User Manual for the Discrete Dipole Approximation Code ADDA 1.2

Maxim A. Yurkin

Institute of Chemical Kinetics and Combustion SB RAS, Institutskaya 3, 630090, Novosibirsk, Russia
and
Novosibirsk State University, Pirogova 2, 630090, Novosibirsk, Russia

Alfons G. Hoekstra

Computational Science Research Group, Faculty of Science, University of Amsterdam,
Science Park 904, 1098 XH, Amsterdam, The Netherlands

email: adda-discuss@googlegroups.com

last revised: June 6, 2013

Abstract

This manual describes the open-source code ADDA, which simulates elastic light scattering from finite 3D objects of arbitrary shape and composition. Besides standard sequential execution, ADDA can run on a multiprocessor system, using MPI (message passing interface), parallelizing a *single* DDA calculation. Hence the size parameter of the scatterer, which can be accurately simulated, is limited only by the available size of the supercomputer. However, if the refractive index is large compared to 1, computational requirements significantly increase. Moreover, ADDA can effectively employ modern GPUs (video cards) to accelerate computations.

ADDA is written in C99 (using routines in Fortran and C++) and is highly portable. It provides full control over the scattering geometry (particle morphology and orientation, incident beam) and allows one to calculate a wide variety of integral and angle-resolved scattering quantities (cross sections, the Mueller matrix, etc.). Moreover, ADDA incorporates a range of state-of-the-art DDA improvements, aimed at increasing the accuracy and computational speed of the method. This manual explains in details how to perform electromagnetic scattering calculations using ADDA, discussing both physical and computational aspects.

This manual can be cited as:
M.A. Yurkin and A.G. Hoekstra "User manual for the discrete dipole approximation code ADDA 1.2", http://a-dda.googlecode.com/svn/tags/rel_1.2/doc/manual.pdf (2013).

The source of this manual in Microsoft Word format is available at
http://a-dda.googlecode.com/svn/tags/rel_1.2/doc/manual.doc

Parts of this manual were published in [1].

Contents

1 Introduction

The discrete dipole approximation (DDA) is a general method to calculate scattering and absorption of electromagnetic waves by particles of arbitrary geometry. In this method the volume of the scatterer is divided into small cubical subvolumes ("dipoles"). Dipole interactions are approximated based on the integral equation for the electric field [2]. Initially the DDA (sometimes referred to as the "coupled dipole approximation") was proposed by Purcell and Pennypacker [3] replacing the scatterer by a set of point dipoles (hence the name of the technique). Although the final equations are essentially the same, derivations based on the integral equations give more mathematical insight into the approximation, while the model of point dipoles is physically clearer. For an extensive review of the DDA, including both theoretical and computational aspects, the reader is referred to [2] and references therein.

ADDA is a C implementation of the DDA. The development was conducted by Hoekstra and coworkers [4–7] since 1990 at the University of Amsterdam. From the very beginning the code was intended to run on a multiprocessor system or a multicore processor (parallelizing a *single* DDA simulation).[1] The code was significantly rewritten and improved by Yurkin [8], also at the University of Amsterdam, and publicly released in 2006. Since then ADDA is the open-source code developed by an international team.[2] Originally coined "Amsterdam DDA", the code name has been officially abbreviated to reflect this change.

ADDA is intended to be a versatile tool, suitable for a wide variety of applications ranging from interstellar dust and atmospheric aerosols to biological cells and nanoparticels; its applicability is limited only by available computer resources (§4). As provided, ADDA should be usable for many applications without modification, but the program is written in a modular form, so that modifications, if required, should be fairly straightforward.[3]

This code is openly available to others in the hope that it will prove a useful tool. We ask only that:

- If you publish results obtained using ADDA, you should acknowledge the source of the code. A general reference [1] is recommended for that.[4]
- If you discover any errors in the code or documentation, please submit it to the ADDA issue tracker.[5]
- You comply with the "copyleft" agreement (more formally, the GNU General Public License v.3[6]) of the Free Software Foundation: you may copy, distribute, and/or modify the software identified as coming under this agreement. If you distribute copies of this software, you must give the recipients all the rights which you have. See the file `doc/copyleft` distributed with the ADDA software.

We strongly encourage you to identify yourself as a user of ADDA by subscribing to `adda-announce` Google group;[7] this will enable the developers to notify you of any bugs, corrections, or improvements in ADDA. If you have a question about ADDA, which you think is common, please look into the FAQ[8] before searching for the answer in this manual. If neither of these helps, direct your questions to `adda-discuss` Google group.[9] The archive of this

[1] http://code.google.com/p/a-dda/wiki/EarlyHistory
[2] http://code.google.com/p/a-dda/people/list
[3] However, in some parts modularity was sacrificed for the sake of performance. E.g. iterative solvers (§11.1) are implemented not to perform any unnecessary operations (which usually happens when using standard libraries).
[4] See http://code.google.com/p/a-dda/wiki/References for more specific references.
[5] http://code.google.com/p/a-dda/issues
[6] http://www.gnu.org/copyleft/gpl.html
[7] To do this send an e-mail to adda-announce+subscribe@googlegroups.com
[8] http://code.google.com/p/a-dda/wiki/FAQ
[9] Just send an e-mail to adda-discuss@googlegroups.com

group is available,[10] so you may try to find an answer to your question before posting it. However, this also means that you should *not* include (potentially) confidential information in your message; in that case contact one of the developers directly. We also advise users, interested in different aspects of ADDA usage, to join this group and to participate in discussions initiated by other people.[11]

2 How to Use the Manual

This manual is intended to cover the computational and physical aspects of ADDA, i.e. choosing proper values for input parameters, performing the simulations, and analyzing the results. In particular, the succeeding sections contain instructions for:

- running a sample simulation (§3);
- defining a scatterer (§6) and its orientation (§7);
- specifying the type and propagation direction of the incident beam (§8);
- specifying the DDA formulation (§9);
- specifying what scattering quantities should be calculated (§10);
- understanding the computational aspects (§11) and timing of the code (§12);
- understanding the command line options (§A) and formats of input (§B) and output (§C) files.

A lot of technical aspects are covered by online wiki pages, which are extensively referred to throughout the manual. This manual assumes that you have already obtained the executable for ADDA. You can either download the full source code and compile it yourself,[12] or download precompiled executable for some operating systems.[13] Both source and executable packages of the recent release can be downloaded from:

`http://code.google.com/p/a-dda/downloads`

If you want to use the latest features or track the development progress, you may download the recent source directly from the ADDA subversion repository using web browser[14] or any subversion client.[15] Please mind, however, that the latest between-releases version may be unstable. If you decide to try it and discover any problems, comment on this in the issue tracker.[5]

Everywhere in this manual, as well as in input and output files, it is assumed that all angles are in degrees (unless explicitly stated differently). The unit of length is assumed μm; however it is clear that it can be any other unit, if all dimensional values are scaled accordingly. However, scaling of units of radiation forces is more subtle (see §10.4).

3 Running ADDA

3.1 Sequential mode

The simplest way to run ADDA is to type

`adda`[16]

[10] `http://groups.google.com/group/adda-discuss/topics`

[11] To do this send an e-mail to `adda-discuss+subscribe@googlegroups.com`

[12] See `http://code.google.com/p/a-dda/wiki/CompilingADDA` for compiling instructions.

[13] See `http://code.google.com/p/a-dda/wiki/PackageDescription` for available packages.

[14] `http://code.google.com/p/a-dda/source/browse/#svn/trunk`

[15] `svn checkout http://a-dda.googlecode.com/svn/trunk/`

[16] If current directory is not in the PATH system variable you should type "`./adda`". It may also differ on non-Unix systems, e.g. under Windows you should type "`adda.exe`". Moreover, the name of executable is different for MPI (§3.2) and OpenCL versions (§3.3). This applies to all examples of command lines in this manual.

while positioned in a directory, where the executable is located. ADDA will perform a sample simulation (sphere with size parameter 3.367, refractive index 1.5, discretized into 16 dipoles in each direction) and produce basic output (§10, §C). The output directory and terminal output (`stdout`) should look like examples that are included in the distribution: `sample/run000_sphere_g16_m1.5` and `sample/stdout` respectively. ADDA takes most information specifying what and how to calculate from the command line, so the general way to call ADDA is

```
adda -<par1> <args1> -<par2> <args2> …
```

where `<par>` is an option name (starting with a letter), and `<args>` is none, one, or several arguments (depending on the option), separated by spaces. `<args>` can be both text or numerical. How to control ADDA by proper command line options is thoroughly described in the following sections; the full reference list is given in §A. If you prefer a quick hands-on start with ADDA before looking at all possible options, you are welcome to a tutorial.[17] Quick help is available by typing

```
adda -h
```

For some options input files are required, they are described in §B. It is recommended to copy the contents of the directory `input/` of the package (examples of all input files that are silently used) to the directory where ADDA is executed. All the output produced by ADDA is described in §C. Version of ADDA, compiler used to build it,[18] width of memory access registers (32 or 64 bits), build options,[12] and copyright information are available by typing

```
adda –V
```

3.2 Parallel mode

On different systems MPI is used differently, you should consult someone familiar with MPI usage on your system. However, running on a multi-core PC is simple, just type

```
mpiexec –n <N> ./adda_mpi …
```

where all ADDA command line options are specified in the end of the line, and `<N>` stands for the number of cores. Actually, `<N>` is the number of threads created by MPI implementation, and it should not necessarily be equal to the number of cores. However, this choice is recommended. `MPICH2` allows one to force local execution of all threads by additionally specifying command line option `-localonly` after `<N>`. However, this may be different for other MPI implementations.

Running on a cluster is usually not that trivial. For example, on many parallel computers, PBS (portable batch system)[19] is used to schedule jobs. To schedule a job one should first write a shell script and then submit it. An example of such PBS script is included in the distribution (`sample/test.pbs`). It is important to note that usually output of the batch job (both `stdout` and `stderr`) is saved to file. The name of the file usually contains the job id number given by the system. The same number appears in the directory name (§C.3).

Another batch system is SGE (Sun grid engine).[20] We do not give a description of it here, but provide a sample script to run ADDA using SGE (`sample/test.sge`). One can easily modify it for a particular task.

[17] http://code.google.com/p/a-dda/wiki/Tutorial
[18] Only a limited set of compilers is recognized (currently: Borland, Compaq, GNU, Intel, Microsoft).
[19] http://www.openpbs.org/
[20] http://gridengine.sunsource.net/

3.3 OpenCL (GPU) mode

This mode is very similar to the sequential mode, except `ADDA` executable is named `adda_ocl` and part of the calculations is carried on the GPU [9]. Therefore a GPU, supporting double precision calculations, is required as well as recent drivers for it. Currently, `ADDA` can use only a single GPU, and will choose the first (default) one, if several GPUs are available. To use another GPU, specify its index (starting from zero) through the command line option

```
-gpu <index>
```

Therefore, several instances of `adda_ocl` can be run in parallel, each using one CPU core and one GPU.

More details about the OpenCL implementation, including existing limitations are described in the corresponding wiki page.[21] While the developers do their best to ensure reliability of OpenCL mode, it is a good idea to perform selected tests of this mode against the sequential mode on your specific hardware before performing a large set of simulations.

4 Applicability of the DDA

4.1 General applicability

The principal advantage of the DDA is that it is completely flexible regarding the geometry of the scatterer, being limited only by the need to use a dipole size d small compared to both any structural length in the scatterer and the wavelength λ. A large number of studies devoted to the accuracy of DDA results exist, e.g. [8,10–19]. Most of them are reviewed in [2]; here we only give a brief overview.

The rule of thumb for particles with size comparable to the wavelength is: "10 dipoles per wavelength inside the scatterer", i.e. size of one dipole is

$$d = \lambda/10|m|, \tag{1}$$

where m is refractive index of the scatterer. That is the default for `ADDA` (§6.2). The expected accuracy of cross sections is then several percents (for moderate m, see below). With increasing m the number of dipoles that is used to discretize the particle increases; moreover, the convergence of the iterative solver (§11.1) becomes slower. Additionally, the accuracy of the simulation with default dipole size deteriorates, and smaller, hence more dipoles must be used to improve it. Therefore, it is accepted that the refractive index should satisfy

$$|m-1| < 2. \tag{2}$$

Larger m can also be simulated accurately. In that case, however, the required computer resources rapidly increase with m. Fortunately, state-of-the-art DDA formulations (§9) can alleviate this problem and render larger refractive indices accessible to DDA simulations. Note however that the application of the DDA in this large-m regime is investigated much less thoroughly than for moderate refractive indices, and therefore warrants further studies.

When considering larger scatterers (volume-equivalent size parameter $x > 10$) the rule of thumb still applies. However, it does not describe well the dependence on m. When employing the rule of thumb, errors do not significantly depend on x, but do significantly increase with m [8]. However, simulation data for large scatterers is also limited; therefore, it is hard to propose any simple method to set dipole size. The maximum reachable x and m are mostly determined by the available computer resources (§5).

The DDA is also applicable to particles smaller than the wavelength, e.g. nanoparticles. In some respects, it is even simpler than for larger particles, since many convergence problem for large m are not present for small scatterers. However, in this regime there is an additional

[21] http://code.google.com/p/a-dda/wiki/OpenCL

requirement for d – it should allow for an adequate description of the shape of the particle. Although this requirement is relevant for any scatterers, it is usually automatically satisfied for larger scatterers by Eq. (1). For instance, for a sphere (or similar compact shape) it is recommended to use at least 10 dipoles along the smallest dimension, no matter how small is the particle. Smaller dipoles are required for irregularly shaped particles and/or large refractive index. The accuracy of the DDA for gold nanoparticles was studied in [20].

To conclude, it is hard to estimate *a priori* the accuracy of DDA simulation for a particular particle shape, size, and refractive index, although the papers cited above do give a hint. If one runs a single DDA simulation, there is no better alternative than to use rule of thumb and hope that the accuracy will be similar to that of the spheres, which can be found in one of the benchmark papers (e.g. [8,13,18]). However, if one plans a series of simulations for similar particles, especially outside of the usual DDA application domain [Eq. (2)], it is highly recommended to perform an accuracy study. For that one should choose a single test particle and perform DDA simulations with different d, both smaller and larger than proposed by the rule of thumb. The estimate of d required for a particular accuracy can be obtained from a variation of results with decreasing d. You may also make the estimation much more rigorous by using an extrapolation technique, as proposed by Yurkin *et al.* [21] and applied in [20,22,23].

Finally, it is important to note that the price paid for versatility of the DDA is its large computational costs, even for "simple" scatterers. Thus, in certain cases other (more specialized) methods will clearly be superior to the DDA. A review of relevant comparative studies is given in [2]. Additionally, it was recently shown that the DDA performs exceptionally well for large index-matching particles (e.g. biological cells in a liquid medium). In this regime the DDA is 10 to 100 times faster than a (general-purpose) finite-difference time-domain method when required to reach the same accuracy [22], and is comparable in speed to the discrete sources method for red blood cells [24], where the latter method explicitly employs the axisymmetry of the problem.

4.2 Extensions of the DDA

In its original form the DDA is derived for finite particles (or a set of several finite particles) in vacuum. However, it is also applicable to finite particles embedded in a homogeneous non-absorbing dielectric medium (refractive index m_0). To account for the medium one should replace the particle refractive index m by the relative refractive index m/m_0, and the wavelength in the vacuum λ by the wavelength in the medium λ/m_0. All the scattering quantities produced by DDA simulations with such modified m and λ are then the correct ones for the initial scattering problem. The only exception is the radiative force per each dipole which should be additionally scaled in the presence of the medium (see §10.4).

ADDA can not be directly applied to infinite scatterers. In particular, it *cannot* be applied to particles located near an infinite dielectric plane surface or, more generally, particles above or inside a substrate of finite or infinite width. Although a modification of the DDA is known to rigorously solve this problem [25–29], it requires principal changes in the DDA formulation and hence in the internal structure of the computer code. However, the current version of ADDA still provides an opportunity to solve this proble:

- One could consider the substrate only by its influence on the incident field (by adding a reflected wave). This may be accurate enough if the particle is far from the substrate and the contrast between the substrate and the upper medium is small. The corrected incident field can be easily obtained analytically for infinite plane-parallel substrate. Unfortunately, currently such incident field is not implemented, and need first to be coded into ADDA (§8.2).

- Another approach is to take a large computational box around the particle, and explicitly discretize the substrate that falls into it [30]. This is rigorous in the limit of infinite size of computational box, but requires much larger computer resources than that for the initial problem. It also requires one to specify the shape files for all different sizes of the computational box.

The problem with the latter approach is the diffraction of the plane wave on the edges of the computational domain. It can be alleviated by using a Gaussian beam with a width smaller than the computational domain but larger than all structural features of the problem (particles above or inhomogeneities inside the substrate) [31].

A combination of these two approaches was proposed by D'Agostino *et al.* [32] to decrease spurious boundary effects. The total near- of far-field $\mathbf{E}(\mathbf{r})$ is replaced by an adjusted field $\mathbf{E}^{\mathrm{adj}}(\mathbf{r})$

$$\mathbf{E}^{\mathrm{adj}}(\mathbf{r}) = \mathbf{E}(\mathbf{r}) - \mathbf{E}^{\mathrm{sub}}(\mathbf{r}) + \mathbf{E}^{\mathrm{inc}}(\mathbf{r}) + \mathbf{E}^{\mathrm{ref}}(\mathbf{r}), \tag{3}$$

where $\mathbf{E}^{\mathrm{sub}}(\mathbf{r})$ is the result of a DDA simulation for the truncated substrate alone (without particles or inhomogeneities). This technique was proposed and tested for nanoparticles above metallic layers, but it could also be useful for other problems that fall under the general description given above. In other words, it is expected that $\mathbf{E}^{\mathrm{adj}}(\mathbf{r})$ will converge to the correct solution with increasing computational domain faster than $\mathbf{E}(\mathbf{r})$.

Another useful extension of the DDA is introduction of periodic boundary conditions [33,34], which is relevant to photonic crystals and similar applications. This requires relatively simple modification of the algorithm and it was recently implemented in the `DDSCAT` 7 [35]. However, `ADDA` does *not* yet support this feature.

5 System Requirements

Computational requirements of DDA primarily depend on the size of the computational grid, which in turn depends on the size parameter x and refractive index m of the scatterer (§6.2). This section addresses requirements of standard (FFT) mode of `ADDA`, while sparse mode is discussed separately (§11.3). The memory requirements of `ADDA` depend both on the total number of dipoles in a computationxal box (N) and the number of real (non-void) dipoles (N_{real}); it also depends on the number of dipoles along the x-axis (n_x) and number of processors or cores used (n_{p}). The total memory requirement M_{tot} (for all processors) is approximately

$$M_{\mathrm{tot}} = \left[288 + 384\, n_{\mathrm{p}}/n_x \ \left(+192/n_{\mathrm{p}}\right)\right] N + \left[271 \div 463\right] N_{\mathrm{real}} \text{ bytes}, \tag{4}$$

where additional memory (in round brackets) proportional to N is required only in parallel mode (see §11.2 for details). Coefficient before N_{real} depends on the chosen iterative solver, as $31 + 48(n_{\mathrm{vec}} + 1)$, where n_{vec} is the number of vectors used by the solver (§11.1, Table 2). The memory requirements of each processor depends on the partition of the computational grid over the processors that is generally not uniform (see §6.6). Total memory used by `ADDA` and maximum per one processor are shown in `log` (see §C.4). It is important to note that *double* precision is used everywhere in `ADDA`. This requires more memory as compared to single precision, but it helps when convergence of the iterative solver is very slow and machine precision becomes relevant, as is the case for large simulations, or when very accurate results are desired, as in [21].

There is a maximum number of processors, which `ADDA` can effectively employ (§6.6), equal to n_z if the latter is a "round" number (§11.2). This determines the largest problem size solvable on a given supercomputer with very large number of processors but with a limited

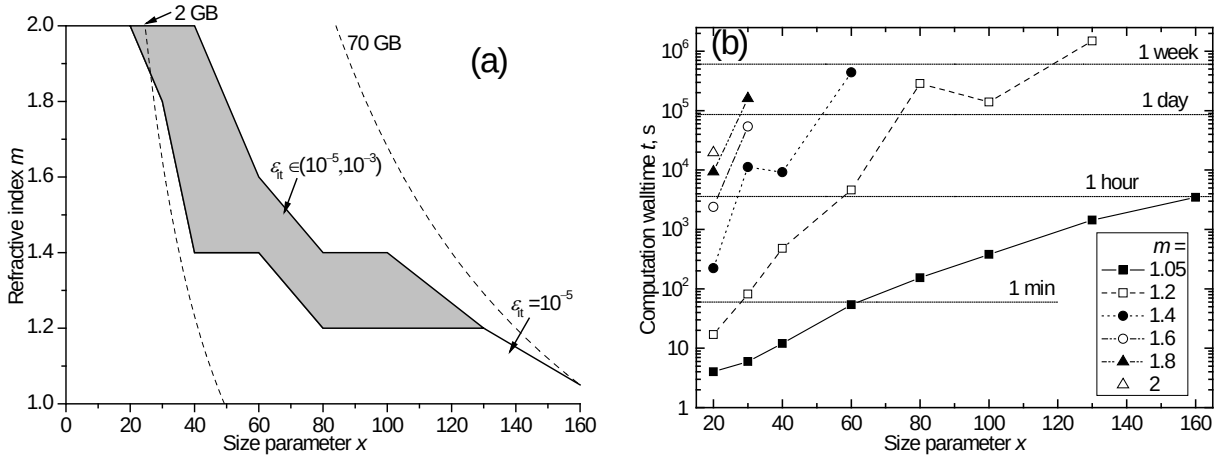

Fig. 1. Capabilities of ADDA 0.76 for spheres with different x and m. (a) Applicability region – the lower-left region corresponds to full convergence and gray region – to incomplete convergence. The dashed lines show two levels of memory requirements for the simulation. (b) Total simulation wall clock time on 64 processors in logarithmic scale. Horizontal dotted lines corresponding to a minute, an hour, a day, and a week are shown for convenience. Adapted from [8].

amount of memory per processor M_{pp}. For a given problem, setting $n_{\text{p}} = n_z$ leads to the following maximum memory requirements per processor:[22]

$$M_{\text{pp}} = \left[288 n_x + 384 n_z + 192\, n_x / n_z\right] n_y + \left[271 \div 463\right] n_{\text{slice}} \text{ bytes}, \tag{5}$$

where $n_{\text{slice}} \le n_x n_y$ is the maximum number of real dipoles in a slice parallel to the yz-plane.

To have a quick estimate of maximum achievable discretization on a given hardware, one may consider a cube and less-memory-consuming iterative solvers. Then, Eqs. (4) and (5) lead to maximum n_x equal to $113[M_{\text{tot}}(\text{GB})]^{1/3}$ and $1067[M_{\text{pp}}(\text{GB})]^{1/2}$ for single-core PC and very large cluster respectively.

In OpenCL mode the memory proportional to N in Eq. (4) is moved to the GPU thus reducing the used amount of main (CPU) memory. In this case additional $8N_{\text{real}}$ bytes of CPU memory is required for CGNR and Bi-CGStab iterative solvers.

ADDA may optimize itself during runtime for either maximum speed or minimum memory usage. It is controlled by a command line option

```
-opt {speed|mem}
```

By default, speed optimization is used. Currently the difference in performance is very small, but it will increase during future ADDA development. A command line option

```
-prognosis
```

can be used to estimate the memory requirements without actually performing the allocation of memory and simulation.[23] It also implies `-test` option (§C.3).

Simulation time (see §11 for details) consists of two major parts: solution of the linear system of equations and calculation of the scattered fields. The first one depends on the number of iterations to reach convergence, which mainly depends on the size parameter, shape and refractive index of the scatterer, and time of one iteration, which depends only on N as $O(N\ln N)$ (see §11.2). Execution time for calculation of scattered fields is proportional to N_{real}, and is usually relatively small if scattering is only calculated in one plane. However, it may be significant when a large grid of scattering angles is used (§10.1, §10.3). Employing multiple processors brings the simulation time down almost proportional to the number of

[22] This limit can be circumvented on a supercomputer consisting of nodes with several processors (cores) sharing the same memory. Using only part (down to 1) of the processors per node increases available memory per processor at the expense of wasted system resources, which are usually measured in node-hours.

[23] Currently this option does need a certain amount of RAM, about $11(N + N_{\text{real}})$ bytes. It enables saving of the particle geometry in combination with `-prognosis`.

processors (see §11.4). To facilitate very long simulations checkpoints can be used to break a single simulation into smaller parts (§11.5).

For example, on a desktop computer (P4-3.2 GHz, 2 Gb RAM) it was possible to simulate light scattering by spheres[24] up to $x = 35$ and 20 for $m = 1.313$ and 2.0 respectively (simulation times are 20 and 148 hours respectively). The capabilities of ADDA 0.76 for simulation of light scattering by spheres using 64 3.4 GHz cores were reported in [8]. Here we present only Fig. 1, showing the maximum reachable x versus m and simulation time versus x and m. In particular, light scattering by a homogenous sphere with $x = 160$ and $m = 1.05$ was simulated in only 1.5 hours, although the runtime steeply increased with refractive index. Examples of more recent and even larger simulations are gathered on a special wiki page.[25]

6 Defining a Scatterer

6.1 Reference frames

Three different reference frames are used by ADDA: laboratory, particle, and incident wave reference frames. The laboratory reference frame is the default one, and all input parameters and other reference frames are specified relative to it. ADDA simulates light scattering in the particle reference frame, which naturally corresponds to particle geometry and symmetries, to minimize the size of the computational grid (§6.2), especially for elongated or oblate particles. In this reference frames the computational grid is built along the coordinate axes. The incident wave reference frame is defined by setting the z-axis along the propagation direction. All scattering directions are specified in this reference frame.

The origins of all reference frames coincide with the center of the computational grid (§6.2). By default, both particle and incident wave reference frames coincide with the laboratory frame. However, they can be made different by rotating the particle (§7) or by specifying a different propagation direction of the incident beam (§8.1) respectively.

6.2 The computational grid

ADDA embeds a scatterer in a rectangular computational box, which is divided into identical cubes (as required for the FFT acceleration, §11.2). Each cube is called a "dipole"; its size should be much smaller than a wavelength. The flexibility of the DDA method lies in its ability to naturally simulate the scattering of any arbitrarily shaped and/or inhomogeneous scatterer, because the optical properties (refractive index, §6.3) of each dipole can be set independently. There are a few parameters describing the computational grid: size of one dipole (cube) d, number of dipoles along each axis n_x, n_y, n_z, total size (in μm) of the grid along each axis D_x, D_y, D_z, volume-equivalent radius r_{eq}, and incident wavelength λ. However, they are not independent. ADDA allows one to specify all three grid dimensions n_x, n_y, n_z as arguments to the command line option

```
-grid <nx> [<ny> <nz>]
```

If omitted n_y and n_z are automatically determined by n_x based on the proportions of the scatterer (§6.4). When particle geometry is read from a file (§6.3) all grid dimensions are initialized automatically.[26] Because of the internal structure of the ADDA all the dimensions are currently limited to be even. If odd grid dimension is specified by any input method, it is automatically incremented. In this case ADDA produces a warning to avoid possible ambiguity.

[24] Only one incident polarization was calculated, execution time for non-symmetric shapes (§6.7) will be at least twice larger.

[25] http://code.google.com/p/a-dda/wiki/LargestSimulations

[26] Specifying all three dimensions (or even one when particle geometry is read from file) make sense only to fix these dimensions (larger than optimal) e.g. for performance studies.

If the `-jagged` option is used the grid dimension is effectively multiplied by the specified number (§6.3).

One can also specify the size parameter of the entire grid kD_x (k is the free space wave vector) or $x = kr_{ef}$, using three command line options:

```
-lambda <arg>
-size <arg>
-eq_rad <arg>
```

which specify (in μm) λ, D_x, and r_{ef} respectively. By default $\lambda = 2\pi$ μm, then `-size` determines kD_x and `-eq_rad` sets x. The last two are related by

$$x = kD_x \sqrt[3]{3 f_{vol} / 4\pi}\,, \tag{6}$$

where f_{vol} is ratio of particle to computational grid volumes, which is known analytically for many shapes available in ADDA (§6.4). It is important to note that D_x denotes the size of *possibly adjusted* computational grid. Although ADDA will warn user of possible ambiguities, it is not recommended to use `-size` command line option for shapes read from file, when inherent n_x of this shape is not guaranteed to be even. It may cause a modeled scatterer to be slightly smaller than originally intended. Some shapes define the absolute particle size themselves (§6.4). However, the size given in the command line (by either `-size` or `-eq_rad`) overrides the internal specification and the shape is scaled accordingly.

The size parameter of the dipole is specified by the parameter "dipoles per lambda" (dpl)

$$\mathrm{dpl} = \lambda / d = 2\pi / (kd)\,, \tag{7}$$

which is given to the command line option

```
-dpl <arg>
```

dpl does not need to be an integer; any real number can be specified.

ADDA will accept at most two parameters from: dpl, n_x, kD_x, and x since they depend on each other by Eq. (6) and

$$kD_x \cdot \mathrm{dpl} = 2\pi \cdot n_x\,. \tag{8}$$

Moreover, specifying a pair of kD_x and x is also not possible. If any other pair from these four parameters is given on the command line (n_x is also defined if particle geometry is read from file) the other two are automatically determined from the Eqs. (6) and (8). If the latter is n_x, dpl is slightly increased (if needed) so that n_x exactly equals an even integer. There is one exception: a pair of x and dpl can only be used for shapes, for which f_{vol} can be determined analytically (§6.4), because numerical evaluation of f_{vol} is only possible when particle is already discretized with a certain n_x. If less than two parameters are defined dpl or/and grid dimension are set by default.[27] The default for dpl is $10|m|$ [cf. Eq. (1)], where m is the maximum (by absolute value) refractive index specified by the `-m` option (or the default one, §6.3). The default for n_x is 16 (possibly multiplied by `-jagged` value). Hence, if only `-size` or `-eq_rad` is specified, ADDA will automatically discretize the particle, using the default dpl (with the exception discussed above for x). This procedure may lead to very small n_x, e.g. for nanoparticles, hence ADDA ensures that n_x is at least 16, when it is auto-set from default dpl. However, if dpl is specified in the command line, ADDA puts absolute trust in it and leaves all the consequences to the user.

6.3 Construction of a dipole set

After defining the computational grid (§6.2) each dipole of the grid should be assigned a refractive index (a void dipole is equivalent to a dipole with refractive index equal to 1). This

[27] If dpl is not defined, it is set to the default value. Then, if still less than two parameters are initialized, grid dimension is also set to the default value.

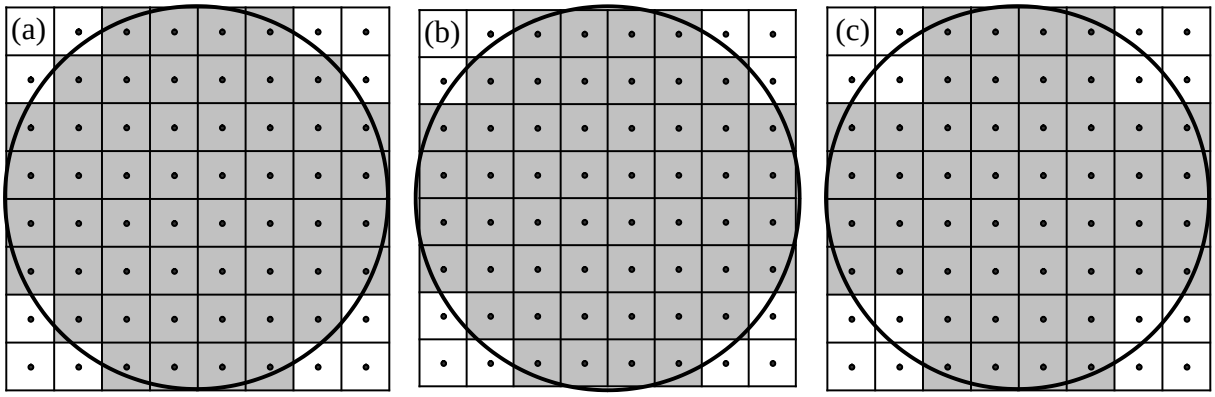

Fig. 2. An example of dipole assignment for a sphere (2D projection). Assigned dipoles are gray and void dipoles are white. (a) initial assignment; (b) after volume correction; (c) with "`-jagged`" option enabled ($J = 2$) and the same total grid dimension. Adapted from [1].

can be done automatically for a number of predefined shapes or in a very flexible way by specifying scatterer geometry in a separate input file. For predefined shapes (§6.4) the dipole is assigned to the scatterer if and only if its center falls inside the shape, see Fig. 2(a) for an example. When the scatterer consists of several domains, e.g. a coated sphere, the same rule applies to each domain. By default, `ADDA` slightly corrects the dipole size (or equivalently dpl) to ensure that the volume of the dipole representation of the particle is exactly correct [Fig. 2(b)], i.e. exactly corresponds to x. This is believed to increase the accuracy of the DDA, especially for small scatterers [10]. However, it introduces a minor inconvenience that the size of the computational grid is not exactly equal to the size of the particle. The volume correction can be turned off by command line option

```
-no_vol_cor
```

In this case `ADDA` tries to match the size of the particle using specified kD_x or calculating it from specified x (if shape permits). Moreover, `ADDA` then determines the "real" value of x numerically from the volume of the dipole representation. Its value is shown in `log` file (§C.4) and is further used in `ADDA`, e.g. for normalization of cross sections (§10.3), although it may slightly differ from specified or analytically derived x.

To read particle geometry from a file, specify the file name as an argument to the command line option

```
-shape read <filename>
```

This file specifies all the dipoles in the simulation grid that belongs to the particle (possibly several domains with different refractive indices). Supported formats include `ADDA` text formats and `DDSCAT` 6 and 7 shape formats (see §B.5 for details). Dimensions of the computational grid are then initialized automatically. Packages `misc/pip` and `misc/hyperfun` allow one to transform a variety of common 3D shape formats to one readable by `ADDA`. It is also possible to add support for new shape format directly into `ADDA`.[28]

Sometimes it is useful to describe particle geometry in a coarse way by larger dipoles (cubes), but then use smaller dipoles for the simulation itself.[29] `ADDA` enables this by the command line option

```
-jagged <arg>
```

that specifies a multiplier J. Large cubes ($J \times J \times J$ dipoles) are used [Fig. 2(c)] for construction of the dipole set. Cube centers are tested for belonging to a particle's domain. All grid

[28] See `http://code.google.com/p/a-dda/wiki/AddingShapeFileFormat` for detailed instructions.
[29] This option may be used e.g. to directly study the shape errors in DDA (i.e. caused by imperfect description of the particle shape) [21].

dimensions are multiplied by J. When particle geometry is read from file it is considered to be a configuration of big cubes, each of them is further subdivided into J^3 dipoles.

ADDA includes a granule generator, which can randomly fill any specified domain with granules of a predefined size. It is described in details in §6.5.

The last parameter to completely specify a scatterer is its refractive index. Refractive indices are given on the command line

```
-m {<m1Re> <m1Im> […]|<m1xxRe> <m1xxIm> <m1yyRe> <m1yyIm>
<m1zzRe> <m1zzIm> […]}
```

Each pair of arguments specifies the real and imaginary part[30] of the refractive index of the corresponding domain (first pair corresponds to domain number 1, etc.). Command line option

```
-anisotr
```

can be used to specify that a refractive index is anisotropic. In that case three refractive indices correspond to one domain. They are the diagonal elements of the refractive index tensor in the particle reference frame (§6.1). Currently ADDA supports only diagonal refractive index tensors; moreover, the refractive index must change discretely. Anisotropy can not be used with either CLDR polarizability (§9.1) or SO formulations (§9.1, §9.2, §9.3), since they are derived assuming isotropic refractive index, and can not be easily generalized. Use of anisotropic refractive index cancels the rotation symmetry if its x and y-components differ. Limited testing of this option was performed for Rayleigh anisotropic spheres.

The maximum number of different refractive indices (particle domains) is defined at compilation time by the parameter MAX_NMAT in the file const.h. By default it is set to 15. The number of the domain in the geometry file (§B.5) exactly corresponds to the number of the refractive index. Numbering of domains for the predefined shapes is described in §6.4. If no refractive index is specified, it is set to 1.5, but this default option works only for single-domain isotropic scatterers. Currently ADDA produces an error if any of the given refractive index equals to 1. It is planned to improve this behavior to accept such refractive index and automatically make corresponding domain void. This can be used, for instance, to generate spherical shell shape using standard option `–shape coated`. For now, one may set refractive index to the value very close to 1 for this purpose, e.g. equal to 1.00001.

ADDA saves the constructed dipole set to a file if the command line option

```
-save_geom [<filename>]
```

is specified, where <filename> is an optional argument. If it is not specified, ADDA names the output file <type>.<ext>. <type> is shape name – a first argument to the -shape command line option, see above and §6.4, possibly with addition of _gran (§6.5). <ext> is determined by the format (geom for ADDA and dat for DDSCAT formats), which itself is determined by the command line option

```
-sg_format {text|text_ext|ddscat6|ddscat7}
```

First two are ADDA default formats for single- and multi-domain particles respectively. DDSCAT 6 and 7 format, which differ by a single line, are descrbied in §B.5. text is automatically changed to text_ext for multi-domain particles. Output formats are compatible with the input ones (see §C.11 for details). The values of refractive indices are not saved (only domain numbers). This option can be combined with -prognosis, then no DDA simulation is performed but the geometry file is generated.

[30] ADDA uses exp(−iωt) convention for time dependence of harmonic electric field, therefore absorbing materials have *positive* imaginary part.

6.4 Predefined shapes

Predefined shapes are initialized by the command line option

```
-shape <type> [<args>]
```

where `<type>` is a name of the predefined shape. The size of the scatterer is determined by the size of the computational grid (D_x, §6.2); `<args>` specify different dimensionless aspect ratios or other proportions of the particle shape. However, some shapes define the absolute size themselves, which can be overridden (§6.2). In the following we describe the supported predefined shapes in alphabetic order:

- `axisymmetric` – axisymmetric homogeneous shape, defined by its contour in ρz-plane of the cylindrical coordinate system, which is read from file (format described in §B.6).
- `bicoated` – two identical concentric coated spheres with outer diameter d (first domain), inner diameter d_{in}, and center-to-center distance R_{cc} (along the z-axis). It describes both separate and sintered coated spheres. In the latter case sintering is considered symmetrically for cores and shells.
- `biellipsoid` – two general ellipsoids in default orientations with centers on the z-axis, touching each other. Their semi-axes are x_1, y_1, z_1 (lower one, first domain) and x_2, y_2, z_2 (upper one, second domain).
- `bisphere` – two identical spheres with outer diameter d and center-to-center distance R_{cc} (along the z-axis). It describes both separate and sintered spheres.
- `box` – a homogenous cube (if no arguments are given) or a rectangular parallelepiped with edge sizes x, y, z.
- `capsule` – cylinder with height (length) h and diameter d with half-spherical caps on both ends. Total height of the capsule is $h + d$.
- `chebyshev` – axisymmetric Chebyshev particle of amplitude ε ($|\varepsilon| \leq 1$) and order n (natural number). Its formula in spherical coordinates (r, θ) is
$$r = r_0\left[1 + \varepsilon\cos(n\theta)\right], \tag{9}$$
where r_0 is determined by the particle size.
- `coated` – sphere with a spherical inclusion; outer sphere has a diameter d (first domain). The included sphere has a diameter d_{in} (optional position of the center: x, y, z).
- `cylinder` – homogenous cylinder with height (length) h and diameter d.
- `egg` – axisymmetric biconcave homogenous particle, which surface is given as
$$r^2 + \nu rz - (1-\varepsilon)z^2 = a^2, \tag{10}$$
where r is the radius in spherical coordinates and a is scaling factor, which can be derived from egg diameter (maximum width perpendicular to the z- axis) or volume [36]. Center of the reference frame in Eq. (10) is shifted along the z-axis relative to the center of the computational grid to ensure that North and South Poles of the egg are symmetric over the latter center.
- `ellipsoid` – homogenous general ellipsoid with semi-axes x, y, z.
- `line` – line along the x-axis with the width of one dipole.
- `plate` – homogeneous plate (cylinder, which sides are rounded by adding half-torus) with height (thickness) h and full diameter d (i.e. the diameter of the constituent cylinder is $d - h$).
- `prism` – homogeneous right prism with height (length along the z-axis) h based on a regular polygon with n sides of size a. The polygon is oriented so that the positive x-axis is a middle perpendicular for one of its sides. D_x equals $2R_{\text{i}}$ and $R_{\text{c}} + R_{\text{i}}$ for even and odd n respectively. $R_{\text{c}} = a/[2\sin(\pi/n)]$ and $R_{\text{i}} = R_{\text{c}}\cos(\pi/n)$ are radii of circumscribed and inscribed circles respectively.

Table 1. Brief description of arguments, symmetries (§6.7), and availability of analytical value of f_{vol} for predefined shapes. Shapes and their arguments are described in the text. "±" means that it depends on the arguments.

<type>	<args>	dom.[a]	symY[b]	symR[c]	f_{vol}	size[d]
`axisymmetric`	filename	1	+	+	–	+
`bicoated`	R_{cc}/d, d_{in}/d	2	+	+	+	–
`biellipsoid`	y_1/x_1, z_1/x_1, x_2/x_1, y_2/x_2, z_2/x_2	2	+	±	+	–
`bisphere`	R_{cc}/d	1	+	+	+	–
`box`	[y/x, z/x]	1	+	±	+	–
`capsule`	h/d	1	+	+	+	–
`chebyshev`	ε, n	1	+	+	+	–
`coated`	d_{in}/d, [x/d, y/d, z/d]	2	±	±	+	–
`cylinder`	h/d	1	+	+	+	–
`egg`	ε, v	1	+	+	+	–
`ellipsoid`	y/x, z/x	1	+	±	+	–
`line`	–	1	–	–	–	–
`plate`	h/d	1	+	+	+	–
`prism`	n, h/D_x	1	±	±	+	–
`rbc`	h/d, b/d, c/d	1	+	+	–	–
`sphere`	–	1	+	+	+	–
`spherebox`	d_{sph}/D_x	2	+	+	+	–

[a] number of domains.
[b] symmetry with respect to reflection over the xz-plane.
[c] symmetry with respect to rotation by 90° over the z-axis.
[d] whether a shape defines absolute size of the particle.

- `rbc` – red blood cell, an axisymmetric biconcave homogenous particle, which is characterized by diameter d, maximum and minimum width h, b, and diameter at the position of the maximum width c. Its surface is defined as
$$\rho^4 + 2S\rho^2 z^2 + z^4 + P\rho^2 + Qz^2 + R = 0\,, \qquad (11)$$
where ρ is the radius in cylindrical coordinates ($\rho^2 = x^2 + y^2$). P, Q, R, and S are determined from the values of the parameters given in the command line. This formula is based on [37], and is similar to the RBC shape used in [38].
- `sphere` – homogenous sphere (used by default).
- `spherebox` – sphere (diameter d_{sph}) in a cube (size D_x, first domain).

For all axisymmetric shapes, symmetry axis coincides with the z-axis. The order of domains is important to assign refractive indices specified in the command line (§6.3). Brief reference information is summarized in Table 1, while examples are provided in Fig. 3. For multi-domain shapes f_{vol} is based on the total volume of the particle. Adding a new shape is straightforward for anyone who is familiar with C programming language.[31]

6.5 Granule generator

Granule generator is enabled by the command line option

```
-granul <vol_frac> <diam> [<dom_number>]
```

which specifies that one particle domain should be randomly filled with spherical granules with specified diameter `<diam>` and volume fraction `<vol_frac>`. The domain number to fill is given by the last optional argument (default is the first domain). Total number of

[31] See http://code.google.com/p/a-dda/wiki/AddingShape for detailed instructions.

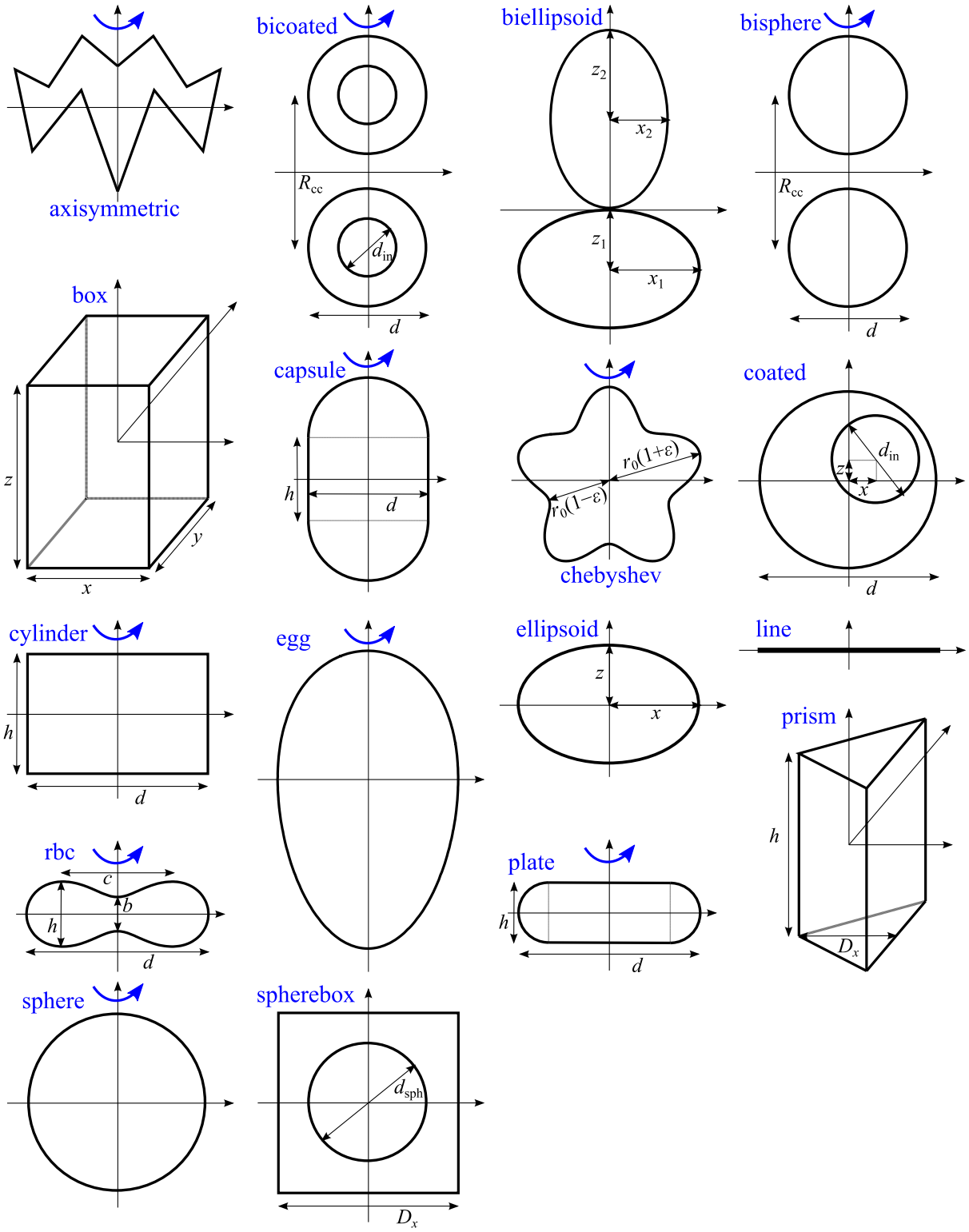

Fig. 3. Examples of predefined shapes. Most shapes are depicted by projections on the *xz*-plane. A blue arrow around the z-axis denotes axisymmetry.

domains is then increased by one; the last is assigned to the granules. Suffix "`_gran`" is added to the shape name and all particle symmetries (§6.7) are cancelled.

A simplest algorithm is used: to place randomly a sphere and see whether it fits in the given domain together with all previously placed granules. The only information that is used about some of the previously place granules is dipoles occupied by them, therefore intersection of two granules is checked through the dipoles, which is not exact, especially for

small granules. However it should not introduce errors larger than those caused by the discretization of granules. Moreover, it allows considering arbitrary complex domains, which is described only by a set of occupied dipoles. This algorithm is unsuitable for high volume fractions, it becomes very slow and for some volume fractions may fail at all (depending on the size of the granules critical volume fractions is 30–50%). Moreover, statistical properties of the obtained granules distribution may be not perfect; however, it seems good enough for most applications. To generate random numbers ADDA uses the Mersenne twister,[32] which combines high speed with good statistical properties [39]. The granule generator was used to simulate light scattering by granulated spheres by Yurkin *et al.* [40].

If volume correction (§6.3) is used, the diameter of the granules is slightly adjusted to give exact *a priori* volume fraction. *A posteriori* volume fraction is determined based on the total number of dipoles occupied by granules and is saved to `log` (§C.4). It is not recommended to use granule diameter smaller than the size of the dipole, since then the dipole grid can not adequately represent the granules, even statistically. ADDA will show a warning in that case; however, it will perform simulation for any granule size.

Currently the granule generator does not take into account `-jagged` option, which is planned to be fixed in the future.[33] For now one may save a geometry file for a particle model scaled to $J = 1$ and then load it using any desired J. The same trick can be used to fill different particle domains and/or using different sizes of granules. To do it the complete operation should be decomposed into elementary granule fills, which should be interweaved with saving and loading of geometry files.

Coordinates of the granules (its centers) can be saved to a file `granules` (§C.12) specifying command line option

```
-store_grans
```

This provides more accurate information about the granulated particle than its dipole representation, which can be used e.g. to refine discretization.

6.6 Partition over processors in parallel mode

To understand the parallel performance of ADDA it is important to realize how a scattering problem is distributed among different processors. Both the computational grid and the scatterer are partitioned in slices parallel to the *xy*-plane (in another words, partition is performed over the *z*-axis); each processor contains several consequtive slices. For the FFT-based task (§11.2) the whole grid is partitioned[34]. The partition over the *z*-axis is optimal for this task if n_z divides the number of processors (at least approximately).

The partition of the scatterer itself also benefits from the latter condition, however it is still not optimal for most of the geometries,[35] i.e. the number of non-void dipoles is different for different processors (Fig. 4). This partition is relevant for the computation of the scattered fields; hence its non-optimality should not be an issue in most cases. However, if large grid of scattering angles is used (§10.1, §10.3), the parallel performance of the ADDA may be relatively low (the total simulation time will be determined by the maximum number of real dipoles per processor).[36]

[32] http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
[33] http://code.google.com/p/a-dda/issues/detail?id=21
[34] More exactly, the grid is doubled in each dimension and then partitioned (see also §11.2).
[35] Exceptions are cubes and any other particles, for which area of any cross section perpendicular to the z-axis is constant.
[36] That is additionally to the communication overhead that always exists (§11.4).

The conclusion of this section is that careful choice of n_z and number of the processors (so that the former divides the latter) may significantly improve the parallel performance. ADDA will work fine with any input parameters, so this optimization is left to the user. Consider also some limitations imposed on the grid dimensions by the implemented FFT routines (§11.2).

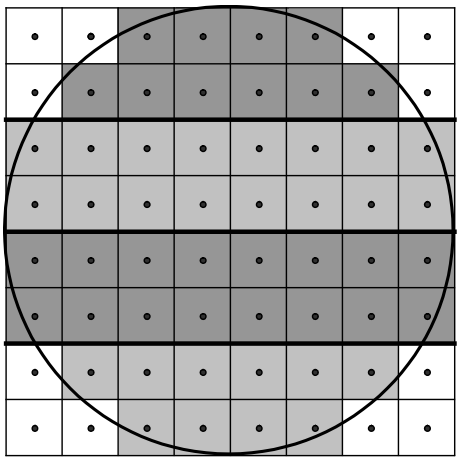

Fig. 4. Same as Fig. 2(a) but partitioned over 4 processors (shown in different shades of gray).

If the particle is prolate or oblate, parallel efficiency of ADDA depends on orientation of the former in the particle reference frame. First, it is recommended to set the *y*-axis along the smallest particle dimension. Second, positioning the longest scatterer dimension along the *x*-axis minimizes the memory requirements for a fixed (and relatively small) n_p [Eq. (4)], while position along the *z*-axis is optimal for the maximum n_p [Eq. (5)]. Unfortunately, currently ADDA can not rearrange the axes with respect to the particle,[37] so the only way to implement the above recommendations is to prepare input shape files (§6.3) accordingly. For predefined shapes (§6.4) one needs to export geometry to a shape file, rotate it in the shape file by a separate routine, and import it back into the ADDA.[38] Direction of propagation of the incident radiation should also be adjusted (§8.1), if axes of the particle reference frames are rearranged, to keep the scattering problem equivalent to the original one.

Finally, this section is not relevant for the sparse mode (§11.3), since then only the non-void dipoles are considered. Those dipoles are uniformly distributed among the processors irrespective of their position in the computational grid. In particular, there is no limitation of a slice to belong to a single processor.

6.7 Particle symmetries

Symmetries of a light-scattering problem are used in ADDA to reduce simulation time. All the symmetries are defined for the default incident beam (§8). If the particle is symmetric with respect to reflection over the *xz*-plane, only half of the scattering *yz*-plane is considered (scattering angle from 0° to 180°, §10.1). If the particle is symmetric with respect to rotation by 90° over the *z*-axis, the Mueller matrix in the *yz*-plane (§10.1) can be calculated from the calculation of the internal fields for just one incident polarization (*y*-polarization is used). The second polarization is then equivalent to the first one but with scattering in the *xz*-plane (in negative direction of *x*-axis). The symmetries are automatically determined for all the predefined shapes (§6.4). Some or all of them are automatically cancelled if not default beam type and/or direction (§8), anisotropic refractive index (§6.3), or granule generator (§6.5) are used.

Use of symmetry can be controlled by the command line option:

```
-sym {auto|no|enf}
```

First option corresponds to the default behavior described before, while `no` and `enf` specify to never use or enforce symmetry respectively. Use the latter with caution, as it may lead to erroneous results. It may be useful if the scattering problem is symmetric, but ADDA do not recognize it automatically, e.g. for particles that are read from file or when non-default incident beam is used, which does not spoil the symmetry of the problem (e.g. plane wave

[37] Rotation of the particle with respect to the *laboratory* reference frame (§6.1) does not help because it does not affect the particle in the *particle* reference frame.

[38] That is probably not worth the efforts for most of the problems.

propagating along the x-axis for a cubical scatterer). It is important to note that not the scatterer but its dipole representation should be symmetric;[39] otherwise the accuracy of the result will generally be slightly worse than that when symmetry is not enforced.

Particle symmetries can also be used to decrease the range of orientation/scattering angles for different averagings/integrations. However, it is user's responsibility to decide how a particular symmetry can be employed. This is described in the descriptions of corresponding input parameters files (§B.2, §B.3, §B.4).

7 Orientation of the Scatterer

7.1 Single orientation

Any particle orientation with respect to the laboratory reference frame can be specified by three Euler angles (α,β,γ). `ADDA` uses a notation based on [41], which is also called "zyz-notation" or "y-convention". In short, coordinate axes attached to the particle are first rotated by the angle α over the z-axis, then by the angle β over the current position of the y-axis (the line of nodes), and finally by the angle γ over the new position of the z-axis (Fig. 5). These angles are specified in degrees as three arguments to the command line option

```
-orient <alpha> <beta> <gamma>
```

`ADDA` simulates light scattering in the particle reference frame (§6.1), therefore rotation of the particle is equivalently represented as an inverse rotation of the incident wave propagation direction and polarization (§8.1), position of the beam center (if relevant, §8.2), and scattering plane (angles). The information about the orientation of a scatterer is saved to the `log` (§C.4).

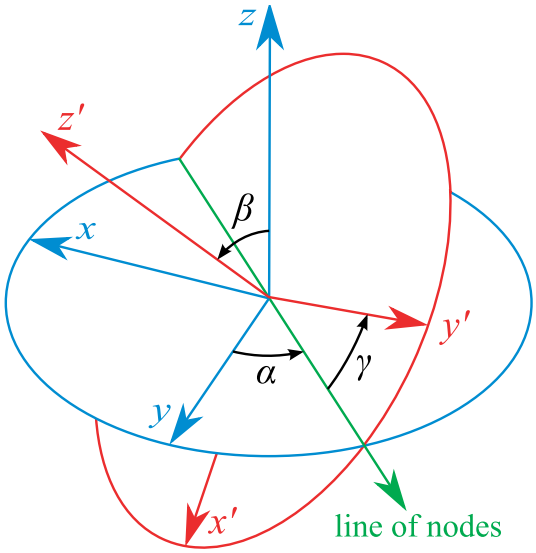

Fig. 5. Transformation of the laboratory reference system xyz into the particle reference frame $x'y'z'$ through consecutive rotation by angles α, β, and γ. Adapted from [1].

7.2 Orientation averaging

Orientation averaging is performed over three Euler angles (α,β,γ). Rotating over α is equivalent to rotating the scattering plane without changing the orientation of the scatterer relative to the incident radiation. Therefore, averaging over this orientation angle is done with a single computation of internal fields; additional computational time for each scattering plane is comparably small. Averaging over other two Euler angles is done by independent DDA simulations (defining the orientation of the scatterer as described in §7.1). The averaging itself is performed using the Romberg integration (§11.6), parameters of the averaging are stored by default in file `avg_params.dat` (§B.2). Orientation averaging is enabled by the command line option

```
-orient avg [<filename>]
```

where `<filename>` is an optional argument that specifies a different file with parameters of the averaging. Integration points for β are spaced uniformly in values of cosβ. Currently only

[39] For example, a sphere is symmetric for any incident direction, but the corresponding dipole set (Fig. 2) is only symmetric for incidence along a coordinate axis.

the Mueller matrix in one scattering plane (§10.1), C_{ext}, and C_{abs} (§10.3) are calculated when doing orientation averaging.

It also can not be used in combination with saving incident beam (§8), internal fields or dipole polarizations (§10.5), or radiation forces (§7.2), nor with calculating scattering for a grid of angles (§10.1).

8 Incident Beam

This section describes how to specify the incident electric field. This field, calculated for each dipole, can be saved to file `IncBeam` (§C.9). To enable this functionality specify command line option

```
-store_beam
```

8.1 Propagation direction

The direction of propagation of the incident radiation is specified by the command line option

```
-prop <x> <y> <z>
```

where arguments are *x*, *y*, and *z* components of the propagation vector. Normalization (to the unity vector) is performed automatically by `ADDA`. By default vector $\mathbf{e}_z = (0,0,1)$ is used. Two incident polarizations are used by default: along the *x*- and *y*-axes. Those are perpendicular ($\perp$) and parallel ($\parallel$) polarizations [42] respectively with respect to the default scattering plane (*yz*). These polarizations are transformed simultaneously with the propagation vector – all three are rotated by two spherical angles (θ,φ) so that $\mathbf{e}_z$ is transformed into the specified propagation vector. Afterwards, the scattering angles are specified with respect to the incident wave reference frame (§6.1) based on the *new* propagation vector (*z*) and two *new* incident polarizations (*x*,*y*).[40]

The option `-prop` is cumulative with rotation of the particle (§7.1) because the latter is equivalent to the inverse rotation of incident wave and scattering angles. If after all transformations the propagation vector is not equal to the default (0,0,1), all the symmetries of the scatterer are cancelled (§6.7).

8.2 Beam type

Additionally to the default planve wave `ADDA` supports several types of finite size incident beams, specified by the command line option

```
-beam <type> [<width> <x> <y> <z>]
```

where `<type>` is one of the `plane`, `lminus`, `davis3`, or `barton5`. All predefined beam types except the default plane wave are approximate descriptions of a Gaussian beam. Four arguments specified in the command line specify width (w_0) and *x*, *y*, *z* coordinates of the center of the beam respectively (all in μm). The coordinates are specified in the laboratory reference plane (§6.1). `lminus` is the simplest approximation [43], `davis3` [44] and `barton5` [45] are correct up to the third and fifth order of the beam confinement factor ($s = 1/kw_0$) respectively. The latter is recommended for all calculations; others are left mainly for comparison purposes. However, for tightly focused beams even `barton5` may be not accurate enough.

Total power of the beam for `barton5` is [45]

$$P = \frac{\pi w_0^2 I_0}{2}\left(1 + s^2 + 1.5s^4\right), \tag{12}$$

[40] For example, the default scattering plane (§10.1), *yz*-plane, will be the one based on the new propagation vector and new incident polarization, which corresponds to the *y*-polarization for the default incidence.

where I_0 is time-averaged irradiance in the beam focal point. For lower-order approximations (`lminus` and `davis3`) corresponding small terms in the parentheses in Eq. (12) should be omitted. For all beam types `ADDA` assumes unity amplitude of the electric field in the focal point of the beam (in Gaussian-CGS system of units), which implies $I_0 = 1/(8\pi)$. Some of the particle symmetries (§6.7) may be cancelled according to the coordinates of the beam center.

Adding of a new beam is straightforward for anyone who is familiar with C programming language.[41] Moreover, arbitrary beam can be read from file, using

```
-beam read <filenameY> [<filenameX>]
```

Normally two files are required for *y*- and *x*-polarizations respectively (§8.1), but a single filename is sufficient if only *y*-polarization is used (e.g. due to symmetry). Note, that "`-beam read`" breaks the symmetry by itself, since the input beam is not guaranteed to satisfy any symmetry. Therefore, to simulate only *y*-polarization "`-sym enf`" should be explicitly specified (§6.7). Incident field should be specified in a *particle* reference frame, see §B.7 for file format.

9 DDA Formulation

Since its introduction by Purcell and Pennypacker [3] the DDA has been constantly developed; therefore a number of different DDA formulations exist [2]. Here we only provide a short summary, focusing on those that are implemented in `ADDA`. All formulations are equivalent to the solution of the linear system to determine unknown dipole polarizations $\mathbf{P}_i$

$$\overline{\boldsymbol{\alpha}}_i^{-1}\mathbf{P}_i - \sum_{j\neq i}\overline{\mathbf{G}}_{ij}\mathbf{P}_j = \mathbf{E}_i^{\mathrm{inc}}, \tag{13}$$

where $\mathbf{E}_i^{\mathrm{inc}}$ is the incident electric field, $\overline{\boldsymbol{\alpha}}_i$ is the dipole polarizability (self-term), $\overline{\mathbf{G}}_{ij}$ is the interaction term, and indices *i* and *j* enumerate the dipoles. For a plane wave incidence

$$\mathbf{E}^{\mathrm{inc}}(\mathbf{r}) = \mathbf{e}^0 \exp(\mathrm{i}\mathbf{k}\cdot\mathbf{r}), \tag{14}$$

where $\mathbf{k} = k\mathbf{a}$, $\mathbf{a}$ is the incident direction, and $|\mathbf{e}^0| = 1$. Other incident beams are discussed in §8.2. The (total) internal electric field $\mathbf{E}_i$ is the one present in a homogenous particle modeled by an array of dipoles, also know as macroscopic field [46]. It should be distinguished from the exciting electric field $\mathbf{E}_i^{\mathrm{exc}}$ that is a sum of $\mathbf{E}_i^{\mathrm{inc}}$ and the field due to all other dipoles, but excluding the field of the dipole *i* itself. Both total and exciting electric field can be determined once the polarizations are known:

$$\mathbf{P}_i = \overline{\boldsymbol{\alpha}}_i\mathbf{E}_i^{\mathrm{exc}} = V_{\mathrm{d}}\chi_i\mathbf{E}_i, \tag{15}$$

where $V_{\mathrm{d}} = d^3$ is the volume of a dipole and $\chi_i = (\varepsilon_i - 1)/4\pi$ is the susceptibility of the medium at the location of the dipole **(**ε_i – relative permittivity**)**. In the following we will also refer to **E** as internal fields (those inside the particle) in contrast to near- and far-fields, which are calculated from **E** or **P** together with other scattering quantities. Below we discuss different formulations for the polarizability prescription (§9.1), interaction term (§9.2) and formulae to calculate scattering quantities (§9.3). Additionally to the published ones, `ADDA` contains options to use new theoretical improvements under development. We do not discuss them here, since they are still in the early research phase. However, you may try them at your own risk.

9.1 Polarizability prescription

A number of expressions for the polarizability are known [2]. `ADDA` implements the following: the Clausius–Mossotti (CM), the radiative reaction correction (RR, [47]), formulation by Lakhtakia (LAK, [48]), digitized Green's function (DGF, [48]), approximate

[41] See `http://code.google.com/p/a-dda/wiki/AddingBeam` for detailed instructions.

integration of Green's tensor over the dipole ($\mathrm{IGT_{SO}}$, [49]), the lattice dispersion relation (LDR, [11]), corrected LDR (CLDR, [50]), and the Filtered Coupled Dipoles (FCD); and the second order (SO) polarizability prescription (under development). The CM polarizability is the basic one, given by

$$\overline{\boldsymbol{\alpha}}_i^{\mathrm{CM}} = \overline{\mathbf{I}} V_{\mathrm{d}} \frac{3}{4\pi} \frac{\varepsilon_i - 1}{\varepsilon_i + 2}, \tag{16}$$

where $\overline{\mathbf{I}}$ is the identity tensor. Other polarizability formulations can be expressed through it, using the correction term $\overline{\mathbf{M}}$ associated with finite size of the dipoles [49]:

$$\overline{\boldsymbol{\alpha}}_i = \overline{\boldsymbol{\alpha}}_i^{\mathrm{CM}} \left(\overline{\mathbf{I}} - \overline{\mathbf{M}}_i \overline{\boldsymbol{\alpha}}_i^{\mathrm{CM}} / V_{\mathrm{d}} \right)^{-1},$$

in particular, $\overline{\mathbf{M}}^{\mathrm{CM}} = 0$. RR is a third-order (in kd) correction to the CM [47]:

$$\overline{\mathbf{M}}^{\mathrm{RR}} = (2/3)\mathrm{i}(kd)^3 \overline{\mathbf{I}}. \tag{17}$$

Following formulations add second-order corrections. LAK is based on replacing the cubical dipole by an equi-volume sphere with radius $a_{\mathrm{d}} = d(3/4\pi)^{1/3}$ and integrating Green's tensor over it [51]:

$$\overline{\mathbf{M}}^{\mathrm{LAK}} = (8\pi/3)\overline{\mathbf{I}}\left[(1 - \mathrm{i}ka_{\mathrm{d}})\exp(\mathrm{i}ka_{\mathrm{d}}) - 1\right]. \tag{18}$$

DGF is an approximation of LAK up to the third order of kd [48]:

$$\overline{\mathbf{M}}^{\mathrm{DGF}} = \overline{\mathbf{I}} \left(b_1^{\mathrm{DGF}} (kd)^2 + (2/3)\mathrm{i}(kd)^3 \right), \; b_1^{\mathrm{DGF}} \approx 1.611992, \tag{19}$$

while $\mathrm{IGT_{SO}}$ is the same-order approximation applied to the original cubical dipole [49]

$$\overline{\mathbf{M}}^{\mathrm{IGT_{SO}}} = \overline{\mathbf{I}} \left(b_1^{\mathrm{IGT}} (kd)^2 + (2/3)\mathrm{i}(kd)^3 \right), \; b_1^{\mathrm{IGT}} \approx 1.586718. \tag{20}$$

LDR is based on consideration of infinite grid of point dipoles [11]:

$$\overline{\mathbf{M}}^{\mathrm{LDR}} = \overline{\mathbf{I}}\left[\left(b_1^{\mathrm{LDR}} + b_2^{\mathrm{LDR}} m^2 + b_3^{\mathrm{LDR}} m^2 S \right)(kd)^2 + (2/3)\mathrm{i}(kd)^3 \right], \tag{21}$$

$$b_1^{\mathrm{LDR}} \approx 1.8915316, \; b_2^{\mathrm{LDR}} \approx -0.1648469, \; b_3^{\mathrm{LDR}} \approx 1.7700004, \; S = \sum\nolimits_{\mu} (a_\mu e_\mu^0)^2, \tag{22}$$

where μ denotes vector components. The LDR prescription can be averaged over all possible incident polarizations [11], resulting in

$$S = \left(1 - \sum\nolimits_{\mu} a_\mu^4\right) \Big/ 2. \tag{23}$$

Corrected LDR is independent on the incident polarization but leads to the diagonal polarizability tensor instead of scalar [50]

$$\overline{\mathbf{M}}_{\mu\nu}^{\mathrm{CLDR}} = \delta_{\mu\nu}\left[\left(b_1^{\mathrm{LDR}} + b_2^{\mathrm{LDR}} m^2 + b_3^{\mathrm{LDR}} m^2 a_\mu^2 \right)(kd)^2 + (2/3)\mathrm{i}(kd)^3 \right], \tag{24}$$

where $\delta_{\mu\nu}$ is the Kronecker symbol. The FCD polarizability is obtained from the value of filtered Green's tensor [Eq. (28)] for zero argument, leading to [23]

$$\overline{\mathbf{M}}^{\mathrm{FCD}} = V_{\mathrm{d}} \lim_{R \to 0} \overline{\mathbf{G}}^{\mathrm{F}}(\mathbf{R}) = \frac{4}{3}(kd)^2 + \frac{2}{3}\left(\mathrm{i} + \frac{1}{\pi} \ln \frac{\pi - kd}{\pi + kd} \right)(kd)^3. \tag{25}$$

Naturally, Eq. (25) is applicable only when $kd < \pi$, i.e. dpl > 2.

All polarizability formulations, except CLDR and SO, can be used together with anisotropic electric permittivity, given by a *diagonal* tensor $\overline{\boldsymbol{\varepsilon}}$. Polarizability is then also a diagonal tensor, calculated by the same formulae [Eqs. (16)–(25)] but separately for each component:

$$\alpha_{\mu\nu} = \delta_{\mu\nu} \alpha(\varepsilon_{\mu\mu}). \tag{26}$$

The choice of the polarizability prescription is performed by command line option

```
-pol <type> [<arg>]
```

where `<type>` is one of the `cldr`, `cm`, `dgf`, `fcd`, `igt_so`, `lar`, `ldr`, `rrc`, `so`. `<arg>` is optional flag that can be only `avgpol` and only for LDR specifying that the LDR polarizability should be averaged over incident polarizations. Default is LDR without averaging. It is important to note that this is not the best option for all cases. Our experience

shows that LDR may perform particularly badly (as compared to CM or RR) for very large refractive indices, and FCD (together with its interaction term, §9.2) becomes the best option [23].

Finally, other DDA improvements are known, which modify the polarizability (or permittivity) of the dipoles near the boundary. These improvements include weighted discretization [52] and spectral filtering of the permittivity that was proposed in combination with the FCD [19]. These ideas have not yet been implemented in ADDA.[42]

9.2 Interaction term

A few formulations for the interaction term are known [2]. Currently, ADDA can use the simplest one (interaction of point dipoles), the FCD (in other words, filtered Green's tensor [19]), quasistatic version of the FCD, the Integrated Green's Tensor (IGT, [14]), approximation of IGT ($\mathrm{IGT_{SO}}$), and the second order (SO) formulation (the latter is under development). The interaction of point dipoles is described by the Green's tensor:

$$\overline{\mathbf{G}}_{ij} = \overline{\mathbf{G}}(\mathbf{r}_i, \mathbf{r}_j) = \frac{\exp(\mathrm{i}kR)}{R}\left[k^2\left(\bar{\mathbf{I}} - \frac{\hat{R}\hat{R}}{R^2}\right) - \frac{1-\mathrm{i}kR}{R^2}\left(\bar{\mathbf{I}} - 3\frac{\hat{R}\hat{R}}{R^2}\right)\right], \tag{27}$$

where $\mathbf{r}_i$ is the radius-vector of the dipole center, $\mathbf{R} = \mathbf{r}_j - \mathbf{r}_i$, $R = |\mathbf{R}|$, and $\hat{R}\hat{R}$ is a tensor defined as $\hat{R}\hat{R}_{\mu\nu} = R_\mu R_\nu$. The filtered Green's tensor is defined [19] as

$$\overline{\mathbf{G}}_{ij}^{\mathrm{FCD}} = \bar{\mathbf{I}}\left(k^2 g_{\mathrm{F}}(R) + \frac{g_{\mathrm{F}}'(R)}{R} + \frac{4\pi}{3}h_r(R)\right) + \frac{\hat{R}\hat{R}}{R^2}\left(g_{\mathrm{F}}''(R) - \frac{g_{\mathrm{F}}'(R)}{R}\right), \tag{28}$$

where h_r is filter impulse response:

$$h_r(R) = \frac{\sin(k_{\mathrm{F}}R) - k_{\mathrm{F}}R\cos(k_{\mathrm{F}}R)}{2\pi^2 R^3}, \tag{29}$$

$k_{\mathrm{F}} = \pi/d$ – the wavenumber corresponding to the grid, and g_{F} is the filtered scalar Green's function:

$$\begin{aligned} g_{\mathrm{F}}(R) = \frac{1}{\pi R}\{&\sin(kR)\left[\pi\mathrm{i} + \mathrm{Ci}\left((k_{\mathrm{F}}-k)R\right) - \mathrm{Ci}\left((k_{\mathrm{F}}+k)R\right)\right] \\ &+ \cos(kR)\left[\mathrm{Si}\left((k_{\mathrm{F}}+k)R\right) + \mathrm{Si}\left((k_{\mathrm{F}}-k)R\right)\right]\}. \end{aligned} \tag{30}$$

To apply this formulation k_{F} must be larger than k, i.e. dpl > 2. Quasistatic FCD is obtained in the limit $kR \to 0$, which leads to a simpler expression [53]:

$$\overline{\mathbf{G}}_{ij}^{\mathrm{FCD,st}} = -\frac{2}{3\pi R^3}\left(\bar{\mathbf{I}} - 3\frac{\hat{R}\hat{R}}{R^2}\right)\left[3\,\mathrm{Si}(k_{\mathrm{F}}R) + k_{\mathrm{F}}R\cos(k_{\mathrm{F}}R) - 4\sin(k_{\mathrm{F}}R)\right]. \tag{31}$$

Although it is just a special case of full FCD, it is implemented in ADDA as a separate option for testing and research purposes. Since FCD was originally designed for high refractive indices, we recommend using it especially in this regime. However, it also woks fine for moderate refractive index, generally not worse than the standard approach of point dipoles [23].

The IGT directly accounts for the finiteness of the cubical dipole, by integrating over its volume V_j [14]:

$$\overline{\mathbf{G}}_{ij}^{\mathrm{IGT}} = \frac{1}{V_{\mathrm{d}}}\int_{V_j} d^3r' \overline{\mathbf{G}}(\mathbf{r}_i, \mathbf{r}'). \tag{32}$$

Implementation of the IGT in ADDA is based on the Fortran code kindly provided by IGT's original authors [14]. $\mathrm{IGT_{SO}}$ is based on approximate evaluation of Eq. (32), up to second order of kd, using tabulated integrals. Thus it is almost as fast as Eq. (27) and almost as

[42] See development branch http://code.google.com/p/a-dda/source/browse/branches/wd/ .

accurate as IGT. Both IGT and $\mathrm{IGT_{SO}}$ are known to perform very good for small scatterers with large and almost real refractive indices.

The choice of the interaction term is performed by the command line option

`-int <type> [<arg1> [<arg2>]]`

where `<type>` is one of the `fcd`, `fcd_st`, `igt`, `igt_so`, `poi`, `so`. Two optional arguments are relevant only for `igt`. `<arg1>` is the maximum distance (in dipole sizes), for which integration is performed. For larger distances simpler Eq. (27) is used. Using value of this parameter from 1 to 3 is recommended – then extra computational time for the IGT is rather small, while most of the accuracy gain is achieved (compared to the default IGT without distance limit). `<arg2>` specifies the relative error of the integration (minus its decimal logarithm). By default the same value as argument of `-eps` command line option (§11.1) is used. For $\mathrm{IGT_{SO}}$ and SO formulation tables of precalculated integrals are used, they are automatically read from files in `tables/` (§D.1).

The default formulation for interaction term is that of point dipoles (`poi`); however, it is expected to be inferior to `fcd` or `igt` in many cases. However, the latter have been studied in much less details. So we recommend testing different formulations for a particular class of scattering problems of interest.

9.3 How to calculate scattering quantities

The simplest way to calculate scattering quantities is to consider a set of point dipoles with known polarizations, as summarized by Draine [47]. The scattering amplitude **F** for any scattering direction **n** is given as

$$\mathbf{F}(\mathbf{n}) = -\mathrm{i}k^3(\bar{\mathbf{I}} - \hat{n}\hat{n})\sum_i \mathbf{P}_i \exp(-\mathrm{i}k\mathbf{r}_i\cdot\mathbf{n})\,. \tag{33}$$

The amplitude and Mueller scattering matrices for direction **n** are determined from **F**(**n**) calculated for two incident polarizations [42]. Scattering cross section C_{sca} and asymmetry vector **g** are determined by integration of **F**(**n**) over the whole solid angle:

$$C_{\mathrm{sca}} = \frac{1}{k^2}\oint \mathrm{d}\Omega \left|\mathbf{F}(\mathbf{n})\right|^2, \tag{34}$$

$$\mathbf{g} = \frac{1}{k^2 C_{\mathrm{sca}}}\oint \mathrm{d}\Omega\, \mathbf{n}\left|\mathbf{F}(\mathbf{n})\right|^2. \tag{35}$$

Extinction and absorption cross section (C_{ext} and C_{abs}) are determined directly from $\mathbf{P}_i$:

$$C_{\mathrm{ext}} = 4\pi k\sum_i \mathrm{Im}\left(\mathbf{P}_i\cdot\mathbf{E}_i^{\mathrm{inc}*}\right), \tag{36}$$

$$C_{\mathrm{abs}} = 4\pi k\sum_i \left[\mathrm{Im}\left(\mathbf{P}_i\cdot\mathbf{E}_i^{\mathrm{exc}*}\right) - (2/3)k^3\left|\mathbf{P}_i\right|^2\right], \tag{37}$$

Variations of Eq. (37) (which is used by default) are possible, for instance [14]:

$$C_{\mathrm{abs}} = 4\pi k\sum_i \mathrm{Im}(\mathbf{P}_i\cdot\mathbf{E}_i^*)\,, \tag{38}$$

which is based on considering radiation correction of a *finite* dipole instead of a *point* dipole used in Eq. (37). The difference between these two expressions is subtle and depends on polarizability formulation (§9.1). In particular, it is zero if and only if $\bar{\mathbf{M}} - (2/3)\mathrm{i}(kd)^3$ is a Hermitian tensor. For LDR, CLDR, and SO the difference is $\mathrm{O}((kd)^2)$, but only when $\mathrm{Im}(m) \neq 0$. For CM and LAK the difference is $\mathrm{O}((kd)^3)$ and $\mathrm{O}((kd)^5)$ respectively for any m. For all other formulations the difference is always zero.

Corrections to the above expressions are possible, considering the far-field limit of IGT interaction term (§9.2), in particular [49]

$$\bar{\mathbf{G}}_i^{\mathrm{IGT}}(\mathbf{r}) \underset{r\to\infty}{\to} \eta(kd, \mathbf{r}/r)\bar{\mathbf{G}}(\mathbf{r}, \mathbf{r}_i)\,, \tag{39}$$

$$\eta(x,\mathbf{n}) = \prod_{\mu} \frac{\sin(n_\mu x/2)}{n_\mu x/2} = 1 - (1/24)\, x^2 + \mathrm{O}(x^4)\,, \tag{40}$$

$\mathrm{IGT_{SO}}$ is based on truncating Taylor expansion in Eq. (40) and applying the result $[1 - (kd)^2/24]$ as a multiplicative factor in Eqs.(33) and (36). Other expressions are unchanged, although C_{sca} is effectively multiplied by the square of this factor.

The choice between different ways to calculate scattering quantities is performed by command line option

```
-scat <type>
```

where `<type>` is `dr`, `fin`, `igt_so`, or `so`. Draine's classical formulation (`dr`) corresponds to Eqs. (33)–(37). Finite dipole correction (`fin`) uses Eq. (38) to calculate C_{abs}, and C_{ext} is obtained as Eq. (36) plus Eq. (38) minus Eq. (37). In other words, C_{ext} is corrected by the same amount as C_{abs} to ensure exact compliance with the optical theorem, which is discussed below. A few quick tests showed that finite dipole correction do *not* improve the accuracy, so using the default `dr` is recommended in most cases. However, `igt_so` is recommended, when the same formulation (or IGT) is used for interaction term (§9.2).

C_{sca} can be determined as $C_{\mathrm{ext}} - C_{\mathrm{abs}}$, which is faster than Eq. (34). However, this issue needs further clarifying. Draine noted [47] that C_{sca} calculated by integration over the solid angle can be more accurate than $C_{\mathrm{ext}} - C_{\mathrm{abs}}$, due to loss of significant digits when the latter two cross sections have close to equal values. Moreover, it has been suggested [54] that the difference between C_{sca} calculated by these two methods can be used as an internal measure of DDA accuracy. However, we stress that this difference is a measure of convergence of the iterative solver and accuracy of the integration over the solid angle, but not of the physical approximation itself. In other words, the difference may be very small, while the values themselves are very inaccurate (compared to the exact solution). To prove this, one may consider a supplementary disconnected particle consisting of a set of point dipoles with the same positions and polarizabilities as dipoles representing the original (connected) particle. For this supplementary particle DDA equations (13) involve no approximations, since they directly follow from the constitutive equations of a point dipole.[43] Therefore, their exact solution will satisfy the optical theorem, which can be formulated as $C_{\mathrm{sca}} = C_{\mathrm{ext}} - C_{\mathrm{abs}}$. Hence, the only possible reasons for the latter to be violated are inaccurate solution of Eq. (13) or inaccurate calculation of Eq. (34). And the difference between the original and supplementary particles, which is the error of the DDA itself, is not relevant. The simulations comply with this conclusion (data not shown, but see parts 12 and 13 of the tutorial[17]).

Similar conclusions are valid for other symmetry-based tests. For instance, recently a reciprocity condition, i.e. the invariancy of the simulation results when incident and scattered directions are exchanged, was studied in the framework of the DDA [55]. It was shown that reciprocity condition is automatically satisfied for the DDA results, whenever polarizability formulation (§9.1) is independent of the incident direction (or at least the same for two reciprocal configurations). In other words, the difference of results between two reciprocal configurations can be very small (on the order of the threshold of the iterative solver, §11.1), while the results themselves are completely wrong. And even for the LDR or CLDR polarizability the level of fulfillment of the reciprocity condition (while not trivial) is still not an adequate measure of the overall DDA accuracy.

10 What Scattering Quantities Are Calculated

All the scattering angles (polar θ and azimuthal φ) are specified with respect to the incident wave (see §7.1 and §8.1 for details). Angle φ is counted from the x-axis.

[43] Except for IGT interaction term (§9.2), then the optical theorem is expected to be correct only within $\mathrm{O}((kd)^2)$.

10.1 Mueller matrix and its derivatives

ADDA calculates the complete Mueller scattering matrix [42] for a set of scattering angles. The Mueller matrix is 4×4 real matrix, which is dimensionless and not normalized.[44] Moreover, it is defined with respect to the scattering reference frame opposed to the laboratory reference frame used in definition of the Stokes phase matrix [56].

By default scattering in the *yz*-plane ($\varphi = 90°$) is calculated and the result is saved in file `mueller` (§C.5). The range of [0°,180°] is equally divided into N_θ intervals, the latter is defined through the command line option

```
-ntheta <arg>
```

By default N_θ is from 90 to 720 depending on the size of the computational grid (§6.2). If the particle is not symmetric (§6.7) and orientation averaging (§7.2) is not used, the range is extended to 360°. Totally $N_\theta + 1$ or $2N_\theta + 1$ points are calculated. To calculate the Mueller matrix in one scattering plane ADDA simulates two incident polarizations, except when the particle is symmetric with respect to the rotation by 90° over the propagation vector of incident radiation (§6.7). In the latter case one incident polarization and two scattering planes are used instead.

More advanced options are available to calculate scattering at any set of angles. If any of the two command line options

```
-store_scat_grid
-phi_integr <arg>
```

is specified, the Mueller matrix is calculated for a set of angles, specified, by default, in file `scat_params.dat` (§B.4). The first flag indicates that values of the Mueller matrix for all calculated angles should be saved to file `mueller_scatgrid` (§C.5), while the second flag turns on the integration of Mueller matrix over φ with different multipliers. The latter is defined by `<arg>`, an integer from 1 to 31, each bit of which, from lowest to highest, corresponds to multipliers 1, $\cos(2\varphi)$, $\sin(2\varphi)$, $\cos(4\varphi)$, and $\sin(4\varphi)$ respectively.[45] Integration results are saved to files `mueller_integr`, `mueller_integr_c2`, `mueller_integr_s2`, `mueller_integr_c4`, and `mueller_integr_s4` respectively (§C.5). It is important to note that the results of the integration are divided by the width of the φ interval (2π by default), i.e. actually averaging over φ takes place. If both above-mentioned command line options are specified, both complete-grid and integrated results are saved to hard disk.

The format of the input file is very flexible (see §B.4 for details) allowing one to use either values uniformly spaced in some interval or any set of values, which is explicitly specified, for θ and φ independently. Even an arbitrary set of (θ,φ) pairs can be used. However, if integration over φ is used, a set of φ values must comply with the Romberg integration (§11.6). A different file describing a set of angles can be used if specified as an argument to the command line option

```
-scat_grid_inp <filename>
```

When a grid of scattering angles is calculated (either for saving or integrating over φ) the scattering in the *yz*-plane is by default *not* calculated. However, ADDA may be forced to calculate it by specifying command line option

```
-yz
```

---

[44] For instance, multiplication by k^{-2} transforms it into the Stokes phase matrix (**Z**, [56]), and further multiplication by $4\pi/C_{\text{sca}}$ (total multiplier $\lambda^2/\pi C_{\text{sca}}$) – into (normalized) Stokes scattering matrix (**F**, [56]), which is often used for particles in random orientation. The 1,1-element of the latter gives unity, when integrated over the whole solid angle.

[45] For example 1 corresponds to one integration with no multipliers, 6 – to two integration with $\cos(2\varphi)$ and $\sin(2\varphi)$ multipliers. The rationale behind these multipliers is their appearance in formulae for the light-scattering patterns measured by the scanning flow cytometer [38,57], however they hopefully may also be useful in other applications.

The Mueller matrix contains the complete intensity-based information about angle-resolved scattering for any incident polarization, but other quantities are also commonly used, e.g. differential scattering cross section. Depending on the incident polarization it can be calculated as:

$$\frac{\mathrm{d}C_{\mathrm{sca}}^{\mathrm{unpol}}}{\mathrm{d}\Omega}=\frac{S_{11}}{k^2},\ \frac{\mathrm{d}C_{\mathrm{sca}}^{x,y}}{\mathrm{d}\Omega}=\frac{1}{k^2}\left[S_{11}\pm S_{12}\cos(2\varphi)\pm S_{13}\sin(2\varphi)\right], \tag{41}$$

where plus and minus correspond to incident polarization along the *x*- and *y*-axis respectively, and φ is the azimuthal angle of the scattering direction (relative to the *x*-axis). "unpol" denotes unpolarized incident light. Eq. (41) is derived from formulae given in [42].

In other applications, the following scattering intensities are of interest:

$$\begin{aligned}
&I_{\perp\perp}=\left|S_1\right|^2=0.5\left(S_{11}-S_{12}-S_{21}+S_{22}\right), \quad I_{\parallel\parallel}=\left|S_2\right|^2=0.5\left(S_{11}+S_{12}+S_{21}+S_{22}\right),\\
&I_{\perp\parallel}=\left|S_3\right|^2=0.5\left(S_{11}-S_{12}+S_{21}-S_{22}\right), \quad I_{\parallel\perp}=\left|S_4\right|^2=0.5\left(S_{11}+S_{12}-S_{21}-S_{22}\right),\\
&I_{\perp}=I_{\perp\perp}+I_{\perp\parallel}=S_{11}-S_{12}, \qquad\qquad I_{\parallel}=I_{\parallel\perp}+I_{\parallel\parallel}=S_{11}+S_{12},
\end{aligned} \tag{42}$$

where $\parallel$ and $\perp$ denote respectively parallel and perpendicular components (relative to the scattering frame) of the incident and scattered fields. S_1–S_4 are elements of the amplitude matrix (§10.2). Corresponding differential cross sections can be obtained from scattering intensities through division by k^2 [cf. Eq. (41)].

In applications involving preferential direction it is customary to denote two components of the incident and scattered fields as vertical (v) and horizontal (h). The correspondence between those and $\parallel$, $\perp$ depends on a particular application. For instance, if all considered scattering planes contain the *z*-axis, $\parallel$ and $\perp$ correspond to v and h respectively [58].

Another related quantity is (radar) backscattering crosssection. The name itself is somewhat misleading, since it denotes not the integral quantity (as in §10.3) but the diffential scattering cross section at exact backscattering direction multiplied by 4π [42]. It can be decomposed into different components, as in Eq. (42), $\sigma_{\alpha\beta}=4\pi I_{\alpha\beta}/k^2$, where α and β can be replaced by $\parallel$, $\perp$, v, h, *x* or *y* (see, e.g., [59]). Moreover, the expressions can be simplified using the symmetry fact – $S_{12}=S_{21}$ for $\theta=180°$ [60]. Therefore, one can obtain the following crosssection using the values in file `mueller` for $\theta=180°$ ($\varphi=90°$), usually in the last line:

$$\sigma_{xx}=\frac{2\pi}{k^2}\left(S_{11}+S_{22}-2S_{12}\right),\quad \sigma_{yy}=\frac{2\pi}{k^2}\left(S_{11}+S_{22}+2S_{12}\right),\quad \sigma_{xy}=\sigma_{yx}=\frac{2\pi}{k^2}\left(S_{11}-S_{22}\right). \tag{43}$$

10.2 Amplitude matrix

An alternative approach to characterize the scattering process is by 2×2 dimensionless complex matrix, the amplitude scattering matrix, which transforms amplitude of the incident field (in plane perpendicular to propagation direction) into that of the scattered field [42]. Compared to the intensity-based Mueller matrix the amplitude matrix contains also the phase of the field, but it can be measured only by limited number of experimental set-ups, mainly in microwave frequency domain.

`ADDA` can calculate the amplitude matrix when enabled by the command line option

```
-scat_matr {muel|ampl|both|none}
```

which allows one to specify whether the Mueller matrix (`muel`, the default choice), the amplitude matrix (`ampl`), `both`, or `none` should be calculated and saved to file.

The set of angles to calculate the amplitude matrix are determined the same way as for the Mueller matrix (§10.1). In particular, by default a uniform grid in the *yz*-plane is used and the result is saved into file `ampl` (§C.6). When `-store_scat_grid` is used, the amplitude matrix for a grid of scattering angles is saved to file `ampl_scatgrid` (§C.6). However, there

is one important difference – no averaging of the amplitude matrix is performed neither over orientation (`-orient avg`) nor over the azimuthal scattering angle (`-phi_integr`).[46]

10.3 Integral scattering quantities

All scattering quantities described in this section are saved to several files (§C.7), corresponding to each of two incident polarizations (`CrossSec-X` and `CrossSec-Y`) and to orientation-averaged results (`CrossSec`). For the latter the result is calculated for unpolarized incident light. However, it is equal to that for any incident polarization, if the full range of Euler angle α is used for averaging (§7.2). ADDA always calculates C_{ext} and C_{abs} (together with corresponding efficiencies Q_{ext}, Q_{abs}). Optionally, it can calculate scattering cross section C_{sca} (and efficiency Q_{sca}) and normalized and non-normalized asymmetry vectors – $\mathbf{g}$ and $\mathbf{g}C_{\text{sca}}$ respectively (the z-component of $\mathbf{g}$ is the usual asymmetry parameter $\langle\cos\theta\rangle$). Values of cross sections are in units of μm^2. All the efficiencies are calculated by dividing the corresponding cross section over the area of the geometrical cross section of the sphere with volume equal to that of dipole representation of the particle, $C_{\text{eq}} = \pi r_{\text{eq}}^2$. Optional calculation of C_{sca}, $\mathbf{g}C_{\text{sca}}$ and $\mathbf{g}$ can be enabled, respectively, by command line options

```
-Csca
-vec
-asym
```

If $\mathbf{g}$ is calculated C_{sca} and $\mathbf{g}C_{\text{sca}}$ are also calculated automatically.[47] The calculation of $\mathbf{g}$ and C_{sca} is performed by integration over the whole solid angle, using a grid of scattering angles. The grid is specified by default in file `alldir_params.dat` (see §B.3 for format) in a form suitable for the Romberg integration (§11.6). Integration points for θ are spaced uniformly in values of $\cos\theta$. Different grid file can be specified through the command line option

```
-alldir_inp <filename>
```

In most cases C_{sca} can be accurately calculated as $C_{\text{ext}} - C_{\text{abs}}$, without using `-Csca` option. As discussed in §9.3 accuracy of this method can be improved when $C_{\text{sca}} \ll C_{\text{abs}}$ by using smaller ε (§11.1). Whether this is more computationally effective than using `-Csca` or not depends on the particular problem. For instance, C_{ext} and C_{abs} may coincide in (almost) all digits saved to file for particles much smaller than the wavelength, leaving integration of scattered fields over the solid angle as the only viable option. However, in this case the integration can be done analytically, using formulae for the Rayleigh regime. For a general non-symmetric particle formulae are cumbersome. However, if total particle polarizability is a diagonal tensor in the laboratory reference frame (e.g. general ellipsoid with axes along coordinate axes), then starting from chapter 5.7 of [42] one can derive:

$$C_{\text{sca}}^{x,y} = \frac{8\pi}{3k^2}\left(S_{11} \mp S_{12}\right)\Big|_{\theta=0,\,\varphi=\pi/2}, \tag{44}$$

where plus and minus correspond to incident polarization along the y- and x-axis respectively. Mueller matrix elements are considered at the forward direction in the yz-plane, i.e. they can be found in the second line of the `mueller` file (§10.1).

Currently ADDA provides no built-in functionality for automatically scanning over a wavelength spectrum. However, a simple workaround is to execute ADDA for each wavelength independently, using a script, and then collect the output data.[48]

---

[46] To the best of our knowledge, all applications involving averaging over particle orientation or over azimuthal scattering angle deal with incoherent addition, i.e. the intensity, not the field amplitude, is averaged.

[47] i.e. `-asym` implies `-Csca` and `-vec`.

[48] http://code.google.com/p/a-dda/issues/detail?id=35

10.4 Radiation forces

Radiation force for the whole scatterer and for each dipole can also be calculated by ADDA. If the command line option

```
-Cpr
```

is specified, the radiation force (pressure) cross section $\mathbf{C}_{\text{pr}}$ and efficiency $\mathbf{Q}_{\text{pr}}$ (vectors) are calculated and saved into file `CrossSec` (§C.7). Total time-averaged force $\mathbf{F}_{\text{rad}}$ acting on the particle is proportional to $\mathbf{C}_{\text{pr}}$ [61,62]:

$$\mathbf{F}_{\text{rad}} = \mathbf{C}_{\text{pr}} w_{\text{rad}}, \quad w_{\text{rad}} = m_0^2 \frac{|\mathbf{E}_0|^2}{8\pi} = \frac{m_0 I_0}{c}, \quad \mathbf{C}_{\text{pr}} = \pi r_{\text{eq}}^2 \mathbf{Q}_{\text{pr}}, \tag{45}$$

where w_{rad} is time-averaged energy density (or pressure) of the electric field, $\mathbf{E}_0$ and I_0 are the electric field amplitude (complex vector) and the time-averaged irradiance of the incident beam. For non-plane beams (§8.2), $\mathbf{E}_0$ and I_0 are considered in the focal point of a beam. The formula connecting w_{rad} and $\mathbf{E}_0$ is given for Gaussian-CGS system of units, which is used throughout ADDA. In SI "$1/8\pi$" should be replaced by "$\varepsilon_0/2$". However, the relation between w_{rad} and I_0 is the same for both systems. It is important to note, that Eq. (45) explicitly contains the refractive index of the surrounding medium m_0. Hence, $\mathbf{C}_{\text{pr}}$ calculated for equivalent scattering problem of particle in the vacuum (§4) is the correct one, but $\mathbf{F}_{\text{rad}}$ should be additionally multiplied by m_0 (for the same I_0). Relation between I_0 and the total beam power is discussed in §8.2.

To save the radiation forces on each dipole into file `RadForce` (§C.8) use

```
-store_force
```

This option implies `-Cpr` and can be used, e.g., to study internal stress inside the particle or forces and/or torques acting on separate parts of a multi-particle system. Unfortunately, currently there is no automatic way to calculate optical torque on the particle. However, using the data from file `RadForce` (coordinates of point dipoles and applied forces) one can easily calculate the total torque using simple mechanical reasoning. It is important to note that these dipole-wise radiation forces are given in the particle reference frame, while integral quantities, saved in `CrossSec`, are given in the incident wave reference frame (§6.1).

The calculation of optical force on each dipole implicitly assumes unity amplitude of the incident beam ($|\mathbf{E}_0| = 1$ statV/cm). Hence, the unit of force in file `RadForce` is 10^{-8} dyne when all lengths are measured in μm (see §2). For other value of $|\mathbf{E}_0|$, the result should be multiplied by $|\mathbf{E}_0|^2$. For instance, if $|\mathbf{E}_0| = 1$ V/m the unit of force is 1.11×10^{-22} N. Moreover, to account for surrounding medium one should multiply the result of the "scaled" problem by m_0^2 and m_0 for the same $\mathbf{E}_0$ and I_0 respectively.

For a plane incident wave $\mathbf{C}_{\text{pr}}$ can also be calculated from momentum conservation [61]:

$$\mathbf{C}_{\text{pr}} = \mathbf{a} C_{\text{ext}} - \mathbf{g} C_{\text{sca}}, \tag{46}$$

where the two terms correspond to momentum extracted from the incident wave and contained in the scattered wave, respectively. The same distinction also applies to force on each dipole [7], and they are calculated separately by ADDA. For convenience, these two resulting vectors are also saved to file `CrossSec`, when `-Cpr` is used. They can be directly compared to the values, obtained by the default approach (§10.3). Unfortunately, Eq. (46) does not hold for non-plane beams, in particular, momentum extracted from the incident wave is no more directly proportional to C_{ext} (extracted energy).

It is important to note that calculation of radiation forces is still under development. There are two major drawbacks of the current implementation:

- It cannot be used in combination with non-plane beams (§8.2), due to more complex expressions for "extinction" force.[49]

It is computationally inefficient with speed $O(N_{\text{real}}^2)$, also it occupies $O(N_{\text{real}})$ extra memory in parallel mode.[50] It is planned to rewrite this implementation according to Hoekstra *et al.* [7], which will make it $O(N \ln N)$ – usually even faster than calculation of $\mathbf{g}C_{\text{sca}}$ by integration of the scattered field (§10.3). However, currently the fastest ways to obtain $\mathbf{C}_{\text{pr}}$ for a *plane* incident wave is through Eq. (46) without calculating the radiation forces at all.

10.5 Internal fields and dipole polarizations

ADDA can save internal electric fields $\mathbf{E}_i$ and/or dipole polarizations $\mathbf{P}_i$ at each dipole (§9) to files `IntField` and `DipPol` respectively (§C.9), using command line options

```
-store_int_field
-store_dip_pol
```

These options allow one to study in details the accuracy of the DDA (see e.g. [17]) or the physics behind it. To save internal fields for future use by ADDA consider using checkpoints of type "always" (§11.5).

10.6 Near-field

Near-field is the electric field near the particle, i.e. neither in the particle itself nor far enough to be considered a scattered field. Currently, ADDA can not calculate the near-field in a completely convenient manner. However, Fabio Della Sala and Stefania D'Agostino [32] have contributed a package `near_field`, which adds this functionality using the dipole polarizations calculated by ADDA. This package is distributed in the `misc/` folder, and is accompanied by all necessary instructions.

While the above option is recommended one, there exist another workaround that may be preferential in some special cases. One may include all the points, where near-field need to be calculated, in the particle geometry, but set their refractive index to a value very close to 1 (e.g. 1.00001). Then the scattering problem is identical to the initial one. Saving the internal fields (§10.5) will directly produce the required near-field together with the field inside the particle. The drawback of this workaround is increase of computational time, which depends on the size of the box containing the extended particle geometry. Moreover, a user is required to specify a shape file, except for the simplest cases. For instance, predefined shapes "`coated`" or "`sphere box`" (§6.4) can be used to calculate near-field in a sphere or cube around the homogenous sphere, respectively. However, this method can be much faster when full 3D picture of the near field is required, because it implicitly uses FFT-acceleration of the simulations. This idea was further developed for even more efficient calculation of the near-field [63], but this approach has not been yet implemented in ADDA.

11 Computational Issues

11.1 Iterative solver

The main computation of a DDA simulation, usually taking the major part the execution time, is finding a solution of a large system of linear equations. ADDA uses an alternative form of Eq. (13):

[49] http://code.google.com/p/a-dda/issues/detail?id=135
[50] http://code.google.com/p/a-dda/issues/detail?id=14

$$\sum_j \mathbf{A}_{ij}\mathbf{x}_j = \overline{\boldsymbol{\beta}}_i \mathbf{E}_i^{\mathrm{inc}}, \text{ where}$$
$$\overline{\boldsymbol{\alpha}}_i = \overline{\boldsymbol{\beta}}_i^T \overline{\boldsymbol{\beta}}_i, \;\; \mathbf{x}_i = \overline{\boldsymbol{\beta}}_i \mathbf{E}_i^{\mathrm{exc}} = \overline{\boldsymbol{\beta}}_i^{-T} \mathbf{P}_i = \overline{\boldsymbol{\beta}}_i^{-T} V_{\mathrm{d}} \chi_i \mathbf{E}_i, \;\; \overline{\mathbf{A}}_{ij} = \overline{\mathbf{I}}\delta_{ij} - \overline{\boldsymbol{\beta}}_i \overline{\mathbf{G}}_{ij} \overline{\boldsymbol{\beta}}_j^T \quad . \tag{47}$$

Decomposition of $\overline{\boldsymbol{\alpha}}_i$ is possible if and only if it is complex-symmetric. This is sufficient for the current version of `ADDA`, which supports only diagonal polarizability tensors. Since $\overline{\mathbf{G}}_{ij} = \overline{\mathbf{G}}_{ji}$, the interaction matrix $\mathbf{A}$ is complex–symmetric and Jacobi-preconditioned (has unity diagonal). `ADDA` incorporates several different iterative methods for solution of Eq. (47): Bi-conjugate gradient (Bi-CG) [64,65], Bi-CG stabilized (Bi-CGStab) [66], enhanced Bi-CGStab(2) [67], conjugate gradient applied to normalized equations with minimization of the residual norm (CGNR) [66], CSYM [68], quasi minimal residual (QMR) [64] and its modification based on 2-term recurrence (QMR_2) [69]. Adding of a new iterative solver is straightforward for anyone who is familiar with C programming language.[51]

Comparison of the iterative solvers in terms of matrix–vector products (MVPs) per iteration and required number of vectors (memory, see §5) is presented in Table 2. Bi-CG, CSYM, QMR and QMR_2 employ the complex symmetric property of $\mathbf{A}$ to calculate only one MVP per iteration [64]. Our experience suggests that QMR is usually the fastest iterative solver, however in some cases Bi-CGStab may be faster (also their modern variants, namely QMR_2 and Bi-CGStab(2) respectively, may be preferred – see below). Performance of Bi-CG is comparable to that of QMR, but convergence behavior of the former is erratic, similar to that of Bi-CGStab. Still, Bi-CG may be preferred when memory is sparse. While being the slowest, CGNR is very simple and its convergence is guaranteed to be monotonic [66]. CSYM has the same properties, and is expected to be faster than CGNR, in terms of MVPs [68]. QMR_2 is expected to be faster than QMR in finite-precision arithmeticis [69] and requires slightly less memory. Bi-CGStab(2) is expected to be faster than Bi-CGStab [67] in terms of MVPs, but requires slightly more memory.

The iterative solver is chosen by the command line option

```
-iter <type>
```

where `<type>` is one of the values specified in Table 2. By default QMR is used. There are several options to choose the initial field, set by the command line option

```
-init_field {auto|inc|read <filenameY> [<filenameX>]|wkb|zero}
```

where `zero` vector is a general approach and `inc` (derived from the incident field, more precisely – equal to the right-hand-side of the linear system) may be closer to the exact solution for small index-matching particles. `wkb` is derived from the incident field corrected for phase shift during propagation in the particle (Wentzel–Kramers–Brillouin approximation [70]). It can be considered as an extension of `inc`, but currently it works only for default incident beam – plane wave propagating along the *z*-axis in particle reference frame (§8).[52] By default (`auto`) `ADDA` automatically choses from `zero` and `inc` the one with lesser residual norm.[53] Finally, the starting field can also be `read` from files. Normally two files are required for *y*- and *x*-polarizations respectively (§8.1), but a single filename is sufficient if only *y*-polarization is used (e.g. due to symmetry). Incident field should be specified in a *particle* reference frame, see §B.7 for file format. The choice of initial field is shown in the `log` (§C.4).

[51] See `http://code.google.com/p/a-dda/wiki/AddingIterativeSolver` for detailed instructions.
[52] `http://code.google.com/p/a-dda/issues/detail?id=59`
[53] It should be noted, however, that smaller residual of the initial vector does not necessarily leads to a faster convergence [66].

Table 2. Characteristics of the iterative solvers.

<type>	Name	Sym[a]	MVPs per iteration	# of vectors[b]	Max stagnation
bcgs2	Bi-CGStab(2)	-	4	8	15 000
bicg	Bi-CG	+	1	4	50 000
bicgstab	Bi-CGStab	-	2	7	30 000
cgnr	CGNR	-	2	4	10
csym	CSYM	+	1	6	10
qmr	QMR	+	1	7	50 000
qmr2	QMR_2	+	1	6	50 000

[a] whether complex symmetric property of the matrix is explicitly used.
[b] includes input residual and output solution, but does not include incident beam.

It is important to note, that initial field `inc` corresponds to $\mathbf{x}_{(0)i} = \overline{\boldsymbol{\beta}}_i \mathbf{E}_i^{\text{inc}} \Leftrightarrow \mathbf{E}_{(0)}^{\text{exc}} = \mathbf{E}^{\text{inc}}$, i.e. it sets exciting field, while both `wkb` and `read` define internal field $\mathbf{E}_{(0)}$, cf. Eq. (47). In particular, "`-beam read fY fX –init_field inc`" is not completely equivalent to "`-init_field read fY fX`" in terms of the starting vector. The connection between the two fields depends on the polarizability formulation (§9.1), but can be approximately described as $\mathbf{E}_i \approx \mathbf{E}_i^{\text{exc}} 3/(\varepsilon_i + 2)$ (this equation is exact for CM polarizability).

The stopping criterion for iterative solvers is the relative norm of the residual. The process stops when this norm is less than ε, which can be specified by the command line option

```
-eps <arg>
```

where $\varepsilon = 10^{-\langle \text{arg} \rangle}$. By default $\varepsilon = 10^{-5}$. It is important to note that the residual is updated iteratively along the execution of the iterative solver, which may cause substantial round-off errors for large number of iterations. True residual at the end of the iterative solver can be calculated by `ADDA` through additional MVP. For that specify

```
-recalc_resid
```

in the command line. The maximum number of iterations can be specified as `<arg>` to the command line option

```
-maxiter <arg>
```

`ADDA` will stop execution if the iterative solver does not converge in the given number of iterations. By default the maximum number of iterations is set to a very high value, which is not expected to be ever reached.[54] `ADDA` will also stop if the iterative solver stagnates for a long time, i.e. residual norm do not decrease during a number of last iterations. Corresponding numbers are specified in Table 2.[55]

All iterative solvers except CGNR and CSYM are susceptible to breakdowns. Although such situations are very improbable, they may happen in specific cases. For instance, simulation of light scattering by cube with edge length that is exactly a multiple of the wavelength results in breakdowns of QMR, QMR_2, and Bi-CG. `ADDA` should detect almost any breakdown and produce a meaningful error message; however, it is possible that some breakdown is not detected. It then results in the stagnation of the residual norm or its oscillations without any further progress. On contrary, `ADDA` may claim breakdown when the convergence is slow but still satisfactory. The behavior of breakdown detection may be modified,[56] but it requires recompilation of the code.

[54] Currently it is set to $3N$, i.e. the number of equations in a linear system.
[55] They are defined in structure array `params` in the beginning of the `iterative.c`.
[56] Changing values in lines starting with "`#define EPS`" in the beginning of a function, implementing a certain iterative solver in `iterative.c`.

If you notice any error of the breakdown detection, both false-negative and false-positive, please communicate it to the developers. If the breakdown does appear, the solution is either to try different iterative solver or adjust input parameters a little bit. Usually changing particle size (or wavelength) by only 0.01% completely eliminates the breakdown.

11.2 Fast Fourier transform

The iterative method only needs the interaction matrix for calculating matrix–vector products. This can be done in O(Nln$N$) operations (where N is the total number of dipoles) using the FFT [71]. In ADDA 3D (parallel) FFT is explicitly decomposed into a set of 1D FFTs, reducing calculations (see [6] for details).

1D FFTs are performed using standard libraries – two are implemented in ADDA: a routine by Temperton (`CFFT99`, [72]), or the more advanced package `FFTW3` [73]. The latter is generally significantly faster, but requires separate installation of the package.[57] The FFT routine to use is chosen at compile time: default is `FFTW3`, and another one is turned on by compilation option `FFT_TEMPERTON`.[12]

FFT is performed on the grid that is doubled in each dimension compared to the computational grid. Temperton's FFT requires that the dimensions of this grid be of the form $2^p3^q5^r$ (all exponents are integers), `FFTW3` works with any grid dimensions but is most effective for dimensions of the form $2^p3^q5^r7^s11^u13^v$ ($u + v \leq 1$). It should not be a limitation for the sequential mode, since ADDA automatically increases the FFT-grid dimensions to the first number of the required form.[58] But in parallel mode these dimensions must also divide the number of processors. Therefore the increase of the dimensions (and hence simulation time) may be substantial, and not possible at all if the number of processors divide any prime number larger than 5 for Temperton FFT or has more than one divisor of either 11 or 13 (or at least one larger than 13) for `FFTW3`. Therefore it is strongly recommended *not* to use such "weird" number of processors.[59] It is the user's responsibility to optimize the combination of computational grid dimensions and number of processors, although ADDA will work, but probably not efficiently, for most of the combinations (see also §6.6).

In OpenCL mode ADDA employs one of the two FFT implementations. The default one is `clAmdFft`, which works for sizes of the form $2^p3^q5^r$, but requires separate package installation[60] and may have problems when run on Nvidia GPUs.[61] Alternative implementation (by Apple) is turned on by compilation option `CLFFT_APPLE`.[12] It is limited to power-of-two sizes and is significantly slower than `clAmdFft`. For both implementations ADDA will automatically adjust the FFT-grid dimensions to the minimum supported size.

Symmetry of the DDA interaction matrix is used in ADDA to reduce the storage space for the Fourier-transformed matrix, except when SO formulae to calculate interaction term are used (§9.2). This option can be disabled (mainly for debugging purposes) by specifying

```
-no_reduced_fft
```

in the command line.

11.3 Sparse mode

For cases when $N \gg N_{\text{real}}$, e.g. for very sparse (porous) aggregates, FFT may not provide desired acceleration. Specifically for such cases there is a special compilation mode of ADDA, named "sparse mode". It is a direct implementation of matrix–vector product in the DDA

[57] http://code.google.com/p/a-dda/wiki/InstallingFFTW3
[58] The maximum increase is 15% for Temperton FFT and 6% for `FFTW3`.
[59] Otherwise Temperton FFT will fail and `FFTW3` will perform less efficiently.
[60] http://code.google.com/p/a-dda/wiki/InstallingclAmdFft
[61] http://code.google.com/p/a-dda/issues/detail?id=157

without FFT. Its main advantage is that the empty dipoles are not included at all in the data model, which decreases required computer memory and simplifies MPI parallelization. The effect on computational speed depends on the porosity (sparseness) of the particle: the standard (FFT) mode is faster for moderately porous particles, while the sparse mode is faster for extremely porous ones. In particular, the sparse ADDA was used for simulations of radar backscattering by snowflakes [74].

In general, usage of sparse mode is very similar to the FFT mode, discarding all FFT-related aspects. However there are certain limitations (incompatibilities with some of other ADDA features), which are discussed at the corresponding wiki page,[62] together with compilation and usage instructions.

System requirements for sparse mode are significantly different from the standard mode (§5). Memory requirements are

$$M_{\text{tot}} = \left[265 \div 457 + 60 n_p\right] N_{\text{real}} \text{ bytes}, \quad M_{\text{pp}} = M_{\text{tot}} / n_p , \tag{48}$$

where definitions of §5 are used and range of numbers correspond to different iterative solvers. A term proportional to n_p indicates poor parallel scaling of the memory for the current implementation, which will be fixed in the future.[63] Computational time (more specifically, time of one iteration) scales as $\mathrm{O}(N_{\text{real}}^2)$, and this time almost perfectly divides by n_p during parallel execution.

While the developers do their best to ensure reliability of sparse mode, it is a good idea to perform selected tests of this mode against the standard mode on a few sample problems before performing a large set of simulations.

11.4 Parallel performance

ADDA is capable of running on a multiprocessor system, parallelizing a single DDA simulation. It uses MPI for communication routines. This feature can be used both to accelerate the computations and to circumvent the single-computer limit of the available memory, thus enabling simulations with a huge number of dipoles. We do not discuss the parallel efficiency of ADDA, which largely depends on the hardware. However, in our experience it was usually larger than 70% on modern computer clusters, and close to 100% in sparse mode (§11.3). It is also important to note that the MPI interface may occupy significant amount of RAM on each node, thereby decreasing the RAM available for ADDA itself (see also §5 and §6.6).

11.5 Checkpoints

To facilitate very long simulations ADDA incorporates a checkpoint system, which can be used to break a single simulation into smaller parts. All the intermediate vectors of the iterative solver (§11.1) are saved, which allows restarting the iterative solver exactly at the position, where the checkpoint was created. Time of a checkpoint is specified by command line option

```
-chpoint <time>
```

where `<time>` is in format "`#d#h#m#s`".[64] There are 3 possible strategies for checkpoints, chosen through the command line option

```
-chp_type <type>
```

where `<type>` is one of `normal`, `regular`, `always`. "Normal" means that after the checkpoint time elapsed, the checkpoint is saved as soon as possible (it waits for the finishing of the current iteration) and ADDA finishes execution without any further actions. This type is

[62] http://code.google.com/p/a-dda/wiki/SparseMode
[63] http://code.google.com/p/a-dda/issues/detail?id=160
[64] All fields are optional, "`s`" can be omitted, the format is not case sensitive. For example: "`12h30M`", "`1D10s`", "`3600`" (equals 1 hour).

useful when one need ADDA to run not longer than certain time. “Regular” checkpoints are saved after every specified time interval but do not influence the normal execution of ADDA – it runs until simulation is fully completed. Use this option when abrupt termination of ADDA may occur (e.g. system crash or the system resources are urgently needed for other tasks). “Always” type is similar to “normal” if checkpoint time elapsed during the execution, however it will also save a checkpoint (after the last iteration) when ADDA finishes normally earlier. That is the only checkpoint type, for which time may be not specified (then it is equivalent to infinite time). It may be useful if the simulation is valuable by itself but may be extended in the future, e.g. to obtain better convergence (lower ε, §11.1) or to calculate different scattering quantities (§10).

To restart the simulation from a checkpoint

```
-chp_load
```

should be specified in the command line. The user should take care that the simulation is restarted with the same parameters that were used when saving the checkpoint. Although some parameters can indeed be different (e.g. those determining the output of ADDA), the consistency of the results is user’s responsibility. By default all the checkpoint data is saved in the directory `chpoint` (§D.2), however a different directory (the same for saving and loading of checkpoints) can be specified through the command line option

```
-chp_dir <dirname>
```

The total size of the checkpoint files is approximately half of the RAM used, therefore 1) enough space on the hard disk should be available; 2) saving of a checkpoint may take considerable time. Both issues are especially relevant for large simulations on multiprocessor systems. If the simulation time is strictly limited (e.g. by a batch system of a supercomputer with shared usage) checkpoint time should be set slightly smaller, so that ADDA would have enough time to finish the iteration and save a checkpoint (and possibly to calculate the scattering quantities if the iterative solver will converge just before the checkpoint time). User should estimate the needed time reserve himself. When loading a checkpoint, ADDA initializes anew, this takes some time. However, this time is usually small compared to the time used for the iterations.

It is also important to note that by default the same checkpoint directory is used for all the simulations on the current system that are run from the same path, therefore new checkpoints overwrite the old ones.[65] To avoid it specify a different checkpoint directory for each instance of ADDA; it is obligatory when several instances of ADDA run in parallel. For now, ADDA *always* saves checkpoints into the same directory where it loads it from.

Currently only the state of the iterative solver is saved at a checkpoint, therefore it is suitable only for simulations for a single incident polarization.

11.6 Romberg integration

Integration is performed in several parts of ADDA: orientation averaging (§7.2), integration of the Mueller matrix over the azimuthal angle (§10.1), and integration of the scattered field over the whole solid angle (§10.3). The same routine is used for all these purposes, which is based on the one- or two-dimensional Romberg integration [75]. This is a high-order technique that may be used in adaptive mode, automatically performing only the necessary number of function evaluations to reach a prescribed accuracy. The latter is relevant for orientation averaging, where each function evaluation is a complete DDA simulation, but not for integration over scattering angles, for which all values are precalculated. Romberg integration also provides an estimate of the integration error, which is reliable for “sufficiently nice” functions [75]. When 2D integration is performed ADDA integrates an error estimate obtained

[65] This is done so to save the hard disk space.

in the inner integration loop simultaneously with the function values, resulting in a reliable estimate of the final error in the outer loop. The information about the integration together with errors is saved to separate log files (§C.10): `log_orient_avg`, `log_int_Csca-X`, `log_int_asym_x-X`, `log_int_asym_y-X`, `log_int_asym_z-X` for orientation averaging, calculation of C_{sca} and each component of **g** respectively (the last 4 file names have different suffixes – `X` or `Y` – for different incident polarizations). For orientation averaging some information is saved to the main `log` (§C.4). For integration of the Mueller matrix over the azimuthal angle only the averaged errors are saved together with the values directly to `mueller_integr` files (§C.5)

The drawback of the Romberg integration is that argument values must be uniformly spaced and their total number is limited to be $2^n + 1$ (n is any integer). The set of integration points is specified by minimum and maximum values and minimum and maximum number of subdivisions (refinements) J_{min} and J_{max} (the latter is equal to n). The required accuracy to reach is also a parameter. In some cases minimum and maximum values of some arguments are equivalent (e.g. 0° and 360° for φ), `ADDA` accounts for it to slightly decrease simulation time.

If the function is "nice" and periodic over the interval of integration, then it is not beneficial and may be even harmful to use higher order integration scheme. For such cases the simplest trapezoid rule (or equivalently midpoint rule) is the most efficient [75]. The same is true, when e.g. the function is integrated over the half of its period and is symmetric with respect to both borders of the integration interval. `ADDA`'s behavior is determined by the flag `periodic` in the input files. If it is `true` then trapezoid rule is used as a special case of the general Romberg scheme. This keeps the limitation on the number of integration points, but provides adaptability and reliable error estimates. All the integration parameters are specified in input files: `avg_params.dat` (§B.2), `scat_params.dat` (§B.4), `alldir_params.dat` (§B.3) corresponding to different integration tasks.

12 Timing

12.1 Basic timing

The basic timing of `ADDA` execution on a single processor is performed using standard ANSI C functions `clock` and `time`, which are completely portable. The drawbacks are low precision (1 s) of wall-time and low precision (0.1 s on most systems) and possible overflows (after 1 hour on 32-bit systems) of the processor timer. `ADDA` uses wall-time only for the total execution time and timing of checkpoints (§11.5), and for the rest processor time is measured. It makes timing more precise especially on desktop computers that are partly occupied by other tasks; however, for long simulations some timing results may become meaningless because of the timer overflow. The problem with `clock` overflows should not be relevant in 64-bit environment because of larger size of `clock_t` type. This timing may also be inaccurate in OpenCL mode, since the standard doesn't defines whether `clock` includes the time of GPU calculations. This can be assessed by comparing total processor and wall times, given in `log` (§C.4).

In parallel mode all the times are wall-times measured by the high-precision `MPI_Wtime` function, which is a part of MPI standard. This solves above-mentioned problems of the ANSI C timers.

Timing results are presented in the end of the `log` (§C.4) together with some statistics (total number of iterations, total number of planes where the scattered field is calculated). It covers all major parts of `ADDA` execution: initialization (including initialization of FFT, §11.2, building the interaction matrix, and constructing a dipole representation of the particle, §6.3),

solution for the internal fields (including generation of incident beam, §8, and iterative solver, §11.1), calculation of the scattering quantities (scattered electric field and others, §10), input/output (including checkpoint loading/saving, §11.5), integration (§11.6). Some are divided into subsections. If `-prognosis` option is used, only the time for constructing a particle is shown.

In parallel mode communication time (between different processors) is shown separately for some tasks, which can be used to estimate the parallel efficiency. To measure this time accurately synchronization is done before the communication takes place. Our experience shows that this should not decrease the performance of ADDA, except for the small effect in the granule generator (§6.5).

12.2 Precise timing

This feature of ADDA is used to perform the thorough timing of the most computationally intensive parts: initialization of interaction matrix and FFT (especially `FFTW3`, §11.2) and matrix–vector product. It gives the detailed information both on FFT and algebraic parts, which can be used for deep optimization or performance studies. However, this regime is incompatible with the normal ADDA execution – it terminates after the first matrix–vector product. Only the `stdout` is produced (§C.2). Precision of the precise timing routines are of order μs, however they measure wall-time[66] and are operating-system-dependent. The latter should not be a big problem, since ADDA contains routines for any POSIX or Windows operating systems; the appropriate one is automatically chosen by compiler directives.[67] To enable precise timing recompile ADDA with option `PRECISE_TIMING`.[12]

13 Miscellanea

Additional files contributed by users will be located in the directory `misc/`. These files should be considered to be supported (to some extent) by their respective authors. They should be accompanied by enough information to explain their use. A brief description of the available tools is given together with description of ADDA package.[13] Moreover, being a command line tool ADDA easily lends itself to scripting and automatization. In particular, there is a wrapper in language R with several usage examples[68] – it can be easily generalized to other scripting languages and applications.

14 Finale

This manual is somewhat inelegant, but we hope that it will prove useful. The structure of the input files is intended to be simple and suggestive so that, after reading the above notes once, the users may not have to refer to them again. Command line help system (`-h` option, §3.1) is intended to provide quick reference on different command line options. If that is not enough, thorough description of all command line options can be found in this manual (§A). A lot of other information about ADDA can be found in corresponding wiki-pages,[69] ranging from compiling instructions to description of how to add new predefined shapes. In particular, an up-to-date list of acknowledgements is available,[70] including contributors and funding sources.

---

[66] It is hard to implement portable measurement of processor time with better precision than that of `clock`.

[67] Timing routines for some other operating system may be implemented instead of the current ones in source files `prec_timing.c` and `prec_timing.h`.

[68] https://github.com/baptiste/adda/wiki/wrapper_primer

[69] See http://code.google.com/p/a-dda/wiki/GettingStarted for an overview.

[70] http://code.google.com/p/a-dda/wiki/Acknowledgements

Users are encouraged to subscribe to adda-announce Google group[7]; bug reports and any new releases of ADDA will be made known to those who do. All known bugs are available at issue tracker,[5] it also lists the future plan for ADDA development. Users may "star" issues, which they consider important, to influence their priorities (Google account is required for that). Specific suggestions for improving ADDA and this manual are welcomed. You may either submit it directly to the issue tracker[5] or discuss it in the ADDA discussion group.[9] A list of frequently asked questions is also available.[8]

15 References

[1] M.A. Yurkin and A.G. Hoekstra, "The discrete-dipole-approximation code ADDA: capabilities and known limitations," *J. Quant. Spectrosc. Radiat. Transfer* **112**, 2234–2247 (2011) [doi:10.1016/j.jqsrt.2011.01.031].

[2] M.A. Yurkin and A.G. Hoekstra, "The discrete dipole approximation: an overview and recent developments," *J. Quant. Spectrosc. Radiat. Transfer* **106**, 558–589 (2007) [doi:10.1016/j.jqsrt.2007.01.034].

[3] E.M. Purcell and C.R. Pennypacker, "Scattering and adsorption of light by nonspherical dielectric grains," *Astrophys. J.* **186**, 705–714 (1973) [doi:10.1086/152538].

[4] A.G. Hoekstra and P.M.A. Sloot, "New computational techniques to simulate light-scattering from arbitrary particles," *Part. Part. Sys. Charact.* **11**, 189–193 (1994) [doi:10.1002/ppsc.19940110304].

[5] A.G. Hoekstra and P.M.A. Sloot, "Coupled dipole simulations of elastic light scattering on parallel systems," *Int. J. Mod. Phys. C* **6**, 663–679 (1995) [doi:10.1142/S0129183195000563].

[6] A.G. Hoekstra, M.D. Grimminck, and P.M.A. Sloot, "Large scale simulations of elastic light scattering by a fast discrete dipole approximation," *Int. J. Mod. Phys. C* **9**, 87–102 (1998) [doi:10.1142/S012918319800008X].

[7] A.G. Hoekstra, M. Frijlink, L.B.F.M. Waters, and P.M.A. Sloot, "Radiation forces in the discrete-dipole approximation," *J. Opt. Soc. Am. A* **18**, 1944–1953 (2001) [doi:10.1364/JOSAA.18.001944].

[8] M.A. Yurkin, V.P. Maltsev, and A.G. Hoekstra, "The discrete dipole approximation for simulation of light scattering by particles much larger than the wavelength," *J. Quant. Spectrosc. Radiat. Transfer* **106**, 546–557 (2007) [doi:10.1016/j.jqsrt.2007.01.033].

[9] M. Huntemann, G. Heygster, and G. Hong, "Discrete dipole approximation simulations on GPUs using OpenCL--Application on cloud ice particles," *J. Comput. Sci.* **2**, 262–271 (2011) [doi:10.1016/j.jocs.2011.05.011].

[10] B.T. Draine and P.J. Flatau, "Discrete-dipole approximation for scattering calculations," *J. Opt. Soc. Am. A* **11**, 1491–1499 (1994) [doi:10.1364/JOSAA.11.001491].

[11] B.T. Draine and J.J. Goodman, "Beyond Clausius–Mossotti: wave propagation on a polarizable point lattice and the discrete dipole approximation," *Astrophys. J.* **405**, 685–697 (1993) [doi:10.1086/172396].

[12] B.T. Draine, "The discrete dipole approximation for light scattering by irregular targets," in *Light Scattering by Nonspherical Particles, Theory, Measurements, and Applications*, M.I. Mishchenko, J.W. Hovenier, and L.D. Travis, Eds., pp. 131–145, Academic Press, New York (2000).

[13] N.B. Piller, "Coupled-dipole approximation for high permittivity materials," *Opt. Commun.* **160**, 10–14 (1999) [doi:10.1016/S0030-4018(98)00645-2].

[14] P.C. Chaumet, A. Sentenac, and A. Rahmani, "Coupled dipole method for scatterers with large permittivity," *Phys. Rev. E* **70**, 036606 (2004) [doi:10.1103/PhysRevE.70.036606].

[15] S.B. Singham, "Theoretical factors in modeling polarized light scattering by arbitrary particles," *Appl. Opt.* **28**, 5058–5064 (1989) [doi:10.1364/AO.28.005058].

[16] A.G. Hoekstra and P.M.A. Sloot, "Dipolar unit size in coupled-dipole calculations of the scattering matrix elements," *Opt. Lett.* **18**, 1211–1213 (1993) [doi:10.1364/OL.18.001211].
[17] A.G. Hoekstra, J. Rahola, and P.M.A. Sloot, "Accuracy of internal fields in volume integral equation simulations of light scattering," *Appl. Opt.* **37**, 8482–8497 (1998) [doi:10.1364/AO.37.008482].
[18] I. Ayranci, R. Vaillon, and N. Selcuk, "Performance of discrete dipole approximation for prediction of amplitude and phase of electromagnetic scattering by particles," *J. Quant. Spectrosc. Radiat. Transfer* **103**, 83–101 (2007) [doi:10.1016/j.jqsrt.2006.06.006].
[19] N.B. Piller and O.J.F. Martin, "Increasing the performance of the coupled-dipole approximation: A spectral approach," *IEEE Trans. Antennas Propag.* **46**, 1126–1137 (1998) [doi:10.1109/8.718567].
[20] M.A. Yurkin, D. de Kanter, and A.G. Hoekstra, "Accuracy of the discrete dipole approximation for simulation of optical properties of gold nanoparticles," *J. Nanophoton.* **4**, 041585 (2010) [doi:10.1117/1.3335329].
[21] M.A. Yurkin, V.P. Maltsev, and A.G. Hoekstra, "Convergence of the discrete dipole approximation. II. An extrapolation technique to increase the accuracy," *J. Opt. Soc. Am. A* **23**, 2592–2601 (2006) [doi:10.1364/JOSAA.23.002592].
[22] M.A. Yurkin, A.G. Hoekstra, R.S. Brock, and J.Q. Lu, "Systematic comparison of the discrete dipole approximation and the finite difference time domain method for large dielectric scatterers," *Opt. Express* **15**, 17902–17911 (2007) [doi:10.1364/OE.15.017902].
[23] M.A. Yurkin, M. Min, and A.G. Hoekstra, "Application of the discrete dipole approximation to very large refractive indices: Filtered coupled dipoles revived," *Phys. Rev. E* **82**, 036703 (2010) [doi:10.1103/PhysRevE.82.036703].
[24] K.V. Gilev, E. Eremina, M.A. Yurkin, and V.P. Maltsev, "Comparison of the discrete dipole approximation and the discrete source method for simulation of light scattering by red blood cells," *Opt. Express* **18**, 5681–5690 (2010) [doi:10.1364/OE.18.005681].
[25] V.L.Y. Loke and M.P. Menguc, "Surface waves and atomic force microscope probe-particle near-field coupling: discrete dipole approximation with surface interaction," *J. Opt. Soc. Am. A* **27**, 2293–2303 (2010) [doi:10.1364/JOSAA.27.002293].
[26] O.J.F. Martin and N.B. Piller, "Electromagnetic scattering in polarizable backgrounds," *Phys. Rev. E* **58**, 3909–3915 (1998) [doi:10.1103/PhysRevE.58.3909].
[27] R. Schmehl, B.M. Nebeker, and E.D. Hirleman, "Discrete-dipole approximation for scattering by features on surfaces by means of a two-dimensional fast Fourier transform technique," *J. Opt. Soc. Am. A* **14**, 3026–3036 (1997) [doi:10.1364/JOSAA.14.003026].
[28] D.W. Mackowski, "A generalization of image theory to predict the interaction of multipole fields with plane surfaces," *J. Quant. Spectrosc. Radiat. Transfer* **111**, 802–809 (2010) [doi:10.1016/j.jqsrt.2009.11.016].
[29] C.M.J. Wijers, "Rayleigh scattering from single-site polysylane adsorbed on silicon: Theory," *Surf. Sci.* **168**, 816–822 (1986) [doi:10.1016/0039-6028(86)90914-3].
[30] H. Parviainen and K. Lumme, "Scattering from rough thin films: discrete-dipole-approximation simulations," *J. Opt. Soc. Am. A* **25**, 90–97 (2008) [doi:10.1364/JOSAA.25.000090].
[31] D.W. Mackowski, "Direct simulation of scattering and absorption by particle deposits," in *Proceedings of the 10th Conference on Electromagnetic and Light Scattering*, G. Videen, M.I. Mishchenko, and M.P. Menguc, Eds., pp. 113–116, Bodrum, Turkey (2007) [doi:10.1615/ICHMT.2007.ConfElectromagLigScat.310].
[32] S. D'Agostino, P.P. Pompa, R. Chiuri, R.J. Phaneuf, D.G. Britti, R. Rinaldi, R. Cingolani, and F. Della Sala, "Enhanced fluorescence by metal nanospheres on metal substrates," *Opt. Lett.* **34**, 2381–2383 (2009) [doi:10.1364/OL.34.002381].

[33] P.C. Chaumet, A. Rahmani, and G.W. Bryant, "Generalization of the coupled dipole method to periodic structures," *Phys. Rev. B* **67**, 165404 (2003) [doi:10.1103/PhysRevB.67.165404].
[34] C.M.J. Wijers and K.M.E. Emmett, "Structural contribution to the anisotropic reflection from the Si (110) surface," *Phys. Scr.* **38**, 435–440 (1988) [doi:10.1088/0031-8949/38/3/017].
[35] B.T. Draine and P.J. Flatau, "Discrete-dipole approximation for periodic targets: theory and tests," *J. Opt. Soc. Am. A* **25**, 2693–2703 (2008) [doi:10.1364/JOSAA.25.002693].
[36] D.V. Hahn, D. Limsui, R.I. Joseph, K.C. Baldwin, N.T. Boggs, A.K. Carr, C.C. Carter, T.S. Han, and M.E. Thomas, "Shape characteristics of biological spores," in *SPIE Proceedings* **6954**, A.W. Fountain III and P.J. Gardner, Eds., p. 69540W, SPIE, Orlando, FL, USA (2008) [doi:10.1117/12.777637].
[37] P.W. Kuchel and E.D. Fackerell, "Parametric-equation representation of biconcave erythrocytes," *Bull. Math. Biol.* **61**, 209–220 (1999) [doi:10.1006/bulm.1998.0064].
[38] M.A. Yurkin, K.A. Semyanov, P.A. Tarasov, A.V. Chernyshev, A.G. Hoekstra, and V.P. Maltsev, "Experimental and theoretical study of light scattering by individual mature red blood cells with scanning flow cytometry and discrete dipole approximation," *Appl. Opt.* **44**, 5249–5256 (2005) [doi:10.1364/AO.44.005249].
[39] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.* **8**, 3–30 (1998) [doi:10.1145/272991.272995].
[40] M.A. Yurkin, K.A. Semyanov, V.P. Maltsev, and A.G. Hoekstra, "Discrimination of granulocyte subtypes from light scattering: theoretical analysis using a granulated sphere model," *Opt. Express* **15**, 16561–16580 (2007) [doi:10.1364/OE.15.016561].
[41] M.I. Mishchenko, "Calculation of the amplitude matrix for a nonspherical particle in a fixed orientation," *Appl. Opt.* **39**, 1026–1031 (2000) [doi:10.1364/AO.39.001026].
[42] C.F. Bohren and D.R. Huffman, *Absorption and Scattering of Light by Small Particles*, Wiley, New York (1983).
[43] G. Gouesbet, B. Maheu, and G. Grehan, "Light-scattering from a sphere arbitrarily located in a Gaussian beam, using a Bromwich formulation," *J. Opt. Soc. Am. A* **5**, 1427–1443 (1988) [doi:10.1364/JOSAA.5.001427].
[44] L.W. Davis, "Theory of electromagnetic beams," *Phys. Rev. A* **19**, 1177–1179 (1979) [doi:10.1103/PhysRevA.19.1177].
[45] J.P. Barton and D.R. Alexander, "Fifth-order corrected electromagnetic field components for a fundamental Gaussian beam," *J. Appl. Phys.* **66**, 2800–2802 (1989) [doi:10.1063/1.344207].
[46] J.D. Jackson, *Classical Electrodynamics*, 3rd ed., Wiley, New York (1998).
[47] B.T. Draine, "The discrete dipole approximation and its application to interstellar graphite grains," *Astrophys. J.* **333**, 848–872 (1988) [doi:10.1086/166795].
[48] G.H. Goedecke and S.G. O'Brien, "Scattering by irregular inhomogeneous particles via the digitized Green's function algorithm," *Appl. Opt.* **27**, 2431–2438 (1988) [doi:10.1364/AO.27.002431].
[49] M.A. Yurkin, "Computational approaches for plasmonics," in *Handbook of Molecular Plasmonics*, F. Della Sala and S. D'Agostino, Eds., pp. 83–135, Pan Stanford Publishing, Singapore (2013).
[50] D. Gutkowicz-Krusin and B.T. Draine, "Propagation of electromagnetic waves on a rectangular lattice of polarizable points" (2004) [http://arxiv.org/abs/astro-ph/0403082].
[51] A. Lakhtakia and G.W. Mulholland, "On 2 numerical techniques for light-scattering by dielectric agglomerated structures," *J. Res. Nat. Inst. Stand. Technol.* **98**, 699–716 (1993).
[52] N.B. Piller, "Influence of the edge meshes on the accuracy of the coupled-dipole approximation," *Opt. Lett.* **22**, 1674–1676 (1997) [doi:10.1364/OL.22.001674].

[53] P. Gay-Balmaz and O.J.F. Martin, "A library for computing the filtered and non-filtered 3D Green's tensor associated with infinite homogeneous space and surfaces," *Comput. Phys. Commun.* **144**, 111–120 (2002) [doi:10.1016/S0010-4655(01)00471-4].
[54] E. Zubko, K. Muinonen, Y. Shkuratov, G. Videen, and T. Nousiainen, "Scattering of light by roughened Gaussian random particles," *J. Quant. Spectrosc. Radiat. Transfer* **106**, 604–615 (2007) [doi:10.1016/j.jqsrt.2007.01.050].
[55] K. Schmidt, M.A. Yurkin, and M. Kahnert, "A case study on the reciprocity in light scattering computations," *Opt. Express* **20**, 23253–23274 (2012) [doi:10.1364/OE.20.023253].
[56] M.I. Mishchenko, L.D. Travis, and A.A. Lacis, *Scattering, Absorption, and Emission of Light by Small Particles*, Cambridge University Press, Cambridge (2002).
[57] V.P. Maltsev and K.A. Semyanov, *Characterisation of Bio-Particles from Light Scattering*, VSP, Utrecht (2004).
[58] L. Tsang, J.A. Kong, and K.H. Ding, *Scattering of Electromagnetic Waves: Theories and Applications*, Wiley, New York (2000).
[59] J. Tyynelä, T. Nousiainen, S. Göke, and K. Muinonen, "Modeling C-band single scattering properties of hydrometeors using discrete-dipole approximation and T-matrix method," *J. Quant. Spectrosc. Radiat. Transfer* **110**, 1654–1664 (2009) [doi:10.1016/j.jqsrt.2009.02.020].
[60] H.C. van de Hulst, *Light Scattering by Small Particles*, Dover, New York (1981).
[61] B.T. Draine and J.C. Weingartner, "Radiative torques on interstellar grains .1. Superthermal spin-up," *Astrophys. J.* **470**, 551–565 (1996) [doi:10.1086/177887].
[62] T.A. Nieminen, V.L.Y. Loke, A.B. Stilgoe, G. Knoner, A.M. Branczyk, N.R. Heckenberg, and H. Rubinsztein-Dunlop, "Optical tweezers computational toolbox," *J. Opt. A* **9**, S196–S203 (2007) [doi:10.1088/1464-4258/9/8/S12].
[63] P.J. Flatau and B.T. Draine, "Fast near field calculations in the discrete dipole approximation for regular rectilinear grids," *Opt. Express* **20**, 1247–1252 (2012) [doi:10.1364/OE.20.001247].
[64] R.W. Freund, "Conjugate gradient-type methods for linear systems with complex symmetrical coefficient matrices," *SIAM J. Sci. Stat. Comput.* **13**, 425–448 (1992) [doi:10.1137/0913023].
[65] H.A. van der Vorst and J.B.M. Melissen, "A Petrov–Galerkin type method for solving Ax=b, where A is symmetric complex," *IEEE Trans. Magn.* **26**, 706–708 (1990) [doi:10.1109/20.106415].
[66] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed., SIAM (1994).
[67] D.R. Fokkema, "Enhanced implementation of BiCGstab(l) for solving linear systems of equations," Preprint 976, Department of Mathematics, Utrecht University (1996).
[68] A. Bunse-Gerstner and R. Stover, "On a conjugate gradient-type method for solving complex symmetric linear systems," *Lin. Alg. Appl.* **287**, 105–123 (1999) [doi:10.1016/S0024-3795(98)10091-5].
[69] R.W. Freund and N.M. Nachtigal, "An implementation of the QMR method based on coupled 2-term recurrences," *SIAM J. Sci. Comput.* **15**, 313–337 (1994) [doi:10.1137/0915022].
[70] S.K. Sharma and D.J. Somerford, *Light scattering by optically soft particles: theory and applications*, Springer–Verlag, Berlin (2006).
[71] J.J. Goodman, B.T. Draine, and P.J. Flatau, "Application of fast-Fourier-transform techniques to the discrete-dipole approximation," *Opt. Lett.* **16**, 1198–1200 (1991) [doi:10.1364/OL.16.001198].

[72] C. Temperton, “Self-sorting mixed-radix fast Fourier transforms,” *J. Comput. Phys.* **52**, 1–23 (1983) [doi:10.1016/0021-9991(83)90013-X].
[73] M. Frigo and S.G. Johnson, “The design and implementation of FFTW3,” *IEEE Proc.* **93**, 216–231 (2005) [doi:10.1109/JPROC.2004.840301].
[74] J. Leinonen, D. Moisseev, and T. Nousiainen, “Linking snowflake microstructure to multi-frequency radar observations,” *J. Geophys. Res.: Atmos.* **118**, 3259–3270 (2013) [doi:10.1002/jgrd.50163].
[75] P.J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Academic Press, New York (1975).
[76] M.A. Yurkin, “Symmetry relations for the Mueller scattering matrix integrated over the azimuthal angle,” *J. Quant. Spectrosc. Radiat. Transfer* **accepted** [doi:10.1016/j.jqsrt.2012.11.023].
[77] B.T. Draine and P.J. Flatau, “User guide for the discrete dipole approximation code DDSCAT 6.1” (2006) [http://arxiv.org/abs/astro-ph/0409262v2].
[78] B.T. Draine and P.J. Flatau, “User guide for the discrete dipole approximation code DDSCAT 7.1” (2010) [http://arxiv.org/abs/1002.1505v1].

A Command Line Options

Most of the parameters are specified to ADDA in the command line. ADDA will automatically detect most inconsistencies in input parameters, and produce an error or warning if necessary (§C.1). If you notice that particular command line option causes abrupt termination or any erroneous behavior of ADDA, please communicate it to the developers. Command line option `-h` can be used to get both general help and detailed description of other command line options.

Below is the full list of command line options in alphabetical order. "`<…>`" denotes an argument, "`[…]`" denotes an optional part, and "`{…|…}`" denotes different possibilities. "Default:" and "Example:" fields specify the values of parameters excluding the command line option itself. Default value for all flags (those that can be used without arguments) is false, i.e. if the flag is not specified, the corresponding option is not enabled.

`-alldir_inp <filename>`
: Specifies a file with parameters of the grid of scattering angles for calculating integral scattering quantities (§10.3). Input format is described in §B.3.
Default: alldir_params.dat

`-anisotr`
: Specifies that refractive index is anisotropic (§6.3). Option `-m` then accepts 6 arguments per each domain. Can not be used with CLDR polarizability (§9.1) and all SO formulations (§9.1, §9.2, §9.3).

`-asym`
: Calculate the asymmetry vector (§10.3). Implies `-Csca` and `-vec`.

`-beam <type> [<args>]`
: Sets incident beam, either predefined or "`read`" from file (§8.2). All parameters of predefined beam types are floats.
Default: `plane`

`-chp_dir <dirname>`
: Sets directory for the checkpoint (both for saving and loading, §11.5).
Default: `chpoint`

`-chp_load`
: Restart a simulation from a checkpoint (§11.5).

`-chp_type {normal|regular|always}`
: Sets type of the checkpoint (§11.5). All types, except `always`, require `-chpoint`.
Default: `normal`

`-chpoint <time>`
: Specifies the time for checkpoints (§11.5) in format "`#d#h#m#s`". All fields are optional, numbers are integers, "`s`" can be omitted, the format is not case sensitive.
Examples: `12h30M`, `1D10s`, `3600`

`-Cpr`
: Calculate the total radiation force, expressed as cross section (§10.4).

`-Csca`
: Calculate scattering cross section (by integrating the scattered field, §10.3).

`-dir <dirname>`
: Sets directory for output files.
Default: constructed automatically, see §C.3.

`-dpl <arg>`

Sets parameter “dipoles per lambda” (§6.2), float.
Default: $10|m|$, where m is the maximum (by absolute value) refractive index specified by the `-m` option (§6.3).

`-eps <arg>`

Specifies the stopping criterion for the iterative solver (§11.1) by setting the relative norm of the residual ε to reach. `<arg>` is an exponent of base 10 (float), i.e. $\varepsilon = 10^{-\langle arg \rangle}$.
Default: `5` ($\varepsilon = 10^{-5}$)

`-eq_rad <arg>`

Sets volume-equivalent radius of the particle in μm (§6.2), float. If default wavelength is used, this option specifies the volume-equivalent size parameter. Can not be used together with `-size`. Size is defined by some shapes themselves, then this option can be used to override the internal specification and scale the shape.
Default: determined by the value of `–size` or by `-grid`, `-dpl`, and `–lambda`.

`-gpu <index>`

Specifies index of GPU that should be used (starting from 0). Relevant only for OpenCL version of ADDA, running on a system with several GPUs (§3.3).
Default: `0`

`-granul <vol_frac> <diam> [<dom_number>]`

Specifies that one particle domain should be randomly filled with spherical granules with specified diameter `<diam>` and volume fraction `<vol_frac>` (§6.5). Domain number to fill is given by the last optional argument.
Default `<dom_number>`: `1`

`-grid <nx> [<ny> <nz>]`

Sets dimensions of the computation grid (§6.2). Arguments should be even integers (otherwise corrected automatically by `ADDA`). In most cases `<ny>` and `<nz>` can be omitted (they are automatically determined by `<nx>` based on the proportions of the scatterer). This command line option is not relevant when particle geometry is read from a file (`-shape read`, §6.3). If `-jagged` option is used the grid dimension is effectively multiplied by the specified number (§6.3).
Default: `16` (if neither `-size` nor `-eq_rad` are specified) or defined by `–size` or `-eq_rad`, `-lambda`, and `-dpl`.

`-h [<opt> [<subopt>]]`

Shows help. If used without arguments, `ADDA` shows a list of all available command line options. If first argument is specified, help on specific command line option `<opt>` is shown (only the name of the option should be given without preceding dash). For some options (e.g. `-beam` or `-shape`) specific help on a particular suboption `<subopt>` may be shown.
Example: shape coated

`-init_field {auto|inc|read <filenameY> [<filenameX>]|wkb|zero}`

Sets prescription to calculate initial (starting) field for the iterative solver (§11.1). `auto` – automatically choose from `zero` and `inc` based on the lower residual value, `inc` – derived from the incident field, `wkb` – from WKB approximation, `read` – defined by separate files, which names are given as arguments, `zero` – a zero vector. Format of the files for option `read` is described in §B.7.
Default: `auto`

`-int {fcd|fcd_st|igt [<lim> [<prec>]]|igt_so|poi|so}`
Sets prescription to calculate interaction term (§9.2). The SO formulation can not be used with `-anisotr` (§6.3). `fcd` requires dpl to be larger than 2. Parameters of `igt` are: `<lim>` – maximum distance (in dipole sizes), for which integration is performed, (default: ∞); `<prec>` – minus decimal logarithm of relative error of the integration, i.e. $\varepsilon = 10^{-\texttt{<prec>}}$ (default: same as argument of `-eps` command line option).
Default: `poi`

`-iter {bcgs2|bicg|bicgstab|cgnr|csym|qmr|qmr2}`
Sets the iterative solver (§11.1).
Default: `qmr`

`-jagged <arg>`
Sets a size of a big dipole in units of small dipoles (§6.3), integer. It is used to improve the discretization of the particle without changing the shape.
Default: `1`

`-lambda <arg>`
Sets incident wavelength in μm (§6.2), float.
Default: 2π

`-m {<m1Re> <m1Im> […]|`
`<m1xxRe> <m1xxIm> <m1yyRe> <m1yyIm> <m1zzRe> <m1zzIm> […]}`
Sets refractive indices (§6.3), float. Each pair of arguments specifies real and imaginary parts of the refractive index of one of the domains. If `-anisotr` is specified, three refractive indices correspond to one domain (diagonal elements of the refractive index tensor in particle reference frame). None of the given refractive indices may be equal to 1.
Default: `1.5 0`

`-maxiter <arg>`
Sets the maximum number of iterations of the iterative solver, integer.
Default: very large, not realistic value (§11.1).

`-no_reduced_fft`
Do not use symmetry of the interaction matrix to reduce the storage space for the Fourier-transformed matrix (§11.2).

`-no_vol_cor`
Do not use “dpl (volume) correction” (§6.3). If this option is given, `ADDA` will try to match size of the dipole grid along the *x*-axis to that of the particle, either given by `-size` or calculated analytically from `-eq_rad`. Otherwise (by default) `ADDA` will try to match the volumes, using either `-eq_rad` or the value calculated analytically from `-size`.

`-ntheta <arg>`
Sets the number of intervals into which range of scattering angles [0°,180°] is equally divided (§10.1), integer. This is used for scattering angles in the *yz*-plane. If particle is not symmetric (§6.7) and orientation averaging (§7.2) is not used, the range is extended to 360 degrees (with the same length of elementary interval).
Default: from 90 to 720 depending on the size of the computational grid (§6.2).

`-opt {speed|mem}`
Sets whether `ADDA` should optimize itself for maximum speed or for minimum memory usage (§5).
Default: `speed`

`-orient {<alpha> <beta> <gamma>|avg [<filename>]}`

Either sets an orientation of the particle by three Euler angles α, β, γ (§7.1) or specifies that orientation averaging should be performed (§7.2). `<filename>` sets a file with parameters for orientation averaging (input format is described in §B.2).

Default orientation: `0 0 0`

Default <filename>: avg_params.dat

`-phi_integr <arg>`

Turns on and specifies the type of Mueller matrix integration over azimuthal angle φ (§10.1). `<arg>` is an integer from 1 to 31, each bit of which, from lowest to highest, indicates whether the integration should be performed with multipliers 1, cos(2φ), sin(2φ), cos(4φ), and sin(4φ) respectively.

Examples: `1` (one integration with no multipliers), `6` (two integration with cos(2φ) and sin(2φ) multipliers).

`-pol {cldr|cm|dgf|fcd|igt_so|lak|ldr [avgpol]|rrc|so}`

Type of polarizability prescription (§9.1). An optional flag `avgpol` can be added for LDR – it specifies that LDR polarizability should be averaged over incident polarizations. CLDR and SO can not be used with `-anisotr` (§6.3). `fcd` requires dpl to be larger than 2.

Default: `ldr` (without averaging).

`-prognosis`

Do not actually perform simulation (not even memory allocation) but only estimate the required RAM (§5). Implies `-test`.

`-prop <x> <y> <z>`

Sets propagation direction of incident radiation (§8.1), float. Normalization (to the unity vector) is performed automatically by `ADDA`.

Default: `0 0 1`

`-recalc_resid`

Recalculate residual at the end of iterative solver (§11.1).

`-save_geom [<filename>]`

Saves dipole configuration to a file `<filename>` (a path relative to the output directory, §6.3). Output format is determined by `–sg_format` and described in §C.11. This option can be used with `-prognosis`.

Default: `<type>.geom`

(`<type>` is shape name, to which `_gran` is added if `–granul` is used; file extension can differ depending on argument of `-sg_format`)

`-scat {dr|fin|igt_so|so}`

Sets prescription to calculate scattering quantities (§9.3). The SO formulation can not be used with `-anisotr` (§6.3).

Default: `dr`

`-scat_grid_inp <filename>`

Specifies a file with parameters of the grid of scattering angles for calculating the Mueller or the amplitude matrix. The Mueller matrix can be integrated over φ (§10.1). Input format is described in §B.4.

Default: scat_params.dat

`-scat_matr {muel|ampl|both|none}`
: Specifies which scattering matrices, the Mueller (§10.1) and/or the amplitude (§10.2), should be saved to file. Amplitude matrix is never integrated, i.e. the effect of the corresponding option does not combine with that of `-orient avg` or `-phi_integr`.
Default: `muel`

`-sg_format {text|text_ext|ddscat6|ddscat7}`
: Specifies format for saving geometry files (§6.3). First two are `ADDA` default formats for single- and multi-domain particles respectively. `DDSCAT` 6 and 7 formats are described in §C.11.
Default: `text`

`-shape <type> [<args>]`
: Sets shape of the particle, either predefined or "`read`" from file (§6.3). All the parameters of predefined shapes are floats described in detail in §6.4, except for file names.
Default: `sphere`

`-size <arg>`
: Sets the size of the computational grid along the *x*-axis in μm (§6.2), float. If default wavelength is used, this option specifies the "size parameter" of the computational grid.
Default: determined by the value of `-eq_rad` or by `-grid`, `-dpl`, and `-lambda`. Size is defined by some shapes themselves, then this option can be used to override the internal specification and scale the shape.

`-store_beam`
: Save incident electric fields to a file (§8). Output format is described in §C.9

`-store_dip_pol`
: Save dipole polarizations to a file (§10.5). Output format is described in §C.9.

`-store_force`
: Calculate the radiation force on each dipole (§10.4). Output format is described in §C.9. Implies `-Cpr`.

`-store_grans`
: Save granule coordinates to a file, if `-granul` option is used (§6.5). Output format is described in §C.12.

`-store_int_field`
: Save internal fields to a file (§10.5). Output format is described in §C.9.

`-store_scat_grid`
: Calculate Mueller matrix for a grid of scattering angles and save it to a file (§10.1). Output format is described in §C.5.

`-sym {auto|no|enf}`
: Automatically determine particle symmetries (`auto`), do not take them into account (`no`) or enforce them (`enf`, §6.7).
Default: `auto`

`-test`
: Begin name of the output directory with `test` instead of `run` (§C.3)

`-V`
: Show `ADDA` version, compiler used to build this executable, build options, and copyright information (§3.1).

`-vec`
: Calculate the not-normalized asymmetry vector (§10.3).

`-yz`
: Calculate the Mueller matrix in the *yz*-plane even if it is calculated for a scattering grid (§10.1). If the latter option is not enabled, scattering in the *yz*-plane is always calculated. Output format is described in §C.5.

B Input Files

All the input files should be located in the directory, in which ADDA is executed. Exceptions are the files specified in the command line – they may be located in a different directory. Some auxiliary files that may be required for ADDA execution (but which should not be modified by user) are described in §D. Comments can be used in most of the input files, they are defined as lines that start with # character. In most cases ADDA will detect incompatible format and terminate with an error message, consistency checks are also performed (producing error messages if necessary, §C.1). If you notice that particular input files cause abrupt termination or any erroneous behavior of ADDA, please communicate it to the developers. All files in this section are in ASCII format to ensure portability.

B.1 ExpCount

This is a very simple file, consisting of a single number ("run number"). In the beginning of its execution ADDA reads this number, increments it and saves back to the file. The read number appears in the name of the output directory (§C.3). The name of the `ExpCount` file can not be changed, however this file is not required. If it does not exist ADDA creates the `ExpCount` file and saves "1" in it. The purpose of the run number is two-fold: to provide convenience in sorting and analysis of output directories and guaranteeing that the name of the output directory is unique, so that ADDA will not overwrite any valuable data by its output. The first task (convenience) can be influenced by the user, who may change the number in `ExpCount` manually or delete this file to restart numbering.

The uniqueness of the directory name is a bit tricky, when several instances of ADDA run in parallel (each instance may be in sequential or parallel mode). It is possible albeit improbable that one instance of ADDA will read `ExpCount` between the other instance reads and updates the file. Then both instances will read the same run number. It may lead, though not necessarily, to the same name of output directories of these instances. On systems employing PBS or SGE (see §3.2) this problem is alleviated by adding a job id (which is unique) to the directory name (§C.3). Another option is used for all systems to guarantee the uniqueness of the run number – a file locking. Before reading `ExpCount` ADDA creates a file `ExpCount.lck` and removes it after updating `ExpCount`. All other ADDA instances wait until `ExpCount.lck` is removed. On POSIX systems `ExpCount.lck` is additionally locked to ensure robustness when working over the network file system, e.g. on parallel supercomputer.

Though highly improbable, permanent lock may occur under certain circumstances. That is when `ExpCount.lck` permanently exists (e.g. if ADDA is abruptly terminated between creating and removing this file). ADDA detects the permanent lock, using timeout, specified by parameters `LOCK_WAIT` (length of one wait cycles in seconds) and `MAX_LOCK_WAIT_CYCLES` (maximum number of wait cycles, after which timeout is declared), defined in the beginning of `io.c`. By default values 1 and 60 are used for these parameters respectively, i.e. if `ExpCount.lck` exists for one minute, ADDA exits with an error message. To solve the permanent lock problem remove `ExpCount.lck` manually.

The other potential problem of file locking is that its implementation is operating-system-dependent. ADDA should perform locking of `ExpCount` through lock file correctly on any POSIX-compliant or Windows operating system. However, it will produce an error during compilation if an operation system is not supported. If you experience this problem or unexplainable permanent locks occur, turn off file locking completely by recompiling with option `NOT_USE_LOCK`.[12] Some POSIX file systems do not support or do not allow (advanced) locking of files. ADDA should detect this, produce a warning, and continue without using this feature. However, the detection may take some time (up to 1 min). Therefore, if you

get such warning from ADDA you are advised to turn off advanced file locking by recompiling with option LOCKFILE_ONLY.[12]

B.2 avg_params.dat

This file specifies parameters for orientation averaging (§7.2). It consists of three sections, each specifying the parameters for each of the Euler angles: α, β, and γ (using "*zyz*-notation"). The first section looks like

```
alpha:
min=0
max=360
Jmin=2
Jmax=4
eps=0
equiv=true
periodic=true
```

specifying minimum and maximum angles, minimum and maximum number of refinements, required accuracy, whether the minimum and maximum angles are equivalent, and whether the function is periodic on the given interval (see §11.6 to understand the meaning of these parameters). Sections for other Euler angles contain the same parameters, but start with "`beta:`" and "`gamma:`" respectively. Specified `eps` and `Jmin` are relevant for β and γ, but they are not actually used for α, because values for integration over this angle are precalculated. If `min` and `max` are the same for some angle all other parameters are ignored for it and averaging over this angle is not performed. Values of β are spaced uniformly in values of $\cos\beta$ inside the specified interval. Currently the error of the integration of C_{ext} is used as a stopping criterion (it is compared to the specified `eps`). Thus the error of integration of other computed quantities may significantly differ from `eps`.

Particle symmetries may be considered by the user to decrease the ranges of Euler angles used for averaging. For example, if particle is axisymmetric (over the *z*-axis), γ is not relevant and user should set

```
gamma:
min=0
max=0
```

This will dramatically increase the speed of the orientation averaging. However even if particle is axisymmetric, its dipole representation has only 90° rotation symmetry. Hence, to be more precise, user should set `max=45;equiv=false` and decrease `Jmax` by 3. If a particle is symmetric with respect to the *xy*-plane, then the β range may be limited to [0°,90°], reducing corresponding `Jmax` by 1. Most of the particle symmetries can be employed, but that is user's responsibility to carefully account for them.

When $\beta = 0°$ or 180°, γ and α are not relevant independently but only as a combination $\alpha \pm \gamma$. If α is varied in a full range [0°,360°), ADDA simulates only one γ value for these specific values of β, saving a few evaluations of internal fields. Therefore, do *not* change the default (full) range of α, unless you have a good reason for it. It will cause all γ values to be honestly simulated. However, if you do have such a reason, please communicate it to the developers, motivating implementation of special optimizations for such "exotic" case. Moreover, a range of α is completely irrelevant (and ignored by ADDA) if only the cross sections need to be averaged (`-scat_matr none`), since the latter are calculated for unpolarized incident light (§10.3).

The example of the parameter file is included in the distribution (`input/avg_params.dat`), it is commented to facilitate its editing. The order of all the parameters is important, however comments can be inserted anywhere. A file with a different name can be used if specified in the command line (see §7.2).

B.3 alldir_params.dat

This file specifies parameters for averaging over the scattering angles for calculating integral scattering quantities (§10.3). It consists of two sections, specifying parameters for two scattering angles θ and φ. Each section is completely the same as in the `avg_params.dat` (§B.2), but starts with "`theta:`" or "`phi:`" respectively. A specified `eps` does not decrease the computational time, since all the integrated values are precalculated, but may decrease accuracy. If `min` and `max` are the same for some angle all other parameters are ignored for it and averaging over this angle is not performed. Values of θ are spaced uniformly in values of $\cos\theta$ inside the specified interval. All scattering angles are defined in the incident wave reference frame (§6.1).

Particle symmetries may be considered by the user to decrease the ranges of scattering angles used for averaging. For example, if a particle is axisymmetric (over the z-axis), range of φ can be decreased and user should set

```
phi:
min=0
max=90
equiv=false
```

and decrease `Jmax` by 2. It will significantly increase the speed of the averaging. Many of the particle symmetries can be employed, but that is user's responsibility to carefully account for them.

The example of the parameter file is included in the distribution (`input/alldir_params.dat`), it is commented to facilitate its editing. The order of all the parameters is important, however comments can be inserted anywhere. A file with a different name can be used if specified in the command line (see §10.3).

B.4 scat_params.dat

This file specifies parameters to calculate Mueller matrix on a grid of scattering angles (θ and φ), and possibly integrate result over the azimuthal angle φ (§10.1). It consists of one "global" section, two sections specifying the set of values for θ and φ, and one section for parameters of integration over φ. The first section looks like

```
global_type=grid
N=2
pairs=
0 0
30 90
…
```

First argument can be either `grid` or `pairs`. Grid is constructed as a Cartesian product of two sets of angles (described below). Pairs are specified by total number `N` and list of (θ,φ) values separated by space (each pair comes on a separate line). No comments can be inserted between "`pairs=`" and the end of the pairs list. `pairs` option is not compatible with integration over φ. The second section looks like

```
theta:
type=range
N=91
min=0
max=180
values=
0
10
…
```

`type` can be either `range` or `values`. Range is determined by `min` and `max` values, in which `N` points (including boundary points) are uniformly distributed. Values are specified by the total number `N` and a list (each value comes on a separate line). No comments can be inserted between "`values=`" and the end of the values list. A set of φ angles is defined by the similar section that starts with "`phi:`", however if integration over φ is enabled a range of φ is initialized based on the last section. This section is completely the same as in the `avg_params.dat` (§B.2), but starts with "`phi_integr:`". Specified `eps` and `Jmin` are not actually used, since all the integrated values are precalculated. All options that are not relevant for current configuration (e.g. number and list of pairs when grid of scattering angles is used) are ignored by `ADDA`, so one doesn't need to remove them from the file. All scattering angles are defined in the incident wave reference frame (§6.1).

Particle symmetries may be considered by the user to decrease e.g. the range of φ used for integration. For example, if particle is symmetric with respect to the *xz*-plane, then φ range may be limited to [0°,180°], reducing corresponding `Jmax` by 1, for integration of the block-diagonal Mueller matrix elements (S_{11}, S_{12}, S_{21}, S_{22}, S_{33}, S_{34}, S_{43}, S_{44}) without multipliers. A number of similar examples is discussed in [76]. Overall, many of the particle symmetries can be employed, but that is user's responsibility to carefully account for them.

The example of the parameter file is included in the distribution (`input/scat_params.dat`), it is commented to facilitate its editing. The order of all the parameters is important, however comments can be inserted anywhere, except in lists of angle values or pairs. A file with a different name can be used if specified in the command line (see §10.1).

B.5 Geometry files

This file specifies the shape of the scatterer (§6.3). Two default formats are supported: for single- and multi-domain particles. A single-domain particle is described as the following:

```
#comments
0 0 1
2 1 0
…
```

First several lines are comments, after that each line contains three integers separated by space. That is *x*, *y*, and *z* coordinates of a dipole (in units of the dipole size). This coordinates can be considered relative to any (integer) reference point; `ADDA` automatically places the minimum computational box (§6.2) around all dipoles and centers it as described in §6.1. The format for multi-domain particles is similar:

```
#comments
Nmat=2
4 4 0 1
5 4 0 1
…
```

The first uncommented line specifies number of domains (different materials) and the last integer in every line specifies the domain number (1,…,`Nmat`). However, given `Nmat` is largely ignored, since its more robust estimate is the maximum of the domain numbers among all dipoles. If these two are not equal, `ADDA` produces a warning and continues with the latter, ignoring the former.

`ADDA` also supports `DDSCAT` formats corresponding to its shape option `FRMFIL` (in version 6) and `FROM_FILE` (in version 7) and output of `calltarget` utility [77,78], which looks like (the 6th line is present only in `DDSCAT` 7):

```
comment line
2176 = NAT
```

```
1 0 0 = A_1 vector
0 1 0 = A_2 vector
1 1 1 = lattice spacings (d_x,d_y,d_z)/d
0 0 0 = coordinates (x0,y0,z0)/d of the zero dipole (IX=IY=IZ=0)
```
comment line, e.g. `JA  IX  IY  IZ ICOMP(x,y,z)`
dummy `4 4 0 1 1 1`
dummy `5 4 0 1 1 1`
…

However, ADDA's philosophy is slightly different from that of DDSCAT; in particular, shape file should describe only the dipole configuration, possibly multi-domain, but *neither* orientation nor size of the particle, which should be specified independently in the command line. Moreover, in case of anisotropic refractive index ADDA uses one domain number to distinguish all different refractive index tensors, even those which differ only by rotation of the tensor. Therefore, support of DDSCAT formats is limited in the following respects: (a) ADDA completely discards first 6 or 7 lines of the shape file, i.e. it always assumes default orientation and size of the particle, unless these are specified in the command line. (b) For each "dipole" row ADDA uses only dipole positions (2^{nd}–4^{th} numbers) and identifier of x-component of refractive index tensor (5^{th} number) as domain number, discarding the other two components.[71]

ADDA automatically determines the format of input dipole file, analyzing its several first lines. However, with increasing number of supported formats, probability of autodetection failure increases. If this happens, ADDA produces an error because of different format of further lines. The error message contains the detected format, which can be used to diagnose the problem. Also, for all shape formats the positions of any two dipoles must not coincide. Usually ADDA will detect such an error, but not in sparse mode.

B.6 Contour file

This file specifies contour in the ρz-plane defining the axisymmetric shape §6.4. It looks like:

```
#comments
0 0.5
0.2 -0.8
…
```

First several lines are comments, after that each line contains two floats defining ρ and z coordinates (in μm) of a contour point. At least three points should be specified. The contour is automatically closed by connecting the last and the first points. Linear interpolation is used between the specified points. The only requirement for the contour is that (by definition) ρ coordinates should be non-negative – ADDA will produce an error otherwise. For example, if a particle contains the interval of the z-axis, it is natural (although not necessary) to choose the first and the last points of the contour on the z-axis. Apart from this requirement, any set of points can be used; in particular, the contour may intersect itself or contain identical points.

B.7 Field files

This file determines the incident wave (§8.2) or initial field for the iterative solver (§11.1) on dipole-per-dipole basis. The format is the same as output format for file `IncBeam` or `IntField` (§C.9). In particular, the file should look like:

```
x y z |E|^2 Ex.r Ex.i … Ez.i
```

[71] It is possible to analyze all three components and construct bijective mapping of all different combinations of them into domain numbers. However, in general case this mapping may be not intuitively clear, which may lead to confusion in assigning (anisotropic) refractive indices to different domains. Hence, it is currently not done. Instead ADDA produces a warning if it encounters anisotropic dipole in the dipole file.

```
#comments
-0.2094395102 … -0.1844808236
…
0.2094395102 … -0.32940396
```

The first (header) line is ignored, as well as any number of following comment lines (starting with #). For each following data line first four numbers are ignored and next six numbers define the complex vector field in the following order: real and imaginary parts of *x*-component, the same for *y*-component, and then for *z*-component (all in particle reference frame). It should be stressed that dipole coordinates (first three numbers) are ignored, and correspondence between a field value and a dipole is based solely on the order of lines in the file. In other words, index of non-trivial data line exactly equals index of dipole. Thus, the safest way to produce correct input files is to use file `IncBeam` or `IntField`, produced by `-store_beam` (§8) or `-store_int_field` (§10.5) command line option respectively, as a template, changing only the values of the fields according to dipole coordinates. Finally, keep in mind that `ADDA` assumes unity amplitude of the incident field in its center (focus). So the incident beam specified in the file should satisfy this condition, otherwise most of the scattering quantities (§10) will be scaled accordingly.

C Output Files

ADDA outputs some information about its execution to `stdout` (§C.2), but most of information is saved in special files, which are created in a separate output directory (§C.3). All spreadsheet files use space as separator (both between column names and values). All files in this section are in ASCII format to ensure portability.

C.1 stderr, logerr

If any inconsistency is detected by ADDA during execution, it produces error and warnings. They are shown in `stderr` and duplicated in log files, as lines starting with

```
ERROR:
```

or

```
WARNING:
```

respectively. Most error messages are saved in the main log (§C.4), however in parallel mode each processor may produce specific error messages. The latter are saved in special files, named `logerr.n`, where `n` is a number of processor. In case of an error ADDA terminates execution, hence the errors that were detected before the output directory was created (§C.3) are not saved to file and appear only in `stderr`. Warnings indicate that probably the simulation goes not exactly the way intended by user, but ADDA continues execution.

C.2 stdout

ADDA's output to `stdout` is mainly designed to show the progress of the execution, when ADDA is run in the terminal session. More detailed information is saved to `log` (§C.4) and other output files, except for few information messages that appear only in `stdout`. The `stdout` from the sample calculation (§3.1) looks like

```
all data is saved in 'run000_sphere_g16_m1.5'
box dimensions: 16x16x16
lambda: 6.283185307   Dipoles/lambda: 15
Required relative residual norm: 1e-005
Total number of occupied dipoles: 2176
Memory usage for MatVec matrices: 1.3 MB
Calculating Green's function (Dmatrix)
Fourier transform of Dmatrix......
Initializing FFTW3
Total memory usage: 2.2 MB

here we go, calc Y

CoupleConstant:0.005259037197+1.843854148e-005i
x_0 = 0
RE_000 = 1.0000000000E+000
RE_001 = 8.4752662637E-001  +
…
RE_022 = 3.1681489046E-006  +
Cext   = 135.0449046
Qext   = 3.79114961
Cabs   = -1.937494758e-016
Qabs   = -5.439177817e-018
```

It provides name of the output directory for this calculation, basic information about the scattering problem, memory requirements, progress of the iterative solver, and results for extinction and absorption cross section. It may provide more information depending on the

particular simulation parameters. When precise timing (§12.2) is enabled, all results go to `stdout`.

In some cases reading long lines in `stdout` (as well as in `stderr`) can be "inconvenient", since such lines can be broken in the middle of the word to fit the terminal width. To improve this situation ADDA automatic wraps lines moving whole words to the new line. It tries to determine terminal width from the environmental variable `COLUMNS`,[72] otherwise uses the value, which is defined at compilation time by the parameter `DEF_TERM_WIDTH` in the file `const.h` (80 by default).

C.3 Output directory

Output directory is generated by ADDA automatically to store all output files. The name of the directory has the following format

```
<type><N>_<shtype>_g<nx>_m<m1Re>[_id<jobid>]
```

where `<type>` is either `run` or `test`. The latter is only used if `-prognosis` (§5) is enabled or

```
-test
```

command line option is specified. `<N>` is a run number that is read from ExpCount file (§B.1) and written in a format including at least three digits. `<shtype>` is shape name (§6.3). `<nx>` is dimension of the computational grid along the *x*-axis (§6.2), `<m1Re>` is real part of the first given refractive index (§6.3) written with up to 4 significant digits and decimal point replaced by "_". The last part of the name is added only if any of environmental variables `PBS_JOBID`, `JOB_ID`, or `SLURM_JOBID` is defined (they are defined by PBS, SGE, and SLURM[73] respectively, §3.2), then `<jobid>` is its value. For example, directory name may look like:

```
run000_sphere_g16_m1.5
test123_box_g40_m1.33_id123456
```

The first one corresponds to the sample simulation (§3.1), it is included in the distribution with 3 output files in it (`sample/run000_sphere_g16_m1.5`). To disable automatic naming of the output directory specify its name as an argument to the command line option

```
-dir <dirname>
```

C.4 log

This file contains most of the information that characterize the ADDA simulation. The `log` for the sample calculation (§3.1) is the following

```
Generated by ADDA v.1.2
The program was run on: YURKIN
command: 'D:\Maxim\Current\adda\win32\adda.exe '
lambda: 6.283185307
shape: sphere; diameter:6.734551818
box dimensions: 16x16x16
refractive index: 1.5+0i
Dipoles/lambda: 15
 (Volume correction used)
Required relative residual norm: 1e-005
Total number of occupied dipoles: 2176
Volume-equivalent size parameter: 3.367275909

---In laboratory reference frame:---
```

[72] This variable is defined on any POSIX system, but it is not necessarily exported, i.e. made available, to ADDA. On other systems it can be set manually if needed. It seems that using this variable is currently the most portable way.

[73] https://computing.llnl.gov/linux/slurm/

```
Incident beam: plane wave
Incident propagation vector: (0,0,1)
Incident polarization Y(par): (0,1,0)
Incident polarization X(per): (1,0,0)

Particle orientation: default

Calculating only Mueller scattering matrix
Polarization relation: 'Lattice Dispersion Relation'
Scattering quantities formulae: 'by Draine'
Interaction term prescription: 'as Point dipoles'
FFT algorithm: FFTW3
Iterative Method: QMR (complex symmetric)
Optimization is done for maximum speed
The FFT grid is: 32x32x32
Memory usage for MatVec matrices: 1.3 MB
Total memory usage: 2.2 MB

here we go, calc Y

CoupleConstant:0.005259037197+1.843854148e-005i
x_0 = 0
RE_000 = 1.0000000000E+000
RE_001 = 8.4752662637E-001  +   progress = 0.152473
…
RE_022 = 3.1681489046E-006  +   progress = 0.791154

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                Timing Results
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Total number of iterations: 22
Total planes of E field calculation (each 181 points): 2

Total wall time:     1
…
```

Most of the information is self-descriptive. The hostname (on the second line) is read from the environmental variable `HOST` (in Unix) or by function `GetComputerName` (in Windows). Command line that was used to call `ADDA` is duplicated. The scatterer (§6) is completely described, then the incident beam (§8), scatterer orientation (§7), and which scattering matrices are calculated (§10.2). The DDA formulation (§9) is described, then FFT algorithm (§11.2), iterative method (§11.1), and type of optimization (§5) are specified. Memory usage is given (both total and for FFT part, §5). "`calc Y`" denotes beginning of calculation for *y* incident polarization. "`CoupleConstant`" is dipole polarizability, "`x_0`" denotes which initial vector is used for iterative solver (§11.1). After each iteration the relative norm of the residual is shown together with its relative decrease compared to the previous iteration (`progress`). A sign in between is one of `+`, `-` or `-+` indicating respectively that the residual is the smallest of all the previous iterations, larger than the previous one, and smaller than the previous one but not the smallest of all the previous. `log` finishes with timing information (§12.1). This file may contain more information depending on the particular simulation parameters, the one that is described above is included in the distribution (`sample/run000_sphere_g16_m1.5/log`).

Package `misc/parse_log` contains Mathematica program to parse multiple log files, which may be useful for analyzing results of massive `ADDA` simulations.

C.5 mueller

This file contains results of the Mueller matrix at different scattering angles (§10.1). There are a number of output files, which name starts with `mueller`, but they all look very similar. When scattering is computed in one scattering plane or orientation averaging (§7.2) is performed the simplest file `mueller` is produced:

```
    theta s11 s12 s13 s14 s21 … s44
    0.00 1.4154797793E+002 0.0000000000E+000 0.0000000000E+000 \
-5.0959034824E-012 0.0000000000E+000 … 1.4154797793E+002
    1.00 1.4140075337E+002 -5.8903788393E-003 7.3212262090E-013 \
-2.0347873246E-012 -5.8903788393E-003 … 1.4140075290E+002
    …
    180.00 2.9143742276E+000 1.0348709736E-015 -1.0104462691E-017 \
3.8122857626E-012 1.0348709736E-015 … -2.9143742276E+000
```

where "\" denotes continuation of the line. All 16 Mueller matrix elements for each scattering angle are saved. Shown is the output of the sample calculation (§3.1) – `sample/run000_sphere_g16_m1.5/mueller`. If grid of scattering angles (any type) is calculated `mueller_scatgrid` is produced, which differs only by the additional column of azimuthal angle φ (and usually larger number of lines):

```
    theta phi s11 … s44
    0.00 0.00 1.4154797792E+002 … 1.4154797792E+002
    …
    180.00 360.00 2.9143742274E+000 … -2.9143742274E+000
```

This file can be produced by the command

```
    adda -store_scat_grid
```

with default `scat_params.dat` (§B.4). If integration over φ is enabled, up to 5 different files are produced depending on the parameters (different multipliers for integration, §10.1). They are called `mueller_integr`, `mueller_integr_c2`, `mueller_integr_s2`, `mueller_integr_c4`, and `mueller_integr_s4`. The format is the same as that of `mueller` but with addition of the column with error

```
    theta s11 … s44 RMSE(integr)
    0.00 1.4154797792E+002 … 1.4154797792E+002 2.150E-017
    …
    50.00 8.7607151503E+000 … 8.4048020875E+000 3.663E-010
    …
    180.00 2.9143742274E+000 … -2.9143742274E+000 3.601E-017
```

The shown error is relative root-mean-square error over all 16 elements of Mueller matrix,[74] integration error of each element is an estimation from the Romberg routine (§11.6). In this example the error is extremely low because of the spherical symmetry of the target and large number of integration points used (32). It is important to note that, strictly speaking, averaging but not integration is performed over φ. The above file can be produced by the command

```
    adda -phi_integr 1
```

with default `scat_params.dat` (§B.4). All scattering angles are specified with respect to the incident wave reference frame (§6.1).

C.6 ampl

This file contains results of the amplitude matrix at different scattering angles (§10.2). There are two output files, which name starts with `ampl`, but they all look very similar. When scattering is computed in one scattering plane, file `ampl` is produced:

```
    theta s1.r s1.i … s4.i
```

[74] In other words, RMS absolute error is divided by RMS value.

```
0.00 1.0746532054E+001 -5.1049022279E+000 … 2.4881279126E-013
1.00 1.0741905097E+001 -5.1007958857E+000 … 3.8850759557E-013
…
180.00 3.5097745561E-001 1.6706852073E+000 … -2.6713706097E-013
```

Four complex amplitude matrix elements for each scattering angle are saved (separately real and imaginary parts). Traditional way to number the amplitude matrix elements is by a single index, i.e. S_1–S_4 [42]. The above file can be produced by the command

```
adda –scat_matr ampl
```

If a grid of scattering angles (any type) is calculated, `ampl_scatgrid` is produced, which differs only by the additional column of azimuthal angle φ (and usually larger number of lines):

```
theta phi s1.r … s4.i
0.00 0.00 1.0746532054E+001 … 1.1550780581E-013
…
180.00 360.00 3.5097745561E-001 … -1.0410157053E-013
```

This file can be produced by the command

```
adda –scat_matr ampl -store_scat_grid
```

with default `scat_params.dat` (§B.4).

C.7 CrossSec

This file contains the results for integral scattering quantities (§10.3). If orientation averaging (§7.2) is performed the result is saved to `CrossSec` file, otherwise a separate file is used for each incident polarization: `CrossSec-X` and `CrossSec-Y`. Only one file (`CrossSec-Y`) is produced if symmetry of the particle is used to simulate only one incident polarization independently (§6.7). The format is self-explanative, for example the output of the sample simulation (§3.1) looks like (`sample/run000_sphere_g16_m1.5/Crosssec-Y`):

```
Cext   = 135.0449046
Qext   = 3.79114961
Cabs   = 0
Qabs   = 0
```

More results are shown in this file if additional (to default) scattering quantities are calculated.

C.8 RadForce

This file stores the results for radiation force on each dipole (§10.4). A separate file is used for each simulated incident polarization: `RadForce-X` and `RadForce-Y`. It looks like

```
x y z |F|^2 Fx Fy Fz
-0.2094395102 -1.047197551 -3.141592654 2.348525769e-005 \
-0.001450116615 0.004378387866 -0.001487326185
…
0.2094395102 1.047197551 3.141592654 0.001840340346 \
0.01038653728 -0.04118221174 -0.006040333213
```

where "\" denotes continuation of the line. This file describes the dependence of the force vector on the coordinates inside the particle (in μm). All values are specified in the particle reference frame (§6.1). The squared norm of the force vector is presented for convenience. The unit of force is 10^{-8} dyne when all lengths are measured in μm and unity amplitude (1 statV/cm) of incident field is assumed (see §10.4 for details). The above file can be produced by the command

```
adda -store_force
```

C.9 IntField, DipPol, and IncBeam

Internal fields, dipole polarizations (§10.5), and incident fields (§8) are saved to files `IntField-X`, `DipPol-X`, and `IncBeam-X` respectively. A separate file is used for each

simulated incident polarization, their names end with different suffixes (`X` or `Y`). For instance, `IntField-Y` looks like:

```
    x y z |E|^2 Ex.r Ex.i … Ez.i
    -0.2094395102 -1.047197551 -3.141592654 0.6988347019 \
-0.01668015393 0.006582289815 … -0.1844808236
    …
    0.2094395102 1.047197551 3.141592654 5.876037053 \
0.01112798497 -0.06772761653 … -0.32940396
```

where "\" denotes continuation of the line. This file describes the dependence of the electric field vector (normalized to the incident field) on the coordinates inside the particle (in μm), suffixes `r` and `i` denote real and imaginary parts respectively. All values are specified in the particle reference frame (§6.1). The squared norm of the electric field is presented for convenience, it is proportional to the energy density of the electric field and absorption power per unit volume for isotropic materials. The above file can be produced by the command

```
adda -store_int_field
```

Symbol used in the first row for column labels is different for different fields. It is "`E`" for the internal field, "`P`" for the dipole polarization, and "`Einc`" for the incident field. Electric fields are considered relative to the amplitude of the incident field, i.e. the values in the files are effectively dimensionless. The unit of the dipole polarizations is then the same as of volume (μm^3).

C.10 log_orient_avg and log_int

These files contain information about the 2D Romberg integration (§11.6), they are produced directly by the routine and hence have the same format. Their names are `log_orient_avg`, `log_int_Csca-X`, and `log_int_asym_x-X`, `log_int_asym_y-X`, `log_int_asym_z-X` for orientation averaging (§7.2) and calculation of scattering cross section and asymmetry vector (§10.3) respectively. The latter 4 files have different suffixes (`X` or `Y`) for different incident polarizations. For example, `log_int_Csca-Y` looks like:

```
                        PHI(rad)   cos(THETA)
    EPS                    0          0
    Refinement stages:
    Minimum                2          2
    Maximum                5          6
    lower boundary         0          -1
    upper boundary         6.28319    1
    equivalent min&max     true       false
    periodic               true       false


    Outer-Loop  Inner Loop
    Inner_qromb converged only to d=7.61895e-017 for cosine value #0
    Inner_qromb converged only to d=0 for cosine value #64
    init        64 integrand-values were used.
    Inner_qromb converged only to d=5.88e-010 for cosine value #32
    1      32 integrand-values were used.
    Inner_qromb converged only to d=4.77002e-010 for cosine value
#16
    Inner_qromb converged only to d=1.74306e-010 for cosine value
#48
    2      64 integrand-values were used.
    …
    6      1024 integrand-values were used.
    65 inner integrations did not converge.
```

```
The outer integration did not converge
The outer integration reached d=3.70825e-009
In total 2080 evaluations were used
```

The first part with parameters is self-descriptive. Then for every step of outer integration (over θ), convergence of all the inner integrations and total number of integrand evaluation are shown. At the end final statistics over the whole integration is shown. An integration (outer or one of the inner) is considered converged if its estimated relative error falls below `eps` (shown in the second line), which is given in a corresponding parameter file. For orientation averaging relative error of C_{ext} is shown. For this example no adaptation is used (`eps` = 0), hence none of the integrals converge, but the reached relative errors are informative. The range for angles (in the first column, φ or γ) is specified in *radians*. The above file can be produced by the command

```
adda -Csca
```

C.11 Geometry files

These files hold the information about the scatterer shape. They have exactly the same format as *input* geometry files (§B.5), the only difference is that `ADDA` automatically puts basic information in comments. By default, either single- or multi-domain format is used depending on the number of domains in the specified shape. For example, the command

```
adda -save_geom
```

produces the file `sphere.geom` that looks like:

```
#generated by ADDA v.1.2
#shape: 'sphere'
#box size: 16x16x16
7 5 0
…
8 10 15
```

It is important to note, that specified "box size" is the size of the computational domain. It always includes all the listed dipole positions, but in certain rare cases it may be larger than the minimal possible circumscribing box. In other words, for box size $n_x \times n_y \times n_z$ the first number is always between 0 and $n_x - 1$ and so on, but the limiting bounds are not necessarily reached. Moreover, the center of the computational domain can (rarely) be different from the center of the minimal circumscribing box around the particle. These issues should be carefully considered when processing the shape files.

`ADDA` can also save geometry in `DDSCAT` formats, if specified by the command line option `-sg_format` (§6.3), with the following limitations: (a) particle orientation and size is always saved as default, ignoring the actual values used in a specific simulation; (b) all three identifiers of components of the refractive index tensor are saved equal to the domain number, thus effectively transforming anisotropic particles to isotropic with (possibly) larger number of domains[75] (see §B.5 for philosophical reasons of these limitations). First number saved in each row is a dipole number, starting from 1.

C.12 granules

This file stores the coordinates of the granules produced by granule generator (§6.5). Currently the result looks like:

```
#generated by ADDA v.1.2
#granule diameter = 0.9980867612
-0.6693706848 1.833970089 -0.9894352851
```

[75] It is possible to remove this limitation, e.g. by mapping each domain number n into trio $3n-2$, $3n-1$, $3n$. However, it will cause a lot of unnecessary complications for isotropic particles.

…
```
-2.217014497 0.5198635377 -2.576990365
```
where each line specifies *x*, *y*, and *z* coordinates of a granule center in particle reference frame (§6.1). However the format should be considered in development and it may change in the future, e.g., to correspond the input shape format for cluster of spheres (when the latter will be developed). The above file, with different numbers due to random placement, can be produced by the command
```
adda -granul 0.1 1 -store_grans -prognosis
```

D Auxiliary Files

These files can be used by ADDA under certain circumstance, however they are not intended to be inspected or modified by user (except `chp.log`, §D.2). This is just general information to facilitate the understanding of how the ADDA actually works.

D.1 tables/

This is the directory that contains precalculated tables of some integrals, that are used by ADDA for IGT_{SO} and SO prescriptions for the interaction term (§9.2). The tables and their format may change in future. Currently it contains 10 files: `t1f.dat`, … , `t10f.dat`. These files are in text format to be completely portable, and occupy totally about 150 kB of disk space. They are included in the distribution (`input/tables/`).

D.2 Checkpoint files

These files are produced when ADDA saves a checkpoint (§11.5), and are used when ADDA loads one. By default the directory `chpoint` is used to store all files, however different directory name can be specified in a command line (§11.5). This directory contains one text file `chp.log`, which contains some information for the user. Currently it is only the name of the directory, where the output of the ADDA instance, which produced the checkpoint, was saved. Each processor (number *k*) produces a file named `chp.`*k* that contains the information, completely describing the state of the iterative solver on this processor, in binary format.[76]

[76] Hence it is not necessarily portable between different systems.