SoftwareX 24 (2023) 101579

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Georges: A modular Python library for seamless beam dynamics simulations and optimization

Robin Tesse [a,*], Cédric Hernalsteens [b,a], Eustache Gnacadja [a], Nicolas Pauly [a], Eliott Ramoisiaux [a], Marion Vanwelde [a]

[a] Service de Métrologie Nucléaire (CP165/84), Université libre de Bruxelles, Avenue Franklin Roosevelt 50, 1050 Brussels, Belgium
[b] CERN, European Organization for Nuclear Research, 1211 Geneva 23, Switzerland

ARTICLE INFO

Keywords:
Particle tracking
Optimization
Energy degradation
Python

ABSTRACT

Particle tracking codes such as MAD-X or TRANSPORT commonly use a matrix formalism to propagate beams through magnetic elements as it simplifies the analysis of particle behavior, facilitates beam optimization and component design, and enables accurate particle accelerator simulations. However, these codes are inefficient when tracking many particles or accounting for energy degradation along the beamline. To overcome these limitations, we introduce Georges, a Python library used in the field of particle accelerators for medical applications comprising two modules: Manzoni and Fermi. Manzoni is an efficient particle tracking code that can track many particles while calculating beam losses and energy degradation using the Fermi–Eyges formalism implemented in the Fermi module. In this paper, we present the implementation details of Georges, which includes a verification conducted against other software tools such as MAD-X and BDSIM, along with a documentation on computational time.

Code metadata

Current code version	2023.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00354
Code Ocean compute capsule	
Legal Code License	GPL-3.0 license
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	numpy, pandas, numba, poetry
If available Link to developer documentation/manual	https://ulb-metronu.github.io/georges/
Support email for questions	cedric.hernalsteens@ulb.be and robin.tesse@ulb.be

1. Motivation and significance

The transport of charged particles in a magnetic element can be described using the following matrix formalism [1,2]:

$$X_i = \sum_j X_j R_{ij} + \sum_{jk} X_{ij} T_{ijk}, \tag{1}$$

where X is a vector containing the particle's properties, and R and T are the propagation matrices and tensors, respectively.

This matrix formalism is notably employed in the particle accelerator field due to its effectiveness in mathematically describing the dynamics of charged particles moving through electromagnetic fields. It simplifies the analysis of particle behavior, facilitates beam optimization and component design, and enables accurate particle accelerator simulations.

Many particle tracking codes such as MAD-X [3], MAD-8 [4] or TRANSPORT [1] implement this relation. However, a more detailed description is needed when tracking many particles or when the beam interacts with elements in the beamline, such as a degrader (reducing

* Corresponding author.
E-mail address: robin.tesse@ulb.be (Robin Tesse).
URL: https://protons.ulb.be (Robin Tesse).

https://doi.org/10.1016/j.softx.2023.101579
Received 30 May 2023; Received in revised form 19 October 2023; Accepted 3 November 2023
Available online 16 November 2023

the beam energy) or a collimator (removing the beam halo). The transport code TURTLE (Trace Unlimited Rays Through Lumped Elements) is based on TRANSPORT and implements multiple scattering of charged particles in the matter to estimate the emittance increasing or the beam losses [5] but is not actively maintained. Therefore, complete beam tracking is required, where each particle is propagated individually in the beamline, and the properties of the beam are statistical properties calculated over the whole distribution.

2. Software description

The Georges Python library has been developed to efficiently track a large number of particles (millions at a time) while also accounting for energy degradation. It comprises two main modules: Fermi and Manzoni.

The Fermi module is a Python implementation of the extended Fermi–Eyges model that computes the beam properties after the passage through matter [6]. Various methods are provided to calculate energy losses over a given thickness or reciprocally the required thickness for a given degradation. A complete description of the implementation can be found in Ref. [7].

The Manzoni module aims to provide fast particle beam tracking through the most encountered accelerator and beamline elements, such as magnets, scatterers, degrader, and collimators. Manzoni computes the particle output coordinates for a given magnetic element by simply applying the magnet matrix, defined by its parameters, to the input coordinates. The Fermi module is seamlessly integrated when particles interact with matter. Special attention has been paid to the speed of the code to reduce the time for propagating a large number of particles by using the Numba python library through the concept of *Just in Time Compiler* [8].

2.1. Software architecture

As explained, Georges is composed of the Fermi and the Manzoni modules. The entire structure of the library is presented in Fig. 1 and detailed in the following sections.

2.1.1. Elements

The "elements" submodule contains the physical implementation of different beamline elements, depending on their properties:

- Magnetic: Drifts, dipoles, quadrupoles, sextupoles, and multipoles.
- Collimators: Elements that interact with the beam to cut the halo. Different apertures, including rectangular, circular, elliptical, or phase-space, are available.
- Scatterers: Thin materials that interact with the particles and only modify the transverse angle along the two transverse axes.
- Degraders: Thick materials that combine particle interaction with a simple drift-type propagation along the material where the energy reduction is considered through the Fermi module.

For each element, a first ("R in (1)") and second-order ("T in (1)") type propagation is implemented, allowing the user to select the proper tracking order for his specific application. The user should be aware that the canonical variables of the particles are not the same for the three different integrator types, so the definition of the beam must be done accordingly to be consistent. The integrators "Mad8-type" and "Transport" add higher-order terms ("T in (1)") to the matrix formalism, as the output coordinate does not depend linearly on the coordinate of the incoming particle. On the other hand, the "MadX" integrator is linear in the spatial coordinates and angles but exact in the momentum deviation, making it much more precise when we are interested in beams with a large energy spread. It is also possible to specify the aperture of an element which can be rectangular, circular, elliptical, or phase-space and is used to compute the losses into an element.

```
quad = georges.Element.Quadrupole(
NAME="Q1",
L=0.3 * _ureg.m,
K1=2 * _ureg.m**-2,
APERTYPE="RECTANGULAR",
APERTURE=[5 * _ureg.cm, 3 * _ureg.cm])
quadrupole.integrator = MadXIntegrator
```

2.1.2. Definition of a sequence

Georges allows users to define a sequence consisting of magnetic (drift, quadrupole, sextupole) or non-magnetic (energy degrader, collimator) elements arranged to efficiently transport a beam of charged particles by defining each element and placing it in a sequence. A python module has been developed to convert sequences from CSV, TFS,[1] or Beam Delivery Simulation (BDSIM), a Geant4-based C++ library that can propagate charged particles through high-energy and low-energy beamlines, output files [9]. The module loops over each element and converts it into a georges.Element object. They are then placed in a georges.Sequence object, which includes methods to modify the properties of an element.

```
seq = georges.Sequence()
seq.set_parameters("el_name",  dict_of
    ↪ properties)
```

2.1.3. Beam input distribution

The module "beam" contains the implementation of the class beam for tracking with Manzoni. A beam definition requires the "kinematics" of the particles and a beam distribution to allow the generation of the particles for tracking. The kinematics module deals with relativistic physics computations and mainly concerns conversions between kinematic quantities *e.g.,* computing the momentum from the kinetic energy. A beam distribution is a numpy array with the coordinates of each particle. Various methods are available to generate distributions, including defining them based on Gaussian mean and standard deviation values, Twiss parameters, Σ-matrix, or from a file such as a CSV or parquet file.

```
kin = georges.Kinematics(230*_ureg.MeV)
gdist = georges.Distribution
beam = gdist.from_5d_sigma_matrix(**kwargs)
mi_beam = Beam(kinematics=kin, distribution=
    ↪ beam.distribution.values)
```

2.1.4. Observers

By default, Manzoni does not provide output data during a tracking simulation. The user must add an "observer" at the simulation to obtain information on the beam properties. The syntax to use is the following:

```
mi.track(beam=beam, observers=observers)
```

Different observers are implemented in Manzoni to analyze the beam at the entry and the exit of each element during the tracking and assess several properties such as the beam size (1σ), the losses, the centroid, the symmetry, the Twiss parameters or directly all the coordinates of each particle. The code architecture allows choosing at which positions the user wants to observe the beam by giving the name of the corresponding element(s) inside the beamline.

```
observer = BeamObserver()
```

[1] A TFS (Table File System) file in MAD-X is a text file that contains a table of data used to describe the properties of a beamline or other accelerator components.

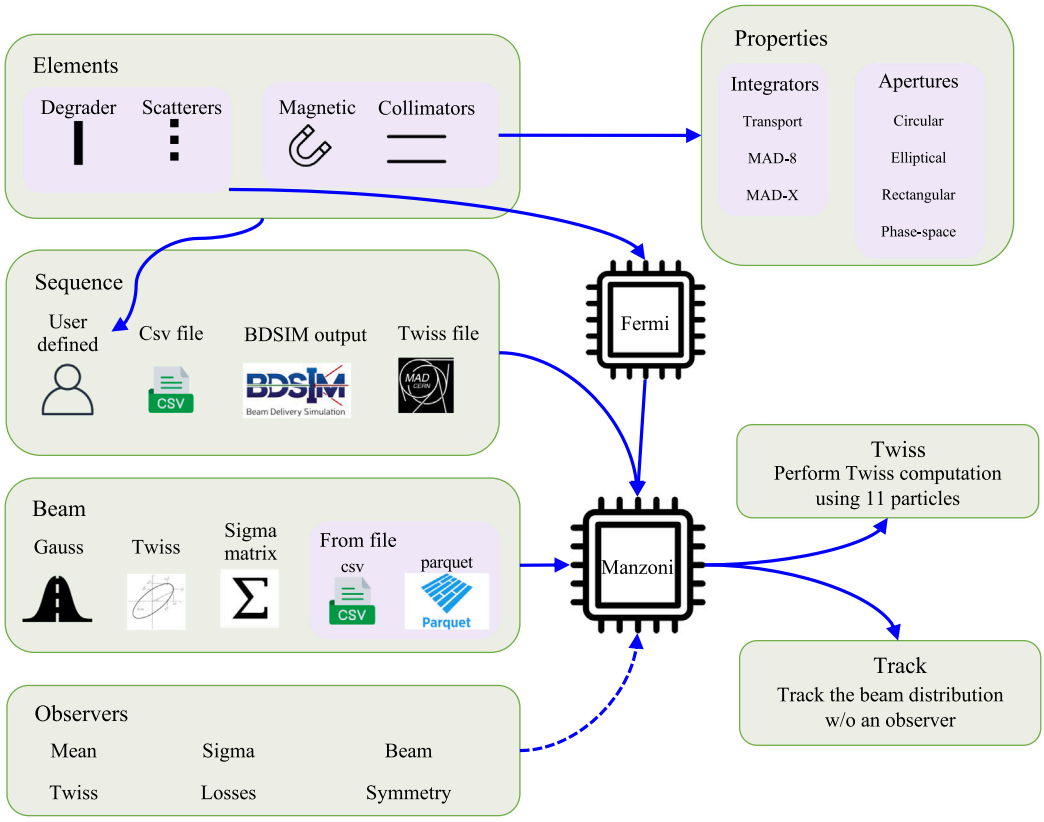

Fig. 1. Structure of the Python library Georges.

2.2. Software functionalities

All these objects are then sent to Manzoni for tracking or Twiss computation. The "track" method contains the loop over the different elements, with the option to check the apertures to select the particles that survive the tracking at the end of each component, and the use of "observers" is defined by the user to save data during the tracking. The Twiss method allows computing the matrix elements of all the elements along a given beamline for the Twiss functions calculation based on the 11 particles method [10]. Finally, the results are displayed using Matplotlib or Plotly. The user can superimpose the cartouche to the plot, corresponding to the element's position along the S coordinates (see Fig. 3 for example).

```
mi = Input.from_sequence(sequence=seq)
mi.track(beam=beam)
manzoni_plot = vis.ManzoniMatplotlibArtist()
manzoni_plot.plot_beamline(seq.df)
manzoni_plot.tracking(obervers)
```

3. Illustrative examples

3.1. Fermi implementation

As described in detail in Ref. [7], the Fermi module allows the computation of the properties of the beam after the passage in the material. In particular, it is possible to calculate the beam size as a function of the thickness of the material and to estimate the energy degradation induced. Fig. 2 shows the validation of the module's implementation via a comparison between the results obtained with the Fermi module and the results described in Ref. [6]. The code for the validation of the module is depicted below.

```
material = georges.fermi.materials.Beryllium
mrange = material.range(158.6 * _ureg.MeV)
depth = np.logspace(-3, -0.01, 10) * mrange
epost = list(map(lambda e: material.stopping
    ↪ (thickness=e,kinetic_energy=158.6*
    ↪ _ureg.MeV).ekin.m_as("MeV"), depth))
angle = list(map(lambda e: material.
    ↪ scattering(thickness=e,kinetic_energy
    ↪ =158.6 * _ureg.MeV)["A"][0], depth))
```

3.2. Manzoni verification with the MAD-X tracking code

It is a common practice to validate tracking software using the "golden standard" of the field, namely MAD-X. In this study, we define a beamline of dipoles and quadrupoles, which is converted into a MAD-X input, and the Python library cpymad [11] is used to compute the Twiss parameters along the beamline. The results obtained with MAD-X and Manzoni are shown in Fig. 3 and, an excellent agreement between both software is observed.

3.3. Performances

Manzoni is designed to optimize a beam transport line, making computation time a crucial parameter for the design process. To decrease the computational time, Manzoni uses the Numba package that allows to compile Python code into optimized code machine which significantly improves the performance of numerical operations and loops. We evaluate the computation time for tracking particles along the beamline described in the previous section as a function of the number of particles, and the results are presented in Fig. 4. This simulation uses a MacBook Pro with a 2.8 GHz Quad-Core Intel i7 processor. The results show that the implementation of the software (including the

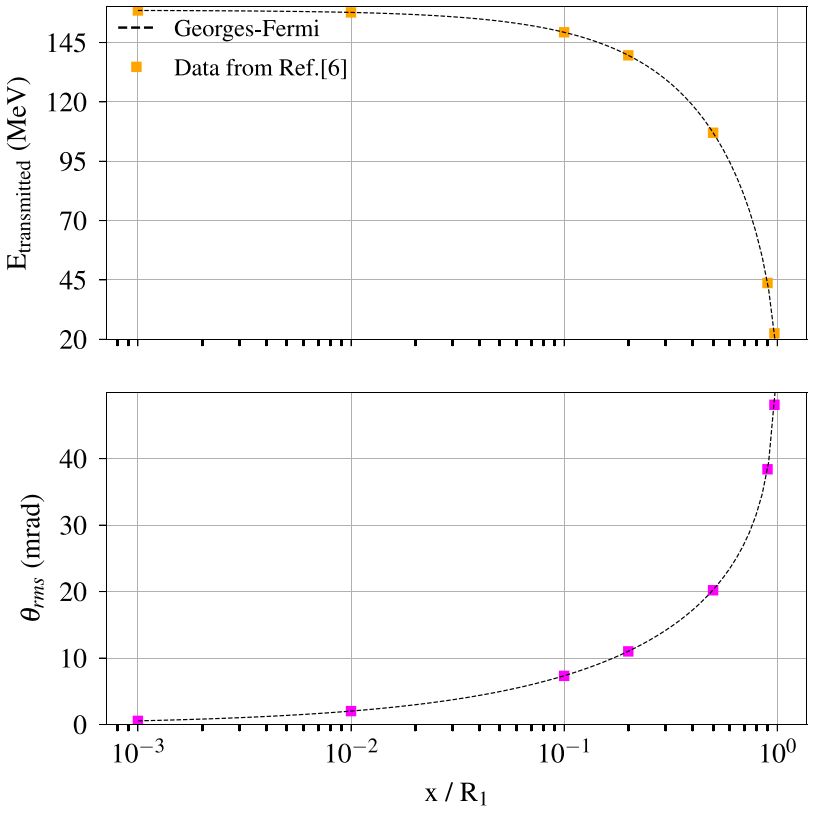

Fig. 2. Validation of the Fermi module (dashed line) by comparison with data from Ref. [6] (square). The energy degradation and the scattering angle are successfully computed with the Fermi module.

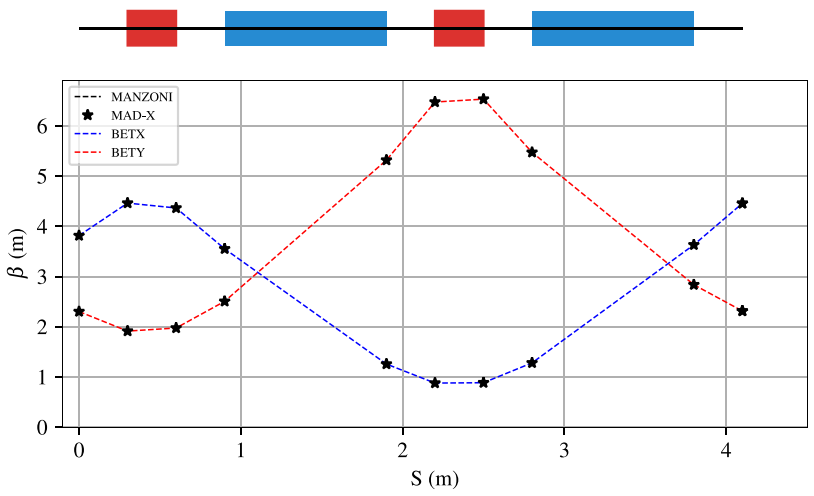

Fig. 3. Verification of Manzoni (dashed line) with the golden standard code `MAD-X` (black stars) on a simple beamline made of 2 quadrupoles and 2 dipoles. The horizontal (in blue) and vertical (in red) β-functions are in excellent agreement for both software. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

choice of Numba) meets the runtime requirements (sub-second runtime on "commodity" computers like laptop) of the targeted application scenario. With this relatively short computation time (~0.5 s), Manzoni is well-suited for conducting optimization studies.

3.4. BDSIM comparison

As a final step of the complete validation of the Georges library, we applied its different features to the modeling of the Ion Beam Applications (IBA) Proteus®One (P1) system. The P1 system is a compact single-room proton therapy facility equipped with modern pencil beam scanning techniques. It comprises a Synchro-cyclotron (S2C2), followed by an energy degradation system consisting of a wheel made of three materials: beryllium, graphite, and aluminum [12]. The system also includes a circular collimator and a rotating gantry with quadrupoles, dipoles, and kickers, efficiently transporting transport the beam to the desired position at the isocenter. As the P1 system combines energy degradation with beam tracking through several magnet types, it is an

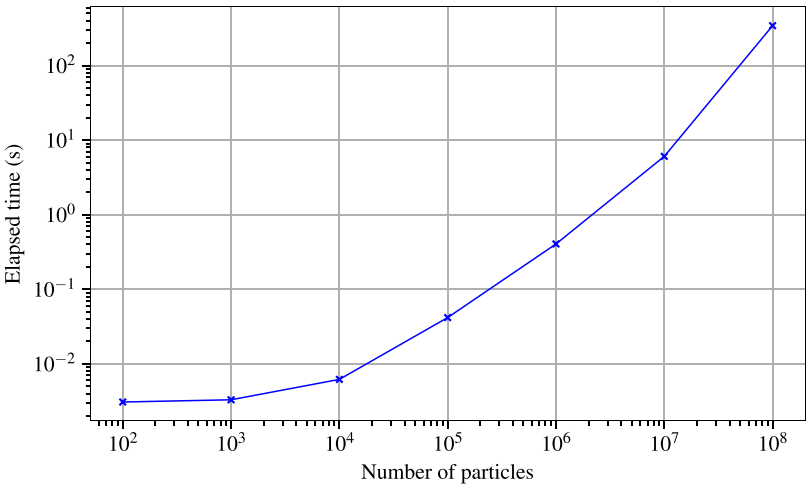

Fig. 4. Elapsed time as a function of the number of tracked particles. The line guides the eye.

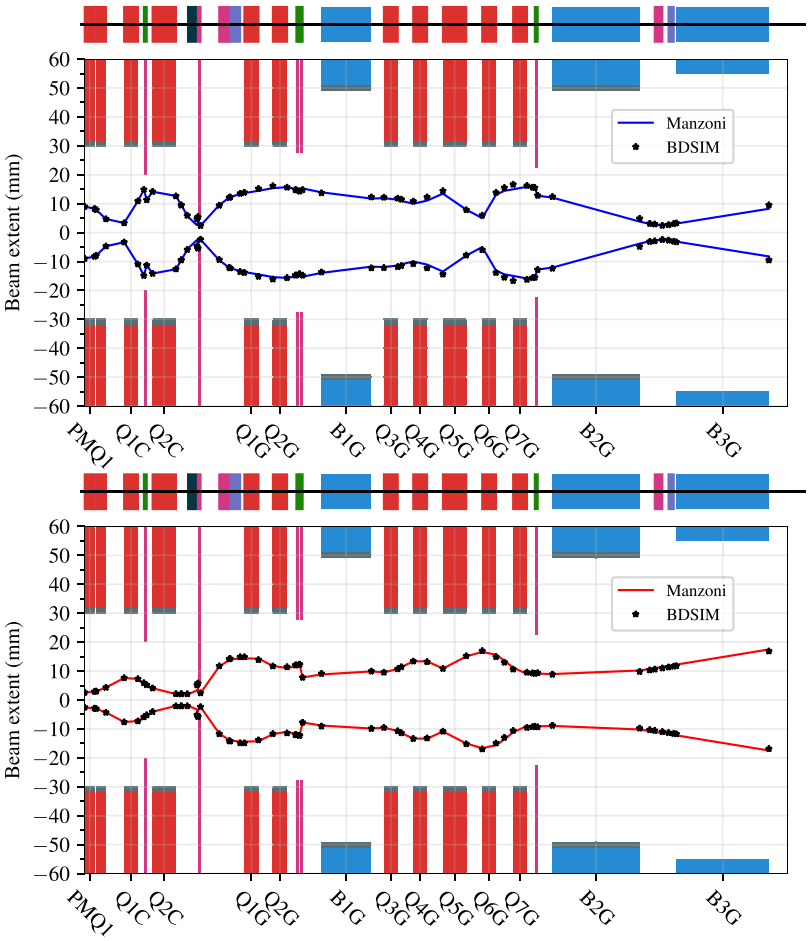

Fig. 5. Validation of Georges through a comparison of the beam size along the IBA Proteus®One system where the beam is propagated with energy degradation and losses are induced. Georges is able to give the same beamsize and therefore the Fermi and Manzoni modules are validated. The black stars are the results obtained with BDSIM and the blue and red lines are determined using the Python library Georges. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

ideal use case for self-consistent validation of both the Fermi and Manzoni modules of Georges. We compared in Fig. 5 the evolution of the beam size (1σ) along the horizontal and vertical axes of the transverse plane as calculated with Georges (blue and red lines) to the results obtained from the particle tracking and beam–matter interactions code `BDSIM` (depicted by black stars). The nominal energy at the exit of the degrader is 70 MeV, which is the lowest possible with the P1 system. An excellent agreement is observed between both codes, validating the capability of Georges to model low-energy beam transport systems accurately. It is also important to mention that the BDSIM simulation takes hours to run while Manzoni computes the standard deviation in about 150 ms.

4. Impact

Fast and accurate tracking codes are crucial for designing and optimizing charged particle therapy beamlines for medical applications. In this regard, the Georges library is a valuable tool for various studies in the field of medical accelerators and has already been applied to optimize most modern proton therapy systems [13–16].

With the Manzoni module's fast-tracking capabilities, the clinical performances of most proton therapy systems can be studied and improved through simulation. To illustrate, Georges has been coupled with optimization routines, enabling the determination of the optimal beamline configuration required to deliver an efficient and clinically acceptable beam at the treatment room entrance, all while minimizing beamline losses [13].

To facilitate the use of the Georges library, we have developed several tools that simplify the installation process and allow for direct usage via Docker or jupyter-lab. The Python code of Georges is open source and available on GitHub for researchers.

5. Conclusions

Existing tracking codes like `MAD-X` or `TRANSPORT` cannot accurately model the propagation of charged particles considering energy degradation. On the other hand, Monte-Carlo codes such as `BDSIM` are not designed to perform optimization of an existing beamline due to the extensive computation time, *e.g.,* depending on the application, a Monte-Carlo simulation could take more than one day (compared to 150 ms for Manzoni as written above) to obtain results with a small standard deviation. Typically, a Monte-Carlo simulation is often performed with high-performance computing (HPC) clusters. The Georges library offers users a straightforward and effective way of propagating particles through beamlines while simultaneously simulating the beam–matter interaction using the Fermi module. Thanks to its fast computation via the Just in Time compiler, it is also possible to use Georges to optimize beamlines to limit losses, for example. The library is also easy to use since it can be used via a Docker and interfaces easily with jupyter-lab. Future developments include creating a graphical interface that will make Georges more user-friendly.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work has received funding from the Walloon Region (SPW-EER) Win^{2}Wal program under grant agreement No. 1810110 and PIT Prother-wal program under grant agreement No. 7289. M. Vanwelde is a Research Fellow of the Fonds de la Recherche Scientifique - FNRS. The authors are thankful for the support provided by Ion Beam Applications (IBA), Belgium and for the numerous discussions on the modeling of the Proteus®One.

References

[1] Rohrer U. PSI graphic transport framework. 2007, URL http://aea.web.psi.ch/Urs_Rohrer/MyWeb/trans.htm.

[2] Brown K, et al. TRANSPORT: a computer program for designing charged-particle beam-transport systems. Tech. rep., CERN; 1980, http://dx.doi.org/10.5170/CERN-1980-004.

[3] Deniau L, et al. Methodical accelerator design (MAD-X). 2023, http://dx.doi.org/10.5281/zenodo.7900976, URL https://mad.web.cern.ch.

[4] Iselin F. The MAD program. 1992, URL https://mad8.web.cern.ch/mad8/doc/phys_guide.pdf.

[5] Brown K, Iselin C. DECAY TURTLE (Trace Unlimted Rays Through Lumped Elements): a computer program for simulating charged-particle beam transport systems, including decay calculations. Tech. rep., CERN; 1974, http://dx.doi.org/10.5170/CERN-1974-002.

[6] Gottschalk B. On the scattering power of radiotherapy protons. Med Phys 2009;37(1):352–67. http://dx.doi.org/10.1118/1.3264177.

[7] Hernalsteens C, Tesse R, Gnacadja E, Pauly N, Ramoisiaux E, Saghir Y, et al. A hybrid numerical approach to the propagation of charged particle beams through matter for hadron therapy beamline simulations. Nucl Instrum Methods Phys Res B 2022-11;531:56–64. http://dx.doi.org/10.1016/j.nimb.2022.09.005.

[8] Lam SK, Pitrou A, Seibert S. Numba: A LLVM-based python JIT compiler. In: Proceedings of the second workshop on the LLVM compiler infrastructure in HPC. 2015, p. 1–6.

[9] Nevay L, et al. BDSIM: An accelerator tracking code with particle–matter interactions. Comput Phys Comm 2020;252:107200. http://dx.doi.org/10.1016/j.cpc.2020.107200.

[10] Méot F. The ray-tracing code Zgoubi. Nucl Instrum Methods Phys Res A 1999;427(1–2):353–6. http://dx.doi.org/10.1016/s0168-9002(98)01508-3.

[11] Gläßle T, et al. Cpymad. 2023, http://dx.doi.org/10.5281/zenodo.7904100, URL https://hibtc.github.io/cpymad/index.html.

[12] Tesse R, Dubus A, Pauly N, Hernalsteens C, Kleeven W, Stichelbaut F. Numerical simulations to evaluate and compare the performances of existing and novel degrader materials for proton therapy. J Phys Conf Ser 2018;1067:092001. http://dx.doi.org/10.1088/1742-6596/1067/9/092001.

[13] Gnacadja E, Hernalsteens C, Boogert S, Flandroy Q, Fuentes C, Nevay LJ, et al. Optimization of proton therapy eye-treatment systems toward improved clinical performances. Phys Rev Res 2022-02;4(1):013114. http://dx.doi.org/10.1103/physrevresearch.4.013114.

[14] Hernalsteens C, Tesse R, Boogert ST, Flandroy Q, Fuentes C, Gnacadja E, et al. A novel approach to seamless simulations of compact hadron therapy systems for self-consistent evaluation of dosimetric and radiation protection quantities. Europhys Lett 2020;132(5):50004. http://dx.doi.org/10.1209/0295-5075/132/50004.

[15] Gnacadja E, Boogert S, Hernalsteens C, Nevay L, Pauly N, Shields W, et al. Study of a proton therapy beamline for eye treatment with beam delivery simulation (BDSIM) and an in-House tracking code. In: Proceedings of the 10th int. particle accelerator conf. 2019, p. 3088–91. http://dx.doi.org/10.18429/JACOW-IPAC2019-WEPTS002.

[16] Gnacadja E, Hernalsteens C, Pauly N, Ramoisiaux E, Tesse R, Vanwelde M. Upgrade of a proton therapy eye treatment nozzle using a cylindrical beam stopping device for enhanced dose rate performances. J Phys Conf Ser 2023-01;2420(1):012095. http://dx.doi.org/10.1088/1742-6596/2420/1/012095.