Nano-particle Transition Matrix code Release Notes

Report for M9.01

G. La Mura, G. Mulas

October 2024

Contents

Scope of the document

This document is provided along with the M9 release of the `NP_TMcode` suite as a quick report on the status of the project. The aim of this document is to give an overview of the project goals, to provide a quick guide to navigate the progress achieved with respect to milestones and to give a set of fundamental instructions to perform tests of the code functionality. More detailed documentation, explaining the structure of the code and the role of its main components (data structures, functions and variables) is given in the form of `doxygen`-handled inline documentation.

This document is organized in chapters:

- Chapter 1 presents the general scope of the project and a description of its milestones;

- Chapter 2 provides a description of the current code release and discusses how the project guidelines were implemented;

- Chapter 3 finally gives instructions on how to test the code release in its current form.

The contents of this document update the information included in the *Release Notes* of M7.00 and M8.03, also included in the distribution. The introductory discussion presented in Chapter 1 is similar to the one given in the previous releases. The contents of Chapters 2 and 3 assume that the reader is aware of the discussion presented in the release notes of M7.00 and M8.03.

Chapter 1

Aim of the project

1.1   Introduction

This project aims at implementing High Performance Computing (HPC) strategies to accelerate the execution of the Nano-Particle Transition Matrix code, developed by Borghese et al. (2007), to solve the scattering and absorption of radiation by particles with arbitrary geometry and optical properties. The goal is to migrate the original code, written in *FORTRAN 66*, to a modern programming language, able to access new hardware technologies, thus substantially reducing the amount of time required for calculations through the use of parallel code execution handled by multi-core computing units.

The problem of radiation absorption and scattering has a large variety of applications, ranging from Astrophysics of the interstellar medium (ISM) and (exo)planetary atmospheres, all the way to material investigation through optical techniques and nano-particle handling by means of optical tweezers. In spite of the large impact of such problems on both scientific and technological applications, the theoretical framework of radiation/matter interactions has mostly been treated under the assumption of simplifying conditions, such as plane-wave radiation fields interacting with spherical particles.

Dealing with more realistic cases is only possible through numerical calculations. These can be broadly distinguished in the *Discrete Dipole Approximation* (DDA) based approach (Draine & Flatau 1994), which subdivides a general material particle in a properly chosen combination of dipoles, thereby

solving their interaction with the radiation field, and the *Transition Matrix* (T-matrix) solutions (Borghese et al. 2007) which, on the contrary, take advantage from the expansion of the radiation fields in multi-polar spherical harmonics to create a set of boundary conditions that connect the properties of the incident and of the scattered radiation at the surface of the interacting particle layers. In the latter case, the particle is approximated as a collection of spherical sub-particles that acts as a mathematical operator connecting the properties of the incident and scattered field.

The T-matrix method has the main advantage of creating a unique link between the incident and the scattered radiation fields, offering a solution that is valid for any combination of incident and scattered directions and at various distance scales, from within the particle itself, all the way up to remote regions. While the T-matrix itself is computed numerically, it then provides an analytical expression of incident and scattered fields for any arbitrary incident wave, enabling (relatively) easy arbitrary averages over combinations of relative orientations of incident fields and complex scattering particles. In particular, it is easy to account for different partial alignment conditions, with no need to repeat the calculation of the T-matrix.

Conversely, DDA calculations need to be entirely solved for each combination of incident and scattered radiation fields and they require a different description of the particle, depending on the dimension scale they apply to. As a consequence, the T-matrix method is largely preferable for the investigation of all types of integrated effects, thus naturally covering also the dynamic and thermal effects of the radiation-particle interaction.

While T-matrix based solutions are ideal to solve the scattering problem in all circumstances where multiple scales and directions need to be accounted for, the calculation of the Transition Matrix for realistic particles is a computationally demanding task. The `NP_TMcode` project takes on the challenge of porting the original algorithms to modern hardware, in order to substantially reduce the computational times and allow users to model more complicated and realistic particle structures, and/or large populations of different particles, in shorter execution times.

4

1.2   Project break-down

We can broadly divide the project in three main stages, namely corresponding to:

1. code porting to C++ (completed in M7)

2. implementation of parallel algorithms (addressed in M8)

3. **deployment of general radiation/particle interaction solver**

These three stages are associated with an equal number of Key Performance Indicators (KPI) which consist in code releases with enhanced computational abilities. The current project stage provides a parallel implementation of the calculation, built on the basis of the *C++* ported algorithms, that is able to detect the hardware and software capabilities of the host system where it is executed, in order to perform adaptive computing configuration.

1.3   Project status

We refer to the current release of the project as `NP_TMcode-M9.01`, since it includes updates on the documentation and the distribution of the code that address the targets of the project Milestone 9 (namely, a configurable implementation of the radiation/particle interaction solver using different types of multi-thread and multi-core acceleration technologies). The code is distributed through a public `gitLab` repository, under GNU General Public License Version 3. The current implementation provides the following new features with respect to `NP_TMcode-M8.03`:

- a configuration script that detects the hardware and software capabilities of the host, to assist in the identification of the best compilation options;

- the implementation of parallelized algorithms to perform directional calculations;

- the possibility to use multiple GPUs, if available on the host;

- the possibility to distinguish between internal and external maximum field expansion order to speed-up calculations;

- the control of code compilation options through custom configuration settings;

- an expanded set of test cases, diagnostic tools and debug options.

This set of newly implemented features are intended to enhance the user's ability to build the code either on personal workstations or on shared computing systems with different hardware characteristics. In addition, the code optimization features allow for the calculation of higher complexity models with relatively smaller computational effort, leading to the execution of the first scientific testing runs.

1.4   Distribution

The `NP_TMcode` project is developed and distributed at the following gitLab public repository:

> `https://www.ict.inaf.it/gitlab/giacomo.mulas/np_tmcode`

Discussion about the code performance and features is also possible through the gitHub project release portal:

> `https://github.com/GLAMURA81/NP_TMcode_release`

Chapter 2

Release description

2.1 Parallel implementation

`NP_TMcode-M9.01` is the third official release of the *Nano-particle Transition Matrix Project* code,[1] funded under the project *CN-HPC, Big Data and Quantum Computing CN_00000013, Spoke 3 "Astrophysics and Cosmos Observations"* (CUP C53C22000350006). The aim of this release is to to test the ability of the code to be installed and executed on different hardware systems, to solve models of increasing complexity, with respect to the development test cases.

The features introduced in this release followed two main guidelines:

1. All newly introduced features must preserve the consistency with pre-computed test cases.

2. They must result in performance improvements in terms of calculation time and resource management.

In order to improve the code set up flexibility, this release implements a configuration script generated through the `autoconf` system. When executed, the configuration script runs a set of standard tests to detect the host system hardware and software capabilities to choose the most convenient

---

[1]See `https://www.ict.inaf.it/gitlab/giacomo.mulas/np_tmcode/-/releases/NP_TMcode-M7.00` for the first official release and `https://www.ict.inaf.it/gitlab/giacomo.mulas/np_tmcode/-/releases/NP_TMcode-M8.03` for the second.

compiler options. As a consequence, the user is no longer required to manually set compiler flags, as it happened in the previous releases, although the possibility to customize the compilation by manually setting various compiler variables is still available in the form of configuration options. More details on how to configure and use the code to access these possibilities are given in Chapter 3 of this document.

2.2 New release features

Compared with the previous release, `NP_TMcode-M9.01` has some fundamental improvements. These mostly concern the code management of resources at runtime and they do not require large changes in the way the user is expected to work with the code. Many of the newly implemented features are automatically managed, meaning that the code will scan the system to detect the available resources and derive the best set of compilation flags. If necessary, the code can still be compiled and executed in the old-fashioned serial implementation by systems that do not have parallel compilation and execution capabilities. To manually override the default configuration, users can provide their configuration preferences using a set of configuration options that allow to toggle the inclusion of code optimization, of debug information and of the use of external libraries that may reside in different locations, with respect to the system defaults. All such features are detailed in the configuration script run-time help system. The main novelties that `NP_TMcode-M9.01` offers, with respect to the previous releases are:

1. The presence of a configuration script to optimize compilation settings.

2. The possibility to access multiple GPUs to speed up the calculation, if available on the system.

3. The possibility to use distinct field expansion orders to model sub-particle and global scale interactions.

4. An improvement of the system memory management, fixing some bugs uncovered by the introduction of more realistic particle models.

The introduction of these features greatly reduces the complexity of the compilation stage, detailed in Chapter 3, with respect to the requirements of `NP_TMcode-M8.03`. In order to better illustrate the effect, the code has been tested on a set of new particle models, intended to represent a more realistic scattering problem that may serve as a baseline to run simulations of interstellar dust grains. The details of these calculations are presented in the next section.

2.3   Code performance

The code performance for Milestone 9 has been evaluated by applying the code to a new particle model, illustrated in Fig. 2.1. The particle is composed by a combination of 42 spherical monomers, arranged in a core-mantle structure. Such model is intended as a starting point to test the code ability to deal with non-spherically symmetric clusters of units composed by different materials, therefore allowing for a more accurate representation of the conditions that can occur in physical particles such as interstellar dust grains. More in detail, the particle core is represented by 2 spherical monomers, each with radius $\rho_{core} = 68.63$ nm and composed by astronomical silicates, while the mantle is made of 40 spherical monomers, with radius $\rho_{mantle} = 34.31$ nm and assuming amorphous carbon as the composing material. The total particle size, along its largest extension is $l = 400$ nm and the scattering problem is solved for 181 different wavelengths, ranging between 100 nm $\leq \lambda \leq 1 \, \mu$m, in steps of 5 nm. The optical properties of the materials were assumed to be the ones reported by Draine & Lee (1984) for astronomical silicates and by Palik (1991) for amorphous carbon.

Since this case is already too advanced to obtain reference results from the original implementation on reasonable times, the code can no longer be tested by comparing the results of the original implementation with those provided by the `NP_TMcode` project. To solve the problem, instead, we adopted a new approach, based on the execution of the same model on different hardware architectures, using different configuration parameters. We used as a reference case the calculation of the particle's extinction cross-section as a function of wavelength for a fixed maximum field expansion order of $l_{MAX} = 14$ and we
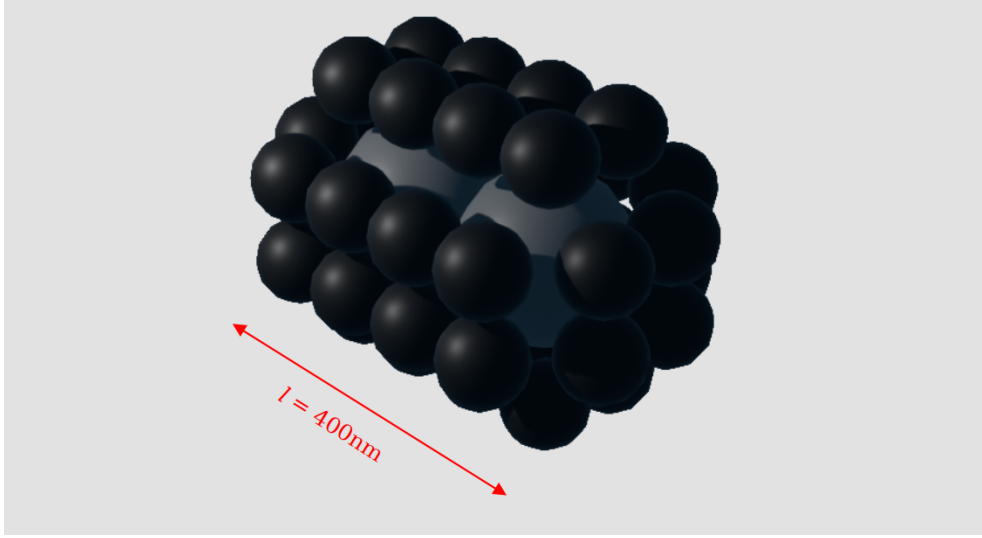
Figure 2.1: Illustration of the model particle used to evaluate the code performance. The particle is made up by a core of silicate materials (grey spheres) surrounded by a mantle of amorphous carbon (black spheres). The particle size along its largest extension is 400 nm.

investigated the effects of adopting different field expansion orders, both on the execution times and on the stability of the final output. We compared our calculations with the results predicted for the equivalent spherical particle, i.e. a particle with the same mass and composition as the model presented in Fig. 2.1, but in the assumption of spherical geometry. This configuration, which is represented as a grey dashed line in the right panel of Fig. 2.2, is a useful test because the scattering problem in spherical geometry can be solved very quickly and, thanks to the *Rayleigh approximation*, we expect all physical solutions to converge to the same values at long wavelengths, while large deviations are predicted at shorter ones.

To evaluate the performance of the code and the stability of the results, we ran the calculation on two hardware configuration. The first one, named configuration A, is a single node from the GPU partition of computing facility at the Astronomical Observatory of Cagliari, featuring 64 CPU cores, 512 Gb RAM and 2 NVIDIA A40 GPUs with 45 Gb of dedicated RAM. The second one, named configuration B, is an ASUS Zenbook laptop computer, with
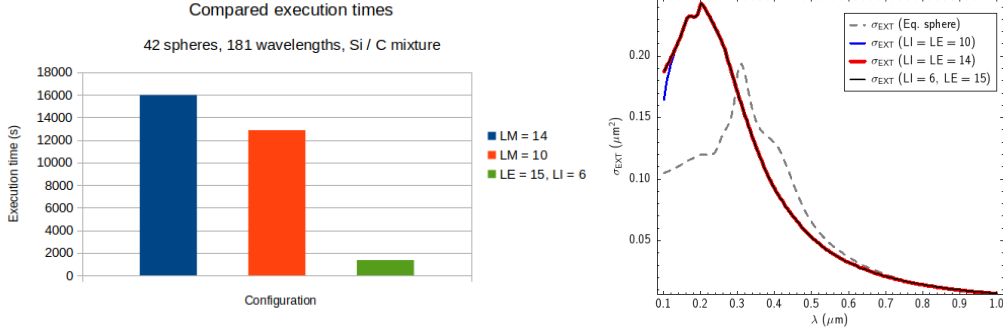
Figure 2.2: Calculation execution times (left panel) and particle extinction cross-section as a function of wavelength (right panel) for different problem configurations. The code performance is evaluated by means of the reduction in execution time and of the stability of the result with respect to the most computationally intensive task.

32 Gb of RAM, 20 CPU cores and a NVIDIA GeForce RTX 4060 Laptop GPU with 8 Gb of dedicated RAM.

The complete solution of the scattering problem with fixed maximum field expansion order $l_{MAX} = 14$ involves the inversion of a $[18816 \times 18816]$ elements complex matrix, which, when represented in double precision, requires 5.3 Gb of memory space to be stored. We, therefore, handled this case using the hardware configuration A to obtain our reference cross-sections, because it is the only one able to simultaneou8sly work on multiple wavelengths. However, thanks to the fact that the particle's spherical components are smaller than the whole particle and can therefore be solved with a lower field expansion order than the one used for the whole particle, we took advantage of the new ability to distinguish among the internal maximum field expansion order l_{INT} and the external maximum field expansion order l_{EXT} to investigate the effects of using different calculation orders on the execution time and the result stability. As it turns out, the size of the T-matrix is controlled by l_{INT}, while still preserving the accuracy of the solution using a higher value of l_{EXT} (Iatì et al. 2004). The results of our tests are illustrated in Fig. 2.2 and further detailed in Table 2.1.

As it is shown by Fig. 2.2, the choice of the field expansion order has a

11

Table 2.1: Execution times of the test model particle for different hardware and field expansion order configurations. The table columns contain, respectively, the system configuration, the size of the T-matrix, the number of simultaneously working MPI processes, the internal field maximum expansion order, the external field maximum expansion order and the calculation execution time in seconds.

CONFIG.	T-matrix size	N_{proc}	l_{INT}	l_{EXT}	EXEC. TIME
A	$[18816 \times 18816]$	8	14	14	15963.02 s
B	$[10080 \times 10080]$	3	10	10	12851.70 s
B	$[4032 \times 4032]$	5	6	15	1348.13 s

critical role. In first place, we can observe that all the computed models converge to the *Rayleigh* solution in the long wavelength regime, as expected. at shorter wavelengths there are large deviations from the behaviour predicted in spherical symmetry. Here we observe that, although all calculations converge to very similar solutions, lowering the maximum field expansion order l_{MAX} from 14 to 10 leads to a performance improvement at the cost of loss of precision at the shortest wavelengths (i.e. the wavelengths where the use of high maximum field expansion orders are more critically required). Conversely, the use of distinct maximum field expansion orders for the internal and external calculations implies a strong reduction of execution time, while fully recovering the precision achieved in the highest order calculation. Indeed, reducing the size of the T-matrix has the double effect of increasing the speed of the calculation and of allowing for more wavelengths to be solved simultaneously within the same memory occupation limits.

Chapter 3

Instructions for testing

3.1   Set up operations

Testing `NP_TMcode-M9.01` can be achieved through two main approaches.
The first is to obtain a local install, building the source code on the user's
own machine. The second is to use a pre-built image, choosing between
the `docker` and the `singularity` implementations. This section deals with
testing the code release on a local machine. Detailed instructions on how to
use pre-built images, instead, are given in § 3.4.1, later on.

The first operation needed to build and execute the code is to replicate
the project release from the *gitLab* repository. We assume that this step has
been already performed, since the present document is distributed as part
of the release bundle. The following steps, therefore, are just building and
execution.

3.1.1   Building the code

To build the code, the user needs a set of compilers and some libraries. The
recommended set up includes an up to date installation of the *GNU Compiler
Collection* (`gcc`), of the *GNU* `make` builder, a *FORTRAN* compiler (again,
the recommended option is to rely on *GNU*'s `gfortran`) and the *doxygen*
document manager. An optional dependency is a working LaTeXdistribution
with recommended package set-up, in order to build the full `PDF` documenta-
tion. If the aforementioned system requirements are met, building the code

just requires to go in the `build` folder:

```
~/np_tmcode> cd build
```

and then following the standard steps of configuring and compiling the code with:

```
~/build> ./configure
```

and:[1]

```
~/build> make
```

The compiler flags that were used to toggle optional features up to version `NP_TMcode-M8.03` have been replaced by a set of configuration options, which can be inspected by issuing:

```
~/build> ./configure --help
```

from the `build` folder.

The build process will take care of building all of the *FORTRAN* and *C++* codes, placing the relevant binary files in a directory structure based in the `build` folder, located at the same level of the `src` folder in the `np_tmcode` directory structure.

If desired, there is the additional option to build the code inline documentation by entering the `np_tmcode/doc/src` folder and issuing:

```
~/src> doxygen config.dox
```

This will generate a folder named `doc/build` under the the `np_tmcode` directory, with two additional sub-folders, respectively named `html` and `latex`. The `html` folder contains a browser formatted version of the inline code documentation, starting from a file named `index.html`. The `latex` folder, on the other hand, contains the instructions to build a PDF version of the documents, using LaTeX, by issuing the further `make` command (*after* the previous "doxygen config.dox" step):

```
~/src> make -C ../doc/build/latex
```

---

[1]In case the testing system supports multi-core architecture, it is possible to issue `make -j` instead of simply `make`. This will build the various code sections in parallel, reducing the compilation time.

14

3.2 Execution of the `sphere` case

Once the build process has been completed, the code is ready to be run on the available test cases. The configuration files and the expected *FORTRAN* output files are collected in a folder named `test_data` under `np_tmcode`. To run the *FORTRAN* code on the case of the single sphere, move to the `sphere` binary folder:

```
~/src> cd ../build/sphere
```

then run the *FORTRAN* configuration program:

```
~/sphere> ./edfb_sph
```

The `edfb_sph` program looks for the problem configuration data in a file named DEDFB[2] and located in the `test_data/sphere` folder, then it writes a formatted output file named OEDFB and a binary configuration file named TEDF in the current working directory. After checking for the existence of these files, the calculation of the scattering process, according to the *FOR-TRAN* implementation, can be executed by issuing:

```
~/sphere> ./sph
```

The `sph` program gets its input from a file named DSPH in the test data folder. Its execution provides essential feedback on the status of the calculation and writes the results in a binary file named TPPOAN and a text file named OSPH.

The calculation executed in *FORTRAN* can be replicated in *C++* simply by invoking:

```
~/sphere> ./np_sphere
```

The `np_sphere` program adopts the default behaviour of looking for the same input data as the *FORTRAN* code and writing the same type of output files, but appending a `c_` prefix to its output. Optionally, it can be run as:

```
~/sphere> ./np_sphere PATH_TO_DEDFB PATH_TO_DSPH OUTPUT_FOLDER
```

---

[2]Explanations on the structure of the input data file are given in the `README.md` file stored in the `test_data` folder.

15

to let it get input and write output other than the default behaviour.

After the execution of the *FORTRAN* and the *C++* versions, the final outcome can be compared by checking the contents of the `OSPH` and the `c_OSPH` files (see § 3.5 for suggestions on how to compare files).

3.3 Execution of the `cluster` case

Execution of the cluster calculation is very similar to the sphere case. The first step is to move to the `cluster` folder:

```
~/sphere> cd ../cluster
```

then run the *FORTRAN* configuration program:

```
~/cluster> ./edfb_clu
```

followed by the *FORTRAN* calculation:

```
~/cluster> ./clu
```

This command sequence will read the default cluster development case, which is a quick calculation of the scattering problem for 2 scales on a cluster made up by 4 spheres. As a result, the two binary files `TEDF` and `TPPOAN` will be written to the current working directory, together with the text file `OCLU` containing the results of the calculation.

In a similar way, the *C++* calculation that replicates this process can be executed. If the code was built without parallel optimization compiler flags, the command line to run the test will be just:

```
~/cluster> ./np_cluster
```

In case the code has been built with the `OpenMP` enabled, the recommended execution is:

```
~/cluster> OMP_NUM_THREADS=1 ./np_cluster
```

while a build using both OpenMP and MPI would be tested with:

```
~/cluster> OMP_NUM_THREADS=1 mpirun -n 1 ./np_cluster
```

`np_cluster` will look for the same configuration files used by `edfb_clu` and `clu`, namely the `DEDFB` and `DCLU` files stored in `test_data/cluster`, and write `c_`-prefixed output files. The results can be compared by looking in the `OCLU` and `c_OCLU` files (see § 3.5).

To further test the code, the `test_data/cluster` contains further testing cases, some of which include the *FORTRAN 66* pre-computed outputs, in text files named `OCLU`. The *C++* implementation of the code can directly handle these cases, by running:

```
~/cluster> ./np_cluster PATH_TO_DEDFB_XX PATH_TO_DCLU_XX \

           OUTPUT_PATH
```

for a serial implementation, or

```
~/cluster> OMP_NUM_THREADS=1,X mpirun -n Y ./np_cluster \

           PATH_TO_DEDFB_XX PATH_TO_DCLU_XX OUTPUT_PATH
```

where `X` and `Y` represent, respectively, the number of OpenMP threads and MPI processes that are intended for use at runtime.[3]

3.4   Testing with `docker/singularity`

Since `NPTMcode` is currently under development, the most straightforward way to obtain an executable version of the code is to download the latest release of the source code and build the program binaries locally. While the process should be fairly straightforward for any Linux based architecture offering the necessary requirements, building on other systems or finding the proper implementation of pre-requisites may not be equally easy. For this reason, the code is also distributed in the form of pre-assembled container

---

[3]The recommended way to use the current release of the code is to always set the `OMP_NUM_THREADS` environment variable in such a way that the first layer of OpenMP threading is configured to use 1 thread. In case of use of multiple threads at the first threading layer, OpenMP will start multiple threads on the wavelength range, which may result in resource consumption and performance degradation.

images, running under the `docker` and the `singularity` systems.[4] Refer to the installation instructions on the docker and singularity web sites to obtain and install one of these, if needed. The free versions are perfectly adequate for testing `NP_TMcode-M9.01`. The instructions to recreate local images are given in the `README.md` files in the `docker` and the `singularity` sub-folders of the `containers` directory. Both images include the standard `HDF5` tools, which can be used to inspect the binary output files in `HDF5` format, and compiled implementations of LAPACK and MAGMA.

3.4.1 Docker

A publicly accessible docker image can be obtained directly from Docker hub, under the name of `gmulas/np-tmcode-run`.

To test the `NP_TMcode-M9.01` in the `docker` image, one can start an interactive shell in the container image instance. This can be achieved either using the docker graphical user interface or using the command line, such as e. g.:

```
~/docker> docker run -it gmulas/np-tmcode-run:M9 /bin/bash
```

This will start an instance of the docker image, and open an interactive bourne shell within it. Then, one can go into the installation folder of `NP_TMcode` inside the container

```
root@74a5e7e7b79d:~# cd /usr/local/np-tmcode/build
```

and from there proceed as in Sections 3.2,3.3, and 3.5. Bear in mind that, unless a persistent docker volume was created (see `docker` reference documentation to do this), and mounted, on the docker image instance, whatever files are created inside the running instance are *lost* when the instance is closed, i. e., in the example above, upon exiting the shell. Whatever one wants to keep should be copied out of the running instance before closing it, e. g. using `docker cp` commands.

---

[4]Since the `docker` and the `singularity` images are large files, they are not included in the `NPTMcode` source distribution, due to space constraints, and they must be downloaded individually. This is automatically handled by `docker`, while instructions to work with `singularity` are given in § 3.4.2.

3.4.2   Singularity

To test the `NP_TMcode-M9.01` in the `singularity` image, one can make use of the feature of singularity that automatically mounts the user's home directory in the container image, and directly run the programs in the image. The image can be downloaded from the singularity image distribution site, which contains an implementation built on `NP_TMcode-M9.01`. Before attempting execution, the image file `np-tmcode-run.sif` *must* be placed in the `containers/singularity` folder of the `np_tmcode` project. The command line to run the image would then be:

```
~/singularity> singularity exec \

                 COMPLETE_PATH/np-tmcode-run.sif /bin/bash
```

where `COMPLETE_PATH` above is supposed to be the complete path to the singularity image file. This would open a shell within the image. One can therefore proceed as in Sections 3.2, 3.3, and 3.5, just running the executable files via the singularity image file, instead of directly from the host machine. The only caveat is that the *FORTRAN* binaries expect to read their input data from a `test_data` folder located two levels above the singularity execution directory.

3.5   Comparing results

The comparison of results for a realistic case is, in general, not straightforward. In comparing the output of *FORTRAN 66* and *C++*-based calculations, several effects may introduce artifacts that result in more or less significant differences. The output of the code, indeed, includes both unformatted binary files as well as formatted text files. Due to the facts that only the code is able to read its proprietary binary format and that the formatted output is a following step, it makes perfect sense to compare the results saved in formatted files, since they are derived from the binary ones.

However, comparison of formatted text files can still be a hard task, due to the large amount of information included in each file and to the possibility of observing *numeric noise*. This noise arises on values that are negligible

with respect to the typical orders of magnitude probed by a given level of approximation. A similar effect may also be observed when executing the same code on different hardware architectures. In order to make the task of comparing the output of the `sphere` and `cluster` calculations between the *FORTRAN* and the *C++* versions easier, NP_TMcode-M9.01 includes an executable *python3* script named `pycompare.py`. The scope of this script is to parse the formatted output files produced by the *FORTRAN* and the *C++* implementations, to check for the consistency of the file structures and to verify the coincidence of significant numeric values. This script is located in a folder named `src/scripts` and it can be invoked from there with the following syntax:

```
~/scripts> ./pycompare.py --ffile PATH_TO_FORTRAN_RESULT \

            --cfile PATH_TO_C++_RESULT
```

where the files to be passed as input are those named `OSPH` and `OCLU` by the *FORTRAN* code and `c_OSPH` and `c_OCLU` by the *C++* code. The script checks in the result files and it returns a result flag of 0 (OS definition of success), in case of consistent results, or some non-zero integer number (OS indication of failure) otherwise. The script also writes a summary of its diagnostics to the standard output, including the number of inconsistencies that were considered noisy values, warning values (i.e. values with a substantial difference but within a given tolerance threshold) and error values (i.e. values disagreeing by an above-threshold significant difference). If needed, the user may produce a detailed `html` log of the comparison by invoking:

```
~/scripts> ./pycompare.py --ffile PATH_TO_FORTRAN_RESULT \

            --cfile PATH_TO_C++_RESULT \

            --html[=HTML_LOG_NAME]
```

where the part in square brackets is an optional name of the log file (which, if not specified, defaults to `pycompare.html`). Invoking the script without arguments or with the `--help` option will result in the script printing a detailed help screen on terminal and then exit with success code.

20

Bibliography

Borghese, F., Denti, P., & Saija, R. 2007, Scattering from Model Nonspherical Particles (Berlin: Springer)

Draine, B. T. & Flatau, P. J. 1994, J. Opt. Soc. Am. A, 11, 1491

Draine, B. T. & Lee, H. M. 1984, *The Astrophysical Journal*, 285, 89

Iatì, M. A., Giusto, A., Saija, R., et al. 2004, *The Astrophysical Journal*, 615, 286

Palik, E. D. 1991, "Handbook of optical constants of solids II" (Elsevier)