High-level interface to *T*-matrix scattering calculations: architecture, capabilities and limitations

Jussi Leinonen

Finnish Meteorological Institute, Earth Observation, P.O. Box 503, FI-00101 Helsinki, Finland

jussi.leinonen@fmi.fi

Abstract: The PyTMatrix package was designed with the objective of providing a simple, extensible interface to *T*-Matrix electromagnetic scattering calculations performed using an extensively validated numerical core. The interface, implemented in the Python programming language, facilitates automation of the calculations and further analysis of the results through direct integration of both the inputs and the outputs of the calculations to numerical analysis software. This article describes the architecture and design of the package, illustrating how the concepts in the physics of electromagnetic scattering are mapped into data and code models in the computer software. The resulting capabilities and their consequences for the usability and performance of the package are explored.

OCIS codes: (290.5850) Scattering, particles; (000.4430) Numerical approximation and analysis; (010.1310) Atmospheric scattering; (280.5600) Radar.

References

1. P. C. Waterman, "Matrix formulation of electromagnetic scattering," Proc. IEEE **53**, 805–812 (1965).
2. M. I. Mishchenko, L. D. Travis, and D. W. Mackowski, "*T*-matrix computations of light scattering by nonspherical particles: A review," J. Quant. Spectrosc. Radiat. Transfer **55**, 535–575 (1996).
3. M. I. Mishchenko, L. D. Travis, and A. Macke, "*T*-matrix method and its applications," in *Light Scattering by Nonspherical Particles*, M. I. Mishchenko, J. W. Hovenier, and L. D. Travis, eds. (Academic, 2000), chap. 6.
4. A. Gogoi, P. Rajkhowa, A. Choudhury, and G. A. Ahmed, "Development of TUSCAT: A software for light scattering studies on spherical, spheroidal and cylindrical particles," J. Quant. Spectrosc. Radiat. Transfer **112**, 2713–2721 (2011).
5. J. Hellmers, K. Heiken, E. Foken, J. Thomaschewski, and T. Wriedt, "Customizable web service interface for light scattering simulation programs," J. Quant. Spectrosc. Radiat. Transfer **113**, 2243–2250 (2012).
6. J. Fung, R. W. Perry, T. G. Dimiduk, and V. N. Manoharan, "Imaging multiple colloidal particles by fitting electromagnetic scattering solutions to digital holograms," J. Quant. Spectrosc. Radiat. Transfer **113**, 2482–2489 (2012).
7. J. Leinonen, "Python code for T-matrix scattering calculations," https://github.com/jleinonen/pytmatrix.
8. E. Jones, T. Oliphant, P. Peterson, and others, "SciPy: Open source scientific tools for Python," http://www.scipy.org/ (2001–).
9. T. E. Oliphant, "Python for scientific computing," Comput. Sci. Eng. **9**, 10–20 (2007).
10. M. I. Mishchenko and L. D. Travis, "Capabilities and limitations of a current FORTRAN implementation of the T-matrix method for randomly oriented, rotationally symmetric scatterers," J. Quant. Spectrosc. Radiat. Transfer **60**, 309–324 (1998).
11. K. Aydın, "Centimeter and millimeter wave scattering from hydrometeors," in *Light Scattering by Nonspherical Particles*, M. I. Mishchenko, J. W. Hovenier, and L. D. Travis, eds. (Academic, 2000), chap. 16.
12. W. Gautschi, "Algorithm 726: ORTHPOL–a package of routines for generating orthogonal polynomials and Gauss-type quadrature rules," ACM Trans. Math. Software **20**, 21–62 (1994).

#200830 - $15.00 USD Received 7 Nov 2013; revised 20 Dec 2013; accepted 4 Jan 2014; published 16 Jan 2014

13. A. D. Fernandes and W. R. Atchley, "Gaussian quadrature formulae for arbitrary positive measures," Evol. Bioinform. Online **2**, 251–259 (2006).
14. J. Testud, S. Oury, R. A. Black, P. Amayenc, and X. Dou, "The concept of "normalized" distribution to describe raindrop spectra: A tool for cloud physics and cloud remote sensing," J. Appl. Meteorol. **40**, 1118–1140 (2001).
15. J. Leinonen, D. Moisseev, M. Leskinen, and W. Petersen, "A climatology of disdrometer measurements of rainfall in Finland over five years with implications for global radar observations," J. Appl. Meteorol. Climatol. **51**, 392–404 (2012).
16. H. C. van de Hulst, *Light Scattering by Small Particles* (John Wiley, 1957).
17. V. N. Bringi and V. Chandrasekar, *Polarimetric Doppler weather radar: principles and applications* (Cambridge University, 2001).
18. J. Leinonen, "PyTMatrix K_{dp} example," https://github.com/jleinonen/pytmatrix/wiki/PyTMatrix-Kdp-example.

1. Introduction

The *T*-matrix method [1] has been widely adopted for the numerical modeling of electromagnetic and light scattering by nonspherical particles that are in the resonance region, or comparable in size to the wavelength of the incident radiation. It is used in diverse applications such as atmospheric optics, light and microwave remote sensing, astronomy, the optics of colloids, particle size determination and biomedical applications (see [2, 3] and references thereof). Due to its numerically accurate results and reasonable computational requirements, it is also frequently used as reference method for evaluating and validating other computational scattering methods.

Although the computational cost of the *T*-matrix method is lower than that of many numerical scattering methods, it is still nontrivial, which generally justifies implementing it using highly optimizable, low-level programming languages such as FORTRAN. This has led to the development of standalone scattering codes which, although fast, can be cumbersome to use, as they have to be reconfigured for each new set of particle parameters by modifying either a configuration file or the source code itself, the latter approach requiring one to recompile the code after modifications. Other researchers (e.g. [4, 5, 6]) have built software with graphical user interfaces (GUIs) on top of numerical code, improving the usability, but using a GUI may come at a cost of lessened ability of the user to automate computations.

It it common for researchers to have to build new, specialized computer code for each new study they undertake. Computer-assisted analysis of the results of the numerical code is also usually a necessary step in the process of producing scientific results. However, neither low-level computer codes nor GUI software lend themselves well to rapid development of application-specific code or to the integration of the results into further analysis using standard data analysis and visualization software. For truly seamless integration to be possible, the inputs have to be directly modifiable and the results directly accessible from within the analysis software itself, without intermediate steps taken by the user. Such integration enables straightforward access to the scattering code for the user of the analysis software.

The PyTMatrix software package was designed to provide these capabilities for the modeling of scattering with the *T*-matrix method. It was built for the Python programming language using the NumPy/SciPy numerical analysis framework, whose popularity is increasing rapidly. The architecture of PyTMatrix was built to reflect the physics of the scattering problem, separating the general, low-level matrix representation of scattering from the specific application. This resulted in a modular, extensible framework that provides tools for common use cases while allowing the users to adapt them to their own needs with modest effort. Having these high-level tools available as open source code enables the users to submit fixes and improvements to the code; these changes are propagated to the other users as they are integrated into the codebase, thus reducing both the chance of errors and duplication of effort.

This article describes the design of PyTMatrix and the reasoning behind various architectural choices, and explores of the consequences of these choices for the usability of the package. For

#200830 - $15.00 USD Received 7 Nov 2013; revised 20 Dec 2013; accepted 4 Jan 2014; published 16 Jan 2014

a more practical user guide to the installation and use of the package, the reader is referred to the PyTMatrix online documentation [7].

2. Architecture

2.1. Overview

Motivated by the considerations presented in section 1, the objectives of the PyTMatrix package are to:

1. Provide reliable, high-performance *T*-matrix scattering computations.
2. Integrate the results directly into a larger, generic analysis framework.
3. Allow the users to adapt the package to their specific needs with minimal additions.

The design of PyTMatrix approaches these objectives by mapping the structure of the physical problem at hand to the structure of the code used to solve it. Thus, conceptually separate parts of the physics of the problem were likewise separated in the architecture of the package. In many cases, adopting this approach was found to lead naturally to intuitive ways to use and extend the package.

PyTMatrix is built on the NumPy and SciPy numerical packages [8, 9] for Python, which provide fast manipulation of numerical arrays (NumPy) and scientific analysis methods (SciPy). Combined with the matplotlib visualization tools and the IPython interactive shell (among others), these can be used as an analysis environment comparable in functionality to analysis frameworks such as MATLAB. PyTMatrix allows access direct access from Python to the *T*-Matrix computations via classes and functions that correspond to scattering concepts such as the particle properties, orientation distributions, particle size distributions, and application-specific parameters.

2.2. Computation of the T matrix

To ensure the performance and reliability of the low-level numerics, the widely used and extensively validated FORTRAN code by Mishchenko and Travis [10] was, with permission, adopted as the numerical core of the package. This code was modified in order to allow the properties of the scatterer and the incident radiation to be passed directly as function arguments rather than by modifying the source code, and by returning the results via output arguments instead of printed output.

An distinct advantage of the *T*-matrix formalism is that after computing the *T* matrix, the scattering properties can be derived at any particle orientation and scattering geometry (*i.e.* the directions of the incident and scattered beams) with a relatively small amount of computation. Reflecting this, two functions that require separate calls are implemented in the modified code: one (`CALCTMAT`) to compute the *T* matrix, and another (`CALCAMPL`) to compute the amplitude scattering matrix **S** and the phase matrix **Z** for a given orientation and scattering geometry when the *T* matrix is known; **S** and **Z** are defined as matrices relating the scattered and incident electric fields **E** and Stokes vectors **I**, respectively [3]:

$$\mathbf{E}_{\mathrm{sca}} = \frac{\exp(-ikR)}{R}\mathbf{S}\mathbf{E}_{\mathrm{inc}} \tag{1}$$

$$\mathbf{I}_{\mathrm{sca}} = \frac{1}{R^2}\mathbf{Z}\mathbf{I}_{\mathrm{inc}} \tag{2}$$

where k is the wavenumber and R is the distance; the relations hold in the far field where R is much larger than the wavelength and the particle size. `CALCAMPL` can be used several times

#200830 - $15.00 USD Received 7 Nov 2013; revised 20 Dec 2013; accepted 4 Jan 2014; published 16 Jan 2014

with different parameters after a call to `CALCTMAT`. The F2PY interface is used to allow calls to these functions from Python.

2.3. *Representation of the scatterer*

The properties of the scatterer are encapsulated in an instance of the Python class `Scatterer`. This class includes the parameters of the scatterer as data members. Its main user-visible functionality consists of functions that are used to compute the **S** and **Z** matrices. These call the low-level functions described, but abstract away the requirement to separately compute the T matrix.

The T matrix is recomputed only when necessary by implementing a lazy evaluation and caching scheme where a stored version of the T matrix is reused when possible. Changes to the incident wavelength, the size and shape of the particle, or the complex refractive index force a recalculation of the T matrix, while changes to the particle orientation or scattering geometry only trigger a recalculation of the **S** and **Z** matrices for that particular geometry. If no parameters are changed between calls to these functions, stored versions of these matrices are also reused. These techniques improve both the performance and the ease of implementation of many of the higher-level features.

2.4. *Orientation averaging*

It is often necessary to calculate the scattering properties of particles at varying orientations, and especially in atmospheric applications, the orientation distribution is often not uniform. Thus, averaging over different orientation distributions is implemented in PyTMatrix.

The calculation of the amplitude scattering matrix **S** and the phase matrix **Z** can be set to use functions that correspond to different types of orientation averaging. The user chooses one of these functions, which, in turn, computes the single-orientation scattering for several orientations, as appropriate for the chosen orientation averaging scheme. The orientation averaging is implemented at the **S**/**Z**-matrix level rather that at the level of individual scattering properties, as most scattering properties of an ensemble of many particles can generally be computed from the orientation-averaged version of one of these matrices. For example, the ensemble-averaged backscattering cross section $\langle\sigma_{\mathrm{vv}}\rangle$ and extinction cross section $\langle\sigma_{\mathrm{ev}}\rangle$ at vertical polarization can be computed from the elements of the ensemble-averaged matrices $\langle\mathbf{S}\rangle$ and $\langle\mathbf{Z}\rangle$ as [11]

$$\langle\sigma_{\mathrm{vv}}\rangle = 2\pi\langle Z_{11}+Z_{12}+Z_{21}+Z_{22}\rangle \tag{3}$$

$$\langle\sigma_{\mathrm{ev}}\rangle = \frac{4\pi}{k}\,\mathrm{Im}\left[\langle S_{11}(\mathbf{n},\mathbf{n})\rangle\right]. \tag{4}$$

The current version includes two orientation-averaging schemes (in addition to the option of using a single, fixed orientation): fixed-point numerical integration and adaptive integration. The averaging schemes accept an arbitrary Python function representing the orientation distribution $p(\beta)$ as a function of the vertical alignment of the symmetry axis (the Euler angle β); the current implementation assumes a uniform distribution for the azimuth angle (the Euler angle α). In the fixed-point scheme, the integral over the β angle is estimated as

$$\int_0^\pi p(\beta)\mathbf{Z}(\alpha,\beta)\,d\beta \approx \sum_{i=1}^{N} w_i\mathbf{Z}(\alpha,\beta_i) \tag{5}$$

(and analogously for **S**) where the integration points β_i and weights w_i are obtained from Gaussian quadrature rules using the angular distribution as the weighting function. The calculation algorithm for the points and weights is numerically stable and can use an arbitrary positive weighting function, following the techniques of [12, 13]. The adaptive integration method,

#200830 - $15.00 USD Received 7 Nov 2013; revised 20 Dec 2013; accepted 4 Jan 2014; published 16 Jan 2014

which is much slower but more precise than the fixed-point method, uses the adaptive integration over two variables function (`dblquad`) found in SciPy.

Users can easily write their own orientation distribution functions, but a uniformly random distribution, as well as a Gaussian distribution around vertical orientation with a given standard deviation, are included in the package.

2.5. Particle size distributions

Simulating the scattering from an ensemble of particles of varying sizes is conceptually similar to that of scattering from different orientations. In PyTMatrix, the integration over particle size distributions (PSDs) is implemented similarly to the orientation averaging: a user specifies that orientation averaging is to be used by creating a `PSDIntegrator` object from the `psd` module and attaching it to the `Scatterer` object. The user then specifies a particle size distribution (PSD), and can access the results using the same functions as for the fixed-size calculations.

PSD integration can be relatively time-consuming because unlike with orientation averaging, the T matrix must be recomputed for every size considered. For this reason, and because in a typical use case the user wants to calculate scattering from the same type of particles for several different PSDs, `PSDIntegrator` precalculates the **S** and **Z** matrices at a predetermined number of points (default 1024) and stores them in lookup tables as 3D arrays. It then averages over a PSD by weighting the lookup table by the distribution values and integrating over all sizes using the fixed-point trapezoidal integration routine found in SciPy.

Some commonly used PSDs, such as the normalized gamma model popular in atmospheric sciences [14], are included in the `psd` module. Similar to orientation averaging, the user can easily use their own PSD functions simply by creating callable Python objects (typically functions) that take the particle diameter as an argument and return the corresponding PSD value. These can be any nonnegative functions $N(D)$ defined over the diameter interval $0 < D \leq D_{\max}$, where $D_{\max}$ is the maximum particle diameter considered, and which yield the particle number concentration $N = \int_0^{D_{\max}} N(D)\, dD$. By adding custom PSD functions it simple to, for example, integrate over measured size distributions as explained in [15]. Size-dependent particle axis ratios and complex refractive indices can also be implemented by specifying functions that express these.

2.6. Testing

Testing during the development and after the installation of the PyTMatrix package is implemented using the standard Python unit testing framework. It compares the results of calculations from the package to reference values in order to make sure that the two are in agreement. This can be used to verify a functional installation as well as to track errors possibly introduced to the package by code changes. As a computational physics package, PyTMatrix also includes some tests for the physical correctness of the results. For instance, the current development version verifies that for a lossless particle, the numerically calculated scattering cross section agrees with the extinction cross section calculated from the optical theorem, and that at small size parameters, the results agree with the Rayleigh approximation [16].

2.7. Application-specific modules

With the framework laid out by the `Scatterer` interface, it is straightforward to use the retrieved **S** and **Z** matrices to compute scattering properties needed in various applications. PyTMatrix currently includes two modules that contain functions for these purposes.

The functions of the `scatter` module can be used to compute quantities that are of general interest in many scattering applications, such as the scattering and extinction cross sections and

#200830 - $15.00 USD Received 7 Nov 2013; revised 20 Dec 2013; accepted 4 Jan 2014; published 16 Jan 2014

the asymmetry parameter. These can be computed for polarized light, allowing their use also in, e.g., lidar and radar applications. The scattering cross section and the asymmetry parameter are computed using numerical integration, while the extinction cross section is obtained more simply, and much faster, using the optical theorem.

Owing to the development of PyTMatrix originally for the modeling of weather radar signals, a `radar` module is also included. It facilitates the computation of quantities used in polarimetric radars, including radar reflectivity, differential reflectivity and specific differential phase [11, 17].

3. Limitations

The PyTMatrix package directly uses the results of the original low-level *T*-matrix code by Mishchenko and Travis. Therefore, the main limitations of the current version (0.2.0) of PyTMatrix are the same as those of the original code: the target shapes are limited to spheroids and cylinders (the PyTMatrix layer presently lacks support for Chebyshev particles), and large size parameters as well as very high or low aspect ratios present convergence problems; these are detailed in [10]. Users of the code are encouraged to familiarize themselves with these limitations in order to ensure the validity of their results. Furthermore, some *T*-matrix codes also offer the ability to average analytically over different orientations, but the fixed-orientation code used as the backend does not offer this capability, and thus only numerical averaging is available.

A few usability-related limitations remain in the Python interface. Most significantly, in some situations, a failure of the Fortran numerical code to converge can cause a crash that results in a program abort, stopping the execution of not only the numerical code but also the Python environment. Minor issues include incomplete support for the size-distribution integration of the scattering cross section and the asymmetry parameter.

4. Summary

The PyTMatrix package was created to simplify the calculation of electromagnetic scattering parameters of particles using the *T*-matrix method and to streamline the integration of the scattering calculations into the workflow of numerical analysis tools. The result is an interface that utilizes the numerical properties of the *T*-matrix method for accuracy and performance, while hiding the details from the user. It is straightforward and transparent to use orientation averaging, integration over particle size distributions, and to calculate scattering parameters more specific than merely the amplitude and phase matrices. An example calculation in Python code that replicates Fig. 7.7 of [17], demonstrating many of these features, is shown in [18], along with other examples in the documentation.

PyTMatrix can be freely used and modified under the open-source MIT License and is available, with documentation and examples, for download online [7] or for installation from the Python Package Index (PyPI).

Acknowledgments

The research leading to this article was supported by the Academy of Finland (grant 255718) and the Finnish Funding Agency for Technology and Innovation (Tekes; grant 3155/31/2009). The author would like to thank Dr. M. I. Mishchenko for the permission to include his *T*-matrix code as a part of PyTMatrix under the MIT license. Thanks are also in order to the early users of PyTMatrix who submitted feedback and bug reports, in particular Dr. D. Moisseev, Dr. J. Richard, Dr. J. Hardin and K. Mühlbauer. The NumPy and SciPy communities are gratefully acknowledged for making these tools freely available.