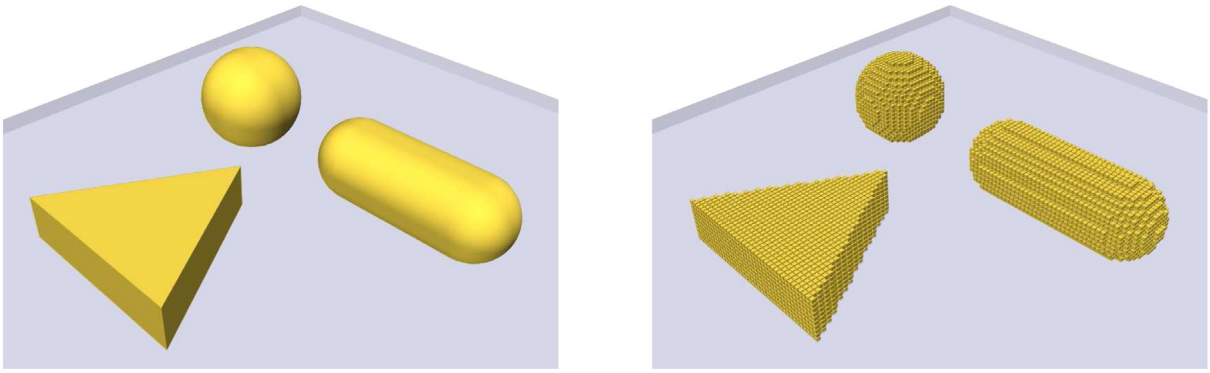

Fig. 3. Arbitrary nanostructure composed of multiple elements lying on a substrate (left) and its volume discretization on a cubic lattice (right).

Obviously, G_0 diverges if $\mathbf{r} = \mathbf{r}'$, which occurs when the field of a point dipole $\mathbf{p}\delta(\mathbf{r} - \mathbf{r}')$ is being evaluated at the dipole's position $\mathbf{r}'$ itself. As a consequence, in order to remove this singularity, we need to apply a regularization scheme [21]. For a three dimensional cubic mesh, a simple renormalization rule for the free-space Green's dyad has been proposed (see Ref. [22], section 4.3):

$$\mathbf{G}^{\mathrm{EE}}_{0,\mathrm{cube}}(\mathbf{r}_i, \mathbf{r}_i) = -\frac{4\pi}{3\epsilon_{\mathrm{env}} d^3}\,\mathbf{I} \tag{17}$$

with d the stepsize of the volume discretization.

The choice of an appropriate mesh can be crucial for the convergence of the method. While structures with flat surfaces and right angles (e.g. cuboids) can be accurately discretized using a cubic mesh, particles with 3-fold symmetry (e.g. prisms) or curved structures like wires of circular section or spherical particles are better described using a hexagonal mesh. A 3D hexagonal compact mesh can be regularized with (see Ref. [16], section 3.1)

$$\mathbf{G}^{\mathrm{EE}}_{0,\mathrm{hex}}(\mathbf{r}_i, \mathbf{r}_i) = -\frac{4\pi\sqrt{2}}{3\epsilon_{\mathrm{env}} d^3}\,\mathbf{I}\,. \tag{18}$$

While a cubic mesh cell has a volume of $V_{\mathrm{cell}} = d^3$, in the hexagonal compact case, the volume of a cell equals $V_{\mathrm{cell}} = d^3/\sqrt{2}$ and also must be accordingly adapted in Eq. (12).

Other geometries like cuboids [23] or tetrahedrons [24] can be used for the mesh as well, but are not implemented in pyGDM so far. Because it accounts for the field of a point dipole at the location of the dipole itself, the sub-matrix $\mathbf{M}_{ii}$ is also called "self-term".

1.4. Multiple monochromatic simulations on the same nanostructure

Once the generalized propagator $\mathbf{K}$ is known, we can calculate the response of the system to arbitrary monochromatic incident fields (e.g. plane waves, focused beams or even fast electrons) by means of a simple matrix–vector multiplication. This can be used for instance to do raster-scan simulations at low numerical cost, by raster-scanning a light source such as a focused incident beam or a dipolar emitter step-by-step over the nano-object, while calculating and eventually post-processing the field at each position [25].

2. Comparison to other electro-dynamical simulation techniques

Before proceeding with a detailed introduction to the pyGDM toolkit, we want to give a non-exhaustive overview of other methods commonly used for solving electro-dynamical problems in nano-optics.

A widely used frequency domain solver is the open source software DDSCAT [26], which implements a frequency domain technique analog to the GDM. It is usually called the "Coupled" or "Discrete Dipole Approximation" (CDA or DDA, respectively). However, there exist two main differences to GDM as used in this work. First, the renormalization problem is circumvented by setting the self-terms to zero and including the corresponding contributions using a physical polarizability for each dipole. Using such physical polarizabilities (usually of spherical entities) for each mesh-cell however leads generally to a worse convergence for larger step-sizes. The second difference is more technical. In the DDSCAT implementation of DDA, the matrix $\mathbf{M}_{\mathrm{DDSCAT}}$ is not stored in memory (cf. Eq. (12)). The resolution of the inverse problem is done by the conjugate gradients method, where the elements $M_{\mathrm{DDSCAT},ij}$ are computed *on-demand* during the calculation of the vector–matrix products $\mathbf{M}_{\mathrm{DDSCAT}} \cdot \mathbf{x} = \mathbf{E}$. To speed up these matrix–vector multiplications, a scheme involving fast Fourier transformations (FFT) is used [27]. A drawback is that without storing $\mathbf{M}$, efficient preconditioning is very difficult (see also Appendix E). Convergence of the DDSCAT conjugate gradient iterative scheme is therefore relatively slow and only obtained for very fine discretization meshes, further slowing down the computation due to the large size of the coupled dipole matrix $\mathbf{M}_{\mathrm{DDSCAT}}$. An obvious advantage of DDSCAT is, that large problems with huge numbers of mesh points can be treated, since the matrix coupling all dipoles is not stored in memory. However, the advantage of the generalized propagator is lost. The calculation of different incident fields at a fixed wavelength (such as raster-scan simulations) requires to re-run the time-consuming conjugate gradients solver for each configuration. Another free implementation of the DDA with particular focus on electron energy loss spectroscopy (EELS) simulations is the DDEELS package [28].

Maxwell's equations can be reformulated as a set of surface-integral equations. It is therefore possible to develop a similar formalism as the above explained volume integral method in which only the surfaces of a nanostructure are discretized instead of the volume [29]. A great advantage of this so-called Boundary Element Method (BEM) is the smaller amount of discretization cells, which however comes at the cost of a more complex mathematical framework and numerical implementation. With MNPBEM an open-source BEM-implementation for MATLAB exists which allows also the consideration of layered environments [30,31].

Another very popular and flexible technique for electro-dynamical simulations is the Finite-Difference Time-Domain (FDTD) method [32–34]. As the name suggests, the calculation is performed in the time domain, which means that Maxwell's equations are iteratively evolved by small time increments. The problem is discretized in both, space and time. An incoming wave travels time-step by time-step across the region of interest and when the wave-packet has passed or turn-on effects have fully decayed (e.g. for plane wave illumination), the actual numerical measurement is performed. With respect to computational time, a disadvantage is the additional dimension (time) that needs to

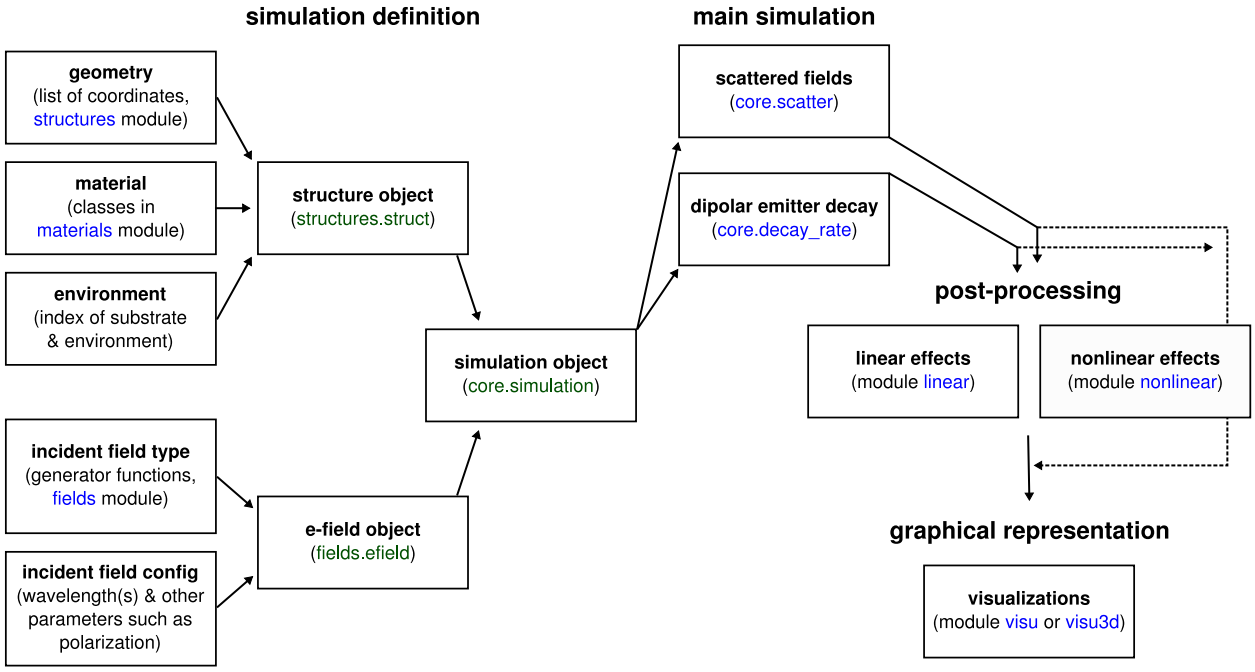

Fig. 4. Structure of the pyGDM package and workflow of a typical simulation: (1) Setup of the geometry, environment and incident electric field. This is bundled in an instance of the **simulation** object. (2) Main GDM simulation. (3) Possible post-processing (e.g. calculation of extinction cross-sections). (4) Visualization of the results.

be discretized. Furthermore, a fraction of the environment around the object of interest has to be included in the discretization space, which is why FDTD is called a "domain discretization technique". Particularly in 3D problems, this can lead to very high computational costs. Another drawback of FDTD can be the low accuracy for near-field intensities if very strong field enhancements occur (*e.g.* in plasmonics) [35]. However, the simplicity and the robustness of the method are great advantages of FDTD. Furthermore, using temporally short and therefore spectrally broad illumination pulses, a large frequency spectrum can be obtained in a single simulation run. Frequency domain techniques on the other hand require each wavelength to be calculated separately. Provided an accurate analytical model for the material dispersion exists, this advantage can compensate the larger discretization domain in spectral simulations, compared to frequency domain methods like the GDM. A powerful open source implementation that comes with a rich toolbox is the software "MEEP" [36]. For a general introduction on finite difference methods, see for example Ref. [37], chapter 17.

Finally, a very popular domain discretization technique in the frequency domain is the Finite Element Method (FEM, e.g. implemented in the commercial software "COMSOL Multiphysics"). Due to its adjustable mesh-size it is particularly apt for plasmonic problems, where extremely localized fields can occur at sharp extremities. However, it suffers from the same drawback as FDTD since a certain volume around the nano-object needs to be discretized and included in the calculation, often leading to high memory and CPU-time requirements.

A review including benchmarks for different numerical techniques in nano-optics can be found in Ref. [38]. An extensive discussion of different DDA variants including a detailed review on their accuracies is given in Ref. [39].

3. Setting up a pyGDM simulation

The structure of the pyGDM package and the main steps to set up and run a simulation are schematically depicted in Fig. 4. The heart of pyGDM is the **simulation** object which contains the information about the structure, its environment and the incident electro-magnetic field(s) used in the simulation:

core.simulation
(class)

A minimal example of a pyGDM python script is provided in Appendix B.

3.1. Geometry and material dispersion

The geometry of the nanostructure and the dielectric constant of both its constituent material and the environment are stored in an instance of

structures.struct
(class)

which contains the geometry as a list of mesh-point coordinates and the material dispersion via an instance of some **materials.dispersion_class**.

3.2. Excitation fields

The second key-ingredient of a pyGDM-simulation is the incident (illuminating) electro-magnetic field.

The fields in the GDM are time-harmonic, oscillating at frequency ω. We describe these fields using the phasor description with complex amplitudes in which we include the phase information:

$$\tilde{\mathbf{E}}(\mathbf{r},\omega,t) = \hat{\mathbf{E}}(\mathbf{r},\omega)\,\mathrm{e}^{-\mathrm{i}\omega t}\,\mathrm{e}^{\mathrm{i}\varphi} = \mathbf{E}(\mathbf{r},\omega)\,\mathrm{e}^{-\mathrm{i}\omega t}. \tag{19}$$

$\tilde{\mathbf{E}}$ is the electric field including the time-dependence. We assume time-harmonicity, thus the time-dependence is expressed by the term $\mathrm{e}^{-\mathrm{i}\omega t}$. $\hat{\mathbf{E}}$ is the real valued amplitude, $\mathbf{E}$ the complex amplitude (the "phasor") which includes the phase-factor $\mathrm{e}^{\mathrm{i}\varphi}$ in its imaginary part.

The information about the incident field is provided to pyGDM via

fields.efield
(class)

Illustrations of the below listed incident fields available in pyGDM are shown in Fig. 5.

3.2.1. Plane wave

fields.planewave
(function)

The probably most common fundamental field is the plane wave, which is in many cases a sufficient approximation. Its complex amplitude can be expressed as

$$\mathbf{E}_0(\mathbf{r}, \omega) = \mathbf{E}_0\, e^{i\mathbf{k}\cdot\mathbf{r}}. \tag{20}$$

3.2.2. Focused plane wave

fields.focused_planewave
(function)

The simplest approximation for a focused beam is a plane wave with a Gaussian intensity profile. For incidence along Z ($\mathbf{k} \parallel \mathbf{e}_z$) this is written as:

$$\mathbf{E}_0(\mathbf{r}, \omega) = \mathbf{E}_0\, e^{i\mathbf{k}\cdot\mathbf{r}} \exp\left(\frac{(x-x_0)^2 + (y-y_0)^2}{2w_{\text{spot}}^2}\right) \tag{21}$$

The beam propagates along (x_0, y_0, z). The full width at half maximum (FWHM) can be obtained via

$$w_{\text{FWHM}} = w_{\text{spot}} \cdot 2\sqrt{2\ln 2}\,. \tag{22}$$

A focused plane wave is often a sufficient approximation (see e.g. Ref. [40]) and can be particularly useful if the divergence of the radius of curvature at the origin of the paraxial Gaussian becomes problematic.

3.2.3. Paraxial Gaussian beam

fields.gaussian
(function)

arguments:

- *paraxial:* **True** (default: **False**)

Often, lasers are used as sources of monochromatic, coherent light with high intensity. Light emitted from a laser-cavity is however not propagating like a plane wave, but as a Gaussian beam. The intensity profile differs significantly from the focused plane wave so the use of a model for Gaussian beams may become necessary — particularly in larger objects, where the "curved" intensity profile of such a beam induces important field gradients along the propagation direction and the particle. A popular approximation to a real Gaussian beam is the so-called *paraxial approximation*, where all $\mathbf{k}$-vectors are parallel to one single propagation direction. It can be calculated using the following formula (propagation along Z-axis)

$$\mathbf{E}_0(\mathbf{r}, \omega) = \mathbf{E}_0 \frac{w_0}{w(z)} \exp\left(\frac{-r^2}{w(z)^2}\right) \times \exp\left(-i\left(k\left(z + \frac{r^2}{2R(z)}\right) - \zeta(z)\right)\right) \tag{23}$$

with the beam width or "waist" w_0 and the squared distance to the beam axis $r^2 = (\Delta x^2 + \Delta y^2)$. Δx and Δy are the distances to the beam axis in X and Y directions, respectively. In Eq. (23) we introduced furthermore the z-dependent beam waist

$$w(z) = w_0\sqrt{1 + \left(\frac{z\lambda}{\pi w_0^2}\right)^2} \tag{24}$$

the radius of curvature

$$R(z) = z\left(1 + \left(\frac{\pi w_0^2}{z\lambda}\right)^2\right) \tag{25}$$

and the *Gouy phase* [41]

$$\zeta(z) = \arctan\left(\frac{z\lambda}{\pi w_0^2}\right). \tag{26}$$

3.2.4. Tightly focused Gaussian beam

fields.gaussian
(function)

arguments:

- *paraxial:* **False** (=default)

Under tight focusing conditions an additional component $E_{0,z}$ parallel to the wave-vector (again assuming $\mathbf{k} \parallel \mathbf{e}_z$) can gain a substantial magnitude, which can be explained by the div$\mathbf{E}$ Maxwell's equation. This can be accounted for by adding the following correction term to the paraxial Gaussian (again assuming propagation along Z) [42]

$$E_{0,z}(x, y, z) = \frac{-2i}{kw(z)^2}\left(\Delta x\, E_{0,x} + \Delta y\, E_{0,y}\right). \tag{27}$$

3.2.5. Dipolar emitter

fields.dipole_electric
(function)

An electric dipole $\mathbf{p}$ placed in a homogeneous environment at $\mathbf{r}_0$ and oscillating at frequency ω creates an electric field at $\mathbf{r}$ which is written as [15]

$$\mathbf{E}_p(\mathbf{r}, \mathbf{r}_0, \omega) = \frac{1}{\epsilon_{\text{env}}}\left(\mathbf{I}\, k^2 + \nabla\nabla\right) G_0(\mathbf{r}, \mathbf{r}_0, \omega) \cdot \mathbf{p}(\omega) \tag{28}$$

where ∇ acts along $\mathbf{r}$, $\mathbf{I}$ is the unitary tensor, k the wavenumber and G_0 the scalar vacuum Green's function (see Eq. (16)).

3.2.6. Magnetic dipole emitter

fields.dipole_magnetic
(function)

Analogously, a magnetic dipole emitter $\mathbf{m}$ at $\mathbf{r}_0$ is the source of an electric field [15,43]

$$\mathbf{E}_m(\mathbf{r}, \mathbf{r}_0, \omega) = ik_0 \nabla \times G_0(\mathbf{r}, \mathbf{r}_0, \omega) \cdot \mathbf{m}(\omega). \tag{29}$$

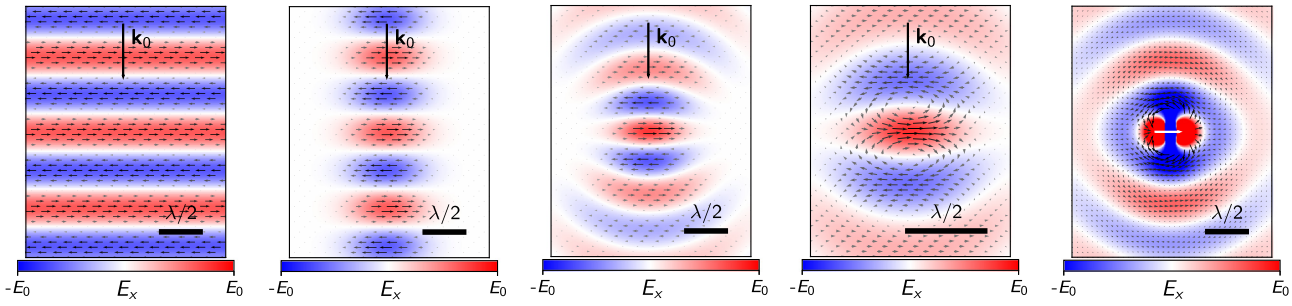

Fig. 5. Real part of E_x for (from left to right): A plane wave, a "focused plane wave", a paraxial Gaussian beam, a tight-focus corrected paraxial Gaussian beam (all X-polarized, $\mathbf{k} \parallel -\mathbf{e}_z$) and a dipole emitter along X (indicated by a white arrow).

4. Solver

4.1. Internal fields

Solving the primary scattering problem is usually the root of a GDM simulation. The self-consistent calculation of the fully retarded (complex) electric field inside the nano-structure is done by inversion of Eq. (13) via

core.scatter
(function)

Usually, the underlying *scipy* libraries used for inversion are multi-thread parallelized, making use of all processors on multi-core CPUs.

For distributed systems like most modern computing clusters, also a multi-processing parallel version of **core.scatter** is implemented in pyGDM, which uses MPI to simultaneously calculate several wavelengths of a spectrum on parallel processes:

core.scatter_mpi
(function)

Note. Each MPI process calculates a single wavelength using the parallelized *scipy*-routines. In this double-parallelized way, spectral simulations can be carried out very rapidly on multi-node computing clusters. **core.scatter_mpi** requires the "mpi4py" package.

4.1.1. Direct inversion

- argument *method*: "lu" (default), "scipyinv", "superlu", "pinv2" (all require *scipy*), "numpyinv" or "dyson" (only *numpy*)

In pyGDM the inversion of $\mathbf{M}$ (Eq. (12)) is by default performed with LU-decomposition (using the implementation in *scipy*). This should be the fastest solver for full inversion (see Fig. 6a) and has furthermore an excellent multi-threaded parallelization scaling, as can be seen in Fig. 7. An extensive explanation of LU-decomposition and details on its implementation can be found for example in Ref. [37] (chapter 2.3). Other scipy solvers can be used in pyGDM, and, if for any reason *scipy* is not available, the "numpyinv" and "dyson" methods are alternatives which do not require *scipy*.

The solver "dyson" uses a sequence of Dyson's equations [20] and comes with pyGDM. Since it does not depend on any libraries it should work in every case, however it will usually be significantly slower than the third-party solvers. An advantage of "dyson" can be the memory requirement which is relatively low, because the Dyson sequence allows an in-place inversion of the matrix (see Fig. 6b). A detailed description of the latter algorithm can be found in Ref. [44] (chapter 2.4).

We note that LU inversion (or in some cases conjugate gradients, e.g. for dense spectra on single-core systems, see below and Appendix) is the preferred technique in pyGDM due to its high efficiency (see Fig. 6a).

4.1.2. Conjugate gradients

- argument *method*: "cg" or "pycg"

Sometimes it is not necessary to calculate the full structure of the inverse of matrix $\mathbf{M}$. Often it is sufficient to only know the *result* of the matrix–vector product $\mathbf{M}^{-1}\mathbf{E}_0$. It turns out that under certain circumstances, iterative approaches such as the "conjugate gradients" method lead to very accurate approximations of this matrix/vector product in significantly less time compared to the inversion of $\mathbf{M}$. For a detailed description and informations related to the conjugate gradients solver, see Appendix E.

4.2. Decay-rate of dipolar emitters

core.decay_rate
(function)

The Green's Dyadic formalism can be used not only to obtain scattered electro-magnetic fields. It gives also direct access to the modification of the decay rate of electric or magnetic dipolar transitions due to the presence of polarizable materials in their vicinity.

Note. The decay rates are proportional to the photonic LDOS [45], hence the values obtained from the calculation of the relative decay rates Γ/Γ_0 are identical to the relative LDOS (specifically to the *partial* LDOS, meaning its electric or magnetic component and/or partial for specific dipole orientations).

4.2.1. Electric dipole

The effect is intuitively understandable for an electric dipole transition $\mathbf{p}$, as a consequence of the enhancement (or weakening) of the electric near-field because of the dielectric contrast and the resulting back-action of the radiated field on the dipole itself. It is possible to derive an integral equation describing the decay rate Γ_e of the dipole transition [45,43]:

$$\Gamma_e(\mathbf{r}_0,\omega) = \Gamma_e^0(\omega) \times \left(1 + \frac{3}{2k_0^3}\mathbf{u}\cdot\mathrm{Im}\left(\mathcal{G}_p^{\mathrm{EE}}(\mathbf{r}_0,\mathbf{r}_0,\omega)\right)\cdot\mathbf{u}\right), \quad (30)$$

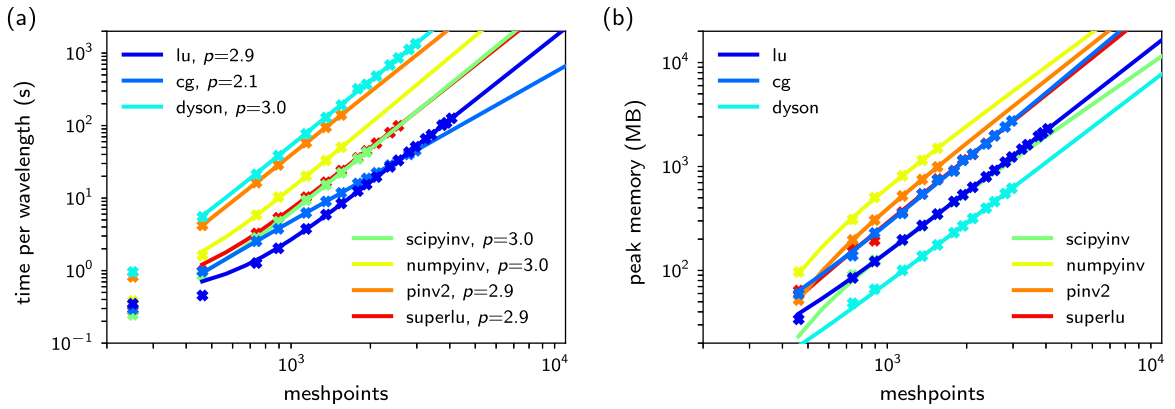

Fig. 6. (a) Timings of a pyGDM-simulation of a spherical dielectric particle as a function of the number of meshpoints for the different available solvers. Solid lines are power-law fits, confirming $p = 3$ for full inversion methods and $p = 2$ for CG (the fitted power p is given in the legend). (b) Memory requirement (in megabytes) as function of the number of meshpoints for the different solvers. All benchmarks were performed on a single core of an AMD FX-8350 CPU.

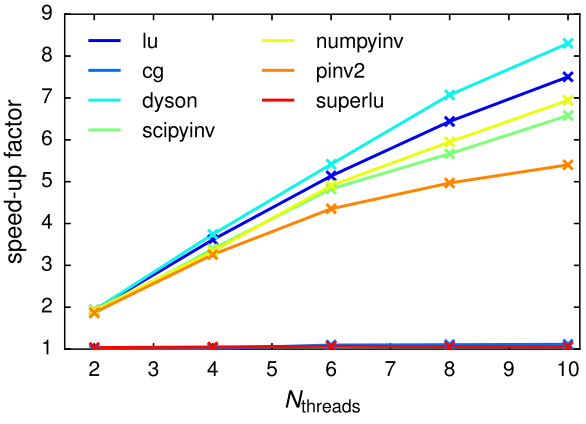

Fig. 7. Speedup of the GDM-calculation using the multi-threaded parallelization capability of the available solvers. Benchmark performed on an Intel E5-2680 10-core CPU.

where

$$\mathcal{G}_p^{\mathrm{EE}}(\mathbf{r}, \mathbf{r}_0, \omega) = \int_V \mathrm{d}\mathbf{r}' \int_V \mathrm{d}\mathbf{r}'' \mathbf{G}_0^{\mathrm{EE}}(\mathbf{r}, \mathbf{r}', \omega) \cdot \boldsymbol{\chi}_{\mathrm{e}}(\mathbf{r}', \omega) \cdot \mathbf{K}(\mathbf{r}', \mathbf{r}'', \omega) \cdot \mathbf{G}_0^{\mathrm{EE}}(\mathbf{r}'', \mathbf{r}_0, \omega) \tag{31}$$

and $\Gamma_e^0(\omega) = 4k_0^3 p^2/3\hbar$ is the decay rate of the electric dipole transition in vacuum. $\mathbf{r}_0$ is the location of the dipolar transition (outside the nanostructure). $\mathbf{u}$ denotes the dipole orientation, p its amplitude. $\mathbf{K}$ is the generalized propagator (see Eq. (14) or Ref. [20]). For the numerical implementation, the integrals in Eq. (31) become sums over the mesh-points of the discretized nano-object(s).

The propagator $\mathbf{G}_0^{\mathrm{EE}}$ can be found by identification using Eq. (28) and following equation for the field of an electric dipole $\mathbf{p}$ at $\mathbf{r}_0$ [15]

$$\mathbf{E}_0(\mathbf{r}, \omega) = \mathbf{G}_0^{\mathrm{EE}}(\mathbf{r}, \mathbf{r}_0, \omega) \cdot \mathbf{p}(\omega)\,, \tag{32}$$

Note. $\mathbf{G}_0^{\mathrm{EE}}$ is given for particles in a homogeneous environment by the Green's Dyad of Eq. (5). In analogy to the scattering simulations it can be easily extended for more complex environments, such as an infinite substrate (see Section 1).

4.2.2. Magnetic dipole

Also the decay rate of a *magnetic* dipole transition close to *non-magnetic* materials is influenced by the presence of the structure. Such magnetic–magnetic response function associated to a structure with no direct magnetic response, arises from the *electric* field emitted by the magnetic dipole, interacting with the material and finally again inducing a magnetic field via the curl of the electric field. In metallic nanostructures, circular plasmonic currents can also lead to significant magnetic near-field enhancements [46]. In complete analogy to Eq. (30), the magnetic decay rate Γ_m is written as

$$\Gamma_m(\mathbf{r}_0, \omega) = \Gamma_m^0(\omega) \times \left(1 + \frac{3}{2k_0^3}\mathbf{u} \cdot \mathrm{Im}\big(\mathcal{G}_p^{\mathrm{HH}}(\mathbf{r}_0, \mathbf{r}_0, \omega)\big) \cdot \mathbf{u}\right)\,, \tag{33}$$

where

$$\mathcal{G}_p^{\mathrm{HH}}(\mathbf{r}, \mathbf{r}_0, \omega) = \int_V \mathrm{d}\mathbf{r}' \int_V \mathrm{d}\mathbf{r}'' \mathbf{G}_0^{\mathrm{HE}}(\mathbf{r}, \mathbf{r}', \omega) \cdot \boldsymbol{\chi}_{\mathrm{e}}(\mathbf{r}', \omega) \cdot \mathbf{K}(\mathbf{r}', \mathbf{r}'', \omega) \cdot \mathbf{G}_0^{\mathrm{EH}}(\mathbf{r}'', \mathbf{r}_0, \omega) \tag{34}$$

and $\Gamma_m^0(\omega) = 4k_0^3 m^2/3\hbar$ is the decay rate of the magnetic transition in vacuum. $\mathbf{u}$ and m are the magnetic dipole orientation and amplitude, respectively, and $\mathbf{K}$ is again the generalized propagator.

In the same way as for the electric dipole, $\mathbf{G}_0^{\mathrm{HE}}$ and $\mathbf{G}_0^{\mathrm{EH}}$ can be found using Eq. (29) with the electric field of a magnetic dipole $\mathbf{m}$ at $\mathbf{r}_0$

$$\mathbf{E}_0(\mathbf{r}, \omega) = \mathbf{G}_0^{\mathrm{EH}}(\mathbf{r}, \mathbf{r}_0, \omega) \cdot \mathbf{m}(\omega) \tag{35}$$

and

$$\mathbf{G}_0^{\mathrm{HE}}(\mathbf{r}, \mathbf{r}', \omega) = \mathbf{G}_0^{\mathrm{EH}}(\mathbf{r}', \mathbf{r}, \omega). \tag{36}$$

For a detailed derivation of the formalism see Ref. [43]. For a comparison of our code with experimental results, see Ref. [47].

4.2.3. LDOS inside a nanostructure

The decay rate (and hence the LDOS) at a position $\mathbf{r}_{0,s}$ *inside* the structure can also be obtained via Eq. (30) (for the electric case), using the field susceptibility $\mathcal{G}_{p,s}^{\mathrm{EE}}$ inside the structure. It is related to the generalized propagator (assuming an isotropic medium with $\chi_{\mathrm{e}} = \mathrm{Tr}(\boldsymbol{\chi}_{\mathrm{e}})/3$), by

$$\mathcal{G}_{p,s}^{\mathrm{EE}}(\mathbf{r}_i, \mathbf{r}_j, \omega) = \frac{\mathbf{K}(\mathbf{r}_i, \mathbf{r}_j, \omega) - \mathbf{I}}{\chi_{\mathrm{e}}\, V_{\mathrm{cell}}}\,, \tag{37}$$

where $\mathbf{r}_i$ and $\mathbf{r}_j$ are positions of nano-particle meshpoints.

For the case of the magnetic LDOS inside the structure, an "electric-magnetic" mixed generalized propagator needs to be calculated. This propagator relates any incident *electric* field to the scattered *magnetic* field inside the structure. This is not implemented in pyGDM so far, but could easily be made available using the mixed tensor $\mathbf{G}_0^{\mathrm{EH}}$ instead of the electric–electric Green's Dyadic function in the inversion problem, defined by Eq. (10).

Note. The frequency shift of the emitter due to the presence of a nano-structure ("Lamb shift") can be obtained in analogy to the decay rate, via the real part of the field susceptibility [48]. This is however not (yet) implemented in pyGDM.

5. Post-processing

5.1. Linear effects

After the main simulation (calculation of the fields inside the structure, decay rate), the information can be further processed to obtain experimentally accessible physical quantities.

5.1.1. Near-field outside the nanostructure

linear.nearfield
(function)

Electric field:. Via Eq. (10) the electric field induced at any point $\mathbf{r}$ at the exterior of the particle can be calculated from the electric polarization inside the structure.

Magnetic field:. The propagator $\mathbf{G}_0^{\mathrm{HE}}$ (see also Eq. (35)) can be used to obtain the magnetic field outside the source region [49]. **linear.nearfield** returns both, the electric and the magnetic field amplitudes for the scattered as well as for the total near-field ($\mathbf{E}_{\mathrm{tot}} = \mathbf{E}_{\mathrm{scat}} + \mathbf{E}_0$).

Note. Alternatively, the $\mathbf{B}$-field may be calculated via finite differentiation: After Faraday's induction law from Maxwell's equations (Eq. (1b)), the magnetic field is written as (for time-harmonic fields)

$$\mathbf{B}(\mathbf{r}, \omega) = \frac{\nabla \times \mathbf{E}(\mathbf{r}, \omega)}{\mathrm{i}k_0}. \tag{38}$$

5.1.2. Extinction, absorption and scattering cross-sections

linear.extinct
(function)

The linear response in the farfield can be characterized by the scattered and absorbed light intensity, the sum of which is called the "extinction". Usually these values are given as cross sections $\sigma_{\mathrm{scat.}}$, $\sigma_{\mathrm{abs.}}$ and $\sigma_{\mathrm{ext.}}$ which have the unit of an area. The extinction and absorption cross sections can be calculated from the near-field in the discretized structure [50]

$$\sigma_{\mathrm{ext}} = \frac{4\pi k}{|E_0|^2} \sum_{i=1}^{N_{\mathrm{cells}}} \mathrm{Im}\left(\mathbf{E}_{0,i}^* \cdot \mathbf{P}_i\right) \tag{39}$$

and

$$\sigma_{\mathrm{abs}} = \frac{4\pi k}{|E_0|^2} \sum_{i=1}^{N_{\mathrm{cells}}} \left(\mathrm{Im}\left(\mathbf{P}_i \cdot \mathbf{E}_i^*\right) - \frac{2}{3}k^3|\mathbf{P}_i|^2\right). \tag{40}$$

$\mathbf{E}_i$ and $\mathbf{P}_i$ are the electric field and polarization at meshpoint i, respectively, induced by an excitation field $\mathbf{E}_{0,i}$. k is the wavenumber in the particle's environment. Complex conjugation is indicated with a superscript asterisk (*).

The scattering cross section finally is the difference of extinction and absorption

$$\sigma_{\mathrm{scat}} = \sigma_{\mathrm{ext}} - \sigma_{\mathrm{abs}}. \tag{41}$$

5.1.3. Far-field pattern of the scattered light

linear.farfield
(function)

The complex electric field in the far-field, radiated from an arbitrary polarization distribution can be calculated using a corresponding Green's Dyad $\mathbf{G}_{\mathrm{ff}}$ (assuming a dipolar emission from each of the N meshpoints):

$$\mathbf{E}_{\mathrm{ff}}(\mathbf{r}) = \sum_i^{N_{\mathrm{cells}}} \mathbf{G}_{\mathrm{ff}}(\mathbf{r}_i, \mathbf{r}) \cdot \mathbf{P}(\mathbf{r}_i). \tag{42}$$

In vacuum, using Eq. (5) with only the far-field term $\mathbf{T}_1$, we can calculate the electric field at any point $\mathbf{r}$ far enough from the scatterer.

A substrate can be included in the asymptotic tensor by means of an appropriate dyadic Green's function. An analytic approximation of a farfield-propagator for a layered system has been derived e.g. by Novotny [51]. Making use of the superposition principle, the radiation of single dipoles via the propagator $\mathbf{G}_{\mathrm{ff}}$ can be generalized to the total far-field radiation of an ensemble of N dipole-emitters by simple summation of all meshpoints' contributions (see Eq. (42)).

Note that the presence of the illuminated nano-structure is fully taken into account also in this scattering formalism, thanks to the self-consistent nature of the Green's method.

Particularly in nano-structures with high absorption, Eq. (41) requires a high accuracy of the extinction and absorption cross-sections, hence small discretization steps, which can be practically not feasible [50]. In such case, Eq. (42) offers a more precise alternative to determine the scattering cross-section. A further drawback of the calculation of the scattering spectra from the near-field via Eq. (41) is obvious: These spectra do not contain any information about the directionality of the scattering. Using Eq. (42) on the other hand, the spatial distribution and polarization of scattered light in the far-field and can be obtained.

In pyGDM, **linear.farfield** implements a Green's dyad including the contribution of an optional dielectric substrate (in a non-retarded approximation [51]).

5.1.4. Heat generation

Having calculated the electric fields inside a nano-object, it is possible to compute the heat deposited inside the nanoparticle by an optical excitation as well as the temperature rise in the vicinity of the structure [52].

linear.heat
(function)

The total heat generated inside the nanoparticle is the product of the imaginary part of the material's permittivity and the electric field intensity:

$$\begin{aligned} Q(\omega) &= \int_V q(\mathbf{r}, \omega)\, \mathrm{d}\mathbf{r} \\ &= \frac{\omega}{8\pi} \int_V \mathrm{Im}\big(\epsilon(\mathbf{r})\big)|\mathbf{E}(\mathbf{r}, \omega)|^2 \mathrm{d}\mathbf{r}. \end{aligned} \tag{43}$$

linear.temperature
(function)

The temperature rise at position $\mathbf{r}_{\text{probe}}$ outside the nanoparticle can be approximated with the heat $q(\mathbf{r}, \omega)$, generated at each meshpoint (located at $\mathbf{r}$) via the thermal Poisson's equation [25,53]

$$\Delta T(\mathbf{r}_{\text{probe}}, \omega) = \frac{1}{4\pi\kappa_{\text{env}}} \int_V \left(\frac{q(\mathbf{r}, \omega)}{|\mathbf{r}_{\text{probe}} - \mathbf{r}|} + \left(\frac{\kappa_{\text{sub}} - \kappa_{\text{env}}}{\kappa_{\text{sub}} + \kappa_{\text{env}}} \right) \frac{q(\mathbf{r}, \omega)}{|\mathbf{r}_{\text{probe}} - \mathbf{r}|} \right) \mathrm{d}\mathbf{r} \tag{44}$$

where κ_{env} and κ_{sub} are the heat conductivities of the environment and substrate, respectively. The second term in the integrand can be derived through a formalism similar to image charges in electrodynamics and accounts for heat reflection at the interface of the substrate [53]. Eqs. (43) and (44) can be for example used to calculate raster-scan mappings of the deposited heat or the temperature increase as function of a focused beam's focal spot position. Eq. (44) can also be used to compute maps of the temperature increase above a nanostructure, by raster-scanning $\mathbf{r}_{\text{probe}}$ under constant illumination conditions.

Note. Eq. (44) assumes that the heat q generated by the optical excitation at each meshpoint induces a static heat distribution inside the nanoparticle. This approximation might become inaccurate in large nanoparticles of material with high heat conductivity (e.g. metals), leading to a rapid redistribution of the heat inside the nanostructure [52]. If the temperature increase is evaluated at sufficiently large distances to the nano-object, Eq. (44) is usually a good approximation also for larger metallic nano-objects [54]. "Sufficiently large distances" could mean comparable to, or larger than the size of the nanoparticle.

5.1.5. Dipolar emitter decay rate

linear.decay_eval
(function)

The decay rate of magnetic or electric dipole emitters can be calculated within the GDM as described in Section 4.2. Via Eq. (30), the tensor $\mathcal{S}_p^{\text{EE}}$ (or $\mathcal{S}_p^{\text{HH}}$ using Eq. (33)) can be used to calculate the decay rate of the transition for arbitrary orientations of the dipole.

core.decay_rate calculates the tensor $\mathcal{S}_p^{\text{EE}}$ or $\mathcal{S}_p^{\text{HH}}$ (for an electric, respectively magnetic dipole emitter) at each user-defined dipole position and wavelength. The final evaluation of the decay rate is done using **linear.decay_eval** for a given dipole orientation and amplitude. The advantage of this two-step approach is that the generalized propagator needs to be computed only once, and the results of this expensive part of the simulation can be re-used for multiple dipole orientations and/or amplitudes.

5.2. Non-linear effects

5.2.1. Two-photon photoluminescence/surface LDOS

nonlinear.tpl_ldos
(function)

Having calculated the electric field distribution inside a nanoparticle, a simple model allows to calculate the two-photon photoluminescence (TPL) signal generated by the excitation: We assume that the TPL is proportional to the square of the electric field intensity. We furthermore consider each meshpoint (at position $\mathbf{r}$) as an incoherent source of TPL, contributing to the total TPL with an intensity proportional to $|\mathbf{E}(\mathbf{r}, \omega)|^4$. Integration over the nano-particle volume V results in the total TPL intensity [25]:

$$I_{\text{TPL}}(\mathbf{r}_{\text{focus}}, \omega) \propto \int_V |\mathbf{E}(\mathbf{r}, \mathbf{r}_{\text{focus}}, \omega)|^4 \mathrm{d}\mathbf{r}. \tag{45}$$

Here we added a further parameter, the focal spot position $\mathbf{r}_{\text{focus}}$ of a focused illumination. By performing a raster-scan over the nanostructure with the focal position coordinate, we can calculate 2D scanning TPL-maps.

This approach allows also to approximate the photonic local density of states at the surface of the nanostructure $\rho_{\text{sf}}(\mathbf{r}, \omega)$ on which the focused spot impinges (surface LDOS), using an unphysically tightly focused beam. In the case of a circularly polarized excitation, it is possible to rewrite the TPL intensity of Eq. (45) [25,40,55]:

$$I_{\text{TPL}}(\mathbf{r}_{\text{focus}}, \omega) \propto \int_V \left| \mathbf{E}_0^{\circlearrowleft}(\mathbf{r}, \mathbf{r}_{\text{focus}}, \omega) \right|^4 \rho_{\text{sf},\parallel}^2(\mathbf{r}, \omega) \mathrm{d}\mathbf{r} \tag{46}$$

where $\mathbf{E}_0^{\circlearrowleft}$ is the incident electric field and $\rho_{\text{sf},\parallel}$ is the component of the LDOS in the plane parallel to the incident electric field vector. Let us now decrease the waist of the focused beam: In the limit of a spatial profile of $\mathbf{E}_0^{\circlearrowleft}$ corresponding to a Dirac delta function, the square root of the TPL intensity Eq. (46) becomes proportional to the LDOS at the position of the focal spot

$$I_{\text{TPL}}(\mathbf{r}_{\text{focus}}, \omega) \propto \rho_{\text{sf},\parallel}^2(\mathbf{r}_{\text{focus}}, \omega). \tag{47}$$

In consequence, a 2D map of the LDOS can be calculated via a raster-scan simulation, which can be done very efficiently in pyGDM thanks to the generalized propagator. By using a linear polarized incident field, it is furthermore possible to extract partial contributions to the LDOS for the corresponding polarization.

Note. The "surface"-LDOS is reproduced by Eq. (47) for a contraction of $\mathbf{E}_0^{\circlearrowleft}$ towards a Dirac delta function. However, due to the finite stepsize in the GDM, the beam waist cannot be reduced to an infinitely small value, hence this method remains approximative. Practical values for the waist must be at least as large as a few times the discretization stepsize. To obtain the exact LDOS, the calculation of the decay-rate is the method of choice (see also Section 4.2).

6. Visualization

pyGDM includes several visualization tools for simple and rapid plotting of the simulation results. They are divided into functions for the visualization of 2D representations and functions for 3D plots.

6.1. 2D visualization tools

The available visualization functions are explained in the following, examples are given in Fig. 8 using a simulation of a 450 nm $\times$ 90 nm large gold rod with stepsize $d = 15$ nm, excited with a plane wave at $\lambda_0 = 600$ nm, linearly polarized along X and incident from the reader towards the paper ($\mathbf{k} = -\mathbf{e}_z\, k$). The plots show projections on the XY plane.

6.1.1. Structure geometry

visu.structure
(function)

Plot a 2D projection of the simulated nano-particle geometry (see Fig. 8a, meshpoints in golden color).

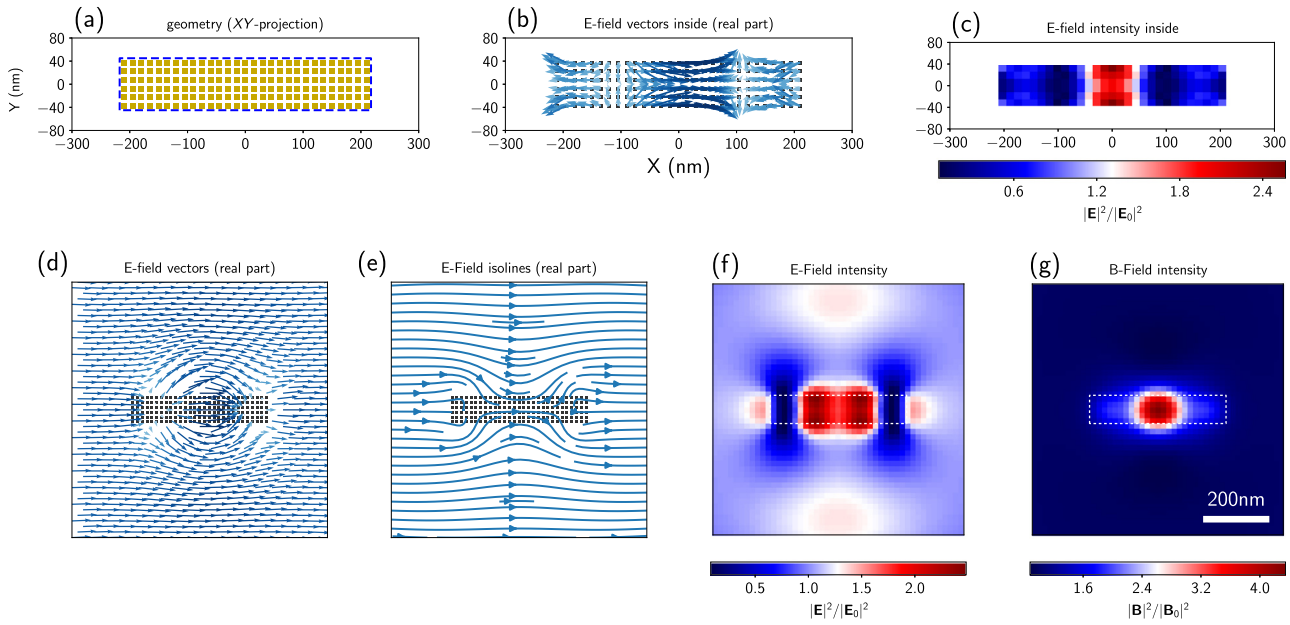

Fig. 8. Visualization tools available in pyGDM on the example of a $450 \times 90 \times 45$ nm^3 ($L \times W \times H$) gold-rod placed in vacuum. Plane wave illumination incident along $-Z$, linear polarization along X, $\lambda = 600$ nm. All plots show projections on the XY-plane. (a) the geometry (gold) and its surface contour (dashed blue), (b) the real part of the internal electric field and (c) the internal electric field intensity at the bottom of the rod. (d–g) show external fields, calculated on an 800×800 nm^2 large area, 30 nm below the structure using **linear.nearfield**: (d) **E**-field real part, (e) isolines of **E**-field, (f) electric field intensity and (g) magnetic field intensity.

visu.structure_contour
(function)

Plot a contour around a 2D projection of the nano-particle, in other words drawing the outer surface of the structure (see Fig. 8, dashed blue line in (a), dashed white lines in (f–g)).

6.1.2. Plot field vectors

2D projections of the real or imaginary part of vector-fields (see Fig. 8d) can be plotted using

visu.vectorfield
(function)

Alternatively, the function can be called using:

visu.vectorfield_by_fieldindex
(function)

The intention of the latter is to be used for direct plotting of fields inside the simulated particle via the **core.simulation** object (see Fig. 8b).

6.1.3. Field lines ("stream-plot")

Isolines of the field amplitude can be plotted using

visu.vectorfield_fieldlines
(function)

For an example, see Fig. 8e.

6.1.4. Scalar field representation (color-plot)

Color-plots are well suited to illustrate a scalar representation of the electric- or magnetic-field. This can be used to represent either the real/imaginary part of an individual field component (such as E_x), or the field intensity ($|\mathbf{E}|^2$, $|\mathbf{B}|^2$). In pyGDM, such a plot can be drawn using

visu.vectorfield_color
(function)

By default, the electric field intensity is plotted, as shown in Fig. 8f-g. Alternatively, to easily plot the field inside the nanoparticle (see Fig. 8c), the same type of color-plots can be generated by calling

visu.vectorfield_color_by_fieldindex
(function)

The above functions are actually plotting scalar fields, the function names "vectorfield…" refer to the fact that vectorial data is taken as input. If the data is available as scalar field (i.e. in tuples (x, y, z, S) with S being a scalar value), one can use the following wrapper to **visu.vectorfield_color**

visu.scalarfield
(function)

6.1.5. Farfield backfocal plane image

Plot the "backfocal plane" image scattered to the farfield from the results obtained by **linear.farfield** (see Section 5.1.3)

visu.farfield_pattern_2D
(function)

An example illustrating the output of the farfield plotting function is shown in Fig. 13.

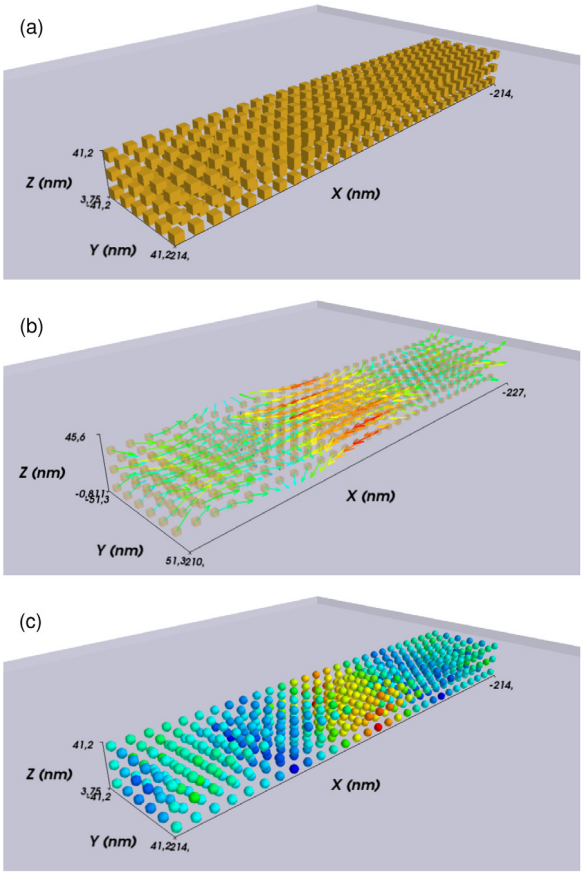

Fig. 9. Examples illustrating the pyGDM 3D visualization tools on the same data as shown in Fig. 8a–c. (a) Structure geometry, (b) electric field (real part) and (c) intensity of the electric field $|\mathbf{E}|^2$ inside the gold nanorod.

6.1.6. Animate fields

The time-dependence of time-harmonic fields is expressed by harmonic oscillations at the fixed frequency ω. After Eq. (19) we can directly calculate the time-dependent field $\tilde{\mathbf{E}}(\mathbf{r}, \omega, t)$ at time t from the complex fields $\mathbf{E}(\mathbf{r}, \omega)$ obtained by the GDM. pyGDM provides a function for simple animations of the electromagnetic fields, which allows to visualize the time-dependent optical response of nanostructures.

visu.animate_vectorfield
(function)

Quiver-plots of the field vectors, the real/imaginary part of individual field components or the field intensity (as color-plots for the latter two) may be animated.

6.2. 3D visualization tools

Similar tools as for two-dimensional data visualization are available in the **visu3d** module for generating 3D figures. The convention for the function names is the same as in the 2D visualization module in order to make switching between 2D and 3D representations as easy as possible. Available plotting functions are **structure**, **vectorfield**, **vectorfield_by_fieldindex**, **vectorfield_color**, **vectorfield_color_by_fieldindex** and **scalarfield**. For a short explanation, see the equivalent 2D-plotting functions, described above. Examples demonstrating the visual output of the 3D-plotting functions are shown in Fig. 9 (on the same data as in Fig. 8a–c).

Finally, also 3D-animations of the time-harmonic fields can be generated. This can be done using **visu3d.animate_vectorfield**.

7. Tools

Apart from visualization, pyGDM includes also several tools to render post-processing as simple as possible.

7.1. 2D-projections of nano-structures

In order to calculate a two-dimensional projection of a nano-structure, use

tools.get_geometry_2d_projection
(function)

7.2. Geometric cross-section

The geometric cross-section of a nano-structure is the area occupied by its projection onto a specific plane (i.e. its "footprint"). It is often used as a reference value, for example for the scattering efficiency. It can be calculated using (in units of nm^2)

tools.get_geometric_cross_section
(function)

By default, the projection on the *XY* plane is used, this can be changed via the parameter "projection".

7.3. Surface of a nano-structure

For surface-effects like surface second-harmonic generation (surface SHG), the meshpoints on the surface of a nanostructure are of particular interest. They can be obtained using

tools.get_surface_meshpoints
(function)

The function also returns the surface-normal unit vectors for each surface-meshpoint.

7.4. Calculating spectra

Calculating spectra of different physical quantities is a very common task in nano-optics. pyGDM therefore provides tools to render this task very simple. Each field-configuration in a **simulation**-object, which is available for several wavelengths, can be obtained via

tools.get_possible_field_params_spectra
(function)

These configurations can then be used together with post-processing routines (such as **linear.extinct** for the extinction cross-section) to calculate a spectrum for some physical quantity. This can be done using

tools.calculate_spectrum
(function)

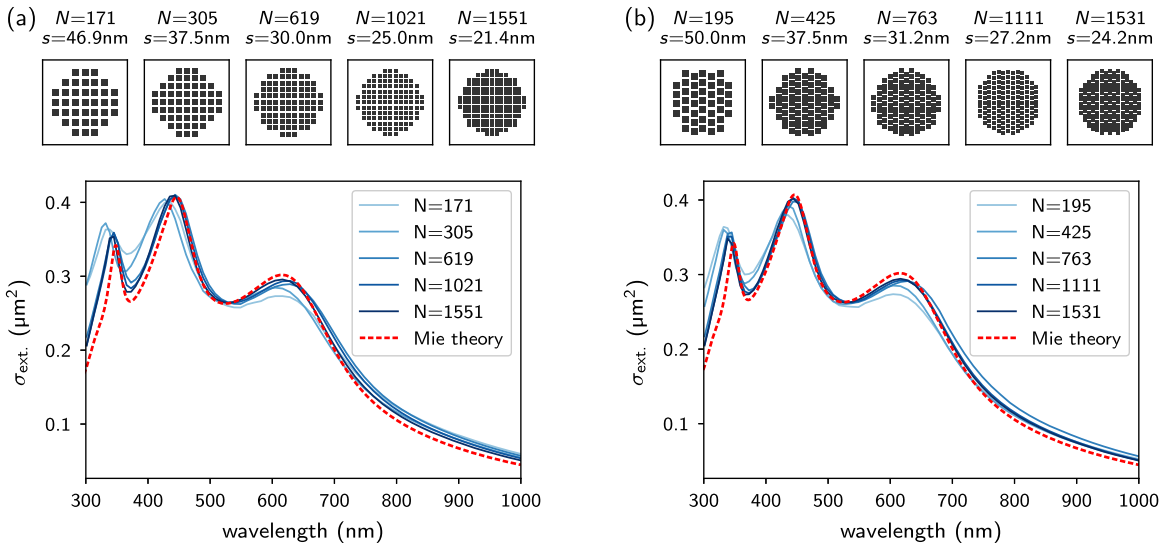

Fig. 10. Comparison of the extinction cross-section of a dielectric sphere ($n_{\text{sphere}} = 2.0$) of diameter $D = 300$ nm, placed in vacuum and illuminated by a linearly polarized plane wave. Calculated either using pyGDM with different numbers of meshpoints (blue lines) or Mie theory (dashed red line). (a) Cubic mesh, (b) hexagonal compact mesh. At the top the number of meshpoints N, the nominal stepsize s and an illustration of the discretization are given, the latter showing XY-slices through the sphere's center.

7.5. Calculating raster-scans

Because pyGDM uses the concept of a generalized propagator, it is particularly suited for application in monochromatic problems with varying illumination conditions, such as raster-scan simulations (varying beam position). If a simulation with a large number of focused beam-positions has been performed, the available incident field configurations (e.g. wavelengths or polarizations) corresponding to full raster-scan maps can be obtained using

tools.get_possible_field_params_rasterscan
(function)

Like in the case of a spectrum, a scalar mapping can be computed from a raster-scan simulation, where each raster-scan position will be attributed a value, according to an evaluation function (like **linear.extinct**, **linear.heat** , ...). Such maps can be obtained using

tools.calculate_rasterscan
(function)

8. Examples

In the following section, we show several examples of pyGDM simulations. In the examples we try to reproduce analytical Mie theory, results from selected publications or we simply intend to demonstrate pyGDM features.

8.1. Comparison to Mie theory

Curved surfaces are generally demanding if it comes to discretization. A popular benchmark problem for electro-dynamical numerical methods is therefore the sphere, for which an analytical solution is given by Mie theory. In the first examples we thus compare pyGDM simulations to Mie theory.

8.1.1. Dielectric nano-sphere

Using **linear.extinct**, we calculate the extinction cross-section σ_{scat} of a dielectric sphere of diameter $D = 300$ nm in vacuum with fixed, purely real refractive index $n = 2$. Results are shown in Fig. 10 for different stepsizes and for (a) a cubic mesh as well as (b) a hexagonal compact lattice.

In comparison with Mie theory, we find that the GDM offers a very good approximation already using rather coarse meshing. Furthermore we note, that the case of a spherical particle seems to be better described using a hexagonal mesh: The agreement with the analytical solution is slightly better for comparable numbers of meshpoints.

8.1.2. Dispersive nano-spheres (Au, Si)

In Fig. 11 we compare spherical particles of dispersive materials. Fig. 11a shows spectra corresponding to a $D = 50$ nm gold nanosphere in vacuum, Fig. 11b gives spectra for a $D = 150$ nm silicon sphere. Simulated spectra are calculated using **linear.extinct** and compared to Mie theory. The resonance positions from Mie theory are reproduced with excellent agreement.

8.2. Other examples

8.2.1. Forward/backward scattering spectra

The far-field propagation routine **linear.farfield** can be used to calculate directionality resolved scattering spectra. This can be done by integrating the intensity in the far-field over limited solid angles. In Fig. 12, the example of a Si sphere with diameter $D = 150$ nm is used again. This time, we calculate the scattering via the **linear.farfield** routine (instead of using **linear.extinct**). The forward (FW) and backward (BW) scattering spectra, as well as the FW/BW ratio are in excellent agreement with the results of Ref. [56].

8.2.2. Far-field radiation pattern

The function **linear.farfield** can also be used to obtain the far-field intensity distribution, comparable to experimental backfocal plane images. In an attempt to reproduce results published in Ref. [57], we put a dipolar emitter ($\lambda_0 = 1\,\mu$m) in the center of a gold split-ring resonator and calculate the scattering to the far-field of the coupled system (for simplicity we consider vacuum as environment). The geometry of the considered arrangement is depicted in Fig. 13a. The dipole is oriented either along X (red) or along Y (blue), the corresponding radiation patterns are shown in Fig. 13b and c, respectively. We can indeed reproduce the dipole-orientation dependent directionality of the scattering from the coupled system.

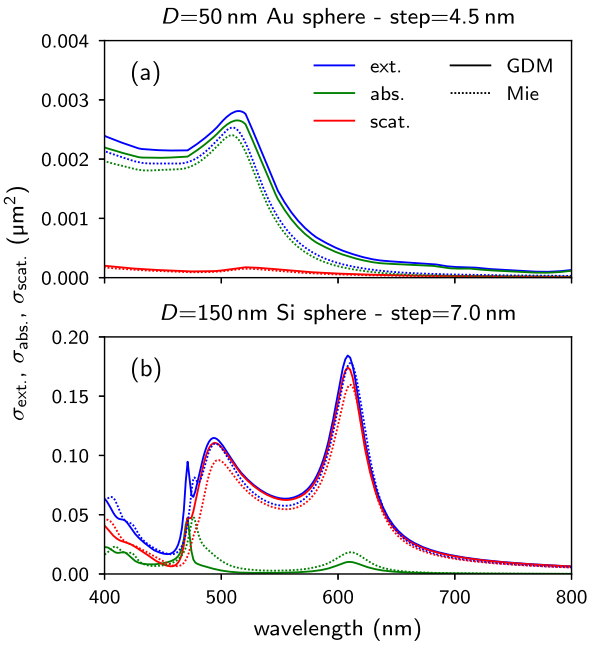

Fig. 11. Comparison of the extinction, absorption and scattering cross-sections of (a) a gold sphere of diameter $D = 50$ nm and (b) a silicon sphere of diameter $D = 150$ nm. Both spheres are placed in vacuum and illuminated by a linearly polarized plane wave. Calculated either using pyGDM (solid lines) or by Mie theory (dotted lines). In both cases, a hexagonal compact mesh is used.

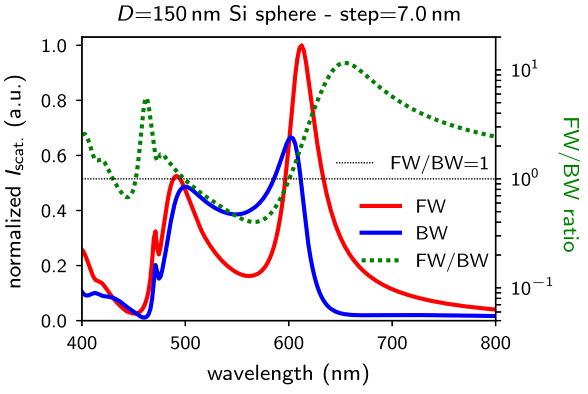

Fig. 12. Forward (FW, red) and backward (BW, blue) scattering spectra and FW/BW ratio (green dotted) for a silicon sphere of diameter $D = 150$ nm in vacuum. A hexagonal compact mesh is used. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

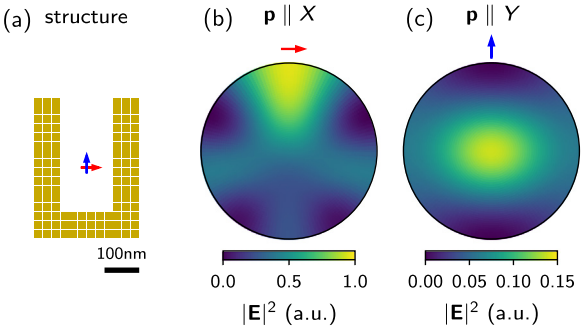

Fig. 13. (a) Sketch of the simulation geometry: A dipolar emitter, radiating at $\lambda = 1$ μm is placed in the center of a gold split-ring resonator (in vacuum). (b–c) Qualitative far-field patterns (backfocal plane images) of the scattering of the quantum emitter coupled to the plasmonic structure for dipole orientations along X and Y in (b), respectively (c).

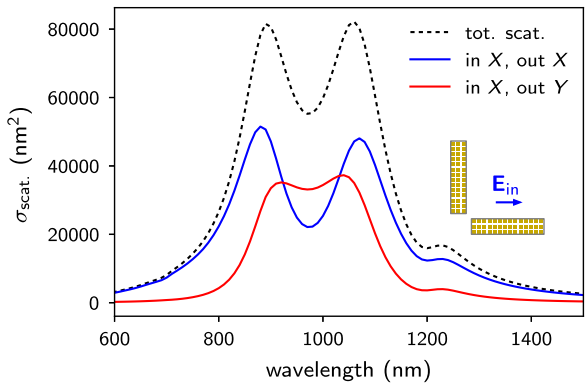

Fig. 14. Polarization filtered scattering spectra from symmetric L-shaped plasmonic antenna (arm length $L = 210$ nm, width and height $W = H = 45$ nm). A sketch of the geometry is shown as inset. The total scattering (dashed black line) as well as the X and Y polarization filtered scattering contributions (blue and red, respectively) is shown. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

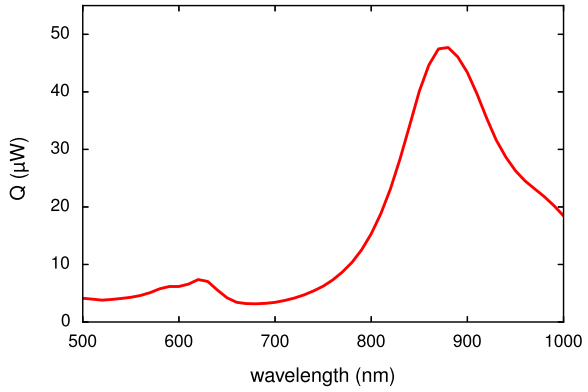

Fig. 15. Spectrally resolved heat generation within a gold prism of side length $L = 115$ nm and height $H = 12$ nm. Incident polarization along one edge of prism.

8.2.3. Polarization conversion

Also the polarization of the scattered light can be analyzed using **linear.farfield**. In Fig. 14 we demonstrate polarization conversion from an L-shaped gold antenna with perpendicular arms of equal dimensions (cf. Refs. [6,7]). An L-shaped plasmonic antenna (in vacuum) with arm dimensions $L = 210$ nm, $W = H = 45$ nm (see inset in Fig. 14) is illuminated by a plane wave of linear polarization along one antenna arm (here along X). The scattered intensity is shown for two different output polarizations in blue ($\mathbf{E}_{\text{scat}} \parallel X$) and red ($\mathbf{E}_{\text{scat}} \parallel Y$), the latter corresponding to a polarization converted scattered field, which is highest if the incident wavelength is spectrally in-between the pure modes (the pure modes correspond to polarization angles of ±45°, see e.g. Ref. [6]).

8.2.4. Heat generation

To demonstrate the capabilities to model nano-optical thermal effects in pyGDM, we reproduce results published in Ref. [52]. A gold prism of side length $L = 115$ nm and height $H = 12$ nm is illuminated by a plane wave, linearly polarized along a side of the prism. The prism is placed on a glass substrate ($n_{\text{subst}} = 1.45$) and is surrounded by water ($n_{\text{env}} = 1.33$). The total deposited heat Q from an incident power density of 1 mW/μm² is shown in Fig. 15 as function of the wavelength.

8.2.5. Decay rate of dipole transition

The modification of the decay rate of an electric and a magnetic dipolar transition close to a very small dielectric nano-particle is

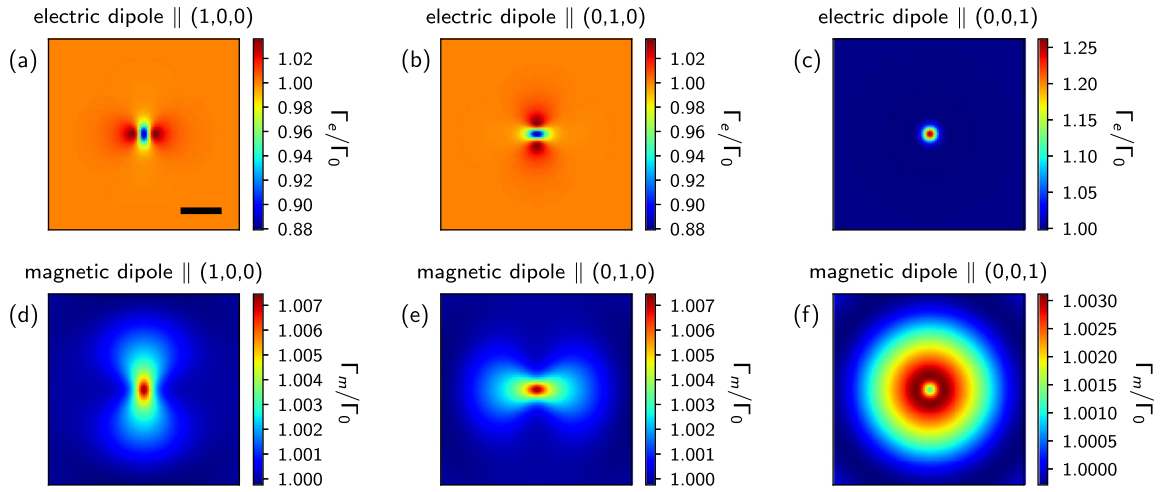

Fig. 16. Decay rate of an electric (a–c) and a magnetic (d–f) dipole transition close to a small dielectric nano-cube ($n = 2$, side length 21 nm, in vacuum) relative to their respective vacuum decay rate Γ_0. The dipoles emit at $\lambda_0 = 500$ nm, are scanned in a 500×500 nm^2 large plane 15 nm above the particle. Dipole orientations along $0X$ (a, d), $0Y$ (b, e) and $0Z$ (c, f). Scale bar is 100 nm.

demonstrated in Fig. 16 (compare also with Ref. [43]). The dipolar emitter ($\lambda_0 = 500$ nm) is raster-scanned in the XY plane at $\Delta z = 15$ nm above a dielectric nano-cube ($n = 2$) of side-length $D = 21$ nm, placed in vacuum. At each position in the raster-scan, the relative decay rate modification with respect to the vacuum value Γ_0 is calculated.

A noteworthy observation is the much narrower confinement of the features in the case of an electric dipole compared to the magnetic transition. Furthermore, also the magnitude of the decay rate variation is much stronger for the electric dipole. Both phenomena can be attributed to the more "direct" interaction of an electric dipole with the nano-structure, compared to the "indirect" magnetic response of the itself *non-magnetic* nano-particle (see also Section 4.2).

8.2.6. Rasterscan simulation: TPL/heat/temperature

To demonstrate a rasterscan simulation, we calculate as function of a focused beam's focal spot position and for several incident linear polarizations: the two-photon photoluminescence (TPL) signal, the total deposited heat Q and the temperature rise at 150 nm above the center of a flat gold rhombus. The object is assumed to lie in water with $n_{\text{water}} = 1.33$ and a thermal conductivity of $\kappa_{\text{water}} = 0.6$ W/mK. The temperature rise is calculated either for the rhombus in a homogeneous water environment, or in water lying on a glass substrate (using $n_{\text{glass}} = 1.5$, $\kappa_{\text{glass}} = 0.8$ W/mK). The rhombus dimensions are defined by a side length of $L = 500$ nm, a height of $H = 20$ nm and a top (and bottom) angle of 60°. A linearly polarized focused plane wave ($\lambda_0 = 750$ nm) of spotsize $w = 200$ nm is used, setting the power density to 1 mW/μm^2. The rasterscans consisting of 50×50 focal spot positions are shown in Fig. 17 for different angles of the linear polarization of the fundamental field. We can clearly observe the correlation between TPL and the heat and temperature mappings. We also see that the temperature rise is slightly stronger if a glass substrate is present. This is a result of heat reflection at the glass surface.

8.2.7. Rasterscan simulation: LDOS

In Fig. 18 we show rasterscan simulations of the photonic LDOS above a U-shaped dielectric planar structure. The length is 800 nm in the X-direction, 400 nm in the Y direction, its height is 60 nm and the bar width is 180 nm. Fig. 18 (a) shows the partial LDOS for X-oriented dipole emitters, (b) the case of Y orientation and (c) the total LDOS in the structure plane. From left to right is shown the LDOS at decreasing distance to the structure (60 nm, 30 nm and 0 nm to the top surface). In the very right column the surface LDOS is calculated using the "TPL"-method (using a spotsize of $w = 100$ nm, see also Section 5.2). Comparing the LDOS at the top surface layer with the TPL-method, the general trends are reproduced. The differences are not very surprising, since the calculated quantities are not exactly the same. The TPL method gives a measure of the energy that can be coupled into the structure at the respective surface position using a focused beam. It is therefore only non-zero if the focused beam intersects with the structure. The LDOS corresponds to the efficiency of the radiative coupling between a dipolar emitter and the structure and is non-zero also outside the structure.

9. Evolutionary optimization of nanostructure geometries

9.1. Evolutionary optimization

A peculiarity of pyGDM is the **EO** module, provided together with the main pyGDM toolkit. The purpose of the **EO** module is to find nanostructure geometries that perform a certain optical functionality in the best possible way. This is also known as the "inverse problem" [58,59]. We try to achieve this goal by formulating the optical property as an optimization problem which takes the geometry of the particle as input. Such problem will usually result in a complex, non-analytical function and hence cannot be solved by classical optimization methods like variants of the "Newton–Raphson method".

In our approach we therefore optimize the problem using evolutionary optimization (EO) algorithms. The latter mimic natural selection to find ideal solutions to complex (often non-analytical) problems. The initial step is to define a "population" of random parameter-sets for the problem. These "individuals" are then evolved through a cycle of "reproduction" (mixing parameters between the individuals and application of random changes) and "selection" (problem evaluation and discarding weak solutions). After a sufficient number of iterations, hopefully an optimum parameter-set for the problem has been found. The evolutionary optimization cycle is depicted in Fig. 19a. Unfortunately, convergence can in principle never be guaranteed in EO. Convergence is therefore probably the most critical point in evolutionary optimization strategies. To ensure the credibility of the optimization results, a good stop-criterion and/or careful testing of the convergence and reproducibility of the solution for different initializations are crucial. For details on EO, we refer to the related literature, e.g. Ref. [60].

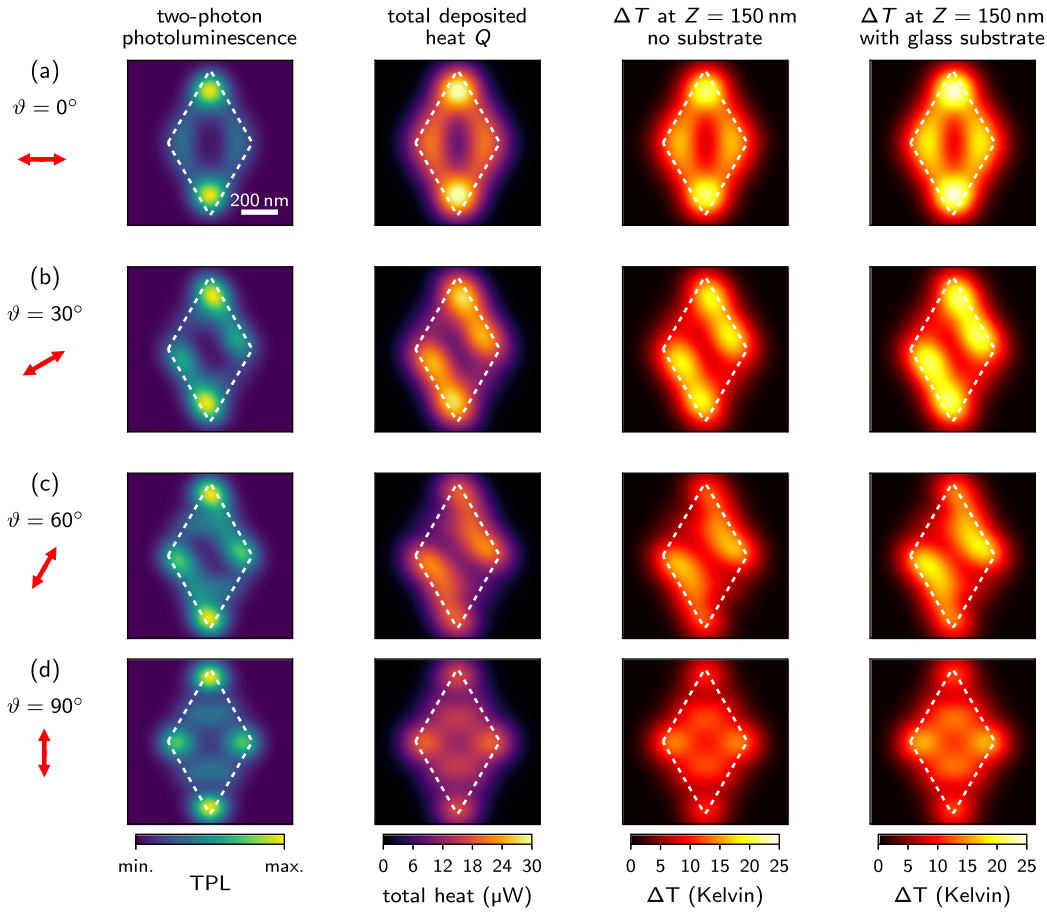

Fig. 17. Thermoplasmonic rasterscan simulations. From left to right: TPL; total deposited heat; temperature rise 150 nm above the center of a gold rhombus (side length 500 nm, top angle 60°) as function of the focal spot position of the incident beam. For the first three columns, the rhombus lies in homogeneous water. The right column shows the temperature rise for the structure in water, but lying on a glass substrate (used heat conductivities are $\kappa_{\text{water}} = 0.6$, $\kappa_{\text{glass}} = 0.8$ W/mK). The incident wavelength is $\lambda_0 = 750$ nm, the linear polarization angle is (a) $\vartheta = 0°$, (b) $\vartheta = 30°$, (c) $\vartheta = 60°$ and (d) $\vartheta = 90°$. Scalebar in (a) is 200 nm, the position of the gold rhombus is indicated by a white dashed contour line (plotted using **visu.structure_contour**).

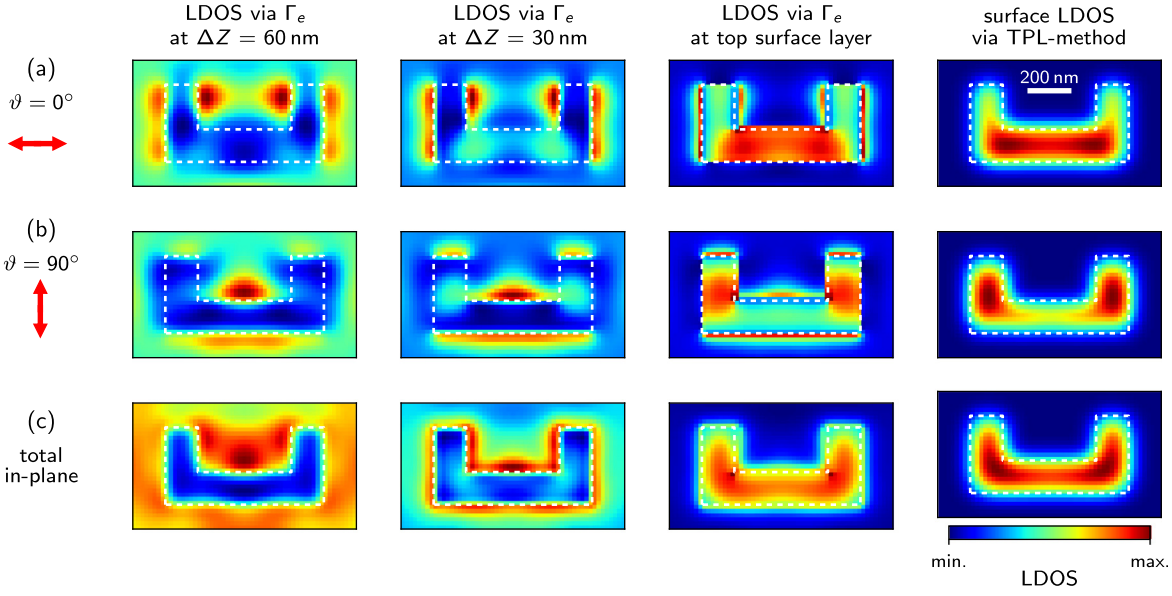

Fig. 18. LDOS rasterscan simulations: Partial LDOS above a planar U-shaped dielectric structure ($n = 2.0$, $H = 60$ nm), calculated using the imaginary part of the field susceptibility (via the decay-rate of a dipolar emitter) or a raster-scanned focused beam ("TPL-method"). From left to right: At $\Delta Z = 60$ nm and $\Delta Z = 30$ nm above the structure surface, at the height of the top-most mesh-point layer inside the structure and using the focused-beam approximation technique (see section "TPL"). The wavelength is $\lambda_0 = 600$ nm, the partial LDOS is shown for (a) $\vartheta = 0°$ (X-direction), (b) $\vartheta = 90°$ (Y-direction) and (c) the total LDOS in the structure plane. The scale bar is 200 nm, the position of the structure is indicated by white dashed contour lines (plotted using **visu.structure_contour**).

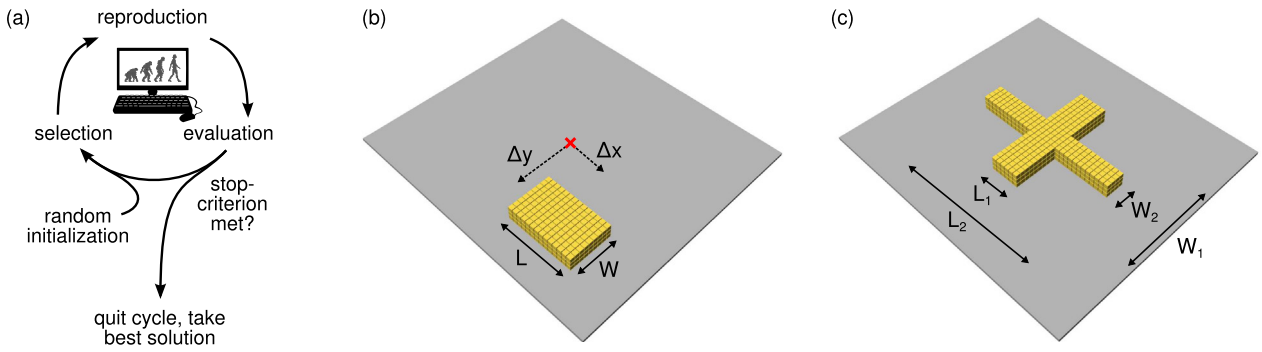

Fig. 19. (a) Illustration of the evolutionary optimization scheme. (b–c) Sketches of the gold antenna geometry models used for the evolutionary optimization examples. Free parameters for the rectangular geometry (b) are the length L and width W of the rectangle and an offset $(\Delta x, \Delta y)$ for the structure position with respect to the origin (indicated by a small red cross). Free parameters for the cross-like geometry (c) are the lengths L_1, L_2 and widths W_1, W_2 of the two rectangular components, forming the cross.

9.2. EO in pyGDM

pyGDM can be used to optimize particle geometries for nano-optical problems via evolutionary optimization. Our approach consists of three main ingredients:

1. The **structure-model**: Constructs a particle geometry as function of a set of input-parameters which will be the free parameters for the optimization algorithm. It furthermore contains the simulation setup (via an instance of **core.simulation**). It is defined by a class inherited from

   EO.models.BaseModel
   (class)

2. The **problem**: Defines the optimization target. This will usually be an optical property of the particle such as the scattering cross-section or its near-field enhancement. It is defined by a class inherited from

   EO.problems.BaseProblem
   (class)

3. The **EO algorithm**: Finally the algorithm to solve the optimization problem

For (3.) we use the PyGMO/paGMO toolkit [61]. PyGMO not only offers a large spectrum of EO algorithms. It can furthermore distribute the population of solutions on several "islands" within the so-called "Generalized island model". This allows for a very easy scaling of the evolutionary optimization on multi-processor architectures [62].

Note: In this context, the use of the generalized propagator in pyGDM is a clear asset. Optimization problems with the incident field shape and/or polarization as variable and a fixed structure geometry can be solved very efficiently. This includes problems like near-field shaping in adaptive optics [63].

9.3. Multi-objective optimization

pyGDM's **EO**-module is also capable of treating multi-objective optimization problems, by internally addressing pyGMO's according API. In other words, it is possible to search for nano-structure geometries optimizing *multiple* target properties simultaneously.

Such evolutionary multi-objective optimization (EMO) can in principle be done in two ways: The first approach consists in summarizing the multiple target values in one single fitness function, hence capturing the problem into a single objective optimization. In this case, the critical part is the construction of an appropriate fitness-value, which is usually not trivial at all. In the second approach, one searches for the set of "non-dominated" or "Pareto optimum" solutions, which is often called the "Pareto front". It consists of solutions that are all optimal in the sense that an improvement in one of the target functions necessarily leads to a decrease in at least one of the other optimization targets. The obvious advantage is, that the individual objectives can be used *as-is* without the need to fiddle them into a single fitness-function. On the other hand, the latter approach additionally requires the selection of a single optimum solution from the set of Pareto optimum solutions. For a detailed introduction to EMO, see e.g. Ref. [64].

9.4. EO-examples

We demonstrate the evolutionary optimization toolkit "**EO**" on some simple but illustrative problems.

9.4.1. Maximize scattering cross-section or scattering efficiency

For a first demonstration, we want to optimize the shape of a rectangular gold-antenna in order to obtain maximum scattering at a certain wavelength and for a fixed angle of the linear incident polarization. The free parameters are the length L and width W of the plasmonic rectangle (see Fig. 19b). The position of the rectangle ($\Delta x = \Delta y = 0$), the stepsize ($s = 15$ nm) of the cubic mesh and the height of the antenna ($H = 45$ nm) are fixed.

We run the EO of the rectangular shape with different optimization targets, the respective final best solutions are shown in Fig. 20: In (a) the scattering efficiency Q_{scat} (i.e. the scattering cross-section σ_{scat} divided by the geometrical cross-section σ_{geo}) is maximized for an incident plane wave with wavelength $\lambda_0 = 800$ nm and linear polarization of $\mathbf{E}_0$ along X. In (b) maximum Q_{scat} is searched for $\lambda_0 = 1000$ nm and $\mathbf{E}_0 \parallel X$. In (c) Q_{scat} is again maximized for $\lambda_0 = 1000$ nm but a perpendicular polarization angle, hence $\mathbf{E}_0 \parallel Y$. Finally, (d) shows and optimization of the scattering cross-section σ_{scat} (instead of Q_{scat}), with otherwise equal configuration as in (c).

The first observation is, that the optimization is indeed capable of adjusting the size of the antenna such that the surface plasmon resonance occurs at the target wavelength. We also observe that while the optimization of the scattering efficiency (Q_{scat} given at the right of each plot) leads to thin rectangles with low geometric cross section, the optimization of the scattering cross section (σ_{scat} given at the left of each plot) leads to a structure of maximum

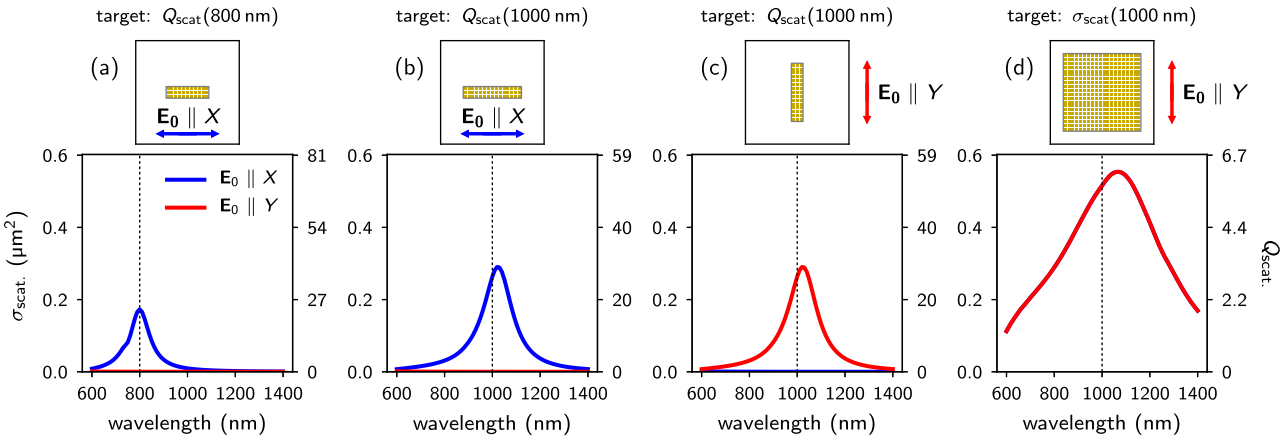

Fig. 20. Evolutionary optimization of the dimensions of a rectangular plasmonic gold antenna for different optimization targets. The optimum geometry is shown in the top panels (showing 400 × 400 nm^2 large areas). The corresponding scattering spectra are shown in the bottom panels for plane wave illumination with X and Y linear polarization (blue, respectively red lines). The left ticks in each spectrum denote the scattering cross section σ_{scat}, the ticks on the right hand side give the corresponding scattering efficiency ($Q_{scat} = \sigma_{scat}/\sigma_{geo}$). (a) Maximization of Q_{scat} for X polarized illumination at $\lambda = 800$ nm. (b) Maximization of Q_{scat} for X polarized illumination at $\lambda = 1000$ nm. (c) Maximization of Q_{scat} for Y polarized illumination at $\lambda = 1000$ nm. (c) Maximization of σ_{scat} for Y polarized illumination at $\lambda = 1000$ nm. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

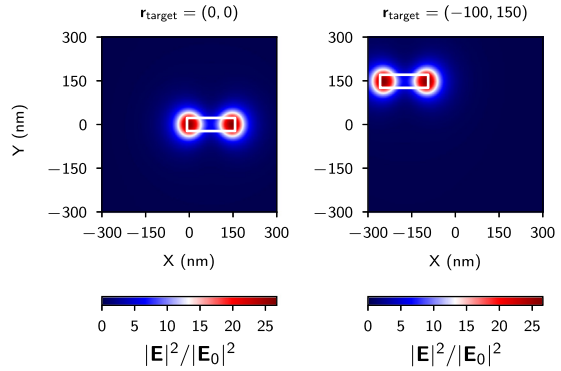

Fig. 21. Evolutionary optimization of the dimensions and the position of a rectangular plasmonic gold antenna for maximum **E**-field intensity enhancement at the location (a) $\mathbf{r}_{target} = (0, 0)$ [nm] and (b) $\mathbf{r}_{target} = (-150, 100)$ [nm]. The z-coordinate of $\mathbf{r}_{target}$ is fixed to 30 nm above the upper surface of the structure. Shown areas are 600 × 600 nm^2.

allowed dimensions. This leads to a cross-section σ_{scat} about twice as large as for the other antennas. The scattering efficiency Q_{scat} on the other hand is significantly lower (about a factor 5) compared to the optimizations shown in Fig. 20a–c.

9.4.2. *Maximize electric field intensity*

In a second example we want to find a structure to maximize the electric field intensity at a specific point $\mathbf{r}_{target}$, 30 nm above the structure surface.

As structure to be optimized, we use again the rectangular gold-antenna of variable length and width with the same configuration as in the first example. Additionally we introduce as free parameters the offsets Δx and Δy, shifting the rectangle with respect to the origin of the coordinate system (see Fig. 19b). The structure is illuminated by a plane wave ($\lambda_0 = 800$ nm), linearly polarized along X.

We run the optimization for two different $\mathbf{r}_{target}$. The results are shown in Fig. 21. In both runs, the optimization found a plasmonic dipole antenna, resonant at the incident wavelength and shifted its position such that the hot-spot of maximum field enhancement lies at $\mathbf{r}_{target}$.

9.4.3. *EMO: Double resonant plasmonic antenna*

In a last example, we show how multiple objectives can be optimized concurrently in a single optimization, by calculating the Pareto-front. For the demonstration, we try to obtain structures that scatter light at two different wavelengths for perpendicular polarization angles of the incident plane wave. We choose a simple cross-like geometry model (structure placed in vacuum), consisting of the four free parameters L_1, L_2 and W_1, W_2 (see Fig. 19c).

The optimization goal is to simultaneously maximize Q_{scat} for (1) a wavelength $\lambda_0 = 800$ nm and an incident polarization along X and (2) $\lambda_0 = 1200$ nm and an incident polarization along Y. The Pareto-front obtained by the evolutionary optimization is shown in Fig. 22. Scattering spectra of selected structures are shown in the top, their geometries are illustrated in the insets, labeled (1)–(3).

The optimization indeed found structures which maximize either one of the scattering-targets ((1) and (3)), or scatter light similarly strong for both target configurations (structure (2)). Note that the Pareto-front is not very smooth. In addition, the model seems not to be sufficiently sophisticated in order to obtain structures with equal Q_{scat} for both target conditions. A more general structure model could probably provide better solutions for the problem.

Note. Further nano-photonic EO problems tackled using the pyGDM toolkit, can be found in Refs. [43,19,65–67].

10. Conclusion

In conclusion, we presented a python toolkit for electrodynamical simulations in nano-optics, based on a volume discretization approach, the Green dyadic method. While other techniques like FEM may offer better accuracy, the main strength of pyGDM is the efficient treatment of large monochromatic problems with many illumination configurations, like raster-scan simulations. Such calculations can be solved very efficiently in pyGDM thanks to the concept of the generalized propagator. Furthermore, its simplicity is a great advantage of pyGDM. The high-level python API as well as many tools for rapid data analysis and visualization renders standard nano-optical simulations very easy. Finally, the evolutionary optimization submodule is a unique feature, which allows to optimize nanostructure geometries for specific target optical properties. Scripts to reproduce all above

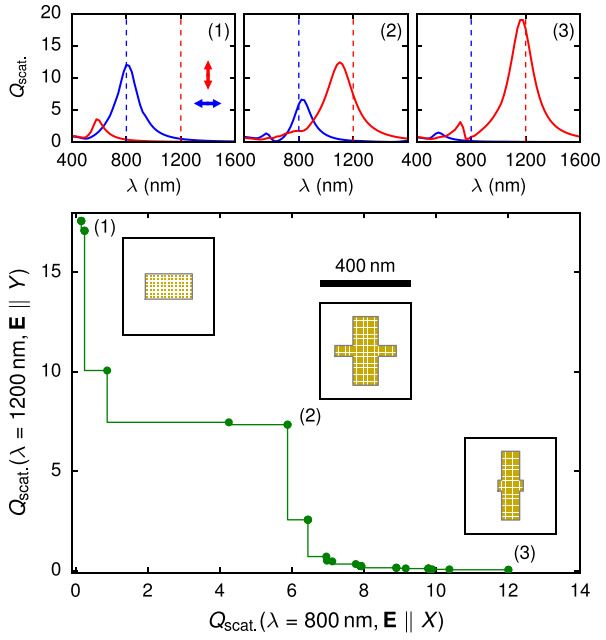

Fig. 22. Multi-objective optimization of double-resonant plasmonic antennas made from gold. Large plot: Pareto front found from a concurrent maximization of Q_{scat} at $\lambda_1 = 800$ nm and $\lambda_2 = 1200$ nm, for polarizations along X and Y, respectively ($Q_{scat} = \sigma_{scat}/\sigma_{geo}$). Top: Spectra for X (blue) and Y-polarized (red) illumination of three selected structures on the Pareto front, shown as insets (labeled by numbers 1–3). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

shown examples, can be found online together with further, extensive documentation.

Acknowledgments

I gratefully thank Christian Girard and Arnaud Arbouet for their advice, help with the theory, careful proof-reading and for the fortran routines. I also want to thank Gérard Colas des Francs for helpful discussions and his contributions to the fortran code to which also Renaud Marty contributed. I finally thank Vincent Paillard and Aurélien Cuche for many inspiring discussions and proof-reading of the manuscript. This work was supported by Programme Investissements d'Avenir under the program ANR-11-IDEX-0002-02, reference ANR-10-LABX-0037-NEXT, and by the computing facility center CALMIP of the University of Toulouse under grant P12167.

Conflicts of interests

The author declares no competing financial interest.

Appendix A. Accuracy, possible system size and limitations

Limitations. The limit for the number N of discretization meshpoints depends mainly on the amount of RAM available in the machine. The memory requirement rises with $3N^2$ (Fig. 6b). The computation time rises even proportional to N^3 (see also Fig. 6a), so at some point, the speed will be limiting as well. This effectively limits the number of meshpoints to $\approx$ 10000–15000.

Accuracy, large systems. To yield a reasonable accuracy, the discretization stepsize has to be sufficiently small (in the order of $\approx$ 10 nm for plasmonics and dielectrics of refractive index $n \lesssim 3$). For dielectrics with higher refractive index, the discretization should be further refined to yield accurate results. However, if the user is aware of the fact that the agreement will be only qualitative, approximative simulations are possible with larger discretization. In consequence, the memory requirement limits the applicability of pyGDM to the amount of material that can be simulated with good accuracy to (very rough estimation) $\approx 10000 \times 10^3$ nm^3).

Small systems. When the size of the system is reduced, the discretization can be finer within the limit of the feasible 10–15k meshpoints. Hence, the accuracy becomes better. One must only be aware, that pyGDM is a purely classical Maxwell solver. Therefore, the size of the system should not be reduced down to scales where quantum effects would occur (usually $\lesssim$ 1–2 nm).

Appendix B. Technical details

B.1. Reference system in pyGDM simulations

In pyGDM an asymptotic Green's dyad is used which describes not only a substrate ("layer 1", refractive index n_1), but also an additional cladding layer at a variable height above the substrate ("layer 3", n_3). The nanoparticle is placed in the sandwich layer ("layer 2", n_2), i.e. in-between layers 1 and 3. The distance between the substrate and the cladding layer can be specified by a *spacing* parameter. By default, $n_3 = n_2$, so if the index of the cladding layer "3" is not specified in the constructor of **structures.struct**, a reference system composed of a homogeneous environment above a dielectric substrate is assumed. To run a simulation without a substrate it is sufficient to simply set $n_1 = n_2$. The geometry of the pyGDM reference system is illustrated in Fig. 2.

The non-retarded Dyad used in pyGDM can be derived using the image charges method and gives good approximations for dielectric interfaces of low refractive index. A fully retarded Dyad for the 3-layer environment can also be calculated, which becomes necessary for instance at metallic interfaces [17,68]. This might be implemented in future versions of pyGDM.

B.2. Structure geometry

The structure geometry in pyGDM is defined as a list of (x, y, z) tuples, defining the positions of the meshpoints on either a cubic, or a hexagonal compact, regular grid.

pyGDM comes with some generators for common structures in nano-photonics. These geometries are available in the **structures** submodule. An overview of the available structures is shown in Fig. 23. Additional structures can easily be implemented at the example of the available generator functions. We suggest using the mesher-routines **structures._meshCubic** and **structures._meshHexagonalCompact**.

Additionally, planar structures can be generated from the brightness contrast of an image file, using

structures.image_to_struct
(function)

This may be used to create structures from a lithography-mask layout or also from scanning electron- or atomic force-microscopy images, to simulate "real" geometries from an experimental sample.

Finally, structure-geometries can be manipulated (rotated, "center of gravity" shifted to the origin) using

structures.rotate_XY
(function)

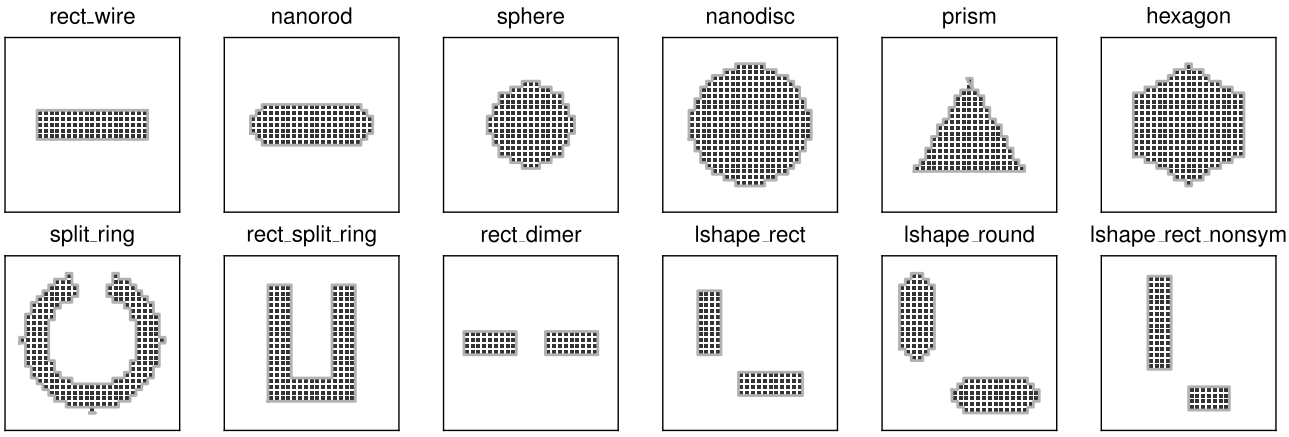

Fig. 23. Top view of some geometries available in the **structures** submodule. The corresponding generator function names are given on top of the example plots.

structures.center_struct
(function)

B.3. Material dispersion

pyGDM provides some basic dispersion models in its **materials** submodule:

materials.dummy generates a material object which returns a constant dielectric function. "**silicon**", "**gold**" and "**alu**" provide the commonly used dispersion data for the respective materials.

However, usually one would use tabulated data for the dispersion. This can be done in pyGDM via

materials.fromFile
(class)

All dispersion containers "**materials.class**" provide an **epsilon(wavelength)** attribute, which is a function that returns the (complex) permittivity at *wavelength* (in nm).

By default, the tabulated data is interpolated linearly using *numpy*'s "*interp*". Optionally, higher order spline interpolation is supported (based on *scipy.interpolate.interp1d*). Note that the latter may cause problems with python's "pickling" technique, particularly in combination with the **EO** module.

B.4. Minimum working example script

```
from pyGDM2 import structures
from pyGDM2 import materials
from pyGDM2 import fields
from pyGDM2 import core
from pyGDM2 import visu

## --- simulation setup ---
## structure: sphere of 120nm radius,
## constant dielectric function (n=2),
## placed in vacuum
step = 20    # nm
geometry = structures.sphere(step, R=6, mesh='cube')
material = materials.dummy(2.0)
norm = structures.get_normalization(mesh='cube')
n1 = n2 = 1.0

struct = structures.struct(step, geometry, material,
                                        n1,n2, norm)

## incident field: plane wave, 500nm, lin. pol. || x
field_generator = fields.planewave
wavelengths = [500]   # nm
kwargs = dict(theta=[0.0], kSign=[-1])
efield = fields.efield(field_generator,
                wavelengths=wavelengths, kwargs=kwargs)

## create simulation object
sim = core.simulation(struct, efield)


## --- run the simulation ---
core.scatter(sim)


## --- plot the near-field inside the sphere ---
## using first (of one) field-config (=index 0)
visu.vectorfield_by_fieldindex(sim, 0, projection='XY')
visu.vectorfield_by_fieldindex(sim, 0, projection='XZ')
visu.vectorfield_by_fieldindex(sim, 0, projection='YZ')
```

Listing 1: Minimum example script. The plots generated by the script are shown in Fig. 24.

B.5. Further tools available in pyGDM

B.5.1. Save and load simulations

To save and reload pyGDM simulations, the following functions are available. Saving and loading relies on python's "pickle" technique:

tools.save_simulation
(function)

tools.load_simulation
(function)

B.5.2. Show information about simulations

To print detailed information about a pyGDM simulation, the following function can be used

tools.print_sim_info
(function)

alternatively, simply use "**print sim_object**".

B.5.3. Generate coordinate list for 2D map

To calculate 2D data in pyGDM (e.g. near-field maps, cf. Fig. 8d-g), we provide a tool to easily generate the 2D grid (in Cartesian 3D space) for such data:

tools.generate_NF_map
(function)

B.5.4. Get index of specific field configuration

pyGDM uses keyword dictionaries to store multiple configurations of the incident field (such as several wavelengths, polarizations, focused beam positions). All possible permutations of the given keywords are stored in the **core.simulation** object and are attributed an index, by which they can unambiguously identified. In order to get the index of the field parameters that closest match specific search values (like a wavelength), one can use:

tools.get_closest_field_index
(function)

All field-configurations available in a simulation, sorted by their *field-index*, can be obtained by

tools.get_field_indices
(function)

B.5.5. Cubic stepsize from discretized structure

If the particle discretization is generated with another program then the pyGDM meshing functions (available in the **structures** submodule), it might be helpful to determine the stepsize of the structure. We provide a function, that computes the stepsize of a cubic mesh by calculating the closest distance between any two meshpoints (using *scipy.spatial.distance.pdist*).

tools.get_step_from_geometry
(function)

B.5.6. Get complex field as list of coordinate/field tuples

After running **core.scatter**, pyGDM stores the fields inside the particle in the **core.simulation** object as lists of the complex field components ($E_{x,i}$, $E_{y,i}$, $E_{z,i}$). The (x_i, y_i, z_i) geometry coordinates are stored separately in the **structures.struct** object within the simulation description object. To generate complete field-lists of tuples (x_i, y_i, z_i, $E_{x,i}$, $E_{y,i}$, $E_{z,i}$), pyGDM provides the following functions:

tools.get_field_as_list
(function)

tools.get_field_as_list_by_fieldindex
(function)

Both return the complex field for a selected illumination configuration as list of coordinate/field tuples (x_i, y_i, z_i, $E_{x,i}$, $E_{y,i}$, $E_{z,i}$), either from the raw field-object or from the **simulation** object and a field-index, respectively.

B.5.7. Generate 2D map from coordinate list

To map spatial data available as list of (coordinate-) tuples onto a plotable 2D grid (e.g. for plotting a mapping with *matplotlib.imshow*), pyGDM provides

tools.map_to_grid_XY
(function)

B.5.8. Raster-scan field configurations

If a simulation with a large number of focused beam-positions has been performed, the available incident field configurations corresponding to full raster-scan maps can be obtained using

tools.get_possible_field_params_rasterscan
(function)

Analogously, the set of indices referring to the fields in the **simulation.E** which correspond to a particularly configured raster-scan, can be obtained via

tools.get_rasterscan_field_indices
(function)

Alternatively, the full set of fields inside the particle for a raster-scan with particular illumination-configuration can be obtained via

tools.get_rasterscan_fields
(function)

B.6. Dependencies

The core functionalities of pyGDM depend only on **numpy**. The compilation of the *fortran* parts require a *fortran* compiler such as *gcc*'s **gfortran**.

*B.6.1. Dependencies: **visu***

All 2D visualization tools require **matplotlib**.

*B.6.2. Dependencies: **visu3D***

All 3D visualization tools require **mayavi**.

*B.6.3. Dependencies: **tools***

Several tools require **scipy**.

*B.6.4. Dependencies: **structures***

Several structure-tools require **scipy**. **image_to_struct** requires **PIL**.

*B.6.5. Dependencies: **core.scatter**: Solver (parameter "method")*

pyGDM includes wrappers to several **scipy** solvers but also other methods are supported. Below is given an exhaustive list of the available solvers and their dependencies. For benchmarks, see Fig. 6.

- "lu" (default) **scipy.linalg.lu_factor** (LU decomposition)
- "numpyinv" **numpy.linalg.inv** (if **numpy** is compiled with LAPACK: LAPACK's "dgesv", else a slower fallback routine)
- "dyson": Own implementation, no requirements (sequence of Dyson's equations [20])
- "scipyinv" **scipy.linalg.inv** (LAPACK's "dgesv")

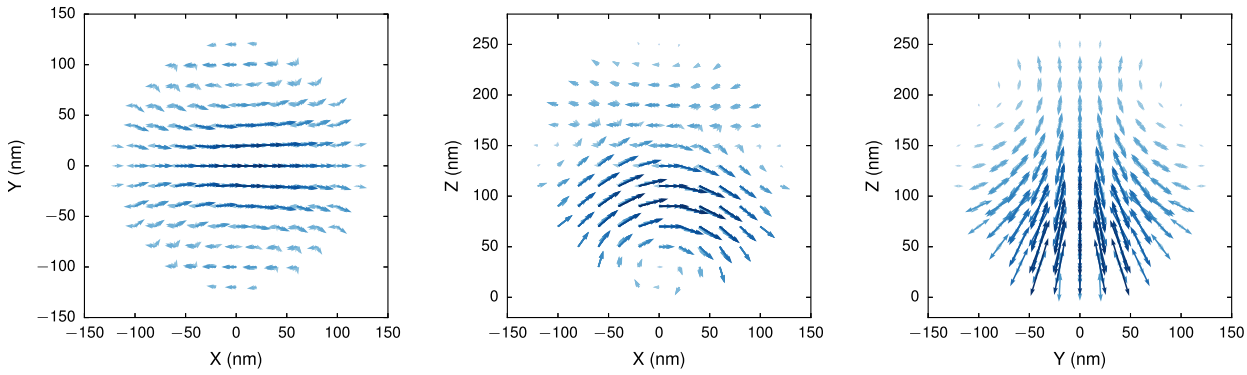

Fig. 24. Plots generated by the demonstration script shown in listing 1. From left to right: *XY*, *XZ* and *YZ* projections of the real part of the electric field inside a dielectric nanosphere ($n = 2$) with radius $R = 120$ nm placed in vacuum. Linear polarized (along X) plane wave illumination with $\lambda = 400$ nm, incident from positive Z ($\mathbf{k} = -\hat{\mathbf{e}}_z k$).

- "pinv2": **scipy.linalg.pinv2** (singular value decomposition, SVD)
- "superlu": **scipy.sparse.linalg.splu** (superLU [69])
- "cg": **scipy** conjugate gradient iterative solver (**scipy.sparse.linalg.bicgstab**), by default preconditioned with **scipy**'s incomplete LU decomposition (superLU [69] via **scipy.sparse.linalg.spilu**)
- "pycg": **pyamg**'s implementation of the *bicgstab* algorithm, optionally preconditioned with **scipy**'s incomplete LU decomposition. Recommended if multi-threading problems are encountered with scipy's *bicgstab* implementation, which is not threadsafe

B.7. Compiling, installation

We provide a script for the easy compilation and installation via python's "distutils" functionality. For this, simply run in the pyGDM root directory

```
python setup.py install
```

Alternatively, pyGDM can be compiled locally without installation via

```
python setup.py build
```

Or it may be installed to a user-defined location using the "`--prefix=...`" option.

Note. The "setup.py" script requires **numpy** as well as a *fortran* compiler (tested with **gfortran**).

B.8. Possible future capabilities

The GDM can be used for manifold further calculations, which are to be included in future versions of pyGDM. A non-exhaustive list of possible future features includes

- 2D structures (assuming infinite length along one coordinate) [70]
- Coherent nonlinear effects like (surface-) second or third harmonic generation [19,71]
- Electron energy loss/gain spectroscopy (EELS, EEGS) or cathodoluminescence (CL) simulations [28,72]
- more environment choices (surface propagator including retardation effects [49,73], multi-layer stratified environments [17,18] or magnetic decay rate calculation including a substrate [49,74])
- cuboidal [23] or non-regular meshes [24]
- materials with anisotropic susceptibility, e.g. birefringent media
- periodic structures [75,76]
- quantum corrected model for plasmonic tunneling currents via junctions of inhomogeneous permittivity [77]
- SNOM image calculation/interpretation [78,79]
- memory-efficient conjugate gradients solver including FFT-accelerated matrix–vector multiplications for large problems [27]

Appendix C. Keyword arguments of the most important classes and functions

Most important classes

For a detailed explanation of the physical information contained by the below classes, see Section 3.

core.simulation
(class)

constructor arguments:

- *struct:* instance of **structures.struct**
- *efield:* instance of **fields.efield**

structures.struct
(class)

constructor arguments:

- *step:* discretization stepsize (in nm)
- *geometry:* list of meshpoint coordinates (x, y, z) (in nm)
- *material:* structure material dispersion, instance of **materials.CLASS**
- *n1, n2:* ref. index of substrate (n1) and environment (n2)
- *normalization (optional):* mesh-type dependent factor, default: "1" (cubic mesh)
- *n3 (optional):* ref. index of cladding
- *spacing (optional):* distance between substrate and cladding. default: "5000" (nm)

fields.efield
(class)

constructor arguments:

- *field_generator:* field generator function (e.g. from **fields** module)
- *wavelengths:* list of wavelengths at which to do the simulation (in nm)
- *kwargs (optional):* dict (or list of dict) with further kwargs for the field generator

Most important functions

For a detailed explanation of the calculations performed by the below functions, see Sections 4–6.

pyGDM core

core.scatter / core.scatter_mpi
(function)

arguments:

- *sim:* instance of **core.simulation**
- *method (optional):* inversion method, default: "lu"
- *multithreaded (optional):* default: "True"

core.decay_rate
(function)

arguments:

- *sim:* instance of **core.simulation**
- *method (optional):* inversion method, default: "lu"

Post-processing

linear.extinct
(function)

arguments:

- *sim*: instance of **core.simulation**
- *field_index*: index of field-configuration

linear.nearfield
(function)

arguments:

- *sim*: instance of **core.simulation**
- *field_index*: index of field-configuration
- *r_probe* : list of (x, y, z) coordinates at which to evaluate the near-field

Visualization

visu.structure
(function)

arguments:

- *sim*: instance of **core.simulation**
- *projection (optional)*: default: "XY"
- *color (optional)*: optional, matplotlib compatible color, default: "auto"
- *scale (optional)*: scaling, default: "0.5"

visu.vectorfield
(function)

arguments:

- *NF*: list containing the complex field (list of 6-tuples $(x_i, y_i, z_i, E_{x,i}, E_{y,i}, E_{z,i})$. See also section 12.5.6: **tools.get_field_as_list**)
- *projection (optional)*: default: "XY"
- *slice_level (optional)*: using only fields at specific height. default: "none" → superpose all vectors

visu.scalarfield
(function)

arguments:

- *NF*: list of 4-tuples containing the coordinates and scalar-field values $((x_i, y_i, z_i, S_i))$

Appendix D. GDM in the SI unit system

In order to facilitate the conversion between SI and CGS unit systems, in this section, we introduce the main GDM equations in SI units.

The Fourier transformed Maxwell equations are then:

$$\nabla \cdot \mathbf{D}(\mathbf{r}, \omega) = \rho(\mathbf{r}, \omega) \tag{48a}$$
$$\nabla \times \mathbf{E}(\mathbf{r}, \omega) = \mathrm{i}\omega \mathbf{B}(\mathbf{r}, \omega) \tag{48b}$$
$$\nabla \cdot \mathbf{B}(\mathbf{r}, \omega) = 0 \tag{48c}$$
$$\nabla \times \mathbf{H}(\mathbf{r}, \omega) = -\mathrm{i}\omega \mathbf{D}(\mathbf{r}, \omega) + \mathbf{j}(\mathbf{r}, \omega) \tag{48d}$$

From which the following wave equation can be derived:

$$(\Delta + k^2)\mathbf{E} = -\frac{1}{\epsilon_0 \epsilon_{\mathrm{env}}} \left(k^2 + \nabla\nabla\right) \mathbf{P}. \tag{49}$$

This leads to the vectorial Lippmann–Schwinger equation in SI units (here for a vacuum reference system):

$$\mathbf{E}(\mathbf{r}, \omega) = \mathbf{E}_0(\mathbf{r}, \omega) + \int \mathbf{G}_0^{\mathrm{EE}}(\mathbf{r}, \mathbf{r}', \omega) \cdot \boldsymbol{\chi}_e \cdot \mathbf{E}(\mathbf{r}', \omega) \mathrm{d}\mathbf{r}' \tag{50}$$

with the Green's Dyad

$$\mathbf{G}_0^{\mathrm{EE}}(\mathbf{r}, \mathbf{r}', \omega) = \frac{\mathrm{e}^{ikR}}{4\pi \epsilon_0 \epsilon_{\mathrm{env}}} \Big(-k^2 \mathbf{T}_1(\mathbf{R}) - ik\mathbf{T}_2(\mathbf{R}) + \mathbf{T}_3(\mathbf{R}) \Big), \tag{51}$$

where the definitions of $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ given in Eqs. (6)–(8) are still valid.

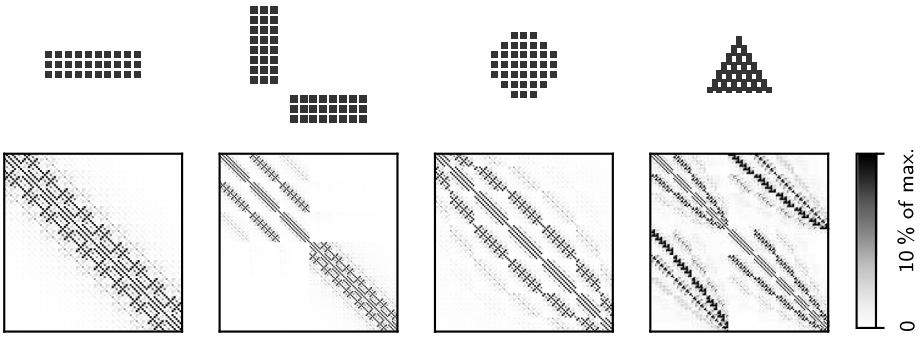

Fig. 25. Population patterns of matrices $\mathbf{M}$ at $\lambda = 1\,\mu$m for a selection of structures (stepsize 10 nm, same scale for all sketches). Cubic meshes for the first three structures, hexagonal compact mesh for the structure on the right. For illustrative purposes the structures are only one layer of mesh-points high (small matrix size). White corresponds to an absolute value of 0, black to $\geq$ 10 % of the matrix's largest element.

In Eq. (50) and its volume discretization

$$\mathbf{E}(\mathbf{r}_i, \omega) = \mathbf{E}_0(\mathbf{r}_i, \omega) + \chi_e \sum_{j=1}^{N} \mathbf{G}_0^{\mathrm{EE}}(\mathbf{r}_i, \mathbf{r}_j, \omega) \cdot \mathbf{E}(\mathbf{r}_j, \omega) V_{\mathrm{cell}} \tag{52}$$

one has to use the susceptibility $\chi_e = (\epsilon_r - \epsilon_{\mathrm{env}})$. For simplicity here we assumed a scalar χ_e.

Finally, the renormalization tensors Eqs. (17) and (18) have to be divided by a factor 4π.

Post-processing routines

Concerning the post-processing routines, some of the prefactors need to be adapted. For instance, the factor before the sums of Eqs. (39) and (40) is written in SI units:

$$\frac{2\pi n}{\lambda_0 |\mathbf{E}_0|^2} \tag{53}$$

with the refractive index n.

Eqs. (43) and (44) for heat-generation, respectively the local temperature increase have to be multiplied by a factor of 4π.

Note. In pyGDM the post-processing routines internally convert the results to SI compatible units. The "**extinct**" function for instance returns the cross sections in units of nm^2, "**heat**" returns nano Watts and "**temperature**" returns °K. For the respective returned units, see the technical documentation of the routines in the online documentation of the API e.g. at https://wiechapeter.gitlab.io/pyGDM2-doc/apidoc.html.

Appendix E. Conjugate gradients

The following applies to all pyGDM functions which solve the main GDM inversion problem. The conjugate gradients solver provides an alternative to full inversion of the coupled dipole problem which can – under circumstances – be preferable to complete matrix inversion.

- argument *method*: "cg" (requires *scipy*) or "pycg" (requires *pyamg*)

If we have a closer look at the matrix $\mathbf{M}$ (see Eq. (12)), we can make an interesting observation: While $\mathbf{M}$ is not exactly sparse, most of the entries have significantly smaller absolute values than very few large matrix elements. In Fig. 25 we show plots of the population of matrix $\mathbf{M}$ for some selected nano-structures. These population plots work as illustrated in the following examples:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} =$$

$\mathbf{M}$ contains also phase-information and is therefore complex, hence we use the absolute values of the matrix elements for the population patterns. In addition, the maximum of the color-code in Fig. 25 is clipped to 10% of the maximum absolute value in the matrix to increase the contrast. Clearly, the matrices contain very few entries with values of more than some % of the overall maximum and yet $>$ 60 % of all elements are generally non-zero.

It turns out, that such matrices are good candidates for iterative solving using so-called "Krylow-subspace methods". The most popular algorithm of this class is the conjugate gradients (CG) method and its derivations like biconjugate gradients (for non-symmetric problems) or complex CG [80]. A detailed description of the method can be found in Ref. [37] (chapter 2.7). The main idea of these iterative methods is, that the inverse of the matrix is in many cases not actually required. For simulations that massively make use of the generalized propagator (like raster-scan simulations), the CG technique is therefore not the method of choice. It may be on the other hand an advantageous approach, if we search a solution for $\mathbf{E}(\omega)$ that satisfies

$$\mathbf{M}(\omega) \cdot \mathbf{E}(\omega) = \mathbf{E}_0(\omega) \tag{54}$$

for one single or only few incident field $\mathbf{E}_0(\omega)$. During the CG-iterations, matrix–vector multiplications $\mathbf{M} \cdot \mathbf{x}$ are performed following a minimization scheme in which $\mathbf{M} \cdot \mathbf{x}$ converges eventually to $\mathbf{E}_0$. Theoretically, for a $N \times N$ matrix CG converge to the exact solution after N iterations and each iteration itself has a computational cost $\propto N^2$. In reality, the convergence is often very rapid in the beginning, and a solution with sufficient precision can be obtained after very few iterations, yielding a total computational cost $\propto N^2$ instead of a N^3 scaling for exact inversion for example with LU-decomposition. Indeed, we find a N^3-scaling for complete inversion by LU or Dyson's sequence and a N^2 dependence when using conjugate gradients (Fig. 6a). Particularly for larger numbers of meshpoints, this allows a reduction of the simulation time, as shown in Fig. 6a.

E.1. Preconditioning

- argument *pc_method*: "ilu", "lu" (both require *scipy*), "amg" (requires *pyamg*) or "none" (no preconditioning)

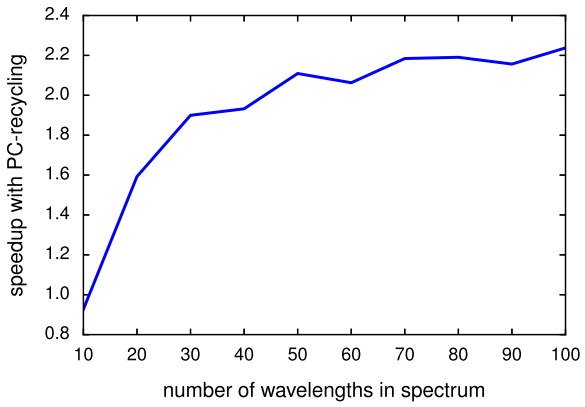

Fig. 26. Speedup of the GDM-calculation of a spectrum (2000 meshpoints Si nanowire, step of 10 nm, λ from 500 nm to 1500 nm) as function of the number of wavelengths, if recycling of the preconditioner is enabled. The narrower the wavelengths in the spectrum, the higher the possible gain of PC-recycling.

The speed of the convergence of conjugate gradients is crucially dependent on the condition of the matrix **M** and generally can be massively improved by doing a *preconditioning* step before starting the actual iterative scheme. Let us assume, **A** of the equation system

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{55}$$

would be the identity matrix **I**. Then CG would have converged within the first iteration. A possible approach for preconditioning is therefore to reshape the problem using a matrix **P**

$$\mathbf{A} \cdot \left(\mathbf{P} \cdot \hat{\mathbf{x}}\right) = \mathbf{b}. \tag{56}$$

If **P** is a close approximation to $\mathbf{A}^{-1}$, $\mathbf{A} \cdot \mathbf{P}$ will be close to the identity **I** and the system would converge very quickly under conjugate gradients iterations. Eq. (56) is called a right-preconditioned system. Consequently, a good preconditioner for our problem is a close approximation to the inverse of **M**. Several algorithms exist to search pseudo-inverse matrices for preconditioning. A very popular one is the *incomplete* LU-decomposition (ILU) [81] that scales with N^2 and which is the default method in pyGDM.

E.2. Preconditioner recycling

- argument *cg_recycle_pc*: "True" (=default)

When calculating spectra using the GDM, the electric field in a particle is usually calculated for a large number of closely spaced wavelengths, at each of which the matrix **M** is (incompletely) inverted. Most often, the electric field distribution changes only marginally for slightly different wavelengths and so does the matrix **M**. Unfortunately, a *very similar* matrix is of little use for exact calculations, but we have seen in the preceding section that an *approximation* to the exact inverse $\mathbf{M}^{-1}$ can be a good preconditioner **P** for CG.

When calculating dense spectra (i.e. many points on the wavelength axis), we can use this fact and significantly accelerate the calculation with conjugate gradients by recycling the preconditioner matrix until a certain lower limit for the speedup factor is reached. In other words, we will be using the same **P** repeatedly for several consecutive wavelengths and only if the acceleration is below a speed-up limit, a new preconditioner is calculated and subsequently re-used for the following wavelengths. As shown in Fig. 26, this technique can divide the total calculation time easily by more than a factor 2.

Another possible application when preconditioner recycling may be beneficial is in series of simulations with many very similar or slowly transformed nano-structures like antennas of gradually increasing size.

Note. The conjugate gradients solver is not very efficient for the moment and will be improved in future versions of pyGDM. In particular, in the specific case of the coupled dipole approximation it is possible to do very efficient vector/matrix multiplications by applying a fast Fourier transformation (FFT) scheme [27]. This is not yet implemented in pyGDM. The currently used third-party sparse-matrix solvers are not ideally suited regarding the dense matrix problem in pyGDM.

References

[1] J.C. Maxwell, Philos. Trans. R. Soc. Lond. 155 (1865) 459–512. http://dx.doi.org/10.1098/rstl.1865.0008.
[2] A.I. Kuznetsov, A.E. Miroshnichenko, M.L. Brongersma, Y.S. Kivshar, B. Luk'yanchuk, Science 354 (6314) (2016) aag2472. http://dx.doi.org/10.1126/science.aag2472.
[3] P. Bharadwaj, B. Deutsch, L. Novotny, Adv. Opt. Photonics 1 (3) (2009) 438. http://dx.doi.org/10.1364/AOP.1.000438.
[4] S. Maier, Plasmonics: Fundamentals and Applications, Springer US, 2010.
[5] C. Girard, O.J.F. Martin, A. Dereux, Phys. Rev. Lett. 75 (17) (1995) 3098–3101. http://dx.doi.org/10.1103/PhysRevLett.75.3098.
[6] L.-J. Black, Y. Wang, C.H. de Groot, A. Arbouet, O.L. Muskens, ACS Nano 8 (6) (2014) 6390–6399. http://dx.doi.org/10.1021/nn501889s.
[7] P.R. Wiecha, L.-J. Black, Y. Wang, V. Paillard, C. Girard, O.L. Muskens, A. Arbouet, Sci. Rep. 7 (2017) 40906. http://dx.doi.org/10.1038/srep40906.
[8] A. Arbabi, Y. Horie, M. Bagheri, A. Faraon, Nature Nanotechnol. 10 (11) (2015) 937–943. http://dx.doi.org/10.1038/nnano.2015.186.
[9] A.G. Curto, G. Volpe, T.H. Taminiau, M.P. Kreuzer, R. Quidant, N.F. van Hulst, Science 329 (5994) (2010) 930–933. http://dx.doi.org/10.1126/science.1191922.
[10] G. Baffou, R. Quidant, Laser Photonics Rev. 7 (2) (2013) 171–187. http://dx.doi.org/10.1002/lpor.201200003.
[11] M. Kauranen, A.V. Zayats, Nat. Photonics 6 (11) (2012) 737–748. http://dx.doi.org/10.1038/nphoton.2012.244.
[12] J. Butet, P.-F. Brevet, O.J.F. Martin, ACS Nano 13 (4) (2015) 10545–10562. http://dx.doi.org/10.1021/acsnano.5b04373.
[13] D. Griffiths, Introduction to Electrodynamics, Prentice-Hall International, 1989.
[14] C. Girard, Rep. Progr. Phys. 68 (8) (2005) 1883–1933. http://dx.doi.org/10.1088/0034-4885/68/8/R05.
[15] G.S. Agarwal, Phys. Rev. A 11 (1) (1975) 230–242. http://dx.doi.org/10.1103/PhysRevA.11.230.
[16] C. Girard, E. Dujardin, G. Baffou, R. Quidant, New J. Phys. 10 (10) (2008) 105016. http://dx.doi.org/10.1088/1367-2630/10/10/105016.
[17] G. Colas des Francs, D. Molenda, U.C. Fischer, A. Naber, Phys. Rev. B 72 (16) (2005) 165111.
[18] M. Paulus, P. Gay-Balmaz, O.J.F. Martin, Phys. Rev. E 62 (4) (2000) 5797–5807. http://dx.doi.org/10.1103/PhysRevE.62.5797.
[19] P.R. Wiecha, Linear and Nonlinear Optical Properties of High Refractive Index Dielectric Nanostructures (Ph.D. thesis), Université de Toulouse, Université Toulouse III - Paul Sabatier, 2016.
[20] O.J.F. Martin, C. Girard, A. Dereux, Phys. Rev. Lett. 74 (4) (1995) 526–529. http://dx.doi.org/10.1103/PhysRevLett.74.526.
[21] A. Yaghjian, Proc. IEEE 68 (2) (1980) 248–263. http://dx.doi.org/10.1109/PROC.1980.11620.
[22] C. Girard, A. Dereux, Rep. Progr. Phys. 59 (5) (1996) 657. http://dx.doi.org/10.1088/0034-4885/59/5/002.
[23] Y. Ould Agha, O. Demichel, C. Girard, A. Bouhelier, G.C. des Francs, Prog. Electromagn. Res. 146 (2014) 77–88. http://dx.doi.org/10.2528/PIER14012904.
[24] J. Kottmann, O. Martin, IEEE Trans. Antennas and Propagation 48 (11) (2000) 1719–1726. http://dx.doi.org/10.1109/8.900229.
[25] A. Teulle, R. Marty, S. Viarbitskaya, A. Arbouet, E. Dujardin, C. Girard, G. Colas des Francs, J. Opt. Soc. Amer. B 29 (9) (2012) 2431. http://dx.doi.org/10.1364/JOSAB.29.002431.
[26] B.T. Draine, P.J. Flatau, J. Opt. Soc. Am. A 11 (4) (1994) 1491. http://dx.doi.org/10.1364/JOSAA.11.001491.
[27] J.J. Goodman, B.T. Draine, P.J. Flatau, Opt. Lett. 16 (15) (1991) 1198. http://dx.doi.org/10.1364/OL.16.001198.
[28] N. Geuquet, L. Henrard, Ultramicroscopy 110 (8) (2010) 1075–1080. http://dx.doi.org/10.1016/j.ultramic.2010.01.013.
[29] F.J. García de Abajo, A. Howie, Phys. Rev. B 65 (11) (2002) 115418. http://dx.doi.org/10.1103/PhysRevB.65.115418.
[30] U. Hohenester, A. Trügler, Comput. Phys. Comm. 183 (2) (2012) 370–381. http://dx.doi.org/10.1016/j.cpc.2011.09.009.

[31] J. Waxenegger, A. Trügler, U. Hohenester, Comput. Phys. Comm. 193 (Supplement C) (2015) 138–150. http://dx.doi.org/10.1016/j.cpc.2015.03.023.

[32] U.S. Inan, R.A. Marshall, Numerical Electromagnetics: The FDTD Method, Cambridge University Press, 2011.

[33] F. Baida, A. Belkhir, Gratings: Theory and Numeric Applications, popov, e. ed., Presses Universitaires de Provence, 2013, pp. 333–366.

[34] Y. Cao, A. Manjavacas, N. Large, P. Nordlander, ACS Photonics 2 (3) (2015) 369–375. http://dx.doi.org/10.1021/ph500408e.

[35] J. Hoffmann, C. Hafner, P. Leidenberger, J. Hesselbarth, S. Burger, Comparison of Electromagnetic Field Solvers for the 3D Analysis of Plasmonic Nanoantennas, Vol. 7390, International Society for Optics and Photonics, 2009, p. 73900J. http://dx.doi.org/10.1117/12.828036.

[36] A.F. Oskooi, D. Roundy, M. Ibanescu, P. Bermel, J.D. Joannopoulos, S.G. Johnson, Comput. Phys. Comm. 181 (2010) 687–702. http://dx.doi.org/10.1016/j.cpc.2009.11.008.

[37] W. Press, Numerical Recipes 3rd Edition: The Art of Scientific Computing, Cambridge University Press, 2007.

[38] J. Smajic, C. Hafner, L. Raguin, K. Tavzarashvili, M. Mishrikey, J. Comput. Theor. Nanosci. 6 (3) (2009) 763–774. http://dx.doi.org/10.1166/jctn.2009.1107.

[39] M.A. Yurkin, A.G. Hoekstra, J. Quant. Spectrosc. Radiat. Transfer 106 (1) (2007) 558–589. http://dx.doi.org/10.1016/j.jqsrt.2007.01.034.

[40] S. Viarbitskaya, A. Teulle, R. Marty, J. Sharma, C. Girard, A. Arbouet, E. Dujardin, Nature Mater. 12 (5) (2013) 426–432. http://dx.doi.org/10.1038/nmat3581.

[41] R.W. Boyd, J. Opt. Soc. Amer. 70 (7) (1980) 877. http://dx.doi.org/10.1364/JOSA.70.000877.

[42] L. Novotny, B. Hecht, Principles of Nano-Optics, Cambridge University Press, Cambridge, New York, 2006.

[43] P.R. Wiecha, A. Arbouet, A. Cuche, V. Paillard, C. Girard, Phys. Rev. B 97 (8) (2018) 085411. http://dx.doi.org/10.1103/PhysRevB.97.085411.

[44] G. Colas des Francs, Optique Sub-Longueur d'onde et Fluorescence Moléculaire Perturbée (Ph.D. thesis), Université Paul Sabatier Toulouse, CEMES-CNRS, 2002.

[45] R. Carminati, A. Cazé, D. Cao, F. Peragut, V. Krachmalnicoff, R. Pierrat, Y. De Wilde, Surf. Sci. Rep. 70 (1) (2015) 1–41. http://dx.doi.org/10.1016/j.surfrep.2014.11.001.

[46] P.A. Huidobro, X. Shen, J. Cuerda, E. Moreno, L. Martin-Moreno, F.J. Garcia-Vidal, T.J. Cui, J.B. Pendry, Phys. Rev. X 4 (2) (2014) 021003. http://dx.doi.org/10.1103/PhysRevX.4.021003.

[47] P.R. Wiecha, C. Majorel, C. Girard, A. Arbouet, B. Masenelli, O. Boisron, A. Lecestre, G. Larrieu, V. Paillard, A. Cuche, Simultaneous mapping of the electric and magnetic photonic local density of states above dielectric nanostructures using rare-earth doped films, 2018, arXiv:1801.09690.

[48] E. Lassalle, A. Devilez, N. Bonod, T. Durt, B. Stout, J. Opt. Soc. Amer. B 34 (7) (2017) 1348–1355. http://dx.doi.org/10.1364/JOSAB.34.001348.

[49] C. Girard, J.-C. Weeber, A. Dereux, O.J.F. Martin, J.-P. Goudonnet, Phys. Rev. B 55 (24) (1997) 16487–16497. http://dx.doi.org/10.1103/PhysRevB.55.16487.

[50] B.T. Draine, Astrophys. J. 333 (1988) 848–872.

[51] L. Novotny, J. Opt. Soc. Amer. A 14 (1) (1997) 105–113. http://dx.doi.org/10.1364/JOSAA.14.000105.

[52] G. Baffou, R. Quidant, C. Girard, Appl. Phys. Lett. 94 (15) (2009) 153109. http://dx.doi.org/10.1063/1.3116645.

[53] G. Baffou, R. Quidant, C. Girard, Phys. Rev. B 82 (16) (2010) 165424. http://dx.doi.org/10.1103/PhysRevB.82.165424.

[54] P.R. Wiecha, M.-M. Mennemanteuil, D. Khlopin, J. Martin, A. Arbouet, D. Gérard, A. Bouhelier, J. Plain, A. Cuche, Phys. Rev. B 96 (3) (2017) 035440. http://dx.doi.org/10.1103/PhysRevB.96.035440.

[55] S. Viarbitskaya, A. Cuche, A. Teulle, J. Sharma, C. Girard, A. Arbouet, E. Dujardin, ACS Photonics 2 (6) (2015) 744–751. http://dx.doi.org/10.1021/acsphotonics.5b00100.

[56] Y.H. Fu, A.I. Kuznetsov, A.E. Miroshnichenko, Y.F. Yu, B. Luk'yanchuk, Nature Commun. 4 (2013) 1527. http://dx.doi.org/10.1038/ncomms2538.

[57] I.M. Hancu, A.G. Curto, M. Castro-López, M. Kuttge, N.F. van Hulst, Nano Lett. 14 (1) (2014) 166–171. http://dx.doi.org/10.1021/nl403681g.

[58] D. Macias, A. Vial, D. Barchiesi, J. Opt. Soc. Am. A 21 (8) (2004) 1465–1471. http://dx.doi.org/10.1364/JOSAA.21.001465.

[59] T.W. Odom, E.-A. You, C.M. Sweeney, J. Phys. Chem. Lett. 3 (18) (2012) 2611–2616. http://dx.doi.org/10.1021/jz300886z.

[60] H.-P.P. Schwefel, Evolution and Optimum Seeking: The Sixth Generation, John Wiley & Sons, Inc., New York, NY, USA, 1993.

[61] F. Biscani, D. Izzo, C.H. Yam, A Global Optimisation Toolbox for Massively Parallel Engineering Optimisation, arXiv:1004.3824 [cs, math].

[62] D. Izzo, M. Ruciński, F. Biscani, in: F.F. de Vega, J.I.H. Pérez, J. Lanchares (Eds.), Parallel Architectures and Bioinspired Algorithms, in: Studies in Computational Intelligence, vol. 415, Springer Berlin Heidelberg, 2012, pp. 151–169.

[63] T. Brixner, F.J. García de Abajo, J. Schneider, C. Spindler, W. Pfeiffer, Phys. Rev. B 73 (12) (2006) 125437. http://dx.doi.org/10.1103/PhysRevB.73.125437.

[64] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, Vol. 16, Wiley, 2001.

[65] P.R. Wiecha, A. Arbouet, C. Girard, A. Lecestre, G. Larrieu, V. Paillard, Nature Nanotechnol. 12 (2) (2017) 163–169. http://dx.doi.org/10.1038/nnano.2016.224.

[66] C. Girard, P.R. Wiecha, A. Cuche, E. Dujardin, J. Opt. 20 (7) (2018) 075004. http://dx.doi.org/10.1088/2040-8986/aac934.

[67] P.R. Wiecha, A. Arbouet, C. Girard, A. Lecestre, G. Larrieu, V. Paillard, Computational Optics II, Vol. 10694, International Society for Optics and Photonics, 2018, p. 1069402. http://dx.doi.org/10.1117/12.2315123.

[68] R. Marty, C. Girard, A. Arbouet, G. Colas des Francs, Chem. Phys. Lett. 532 (Supplement C) (2012) 100–105. http://dx.doi.org/10.1016/j.cplett.2012.02.058.

[69] X.S. Li, ACM Trans. Math. Softw. 31 (3) (2005) 302–325. http://dx.doi.org/10.1145/1089014.1089017.

[70] M. Paulus, O.J.F. Martin, Phys. Rev. E 63 (6) (2001) 066615. http://dx.doi.org/10.1103/PhysRevE.63.066615.

[71] P.R. Wiecha, A. Arbouet, C. Girard, T. Baron, V. Paillard, Phys. Rev. B 93 (12) (2016) 125421. http://dx.doi.org/10.1103/PhysRevB.93.125421.

[72] A. Arbouet, A. Mlayah, C. Girard, G. Colas des Francs, New J. Phys. 16 (11) (2014) 113012. http://dx.doi.org/10.1088/1367-2630/16/11/113012.

[73] C. Girard, A. Dereux, O.J.F. Martin, M. Devel, Phys. Rev. B 52 (4) (1995) 2889–2898. http://dx.doi.org/10.1103/PhysRevB.52.2889.

[74] A. Kwadrin, A.F. Koenderink, Phys. Rev. B 87 (12) (2013) 125123. http://dx.doi.org/10.1103/PhysRevB.87.125123.

[75] B. Gallinet, O.J.F. Martin, Theoretical and Computational Nanophotonics, Vol. 1176, (Tacona-Photonics 2009), 2009, pp. 63–65.

[76] P.C. Chaumet, A. Sentenac, J. Quant. Spectrosc. Radiat. Transfer 110 (6–7) (2009) 409–414. http://dx.doi.org/10.1016/j.jqsrt.2008.12.004.

[77] R. Esteban, A.G. Borisov, P. Nordlander, J. Aizpurua, Nature Commun. 3 (2012) 825. http://dx.doi.org/10.1038/ncomms1806.

[78] J.-J. Greffet, R. Carminati, Prog. Surf. Sci. 56 (3) (1997) 133–237. http://dx.doi.org/10.1016/S0079-6816(98)00004-5.

[79] J.A. Porto, R. Carminati, J.-J. Greffet, J. Appl. Phys. 88 (8) (2000) 4845–4850. http://dx.doi.org/10.1063/1.1311811.

[80] P. Joly, G. Meurant, Numer. Algorithms 4 (3) (1993) 379–406. http://dx.doi.org/10.1007/BF02145754.

[81] X.S. Li, M. Shao, ACM Trans. Math. Softw. 37 (4) (2011) 43. http://dx.doi.org/10.1145/1916461.1916467.