Tutorial

Environment

This code is devloped by C++. Users need to write their own "main" program. To compile the code, please make sure your compiler supports c++ 11 and the option for c++ 11 is turned on. For example, when using GCC, one needs to add "-std=c++11" into the compiler options. A Code Blocks Project (cbp) file is included in the subfolder cooling_release, and the Code Blocks IDE with a proper compiler can be used to compiler the project. Makefiles for multiple platform can be generated using the tool "cbp2make".

Constants

The following physical and mathematical constants are defined in "**constants.h**":

```
const double k_ke = 8.9875517873681764E9;   //Coulomb's constant, in N*m^2/C^2
const double k_c = 299792458.0;             //speed of light, in m/s
const double k_e = 1.602176565E-19;         //Elementary charge, in C
const double k_pi = 3.1415926535897932384626;
const double k_u = 931.49406121;            //Atomic mass unit, in MeV/c^2
const double k_me = 0.510998928;            //electron mass, in MeV/c^2
```

Global variables

For intrabeam scattering (IBS) and/or electron cooling dynamic simulations, it is required to use the following variables to save configurations of respective calculations. (Will be explained in the following sections.) The declaration of them, as shown in the code block below, should be included in the main program. For rate calculation, they can be used, but not required.

```
extern IBSParas * ibs_paras;
extern EcoolRateParas * ecool_paras;
extern ForceParas * force_paras;
extern DynamicParas * dynamic_paras;
//Only used for IBS simulation (without electron cooling) using model beam method
extern Twiss * twiss_ref;
extern int n_ion_model;
```

Define the ion beam

The class **Beam** is provided in the "**beam.h**". An ion beam can be defined using this class. The user should set value for the following paramters:

- charge number of the ion
- mass number of the ion
- kinetic energy in MeV
- normalized horizontal emittance in m*rad
- normalized vertical emittance in m*rad

- mommentum spread dp/p
- rms bunch length (for bunched beam only)
- number of ions in the beam (for coasting beam) or in each bunch (for bunched beam)

The following code defines a bunched proton beam:

```
double m0, KE, emit_nx0, emit_ny0, dp_p0, sigma_s0, N_ptcl, A;
int Z;
Z = 1;                  //Charge number is 1.
m0 = 938.272;          //Mass in MeV
A = m0/k_u             //Mass number
KE = 250e3;            //Kinetic energy in MeV
emit_nx0 = 1e-6;      //Transverse normalized emittance in m*rand
emit_ny0 = 0.5e-6;   //Vertical normalized emittance in m*rad
dp_p0 = 0.0007;       //Momentum spread dp/p
N_ptcl = 6.56E9;      //Number of protons per bunch
sigma_s0 = 2E-2;      //RMS bunch length
//Define the proton beam as p_beam
Beam p_beam(Z, A, KE, emit_nx0, emit_ny0, dp_p0, sigma_s0, N_ptcl);
```

A coasting proton beam can be defined in the same way without the rms bunch length included, such as

```
Beam p_beam(Z, A, KE, emit_nx0, emit_ny0, dp_p0, N_ptcl);
```

Define the ring

The class **Lattice** and the class **Ring** are provided in the "**ring.h**".

The TWISS parameters at different position of the machine in MADX tsf format can be read into the class Lattice. The TWISS parameters should be saved in the following sequency: "s, bet_x, alf_x, mu_x, d_x, dp_x, bet_y, alf_y, mu_y, d_y, dp_y". The following code define a class lattice that saves the TWISS parameters:

```
//The TWISS parameters are saved in MEICColliderRedesign1IP.tfs
std::string filename = "MEICColliderRedesign1IP.tfs";
Lattice lattice(filename);  //Load the TWISS parameters to lattice
```

The ring can be defined once the TWISS parameters and the ion beam are ready.

```
Ring ring(lattice, p_beam);
```

Intrabeam scattering (IBS) rate calculation

The IBS rate can be calculated, after the ring and the ion beam are define. The IBS rate is calculated using Martini model, which assumes the vertical dispersion function is zero everywhere. One needs to set up the grid number for the 3D integral in the Martini formula.

```
int nu = 100;
int nv = 100;
int nz = 40;
ibs_paras = new IBSParas(nu, nv, nz);
```

Coulomb logarithm can be used to replace the longitudinal integral.

```
int nu = 100;
int nv = 100;
double log_c = 19.4;
ibs_paras = new IBSParas(nu, nv, log_c);
```

The transverse coupling rate can be set between 0 to 1.

```
ibs_paras->set_k(0.2); //Set the transverse coupling rate to be 0.2. Default is 0.
```

Now the IBS rate can be calculated as follows.

```
double rx_ibs, ry_ibs, rz_ibs;  //Variables to save the IBS rate results
config_ibs(lattice);
ibs_rate(lattice, p_beam, *ibs_paras, rx_ibs, ry_ibs, rz_ibs);
end_ibs();
//Print the IBS rate to screen.
std::cout<<"ibs rate: "<<rx_ibs<<' '<<ry_ibs<<' '<<rz_ibs<<std::endl;
```

Define the cooler

The **Cooler** class is provide in "cooler.h". The following parameters are required to define an electron cooler: the length of the cooler, the number of coolers, the magnetic field inside the cooler, and the transverse beta functions at the cooler. One can also specifies the dispersion functions, the alpha functions, and the derivative of the dispersion functions at the cooler. If not set, the default values for them are zeros.

```
double cooler_length = 10;
double n_section = 1;
double magnetic_field = 0.1;
double beta_h = 10;
double beta_v = 10;
double dis_h = 0;   //Dispersion functions at the cooler can be set but not required.
double dis_v = 0;
Cooler cooler(cooler_length,n_section,magnetic_field,beta_h,beta_v,dis_h, dis_v);
```

Define the electron beam

The **EBeamShape** class in "beam.h" provides an interface for different shapes of the electron beam. Currently the following three shapes have been defined: coasting electron beam with uniform density and cylinder shape, Gaussian bunch, and uniform bunch. (Other shapes can be defined easily as a derived class of **EBeamShape** too.) The respective classes for them are **UniformCylinder** , **GaussianBunch** and **UniformBunch** , all of which are derived classes of **EBeamShape** . Users can set values of different parameters for each shape, as follows.

```
//Define a coasting electron beam with uniform density and cylinder shape
double current = 2;      //Current of the electron beam in A
double radius = 0.008;  //Transverse radius of the electron beam
UniformCylinder uniform_cylinder(current, radius);
```

```
//Define a bunched electron beam with Gaussian distribution
double ne = 1e8;         //Number of electrons in the bunch
double sigma_x = 1.5e-2;//RMS bunch size in the three dimensions in meters
double sigma_y = 1.5e-2;
double sigma_s = 2e-2;
GaussianBunch gaussian_bunch(ne, sigma_x, sigma_y, sigma_s);
```

```
//Define a bunched electron beam with uniform density and beer can shape
double current = 2;      //Current of the electron beam in A
double radius = 0.008;  //Transverse radius of the electron beam
double length = 2e-2     //Full length of the bunch in meters
UniformBunch uniform_bunch(current, radius, length);
```

Then the electron beam in the cooler can be defined using the **EBeam** class, provided in "beam.h" Users can set value of the temperature and the Lorentz factor γ for the electron beam.

```
double gamma_e = p_beam.gamma();    //Electrons have the same velocity with the ions
double tmp_tr = 0.1;                 //Transverse temperature
double tmp_long = 0.1;               //Longitudianl temperature
EBeam e_beam(gamma_e, tmp_tr, tmp_long, uniform_cylinder); //Coasting electron beam
```

Choose the friction force formula

For now, only one formula is provided for friction force calculation, which is the Parkhomchuk formula for magnetized friction force. Other formulas would be added in future. The user needs to choose the formula to use in electron cooling rate calculation.

```
//Choose the Parkhomchuk formula for friction force calculation
force_paras = new ForceParas(ForceFormula::PARKHOMCHUK);
```

Electron cooling rate calculation

Two methods, the single-particle method and the Monte-Carlo method, are provided for electron cooling rate calculation, following the terminologies used in BETACOOL. The basic idea and the computation process are the same for the both methods: (1) generate sample ions, (2) calculate the friction force on each sample ion and the momentum change for each of them, (3) calculate the new emittance of the ion bunch, and (4) finally calculate the cooling rate as the emittance change rate. The difference between the two methods lies in the sampling of the ions and the calculation of the emittance.

In the single-particle method, the transverse coordinates of the sample ions are generated using the following formulas:

$$x_\beta = \sqrt{I\beta}\sin\phi$$
$$x'_\beta = \sqrt{\frac{I}{\beta}}(\cos\phi - \alpha\sin\phi)$$

where I is the transverse dynamic invariance such as

$$I = \beta {x'_\beta}^2 + 2\alpha x_\beta x'_\beta + \gamma x_\beta^2 = \varepsilon_x$$

ε_x is the transverse emittance, and α, β, γ are the TWISS parameters at the cooler. All the sample ions have the same dynamic invariance I, determined by the given ε_x, but different ϕ . With dispersion D and D', an adjustment on the transverse coordinates, with respect to the momentum spread $\Delta p/p$ of the ion , should be added.

$$x = x_\beta + D\frac{\Delta p}{p}$$
$$x' = x'_\beta + D'\frac{\Delta p}{p}$$

For coasting ion beam, the longitudinal positions are set to zero for all the sample ions. Their momentum spreads have the same absolute value, but could have different signs. For bunched ion beam, the longitudinal distance to the reference particle s and the momentum spread $\Delta p/p$ for each ion is generated as

$$\frac{\Delta p}{p} = \sqrt{I_l}\cos\psi$$
$$s = \beta_l\sqrt{I_l}\sin\psi$$

where I_l is the longitudinal dynamic invariance such as

$$I_l = \left(\frac{\Delta p}{p}\right)^2 + \left(\frac{s}{\beta_l}\right)^2 = 2\varepsilon_l$$

ε_l is the longitudinal emittance, and $\beta_l = \frac{\delta_s}{\delta_p/p}$, in which σ_s is the r.m.s. bunch length, and σ_p/p is the r.m.s. momentum spread. Same with the transverse case, all the ions have the same dynamic variant I_l, determined by the given ε_l, but different ψ. The total number of ions are determined by how many different values of ϕ and ψ are used, which can be chosen by the user. If one wants to use n_t different ϕ and n_l different ψ , the values of ϕ will be calculated from 0 to 2π with the step size of $2\pi/n_t$ and the values of ψ will be calculated from 0 to 2π with the step size of $2\pi/n_l$. The totally number of ions will be $n_t^2 \cdot n_l$ for bunched ion beam and $2n_t^2$ for coasting ion beam.

After generating the the ions, the friction force on each ion is calculated using the selected formula. The cooler is treated as a thin lens, so that the friction force gives a kick to the ion, which leads to a change of the momentum but does not affect the position. Then the new emittances ε'_x and ε'_l can be calculated using the above formulas on I and I_l for each ion and taking average over all the ions. The cooling rate is calculated as

$$R_x = \frac{\varepsilon'_x - \varepsilon_x}{\varepsilon_x\tau}$$
$$R_l = \frac{\varepsilon'_l - \varepsilon_l}{\varepsilon_l\tau}$$

where τ is the circular motion period of the ions.

The following shows a sample code for electron cooling rate calculation using the single particle method. The ion beam, the electron beam, the cooler, the ring, and the friction force formula should be defined before the calculation.

```
unsigned int n_tr = 100;        //number of $\phi$ for transverse direction
unsigned int n_long = 100;      //number of $\psi$ for longitudinal direction
ecool_rate_paras = new EcoolRateParas(n_tr, n_long);  //parameters for electron cooling rate
double rate_x, rate_y, rate_s; //Electron cooling rate in x, y, and s direction
//force_paras - parameters for friction force, i_beam - ion beam, e_beam - electron beam
ecooling_rate(ecool_rate_paras, force_paras, i_beam, cooler, e_beam, ring, rate_x, rate_y,
rate_s);
```

In Monte-Carlo method, the sample ions are have Gaussian distribution in all the three dimensions. Each individual ion has its own dynamic invariant, and statistically the emittance of the ion bunch equals the predetermined value. In the transverse direction, if the TWISS parameter $\alpha = 0$ at the cooler, the coordinates x_β and x'_β can be generated as random numbers with Gaussian distribution, whose expected value is zero and whose standard deviations are $\sigma = \sqrt{\beta\varepsilon_x}$ and $\sigma' = \sqrt{\varepsilon_x/\beta}$ respectively, where β is the TWISS parameter and ε_x is the transverse emittance. If $\alpha \neq 0$, we can rotate the frame into the one with $\alpha = 0$, generate the Gaussian random numbers and then transfer them back to the original frame. The transfer matrix M_{so} converts the coordinates from the original frame $\hat{S}$ to the frame $\hat{O}$ with $\alpha = 0$, such as

$$\begin{pmatrix} x_\beta \\ x'_\beta \end{pmatrix}_{\hat{O}} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_\beta \\ x'_\beta \end{pmatrix}_{\hat{S}}$$

where $\tan(2\theta) = 2\alpha/(\gamma - \beta)$ with α, β, γ the TWISS parameters at $\hat{S}$. The new TWISS parameters at $\hat{O}$ can be calculated using the following formula:

$$\begin{pmatrix} \beta \\ \alpha \\ \gamma \end{pmatrix}_{\hat{O}} = \begin{pmatrix} m_{11}^2 & -2m_{11}m_{12} & m_{12}^2 \\ -m_{11}m_{21} & m_{11}m_{22} + m_{12}m_{21} & -m_{12}m_{22} \\ m_{21}^2 & -2m_{21}m_{22} & m_{22}^2 \end{pmatrix} \begin{pmatrix} \beta \\ \alpha \\ \gamma \end{pmatrix}_{\hat{S}}$$

The coordinates are generated in the frame $\hat{O}$, and then they are transferred back to the frame $\hat{S}$, as follows.

$$\begin{pmatrix} x_\beta \\ x'_\beta \end{pmatrix}_{\hat{S}} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_\beta \\ x'_\beta \end{pmatrix}_{\hat{O}}$$

The transverse emittance ε_x is an invariant for these transformations. If the dispersion functions D and D' are not zero, they will result in an adjust on the transverse coordinates. In longitudinal direction, the positions for coasting ion beams are all zero, while the position for bunched ion beams are Gaussian random number, whose expected value is zero and whose stand deviation equals the r.m.s. bunch length. For both cases, the momentum spreads are Gaussian random numbers, whose expected value is zero and whose stand deviation equals the r.m.s. momentum spread. The emittance is calculated statistically, such as

$$\varepsilon_x = \sqrt{\sigma_x^2\sigma_{x'}^2 - \sigma_{xx'}^2}$$

$$\varepsilon_l = \begin{cases} \frac{\sigma_p}{p}\sigma_s & \text{for} \quad \text{bunched beam} \\ \left(\frac{\sigma_p}{p}\right)^2 & \text{for} \quad \text{coasting beam} \end{cases}$$

where σ_x, $\sigma_{x'}$, σ_p/p, and σ_s are the standard deviation of x, x', Δ_p/p, and s. The cooling rates are calculate using the same formula for R_x and R_l above.

The following shows a sample code for electron cooling rate calculation using the Monte Carlo method.

```
unsigned int n = 1000000;       //number of ions
ecool_rate_paras = new EcoolRateParas(n);  //parameters for electron cooling rate
double rate_x, rate_y, rate_s; //Electron cooling rate in x, y, and s direction
//force_paras - parameters for friction force, i_beam - ion beam, e_beam - electron beam
ecooling_rate(ecool_rate_paras, force_paras, i_beam, cooler, e_beam, ring, rate_x, rate_y,
rate_s);
```

Simulation of the electron cooling process

The electron cooling process is simulated in a four-step procedure, which includes:

1. Initialize the simulation environment.
2. Create sample ions.
3. Calculate the expansion rate under the IBS and/or electron cooling effect.
4. Update the beam parameters and the sample ions. Repeat from the third step till the end of time.

In the first step, one selects which effect, IBS effect, electron cooling or both, to consider in the simulation, and which method, for example the RMS dynamic model, the model beam model, or any other model, to use in the simulation. Other parameters, such as the total time and the step size, should also be set up. In the second, sample ions are created according to the method and the parameters selected in the first step. In the third step, friction force on all the ions are calculated and the expansion rate due to the IBS and/or electron cooling effect at this instant time is calculated. In the last step, the parameters of the ion beam, such as the emittance, the momentum spread and the bunch length (if applicable) are updated. The coordinates of all the ions are also updated.

Both the RMS dynamic method and the model beam method in BETACOOL fit into this four-step procedure. The main difference of them exists in the fourth step.   Using the RMS dynamic method, one assumes the ion beam keeps the Gaussian distribution during the cooling process and concentrates on the evolution of the macroscopic parameters, such as the emittance and the momentum spread, not the individual ions in the simulation. So in step four, one updates the beam parameters first, and then create a new group of sample ions according to the new beam parameters to replace the old sample ions. However, under strong cooling effect, the ion beam is not necessary to maintain the Gaussian distribution. In such a case, the model beam method is prefer. Using the model beam method, both the IBS effect and the cooling effect are treated as kicks to each ion, and the beta oscillation and the synchrotron oscillation during one time step are treated as a random phase advance to each ion. Thus in the fourth step, the coordinates of all the ions are updated and the new beam parameters are statistically calculated.

The following shows a sample code for the electron cooling process simulation with the model beam method.

```
int time = 3600;                                   //Simulate one hour (3600 seconds)
int n_step = 360;                                  //Number of steps
bool effect_ibs = false;                           //IBS effect not included in the simulation
bool effect_ecool = true;                          //Electron cooling effect included
dynamic_paras = new DynamicParas(time, n_step, effect_ibs, effect_ecool);
//Use the model beam method. The default method is RMS dynamic method.
dynamic_paras->set_model(DynamicModel::MODEL_BEAM);

char file[100] = "cooling_dynamic_output.txt";  //Define the output file
std::ofstream outfile;
outfile.open(file);
//Start the simulation. p_beam, cooler, e_beam, and ring shoud have been defined.
dynamic(p_beam, cooler, e_beam, ring, outfile);
outfile.close();                                   //Close the output file
```

If one only needs to include the IBS effect in the simulation, it is not necessary to define the cooler and the electron beam. Since the Gaussian distribution is assumed in the Martini formula, RMS dynamic method is preferred. A sample code is shown as follows.

```
int time = 3600;                                   //Simulate one hour (3600 seconds)
int n_step = 360;                                  //Number of steps
bool effect_ibs = true;                            //IBS effect included in the simulation
bool effect_ecool = false;                         //Electron cooling effect not included
dynamic_paras = new DynamicParas(time, n_step, effect_ibs, effect_ecool);

char file[100] = "ibs_dynamic_output.txt";      //Define the output file
std::ofstream outfile;
outfile.open(file);
//The electron beam and the cooler are not defined.
Cooler *cooler=nullptr;
EBeam *e_beam=nullptr;
//Start the simulation. p_beam and ring shoud have been defined.
dynamic(p_beam, *cooler, *e_beam, ring, outfile);
outfile.close();                                   //Close the output file
```

The output file contains the following data in columns: time [s], emittance [m] in x direction, emittance [m] in y direction, momentum spread dp/p, rms bunch length [m] (for bunched ion beam only), expansion rate [1/s] in x direction, expansion rate [1/s] in y direction, and expansion rate [1/s] in longitudinal direction.

Sample code

The following is a sample of the "main.cc" file.

```cpp
#include <chrono>
#include<fstream>
#include "dynamic.h"
#include "ecooling.h"
#include "ibs.h"
#include "ring.h"

extern DynamicParas * dynamic_paras;
extern IBSParas * ibs_paras;
extern EcoolRateParas * ecool_paras;
extern ForceParas * force_paras;

int main() {
    double m0, KE, emit_nx0, emit_ny0, dp_p0, sigma_s0, N_ptcl;
    int Z;
    //Define the proton beam
    Z = 1;                       //Charge number
    m0 = 938.272;               //Mass in MeV
    KE = 800;                    //Kinetic energy in MeV
    emit_nx0 = 1.039757508e-6;  //Emittance in x direction, [m*rad]
    emit_ny0 = 1.039757508e-6;  //Emittance in y direction, [m*rad]
    dp_p0 = 2e-3;               //Momentum spread
    N_ptcl = 3.6E11;            //Number of particles
    Beam p_beam(Z,m0/k_u, KE, emit_nx0, emit_ny0, dp_p0, N_ptcl);

     // define the lattice of the proton ring
    std::string filename = "MEICBoosterRedesign.tfs"; //Lattice file in MADX tfs format
    Lattice lattice(filename);

    //Define the ring
    Ring ring(lattice, p_beam);

    //Set IBS parameters.
    int nu = 200;
    int nv = 200;
    double log_c = 44.8/2;
    ibs_paras = new IBSParas(nu, nv, log_c);
    ibs_paras->set_k(1.0);      //Fully coupled in transverse directions

    //define the cooler
    double cooler_length = 10;  //Cooler length in m
    double n_section = 1;
    double magnetic_field = 0.1;//Magnetic field in T
    double beta_h = 10;         //Horizontal beta function in m
    double beta_v = 10;         //Vertical beta function in m
    double dis_h = 0;           //Horizontal dispersion function in m
    double dis_v = 0;           //Vertical dispersion function in m
    Cooler cooler(cooler_length,n_section,magnetic_field,beta_h,beta_v,dis_h, dis_v);

    //define electron beam (DC electron beam)
    double current = 2;         //Current in A
    double radius = 0.008;      //Radius in m

    double neutralisation = 0;
    UniformCylinder uniform_cylinder(current, radius, neutralisation);
    double gamma_e = p_beam.gamma();     //Lorentz factor gamma
    double tmp_tr = 0.1;         //Transverse temperature in eV
    double tmp_long = 0.1;       //Longitudinal temperature in eV
    EBeam e_beam(gamma_e, tmp_tr, tmp_long, uniform_cylinder);

    //define cooling model: monte carlo
    unsigned int n_sample = 40000;  //Number of the sample ions
    ecool_paras = new EcoolRateParas(n_sample);
    //define friction force formula
    force_paras = new ForceParas(ForceFormula::PARKHOMCHUK);
    //define dynamic simulation
    double time = 60;            //Total time to simulate
    int n_step = 120;            //Number of steps
    bool effect_ibs = true;      //IBS effect included in smiulation
    bool effect_ecool = true;    //Electron cooling included in simulation
    dynamic_paras = new DynamicParas(time, n_step, effect_ibs, effect_ecool);
    dynamic_paras->set_model(DynamicModel::MODEL_BEAM); //Choose the model beam method

    char file[100] = "ibs_ecool_dynamic_output.txt";    //Output file
    std::ofstream outfile;
    outfile.open(file);
    dynamic(p_beam, cooler, e_beam, ring, outfile);     //Start simulation
    outfile.close();                                       //Close the output file

    return 0;
}
```

Advanced topics