LINAC BEAM DYNAMICS SIMULATIONS WITH PY-ORBIT

A. Shishlo*, ORNL, Oak Ridge, TN 37830, USA

Abstract

Linac dynamics simulation capabilities of the PyORBIT code are discussed. PyORBIT is an open source code and a further development of the original ORBIT code that was created and used for design, studies, and commissioning of the SNS ring. The PyORBIT code, like the original one, has a two-layer structure. C++ is used to perform time-consuming computations, and the program flow is controlled from a Python language shell. The flexible structure makes it possible to use PyORBIT also for linac dynamics simulations. A benchmark of PyORBIT with Parmila and the XAL Online model is presented.

INTRODUCTION

The negative hydrogen ion SNS linac was designed [1] by using the Parmila accelerator simulation code [2]. In addition to envelope tracking codes, such as the XAL online model that is used routinely in the SNS control room, particle in cell (PIC) simulation codes like Parmila are valuable tools for halo growth calculations and beam loss estimation in heavy ion linear accelerators. Unfortunately, the Parmila code is not actively supported today, and a large project, such as SNS, cannot afford to lose PIC simulation capabilities for any part of the accelerator. During the search performed by the SNS accelerator physics group, we could not find an accelerator code that satisfies the necessary conditions of the full control over the source code, the underlying physical models, and possible modifications. To solve this problem we started the development of a home-grown open source linac accelerator code on an existing platform, namely the PyORBIT code.

PY-ORBIT AND ORBIT CODES

PyORBIT is a PIC code developed from the original ORBIT code [3]. ORBIT has been used for the design of the SNS ring and transfer lines, simulations of collective effects for SNS, and for other projects. PyORBIT, like the original ORBIT, has a two-language structure. Time-consuming calculations are performed at the C++ language level, and a simulation flow control is implemented in a scripting language. In PyORBIT the outdated and unsupported SuperCode is replaced by Python, an interpreted, interactive, object-oriented, extensible programming language. The PyORBIT project was started not only to replace the old programming technologies, but to perform calculations for the laser stripping experiment [4] which could not be performed by the original ORBIT. At this moment, PyORBIT does not have all features of the original code, but we are in the process of transferring the old ORBIT modules to the new code. It is not a straightforward process because of ubiquitous SuperCode dependencies in ORBIT. On the other hand, this provides the opportunity to restructure the modules in a more efficient and clear way.

The structure of the PyORBIT code was described in Ref. [3]. From the beginning the code has been developing as a loose structure capable of accommodating many weakly or completely unrelated projects. We took advantage of this feature of PyORBIT when the linac simulation part was included into the code.

LINAC PART OF PY-ORBIT CODE

The new linear accelerator lattice package in PyORBIT is a concrete implementation of abstract accelerator lattice classes described in [3]. Therefore, right now we have two types of the lattices – one for rings and transfer lines, which is similar to the lattice of the original ORBIT code, and another for the linear accelerators. The new implementation was necessary because the energy of a synchronous particle changes along the linac lattice, and the parameters of the lattice elements must be changed accordingly. In addition to this feature of the linear accelerator lattice, we implemented a more complicated structure that includes subsequences and RF cavities that in turn consist of RF gaps. The RF cavities themselves are not lattice elements. They are used to synchronize the phases of all RF gaps that belong to a particular RF cavity. Before using this type of lattice, it must be initialized by tracking a design particle to map its arrival time at each RF cavity. Only after that will changes to the cavity amplitudes or phases in the model reflect the changes in the real machine.

At present, a linac lattice can be built in two ways. First, it can be constructed right in a PyORBIT script by adding lattice elements one by one. Second, it can be built by using an input XML file and linac lattice parser. The parser assumes a certain structure of the input XML file. This structure will be standardized in the future when the list of necessary parameters is agreed upon among all users. At this moment, all classes in the linac packages are considered experimental, and they are kept in a specific SNS linac directory. All these classes are pure Python classes. They are lightweight and can be easily modified to accommodate different requests. It is also possible to implement a third type of abstract accelerator lattice if a more universal approach is required in the future.

There are several new C++ classes that have been created to simulate physics in linear accelerators. They include two types of space charge calculations and a simplified RF gap model. We plan to implement more sophisticated RF models in the near future.

*shishlo@ornl.gov

ISBN 978-3-95450-116-8

THE RF GAP MODEL

An accelerator node representing a zero length RF gap changes the energy of each macro-particle in the bunch and performs transverse focusing or defocusing according the arrival time (a relative phase) of this particle:

$$W_{out} = W_{in} + E_0 TL \cdot \cos(\varphi_{RF} + \varphi) \tag{1}$$

$$r'_{out} = \frac{(\gamma\beta)_{in}}{(\gamma\beta)_{out}} \cdot r' - k_f(\varphi_{RF} + \varphi) \cdot r \tag{2}$$

where W is the kinetic energy, $E_0 TL$ is the maximum energy gain, φ_{RF} and φ are the phases of RF and macro-particle relative to the synchronous particle, γ and β are relativistic parameters, r and r' are the transverse coordinate and angle of the macro-particle, and k_f is a transverse focusing coefficient [5].

THE SPACE CHARGE MODULES

Usually, particle bunches in rings and linear accelerators have different ratios between longitudinal and transverse sizes, and therefore methods of space charge calculation are different. At present, the PyORBIT space charge modules for linacs, where the transverse and longitudinal sizes are comparable, include two methods: 3D uniformly charged ellipsoid field and a 3D FFT Poisson solver.

Electric Field of Uniformly Charged Ellipsoid

The simplest way to calculate space charge forces for the linac bunch is to approximate its charge distribution by a uniformly charged 3D ellipsoid. The electric field inside and outside of such an object can be easily calculated [6], and the space charge force momentum kicks are applied to each macro-particle in the bunch. The parameters of the ellipsoid are found from the condition of equality of rms sizes for the real bunch and the approximation. This scheme is very simple and fast, and it will work even if the linac bunch consists of very few macro-particles (several hundreds is an acceptable number). This approach can be considered a variant of beam envelope calculations in Trace3D or in the XAL online model. If the charge density distribution is more complicated, an arbitrary number of uniformly charged ellipsoids can be used.

3D FFT Poisson Solver

If the linac bunch has an asymmetric shape, the Poisson equation for the electric potential should be solved by an exact method. For this purpose PyORBIT has a 3D FFT Poisson solver that uses the FFTW library. This solver will work in the case of parallel calculations, but it has bad parallel scalability, because all calculations related to the FFT transformations are identical and are performed on each CPU. The parallel efficiency of this module is determined only by the distribution of the macro-particles between CPUs. In the future we plan to add more efficient 3D Poisson Poisson solvers.

PY-ORBIT VS. PARMILA

A benchmark between Parmila and PyORBIT for linear accelerators was performed for the warm linac of the SNS accelerator. It includes the Medium Energy Beam Transfer (MEBT) line, Drift Tube Linac (DTL), and the Coupled Cavity Linac (CCL). All efforts were made to create identical lattices for the two codes. PyORBIT generated an initial “water-bag” distribution of 20000 macro-particles. The exact particle coordinates were translated to Parmila notation and packed into the Parmila’s direct access FORTRAN file. The design initial Twiss parameters were used. The peak current of the linac bunch was set to 38 mA. The uniformly charged ellipsoid model was used for the space charge nodes in the PyORBIT lattice. For Parmila’s space charge calculations the option “3D Picnic” was used. The execution time for the PyORBIT script on one CPU was about five times faster than for Parmila. The results of the simulations for all 90 meters of the warm SNS linac are shown in Fig. 1.

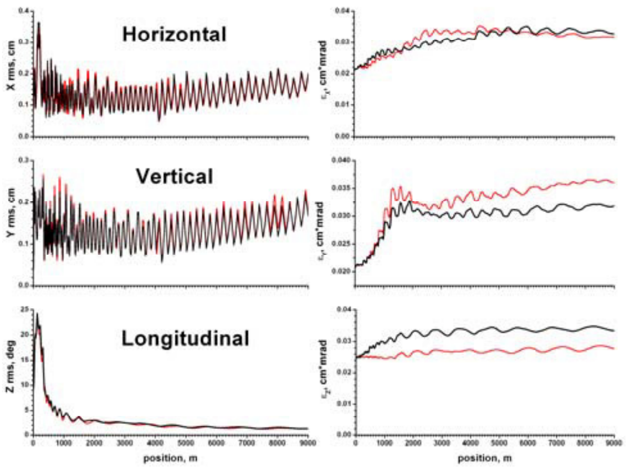

Figure 1: The calculated rms sizes of the bunch (on the left side) and emittances (on the right) in the warm part (MEBT-DTL-CCL) of the SNS linac. The black line is for Parmila, and the red one is for PyORBIT.

There is not perfect agreement between the Parmila and PyORBIT results, but we have to take into account that in the PyORBIT simulations we used a simplified RF gap model. That can explain the big differences between the longitudinal emittances for the two codes seen in Fig. 1. The usage of 3D the Poisson solver instead of the uniform ellipse model did not change the PyORBIT results much. The execution time was twice as fast for PyORBIT with the same number of grid points in the 3D solver.

Our benchmark can be compared to the benchmark between the Track code and Parmila for the DTL part of the SNS linac [7]. The quality of agreement in the Track-Parmila and PyORBIT-Parmila benchmarks are similar, especially if we take into account that here we include 90 meters (MEBT-DTL-CCL) of the SNS linac instead of 40 meters (DTL only) as in Ref. [7].

ISBN 978-3-95450-116-8

PY-ORBIT AND XAL ONLINE MODEL

Another benchmark to check the correctness of the linac part of the PyORBIT code involves the XAL online model (OM) [8]. OM is an envelope tracking code that was carefully tested against Parmila, Impact, Trace3D, and the SNS linac real measurements. The results of PyORBIT and OM for the SNS Superconducting Linac (SCL) are shown in Fig. 2. Both simulations include space charge effects for a peak current of 38 mA. The agreement between the two codes is very good, especially if we keep in mind the fundamental differences between codes: one is a PIC and the other is an envelope-tracking code.

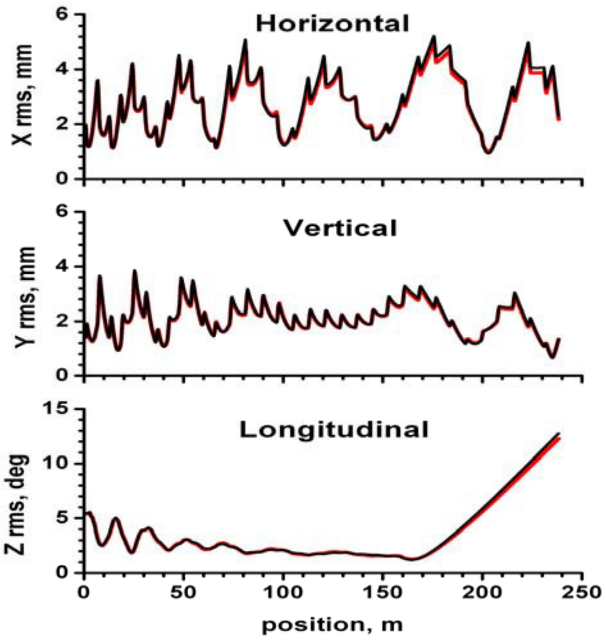

Figure 2: The transverse and longitudinal rms sizes of the bunch in the SNS SCL linac simulated with PyORBIT (red line) and OM (black line) codes.

FUTURE PY-ORBIT DEVELOPMENT

The most urgent need of the linac part of PyORBIT is a development of more comprehensive RF gap models. We are going to develop at least two new models. The first model will reproduce Parmila's approach [5] with the same simulation time, and the second one will include the 3D tracking of each particle in the time dependent electromagnetic field of the cavity. The second model will slow down the simulations, but it will allow verification of the simplified approaches.

Another addition will be a collimation module that will be moved from the original ORBIT. It will be shared with the ring related part of PyORBIT, and it will provide the loss accounting for the linac models. It also can be used for a design of collimation systems at SNS.

The development of the new space charge modules will depend on available manpower and a real necessity in parallel calculations.

CONCLUSIONS

Linac beam dynamics simulation capabilities have been successfully implemented into the PyORBIT code, which was previously restricted to rings and transport lines. The new linac part has been successfully benchmarked against Parmila and the XAL Online Model. The simplified RF cavity model and a non-scalable parallel 3D space charge solver will be supplemented with more realistic and effective models.

PyORBIT is an open source code, and it can be downloaded from the Google project hosting [9]

ACKNOWLEDGMENT

Author is grateful to A. Aleksandrov and C. Allen for help with the Parmila and the XAL online model simulations during the benchmarks.

ORNL/SNS is managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725.

REFERENCES

[1] J. Stovall, et al., “Expected Beam Performance of the SNS Linac,” Proc. of the 2001 Particle. Accelerator Conference, Chicago, Ill., June 18-22, 2001, p. 446.

[2] J.H. Billen and H. Takeda, PARMILA Manual, Report LAUR-98-4478, Los Alamos, 1998 (Revised 2004).

[3] A. Shishlo, J. Holmes, and T. Gorlov, “The Python Shell for the ORBIT code”, Proc. of ICAP ’09, THPSC052 (2009); http://www.JACoW.org

[4] T. Gorlov, A. Shishlo, “Laser stripping computing with the Python ORBIT code”, Proc. of ICAP ’09, TH3IOPK03 (2009); http://www.JACoW.org

[5] T. P. Wangler, “RF Linear Accelerators”, (Wiley-VCH Verlag GmbhH & Co, 2008), 209

[6] O. D. Kellogg, Foundations of Potential Theory, (Dover, New York, 1953), 192.

[7] B. Mustapha, “First Track Simulation of the SNS Linac,” LINAC’06, Knoxville, TN 2006, TUP076, p. 432 (2006); http://www.JACoW.org

[8] http://sourceforge.net/projects/xaldev/

[9] http://code.google.com/p/py-orbit/source/checkout