PYTHON FLUKA BEAMLINE (PYFLUBL) : UPGRADES AND DEVELOPMENTS.

S. T. Boogert *, Cockcroft Institute, Daresbury Laboratory, Daresbury, UK
L. Nevay [1], CERN, Geneva, Switzerland

Abstract

FLUKA simulations of beam lines are essential for understanding different aspects of accelerators, including beam losses, backgrounds, activation, and shielding. Creating a beam line simulation using FLUKA is a time-consuming and error-prone process. This paper describes updates to pyflubl (Python FLUKA beam-line), a set of Python tools which can create a FLUKA simulation using a Python script or accelerator simulation code, like MADX. pyflubl is based on multiple stable and advanced Python packages created to make BDSIM (Geant4) beamline simulations as simple as possible; these are pymadx (an interface to MAD-X), pymad8 (an interface to MAD8), pybdsim (interface to BDSIM) and most importantly pyg4ometry (a geometry engine for Monte Carlo geometry creation). This paper describes developments that complete pyflubl into a fully functioning package, including implementation of the `LATTICE` command, improvement implementation of pole face rotations (with component tilts), custom electric and magnetic fields and particle sources.

INTRODUCTION

This paper describes upgrades to pyflubl a Python/C++ helper code that augments the capabilities of FLUKA [1] to easily simulate particle accelerators. A first functioning but incomplete pyflubl was first presented last year at IPAC 2025 [2] and numerous improvements have been made. There are existing codes to perform a similar task beam LineBuilder [3] and FLUKA beam line 3D builder [4].

pyflubl consists of two main components a PYTHON package which creates input (`inp`) files for FLUKA and custom routines in C++ which allow storage of the output in a ROOT format particularly convenient for analsis, for this paper this will called fluka-pyflubl.

This paper describes the developments for automatic MADX [5], MAD8 [6] and TRANSPORT [7] conversion to FLUKA simulation, important elements of this are complete Python interface of FLUKA cards, custom magnetic and electric field and particle source routines. pyflubl depends heavily on pyg4ometry [8] to load and manipulate FLUKA geometry in python. To replicate accelerator components a more complete interface to the `LATTICE` card was required. With these developments a more complete coupling between beam tracking codes and pyflubl is possible. An example of a short accelerator section is presented and results from pyflubl and BDSIM [9] are compared.

* stewart.boogert@cockcroft.ac.uk

Curvilinear Coordinate Generation

Since the first version of `pyflubl` the entire generation of accelerator component coordinates, as shown in Fig. 1 has been refactored. A dedicated PYTHON class `Coordinates` is used to generate the position and rotation of all components, including the calculation of the outer FLUKA body. This was done to allow the computation of component start and end normals and consistently allow tilts of bending components.

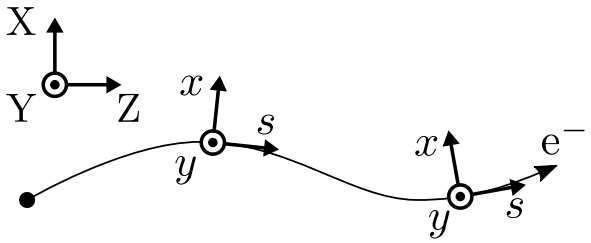

Figure 1: Local curvilinear coordinates and FLUKA global coordinates

Geometry

To allow simple construction of accelerator models in FLUKA, all accelerator components need to have the same basic geometric layout, shown in Fig. 2. The component needs to be completely bounded in a single body (e.g. `ARB`) which we call the outer body. Within the outer body there also must be a beampipe region and a vacuum region to assign the accelerator magnetic fields.

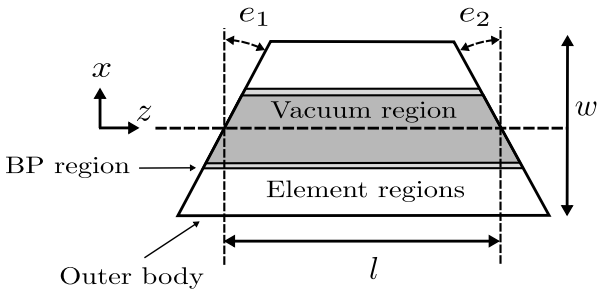

Figure 2: Generic FLUKA geometry used for a beam line component.

pyflubl allows users to load existing geometry from either Geant4 (gdml files) or FLUKA (inp files) to use in a FLUKA simulation. This externally loaded geometry can be instanced using `LATTICE`. This external geometry also needs to have the same hierarchy as in Fig. 2.

UPGRADES AND IMPROVEMENTS

Particle Source

Typically accelerator scientists define the beam in terms of Twiss parameters (α, β and γ) along with the emittance (ϵ). pyflubl now supports a wide range beam definitions which are useful for accelerator scientists, a) Twiss parameters, b) input file and c) elliptical annulus. The beam location is most conveniently defined in terms of local curvilinear coordinates x, y, S and not global FLUKA coodinates X, Y and Z. To allow this pyflubl supplies a custom `source_newgen.f` FORTRAN routine. Input to the new subroutine is supplied by the `SOURCE` card.

FLUKA Lattice

The FLUKA LATTICE command allows the creation of instances of a prototype geometry. This is particularly useful as accelerators are typically constructed from many objects from a limited set of components (e.g. quadrupoles, bends etc). Previously in pyflubl components can be added to a machine via `Machine.AddXX`. This API remains but with an additional LATTICE flag, the geometry is located in a separate parking location surrounded by gold and LATTICE commands are used to link instances and the prototype, an example is given in Listing 1 and the corresponding geometry displayed in Fig. 3.

Listing 1: A simplified Python script to create a FLUKA beamline consisting of 3 drifts of 1 m length. A complete version can be found the project GitHub repository.

```
import pyflubl as _pfbl
m = _pfbl.BuilderNew.Machine()
d1 = m.AddDrift(name="d1", length=1, add=
    False)
m.AddLatticePrototype(d1)
m.AddLatticeInstance("d1i1","d1")
m.AddLatticeInstance("d1i2","d1")
m.AddLatticeInstance("d1i3","d1")
```

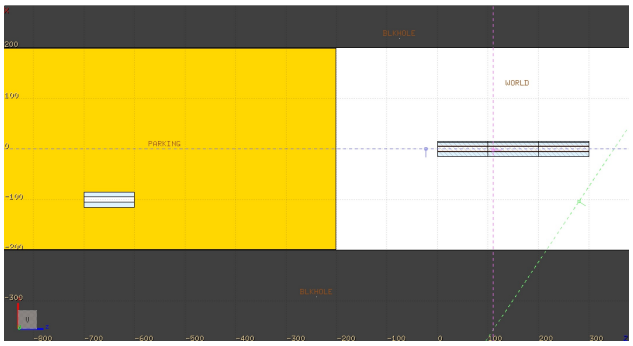

Figure 3: Output of listing 1 to create multiple instances of a drift, displayed in FLAIR.

The operation of the `LATTICE` functionality has been confirmed with components which possess magnetic fields, e.g. quadrupoles and bends.

Tracking in Fields

The FLUKA electric and magnetic field cards allow for a wide range of magnetic fields but there is a need for users to define more complex fields. The fields can be defined by users by implementing `magfld.f` and `elefld.f` FORTRAN routines. `pyflubl` supports the FLUKA `MGNFIELD` and `MGNCREAT` cards which allow multipole fields up to decapole to be defined without custom fortran files. Magnetic (and electric) fields are assigned only to the beampipe vacuum region presently. The new `Coordinates` class is used to generate the transformations from the magnet `REGION` to the magnetic field coordinate system. The `STEPSIZE` for each `REGION` with a magnetic field can be set using pyflubl .

Analysis and Plotting

`pyflubl` generates a book-keeping file that stores transformation information between FLUKA regions and beamline elements and a coordinates (JSON) file that preserves beamline survey. `pyflubl-fluka` generates multiple different files, output from `USRBIN`, `USRBNX` and `USERDUMP` cards and a dedicated ROOT file. All the output can be loaded and analysed by a single dedicated PYTHON class `PyflublOutput`. To demonstrate the capability and flexibility of this approach a simple exmaple consisting of a drift (1 m), quadrupole, drift, target (1 cm iron), drift, sbend (2 m) and drift was created. The beam is defined to be an 1 GeV electron with momentum in the z axis. The beampipe has radius of 5 cm and is 5 mm thick stainless steel. For simplicity there is no other material. Two `USRBIN` all particle are placed on the target and drift after the sbend. Figures 4 and 5 show the output of the `USRBIN` and `USERDUMP` outputs overlayed on a simplified geometry.

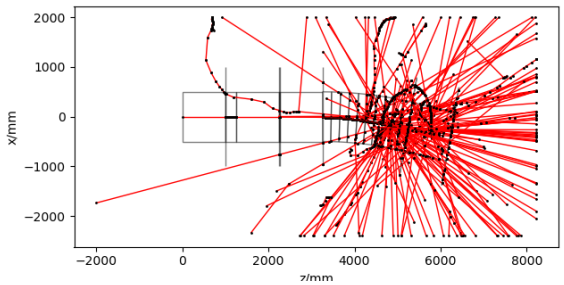

Figure 4: z-x plot of the example beam line with the USERDUMP output overlayed

The transversely wide and longitudinally thin elements in Figs. 4 and 5 are samplers, which record the position (x, y), , angle (x' and y'), energy (E) and type (particle ID) of particles passing through. These coordinates are transformed from global to component local coordinates and recorded in the ROOT file. All energy deposits are also recorded in the ROOT file and transformed to accelerator curvilinear coordinates (x, y and S). An example of this output is shown in Fig. 6, The 4th sampler is located just before the sector bend at approximately S = 3250 mm. The electrons which interact with the target produce electrons,

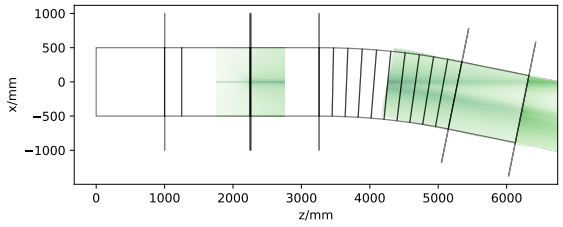

Figure 5: z-x plot of the example beam line with the USRBIN all particle flux plot overlayed.

photons and positrons which interact with the stainless steel beam pipe, so the sampler records a distribution of particles to ± 5 cm the beam pipe radius.

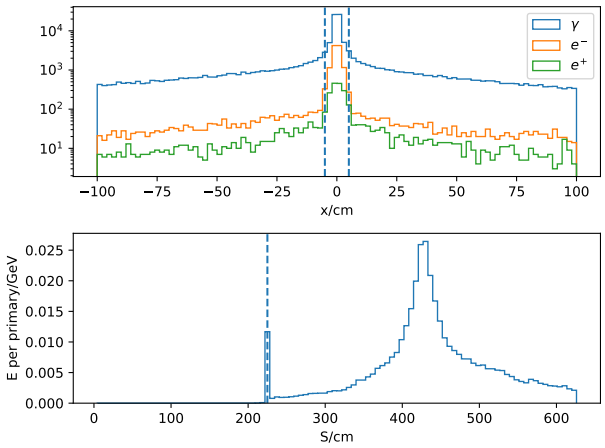

Figure 6: Top: x position of particles passing through the 4th sampler. The dashed lines represent the steel beam pipe radius. Bottom: Energy deposit distribution as function of S. The dashed line represents the location of the 1 cm iron target.

COUPLING WITH EXTERNAL CODES

This first version pyflubl allows the creation of a beamline, a complete consistent simulation of the beamline geometry. A common use of FLUKA for accelerator physics is to simulate particle-matter interactions for particular components. The rest of the tracking is performed in a beam tracking code (e.g. Xsuite), particles are passed to FLUKA and interaction products passed back to the beam tracking code. A prototype interface has been developed which allows particles to be passed to the FLUKA via the `source_newgen.f` routines, these particles are transformed the appropriate coordinates for the LATTICE instance using coordinate transforms stored in the book-keeping file and then the simulation output stored by a user `DUMP` call, which in turn calls custom code in the `mgdraw.f`. This interface is written in C++ and is blocking of operation of FLUKA, so a `pyflubl-fluka` waits until particles are supplied via C++ function call, then also waits until the particles are flushed or received by a C++ set flag which is monitored in `usreou.f`

COMPARISON WITH BDSIM

BDSIM is an advanced Geant4 based code to simulate beamlines, The sample (target-bend) example was created in BDSIM and simulated for the same number of initial primaries. The energy loss as a function of S for both FLUKA and BDSIM is shown in Fig. 7.

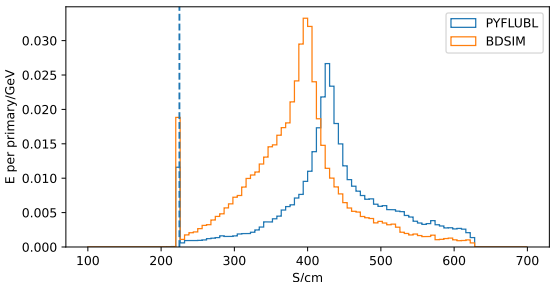

Figure 7: Comparison of energy deposit as function of S between pyflubl and BDSIM.

In the important features BDSIM and pyflubl agree well, specifically the location of the target, largest energy loss in the bend and the end of the distribution. The overall normalisation is in reasonable agreement. Significantly more comparision tests will need to be performed comparing pyflubl with BDSIM.

FUTURE WORK, LIMITATIONS AND CONCLUSIONS

`pyflubl` is becoming a capable simulation tool for generating FLUKA beam line simulations. The code is publicly available via GitHub [10]. A docker file is available in the repository to build an appropriate FLUKA environment with pyflubl . Currently the focus is creating the extensible framework for performing production simulations. An important rationale for pyflubl is to enable rapid comparisons with other codes, e.g. BDSIM, which is shown for the first time in this paper.

FLUKA is limited in its ability to track particles in an accelerator lattice. For example, time-varying fields is important for accelerator tracking and **are** simulated in BDSIM but not FLUKA. If the tracking in performed in a dedicated beam dynamics code and particles are transferred appropriated back and forth between pyflubl and FLUKA these limitations can be avoided. An important limitation of FLUKA is .Thin magnetic kicks can be implemented by hijacking `mgdraw` boundary crossings, but this approach is likely to be error prone and fragile.

REFERENCES

[1] G. Battistoni, *et al.*, "Overview of the FLUKA Code", *Annals of Nuclear Energy*, vol. 82, pp. 10–18, 2015. doi:10.1016/j.anucene.2014.11.007

[2] S. Boogert and L. Nevay, "Python FLUKA beam line, a python library to create FLUKA simulations of accelerators", in *Proc. IPAC'25*, Taipei, Taiwan, Jun. 2025, pp. 2321–2324. doi:10.18429/JACoW-IPAC2025-WEPS038

[3] A. Mereghetti, V. Boccone, F. Cerutti, R. Versaci, and V. Vlachoudis, "The FLUKA LineBuilder and Element DataBase: Tools for Building Complex Models of Accelerator Beam Lines", in *Proc. IPAC'12*, New Orleans, LA, USA, May 2012, paper WEPPD071, pp. 2687–2689.

[4] M. Santana-Leitner, Y. Nosochkov, and T. O. Raubenheimer, "MadFLUKA Beam Line 3D Builder. Simulation of Beam Loss Propagation in Accelerators", in *Proc. IPAC'14*, Dresden, Germany, Jun. 2014, pp. 463–465. doi:10.18429/JACoW-IPAC2014-MOPME040

[5] pymadx Git Hub, https://github.com/bdsim-collaboration/pymadx

[6] pymad8 Git Hub, https://github.com/bdsim-collaboration/pymad8

[7] pytransport Git Hub, https://github.com/bdsim-collaboration/pytransport

[8] S. D. Walker, A. Abramov, L. J. Nevay, W. Shields, and S. T. Boogert, "Pyg4ometry: A Python library for the creation of Monte Carlo radiation transport physical geometries", *Comput. Phys. Commun.*, vol. 272, pp. 108228, 2022. doi:10.1016/j.cpc.2021.108228

[9] L. J. Nevay, *et al.*, "BDSIM: An accelerator tracking code with particle–matter interactions", *Comput. Phys. Commun.*, vol. 252, pp. 107200, 2020. doi:10.1016/j.cpc.2020.107200

[10] pyfubl Git Hub, https://github.com/bdsim-collaboration/pyflubl