ISBN: 978-3-95450-232-5 ISSN: 2673-7000 doi:10.18429/JACoW-NAPAC2022-WEPA24

pyJSPEC - A PYTHON MODULE FOR IBS AND ELECTRON COOLING SIMULATION*

H. Zhang[†], M.W. Bruker, Y. Zhang, S.V. Benson
Thomas Jefferson National Accelerator Facility, Newport News, VA, USA

Abstract

The intrabeam scattering is an important collective effect that can deteriorate the property of a high-intensity beam and electron cooling is a method to mitigate the IBS effect. JSPEC (JLab Simulation Package on Electron Cooling) is an open-source C++ program developed at Jefferson Lab, which simulates the evolution of the ion beam under the IBS and/or the electron cooling effect. The Python wrapper of the C++ code, pyJSPEC, for Python 3.x environment has been recently developed and released. It allows the users to run JSPEC simulations in a Python environment. It also makes it possible for JSPEC to collaborate with other accelerator and beam modeling programs as well as plentiful python tools in data visualization, optimization, machine learning, etc. In this paper we will introduce the features of pyJSPEC and demonstrate how to use it with sample codes and numerical results.

INTRODUCTION

Intrabeam scattering (IBS) [1] is one important problem that hadron collider designers need to consider. Due to small-angle collisions between the ions, the emittance and the momentum spread of the ion beam gradually increase and therefore the luminosity of the collider decreases. Electron cooling [2] is an experimentally proven leading method to reduce the ion beam emittance by overlapping the ion beam with a low-temperature electron beam while both beams co-move inside the cooler in the same velocity to allow thermal energy to transfer from the ion beam to the electron beam. It can be used to mitigate the IBS effect. JLab simulation package for electron cooling (JSPEC) is a program to simulate the effects of both IBS and electron cooling. Although originally designed to support the then on-going electron-ion collider (EIC) project [3] at Jefferson Lab, JSPEC has been extended to include most frequently used formulas for the friction forces and variant models of the electron/ion beam [4] and is provided to the community as an open source tool for IBS and electron cooling simulations. The program is developed using C++ in consideration of efficiency and has been tested on both MS Windows 10 and Ubuntu 18.04 systems. Most computations are parallelized for shared-memory systems using OPENMP to take advantage of the multi-core processors widely available in desktop and laptop computers. The source codes, the documents, and the examples are all available in the *github* repository [5]. An online JSPEC based on an earlier version has been developed by Radiasoft and is accessible through their cloud service SIREPO [6]. It allows one to run JSPEC and visualize the result inside a browser.

Although JSPEC runs independently, we see the need to simulate the IBS and electron cooling together with other collective effects, *e.g.* space charge effect and CSR effect, and the convenience of using optimization tools in accelerator design. This is the reason we developed pyJSPEC [7], which ports most functions in JSPEC to Python 3.x environment and brings the possibility to combine JSPEC with other Python tools for accelerator modeling and optimization.

FEATURES

The basic feature of JSPEC is to calculate the emittance growth rate of the ion beam under the IBS and/or the electron cooling effect. The rate at time t is defined as $r_i(t) = \frac{1}{\epsilon_i(t)}\frac{\mathrm{d}\epsilon_i(t)}{\mathrm{d}t}$, where $i = x, y, s$, representing the horizontal, vertical, and longitudinal direction, and ϵ_i is the emittance in the respective direction. For the IBS rate, JSPEC provides the Martini model [8], the original Bjorken-Mtingwa model [9] calculated by Nagaitsev's method [10], and the complete Bjorken-Mtingwa model with vertical dispersion and non-relativistic terms included [11]. The electron cooling rate is calculated statistically on a group of sample ions, each receiving a *kick* by the friction force. The rate is calculated as the relative change of the emittance per unit time before and after the kick. JSPEC provides several formulas [12-15] for both the non-magnetized and the magnetized friction force. Using different formulas in the transverse and the longitudinal direction is allowed.

JSPEC also simulates the evolution of the ion beam under the IBS effect and/or the electron cooling effect. The RMS dynamic model represents the ion beam by its macroscopic parameters, *i.e.* the emittances, the momentum spread, and the bunch length (for a bunched beam), calculates the instant expansion rate r at a time t and updates the parameters using $\epsilon_i(t+\Delta t) = \epsilon_i(t)\exp(r_i\Delta t)$ for the time step Δt. The particle model applies kicks due to IBS and electron cooling to sample ions and moves them by a random phase advance for the betatron and synchrotron oscillation in Δt. The turn-by-turn model is similar to the particle model but the betatron and synchrotron motion is modeled by a linear transfer matrix and the simulation is carried out in a turn-by-turn manner.

BENCHMARK

JSPEC has been benchmarked with BETACOOL [12] for various scenarios. The two programs agree well. For the typical simulations we have done for the EIC project, a significant improvement of efficiency has been achieved

* Work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177.
† hezhang@jlab.org

even without using multiprocessing in JSPEC. Parallel computation will further improve the efficiency.

We also compared JSPEC simulations with experimental data, obtained from the collaboration of Jefferson Lab in the U.S. and Institute of Modern Physics in China from 2016 to 2019. Figure 1 shows the cooling of the ^{86}Kr^{25+} beam with an energy of 5 MeV/nucleon using electron pulses with a length varying from 600 ns to 1000 ns. A longer pulse length means a longer overlap between the two beams and a stronger cooling, which is observed through the larger slope of the plots. The solid lines in the plot are the results from the simulation using the turn-by-turn model, the parameters used in which are listed in Table 1 [16]. The dots are experimental data. In all the cases, the simulation agrees with the experiment reasonably well.

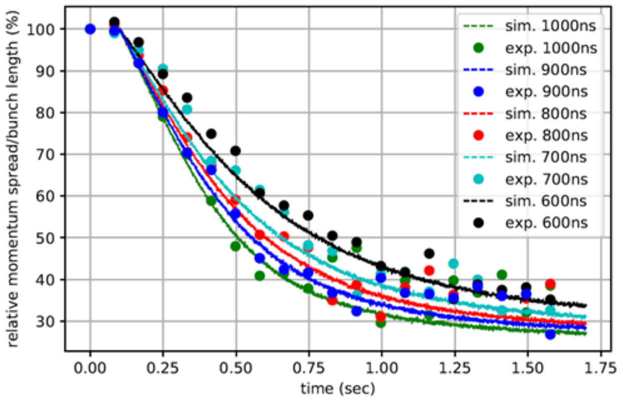


Figure 1: Cooling the ^{86}Kr^{25+} beam using pulsed electron beam, simulation (solid lines) and experiments (dots).

Table 1: Electron Cooling Simulation Parameters

e^- beam radius	15 mm
Cooler length	3.4 m
Magnetic field	0.1 T
β_x/β_y in the cooler	10/17 m
e^- beam peak current	30 mA
e^- beam temperatures	200/6 meV
Ion beam normalized emittance	0.6 mm mrad
Ion beam RMS bunch length	10.5 m
Ion beam RMS momentum spread	7×10^{-4}
Lorentz factor β/γ for both beams	0.103/1.005

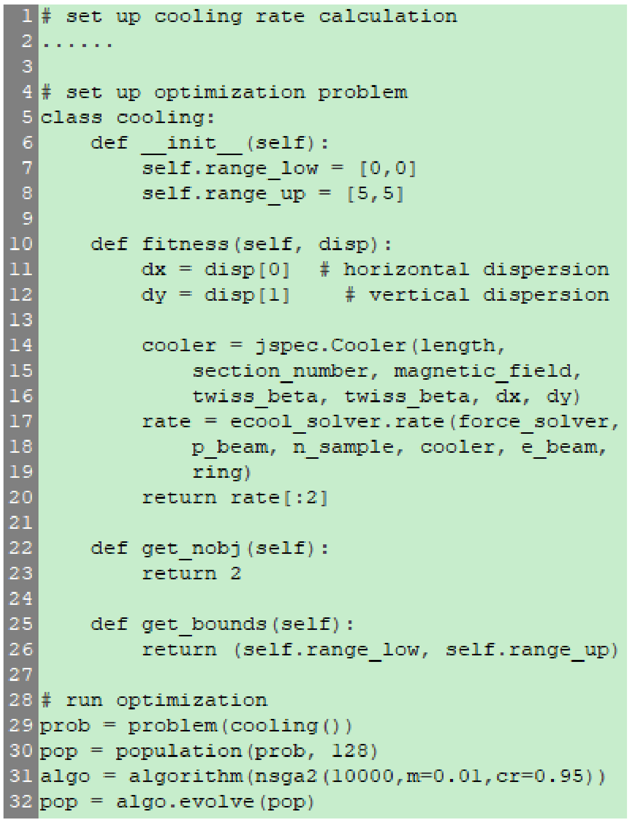
pyJSPEC

pyJSPEC is the Python wrapper for JSPEC, developed with Pybind 11 [17]. It is a library for Python 3.x. After importing the library, users get access to most of the JSPEC functions. Figure 2 shows an example on calculating the expansion rate due to both the IBS and the electron cooling respectively and carry out a dynamic simulation using the particle model. After importing JSPEC, we create the proton beam, the ring, the cooler, and the electron beam by calling the respective functions. We need to assign values to the arguments before calling the functions, which is omitted in the example to save space. Line 25 – 30 shows how to calculate the electron cooling rate. The Parkhomchuk formula is chosen for the friction force calculation and the expansion rate is calculated over 40,000 sample protons. Line 32 – 34 shows how to calculate the IBS expansion rate using the Bjorken-Mtingwa model. In line 37 – 39, sample protons are created for the following dynamic simulation using the particle model. This step can be omitted if the RMS dynamic model is used. In line 43 – 48, we set up the time to simulate and the number of steps, choose which effects to be included, and finally run the simulation.

```
 1 import jspec
 2
 3 # create the ion beam
 4 ...
 5 p_beam = jspec.Beam(n_charge, n_mass,
 6     ke, ex, ey, dp, ds, np)
 7
 8 # create the ring
 9 lat = jspec.Lattice("lattice.tfs")
10 ring = jspec.Ring(lat, p_beam)
11
12 # create the cooler
13 ...
14 cooler = jspec.Cooler(length, section_number,
15     magnetic_field, twiss_beta, twiss_beta,
16     dx, dy)
17
18 # create electron bunch
19 ...
20 e_beam = jspec.GaussianBunch(ne, sigma_x,
21     sigma_y, sigma_z)
22 e_beam.set_gamma(gamma)
23 e_beam.set_tpr(0.5, 0.1)
24
25 # calculate electron cooling rate
26 force_solver = jspec.ForcePark()
27 n_sample = 40000
28 ecool_solver = jspec.ECool()
29 rate = ecool_solver.rate(force_solver, p_beam,
30     n_sample, cooler, e_beam, ring)
31
32 # calculate IBS rate
33 ibs_solver = jspec.IBSSolver_BM(log_c)
34 rate = ibs_solver.rate(lat, p_beam)
35
36 # create proton samples
37 p_samples = jspec.Ions_MonteCarlo(n_sample)
38 p_samples.set_twiss(cooler)
39 p_samples.create_samples(p_beam)
40
41 # run simulation
42 ...
43 simulator = jspec.ParticleModel(time, n_step)
44 simulator.set_ibs(True)
45 simulator.set_ecool(True)
46 simulator.run(p_beam, p_samples, cooler,
47     e_beam, ring, ibs_solver, ecool_solver,
48     force_solver)
```

Figure 2: pyJSPEC simulation.

JSPEC works together with other Python libraries. Figure 3 shows an example, in which we use NSGA_II, a non-dominated sorting-based multi-objective evolutionary optimizer from the pygmo library [18], together with JSPEC. We have seen that introducing transverse dispersions at the cooler for the ion beam can transfer the cooling in the longitudinal direction to the transverse directions. A dispersion in one transverse direction can increase the cooling in the same direction but may reduce the cooling in the other transverse direction. In the example, we ask the optimizer to find better values of the dispersions within 0 to 5 meters to obtain stronger cooling in both the horizontal and the vertical directions. First we set up the cooling rate calculation as usual. Then we define a problem, in which the fitness function takes the dispersions as input and outputs the new cooling rates. In the end, we create the problem for pygmo, select the NSGA_II optimizer, and run it for 10,000 generations. Figure 4 shows the initial population.



```
# set up cooling rate calculation
......

# set up optimization problem
class cooling:
    def __init__(self):
        self.range_low = [0,0]
        self.range_up = [5,5]

    def fitness(self, disp):
        dx = disp[0]  # horizontal dispersion
        dy = disp[1]    # vertical dispersion

        cooler = jspec.Cooler(length,
            section_number, magnetic_field,
            twiss_beta, twiss_beta, dx, dy)
        rate = ecool_solver.rate(force_solver,
            p_beam, n_sample, cooler, e_beam,
            ring)
        return rate[:2]

    def get_nobj(self):
        return 2

    def get_bounds(self):
        return (self.range_low, self.range_up)

# run optimization
prob = problem(cooling())
pop = population(prob, 128)
algo = algorithm(nsga2(10000,m=0.01,cr=0.95))
pop = algo.evolve(pop)
```

Figure 3: JSPEC with pygmo NSGA_II optimizer.

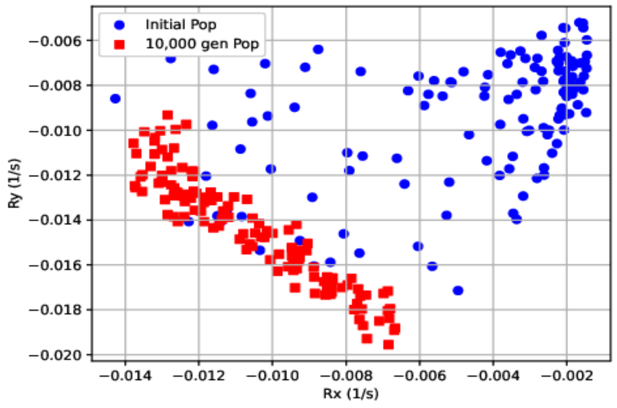


Figure 4: Initial population and the population after 10,000 generation of evolutions.

by blue circles, which is randomly generated, and the final population by red squares. The horizontal/vertical axis shows the horizontal/vertical expansion rate. The negative sign means the emittance reduces. Clearly the optimizer pushes the population to an area with larger cooling rates.

Same with the C++ JSPEC, pyJSPEC also supports parallel computation on the shared-memory structure with OPENMP. The users have the choice to compile a parallel pyJSPEC by adding the flag *OMPFLAGS=-fopenmp* to the *make* command. By default, the parallel version will use all the available threads. But the users can set the number of threads to use. Table 2 shows an example, in which the electron cooling process together with the IBS effect for a proton beam is simulated for 50,000 steps using 40,000 particles on a personal computer running an Intel i7-4820k CPU with four cores and eight hyperthreads. When four threads are used, the computation time is reduced by slightly more than 40%. But the hyperthreads do not help.

Table 2: Computation Using Multi-Threads

No. of threads	Time (s)	No. of threads	Time (s)
1	1085	5	742
2	779	6	864
3	663	7	957
4	648	8	2643

SUMMARY AND FUTURE WORK

JSPEC is an open-source program developed at Jefferson Lab for IBS and electron cooling simulations. It has been benchmarked with other programs and experimental data. The source code, manuals, and examples can be found in the *github* repository. A Python wrapper, pyJSPEC, has been developed and most functions from JSPEC has been ported to Python 3.x environment. We have shown examples on how to perform IBS and electron cooling simulations in Python and how to use an optimizer together with pyJSPEC.

We are currently trying to make JSPEC adapted for BMAD [19], using the C++ interface in BMAD code. BMAD is a widely used accelerator modeling code and it provides many functions for lattice design, particle tracking, beam tracking, spin tracking, space charge effect simulation, CSR effect simulation, *etc*. When JSPEC works together with BMAD, we will be able to include a better dynamic model of the accelerator and some other dynamic effects in IBS and electron cooling simulations.

ACKNOWLEDGMENT

The authors thank all the users for their valuable feedback and thank David Bruhwiler, Paul Moeller, Ilya Pogorelov, and Stephen Coleman at Radiasoft for fruitful discussions and for bringing JSPEC to SIREPO.

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177.

REFERENCES

[1] A. Piwinski, "Intra-beam scattering", in *Proc. 9th Int. Conf. on High Energy Accelerators*, Stanford, CA, USA, 1974, p. 405.

[2] G. Budker *et al.*, "Experimental studies of electron cooling", *Part. Accel.*, 1976, pp. 197-121.

[3] S. Abeyratne *et al.*, "MEIC design summary", 2015. doi:10.48550/arXiv.1504.07961

[4] H. Zhang *et al.*, "Electron cooling simulation code development", Rep. JLAB-TN-21-002, Jefferson Lab, 2021.

[5] JSPEC, https://github.com/JeffersonLab/ElectronCooling

[6] SIREPO, https://www.radiasoft.net/sirepo

[7] pyJSPEC, https://github.com/zhanghe9704/jspec2-python

[8] M. Martini, "Intrabeam scattering in the ACOL-AA machines", Rep. CERN-PS-8-4-9-AA, CERN, 1984.

[9] J. Bjorken and S. Mtingwa, "Intrabeam scattering", *Part. Accel.*, vol. 13, pp. 115-143, 1982.

[10] S. Nagaitsev, "Intrabeam scattering formulas for fast numerical evaluation", *Phys. Rev. ST Accel. Beams*, vol. 8, p. 064403, 2005.

[11] F. Zimmermann, "Intrabeam scattering with non-ultrarelativistic corrections and vertical dispersion for MAD-X", Rep. CERN-AB-2006-002, CERN, 2005.

[12] I. Meshkov *et al.*, "BETACOOL physics guide", Joint Institute for Nuclear Research, Dubna, Russian Federation, 2007.

[13] I. Meshkov, "Electron cooling: status and perspectives", *Phys. Part. Nucl.*, vol. 25, pp. 631-661, 1994.

[14] V. Parkhomchuk, "New insights in the theory of electron cooling", *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 441, pp. 9-17, 2000.

[15] Y. Derbenev and A. Skrinsky, "The effect of an accompanying magnetic field on electron cooling", *Part. Accel.*, vol 8, pp. 235-243, 1978.

[16] M. Bruker *et al.*, "Demonstration of electron cooling using a pulsed beam from an electrostatic electron cooler", *Phys. Rev. Accel. Beams*, vol. 24, p. 012801, 2021.

[17] Pybind11, https://pybind11.readthedocs.io

[18] pygmo, https://doi.org/ 10.5281/zenodo.4585131

[19] BMAD, https:// www.classe.cornell.edu/bmad/