JSPEC : A PROGRAM FOR IBS AND ELECTRON COOLING SIMULATION

H. Zhang, Y. Zhang, S. Benson, M. Bruker
Jefferson Lab, Newport News, VA 23606, USA

Abstract

JSPEC (JLab Simulation Package on Electron Cooling) is an open-source C++ program developed at Jefferson Lab to simulate the evolution of the ion beam under the intrabeam scattering effect and/or the electron cooling effect. JSPEC includes various models of the ion beam, the electron beam, and the friction force, aiming to reflect the latest advances in the field and to provide a useful tool to the community. JSPEC has been benchmarked against other cooling simulation codes and experimental data. A Python wrapper for Python 3.x environment has also been developed, which allows users to run JSPEC simulations in a Python environment and 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. A Fortran interface is being developed, aiming at seamless call of JSPEC functions in Fortran.

Introduction

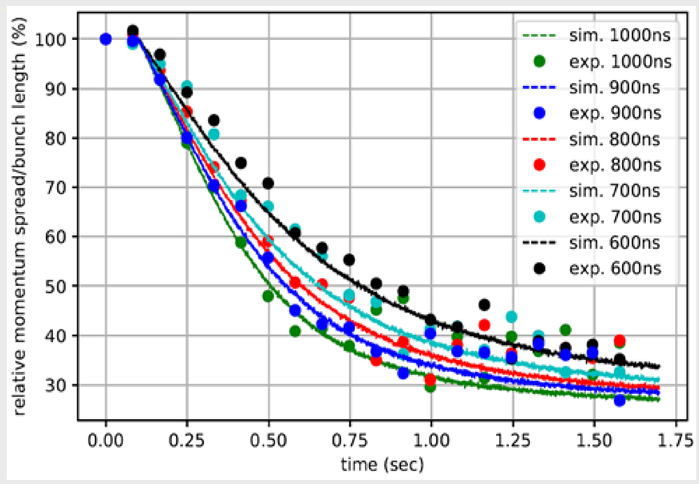
Jlab Simulation Package for Electron Cooling (JSPEC)

- Efficient C++ code for intrabeam scattering (IBS) effect and electron cooling simulation.
- Ion beam model: coasting or bunched
- Electron beam model: DC or bunched with various shapes, *e.g.* Gaussian, beer can, hollow beam, *etc.* User-defined arbitrary shape is also supported.
- Various formulas for friction force calculation in magnetized cooling or non-magnetized cooling.
- Supports ion beam dispersion and electron beam dispersion.
- Source code and documents available at *github*: https://github.com/zhanghe9704/electroncooling
- Support parallel computation in shared-memory structure with OPENMP
- pyJSPEC port most JSPEC functions to Python 3.x environment. https://github.com/zhanghe9704/jspec2-python
- A FORTRAN interface is under development.
- Under active maintenance and development.

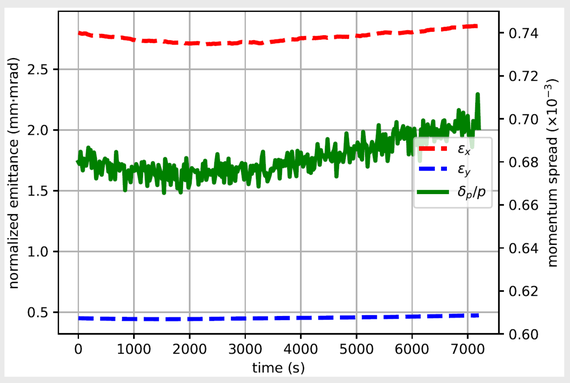
Benchmark with Experiments

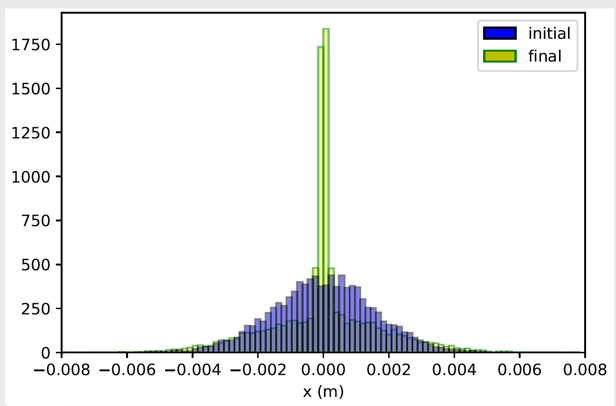
Cooling of $^{86}Kr^{25+}$ beam (5 MeV/nucleon) using electron pulses with the length from 600 ns to 1000 ns in collaboration of Jefferson Lab (US) and Institute of Modern Physics (China) from 2016 to 2019.

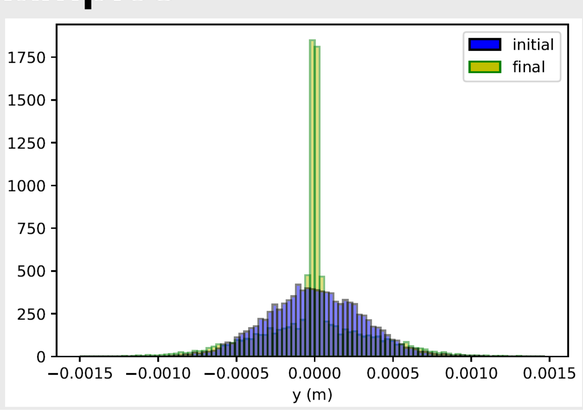
Simulations and experiments agree well.

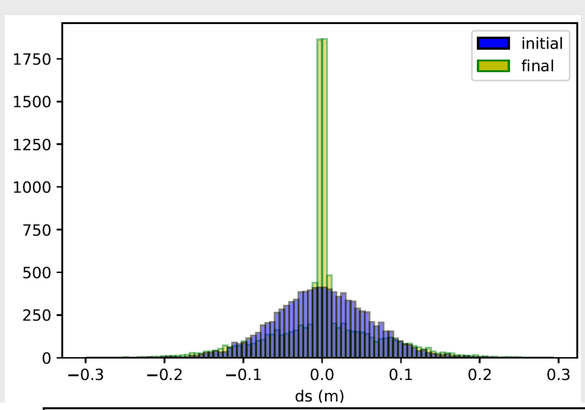


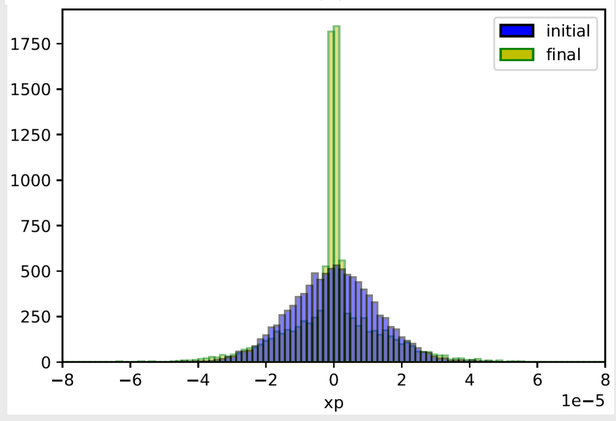
JSPEC Examples

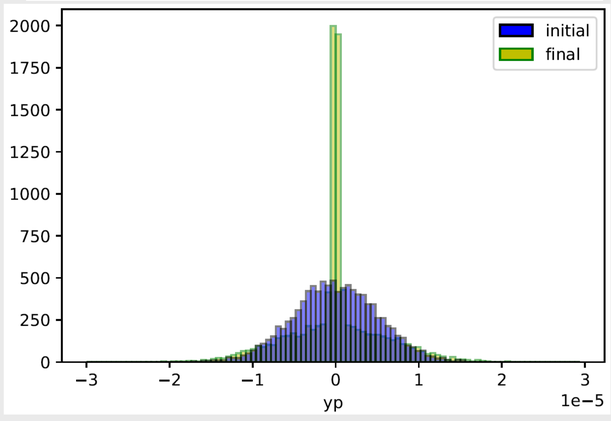


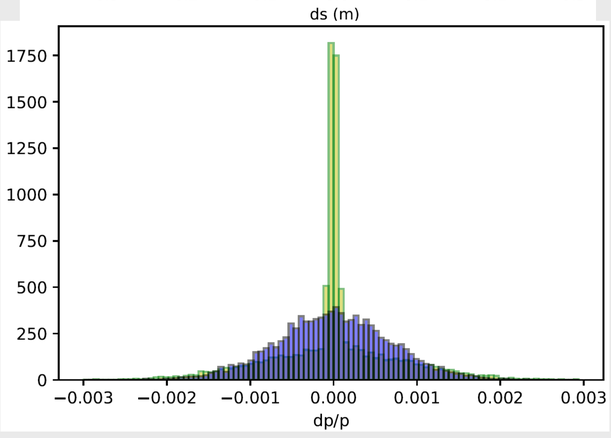












JSPEC

```
section_scratch
        ion_mass = 938.2720882
        ion_ke = 274061.728
        ion_gamma = 1 + ion_ke/ion_mass
section_ion
        charge_number = 1
        mass =  ion_mass
        kinetic_energy = ion_ke
        norm_emit_x = 2.8e-06
        norm_emit_y = 4.5e-07
        momentum_spread = 0.00068
        particle_number = 6.9e10
        rms_bunch_length = 0.06
section_ring
        lattice = ring-lattice.txt
section_ibs
        model = bm
        log_c = 20.0
        coupling = 0.2
section_cooler
        length = 120
        section_number = 1
        magnetic_field = 4
        bet_x = 60
        bet_y = 60
        disp_x = 2.0
section_e_beam
        gamma = ion_gamma
        tmp_tr = 4665.2223
        tmp_l = 0.2798
        shape = bunched_gaussian
        sigma_x = 0.00074
        sigma_y = 0.00016
        sigma_z = 0.025
        e_number = 4.14e11
section_ecool
        sample_number = 10000.0
        force_formula = PARKHOMCHUK
section_run
        create_ion_beam
        create_ring
        create_e_beam
        create_cooler
        set_n_thread 4
        calculate_ibs
        calculate_ecool
        total_expansion_rate
section_simulation
        ibs = on
        e_cool = on
        time = 7200.0
        step_number = 3600
        model = particle
section_run
        run_simulation
```

* Parameters in the above simulation come from Fanglei Lin's talk on Monday 2:30 PM: Perspective of a Dual Energy Storage Ring Cooler for Hadron Beam Cooling.

Python

```
import jspec

#define variables
...

#create the ion beam
p_beam = jspec.Beam(n_charge, n_mass, ke, ex, ey, dp, ds, np)

#create the ring
lat = jspec.Lattice("test.tfs")
ring = jspec.Ring(lat, p_beam)

#create the cooler
cooler = jspec.Cooler(length, section_number, magnetic_field,
            twiss_beta, twiss_beta, dx, dy)

#create electron bunch
e_beam = jspec.GaussianBunch(ne, sigma_x, sigma_y, sigma_z)
e_beam.set_gamma(gamma)
e_beam.set_tpr(t_tr, t_l)

#calculate electron cooling rate
force_solver = jspec.ForcePark() #Parkhomchuk formula
n_sample = 40000
ecool_solver = jspec.ECool()
rate = ecool_solver.rate(force_solver, p_beam, n_sample, cooler, e_beam, ring)

#calculate IBS rate
log_c = 20.2
ibs_solver = jspec.IBSSolver_BM(log_c)
rate = ibs_solver.rate(lat, p_beam)

#create ion samples
p_samples = jspec.Ions_MonteCarlo(n_sample)
p_samples.set_twiss(cooler)
p_samples.create_samples(p_beam)

#run simulation
simulator = jspec.ParticleModel(time, n_step)
simulator.set_ibs(True)
simulator.set_ecool(True)
simulator.run(p_beam, p_samples, cooler, e_beam, ring, ibs_solver,
            ecool_solver, force_solver)
```

Fortran

```
program main
    use jspec
    use iso_c_binding
    implicit none

    type(Beam) :: my_beam
    type(Lattice) :: my_lattice
    type(Ring) :: my_ring
    type(Cooler) :: my_cooler
    type(FrictionForceSolver) :: my_force_solver
    type(EBeam) :: my_ebeam
    type(ECoolRate) :: rate_ec

    ! Define variables
    integer(c_int) :: charge = 1
    real(c_double) :: mass = 938.272
    ...
    ! Create ion beam
    my_beam = create_beam(charge, mass, ke, ex, ey, dp, ds, np)

    ! Create the lattice from some file
    my_lattice = create_lattice("lattice.txt")

    ! Create a ring
    my_ring = create_ring(my_lattice, my_beam)

    ! Create a cooler
    my_cooler = create_cooler(length, n_section, mag_field, &
            twiss_beta, twiss_beta)

    ! Create friction force solver
    my_force_solver = create_force_solver(PARKHOMCHUK)

    ! Create electron beam with Gaussian distribution
    my_ebeam = create_gaussian_bunch(ne, sigma_x, sigma_y, ds)
    call ebeam_set_gamma(my_ebeam, gamma)
    call ebeam_set_temperature(my_ebeam, tmp_tr, tmp_l)

    ! Calculate cooling rate
    rate_ec = create_ecool_rate_calculator()
    call ecool_rate(rate_ec, my_force_solver, my_beam, n_sample, &
        my_cooler, my_ebeam, my_ring, rx, ry, rs)
end program main
```

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.