ISBN: 978-3-95450-253-0 ISSN: 2673-5571 doi:10.18429/JACoW-HB2023-TUC2C2

EVALUATING PyORBIT AS UNIFIED SIMULATION TOOL FOR BEAM-DYNAMICS MODELING OF THE ESS LINAC

C. Zlatanov, J. F. Esteban Müller, Y. Levinsen, N. Milas,
European Spallation Source ERIC, Lund, Sweden
A. Zhukov, A. P. Shishlo, ORNL, Oak Ridge, TN, USA

Abstract

The design of the ESS proton linac was supported by the simulation code TraceWin, a closed-source commercial software for accurate multiparticle simulations. Conversely, high-level physics applications used for beam commissioning and machine tuning rely on the Open XAL framework and its online model for fast envelope simulations. In this paper, we evaluate PyORBIT for both online modeling of the linac for machine commissioning and tuning as well as for more accurate offline simulations for beam-dynamics studies. We present the modifications done to the code to adapt it to this use case, namely porting the code to Python 3, adding an envelope tracker, and integrating with the EPICS control systems. Finally, we show the results of benchmarking PyORBIT against our current modeling tools.

INTRODUCTION

At ESS, we mainly rely on two codes for beam-dynamics simulations: the Open XAL [1] online model, an envelope code; and TraceWin [2], a powerful and feature-rich commercial code for both envelope and multiparticle simulations. This setup has worked fine, but we identified some weak points: we need to maintain two different lattice files, TraceWin is a closed-source code therefore we can't investigate implementation details or extend it, and Open XAL is written in Java while Python remains the favorite language among accelerator physicists.

PyORBIT [3] is an open-source Python code for multiparticle beam-dynamics simulations in linacs and synchrotrons that is very rich in features. It implements three different algorithms for space charge, a numerical integrator for tracking particles under arbitrary electromagnetic fields, routines for beam-coupling impedance and beam-matter interaction, as well as MPI integration for parallel computing.

Being an open-source project, we can extend PyORBIT and adapt it to our needs, for instance, by integrating it with our control system, or by adding an envelope tracker. For this reason, in this paper we are evaluating the code to use it for all beam-dynamics studies at ESS.

Furthermore, at ESS we are also involved in other projects that would require a synchrotron, for example ESSnuSB [4] and the muon collider [5]. In both cases, we are evaluating software tools for the design and simulation of synchrotrons and transfer lines, which PyORBIT can do.

PORTING CODE TO PYTHON 3

At the time when we started considering using PyORBIT at ESS, the latest version only supported Python 2.7 or older, which has been deprecated since 2020. For that reason, we focused the first efforts on porting the code to make it compatible with the latest releases of Python 3.

At the same time, we improved the build mechanism to produce a Python package that can be installed using the `pip` command, which should simplify installation and usage of the tool. In previous versions of PyORBIT, users had to build a custom Python interpreter that automatically loaded the PyORBIT libraries as built-in Python modules.

ESS LATTICE DEFINITION

PyORBIT supports several formats for describing the accelerator lattice, such as MAD-X [6], SAD [7], and a custom XML format.

At ESS, we are exploring the option to generate our lattice directly in Python. We defined a set of helper functions that simplify instantiating the sequences and elements in the lattice, as well as a function to pre-process the lattice to add the drift elements.

On Listing 1 we show a basic example of how we define a lattice in Python. In this example, we create a lattice with one section that contains one RF cavity with a single gap and a quadrupole magnet.

Listing 1: This is a snippet showing an example of a lattice defined in Python. In the snippet, we instantiate a Lattice object and add to it a sequence containing an RF cavity with one RF gap and a quadrupole magnet.

```
from typing import List
from orbit.lattice import AccLattice, AccNode
from orbit.py_linac.lattice import
    LinacAccLattice, Sequence
import lattice_builder as builder

# Instantiating a Lattice element
lattice = LinacAccLattice("ESS Lattice")
frequency = 352.21e6
maxDriftLength = 0.005

accSeqs: List[Sequence] = []

# Adding the DTL sequence
dtl_seq = builder.addSequence(
    lattice=lattice,
    accSeqs=accSeqs,
    name="DTL",
    length=33.1528945,
    bpmFrequency=7.0442e+08,
    start_position=0)

# Adding a cavity element
dtl_tank_1 = builder.addCavity(
    sequence=dtl_seq,
    name="DTL-010:EMR-Cav-001",
```

ISBN: 978-3-95450-253-0 ISSN: 2673-5571 doi:10.18429/JACoW-HB2023-TUC2C2

```
    amplitude=0.003,
    phase=-35.0,
    frequency=frequency,
    position=0
    )

# Adding a quadrupole magnet
builder.addQuad(
    sequence=dtl_seq,
    name="DTL-010:BMD-PMQ-001",
    length=0.05,
    field=-59.9444,
    aperture=0.02,
    aprt_type=1,
    pos=0.025,
    maxDriftLength=maxDriftLength)

# Adding an RF gap to a cavity element
ttfs_element = builder.TTFs(
    beta_max=0.99,
    beta_min=1e-06,
    polyT=[
        1.07731,
        -0.00257565,
        -1.17398e-05])

builder.addRFGap(
    sequence=dtl_seq,
    cavity=dtl_tank_1,
    name="DTL-010:EMR-Cav-001:G1",
    amplitude_factor=0.781302,
    length=0.0752368,
    mode=0,
    EzFile="?",
    aperture=0.02,
    aprt_type=1,
    pos=0.0626084,
    ttf=0.780607,
    ttfs_element=ttfs_element)

# Performs sanity checks and add drifts
builder.processNodes(
  lattice=lattice,
  sequence=dtl_seq,
  thinNodes=thinNodes,
  maxDriftLength=maxDriftLength)

lattice.initialize()
```

The advantages of this option are:

- no need to learn a new syntax.
- possibility to programmatically describe machine sections, e.g., group periodic sections, auto-generate signal names for integration with the control system, etc.
- taking advantage of IDE features such as auto-completion, syntax highlighting, error detection, and display documentation.
- deployment can be done by installing a Python package using `pip`.

The main disadvantages we found for this option are that modifying a lattice adds the extra step of reinstalling the `pip` package and that switching between different lattices at runtime becomes more cumbersome.

Benchmark against TraceWin

PyORBIT is a mature code that has¨ been successfully used to simulate different machines. Nevertheless, we decided to perform a benchmark against our reference multiparticle simulations, done using TraceWin, for validation purposes.

We have tested PyORBIT for the section of the ESS linac that has been installed and commissioned with beam until now, i.e., up to the fourth DTL tank. Figure 1 shows the values for the rms envelope computed using PyORBIT and TraceWin, using similar parameters for the space-charge solver. The initial distribution was also the same. The envelope in the longitudinal plane is shown in Fig. 2.

In both cases, we see that the results agree very well. There are small differences in the transverse plane at the end of the section and some oscillations in the longitudinal plane in the results from PyORBIT. These differences are probably due to the different approaches for meshing the bunch for the space-charge calculation. In TraceWin, the mesh size is calculated using the rms sizes, while in PyORBIT it considers all the particles. The distribution that we use has few particles that drift away before they are lost and make the grid size larger in PyORBIT as compared to TraceWin. In addition, we used different strategies to remove lost particles.

ENVELOPE TRACKER

PyORBIT implements a multiparticle tracker that enables precise beam-dynamics simulations. However, precision comes at a computational cost that some control room applications don't need, since they don't require such accuracy and would benefit from faster simulations. Examples are applications for trajectory correction or phase scan, that don't need even to compute the space-charge effect.

For this reason, we are implementing an envelope tracker based on a first-order matrix model of the machine and a linear space-charge model. This is a work in progress and so far we implemented the drifts, quadrupole magnets, and RF cavities. This model will be enough to be used by a phase scan application. For the trajectory correction application, we still need to implement bending magnets.

For lattice matching applications we are planning to implement the linear space-charge solver.

Benchmarking the Envelope Solver

The envelope tracker has been benchmarked against Open XAL and the envelope solver of TraceWin. Results are shown in Fig. 3. In this case, the results almost perfectly match one another.

We also measured runtime for this example to be in the order of 10 ms on a Macbook Pro with an M1 processor. We expect this time to remain in the order of tens of ms when the linear space-charge solver is implemented. As a comparison, running the PyORBIT multiparticle tracker with 1 million particles, a 5 mm-step for the space-charge solver, and a 64x64x64 grid, the simulation takes around 4 minutes on the same laptop.

ISBN: 978-3-95450-253-0 ISSN: 2673-5571 doi:10.18429/JACoW-HB2023-TUC2C2

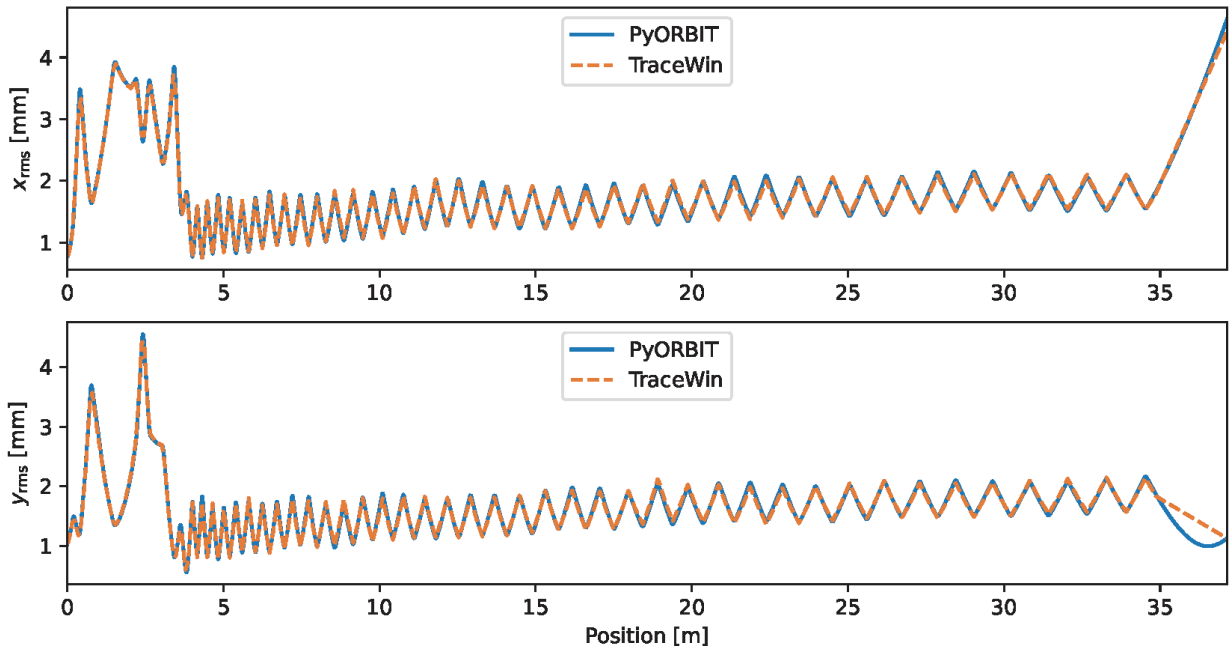

Figure 1: Results from multiparticle simulations using PyORBIT (blue) and TraceWin (orange), showing the transverse beam size (rms) from the MEBT to the fourth DTL tank in the horizontal (top) and vertical (bottom) planes. Simulations were performed for a 62.5 mA beam using a 3D space-charge solver in both codes. Differences in the vertical plane on the right side are due to the limited number of points from the TraceWin simulations output.

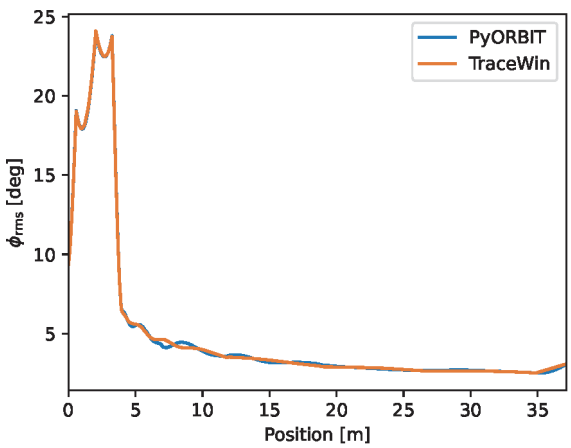

Figure 2: Results from multiparticle simulations using PyORBIT (blue) and TraceWin (orange), showing the longitudinal beam size from the MEBT to the fourth DTL tank. Simulations were performed for a 62.5 mA beam using a 3D space-charge solver in both codes. Oscillations in PyORBIT results are probably due to a larger grid size in the space-charge solver due to particles drifting away.

EPICS INTEGRATION

The control system at ESS is based on the EPICS protocol [8]. In order to use PyORBIT as an online model for high-level beam-physics applications, we have developed an extension that integrates the latest version of EPICS.

The extension is written in Python and it is based on the p4p module [9]. It is bundled as a Python package that can be installed using the `pip` command.

In order to use it, one only needs to add the EPICS PV names into the lattice, as shown on Listing 2. The package provides helper functions to programmatically add the channels to the lattice elements, which enable users to both read and write directly from and to the devices. It also adds synchronization methods to the lattice object to read all parameters from the control system at once.

At the moment of writing this paper, the extension supports electromagnets and RF cavities. Beam diagnostics will be added soon. The source code of the `pyorbit-epics` extension can be found in `https://gitlab.esss.lu.se/ess-crs/pyorbit-epics`.

Listing 2: This snippet shows how EPICS channels can be attached to lattice elements using the pyorbit-epics extension.

```
# Adding EPICS channels to a quadrupole magnet
channels.addMagnetChannels(
  node=quad_1,
  current_channel="MEBT-010:PwrC-PSQV-001:Cur-R",
  current_set_channel="MEBT-010:PwrC-PSQV-001:Cur-S",
  convFactor=-0.162
  )

# Adding EPICS channels to an RF cavity
channels.addCavityChannels(
  node=cavity_1,
  amplitude_channel="MEBT-010:RFS-LLRF-101:SPRampingA",
```

ISBN: 978-3-95450-253-0 ISSN: 2673-5571 doi:10.18429/JACoW-HB2023-TUC2C2

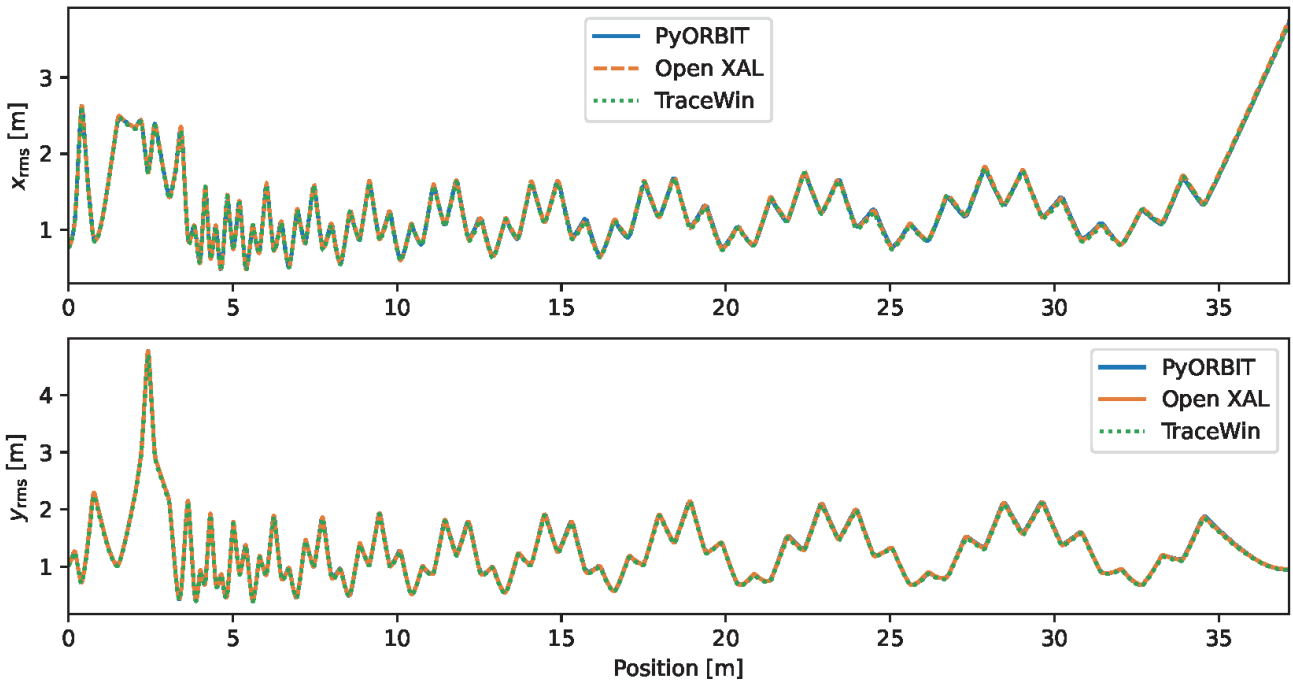

Figure 3: Results from envelope simulations using PyORBIT (blue), Open XAL (orange), and TraceWin (green), showing the transverse beam size (rms) from the MEBT to the fourth DTL tank in the horizontal (top) and vertical (bottom) planes.

```
    amplitude_set_channel="MEBT-010:RFS-DIG
    -101:AI0-SMonAvg-Mag",
    phase_channel="MEBT-010:RFS-DIG-101:
    RFCErrLimSBCmp1",
    phase_set_channel="MEBT-010:RFS-LLRF-101:
    SPRampingPhase")

```

CONCLUSION

The initial evaluation of the PyORBIT code has demonstrated that the code fulfills our requirement for both accurate multiparticle simulations and can be extended with an online model and EPICS integration for control room applications with a reasonable software development effort.

We have ported the code to Python 3, improved the build and deployment process, prototyped an envelope tracker, and integrated it with our EPICS control system.

The code can be found at `https://github.com/PyORBIT-Collaboration/PyORBIT3`.

FUTURE STEPS

As shown in this paper, we have done a general verification of the code, but a more thorough verification of the beam-dynamics model is needed and planned. The test will include misalignments and mismatches.

We also plan to complete the EPICS integration and the envelope model to build the first control room applications.

Another development that is planned to take place in the near future is the study of GPU acceleration for the multiparticle tracker. If we could reach a speed-up of 2 or 3 orders of magnitude, we could start using the multiparticle tracker for the online model.

REFERENCES

[1] Open XAL repository. `https://openxal.github.io`

[2] D. Uriot, TraceWin. `http://irfu.cea.fr/Sacm/logiciels/index3.php`

[3] A. Shishlo, S. Cousineau, J. Holmes, and T. Gorlov, "The particle accelerator simulation code pyorbit," *Procedia Comput. Sci.*, vol. 51, pp. 1272–1281, 2015.

[4] A. Alekou *et al.*, "The European Spallation Source neutrino Super Beam," *Eur. Phys. J. Spec. Top.*, vol. 231, pp. 3779–3955, 2022. `doi:10.1140/epjs/s11734-022-00664-w`

[5] D. Schulte, "The Muon Collider," in *Proc. Int. Part. Accel. Conf. '22*, Bangkok, Thailand, May 2022, pp. 821–826. `doi:10.18429/JACoW-IPAC2022-TUIZSP2`

[6] MAD-X code website. `http://madx.web.cern.ch/mad`

[7] SAD code website. `https://acc-physics.kek.jp/SAD`

[8] EPICS collaboration, The experimental physics and industrial control system. `https://epics-controls.org`

[9] M. Davidsaver, P4P repository. `https://mdavidsaver.github.io/p4p`