pyAT: A PYTHON BUILD OF ACCELERATOR TOOLBOX

W. Rogers, Diamond Light Source, Oxfordshire, UK
N. Carmignani, L. Farvacque, B. Nash, ESRF, Grenoble, France

Abstract

Accelerator Toolbox [1] (AT) is a particle accelerator modelling tool originally written in Matlab. It is used at many accelerator facilities, particularly synchrotron light sources, as an on-line model and is also used for off-line beam dynamics studies. For speed of execution, the tracking engine of AT was written in C and compiled for use in Matlab. The C-based implementation allowed re-use of the tracking engine compiled against the core Python libraries to create a Python version of AT. For additional purposes of speed, the C interface to the integration routines has been modified allowing equal speeds for both Matlab and Python interfaces, with an increase in speed relative to the original Matlab version. This paper describes the adaptation process, including adapting the Matlab build, creating the Python build and laying the foundations for the additional Python library implementation. Speed benchmarks are included with comparison to other tracking codes Elegant and MADX.

INTRODUCTION

Particle tracking is an important part of accelerator physics, used for all kinds of accelerator design and configuration. Accelerator Toolbox is a particle tracking code originally written by A. Terebilo at SLAC. AT is now used at several facilities and gives reliable tracking results.

Matlab is a proprietary programming language widely used in physics. Its strengths include the cross-platform interactive environment including many plotting tools, and a comprehensive set of built-in mathematical functions. However, there are some disadvantages to using the Matlab environment: Matlab can be verbose, particularly in such matters as handling arguments to functions, and designing large and robust Matlab applications can be difficult, due to the flat namespace and limited object-orientation. In addition, Matlab requires a commercial licence.

Python is a free open-source programming language widely used in science and industry. Although it can be more difficult to get started using Python than Matlab, it is more suited to building large applications. There are a number of third-party libraries available that provide most of the functionality provided by Matlab, notably including Numpy, Scipy and Matplotlib. However, none of the tracking codes commonly used at synchrotron light sources are easy to use with Python.

Much of the AT code is written in the Matlab language, but computationally intensive routines were implemented in C and compiled into Matlab MEX files. This design decision to write low-level routines in C allows, with some adjustments, compiling the code into a Python extension in a very similar way to Matlab. Doing this gives the same physics engine to Python code as the one used by the Matlab version of AT. Using this engine, a Python library may be provided that is compatible with AT but has the benefits described above.

C IMPLEMENTATION

Most of the C code in AT is contained in pass methods: functions that numerically calculate the effect in 6-D phase-space on a particle passing through a particular element in a lattice. The Matlab function `atpass` is then provided by a top-level C file that interprets the Matlab structures, executes the compiled code on the appropriate arrays, and converts back into Matlab structures to return.

A very similar structure is now implemented in Python. The original authors of AT were careful to separate the pure physics code from the code that handles interpreting the data in Matlab. This decision greatly helped in using the same code for a separate Python implementation. Each pass method was adapted to allow calling from either a Matlab implementation or a Python implementation depending on flags passed during compilation. There is one new C file that interprets and returns Python objects in order to provide the `atpass` function in Python.

Python has a widely-used numerical array library called Numpy. Since the underlying arrays in Numpy are C-arrays, Numpy was a natural choice to handle the arrays for passing data into and out of the AT C routines.

NUMERICAL INTEGRITY

Unit Tests

A number of unit tests compare the results of Python AT tracking with numbers manually extracted from Matlab. This serves to check that the tracking though basic elements matches what would be expected and would catch any errors introduced by later changes to the code. However, extracting numerical values into tests is tedious and any intentional changes to the physics implementation would cause the tests to fail, so this approach is not used exhaustively.

Comparison Tests

In newer versions of Matlab, a mechanism is provided that allows calling Matlab directly from Python. This allows a second way of comparing results: run the equivalent functions from within Python and then again from Matlab, and compare the output directly. This method is different in that a change to underlying physics would not cause the tests to fail. However, it requires a valid Matlab licence for the tests to be run.

To demonstrate the equivalence of results, ten particles with different initial conditions were tracked through the ESRF EBS lattice [2] for 1000 turns in both AT and pyAT.

ISBN 978-3-95450-182-3

To compare these, `x` and `x'` are plotted against each other in Fig. 1 and Fig. 2. These plots demonstrate visually that the numerical results from AT and pyAT are identical.

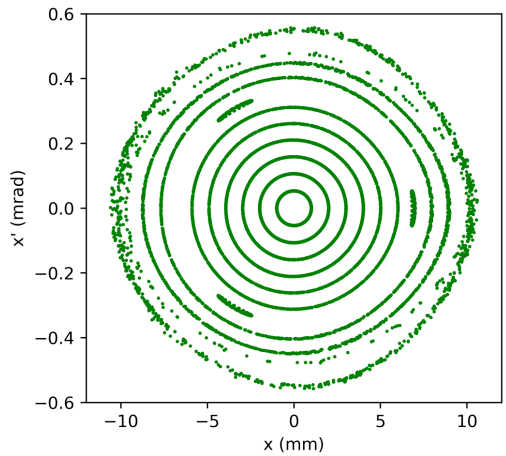

Figure 1: Ten particles with different initial conditions tracked using pyAT and plotted using the Matplotlib Python library.

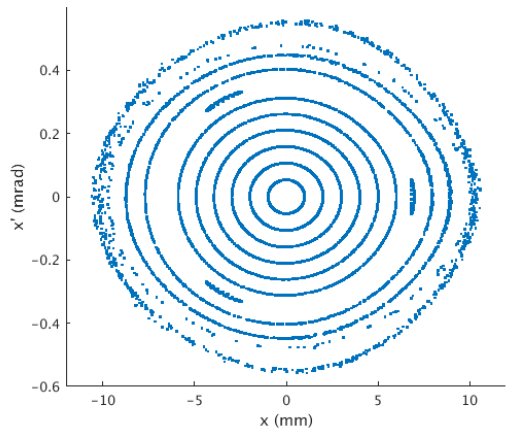

Figure 2: Ten particles with different initial conditions tracked using AT and plotted using Matlab's plotting tools.

COMPUTATIONAL SPEED

In order to track in AT, the elements of the lattice need to be stored so that the pass methods can be used to track through the elements. In the Matlab version of AT, the elements were stored in Matlab structures, and the Mex interface was used to access the element parameters. When the same approach was implemented in Python, it was found to be substantially slower than the Matlab version. To solve this problem, we rewrote part of the interface code in each of the integrators so that the lattice structures are now stored directly using the C code. This resulted in a modest speed-up of the Matlab version of AT, and allows for the Python and Matlab versions to be equally fast.

Benchmarking

It is possible to compare the speed of execution of different tracking codes using the same lattice. Since different codes have different merits, this is not intended to determine the best code for any application, but it does give an indication of how long execution may take for particular studies.

The ESRF EBS lattice [2] was converted for use in AT, pyAT, MADX version 5.00.00 [3] and Elegant version 32.0.0 [4], and a single particle was tracked for a specified number of turns. Each magnet was divided into 30 integration steps; tracking was done without synchrotron radiation and using RF cavities. Each test was executed five times on a Dell Optiplex 9020 PC with an Intel®Core™ i7-4770S CPU. The time of execution is shown in Fig. 1. It is clear that the newer implementation of AT is faster than the original implementation. Although start-up is slower for pyAT than for AT, they converge to the same times of execution for a large number of turns (see Fig. 3 and Table 1).

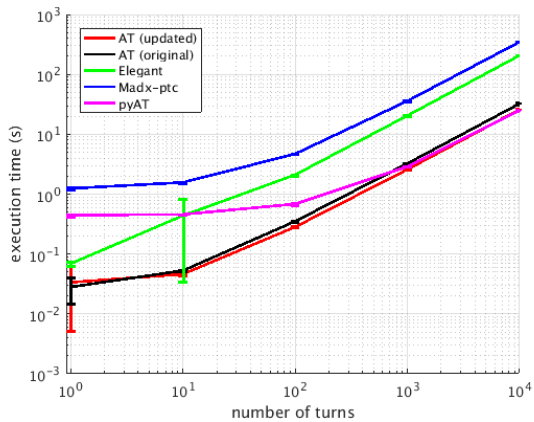

Figure 3: Time of execution for different codes.

Table 1: Execution Times to Track one Particle for 10^4 Turns

Code	Execution time
AT (updated)	25.95 ± 0.29 s
AT (original)	32.50 ± 0.97 s
Elegant	207.41 ± 3.44 s
Madx-ptc	348.65 ± 3.14 s
pyAT	25.26 ± 0.52 s

PYTHON IMPLEMENTATION

Although the core tracking engine of AT can now be used from both Matlab and Python, much of the AT code is written in Matlab. This part cannot be used in the Python implementation, and so Python versions of the AT functions and infrastructure are required. Currently implemented are a number of Element classes of different types; any sequence of these elements is understood as a lattice by the C library and may be used for tracking. The physics functions construct results from the tracking; those written so far include

ISBN 978-3-95450-182-3

finding the closed orbit, computation of the one turn map matrix and Twiss functions. An important function still missing is the calculation of the equilibrium beam-sizes using the Ohmi-Envelope formalism; this requires computing the diffusion matrix in each element. Constructing a useful subset of the AT physics library is the next piece of work.

As with any tracking code, a lattice must be created or imported before tracking. So far utilities have been written to load a saved Matlab file from Python (`load_mat.py`) and to create and save Python objects directly from Matlab (`atwritepy.m`).

USING pyAT

The aim is to provide pyAT for anyone who wants to use it via the command `pip install pyat`. This requires uploading a built version of the pyAT library to the online repository PyPI [5]. The use of compiled C code in this Python extension means that this would involve building the extension for different operating systems and CPU architectures.

The AT code is hosted on Sourceforge [6]. pyAT collaboration is currently taking place on Github [7], and contributions from new collaborators are very welcome.

CONCLUSIONS

With the help of the AT community it has proved possible to adapt the core of the AT library for use as a Python library. This allows using a well-tested tracking engine in Python applications, and it has been demonstrated that the results are numerically identical to the Matlab implementation of AT. Work is continuing converting Matlab physics code into the equivalent Python functions, with the aim of providing a complete and useful accelerator modelling tool in Python.

REFERENCES

[1] A. Terebilo, "Accelerator Toolbox for MATLAB", SLAC-PUB-8732, 2001.

[2] J. C. Biasci et al., "A low emittance lattice for the ESRF", Synchrotron Radiation News, vol. 27, Iss.6, 2014.

[3] MAD, `http://mad.web.cern.ch/mad/`

[4] M. Borland, "elegant: A Flexible SDDS-Compliant Code for Accelerator Simulation," Advanced Photon Source LS-287, September 2000.

[5] PyPI, `https://pypi.python.org/pypi`

[6] AT on Sourceforge, `https://sourceforge.net/projects/atcollab/`

[7] pyAT on Github, `https://github.com/willrogers/at`

ISBN 978-3-95450-182-3