

MNPBEM – A Matlab toolbox for the simulation of plasmonic nanoparticles

Ulrich Hohenester, Andreas Trügler

Institut für Physik. Karl-Franzens-Universität Graz, Universitätsplatz 5, 8010 Graz, Austria

Abstract

MNPBEM is a Matlab toolbox for the simulation of metallic nanoparticles (MNP), using a boundary element method (BEM) approach. The main purpose of the toolbox is to solve Maxwell's equations for a dielectric environment where bodies with homogeneous and isotropic dielectric functions are separated by abrupt interfaces. Although the approach is in principle suited for arbitrary body sizes and photon energies, it is tested (and probably works best) for metallic nanoparticles with sizes ranging from a few to a few hundreds of nanometers, and for frequencies in the optical and near-infrared regime. The toolbox has been implemented with Matlab classes. These classes can be easily combined, which has the advantage that one can adapt the simulation programs flexibly for various applications.

Keywords: Plasmonics, metallic nanoparticles, boundary element method

Program summary

Program title: MNPBEM.

Programming language: Matlab 7.11.0 (R2010b)

Computer: Any which supports Matlab 7.11.0 (R2010b)

Operating system: Any which supports Matlab 7.11.0 (R2010b)

RAM required to execute with typical data: ≥ 1 GByte

Has the code been vectorised or parallelized?: no

Keywords: Plasmonics, metallic nanoparticles, boundary element method

Email address: ulrich.hohenester@uni-graz.at (Ulrich Hohenester)

URL: <http://physik.uni-graz.at/~uxh> (Ulrich Hohenester)

CPC Library Classification: Optics

External routines/libraries used: MESH2D available at www.mathworks.com

Nature of problem: Solve Maxwell's equations for dielectric particles with homogeneous dielectric functions separated by abrupt interfaces

Solution method: Boundary element method using electromagnetic potentials

Running time: Depending on surface discretization between seconds and hours

1. Introduction

Plasmonics is an emerging field with numerous applications foreseen, ranging from sensorics over extreme light concentration and light harvesting to optical and quantum technology, as well as metamaterials and optical cloaking [1–5]. The workhorse of plasmonics are *surface plasmons*, these are coherent electron charge oscillations bound to the interface between a metal and a dielectric [2, 6]. These surface plasmons come along with strongly localized, so-called evanescent electromagnetic fields, which can be exploited for bringing light down to the nanoscale, thereby overcoming the diffraction limit of light and bridging between the micrometer length scale of optics and the nanometer length scale of nanostructures. On the other hand, tiny variations of the dielectric environment close to the nanostructures, e.g. induced by binding of molecules to a functionalized metal surface, can significantly modify the evanescent fields and, in turn, the surface plasmon resonances. This can be exploited for (bio)sensor applications, eventually bringing the sensitivity down to the single-molecule level.

Of particular interest are *particle plasmons*, these are surface plasmons confined in all three spatial dimensions to the surface of a nanoparticle [2, 7]. The properties of these excitations depend strongly on particle geometry and interparticle coupling, and give rise to a variety of effects, such as frequency-dependent absorption and scattering or near field enhancement. Particle plasmons enable the concentration of light fields to nanoscale volumes and play a key role in surface enhanced spectroscopy [8–11].

Simulation of particle plasmons is nothing but the solution of Maxwell's equations for metallic nanoparticles embedded in a dielectric environment. Consequently, the simulation toolboxes usually employed in the field are not specifically designed for plasmonics applications. For instance, the discrete dipole approximation toolbox DDSCATT [12, 13] was originally designed for the

simulation of scattering from interstellar graphite grains, but has in recent years been widely used within the field of plasmonics. Also the finite difference time domain (FDTD) approach [14, 15] has been developed as a general simulation toolkit for the solution of Maxwell’s equations. Other computational approaches widely used in the field of plasmonics are the dyadic Green function technique [16] or the multiple multipole method [6].

In this paper we present the simulation toolbox **MNPBEM** for metallic nanoparticles (MNP), which is based on a boundary element method (BEM) approach developed by Garcia de Abajo and Howie [17, 18]. The approach is less general than the above approaches, in that it assumes a dielectric environment where bodies with homogeneous and isotropic dielectric functions are separated by abrupt interfaces, rather than allowing for a general inhomogeneous dielectric environment. On the other hand, for most plasmonics applications with metallic nanoparticles embedded in a dielectric background the BEM approach appears to be a natural choice. It has the advantage that only the boundaries between the different dielectric materials have to be discretized, and not the whole volume, which results in faster simulations with more moderate memory requirements.

The **MNPBEM** toolbox has been designed such that it provides a flexible toolkit for the simulation of the electromagnetic properties of plasmonic nanoparticles. The toolbox works in principle for arbitrary dielectric bodies with homogeneous dielectric properties, which are separated by abrupt interfaces, although we have primarily used and tested it for metallic nanoparticles with diameters ranging from a few to a few hundred nanometers, and for frequencies in the optical and near-infrared regime. We have developed the programs over the last few years [19, 20], and have used them for the simulation of optical properties of plasmonic particles [21, 22], surface enhanced spectroscopy [23–25], sensorics [26, 27], and electron energy loss spectroscopy (EELS) [28, 29].

In the past year, we have completely rewritten the code using classes within Matlab 7.11. These classes can be easily combined such that one can adapt the simulation programs flexibly to the user’s needs. A comprehensive help is available for all classes and functions of the toolbox through the `doc` command. In addition, we have created detailed help pages, accessible in the Matlab help browser, together with a complete list of the classes and functions of the toolbox, and a number of demo programs. In this paper we provide an ample overview of the **MNPBEM** toolbox, but leave the details to the help pages. As the theory underlying our BEM approach has been

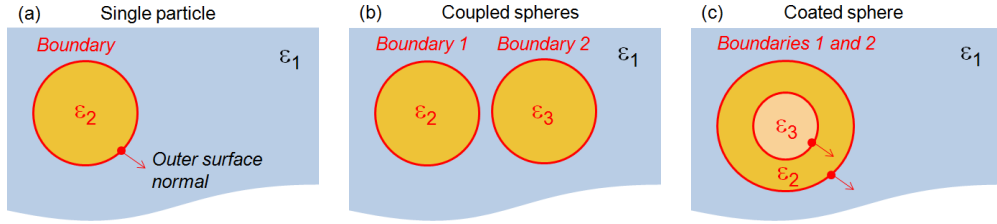


Figure 1: A few representative model systems suited for simulation within the MNPBEM toolbox: (a) Metallic nanosphere embedded in a dielectric background, (b) coupled nanospheres, and (c) coated nanosphere. The dielectric functions are denoted with ε_i . In panels (a) and (c) we also report the outer surface normals of the particle boundaries.

presented in great detail elsewhere [17, 18], in the following we only give a short account of the approach and refer the interested reader to the pertinent literature and the help pages.

Throughout the MNPBEM toolbox, lengths are measured in nanometers and photon energies through the light wavelength λ (in vacuum) in nanometers. In the programs we use for λ the notation `enei` (inverse of photon energy). With the only exception of the classes for the dielectric functions, one could also measure distances and wavelengths in other units such as e.g. micrometers or atomic units. Inside the toolbox we use Gauss units, in accordance with Refs. [17, 18]. This is advantageous for the scalar and vector potentials, which are at the heart of our BEM approach, and which could not be treated on an equal footing with the SI system. For most applications, however, the units remain completely hidden within the core routines of the BEM solvers.

2. Theory

In the following we consider dielectric nanoparticles, described through local and isotropic dielectric functions $\varepsilon_i(\omega)$, which are separated by sharp boundaries ∂V_i . A few representative examples are shown in Fig. 1. When the particles are excited by some external perturbation, such as an incoming plane wave or the fields created by a nearby oscillating dipole, they will become polarized and electromagnetic fields are induced. The goal of the MNPBEM toolbox is to compute for a given external perturbation these induced electromagnetic fields. This is achieved by solving Maxwell's equations and using the boundary conditions at the particle boundaries.

2.1. Quasistatic approximation

We first discuss things for nanoparticles much smaller than the light wavelength, where one can employ the quasistatic approximation. Here one solves the Poisson or Laplace equation for the electrostatic potential [30], rather than the Helmholtz equation for the scalar and vector potentials, but keeps the full frequency-dependent dielectric functions in the evaluation of the boundary conditions [6, 20]. A convenient solution scheme is provided by the electrostatic Green function

$$\nabla^2 G(\mathbf{r}, \mathbf{r}') = -4\pi\delta(\mathbf{r} - \mathbf{r}'), \quad G(\mathbf{r}, \mathbf{r}') = \frac{1}{|\mathbf{r} - \mathbf{r}'|}, \quad (1)$$

which is the proper solution of the Poisson equation for a point-like source in an unbounded, homogeneous medium. In case of an inhomogeneous dielectric environment, with homogeneous dielectric particles V_i separated by sharp boundaries ∂V_i , inside a given region $\mathbf{r} \in V_i$ one can write down the solution in the *ad-hoc* form [17–19]

$$\phi(\mathbf{r}) = \phi_{\text{ext}}(\mathbf{r}) + \oint_{V_i} G(\mathbf{r}, \mathbf{s}) \sigma(\mathbf{s}) da. \quad (2)$$

Here ϕ_{ext} is the external electrostatic potential, and $\sigma(\mathbf{s})$ is a surface charge distribution located at the particle boundary ∂V_i . Eq. (2) is constructed such that it fulfills the Poisson or Laplace equation everywhere except at the particle boundaries. The surface charge distribution $\sigma(\mathbf{s})$ has to be chosen such that the appropriate boundary conditions of Maxwell's equations are fulfilled. Continuity of the parallel electric field implies that σ has to be the same in- and outside the particle. From the continuity of the normal dielectric displacement we find the boundary integral equation [18, 19, 31]

$$\Lambda \sigma(\mathbf{s}) + \oint \frac{\partial G(\mathbf{s}, \mathbf{s}')}{\partial n} \sigma(\mathbf{s}') da' = -\frac{\partial \phi_{\text{ext}}(\mathbf{s})}{\partial n}, \quad \Lambda = 2\pi \frac{\varepsilon_2 + \varepsilon_1}{\varepsilon_2 - \varepsilon_1}, \quad (3)$$

whose solutions determine the surface charge distribution σ . Here $\frac{\partial}{\partial n}$ denotes the derivative along the direction of the outer surface normal, and ε_1 and ε_2 are the dielectric functions in- and outside the particle boundary, respectively. Approximating the integral in Eq. (3) by a sum over surface elements, one arrives at a *boundary element method* (BEM) approach with surface charges σ_i given at the discretized surface elements. From

$$\Lambda \sigma_i + \sum_j \left(\frac{\partial G}{\partial n} \right)_{ij} \sigma_j = - \left(\frac{\partial \phi_{\text{ext}}}{\partial n} \right)_i \quad (4)$$

one can obtain the surface charges σ_i through simple matrix inversion. Equation (4) constitutes the main equation of the quasistatic BEM approach. The central elements are Λ , which is governed by the dielectric functions in- and outside the particle boundaries, the surface derivatives $(\frac{\partial G}{\partial n})_{ij}$ of the Green function connecting surface elements i and j , and the surface derivative $(\frac{\partial \phi_{\text{ext}}}{\partial n})_i$ of the external potential.

2.2. Full Maxwell equations

A similar scheme can be applied when solving the full Maxwell equations. We now need both the scalar and vector potentials, which both fulfill a Helmholtz equation [17, 30]. Again we define a Green function

$$(\nabla^2 + k_i^2) G_i(\mathbf{r}, \mathbf{r}') = -4\pi\delta(\mathbf{r} - \mathbf{r}'), \quad G_i(\mathbf{r}, \mathbf{r}') = \frac{e^{ik_i|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r} - \mathbf{r}'|}, \quad (5)$$

where $k_i = \sqrt{\varepsilon_i}k$ is the wavenumber in the medium $\mathbf{r} \in V_i$, $k = \omega/c$ is the wavenumber in vacuum, and c is the speed of light. The magnetic permeability μ is set to one throughout. In analogy to the quasistatic case, for an inhomogeneous dielectric environment we write down the solutions in the *ad-hoc* form [17, 18]

$$\phi(\mathbf{r}) = \phi_{\text{ext}}(\mathbf{r}) + \oint_{V_i} G_i(\mathbf{r}, \mathbf{s}) \sigma_i(\mathbf{s}) da \quad (6)$$

$$\mathbf{A}(\mathbf{r}) = \mathbf{A}_{\text{ext}}(\mathbf{r}) + \oint_{V_i} G_i(\mathbf{r}, \mathbf{s}) \mathbf{h}_i(\mathbf{s}) da, \quad (7)$$

which fulfill the Helmholtz equations everywhere except at the particle boundaries. σ_i and \mathbf{h}_i are surface charge and current distributions, and ϕ_{ext} and \mathbf{A}_{ext} are the scalar and vector potentials characterizing the external perturbation.

The scalar and vector potentials are additionally related through the Lorentz gauge condition $\nabla \cdot \mathbf{A} = ik\varepsilon\phi$ [17], which in principle allows to express ϕ through the divergence of \mathbf{A} . However, it is advantageous to keep both the scalar and vector potential: in the evaluation of the boundary conditions we then only need the potentials together with their first, rather than also second, surface derivatives. The potential-based BEM approach then invokes only G_i together with its surface derivative $\partial G_i/\partial n$, in contrast to the field-based BEM approach which also invokes the second surface derivatives $\partial^2 G_i/\partial n^2$ [30, 32]. As higher derivatives of G_i translate to functions with

higher spatial variations, it is computationally favorable to keep only first-order surface derivatives. Another advantage of Eq. (7) is that the different components of \mathbf{A} are manipulated *separately*. Thus, when transforming to a BEM approach, by discretizing the boundary integrals, we end up with matrices of the order $N \times N$, where N is the number of boundary elements. In contrast, for field based BEM approaches the matrices are of the order $3N \times 3N$, where the factor of three accounts for the three spatial dimensions.

In the BEM approach the boundary integrals derived from Eqs. (6,7) are approximated by sums over boundary elements. Exploiting the boundary conditions of Maxwell's equations, in analogy to the quasistatic case, we derive a set of rather lengthy equations for the surface charges and currents [17, 18] (see also the help pages of the MNPBEM toolbox), which can be solved through matrix inversions and multiplications. In contrast to the quasistatic case, the surface charges $\sigma_{1,2}$ and currents $\mathbf{h}_{1,2}$ in- and outside the boundary (measured with respect to the surface normal \mathbf{n}) are not identical. Once σ and \mathbf{h} are determined, we can compute through Eqs. (6,7) the potentials everywhere else, as well as the electromagnetic fields, which are related to the potentials through the usual relations $\mathbf{E} = ik\mathbf{A} - \nabla\phi$ and $\mathbf{H} = \nabla \times \mathbf{A}$.

3. Getting started

3.1. Installation of the toolbox

To install the toolbox, one must simply add the path of the main directory `mnpbemdir` of the MNPBEM toolbox as well as the paths of all subdirectories to the Matlab search path. This can be done, for instance, through

```
addpath(genpath(mnpbemdir));
```

For particle shapes derived from 2D polygons, to be described in Sec. 4, one additionally needs the toolbox `MESH2D - Automatic Mesh Generation` available at www.mathworks.com. Again, one should add the path of the corresponding directory to the Matlab path.

To set up the help pages, one must once change to the main directory of the MNPBEM toolbox and run the program `makemnpbemhelp`

```
>> cd mnpbemdir;
>> makemnpbemhelp;
```

Once this is done, the help pages, which provide detailed information about the toolbox, are available in the Matlab help browser. Figure 2 shows a screenshot of the MNPBEM help pages.

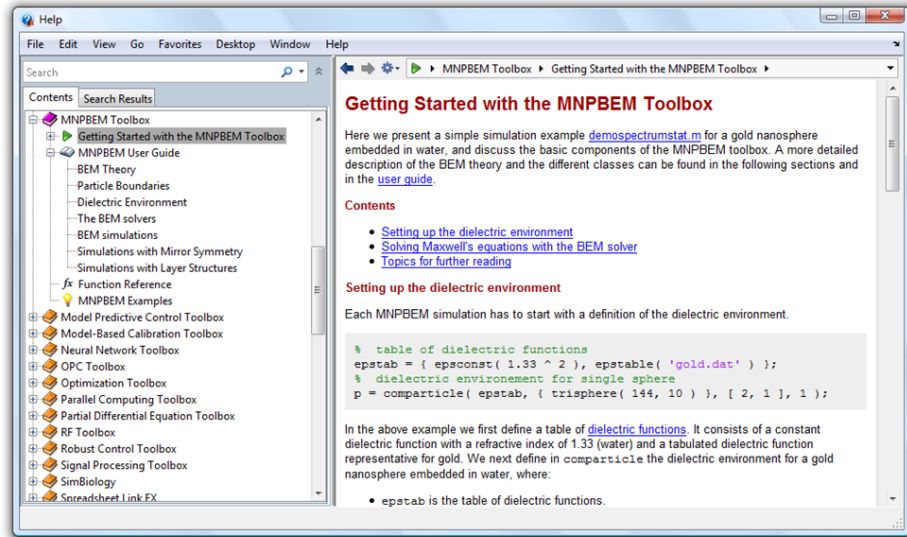


Figure 2: Screenshot of the help pages of the MNPBEM toolbox within the Matlab help browser. The help pages provide a short introduction, a detailed user guide, a list of the classes and functions of the toolbox, as well as a number of demo programs.

3.2. A simple example

Let us start with the discussion of a simple example. We consider a metallic nanosphere embedded in water, which is excited by an electromagnetic plane wave, corresponding to light excitation from a source situated far away from the object. For this setup we compute the light scattered by the nanosphere. The file `demospectrumstat.m` for the corresponding simulation is available in the `Demo` subdirectory and can be opened by typing

```
>> edit demospectrumstat
```

The simulation consists of the following steps:

- define the dielectric functions;
- define the particle boundaries;
- specify how the particle is embedded in the dielectric environment;
- set up a solver for the BEM equations;
- specify the excitation scheme (here plane wave excitation);
- solve the BEM equations for the given excitation by computing the auxiliary surface charges (and currents);

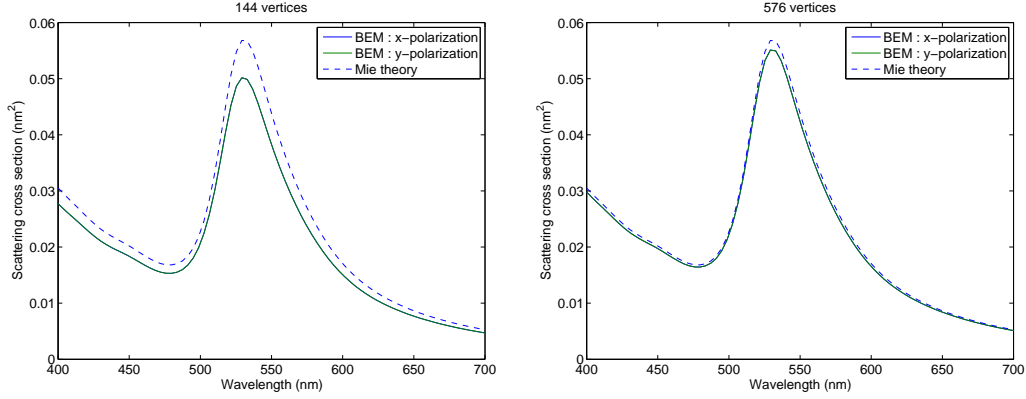


Figure 3: Scattering cross section for a nanosphere with a diameter of 10 nanometers. We compare the results of BEM simulations for a sphere discretization with 144 (left panel) and 576 (right panel) vertices with those of Mie theory. The dielectric function of gold is taken from Ref. [33] and the refractive index of the embedding medium $n_b = 1.33$ is representative for water. The results for x and y polarization are indistinguishable.

- compute the response of the plasmonic nanoparticle (here scattering cross section) to the external excitation.

We next discuss the various steps in more detail. In `demospectrumstat.m` we first set up a table of dielectric functions, needed for the problem under study, and a discretized particle boundary which is stored in the form of vertices and faces. A more detailed description of the different elements of the MNPBEM toolbox will be given in the sections below as well as in the help pages.

```
% table of dielectric functions
epstab = {epsconst(1.33^2),epstable('gold.dat')};
% nanosphere with 144 vertices and 10 nanometers diameter
p = trisphere(144,10);
```

In the above lines we first define two dielectric functions, one for water (refractive index $n_b = 1.33$) and one for gold, and then create a discretized sphere surface with 144 vertices. For all functions and classes of the toolbox additional information can be obtained by typing

```
>> doc trisphere
```

We next have to define the dielectric properties of the nanosphere, depicted in Fig. 1(a), by setting up a `comparticle` object

```
p = comparticle(epstab,{p},[2,1],1);
```

The first two arguments are cell arrays for the dielectric functions and for the particle boundaries. The third argument `inout=[2,1]` describes how the particle boundaries and the dielectric functions are related. In the above example we specify that the material at the in- and out-side of the boundary are `epstab{2}` and `epstab{1}`, respectively. Note that the in- and out-side are defined with respect to the surface normal \mathbf{n} , whose direction is given by the order of the face elements. To check that these surface normals point into the right direction one can plot the particle with

```
>> plot(p,'nvec',true);
```

Finally, the last argument in the call to `comparticle` indicates that the particle surface is closed. It is important to provide this additional information, as will be discussed in Sec. 5. Once the `comparticle` object is set up, it is ready for use with the BEM solvers. With

```
% quasistatic BEM solver
bem = bemstat(p);
% plane wave excitation for given light polarizations
exc = planewavestat([1,0,0;0,1,0]);
```

we set up a solver for the BEM equations within the quasistatic approximation, and a plane wave excitation for polarizations along x and y . We next make a loop over the different wavelengths `enei`. For each wavelength we solve the BEM equations and compute the scattering cross section [34]

```
% light wavelength in vacuum in nanometers
enei = linspace(400,700,80);
% scattering spectrum (initialization of array with zeros)
sca = zeros(length(enei),2);

% main loop over different excitation wavelengths
for ien = 1:length(enei)
    sig = bem \ exc(p,enei(ien));
    sca(ien,:) = exc.sca(sig);
end
```

The solution of the BEM equations is through `sig=bem\exc(p,enei)`, where `exc(p,enei)` returns for the external light illumination the surface derivative

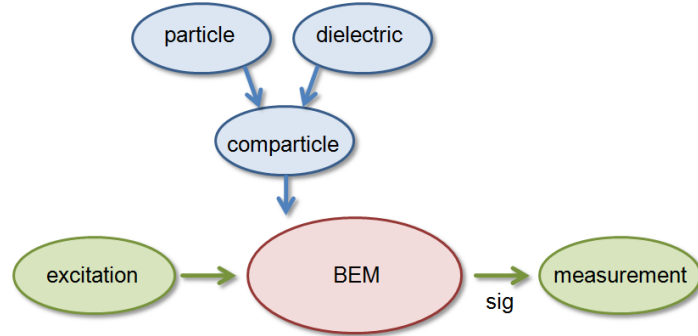


Figure 4: Flow chart for a typical BEM simulations: we first initialize the BEM solver with a `comparticle` object, that holds tables of the dielectric functions and of the discretized particle boundaries. The BEM solver (quasistatic or retarded) communicates with additional excitation-measurement classes, such as `planewave` for plane wave illumination or `dipole` for the excitation from an oscillating dipole. The BEM solver computes the surface charges (currents) `sig` which can be further processed for measurement purposes.

of the potential at the particle boundary. Finally, we can plot the scattering cross section and compare with the results of Mie theory [34], Fig. 3, using the `miestat` class provided by the toolbox. Similarly, the extinction cross section can be computed with `exc.ext(sig)`.

Figure 4 shows the flow chart for a typical BEM simulation. First, we initialize the BEM solver with a `comparticle` object, that holds tables of the dielectric functions and of the discretized particle boundaries. The BEM solver then computes for a given excitation the surface charges `sig`, which can then be used for the calculation of measurement results, such as scattering or extinction cross sections. As the excitation and measurement commands are not hidden inside a function, but appear explicitly inside the wavelength loop, it is possible to further process the results of the BEM simulation. For instance, we could plot the surface charges through `plot(p,real(sig.sig))` or compute the induced electric fields, as will be described further below.

To compute the scattering cross section for the retarded, i.e. full, Maxwell's equations, we simply have to use a different BEM solver and excitation-measurement class

```

% full BEM solver
bem = bemret(p);
% plane wave excitation for given light polarizations
exc = planewaveret([1,0,0;0,1,0],[0,0,1;0,0,1]);

```

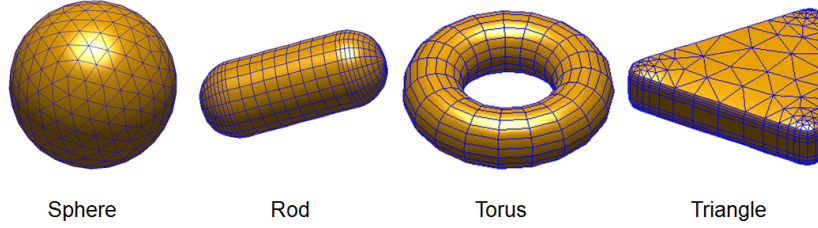


Figure 5: Discretized particle boundaries, as created with the functions `trisphere`, `trirod`, and `tritorus` of the MNPBEM toolbox. The triangle on the right-hand side is created from a `polygon` object, which is extruded with the `tripolygon` function using the `Mesh2d` toolbox.

Note that in the call to `planewaveret` we now have to specify also the light propagation directions.

4. Particle boundaries

The first, and usually most time-consuming job in setting up a MNPBEM simulation is to discretize the particle boundaries. The discretized boundary is stored as a `particle` object

```
p = particle(verts,faces);
```

Here `verts` are the vertices and `faces` the faces of the boundary elements, similarly to the `patch` objects of Matlab. `faces` is a $N \times 4$ array with N being the number of boundary elements. For each element the four entries point to the corners of a quadrilateral. For triangular boundary elements the last entry should be a `NaN`.

In principle, in Matlab there exist myriads of surface discretization functions, and the MNPBEM toolbox only provides a few additional ones

```
% sphere with given number of vertices and diameter
p = trisphere(nverts,diameter);
% nanorod with given diameter and height
p = trirod(diameter,height);
% torus with given outer and inner radius
p = tritorus(rout,rin);
```

Figure 5 shows the corresponding particle surfaces. All functions can receive additional options to control the number of discretization points, as detailed

in the `doc` command or the help pages. In addition to the vertices and faces, the `particle` object also stores for each boundary element the area and centroid, as well as three orthogonal vectors, where `nvec` is the outer surface normal.

In many cases one has to deal with flat particles, where a 2D polygon is extruded along the third direction. The `MNPBEM` toolbox provides a `polygon` class together with surface discretization and extrusion functions, building upon the `Mesh2d` toolbox. Let us look to an example for discretizing the surface of a triangular particle

```
% 2D polygon for triangle with rounded edges
poly = round(polygon(3,'size',[20,20]));
% height profile for extruding polygon
[edge,z] = edgeprofile(4);
% extrude polygon
p = tripolygon(poly,z,'edge',edge);
```

The corresponding particle is displayed in the right panel of Fig. 5. The command `edgeprofile` returns an array of (x,y) values that control the rounding-off of the particle. With `tripolygon` we triangulate the rounded 2D triangle and extrude the particle along the z direction. In the help pages of the toolbox we explain in more detail how to call `tripolygon` and related functions in order to utilize the full potential of the `Mesh2d` toolbox.

5. Dielectric environment

5.1. Dielectric functions

There exist three classes for dielectric functions.

epsconst A constant dielectric function is initialized with `epsconst(val)`, where `val` is the dielectric constant.

epsdrude A Drude dielectric function for metals $\varepsilon(\omega) = \varepsilon_0 - \omega_p^2/\omega(\omega + i\gamma)$ is initialized with `epsdrude(name)`, where `name` is the name of the metal. We have implemented 'Au', 'Ag', and 'Al' for gold, silver, and aluminum.

epstable A tabulated dielectric function is initialized with `epstable(finp)`, where `finp` is the name of an input file. This file must be in ASCII

format where each line holds the values `ene n k`, with `ene` being the photon energy in eV, and `n` and `k` the real and imaginary part of the refractive index $\sqrt{\epsilon}$, respectively. In the toolbox we provide the files `'gold.dat'` and `'silver.dat'` for the gold and silver dielectric functions tabulated in Ref. [33].

For a dielectric object `objeps` of one of these classes, one can compute the dielectric function and the wavenumber inside the medium with

```
[eps,k] = epsobj(enei);
```

where `enei` is as usual the wavelength of light in vacuum.

5.2. The `comparticle` class

The `comparticle` class defines how the particle boundaries are embedded in the dielectric environment. As previously discussed, the outer surface normal \mathbf{n} allows to distinguish between the boundary in- and out-side. For more complex particles, such as a dumbbell-like particle, it is not always possible to choose the particle boundaries such that only particle insides or outsides are connected. In these cases the meaning of in- and out-side is just a matter of convention.

To initialize a dielectric environment, one calls

```
p = comparticle({eps1,eps2,...},{p1,p2,...},inout,closed);
```

The first and second argument are cell arrays of the dielectric functions and particle boundaries characterizing the problem. In general, we recommend to set the first entry of the dielectric functions to that of the embedding medium, as several functions, such as for the calculation of the scattering or extinction cross sections, assume this on default. The third argument `inout=[i1,o1;i2,o2;...]` defines for each particle boundary `p`. the dielectric functions `eps{i.}` and `eps{o.}` at the in- and out-side of the boundary. For instance, for the particles depicted in Fig. 1 we get

```
% single sphere of Fig. 1(a)
p = comparticle({eps1,eps2},{p},[2,1],1);
% coupled spheres of Fig. 1(b)
p = comparticle({eps1,eps2,eps3},{p1,p2},[2,1;3,1],1,2);
% coated particles of Fig. 1(c)
p = comparticle({eps1,eps2,eps3},{p1,p2},[2,1;3,2],1,2);
```

The last argument (or arguments) of the `comparticle` initialization define closed boundaries. In general, for a closed boundary ∂V_i the following sum rule applies [35]

$$\oint_{\partial V_i} \frac{\partial G(\mathbf{s}, \mathbf{s}')}{\partial n} da' = 2\pi, \quad \sum_j \left(\frac{\partial G}{\partial n} \right)_{ij} = 2\pi, \quad (8)$$

which can be used to compute the diagonal elements of the surface derivative of the Green function. Surprisingly, these diagonal elements play an extremely important role for obtaining accurate results even for coarse boundary discretizations. For this reason it is important to indicate closed particle boundaries. In the above examples we specify that boundaries `p`, `p1`, and `p2` are closed. For instance, when the boundary of a particle is composed of two objects `p1` and `p2` one indicates this with

```
p = comparticle({eps1,eps2},{p1,p2},[2,1;2,1],[1,2]);
```

These calling sequences for `comparticle` are sufficiently general to cope with even more complicated compositions of dielectric particles.

5.3. The `compoint` class

The `MNPBEM` toolbox also provides a class `compoint` for points within a dielectric environment. These objects are particularly useful for computing maps of electromagnetic fields or defining the properties of oscillating dipoles. Consider a list of n positions `poslst`, i.e. a $n \times 3$ array, together with a `comparticle` object `p`. With

```
% place the points into the dielectric environment
pt = compoint(p,poslst);
pt = compoint(p,poslst,'medium',1);
pt = compoint(p,poslst,'mindist',2);
```

we can place these points inside the dielectric environment of `p`. By passing the additional property name `'medium'` we can select the dielectric media within which points are kept, and with `'mindist'` we can set a minimum distance between the points and the particle boundaries. Below we give some examples for using `compoint` objects.

5.4. The `compstruct` class

Internally, the MNPBEM toolbox stores the scalar and vector functions as `compstruct` objects, which behave very much like Matlab `struct` objects, but must always hold a `comparticle` or `compoint` object `p` and the light wavelength `enei`

```
% set up a compstruct object
c = compstruct(p,enei);
% add fields to the object
c.val = val;
```

In many respects one can treat these objects as normal arrays. This means, we can add or subtract `compstruct` objects and we can multiply them with a constant value. In adding or subtracting them, fields that are missing in one of the objects are treated as zeros. Upon multiplication all fields of the `compstruct` object are multiplied with the same value. These features are particularly useful for `compstruct` object with electromagnetic potentials or fields that can be easily added or scaled.

6. BEM solvers

The quasistatic BEM solver is initialized with

```
% initialize BEM solver
bem = bemstat(p);
% initialization for given wavelength enei
bem = bemstat(p,enei);
% initialization passing arguments to the BEM solver
bem = bemstat(p,[],op);
```

For the solution of the full Maxwell equations we simply have to use the solver `bemret`. When we pass the argument `enei` to the BEM solver, the matrices of the BEM approach are computed for the given light wavelength. Alternatively, we can compute the matrices later through

```
bem = bem(enei);
```

For practically all BEM simulations this call will consume most of the computer time.

One can also pass options to the BEM solvers. In general, the BEM approach implemented within the **MNPBEM** toolbox uses a *collocation* scheme where the surface charges and currents σ and \mathbf{h} are assumed to be situated at the centroids of the boundary elements. More accurate results could be achieved by performing a linear interpolation of σ and \mathbf{h} within the boundary elements. However, so far we have refrained from such an interpolation because it would make the implementation of the BEM equations much more complicated. In addition, in comparison with the field-based BEM approaches, which invoke second-order surface derivatives of the Green function, the collocation scheme for the potential-based BEM approach is expected to be of the same order of accuracy as a linear-interpolation scheme for a field-based BEM approach. In some cases, e.g. when encountering the elongated surface elements of extruded particles, see Fig. 5, a pure collocation approach is problematic and it is better to assume that σ and \mathbf{h} are constant over the face elements. We then have to integrate in G_{ij} and its surface derivative over boundary elements j which are sufficiently close to the element i . In order to do so, one can define in the options

```
op = green.options('cutoff',cutoff);
```

a **cutoff** parameter that determines whether such a face integration is performed or the function value between the collocation points is taken. As will be discussed below, we can pass **op** also directly to **compgreen** objects.

6.1. Solving the BEM equations

In the following we examine the working principle of the BEM solvers for the planewave excitation discussed in Sec. 3

```
% plane wave excitation for given light polarizations
exc = planewavestat([1,0,0;0,1,0]);
% plane wave excitation
e = exc(p,enei);
```

In the last call we receive a **compstruct** object **e** which holds the field **phip** for the surface derivative $\frac{\partial \phi_{\text{ext}}}{\partial n}$ of the scalar potential. Similarly, in the retarded case the returned object contains fields for the scalar and vector potentials at the boundaries, together with their surface derivatives.

For this external excitation we can now compute the surface charges through

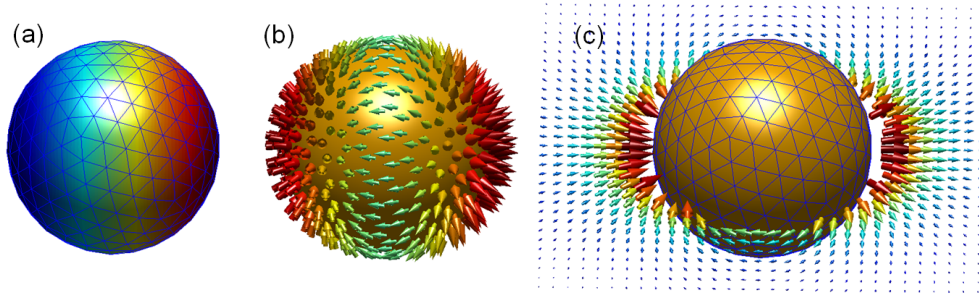


Figure 6: Results of BEM simulations: (a) surface charges σ_i , and real part of induced electric field at (b) particle boundary and (c) elsewhere. In the simulations (consult the demo file `demofieldstat.m` for details) we consider a plane wave excitation of a gold nanoparticle with a diameter of 10 nm embedded in water.

```
% initialize BEM solver
bem = bem(enei);
% compute surface charge
sig = bem \ e;
```

Alternatively, we can also put all commands into a single line

```
% set up BEM solver
bem = bemstat(p);
% initialize BEM solver and compute surface charge
sig = bem \ exc(p,enei);
```

This calling sequence is simpler but it has the disadvantage that the BEM matrices, whose computation is rather time consuming, are not stored in `bem` after the call. If we are only interested in a single type of excitation this is not a problem, but if we want to compute surface charges for the same wavelength `enei` but for a different excitation, e.g. dipole excitation, it is better to store the matrices through `bem=bem(enei)`.

We can now plot the surface charge `sig.sig` and the electric field at the particle boundary through

```
% plot real part of surface charge
plot(p,real(sig.sig),'EdgeColor','b');
% electric field at outside of particle boundary
field = bem.field(sig,2);
% plot real part of electric field
```

```
plot(p,'cone',real(field.e),'scale',0.6);
```

The results are shown in panels (a) and (b) of Fig. 6.

6.2. Green functions

Finally, we show how to proceed when we want to compute the electromagnetic fields or potentials elsewhere. To this end, we have to set up a `compgreen` object, which remains usually hidden within the BEM solvers. With `pt` a `compoint` object of the positions where the electromagnetic field should be computed and `p` a `comparticle` object for the particle boundaries, the Green function can be initialized with

```
% set up Green function between points and particle
g = compgreen(pt,p);
% same as above but with additional cutoff parameter
g = compgreen(pt,p,green.option('cutoff',cutoff));
```

With the help of the Green function we can compute according to Eq. (2) the induced fields and potentials everywhere else. Below we show the code needed in order to produce Fig. 6(c)

```
% regular mesh
[x,y] = meshgrid(linspace(-10,10,31));
pt = compoint(p,[x(:),y(:),0*x(:)],'mindist',1);
% set up Green function between mesh points and particle
g = compgreen(pt,p);
% compute electric field
field = g.field(sig);
% plot particle and real part of electric field
plot(p,'EdgeColor','b');
coneplot(pt.pos,real(field.e),'scale',0.6);
```

7. BEM Simulations

The MNPBEM toolbox provides two excitation-measurement schemes: one for planewave excitation and the calculation of the scattering and extinction cross sections, which we have already discussed in Sec. 3, and one for the excitation of an oscillating dipole and the calculation of the enhancement of the radiative and total scattering rates.

7.1. Quasistatic versus full BEM simulations

Let us look to the file `democputime.m`, available in the `Demo` subdirectory, which compares for the calculation of the scattering cross sections of a nanosphere the CPU times for different surface discretizations and for the different BEM solvers. In addition to `bemstat` and `bemret` we also show results for the `bemstateig` solver, which solves the BEM equations using a restricted number of eigemodes of the matrix $(\frac{\partial G}{\partial n})_{ij}$ [22, 31, 36]

CPU time elapsed for BEM simulations in seconds

#verts	#faces	bemstat	bemstateig	bemret
144	284	1.07	0.14	9.31
256	508	4.07	0.42	37.40
400	796	14.01	1.26	129.57
676	1348	62.24	2.97	651.25

From the results it is apparent that the simulations based on the full Maxwell equations are about a factor of ten slower than those based on the quasistatic approximation. An additional speedup can be achieved for the eigenmode expansion, in the above example we have used 20 eigenmodes.

The question which BEM solver is the most appropriate one cannot be answered in a unique way.

Quasistatic solvers. The quasistatic solvers `bemstat` and `bemstateig` are ideal for testing and getting a feeling of how the results will approximately look like, at least for structures which are significantly smaller than the light wavelength. It is a matter of taste what one calls “significantly smaller”, but metallic spheres with diameters below say 50 nm and flat or elongated particles with dimensions below 100 nm will probably do. If you are dealing with even smaller structures, with dimensions of a few tens of nanometers, the quasistatic approximation will probably work perfectly in all cases. However, we recommend to compare from time to time with the results of the full BEM solver `bemret`.

Full BEM solver. BEM simulations based on the full Maxwell equations are much slower than those performed with the quasistatic BEM solvers, the main reason being the numerous matrix inversions. For a given number N of particle faces, the time needed for a matrix inversion is of the order N^3 . For this reason it is good to keep the number of faces

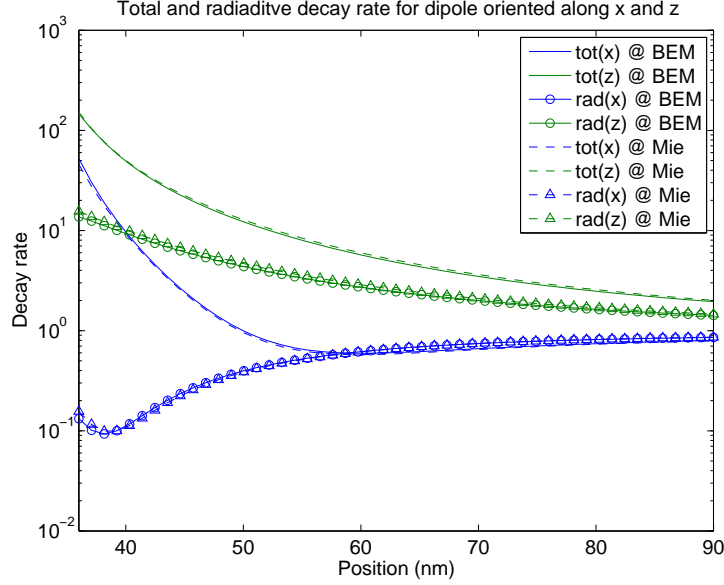


Figure 7: Enhancement of the total and radiative decay rates of a dipole located in the vicinity of a metallic nanosphere with a diameter of 60 nm. The dipole is located on the z axis at different distances from the sphere center. We use a dielectric function representative for gold [33] and a refractive index of $n_b = 1.33$ for the embedding medium. The results of the BEM simulation (`demospectrumret.m`) are almost indistinguishable from those of Mie theory.

and vertices as small as possible. Nevertheless, in many cases it is indispensable to solve the full Maxwell equations. Typical simulation times for the `bemret` solver are in the range between minutes and a few hours.

7.2. Dipole excitations

We next discuss how to simulate the excitation of an oscillating electric dipole located at some distance from a nanoparticle. This situation corresponds to the decay of an excited molecule or quantum dot, whose decay rates become enhanced through the modified photonic environment. Similarly, the dipole excitation can be also used to compute the dyadic Green function of Maxwell's theory (see below).

The class `dipolestat` allows for an implementation of this problem. Suppose that we have a `compoint` object `pt` with the positions of the dipoles. To implement a `dipolestat` object we call

```

% default dipole orientations x, y, z
dip = dipolestat(pt);
% dipole orientations x, z
dip = dipolestat(pt,[1,0,0;0,0,1]);
% user-defined dipole vectors for each position
dip = dipolestat(pt,vec,'full');

```

Once we have set up the dipole excitation, we can use it in combination with the quasistatic BEM solvers (for the simulation of the full Maxwell equations we simply have to use `dipoleret`, in order to compute both the scalar and vector potentials of the external dipole excitation, as well as `bemret`). The enhancement of the radiative and total decay rates, with respect to the free-space decay rate, is then computed with [6, 20]

```

% BEM simulation
sig = bem \ dip(p,enei );
% enhancement of total and radiative decay rate
[tot,rad] = dip.decayrate(sig);

```

Figure 7 shows the results of `demodipoleret.m` for an oscillating dipole located in the vicinity of a metallic nanosphere, as well as the comparison with Mie theory.

Finally, through the relation

$$\mathbf{E}(\mathbf{r}) = k^2 \mathbf{G}(\mathbf{r}, \mathbf{r}'; \omega) \cdot \mathbf{d} \quad (9)$$

we are in the position to compute the dyadic Green function $\mathbf{G}(\mathbf{r}, \mathbf{r}'; \omega)$ of Maxwell theory. To this end we place an oscillating dipole at position \mathbf{r}' , solve the BEM equations for a given wavenumber $k = \omega/c$, and finally compute the electric field at the positions \mathbf{r} according to the prescription given in Sec. 6.

7.3. Setting up a new excitation-measurement scheme

In some situations one might like to set up a new excitation-measurement scheme, e.g., to simulate a nearfield excitation or electron energy loss spectroscopy (EELS). Setting up a new excitation-measurement scheme can be done with a moderate amount of work. In the following we sketch how this should be done. Further information can be found in the help pages.

We recommend to use Matlab classes. An object `exc` of this class should return with `exc(p,enei)` the potentials (and their surface derivatives) for

the external perturbation at the particle boundary, which can be achieved by implementing the `subsref` function. To see how this can be done efficiently, we suggest to inspect the `planewave` and `dipole` classes of the toolbox. As for the measurement implementation, it probably suffices to set up a `compgreen` object and to compute with it the induced electromagnetic fields or potentials, which then can be further processed.

8. Summary and outlook

To summarize, we have presented a Matlab toolbox `MNPBEM` suited for the simulation of metallic nanoparticles (MNP) using a boundary element method (BEM) approach. The toolbox relies on the concept of Matlab classes which can be easily combined, such that one can adapt the simulation programs flexibly for various applications. All technicalities of the potential-based BEM approach remain hidden within the classes. The toolbox provides detailed help pages and a collection of demo programs. Several plot commands allow to access the full potential of the Matlab program and facilitate the analysis and interpretation of the simulation results.

As regarding simulation time and accuracy, we believe that the toolbox performs well and can compete with the other simulation toolkits used in the plasmonics community, although more detailed tests are needed for clarification. There is plenty of room for improvements, such as multigrid methods or interpolation schemes beyond the presently used collocation scheme. We are presently testing the implementation of mirror symmetry, which can speed up simulations by about one order of magnitude, as well as of particles placed on substrates or embedded in layer structures. Both features are working fine, but we have not included them in this version of the toolbox because they are still somewhat experimental. Future work will also address periodic structures and static electric fields, such as needed for electrochemistry. Altogether, we hope that the `MNPBEM` toolbox will serve the plasmonics community as a useful and helpful simulation toolkit.

Acknowledgment

We thank the `nanooptics` group in Graz for most helpful discussions. This work has been supported in part by the Austrian science fund FWF under project P21235–N20.

References

- [1] H. Atwater, The promise of plasmonics, *Scientific American* 296(4) (2007) 56.
- [2] S. A. Maier, *Plasmonics: Fundamentals and Applications*, Springer, Berlin, 2007.
- [3] J. B. Pendry, D. Schurig, D. R. Smith, *Science* 312 (2006) 1780.
- [4] J. A. Schuller, E. S. Barnard, W. Cai, Y. C. Jun, J. S. White, M. L. Brongersma, *Nature Materials* 9 (2010) 193.
- [5] L. Novotny, N. van Hulst, *Nature Photonics* 5 (2011) 83.
- [6] L. Novotny, B. Hecht, *Principles of Nano-Optics*, Cambridge University Press, Cambridge, 2006.
- [7] U. Kreibig, M. Vollmer, *Optical properties of metal clusters*, Vol. 25 of Springer series in material science, Springer, Berlin, 1995.
- [8] R. R. Chance, A. Prock, R. Silbey, *Adv. Chem. Phys.* 37 (1978) 1.
- [9] K. Kneipp, Y. Wang, H. Kneipp, L. T. Perelman, I. Itzkan, R. R. Dasari, M. S. Feld, *Phys. Rev. Lett.* 78 (1997) 1667.
- [10] S. Nie, S. R. Emory, *Science* 275 (1997) 1102.
- [11] P. Anger, P. Bharadwaj, L. Novotny, *Phys. Rev. Lett.* 96 (2006) 113002.
- [12] B. T. Draine, *Astrophys. J.* 333 (1988) 848.
- [13] B. T. Draine, P. J. Flateau, *J. Opt. Soc. Am. A* 11 (1994) 1491.
- [14] K. S. Yee, *IEEE Trans. on Antennas and Propagation* 14 (1966) 302.
- [15] A. J. Ward, J. B. Pendry, *Comp. Phys. Commun.* 128 (2000) 590.
- [16] O. J. F. Martin, C. Girard, A. Dereux, *Phys. Rev. Lett.* 74 (1995) 526.
- [17] F. J. Garcia de Abajo, A. Howie, *Phys. Rev. B* 65 (2002) 115418.
- [18] F. J. G. de Abajo, *Rev. Mod. Phys.* 82 (2010) 209.

- [19] U. Hohenester, J. R. Krenn, Phys. Rev. B 72 (2005) 195429.
- [20] U. Hohenester, A. Trügler, IEEE J. of Selected Topics in Quantum Electronics 14 (2008) 1430.
- [21] A. Trügler, U. Hohenester, Phys. Rev. B 77 (2008) 115403.
- [22] A. Trügler, J. C. Tinguely, J. R. Krenn, A. Hohenau, U. Hohenester, Phys. Rev. B 83 (2011) 081412(R).
- [23] S. Gerber, F. Reil, U. Hohenester, T. Schlagenhaufen, J. R. Krenn, A. Leitner, Phys. Rev. B 75 (2007) 073404.
- [24] F. Reil, U. Hohenester, J. Krenn, A. Leitner, Nano Lett. 8 (2008) 4128.
- [25] D. Koller, U. Hohenester, A. Hohenau, H. Ditlbacher, F. Reil, N. Galler, F. Aussenegg, A. Leitner, A. Trügler, J. Krenn, Phys. Rev. Lett. 104 (2010) 143901.
- [26] J. Becker, A. Trügler, A. Jakab, U. Hohenester, C. Sönnichsen, Plasmonics 5 (2010) 161.
- [27] A. Jakab, Y. Khalavka, J. Becker, A. Trügler, C. Rosman, U. Hohenester, and C. Sönnichsen, to appear in ACS Nano (2011).
- [28] B. Schaffer, U. Hohenester, A. Trügler, F. Hofer, Phys. Rev. B 79 (2009) 041401(R).
- [29] U. Hohenester, H. Ditlbacher, J. Krenn, Phys. Rev. Lett. 103 (2009) 106801.
- [30] J. D. Jackson, Classical Electrodynamics, Wiley, New York, 1999.
- [31] R. Fuchs, Phys. Rev. B 11 (1975) 1732.
- [32] W. C. Chew, Waves and fields in inhomogeneous media, IEEE Press, Piscataway, 1995.
- [33] P. B. Johnson, R. W. Christy, Phys. Rev. B 6 (1972) 4370.
- [34] H. C. van de Hulst, Light Scattering by Small Particles, Dover, New York, 1981.

- [35] R. Fuchs, S. H. Liu, Phys. Rev. B 14 (1976) 5521.
- [36] I. D. Mayergoyz, D. R. Fredkin, Z. Zhang, Phys. Rev. B 72 (2005) 155412.