ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPL031

STATUS AND RECENT DEVELOPMENTS OF PYTHON ACCELERATOR TOOLBOX

S. White, L. R. Carver, L. Farvacque, S. M. Liuzzo, ESRF, Grenoble, France

Abstract

The Accelerator Toolbox (AT) is a multipurpose tracking and lattice design code relying on a C tracking engine. Its MATLAB interface is widely used in the light source community for beam dynamics simulations and can be integrated in control systems through the MATLAB Middle Layer. In recent years major effort was made to develop a Python interface to AT: pyAT. In this framework, several features were added to pyAT, in particular, the introductions of the 6D optics dynamic aperture and lifetime calculation, single and multi-bunch collective effects simulations and parallel tracking capabilities. A Python ring simulator was also developed at ESRF based on pyAT for offline modeling of the accelerator control system. Following a presentation of the structure and main features of AT, an overview of these recent developments is provided.

INTRODUCTION

The Accelerator Toolbox [1, 2] was originally developed at the Stanford Linear Accelerator Center (SLAC) as a collection of tools to perform beam dynamics simulations in the MATLAB framework [3]. AT applications were later extended to accelerator controls with the introduction of the MATLAB Middle Layer (MML) [4]. Following several developments and increasing number of contributors [5] AT has evolved into a fully open source international collaboration project [6]. Although it is well established and widely used in science applications, MATLAB is a proprietary software with its own programming language and it was found desirable to provide an AT implementation in a free open-source language. Users would then be able to run AT without having to purchase MATLAB. Python benefits from a large user base in the scientific community and the implementation of a large number of third-party libraries that provide most of the functionalities available in MATLAB. It also allows to adopt an object-oriented approach and to build simple interfaces with other Python codes. It was therefore selected as the most promising language for this project. A new Python interface, pyAT [7], was therefore developed and is now matching, or even surpassing, most of the MATLAB AT features.

IMPLEMENTATION

The MATLAB AT implementation is organized as follows: the tracking engine and most of the computationally intensive routines are implemented in C to optimize run time. All the high level functions to derive characteristic lattice quantities such as optics or closed orbit from the tracking data are implemented in MATLAB. The C code is organized in so-called pass methods, associated with lattices elements such as bending magnets or quadrupoles. They transform the particles 6D coordinates in-place following the lattice magnetic layout. The pass methods were entirely re-used in pyAT with some adaptations to interpret Python objects. In this way, the tracking results are strictly identical and only the top level functions and user interface have to be rewritten. This also makes any new C pass method immediately available to both the MATLAB and Python implementations of AT. More details on the implementation and comparisons between both interfaces can be found in Ref. [7].

PARALLEL TRACKING

Parallel tracking functionalities are implemented in pyAT using 3 different methods: OpenMP [8], the Python multiprocessing library and MPI [9]. OpenMP is implemented at the pass method level where directives were added to execute the loop over the particles in parallel. This method is activated when compiling the C code. The Python multiprocessing method is provided through a dedicated tracking function, it does not require any specific compilation but does not allow to pass information between processes. It is therefore well suited for intensive single particle simulations such as dynamic aperture or lifetime calculations but cannot be used for collective effects. Parallel collective effects simulations are the most demanding as they generally require a very large number of processes running on several hosts. This is not possible to implement with either of the 2 methods cited above and an MPI implementation of the C collective effects pass methods was developed. It is also activated at compile time. These 3 methods provide sufficient flexibility to allow for parallel computation of most of the typical pyAT tracking simulations and the user can select either of these depending on the physics process to model.

Figure 1 shows the speed-up factor for the 3 methods. A simple, perfectly parallel, problem where particles do not interact with each other is considered in this case. However, for MPI, the results are gathered together to model realistic usage. On the architecture used for this test (3.4 MHz AMD EPYC 7542 32 core processors running on ubuntu), OpenMP is less performant and seems to be more appropriate for calculations involving few cores. It is nevertheless the only method that can be used in the MATLAB framework. For the other 2 methods and for large number of particles per cores the speed-up is linear with the number of cores as expected. Nevertheless, for smaller number of particles per core the python multiprocessing suffers from a visible overhead. It should be noted that for heavy collective effect simulations involving large memory exchange between processes this linear behavior will not hold. Similarly, for simulations requiring very little number of particles over one or few turns

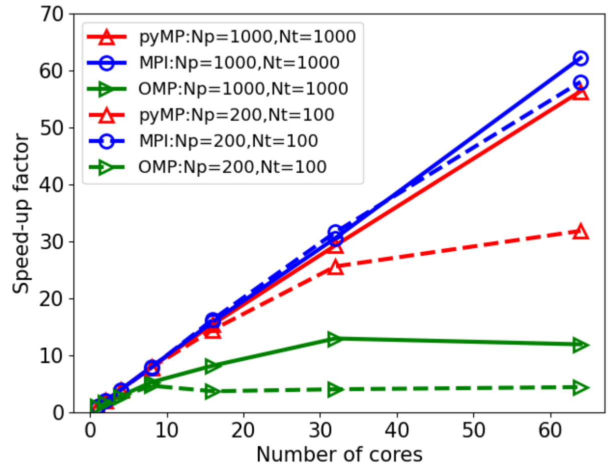

Figure 1: Multi-particle tracking speed-up factor for the 3 parallelization methods provided in pyAT. pyMP stands for Python multiprocessing, OMP for OpenMP, Np is the number of particles and Nt is the number of turns.

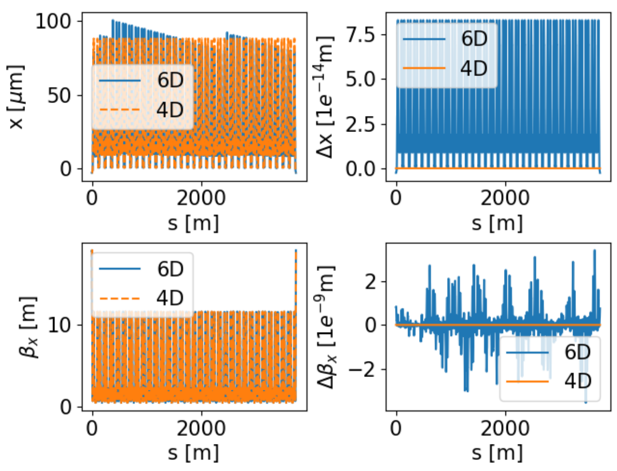

Figure 2: 4D without radiation and 6D with radiation horizontal closed orbit and β-functions with $\delta_p = 0.1\%$ (left column) and the difference between the Python and MATLAB implementations (right column) for the HMBA lattice.

(such as closed orbit or optics calculations) the initialization overhead become detrimental and parallel tracking may not give any improvement.

SINGLE PARTICLE DYNAMICS

Closed orbit and linear optics calculations are available in 6D including radiation damping effects. For 6D lattices, the particle momentum offset is determined by the deviation of the Radio-Frequency (RF) cavities frequency with respect to the nominal frequency given by the length of the machine. All calculations are available for circular machines and transfer lines.

Figure 2 shows the 4D and 6D (including RF cavities and synchrotron damping) optics and closed orbit for a δ_p offset of 0.1%. Differences between the Python and MATLAB implementations are also shown. The calculation in 4D assumes a constant δ_p offset along the ring circumference while in 6D an equivalent RF frequency shift is applied and the tracking is performed including synchrotron motion. The effect of synchrotron radiations and of the energy kick from the cavities is clearly seen on the 6D orbit. The closed orbit is calculated with iterative tracking simulations until a convergence criteria is met. This could explain the differences observed between the MATLAB and Python implementations for the 6D cases. 6D optics calculations are based on the formalism presented in Refs. [10, 11]. 6D tracking is also available for more advanced calculations such as optics matching, dynamic aperture or lifetime calculations therefore providing all the necessary tools for lattice design and characterization.

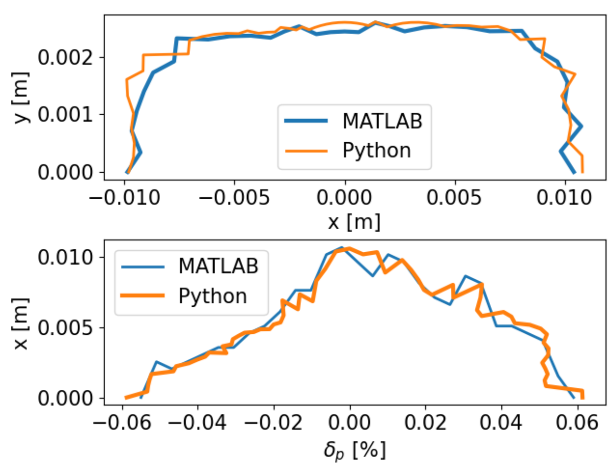

Figure 3: (x, y) and (δ_p, x) dynamic aperture computations for the HMBA lattice with the Python and MATLAB implementations.

Figure 3 shows the (x, y) and (δ_p, x) dynamic aperture computations with the Python and MATLAB implementations. Calculations were done for the HMBA lattice with synchrotron motion and radiation damping activated. The vertical dynamic aperture is cut due to a physical aperture restriction at 2.5 mm in straight sections corresponding to the half gap of the in-vacuum undulators. Overall the results are equivalent, however small differences are observed that can be explained by the different grid generation (cartesian or radial) and boundary search algorithms used in both cases. Python multiprocessing was integrated in the pyAT dynamic aperture calculation reducing the computation time from almost 10 minutes to 30 seconds on a 64 cores machine for this specific case.

COLLECTIVE EFFECTS

Collective effects were added to AT during the ESRF-EBS design phase to allow lattice and instability threshold calculations within the same framework. At the time, it was only possible to model single bunch effects on a single process. The bunch (or beam) is sliced longitudinally and convoluted with the wake field to derive the kick to apply to each particle. This operation is done every time the beam passes through

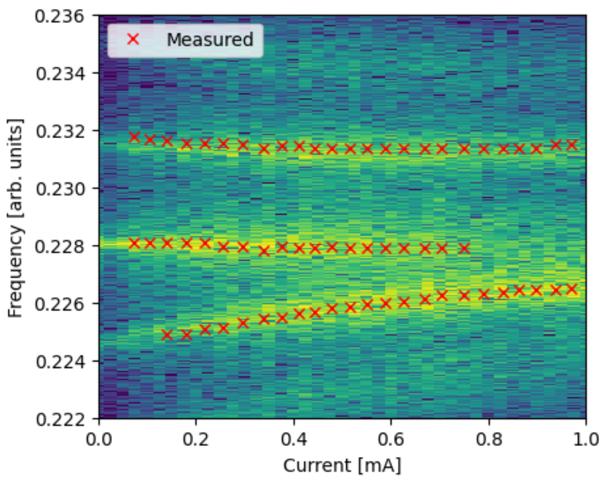

Figure 4: Measured and simulated vertical headtail modes at the ESRF-EBS storage ring [12].

the collective effects elements to account for any change in bunch distribution. The lattice can be modeled with either a one turn map or the full lattice description. Collective effects elements are treated like any other AT element and can be inserted at any location in the lattice to model local impedance contributions. The AT implementation was benchmarked against well established codes and experimental data [5,12]. Figure 4 shows a comparison between the measured and simulated vertical headtail modes at ESRF-EBS. Parallel tracking functionalities for single and multi-bunch collective effects are now fully integrated in pyAT using MPI. When MPI is activated, particles are evenly distributed on the available cores therefore ensuring optimal CPU load. Only the slices center of mass information, needed the derive the kick, are shared between processes to optimize overhead due to memory exchange between processes. Analytical wake field elements for resistive wall and resonators or arbitrary wake field tables can be used in all 3 planes. A specific implementation of RF cavity beam loading is also available that allows to automatically maintain the cavity set point [13].

ESRF RING SIMULATOR

An ESRF-EBS ring simulator was first developed in MATLAB and then ported to pyAT. The simulator consists in a pyAT loop continuously running and returning characteristic machine observables such as the closed orbit, the tune or the optics in a Tango [14] device server (DS). This DS therefore represents the virtual accelerator that mimics the behavior of the real machine. In parallel, a clone of the real ESRF-EBS control system is running that allows the users to modify the settings of the lattice used in the pyAT loop. Consequently, the simulated observables are affected in real time using the same interface and environment as the ones found in the control room. In Fig. 5, the structure of the real control system (acs.esrf.fr) and the simulator (ebs-simu) are shown side-by-side. The 2 structures are identical except for the electron beam that is replaced by the ring simulator for

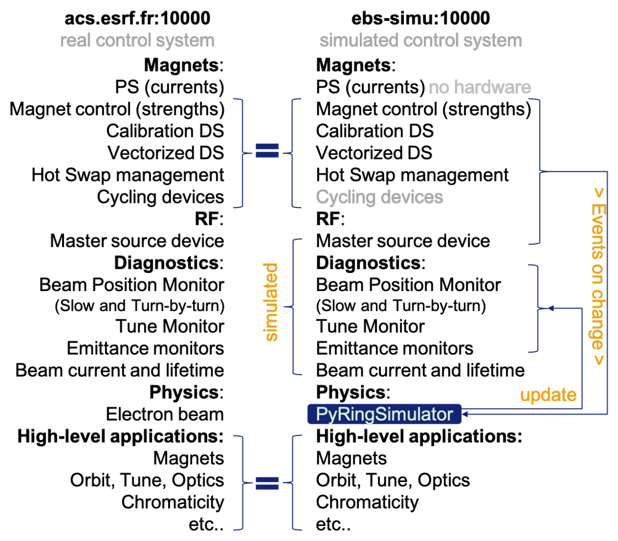

Figure 5: ESRF-EBS control system and ring simulation structures [15].

the virtual machine. A more detailed description of the implementation of the simulator can be found in Ref. [15]. The simulator made it possible to develop and validate most control applications and scripts before the beam commissioning started. This helped to optimize beam time usage to concentrate on real physics issues faced during the ESRF-EBS commissioning rather than online software debugging.

SUMMARY AND OUTLOOK

The Accelerator Toolbox has been extended with a Python interface and most features from the original MATLAB implementation have been ported to this new framework. In the context of this development new features have been added to pyAT such as parallel tracking, 6D optics calculation (now also available in MATLAB) and collective effects. These features have been thoroughly benchmarked with the MATLAB implementation when possible or experimental data and other codes in the case of collective effects. At ESRF, a ring simulator was developed as a realistic virtual representation of the ESRF-EBS control room environment. It was used to test and validate software applications and beam commissioning scripts ahead of the beam commissioning period. It is still extensively used to validate new controls developments. pyAT now features all the necessary tools to perform lattice design and characterization. The GitHub repository is very active and developments are ongoing such as lattice errors modeling and corrections, tools for commissioning simulations or the implementation of exact Hamiltonian pass methods.

REFERENCES

[1] A. Terebilo, “Accelerator toolbox for MATLAB”, SLAC, CA, USA, SLAC-PUB-8732, 2001.

[2] A. Terebilo, "Accelerator Modeling with MATLAB Accelerator Toolbox", in *Proc. PAC'01*, Chicago, IL, USA, Jun. 2001, pp. 3203–3205. doi:10.1109/PAC.2001.988056

[3] The MathWorks Inc., "MATLAB", Natick, Massachusetts, USA, https://www.mathworks.com

[4] G. J. Portmann, W. J. Corbett, and A. Terebilo, "An Accelerator Control Middle Layer Using Matlab", in *Proc. PAC'05*, Knoxville, TN, USA, May 2005, pp. 4009–4011. doi:10.1109/PAC.2005.1591699

[5] B. Nash *et al.*, "New Functionality for Beam Dynamics in Accelerator Toolbox (AT)", in *Proc. IPAC'15*, Richmond, VA, USA, May 2015, pp. 113–116. doi:10.18429/JACoW-IPAC2015-MOPWA014

[6] ATCollab, "Accelerator Toolbox", https://github.com/atcollab/at

[7] W. A. H. Rogers, N. Carmignani, L. Farvacque, and B. Nash, "pyAT: A Python Build of Accelerator Toolbox", in *Proc. IPAC'17*, Copenhagen, Denmark, May 2017, pp. 3855–3857. doi:10.18429/JACoW-IPAC2017-THPAB060

[8] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*, Morgan kaufmann, 2001

[9] L. Dalcín, R. Paz, M. Storti, "MPI for Python", *J. Parallel Distrib. Comput.*, vol. 65, pp. 1108–1115, 2005. doi:10.1016/j.jpdc.2005.03.010

[10] E. Forest, "Dispersive lattice functions in a six-dimensional pseudo-harmonic-oscillator", *Phys. Rev. E*, vol. 58, pp. 2481–2488, 1998. doi:10.1103/PhysRevE.58.2481

[11] A. Wolski, "Alternative approach to general coupled linear optics", *Phys. Rev. ST Accel. Beams*, vol. 9, p. 024001, 2006. doi:10.1103/PhysRevSTAB.9.024001

[12] L. R. Carver *et al.*, "Single Bunch Collective Effects in the EBS Storage Ring", in *Proc. IPAC'21*, Campinas, Brazil, May 2021, pp. 425–428. doi:10.18429/JACoW-IPAC2021-MOPAB117

[13] L. R. Carver, N. Carmignani, A. D'Elia, J. Jacob, V. Serriere, S. White, "Beam loading simulations in PyAT for the ESRF", presented at IPAC'23, Venice, Italy, 2023, paper WEPL030, this conference.

[14] J. M. Chaize, A. Gotz, W. D. Klotz, J. Meyer, M. Perez, and E. Taurel, "TANGO - an object oriented control system based on CORBA", in *Proc. ICALEPS'99*, Trieste, Italy, 1999, pp. 475–479.

[15] S. M. Liuzzo *et al.*, "The ESRF-EBS Simulator: A Commissioning Booster", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 132–137. doi:10.18429/JACoW-ICALEPCS2021-MOPV012