User Manual
for the
Discrete Dipole Approximation Code
"Amsterdam DDA"
(version 0.74.2)

Maxim A. Yurkin*† and Alfons G. Hoekstra*

**Faculty of Science, Section Computational Science, of the University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands, tel: +31-20-525-7530, fax: +31-20-525-7490*
†Institute of Chemical Kinetics and Combustion, Siberian Branch of the Russian Academy of Sciences, Institutskaya 3, Novosibirsk 630090 Russia, tel: +7-3832-333240, fax: +7-3832-342350

email: myurkin@science.uva.nl, alfons@science.uva.nl

last revised: April 5, 2006

Abstract

This manual describes using of the Amsterdam DDA (**ADDA**) code. **ADDA** simulates elastic light scattering from finite 3D objects of arbitrary shape and composition in vacuum or non-absorbing homogenous media. **ADDA** allows execution 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. The refractive index should not be large compared to 1, otherwise computational requirements increase drastically.

ADDA can be installed on its own, or linked with the FFTW 3 (Fastest Fourier Transform in the West) package. The latter is generally significantly faster than the built-in FFT, however needs separate installation of the package.

ADDA is written in C and is highly portable. It supports a variety of predefined particle geometries (ellipsoid, rectangular solids, coated spheres, red blood cells, etc.) and allows importing of an arbitrary particle geometry from a file. **ADDA** automatically calculates extinction and absorption cross sections and the complete Mueller matrix for one scattering plane. The particle may be rotated relative to the incident wave, or results may be orientation averaged.

This manual explains how to perform electromagnetic scattering calculations using **ADDA**. CPU and memory usage are discussed.

Contents

1 Introduction 4
2 Applicability of DDA 5
3 What's New 5
4 Obtaining the Source Code 6
5 Compiling and Linking 6
5.1 Compiling ADDA 6
5.2 Installing FFTW 3 8
6 Running ADDA 8
6.1 Sequential mode 8
6.2 Parallel mode 9
7 System Requirements 9
8 Defining a Scatterer 10
8.1 The computational grid 10
8.2 Construction of a dipole set 11
8.3 Predefined shapes 13
8.4 Partition over processors in parallel mode 14
8.5 Particle symmetries 14
9 Orientation of the Scatterer 15
9.1 Single orientation 15
9.2 Orientation averaging 15
10 Incident Beam 16
10.1 Propagation direction 16
10.2 Beam type 16
11 DDA Formulation 16
11.1 Polarization prescription 16
11.2 Interaction term 17
11.3 How to calculate scattering quantities 17
12 What Scattering Quantities Are Calculated 17
12.1 Mueller matrix 17
12.2 Integral scattering quantities 18
12.3 Radiation forces 19
12.4 Internal fields 19
13 Computational Issues 19
13.1 Iterative solver 19
13.2 Fast Fourier Transform 20
13.3 Parallel performance 21
13.4 Checkpoints 21
13.5 Romberg integration 22
14 Timing 23
14.1 Basic timing 23
14.2 Precise timing 23
15 Acknowledgements 23
References 25
A Command Line Options 27
B Input Files 31
B.1 ExpCount 31
B.2 avg_params.dat 32
B.3 alldir_params.dat 32

B.4 scat_params.dat 33
B.5 Geometry files 34
C Output Files 35
C.1 stderr, logerr 35
C.2 stdout 35
C.3 Output directory 36
C.4 log 36
C.5 mueller 38
C.6 CrossSec 38
C.7 VisFrp 39
C.8 IntField 39
C.9 log_orient_avg and log_int 39
C.10 Geometry files 40
D Auxiliary Files 41
D.1 tables/ 41
D.2 Checkpoint files 41

1 Introduction

ADDA is a C software package to calculate scattering and absorption of electromagnetic waves by particles of arbitrary geometry using the Discrete Dipole Approximation (DDA). In this approximation the volume of the scatterer is divided into small cubical subvolumes ("dipoles"), interaction of which is considered approximately based on the integral equation for the electric field [1]. Initially DDA (sometimes referred to as the "coupled dipole approximation") was proposed by Purcell and Pennypacker [2] by replacing the scatterer by a set of point dipoles (hence the name of the technique). DDA theory (considering point dipoles) was reviewed and developed further by Draine and coworkers [3-6]. Derivation of DDA based on the integral equation for the electric field was apparently first performed by Goedecke and O'Brien [7] and further developed by others (e.g. [8-11]). It is important to note that the final equations are essentially the same (small differences are discussed in §11). Derivations based on the integral equations give more mathematical insight into the approximation, while the model of point dipoles is physically more clear.

ADDA is a C implementation of the DDA developed by the authors. The development was conducted by Hoekstra and coworkers [12-15] for more than 10 years in University of Amsterdam. From the very beginning the code was intended to run on a multiprocessor system (parallelizing a *single* DDA simulation). Recently the code was significantly rewritten and improved by Yurkin. **ADDA** is intended to be a versatile tool, suitable for a wide variety of applications ranging from interstellar dust and atmospheric aerosols to biological particles; its applicability is limited only by available computer resources (§2). 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.[1]

The authors make this code 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.
- If you discover any errors in the code or documentation, please promptly communicate them to the authors (adda@science.uva.nl).
- You comply with the "copyleft" agreement (more formally, the GNU General Public License) 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 also strongly encourage you to send email to the authors identifying yourself as a user of **ADDA**; this will enable the authors to notify you of any bugs, corrections, or improvements in **ADDA**.

This manual assumes that you have already obtained the C source code for **ADDA** (see §4 for instructions). In §3 we describe the principal changes between **ADDA** and the previous releases. The succeeding sections contain instructions for:

- compiling and linking the code (§5);
- running a sample simulation (§6);
- defining a scatterer (§8) and its orientation (§9);
- specifying the type and propagation direction of the incident beam (§10);
- specifying the DDA formulation (§11);

[1] However, in some parts modularity was sacrificed for the sake of performance. E.g. iterative solvers (§13.1) are implemented not to perform any unnecessary operations (which usually happens when using standard libraries).

- specifying what scattering quantities should be calculated (§12);
- understanding the computational aspects (§13) and timing of the code (§14);
- understanding the command line options (§A) and formats of input (§B) and output (§C) files.

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 the dimensional values are scaled accordingly.

2 Applicability of DDA

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 dipole size d small compared to any structural length in the scatterer and the wavelength λ. A number of studies devoted to the accuracy of DDA results exist, e.g. [4-6, 11, 16-18] Here we only give a very brief overview.

The rule of thumb 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** (§8.1). 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 (§13.1) becomes slower. Additionally, accuracy of the simulation with default dipole size becomes worse, and smaller dipoles (hence larger number of them) must be used to improve it. Therefore, it is accepted that the refractive index should satisfy

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

However, higher m can be simulated accurately, but it requires very large computer resources. The examples of scattering problems that can be simulated on a modern desktop[2] computer and on massively parallel supercomputer are described in §7.

Extended review will be provided in a future version of the manual.

3 What's New

The most important changes between the current **ADDA** version (0.74.2) and the previous (0.74) are the following:

- Makefiles has been improved to allow simple usage of Intel compilers (§5.1).
- A section describing FFTW 3 installation has been added to the manual (§5.2).

Changes between versions 0.74 and 0.73 are:

- "`buggy`" beam type has been removed (obsolete, §10.2).
- Command line options `-nosym` and `-sym_enf` options has been combined into "`-sym {no|enf}`" (§8.5).
- Internal fields are now automatically collected into one file, when running in parallel mode (named `IntField-X` and `IntField-Y`, §12.4)
- MPI-specific command line options has been enabled (§6.2).
- Geometry filename given to the command line option `-save_geom` should now be specified relative to the output directory (§8.2).

[2] At time of writing, spring 2006.

- In parallel mode each processor (except root) produces a separate logfile, if any errors or warnings were detected (on this specific processor). These files are named `logerr.n` where *n* – is a number of processor (§C.1).
- Overall robustness of the code is improved (added many consistency checks).
- Help system added (new command line option `-h`, §6.1).

The full history of **ADDA** releases and differences can be found in `doc/history`.

4 Obtaining the Source Code

ADDA is a free software (§1). We are currently working to make a web page for **ADDA**, where it will be possible to download it. Currently, the latest version of **ADDA** can be obtained by sending a request to the authors: adda@science.uva.nl. The package contains the following:

`doc/` – documentation
- `copyleft` – GNU General Public License
- `history` – complete history of **ADDA** development
- `manual.pdf` – this manual in PDF format
- `README` – brief description of **ADDA**

`input/` – default input files
- `tables/` – 10 auxiliary files with tables of integrals (§D.1)
- `alldir_params.dat` – parameters for integral scattering quantities (§B.3)
- `avg_params.dat` – parameters for orientation averaging (§B.2)
- `scat_params.dat` – parameters for grid of scattering angles (§B.4)

`sample/` – sample output and other files
- `run000_sphere_g16m1_5/` – sample output directory (§C.3), contains `log` (§C.4), `mueller` (§C.5), and `CrossSec-Y` (§C.6)
- `batch` – sample PBS script for MPI system (§6.2)
- `stdout` – standard output of a sample simulation (§C.2)

`src/`
- `Makefile`, `make_seq`, `make_mpi` – makefiles (§5)
- `ADDAmain.c`, `CalculateE.c`, `calculator.c`, `cmplx.h`, `const.h`, `crosssec.c/h`, `comm.c/h`, `debug.c/h`, `fft.c`, `GenerateB.c`, `io.c/h`, `iterative.c`, `make_particle.c`, `matvec.c`, `memory.c/h`, `param.c/h` `prec_time.c/h`, `Romberg.c/h`, `timing.c`, `types.h`, `vars.c/h` – source and header files of **ADDA**
- `cfft99D.f` – source file for Temperton FFT (§13.2)

5 Compiling and Linking

5.1 Compiling ADDA

ADDA is written in C, but it contains one Fortran file (`cfft99D.f`) for built-in Fourier routines (they are only used if FFTW 3 is not installed) – see §13.2. On Unix systems **ADDA** can be easily compiled using provided `Makefile` – just type

```
make seq
```

or

```
make mpi
```

while positioned in `src/` directory, for the sequential or MPI version respectively. Default compilers (`gcc` and `g77` for sequential, and `mpicc` and `mpif77` for MPI versions respectively) will be used together with maximum optimization flags. You may change the compilers (and compilation flags), by modifying the variables `CC`, `CF` (and `CFLAGS`, `FFLAGS`) in the files `make_seq` and `make_mpi`.

In order to compile **ADDA** with FFTW 3 support you need first to install the FFTW 3 package (§5.2). In order to compile a parallel version (§13.3) MPI should be installed on your system. You should probably consult someone familiar with the particular MPI package. **ADDA**'s usage of MPI is based on the MPI 1.1 standard,[3] and it should work with any implementation that is compliant with this or higher versions of the standard. At the University of Amsterdam we use MPICH,[4] a publicly available implementation of MPI.

There are four options that may be changed in the `Makefile` uncommenting the corresponding lines:

"`CFLAGS += -DDEBUG`" – debugging. Turns on additional information messages during the code execution.

"`CFLAGS += -DFFT_TEMPERTON`" – use FFT by C. Temperton (§13.2). Use it if you have problems installing the FFTW 3 package.

"`CFLAGS += -DPRECISE_TIMING`" – enable precise timing routines, which give extensive timing of all the computation parts of **ADDA**, useful for debugging or optimization studies (§14.2).

"`CFLAGS += -DNOT_USE_LOCK`" – do not use file lock for `ExpCount`, enable this flag if you experience permanent locks (§B.1).

It is possible to compile **ADDA** using Intel compiler.[5] Supplied makefiles contain all the necessary flags for optimal performance, one only need to change the value of flag `COMPILER` in `Makefile` to "`intel`". We have tested the compilation with `gcc` version 3.3.5 and higher and `icc` version 8.1 and higher. We have found about 20% overall improvement of speed when using `icc` compared to `gcc`, and more than twice increase in speed of particular **ADDA** parts, e.g. calculation of the scattered field. Therefore, we recommend using Intel compiler wherever possible. Note that it is freely available for non-commercial use on Linux systems. However, we have noticed that combination of Intel compilers and MPICH can be buggy. That is, of course, dependent on a particular software and hardware used.

All compilation warnings are suppressed in public releases of **ADDA**. However, you may turn them on, by commenting a line

```
RELEASE =
```

in `Makefile`. If you do so, please communicate obtained warnings to the authors.

Compilation on non-Unix systems is also possible, however it should be done manually – compile all the source files (with maximum possible optimizations) and link them in executable `adda`.

So far as we know there are only two operating-system-dependent aspects of **ADDA**: precise timing (§14.2), and file locking (§B.1). Both are optional and can be turned off by compilation flags. However these features should be functional for any 32bit Windows or POSIX-compliant (Unix) operating system.

[3] http://www.mpi-forum.org
[4] http://www.mcs.anl.gov/mpi/mpich/
[5] http://www.intel.com/cd/software/products/asmo-na/eng/compilers/index.htm

5.2 Installing FFTW 3

The installation of FFTW 3 package[6] on any Unix system is straightforward and therefore is *highly recommended*, since it greatly improves the performance of **ADDA**. The easiest is to install FFTW 3 for the entire system, using `root` account.[7] However, it also can be installed under any user account as follows:

- Download the latest version of FFTW 3 from http://www.fftw.org/download.html
- Unpack it, cd into its directory, and type
  ```
  ./configure --prefix=$HOME [--enable-sse2|--enable-k7]
  ```
  where "`--enable`" options are specialized for modern Intel or AMD processors respectively. Then type
  ```
  make
  make install
  ```
- Modify few environmental variables, affecting the compiler. For example, if you are using `bash`, add the following two lines to any login script:
  ```
  export C_INCLUDE_PATH=$HOME/include
  export LIBRARY_PATH=$HOME/lib
  ```

Installation of FFTW 3 on non-Unix systems is slightly more complicated. It is described in http://www.fftw.org/install/windows.html.

6 Running ADDA

6.1 Sequential mode

The simplest way to run **ADDA** is to type

```
adda
```[8]

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 (§12, §C). The output directory and terminal output (`stdout`) should look like examples that are included in the distribution: `sample/run000_sphere_g16m1_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. 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 distribution (which contains examples of all input files) to the directory where **ADDA** is executed. All the output produced by **ADDA** is described in §C.

[6] http://www.fftw.org

[7] Details are in http://www.fftw.org/fftw3_doc/Installation-on-Unix.html

[8] 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`". This applies to all examples of command lines in this manual.

6.2 Parallel mode

ADDA uses MPI for parallel execution. On different systems MPI is used differently, you should consult someone familiar with MPI usage on your system.

At the University of Amsterdam we employ the Dutch national compute cluster LISA.[9] There, as on many other parallel computers, PBS (Portable Batch System)[10] is used to schedule jobs. Here we describe briefly how to use PBS to start **ADDA** job. One should first write a shell script such as the following file `batch`:

```
#PBS -N ADDA
#PBS -l nodes=2:ppn=2
#PBS -l walltime=0:05:00
#PBS -j oe
#PBS -m bea
#PBS -M myurkin@science.uva.nl
#PBS -S /bin/bash
#PBS -V
cd $PBS_O_WORKDIR
module load gnu-mpich-ib
mpiexec ./adda
```

The line beginning with "`#PBS -N`" specifies the name of the job. The lines beginning with "`#PBS -l`" specify the required recources: number of nodes, number of processors per node, and walltime. "`#PBS -j oe`" specifies that the output from `stdout` and `stderr` should be merged to one output file. "`#PBS -m bea`" specifies that PBS should send email when the job begins (b), and when it ends (e) or aborts (a). Emails are sent to the address specified in the line beginning with "`#PBS -M`". "`#PBS -S`" specifies a shell to execute the script. "`#PBS -V`" specifies that all the environmental variables should be exported to the job. The execution part consists of three commands: cd into working directory, load appropriate module, and start **ADDA**. Any command line options may be specified to the right of `adda`, MPI command line options (specific for a particular MPI implementation) may also be given here.[11] The extended version of the script file with comments is included in the distribution (`sample/batch`).

On our system the `stdout` and `stderr` of the parallel **ADDA** are redirected to the file named like `ADDA.o123456`, where the number is PBS job id. The same number appears in the directory name (§C.3).

7 System Requirements

Computational requirements of DDA primarily depend on the size of computational grid, which in turn depends on the size parameter x and refractive index m of the scatterer (§8.1). The memory requirements of **ADDA** depend both on the total number of dipoles in a computational box (N) and number of real (non-void) dipoles (N_{real}). For single-processor mode memory requirements (in bytes) are approximately

$$mem = 288N + 271N_{real} \quad (+144N_{real}), \tag{3}$$

where additional memory (in brackets) is required for the QMR and Bi-CGSTAB iterative solvers (§13.1). In multiprocessor mode the total memory requirements are described by Eq. (3), except that part proportional to N may be slightly higher (see §13.2 for details). The memory requirements of each processor depends on the partition of the computational grid

[9] http://www.sara.nl/userinfo/lisa/description/index.html
[10] http://www.openpbs.org/
[11] To view the list of these options type '`man mpi_init`'. The way they are handled is dependent on a particular MPI implementation. We tested it for MPICH 1.2.5, but it should also work for others.

over the processors that is generally not uniform (see §8.4). 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 (compared to single precision), but it helps when convergence of the iterative solver is very slow and machine precision becomes relevant (that is the case for large simulations) or when very accurate results are desired, as in [19]. A command line option

```
-prognose
```

can be used to estimate the memory requirements without actually performing the allocation of memory and simulation.[12] It also implies `-test` option (§C.3).

Simulation time (see §13 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 scaling as $O(N\ln N)$ (see §13.2). 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 (§12.1, §12.2). Employing multiple processors brings the simulation time down almost proportional to the number of processors (see §13.3). To facilitate very long simulations checkpoints can be used to break a single simulation into smaller parts (§13.4).

For example, on a modern desktop computer (P4-3.2 GHz, 2 Gb RAM) it is possible to simulate light scattering by particles[13] up to $x = 35$ and 23 for $m = 1.313$ and 2.0 respectively (simulation times are 1 and 17 days respectively). Using 48 nodes of LISA (each dual P4-3.4 GHz with 2Gb RAM), we were able to simulate light scattering by a homogenous sphere with $x = 130$ and $m = 1.2$ in 12 hours [20].

8 Defining a Scatterer

8.1 The computational grid

ADDA embeds any scatterer in a rectangular computational box, which is divided into identical cubes.[14] Each cube is called a "dipole", its size should be much less 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, §8.2) of each dipole can be set independently. There are a few parameters describing the simulation 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, and incident wavelength λ. However not all of them are independent. **ADDA** allows one to specify all three grid dimensions n_x, n_y, n_z as corresponding arguments to the command line option[15]

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

however in most cases `<ny>` and `<nz>` can be omitted. Then n_y, n_z are automatically determined by n_x based on the proportions of the scatterer (§8.3). If particle geometry is read

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

[13] Shown values are for spheres, for other shapes they may vary. Only one incident polarization was calculated, execution time for non-symmetric shapes (§8.5) will be doubled.

[14] The equally spaced cubical grid is required for the FFT-based method (§13.2) that is used to accelerate matrix-vector products in iterative solution of the DDA linear system (§13.1). Otherwise DDA computational requirements are practically unbearable.

[15] Because of the internal structure of the **ADDA** all the dimensions are limited to be even. If odd grid dimension is specified by any input method, it is automatically incremented.

from a file (§8.2) all the grid dimensions are initialized automatically.[16] If the `-jagged` option is used the grid dimension is effectively multiplied by the specified number (§8.2).

ADDA allows also to specify size parameter of the entire grid and size parameter of the dipole. The first one is specified by two command line options:

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

which specify λ and D_x (in μm) respectively. By default $\lambda = 2\pi$ μm, then `-size` specifies the dimensionless size parameter of the grid kD_x (k is free space wave vector). The size parameter of the dipole is specified by the parameter "dipoles per lambda" (dpl)

$$\mathrm{dpl} = \frac{\lambda}{d} = \frac{2\pi}{kd}, \tag{4}$$

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 not accept all three parameters (dpl, n_x, and kD_x) since they depend on each other

$$kD_x \cdot \mathrm{dpl} = 2\pi \cdot n_x . \tag{5}$$

If any two of them is given on the command line (n_x is also defined if particle geometry is read from file) the third is automatically determined from the Eq.(5). If the latter is n_x, dpl is slightly increased (if needed) so that n_x exactly equals an even integer. If less than two parameters are defined dpl or/and grid dimension are set by default.[17] The default for dpl is $10|m|$ (cf. Eq.(1)), where m is the first refractive index specified by the "`-m`" option (or the default one, §8.2). The default for n_x is 16 (possibly multiplied by `-jagged` value). Hence, if only `-size` is specified, **ADDA** will automatically discretize the particle, using the default dpl.

8.2 Construction of a dipole set

After defining the computational grid (§8.1) 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 can be done automatically for a number of predefined shapes or in a very flexible way – specifying a scatterer geometry in a separate input file. Predefined shapes are described in detail in §8.3. The dipole is assigned to the scatterer if its center belongs to it (see Fig. 1 for an example). When the scatterer consists of several domains, e.g. coated sphere, the same rule applies to each domain. **ADDA** has an option to slightly correct the dipole size (or equivalently dpl) to ensure that the volume of the dipole representation of the particle is exactly correct (Fig. 2). This is believed to increase the

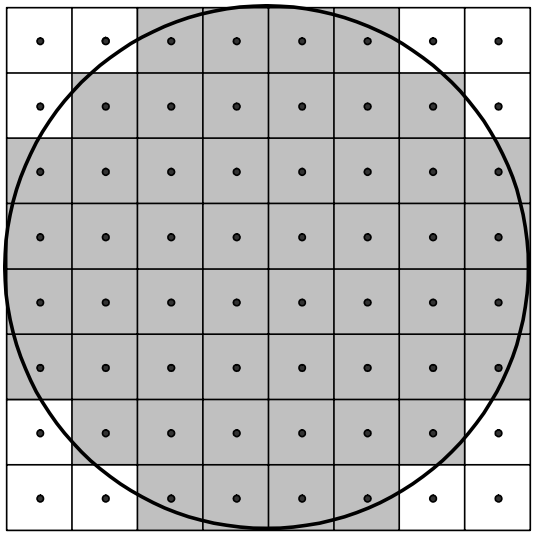

Fig. 1. Example of dipole assignment for a sphere (2D projection). Assigned dipoles are gray and void dipoles are white.

[16] 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.

[17] 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.

accuracy of DDA, especially for small scatterers [5]. However, it introduces a small inconvenience that the size of the computational grid is not exactly equal to the size of the particle. "dpl correction" is performed automatically by **ADDA** for most of the predefined shapes (see §8.3 for details), but can be turned off by the command line option

```
-no_vol_cor
```

To read a 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). Format of the input file is described in §B.5. Dimensions of the computational grid are then initialized automatically.

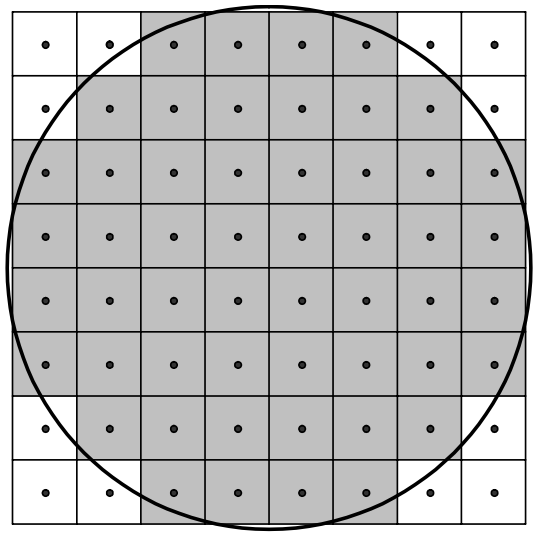

Fig. 2. Same as Fig. 1 but after the "dpl correction".

Sometimes it is useful to describe a particle geometry in a coarse way by bigger dipoles (cubes), but then use smaller dipoles for the simulation itself.[18] **ADDA** enables it by the command line option

```
-jagged <arg>
```

which specifies a multiplier J. For construction of the dipole set big cubes ($J\times J\times J$ dipoles) are used (Fig. 3) – center of big cubes are tested for belonging to a particle's domain. All grid 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.

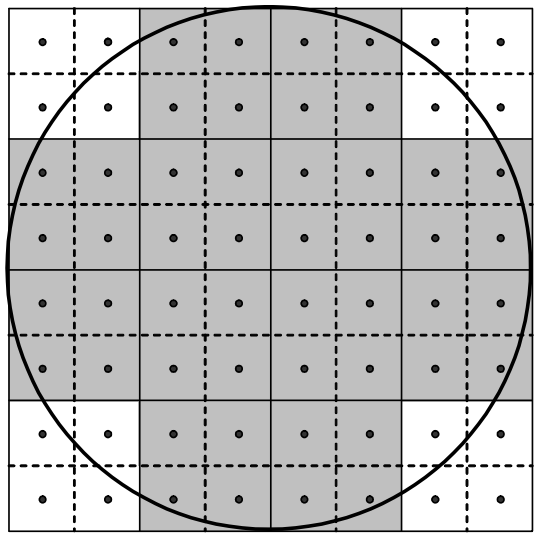

Fig. 3. Same as Fig. 1 but with "`-jagged`" option enabled ($J = 2$). The total grid dimension is the same.

The last parameter to completely specify a scatterer is its refractive index. Currently **ADDA** supports scatterers with isotropic refractive index, moreover this refractive index should change discretely.[19] Refractive indices are given on the command line

```
-m <m1Re> <m1Im> [<m2Re> <m2Im>…]
```

Each pair of arguments specifies the real and imaginary part of the refractive index of one of the domains. 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 10. The number of the domain in the geometry file (§B.5) exactly corresponds to the number of the refractive index. This correspondence for the predefined shapes in described in §8.3. If no refractive index is specified, it is set to 1.5, but this default option works only for one-domain scatterers.

ADDA is able to save the constructed dipole set to a file if the command line option

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

[18] 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) [19].

[19] We are working to remove these limitations.

is specified. `<filename>` is an optional argument (it is a path relative to the output directory, §C.3). If it is not specified, **ADDA** names the output file `<type>.geom` (`<type>` is a first argument to the `-shape` command line option, see above and §8.3). The format of this file is compatible with the input one (see §C.10 for details). The values of refractive indices are not saved (only domain numbers). This option can be combined with `-prognose`, then no DDA simulation is performed but the geometry file is generated.

8.3 Predefined shapes

Predefined shapes are initialized by the command line option

```
-shape <name> <args>
```

where `<name>` is a name of the predefined shape. The size of the scatterer is determined by the size of the computational grid (D_x, §8.1); `<args>` specify different dimensionless aspect ratios or other proportions of the particle shape.

In the following we describe all the supported predefined shapes, all the reference information is summarized in Table 1. "`box`" is a homogenous cube. "`coated`" is a sphere with a spherical inclusion; outer sphere has a diameter d (first domain[20]). The included sphere has a diameter d_{in} (optional position of the center: x, y, z). "`cylinder`" is a homogenous cylinder with height (length) h and diameter d (its axis of symmetry coincides with the z-axis). "`ellipsoid`" is a homogenous general ellipsoid with semi-axes x, y, z. "`line`" is a line along the x-axis with the width of one dipole. "`rbc`" is a Red Blood Cell, an axisymmetric (over z-axis) 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.[21] "`sphere`" is a homogenous sphere (used by default). "`spherebox`" – a sphere (diameter d_{sph}) in a cube (size D_x, first domain). For multi-domain shapes "dpl correction" is performed based on the volume of an outer domain.

Table 1. Brief description of arguments, symmetries (§8.5) and "dpl correction" for predefined shapes. Shapes and their arguments are described in the text. "±" means that it depends on the arguments.

`<name>`	`<args>`	symmetry		dpl correction
		reflection over xz-plane	rotation by 90° over z-axis	
`box`	–	+	±	–*
`coated`	d_{in}/d, [x/d, y/d, z/d]	±	±	+
`cylinder`	h/d	+	+	+
`ellipsoid`	y/x, z/x	+	±	+
`line`	–	–	–	–*
`rbc`	h/d, b/d, c/d	+	+	–†
`sphere`	–	+	+	+
`spherebox`	d_{sph}/D_x	+	±	–*

*dpl correction is not needed (volume is automatically correct);
†volume of the particle can not be easily determined based on the input parameters.

We are currently working to greatly increase the number of predefined shapes, at least to include all the shapes that are available in DDSCAT 6.1 [23].

[20] The order of domains is important to assign refractive indices specified in the command line (§8.2).
[21] Based on [21]. It is similar to the RBC shape used in [22].

8.4 Partition over processors in parallel mode

To understand the parallel performance of **ADDA** it is important to realize how a scattering problem is partitioned over the processors. It is done in a simple and robust, but not necessarily the most efficient way. Both the computational grid and the scatterer are partitioned in slices parallel to *xy*-plane (in another words, partition is performed over the *z*-axis); each processor contains several of these slices. For the FFT-based task (§13.2) – the matrix-vector product that takes most of the time of iterative solution (§13.1) – the whole grid is partitioned[22]. The partition over the *z*-axis is optimal for this task if n_z divides the number of processors (at least approximately).

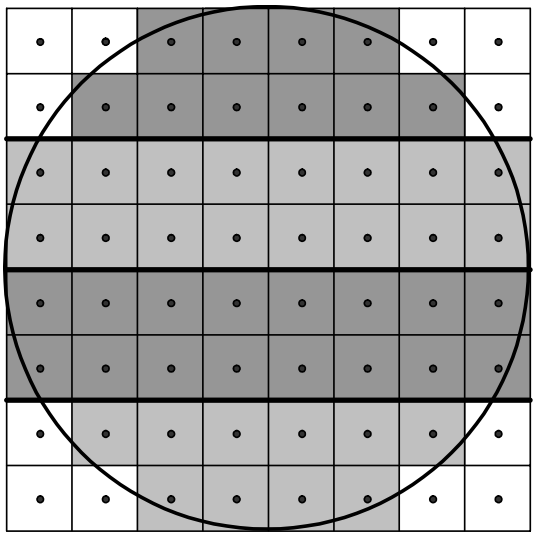

Fig. 4. Same as Fig. 1 but partitioned over 4 processors (shown in different shades of gray).

The partition of the scatterer itself also benefits from the latter condition, however it is still not optimal for most of the geometries,[23] 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 (§12.1, §12.2), 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).[24]

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 (§13.2).

8.5 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 (§10). If the particle is symmetric with respect to reflection over the *xz*-plane, only half of the scattering *yz*-plane is calculated (scattering angle from 0° to 180°, §12.1). If the particle is symmetric with respect to rotation by 90° over the *z*-axis, the Mueller matrix in the *yz*-plane (§12.1) 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 *xz*-plane (in negative direction of *x*-axis). The symmetries are automatically determined for all the predefined shapes (§8.3) when default incident beam type and direction (§10) are used. If beam type and/or direction is not default **ADDA** cancels all symmetries. We are working to extend automatic recognition of symmetries to these cases.

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

```
-sym <type>
```

[22] More exactly, the grid is doubled in each dimension and then partitioned (see also §13.2).

[23] Exceptions are cubes and any other particles, for which area of any cross section perpendicular to z-axis is constant.

[24] That is additionally to the communication overhead that always exists (§13.3).

where `<type>` is either `no` or `enf` 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 not-default incident beam is used, which does not spoil the symmetry of the problem (e.g. plane wave 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,[25] otherwise the accuracy of the result will generally be slightly worse than that when symmetry is not used.

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).

9 Orientation of the Scatterer

9.1 Single orientation

Any particle orientation with respect to the laboratory reference frame can be specified by three Euler angles (α, β, γ). 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, which naturally corresponds to particle geometry and symmetries, to minimize the size of computational grid (§8.1), especially for elongated or oblate particles. Therefore rotation of the particle is equivalently represented as an inverse rotation of the incident wave propagation direction and polarization (§10.1) and scattering plane (angles). The information about the orientation of a scatterer is saved to the `log` (§C.4).

9.2 Orientation averaging

Orientation averaging is performed in **ADDA** 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 computation time for each scattering plane is comparably small. Averaging over the other two Euler angles is done by independent DDA simulations (defining the orientation of the scatterer as described in §9.1). The averaging itself is performed using Romberg integration (§13.5), parameters of 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 averaging. Integration points for β are spaced uniformly in values of cosβ. Currently only the Mueller matrix in one scattering plane (§12.1), C_{ext}, and C_{abs} (§12.2) are calculated when doing orientation averaging. We are currently working to include the asymmetry vector **g** (§12.2) in this list.

[25] For example, a sphere is symmetric for any incident direction, but the corresponding dipole set (Fig. 1) is only symmetric for incidence along a coordinate axis.

10 Incident Beam

10.1 Propagation direction

The direction of propagation of 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 (0,0,1) is used. Two incident polarizations are used by default: along the *x* and *y* axis.[26] Those are perpendicular ($\perp$) and parallel ($\parallel$) polarizations [24] 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 (0,0,1) is transformed into the specified propagation vector. All the scattering angles are specified with respect to the reference frame based on the *new* propagation vector (*z*) and two *new* incident polarizations (*x*,*y*).[27]

The option `-prop` is cumulative with rotation of the particle (§9.1) 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 (§8.5).

10.2 Beam type

ADDA supports not only the ideal plane wave incident radiation, but also several types of finite size beams. The choice is determined by the command line option

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

where `<type>` is one of the `plane`, `barton1`, `barton3`, `barton5`, `davis1`, `davis3`, or `lminus`, which corresponds to the different beam types with the same names (see [25-27] for details). Four other arguments must be specified for all beam types except the default plane wave. These are the width and *x*, *y*, *z* coordinates of the center of the beam respectively (all in μm). All the finite beam types currently work properly only for default incidence and orientation of the particle. We are working to fix this issue and provide a detailed description of all the beam types.

11 DDA Formulation

Since its introduction by Purcell and Pennypacker [2] DDA has been constantly developing, therefore different formulations of DDA differ in some respects. These are polarization prescription (self-term, §11.1), interaction term (§11.2) and formulae to calculate scattering quantities (§11.3). The classical reviews of DDA [5,6] cover most of the differences, except the modern improvements, which are still in the research phase – they have not yet been implemented in any publicly available code. **ADDA** incorporates some new theoretical improvements that we are developing ourselves (see the following subsections). They are in the early research phase, but you can use them at your own risk.

11.1 Polarization prescription

The basic polarization prescription is Clausius-Mossotti [2]. Improvements are the Radiative Reaction correction (RR) [3], the Digitized Green's Function (DGF) [7], the formulation by Lakhtakia (LAK) [28,29], the a_1-term method [30,31], the Lattice Dispersion Relation

[26] We are currently working to include an option to specify arbitrary (possibly complex) incident polarization. It can be useful if only one particular polarization need to be simulated.

[27] For example, the default scattering plane (§12.1) – *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.

(LDR) [4], the formulation by Peltoniemi (PEL) [32], and the Corrected LDR (CLDR) [33]. LDR is most widely used, advertised by the code DDSCAT [23].

ADDA can use 4 of the above: CM, RR, LDR, CLDR and one new Second Order (SO) polarization prescription, which we are developing ourselves. The choice is performed by command line option

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

where `<type>` is one of the `cm`, `rrc`, `ldr`, `cldr`, `so`. `<arg>` is optional flag that can be only `avgpol` and only for LDR – it specifies that LDR polarizability should be averaged over incident polarizations [4]. Default is LDR without averaging.

11.2 Interaction term

The most commonly used way to calculate the interaction term in DDA is to consider an interaction of two point dipoles [2,5]. Improvements based on the integral equations exist: Filtered Coupled Dipoles (FCD) [34] and Integration of Green's Tensor (IT) [11], however they are not widely used yet. **ADDA** can use the standard one (point dipoles), or the new Second Order (SO) formulation, which we are developing. The latter can be considered an extension of IT. The choice is performed by the command line option

```
-int <type>
```

where `<type>` is either `poi` or `so`. For SO formulation tables of precalculated integrals are used, they are automatically read from files in `tables/` (§D.1).

11.3 How to calculate scattering quantities

The simplest way to calculate the scattering quantities (scattering amplitude, cross sections) in DDA is to consider a radiation emitted by a set of point dipoles [2]. Draine [3] proposed a correction of the formula for C_{abs} so that it will give zero absorption for non-absorbing particles (when RR is employed, or any other polarization prescription that incorporates RR, §11.1). The formulation by Draine is the most widely used nowadays (e.g. in DDSCAT [23]), however some minor corrections are possible [11]. **ADDA** can use the standard one (by Draine), or the new Second Order (SO) formulation, which we are developing. The choice is performed by command line option

```
-scat <type>
```

where `<type>` is either `dr` or `so`.

12 What Scattering Quantities Are Calculated

All the scattering angles (polar θ and azimuthal φ) are specified with respect to the incident wave (see §9.1 and §10.1 for details).

12.1 Mueller matrix

ADDA calculates a complete Mueller matrix (see e.g. [24]) for a set of scattering angles. By default scattering in the *yz*-plane is calculated. The range of [0°,180°] is equally divided into N_θ intervals. If the particle is not symmetric (§8.5) and orientation averaging (§9.2) is not used the range is extended to 360 degrees. Totally $N_\theta + 1$ or $2N_\theta + 1$ points are calculated. N_θ is specified as an `<arg>` in command line option

```
-ntheta <arg>
```

By default N_θ is from 90 to 720 depending on the size of the computational grid (§8.1). To calculate the Mueller matrix in one scattering plane **ADDA** simulates two incident polarizations, however one is enough if the particle is symmetric with respect to the rotation by 90° over the propagation vector of incident radiation (§8.5).

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, that are by default specified in a file `scat_params.dat` (§B.4). The first flag indicates that values of the Mueller matrix at 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. `<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.[28] Results of the integrations with multipliers specified by the `<arg>` 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 command line options are specified, both initial and integrated results are saved to hard disk.

The format of the input file is very flexible (see §B.4 for details) allowing using either uniformly spaced values 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 Romberg integration (§13.5). 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 *yz*-plane is by default not calculated. However, **ADDA** may be forced to calculate it by specifying command line option

```
-yz
```

12.2 Integral scattering quantities

All the scattering quantities described in this section are saved to file `CrossSec` (§C.6). Different files are used for two incident polarizations and when doing orientation averaging: `CrossSec-X`, `CrossSec-Y`, and `CrossSec` respectively. **ADDA** always calculates extinction and absorption cross sections C_{ext}, C_{abs} (together with corresponding efficiencies Q_{ext}, Q_{abs}) and it can optionally calculate scattering cross section C_{sca} (and efficiency Q_{sca}) and normalized and not-normalized asymmetry vectors – $\mathbf{g}$ and $\mathbf{g}C_{sca}$ respectively (the *z*-component of the $\mathbf{g}$ is the usual asymmetry parameter $\langle\cos\theta\rangle$). All the efficiencies are calculated by dividing the corresponding cross section over the area of the geometrical cross section of the volume-equivalent sphere. The optional features are turned on by command line options

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

[28] For example 1 corresponds to one integration with no multipliers, 6 – to two integration with cos(2φ) and sin(2φ) multipliers. Integration over φ with such multipliers is implemented because it appears in formulae for the light scattering patterns measured by the Scanning Flow Cytometer [22, 35], however they hopefully may be also useful in other applications.

for calculation of C_{sca}, $\mathbf{g}C_{sca}$ and $\mathbf{g}$ respectively. If $\mathbf{g}$ is calculated C_{sca} and $\mathbf{g}C_{sca}$ are also calculated automatically.[29] The calculation of $\mathbf{g}$ and C_{sca} is performed by integration over the whole solid angle, the grid of scattering angles is used for this. The grid is specified by default in file `alldir_params.dat` (see §B.3 for format) in a form suitable for Romberg integration (§13.5). Integration points for θ are spaced uniformly in values of $\cos\theta$. Different file describing the grid can be used if specified as an argument to the command line option

```
-alldir_inp <filename>
```

12.3 Radiation forces

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

```
-Cpr_mat
```

is specified, the radiation force and efficiency vector are calculated and saved into file `CrossSec` (§C.6). If *additionally* an option

```
-store_force
```

is specified, the radiation forces on each dipole is saved into file `VisFrp` (§C.7). These features are still under development. More information can be found in a paper by Hoekstra *et al.* [15].

12.4 Internal fields

ADDA can save internal electric fields (actual, not exciting) on each dipole to file `IntField` (§C.8). To enable this functionality specify command line option

```
-store_int_field
```

This option is mainly for graphical representation and/or analysis of the internal fields. To save internal fields for future use by **ADDA** consider using checkpoints of type "always" (§13.4). This feature was used by Hoekstra *et al.* to analyze the accuracy of the internal fields in DDA simulation [18].

13 Computational Issues

13.1 Iterative solver

Main calculation part of a DDA simulation is finding a solution of a large system of linear equations; an iterative method is used for this purpose. **ADDA** incorporates 4 different methods: Conjugate Gradient applied to Normalized equations with minimization of the Residual norm (CGNR) [36], Bi-Conjugate Gradient (Bi-CG) [37,38], Bi-CG STABilized (Bi-CGSTAB) [36] and Quasi Minimal Residual (QMR) [37]. Bi-CG and QMR employ the complex symmetric property of DDA interaction matrix to reduce the number of matrix-vector products per iteration by a factor of two [37].

The linear system, which is actually solved in **ADDA**, is in the form that is equivalent to the one that is Jacobi-preconditioned but kept complex-symmetric. Our experience suggests that QMR is generally the most efficient iterative solvers, however Bi-CGSTAB is faster in some cases. Performance of Bi-CG is comparable to that of QMR, but its convergence behavior is erratic, compared to that of Bi-CGSTAB. CGNR is the slowest of the four, however it is very simple and its convergence is guaranteed to be monotonic [36]. QMR and BiCGSTAB require about 20% more RAM (for additional intermediate vectors) than CGNR and Bi-CG. Hence, Bi-CG may be preferential when memory is sparse.

The iterative solver is chosen by the command line option

```
-iter <type>
```

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

where `<type>` is one of: `cgnr`, `bicg`, `bicgstab`, `qmr`. By default QMR is used. The initial vector is automatically chosen by **ADDA** from two variants: zero or the incident field. The former is completely general, while the latter may be closer to the exact solution for small index-matching particles. **ADDA** chooses the variant that gives lesser residual norm,[30] this choice is shown in the `log` (§C.4).

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

```
-eps <arg>
```

where $\varepsilon = 10^{-\langle arg \rangle}$. By default $\varepsilon = 10^{-5}$. 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, that is not expected to be ever reached.[31]

13.2 Fast Fourier Transform

The iterative method to solve a system of linear equations that arouse in DDA accesses the interaction matrix only by the means of calculating matrix-vector products. This can be done in O(Nln$N$) operations (N – total number of dipoles) using the FFT [39]. 3D (parallel) FFT is used in **ADDA**, however it is explicitly decomposed into a set of 1D FFTs, which allows reduction of calculations since only part of the array, on which FFT is performed, is actually used (see [14] for details).

1D FFTs are performed using standard libraries – two are implemented in **ADDA**: a routine by Temperton (`CFFT99`, [40]), which is included in the code, or the more advanced package FFTW 3 [41]. The latter is generally significantly faster, but requires separate installation of the package (§5). The FFT routine to use is chosen at compile time. By default FFTW 3 is used; to use Temperton's routine uncomment the line

```
CFLAGS += -DFFT_TEMPERTON
```

in `Makefile` and recompile (see §5).

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), FFTW 3 works with any grid dimensions but is most effective for dimensions of the form $2^p3^q5^r7^s$. It should not be a problem for sequential mode, since **ADDA** automatically increases the FFT-grid dimensions to the first number of the required form.[32] 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 or 7 for Temperton FFT and FFTW 3 respectively. Therefore it is strongly recommended *not* to use such "weird" number of processors.[33] 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 §8.4).

[30] It should be noted, however, that smaller residual of the initial vector does not necessarily leads to a faster convergence [36].
[31] Currently it is set to 3N, i.e. the number of equations in a linear system.
[32] The maximum increase is 15% for Temperton FFT and 9% for FFTW 3.
[33] Otherwise Temperton FFT will fail and FFTW 3 will perform less efficiently.

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 (§11.2). This option can be disabled (mainly for debugging purposes) by specifying

```
-no_reduced_fft
```

in the command line.

13.3 Parallel performance

ADDA is capable of running on a multiprocessor system, parallelizing a single DDA simulation. It uses MPI for communication routines. The principal limitation of DDA simulations on a desktop system is the amount of RAM available. For **ADDA** this limitation only specifies the minimum number of nodes (with separate memory for each node) to use. More nodes can be used to accelerate the computations. However, the more nodes assigned – the more simulation time (relative to the total time) is spent on communications. One should also take into account that when many nodes (processors) are used the MPI interface may occupy significant amount of RAM on each node, thereby decreasing the RAM available for **ADDA** itself (see also §7 and §8.4). This depends on the specific MPI implementation and/or hardware.

13.4 Checkpoints

ADDA is capable of creating checkpoints, in which the complete running state is saved and can be restored afterwards. All the intermediate vectors of the iterative solver (§13.1) are saved. This allows restarting the iterative solver exactly at the position, where the checkpoint was saved. Time of a checkpoint is specified by command line option

```
-chpoint <time>
```

where `<time>` is time in format "`#d#h#m#s`".[34] There are 3 possible strategies for checkpoints, which are specified by 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 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 (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, §13.1) or to calculate different scattering quantities (§12).

To restart the simulation from a checkpoint specify in the command line

```
-chp_load
```

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

[34] All fields are optional, "`s`" can be omitted, the format is not case sensitive. For example: "`12h30M`", "`1D10s`", "`3600`" (equals 1 hour).

different directory (the same for saving and loading of checkpoints) can be specified as an argument to 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). The 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.[35] 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 to checkpoint, therefore it is suitable only for a simulation for a single incident polarization. We are working to extend its applicability to standard non-symmetric particles (two incident polarizations) and to orientation averaging.

13.5 Romberg integration

Integration is performed in **ADDA** for different purposes: orientation averaging (§9.2), integration of the Mueller matrix over the azimuthal angle (§12.1), and integration of the scattered field over the whole solid angle (§12.2). The same routine is used for all these purposes, which is based on the one- or two-dimensional Romberg integration [42]. It is a high-order technique that may be used in adaptive mode (it automatically calculates only the necessary number of function values to reach a prescribed accuracy). Adaptability is relevant for orientation averaging, where each function value is a complete DDA simulation, but not for integration over scattering angles, because in this case all the values are precalculated. Romberg integration also provides an estimate of the integration error, however this estimate may be far from real. The information about the integration together with errors is saved to separate log files (§C.9): `log_orient_avg`, `log_int_Csca`, `log_int_asym_x`, `log_int_asym_y`, `log_int_asym_z` for orientation averaging, calculation of C_{sca} and each component of **g** respectively. 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 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). These values are specified by minimum and maximum values and maximum number of subdivisions $J_{\max}$ (equal to $n+1$). The parameter K specifies the number of points for extrapolation, which effectively characterizes the order of the integration scheme (see [42] for details). Extrapolation starts only after several subdivisions.[36] 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

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

[36] defined by `ROMB_KMIN` in the beginning of the `Romberg.h` (now it is 3). $J_{\max}$ must not be smaller than it.

φ), **ADDA** accounts for it to slightly decrease simulation time. All these 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.

14 Timing

14.1 Basic timing

The basic timing of **ADDA** execution 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 most systems) of the processor timer. **ADDA** uses wall-time only for the total execution time and timing of checkpoints (§13.4), and for the rest processor time is measured. This makes timing more precise especially on desktop computers that are partly occupied by other tasks, however for long simulations some timing results become meaningless because of timer overflow. We are currently working on this problem.

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). Timing covers all the major parts: initialization (including initialization of FFT, §13.2, building the interaction matrix, and constructing a dipole representation of the particle, §8.2), solution for the internal fields (including iterative solver, §13.1), calculation of the scattering quantities (scattered electric field and others, §12), input/output (including checkpoint loading/saving, §13.4), integration (§13.5). Some are divided into subsections. Communication time (between different processors on a multiprocessor system) is shown separately where relevant.

14.2 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 FFTW 3, §13.2) 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[37] 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 right one is automatically chosen by compiler directives.[38] To enable precise timing uncomment the line

```
CFLAGS += -DPRECISE_TIMING
```

in `Makefile` and recompile (see §6).

15 Acknowledgements

- The CFFT99 Fortran routine was written by Clive Temperton (1980).
- The FFTW 3 package, to which we link, was written by Matteo Frigo and Steven G. Johnson (fftw@fftw.org).

[37] It is hard to implement routines to measure processor time with better precision than that of standard `clock`, since all such routines are processor dependent.

[38] 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`.

- The FFT part of **ADDA** (`fft.c`), matrix-vector product (`matvec.c`), and most of the non-standard beam types (`GenerateB.c`) were first implemented by Michel D. Grimminck. He also contributed to the particle generation routines (`make_particle.c`)
- The MPI part of **ADDA** (`comm.c`), 2D Romberg integration (`Romberg.c`), and calculation of the radiation force and scattering quantities obtained by integration (`crosssec.c`) were first implemented by Martin Frijlink.
- Several new shapes were added to the older version of particle generation routine (`make_particle.c`) by Konstantin A. Semyanov.

We are deeply indebted to all of these authors for making their code available.

We wish also to thank Bruce T. Draine and Piotr J. Flatau for creating and developing the first publicly available DDA code "DDSCAT" [23]. They maintain a high standard, which we try to match, both for highly productive convenient code and clear detailed manual. Parts of the User Guide for DDSCAT 6.1 [23] were used when writing this manual.

Development of ADDA was supported by The University of Amsterdam (The Netherlands), The Dutch Science Foundation NWO (The Netherlands), The Department of Defense (USA), and The NATO Science for Peace program.

References

1. F. M. Kahnert, "Numerical methods in electromagnetic scattering theory," J. Quant. Spectrosc. Radiat. Transf. **79**, 775-824 (2003).
2. E. M. Purcell and C. R. Pennypacker, "Scattering and adsorption of light by nonspherical dielectric grains," Astrophys. J. **186**, 705-714 (1973).
3. B. T. Draine, "The discrete-dipole approximation and its application to interstellar graphite grains," Astrophys. J. **333**, 848-872 (1988).
4. 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).
5. B. T. Draine and P. J. Flatau, "Discrete-dipole approximation for scattering calculations," J. Opt. Soc. Am. A **11**, 1491-1499 (1994).
6. 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., (Academic Press, 2000), pp. 131-145.
7. 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).
8. A. Lakhtakia, "Strong and weak forms of the method of moments and the coupled dipole method for scattering of time-harmonic electromagnetic-fields," Int. J. Mod. Phys. C **3**, 583-603 (1992).
9. J. Rahola, "Solution of dense systems of linear equations in the discrete-dipole approximation," SIAM J. Sci. Comp. **17**, 78-89 (1996).
10. N. B. Piller, "Coupled-dipole approximation for high permittivity materials," Opt. Comm. **160**, 10-14 (1999).
11. P. C. Chaumet, A. Sentenac, and A. Rahmani, "Coupled dipole method for scatterers with large permittivity," Phys. Rev. E **70**, 036606 (2004).
12. 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).
13. 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).
14. 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).
15. 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).
16. S. B. Singham, "Theoretical factors in modeling polarized light scattering by arbitrary particles," Appl. Opt. **28**, 5058-5064 (1989).
17. 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).
18. 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).
19. M. A. Yurkin, V. P. Maltsev, and A. G. Hoekstra, "Convergence of the discrete dipole approximation. Part 2: an extrapolation technique to increase the accuracy," submitted to J. Opt. Soc. Am. A (2006).
20. M. A. Yurkin and A. G. Hoekstra, "Capabilities of the discrete dipole approximation for simulation light scattering by biological cells," in *Proceedings of Optics of Biological Particles* (Novosibirsk, Russia, 2005), pp. 19-20.

21. P. W. Kuchel and E. D. Fackerell, "Parametric-equation representation of biconcave erythrocytes," Bull. Math. Biol. **61**, 209-220 (1999).
22. 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).
23. B. T. Draine and P. J. Flatau "User guide for the discrete dipole approximation code DDSCAT 6.1," http://xxx.arxiv.org/abs/astro-ph/0409262 (2004).
24. C. F. Bohren and D. R. Huffman, *Absorption and scattering of Light by Small Particles*, (Wiley, New York, 1983).
25. L. W. Davis, "Theory of electromagnetic beams," Phys. Rev. A **19**, 1177-1179 (1979).
26. 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).
27. G. Gouesbet, G. Grehan, and B. Maheu, "Localized interpretation to compute all the coefficients gn(M) In the generalized lorenz-mie theory," J. Opt. Soc. Am. A **7**, 998-1007 (1990).
28. 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).
29. J. I. Hage and J. M. Greenberg, "A model for the optical-properties of porous grains," Astrophys. J. **361**, 251-259 (1990).
30. C. E. Dungey and C. F. Bohren, "Light-scattering by nonspherical particles - a refinement to the coupled-dipole method," J. Opt. Soc. Am. A **8**, 81-87 (1991).
31. H. Okamoto, "Light scattering by clusters: The A1-term method," Opt. Rev. **2**, 407-412 (1995).
32. J. I. Peltoniemi, "Variational volume integral equation method for electromagnetic scattering by irregular grains," J. Quant. Spectrosc. Radiat. Transf. **55**, 637-647 (1996).
33. D. Gutkowicz-Krusin and B. T. Draine "Propagation of electromagnetic waves on a rectangular lattice of polarizable points," http://xxx.arxiv.org/abs/astro-ph/0403082 (2004).
34. N. B. Piller and O. J. F. Martin, "Increasing the performance of the coupled-dipole approximation: A spectral approach," IEEE Trans. Ant. Propag. **46**, 1126-1137 (1998).
35. V. P. Maltsev and K. A. Semyanov, *Characterisation of Bio-Particles from Light Scattering*, (VSP, Utrecht, 2004).
36. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. A. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, (SIAM, 1994).
37. R. W. Freund, "Conjugate gradient-type methods for linear-systems with complex symmetrical coefficient matrices," SIAM J. Sci. Stat. Comp. **13**, 425-448 (1992).
38. 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).
39. 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).
40. C. Temperton, "Self-sorting mixed-radix fast Fourier transforms," J. Comp. Phys. **52**, 1-23 (1983).
41. M. Frigo and S. G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," Proc. ICASSP **3**, 1381-1384 (1998).
42. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C. The Art of Scientific Computing*, (Cambridge University Press, 1990).

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 options cause abrupt termination or any erroneous behavior of **ADDA**, please communicate it to the authors. 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 for all the flags (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 (§12.2). Input format is described in §B.3.
Default: `alldir_params.dat`

`-asym`
: Calculate the asymmetry vector (§12.2). Implies `-Csca` and `-vec`.

`-beam {plane|{barton1|barton3|barton5|davis1|davis3|lminus} <width> <x> <y> <z>}`
: Sets a type of the incident beam (§10.2). Four other float arguments must be specified for all beam types except the plane wave. These are the width and *x*, *y*, *z* coordinates of the center of the beam respectively (all in μm).
Default: `plane`
Example: `barton1 5 0.3 0 0`

`-chp_dir <dirname>`
: Sets directory for the checkpoint (both for saving and loading, §13.4).
Default: `chpoint`

`-chp_load`
: Restart a simulation from a checkpoint (§13.4).

`-chp_type {normal|regular|always}`
: Sets type of the checkpoint (§13.4). All types, except `always`, require `-chpoint`.
Default: `normal`

`-chpoint <time>`
: Specifies the time for checkpoints (§13.4) 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_mat`
: Calculate the total radiation force (§12.3).

`-Csca`
: Calculate scattering cross section (by integrating the scattered field, §12.2).

`-dir <dirname>`

Sets directory for output files.
Default: constructed automatically, see §C.3.

`-dpl <arg>`

Sets parameter "dipoles per lambda" (§8.1), float.
Default: $10|m|$, where m is the first refractive index specified by the "`-m`" option (§8.2).

`-eps <arg>`

Specifies the stopping criterion for the iterative solver (§13.1) by setting the relative error of the residual ε to reach. `<arg>` is an exponent of base 10 (float), i.e. $\varepsilon = 10^{-\langle arg \rangle}$.
Default: `5` ($\varepsilon = 10^{-5}$)

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

Sets dimensions of the computation grid (§8.1). 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`, §8.2). If `-jagged` option is used the grid dimension is effectively multiplied by the specified number (§8.2).
Default: `16` (if `-size` is not specified) or defined by `-size`, `-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`

`-int {poi|so}`

Sets prescription to calculate interaction term (§11.2).
Default: `poi`

`-iter {cgnr|bicg|bicgstab|qmr}`

Sets the iterative solver (§13.1).
Default: `qmr`

`-jagged <arg>`

Sets a size of a big dipole in units of small dipoles (§8.2), integer. It is used to improve the discretization of the particle without changing the shape.
Default: `1`

`-lambda <arg>`

Sets incident wavelength in μm (§8.1), float.
Default: 2π

`-m <m1Re> <m1Im> [<m2Re> <m2Im>…]`

Sets refractive indices (§8.2), float. Each pair of arguments specifies real and imaginary part of the refractive index of one of the domains. Maximum number of different refractive indices (particle domains) is defined at compilation time by the parameter `MAX_NMAT` in file `const.h` (by default, 10).
Default: `1.5 0`

`-maxiter <arg>`

Sets the maximum number of iterations of the iterative solver, integer.
Default: very large, not realistic value (§13.1).

`-no_reduced_fft`

Do not use symmetry of the interaction matrix to reduce the storage space for the Fourier-transformed matrix (§13.2).

`-no_vol_cor`

Do not use "dpl correction", which ensures (if used) that the volume of the dipole representation of the particle is exactly correct (§8.2).

`-ntheta <arg>`

Sets the number of intervals into which range of scattering angles [0°,180°] is equally divided (§12.1), integer. This is used for scattering angles in *yz*-plane. If particle is not symmetric (§8.5) and orientation averaging (§9.2) 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 (§8.1).

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

Either sets an orientation of the particle by three Euler angles α, β, γ (§9.1) or specifies that orientation averaging should be performed (§9.2). `<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 (§12.1). `<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, $\cos(2\varphi)$, $\sin(2\varphi)$, $\cos(4\varphi)$, and $\sin(4\varphi)$ respectively.
Examples: `1` (one integration with no multipliers), `6` (two integration with $\cos(2\varphi)$ and $\sin(2\varphi)$ multipliers).

`-pol {cm|rrc|ldr [avgpol]|cldr|so}`

Type of polarization prescription (§11.1). An optional flag `avg` can be added for LDR – it specifies that LDR polarizability should be averaged over incident polarizations.
Default: `ldr` (without averaging).

`-prognose`

Do not actually perform simulation (not even memory allocation) but only estimate the required RAM (§7). Implies `-test`.

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

Sets propagation direction of incident radiation (§10.1), float. Normalization (to the unity vector) is performed automatically by **ADDA**.
Default: `0 0 1`

`-save_geom [<filename>]`

Saves dipole configuration to a file `<filename>` (a path relative to the output directory, §8.2). Output format is described in §C.10. Can be used with `-prognose`.
Default: `<type>.geom` (`<type>` is a first argument to the `-shape` option).

`-scat {dr|so}`
: Sets prescription to calculate scattering quantities (§11.3).
Default: `dr`

`-scat_grid_inp <filename>`
: Specifies a file with parameters of the grid of scattering angles for calculating Mueller matrix (possibly integrated over φ , §12.1). Input format is described in §B.4.
Default: `scat_params.dat`

`-shape {box|coated <d_in/d> [<x/d> <y/d> <z/d>]|`
`cylinder <h/d>|ellipsoid <y/x> <z/x>|line|`
`rbc <h/d> <b/d> <c/d>|read <filename>|sphere|`
`spherebox <d_sph/D_x>}`
: Sets shape of the particle, either predefined (§8.3) or "`read`" from file (§8.2). All the parameters of predefined shapes are floats, described in detail in §8.3.
Default: `sphere`

`-size <arg>`
: Sets the size of the computational grid along the *x*-axis in μm (§8.1), float.
Default: determined by the values of `-grid`, `-dpl`, and `-lambda`.

`-store_force`
: Calculate the radiation force on each dipole (§12.3). Requires `-cpr_mat`.

`-store_int_field`
: Save internal fields to a file (§12.4). Output format is described in §C.8.

`-store_scat_grid`
: Calculate Mueller matrix for a grid of scattering angles and save it to a file (§12.1). Output format is described in §C.5.

`-sym {no|enf}`
: Do not take into account (`no`) or enforce (`enf`) all particle symmetries (§8.5).

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

`-vec`
: Calculate the not-normalized asymmetry vector (§12.2).

`-yz`
: Calculate the Mueller matrix in *yz*-plane even if it is calculated for a scattering grid (§12.1). If the latter option is not enabled, scattering in *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 that may be 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 authors. All files in this section are in ASCII format to ensure portability.

B.1 ExpCount

This is 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 (see §6.2) this problem is alleviated by adding PBS 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 Unix systems `ExpCount.lck` is additionally locked to ensure robustness when working over the Network File System (NFS), 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 this 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 file locking correctly on any POSIX-compliant or Windows operating system. If any problem with file locking takes place it can be completely turned off by uncommenting the line

```
CFLAGS += -DNOT_USE_LOCK
```

in `Makefile` and recompiling (see §5).

B.2 avg_params.dat

This file specifies parameters for orientation averaging (§9.2). It consists of three sections, each specifying the parameters for each of the Euler angles: α, β, and γ. The first section looks like

```
alpha:
min=0
max=360
Jmax=5
K=5
eps=0
equiv=true
```

specifying minimum and maximum angles, number of subdivisions, order of extrapolation, required accuracy, and whether the minimum and maximum angles are equivalent (see §13.5 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` is relevant for β and γ, but it is 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 cosβ inside the specified interval.

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 *z*-axis), γ is not relevant and user should set

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

This will dramatically increase the speed of orientation averaging. 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.

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 §9.2).

B.3 alldir_params.dat

This file specifies parameters for averaging over the scattering angles for calculating integral scattering quantities (§12.2). 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 cosθ inside the specified interval.

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), φ is not relevant and user should set

```
phi:
min=0
max=0
```

It will dramatically 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 §12.2).

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 (§12.1). 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 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 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` does not decrease the computational time, since all the integrated values are precalculated, but may decrease accuracy. 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.

Particle symmetries may be considered by the user to decrease e.g. the range of φ that is used for integrating. 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. 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 §12.1).

B.5 Geometry files

This file specifies the shape of the scatterer (§8.2). Two formats are supported: for one- and multi-domain particles. One-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). They should be non-negative, the maximum coordinates in each direction is automatically assigned (incremented by 1) to be the size of the computational grid (§8.1). 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`).

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 different warnings, that appear only in `stdout`. The `stdout` from the sample calculation (§6.1) looks like

```
'Amsterdam DDA' v.0.74.2
Copyright (C) 2006 M.A. Yurkin and A.G. Hoekstra

all data is saved in 'run000_sphere_g16m1_5'
0 :  0 16 16 4096 32 0
lambda: 6.28319   m0: 1.5+0i   Dipoles/lambda: 15
Required relative error: 1e-05
Total number of occupied dipoles: 2176
Memory usage for MatVec matrices: 1.3 Mb
Calculating Dmatrix......
Initializing FFTW3
Total memory usage: 2.2 Mb

here we go, calc Y

CoupleConstant:0.005259037197+1.843854148e-05i
x_0 = 0
RE_000 = 1.0000000000E+00
RE_001 = 8.4752662637E-01  +
…
RE_022 = 3.1681098360E-06  +
Cext   = 135.0449046
Qext   = 3.791149609
Cabs   = 1.36464414e-16
Qabs   = 3.830999855e-18
end 0
```

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. The third line in the example above displays some internal information (for parallel run it shows subdivision of dipoles over the processors). At the end each processor generates an exit message. When precise timing (§14.2) is enabled, all results go to `stdout`.

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<PBSid>]
```

where `<type>` is either `run` or `test`. The latter is only used if `-prognose` (§7) 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 a first argument to the `-shape` command line option (§8.2, §8.3). `<nx>` is dimension of the computational grid along *x*-axis (§8.1), `<m1Re>` is real part of the first given refractive index (§8.2) written with up to 4 significant digits and decimal point replaced by "_". The last part of the name is added only if environmental variable `PBS_JOBID` is defined (§6.2), then `<PBSid>` is its value. For examples, directory name may look like:

```
run000_sphere_g16m1_5
test123_box_g40m1_33id123456
```

The first one corresponds to the sample simulation (§6.1), it is included in the distribution with 3 output files in it (`sample/run000_sphere_g16m1_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 (§6.1) is the following

```
Generated by ADDA v.0.74.2
The program was run on: dda
command: './adda '
lambda: 6.28319
shape: sphere; diameter:6.73455
box dimensions: 16x16x16
refractive index: 1.5+0i
Dipoles/lambda: 15
      (Volume correction used)
Required relative error: 1e-05
Total number of occupied dipoles: 2176
Volume-equivalent size parameter: 3.36728

Incident propagation vector: (0,0,1)
Incident polarization Y(par): (0,1,0)
Incident polarization X(per): (1,0,0)

Particle orientation (deg): alpha=0, beta=0, gamma=0
```

```
Polarization relation: 'Lattice Dispersion Relation'
Scattering quantities formulae: 'by Draine'
Interaction term prescription: 'as Point dipoles'
FFT algorithm: FFTW3
Iterative Method: QMR (complex symmetric)
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-05i
x_0 = 0
RE_000 = 1.0000000000E+00
RE_001 = 8.4752662637E-01  +   progress = 0.152473
…
RE_022 = 3.1681098360E-06  +   progress = 0.791157

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

total time:          0.5800
Wall time:           0.0
Initialization time:   0.0800
  init Dmatrix           0.0200
  FFT setup:             0.0600
  make particle:         0.0000
Internal fields:       0.3700
  one solution:          0.3700
    init solver:           0.0100
    one iteration:         0.0200
      calculation:           0.0200
      communication:         0.0000
E field calculation:   0.1100
  one plane:             0.0500
    calculation:           0.0500
    communication:         0.0000
Other scat.quantities: 0.0000
file io:               0.0100
Integration:           0.0000
```

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 (§8) is completely described, then the incident beam (§10) and scatterer orientation (§9). The DDA formulation (§11) is described, then FFT algorithm (§13.2) and iterative method (§13.1) are specified. Memory usage is given (both total and for FFT part, §7). "`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 (§13.1). 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 (§14.1). 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_g16m1_5/log`).

C.5 mueller

This file contains results for Mueller matrix at different scattering angles (§12.1). 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 (§9.2) is performed the simplest file `mueller` is produced:

```
    theta s11 s12 s13 s14 s21 … s44
    0.00 1.4154797788E+02 0.0000000000E+00 0.0000000000E+00 \
-5.4378623975E-11 0.0000000000E+00 … 1.4154797788E+02
    1.00 1.4140075332E+02 -5.8903788409E-03 -2.1360205695E-12 \
-5.0978006572E-11 -5.8903788409E-03 … 1.4140075285E+02
    …
    180.00 2.9143742348E+00 4.6913428003E-16 1.7535128407E-18 \
4.4031933351E-12 4.6913428003E-16 … -2.9143742348E+00
```

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 (§6.1) – `sample/run000_sphere_g16m1_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 1.4154797788E+02 … 1.4154797788E+02
    …
    180.00 360.00 2.9143742348E+00 … -2.9143742348E+00
```

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, §12.1). 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 1.4154797788E+02 1.4154797788E+02 9.564E-18
    …
    50.00 8.7607151454E+00 … 8.4048020863E+00 3.502E-08
    …
    180.00 2.9143742348E+00 … -2.9143742348E+00 4.385E-18
```

The shown error is root mean-square error over all 16 elements of Mueller matrix, integration error of each element is an estimation from Romberg routine (§13.5) – it should be taken very seriously, since it can greatly underestimate the real error. 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).

C.6 CrossSec

This file contains the results for integral scattering quantities (§12.2). If orientation averaging (§9.2) 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 (§8.5). The format is self-explanative, for example the output of the sample simulation (§6.1) looks like (`sample/run000_sphere_g16m1_5/Crosssec-Y`):

```
    Cext   = 135.0449046
    Qext   = 3.791149609
    Cabs   = 1.36464414e-16
    Qabs   = 3.830999855e-18
```

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

C.7 VisFrp

This file stores the results for radiation force on each dipole (§12.3). A separate file is used for each simulated incident polarization: `VisFrp-X.dat` and `VisFrp-Y.dat`. Currently the result looks like

```
    #sphere  x=3.367275909  m=1.5+0i
    #number of dipoles  2176
    #Forces per dipole
    #r.x r.y r.z F.x F.y F.z
    -0.2094395102 -1.047197551 -3.141592654 -0.001450116669 \
0.004378388086 -0.001487326112
    …
    0.2094395102 1.047197551 3.141592654 0.01038653738 \
-0.04118221215 -0.006040333419
```

However it is going to be significantly revised in the future (no comments are given here, this feature should be considered as being in development). The above file can be produced by the command

```
    adda -Cpr_mat -store_force
```

C.8 IntField

Internal field are saved to this file. A separate file is used for each simulated incident polarization: `IntField-X and IntField-Y`. The file 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). The squared norm of the electric field is presented for convenience (it is electromagnetic energy density, scaled by a constant). The above file can be produced by the command

```
    adda -store_int_field
```

C.9 log_orient_avg and log_int

These files contain information about the 2D Romberg integration (§13.5), they are produced directly by the routine and hence have the same format. Their names are `log_orient_avg`, `log_int_Csca`, and `log_int_asym_x`, `log_int_asym_y`, `log_int_asym_z` for orientation averaging (§9.2) and calculation of scattering cross section and asymmetry vector (§12.2) respectively. For example, `log_int_Csca` looks like:

```
                    PHI(rad)   cos(THETA)
EPS                    0          0
Maximum number of
refinement-stages      6          6
Number of evaluations
for an extrapolation   6          6
lower boundary         0          -1
upper boundary         6.28319    1


Outer-Loop  Inner Loop
Inner_qromb converged only to d=6.26182e-21 for cosine value #0
Inner_qromb converged only to d=6.01778e-20 for cosine value #32
1      64 integrand-values were used.
Inner_qromb converged only to d=3.35879e-07 for cosine value #16
2      32 integrand-values were used.
Inner_qromb converged only to d=1.34869e-07 for cosine value #8
Inner_qromb converged only to d=1.64201e-07 for cosine value #24
3      64 integrand-values were used.
…
Inner_qromb converged only to d=4.95342e-09 for cosine value #31
6      512 integrand-values were used.
33 inner integrations did not converge.
The outer integration did not converge
The outer integration reached d=1.12252e-08
In total 1056 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 error falls below `eps` (shown in the second line), which is given in a corresponding parameter file. For this example no adaptation is used (`eps` = 0), hence none of the integrals converge, but the reached errors are informative. The range for angles (φ or γ) is specified in *radians*. The above file can be produced by the command

```
adda -Csca
```

C.10 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. Either one- or multi-domain format (§B.5) 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.0.74.2
#shape: 'sphere'
#box size: 16x16x16
7 5 0
…
8 10 15
```

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 SO prescription for the interaction term (§11.2). Since the latter feature is still in development, 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 (§13.4), 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 (§13.4). 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.[39]

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