BNL-99534-2013-TECH
C-A/AP/388;BNL-99534-2013-IR

Sim Track User's Manual (v 1.0)

Y. Luo

January 2010

Collider Accelerator Department
Brookhaven National Laboratory

U.S. Department of Energy
USDOE Office of Science (SC)

DISCLAIMER

BNL-91006-2010-IR

C-A/AP/#388
January 2010

Sim Track User's Manual (v 1.0)

Y. Luo

Collider-Accelerator Department
Brookhaven National Laboratory
Upton, NY 11973

January 26, 2010

SimTrack User's Manual (v 1.0)

Yun Luo
Brookhaven National Laboratory, Upton, NY 11973, USA

SimTrack is a simple c++ library designed for the numeric particle tracking in the high energy accelerators. It adopts the 4th order symplectic integrator [1, 2] for the optical transport in the magnetic elements. The 4-D and 6-D weak-strong beam-beam treatments [3, 4] are integrated in it for the beam-beam studies. SimTrack is written with c++ class and standard template library. It provides versatile functions to manage elements and lines. It supports a large range of types of elements. New type of element can be easily created in the library. SimTrack calculates Twiss, coupling and fits tunes, chromaticities and corrects closed orbits. AC dipole and AC multipole are available in this library. SimTrack allows change of element parameters during tracking.

1 Get Started

1.1 Including simtrack.h

SimTrack library is written in C++. It currently has only one file simtrack.h with 6000+ lines. To use SimTrack library in a general C++ program, you simply need to include simtrack.h in the beginning of your source code. The syntax and functions of C++ and SimTrack library will be applied to the source code.

As an example, following is a source code "test.cpp" which uses SimTrack library to calculate some accelerator physics parameters. In test.cpp, some SimTrack functions are used, such as Read_MADXLattice() to read in the lattice file generated from MADX, Get_Twiss() to calculate Twiss parameters, Get_Chrom() to calculate the chromaticities.

```
#include <iostream>
#include "simtrack.h"

using namespace  std;

int main()
{
  int  i;
  Line  rhic;
  double x[6];

  //---read in lattices
  Read_MADXLattice("./parameters_input_2010AuAu_Blue",rhic);

  //---calculate tunes
  Get_Twiss(rhic, 0.0);
  cout<<''Eigen tunes :''<<rhic.Tune1<<"  "<<rhic.Tune2<<endl;

  //---calculate chromaticities
  Get_Chrom(rhic);
  cout<<''First order chroms:  ''<< rhic.chrom1x<<"  "<<rhic.chrom1y<<endl;
  cout<<''Second order chroms:  ``<<rhic.chrom2x<<"  "<<rhic.chrom2y<<endl;

  //---print out the Twiss parameters
  for(i=0;i<rhic.Ncell;i++)
      cout<<rhic.Cell[i]->NAME<<''  ``<<rhic.Cell[i]->S<<''   ``<<rhic.Cell[i]->Beta1<<endl;

  //---track a particle one turn
  x[x_] =0.00001;
  x[px_]=0.0;
  x[y_] =0.00001;
  x[py_]=0.0;
  x[z_] =0.0;
  x[delta_]=1.0e-04;

  for(i=0;i<rhic.Ncell;i++) {
     rhic.Cell[i]->Pass(x);
```

```
    cout<<rhic.Cell[i]->S<<'' ''<<x[0]<<'' ''<<x[2]<<endl;
  }

  //---the end
  return 0;

}
```

1.2 Input file generated from MADX

The lattice input file"parameters_input_2010AuAu_Blue" in the above code is generated from MADX. To generate it from MADX, we simply include the following block in the MADX input file and run MADX. The output file "parameters_input_2010AuAu_Blue" includes all the magnetic strengths of each element of the ring "rhic". Please pay attention that function Read_MADXLattice() may omit some types of elements and simply transfers them into "DRIFT" type. Function Read_MADXLattice() also zero the cavity voltage ("VRF") of RFCAV element and the bunch particle number("NP") of BEAMBEAM element and so on. Actually we can write interface functions in SimTrack to read in any format of lattice. SimTrack also can define and build up a ring or line from single element by using line manipulation functions to be discussed later.

```
use, period=rhic;
select, flag=twiss,  column=NAME, KEYWORD,S,L,ANGLE,E1,E2,tilt, K0L,K0SL,K1L,K1SL,K2L,
K2SL,K3L,K3SL,K4L,K4SL,K5L,K5SL,K6L,K6SL,K7L,K7SL, K8L,K8SL,K9L,K9SL, K10L,K10SL, KS;
twiss,table=twiss,file=pparameters_input_2010AuAu_Blue;
stop;
```

1.3 Install GSL library

SimTrack library uses GSL library to solve 4×4 eigen system to get eigen vectors and eigen tunes. Therefore, to compile source code using SimTrack library, the GSL library should be included too. If GSL library is not installed in your system by your system administer, you can download the source code from http://www.gnu.org/software/gsl/ and copy to a work directory. Under that work directory, follow the following commands to install it under your local directory $HOME/bin/GSL.

```
>tar xvf gsl-1.9.tar
>cd  gsl-1.9
>./confifure --prefix=\$HOME/bin/GSL --disable-shared
>make
>make install
```

1.4 Compiling source code

For simplicity, the author created a short script called "MakeST" under $HOME/bin to facilitate the compiling source codes using SimTrack library. "MakeST" reads

```
rm -r  \$1
g++ -O2  \$1.cpp  -I$HOME/bin/GSL/include -L$HOME/bin/GSL/lib -lgsl -lgslcblas  -lm -o \$1
```

We can compile a source code, for example test.cpp, using SimTrack library under any directory by typing

```
> MakeSimTrack  test
```

If compiling is successful, you will have the executable file named "test". To run it, you simply type

```
> ./test
```

2 Element

2.1 Common parameters and functions of element

SimTrack supports a large range of element types. Regarding the specific type of element, each element has the following common parameters and functions.

```
  string  NAME, TYPE;
  double  S, L, DX, DY, DT;
  double  X[6], T[36], M[36], A[16];
  double  Beta1, Alfa1, Beta2, Alfa2,  Mu1, Mu2,r, c11, c12, c21, c22;
  double  Etax, Etay, Etaxp,  Etayp;
  double  APx, APy;
```

```
virtual void    SetP(const char *name, double value)=0;
virtual double  GetP(const char *name)=0;
virtual void    Pass(double x[6])=0;
virtual void    DAPass(tps x[6])=0;
```

The definitions of the above parameters are:

```
NAME :  name of the element, different elements can have same NAME;
TYPE :  can be DRIFT, SBEND, QUAD, SKEWQ, KICKER, BPM, RFCAVITY,...
S    :  logintudinal location of the element end;
L    :  the length of the element;
DX, DY, DT: the misaligments with respect to the reference orbit
            reference orbit only decided by DRIFT and SBEND
X[6] :  closed orbit, x, px, y, py, z, delta=dp/p0, z>0 means earlier than synchronous particle
T[36]:  linear transfer matrix of this element;
M[36]:  one turn map at the end of this element;
A[16]:  matrix for the normal form of transverse matrix;
Beta1, Alfa1, Beta2, Alfa2,  Mu1, Mu2: Twiss parameters for eign motions;
r, c11, c12, c21, c22: transverse coupling parameters;
Etax, Etay, Etaxp,  Etayp: dispersions and their slopes;
APx, APy: physical apertures of this element;
```

2.2 Access to common parameters and functions

To access the above common parameters and functions of element, we simply use pointers. For example, to get the Beta1 of an element in the line "rhic", we use

```
rhic.Cell[i]->Beta1
```

where i is the index of the elements in a ring or line, i starts from 0. For the line "rhic", there are rhic.Ncell elements. The index of the last element is (rhic.Ncell-1).

To transfer a particle with coordinate double x[6] through an element, we use

```
rhic.Cell[i]->Pass(x)  .
```

Pass() and DAPass() only differ in the input types of data. Pass() is used for double x[6] while DAPass() is used to transfer the linear tpsa data. DAPass() is used during one-turn map and Twiss calculations.

2.3 Specific types of element

SimTrack supports the following types of element. Each type of element has its specific parameters. For example, QUAD element has parameter "K1L" while SEXT element has "K2L". These specific parameters only can be accessed and modified with functions GetP() and SetP(). Following lists the accepted types of element and their specific parameters in SimTrack library.

```
DRIFT:   drift

SBEND:   sbend
         int Nint;
         double  ANGLE, E1,E2;

QUAD :   quadrupole
         double  K1L
         int    Nint, Norder

SEXT :   sextupole,
         double K2L
         int Nint, Norder

SKEWQ:   skew quadrupole
         double  K2SL
         int Nint, Norder

OCT  :   octupole
         double K3L
         int Nint, Norder

MULT :   multipole
         double  KNL[11], KNSL[11]
         int Nint, Norder

KICKER:  dipole kicker
         double  HKICK, VKICK

HKICKER: horizontal dipole kicker
```

```
        double  HKICK

VKICKER: vertical dipole kicker
        double  VKICK

HACDIP:  horizontal ac dipole
        int  TURNS, TURNE
        double  HKICKMAX, NUD, PHID

VACDIP:  vertical ac dipole
        int TURNS, TURNE
        double  VKICKMAX, NUD, PHID

ACMULT:  AC multipole
        int  Norder, Tturns
        double  KL, PHI0

COOLING: artificial cooling element
        double  ALPHA

HBPM:    horizontal BPM

VBPM:    vertical BPM

BPM:     dual plane BPM

MARKER:  marker

RCOLL :  round collimator
        double RSIZE

ECOLL :  rectanglar collimator
        double  XSIZE,YSIZE

SOLEN :  solenoid
        double KS

RFCAV :  rf cavity
        double  VRF, FRF, PHASE0

INSTR :  instrumentation

BEAMBEAM: beam-beam
         int NSLICE,
         double NP, TREATMENT, EMITX, EMITY, SIGMAL, BETAX, ALFAX, BETAY, ALFAY

ELENS:    electron lens
         int NSLICE
         double NE, BETAE, SIGMAX, SIGMAY,  NSLICE

MATRIX:   6*6 linear matrix
         double  M66[36]
         double  XCO_IN[6], XCO_OUT[6]

ELSEP:    stastic electric separator
         double Ex, Ey;
```

In the above, "Nint" and "Nslice" is the integration steps in the particle transferring. "Norder" is maximum order of the magnetic field. They are automatically set in the construction functions of magnetic elements. SimTrack uses 4-th order symplectic integrator to track particles.

2.4 Access to specific parameters

As an example, the following block source code prints out the strengths of all quadrupoles in the line "rhic", where GetP()is used,

```
//---print K1L of all quadrupole in line rhic
for(i=0;i<rhic.Ncell;i++)
  if(rhic.Cell[i]->TYPE==string(''QUAD'') )
       cout<<rhic.Cell[i]->NAME<<'' ''<<rhic.Cell[i]->S<<'' ''<<rhic.Cell[i]->GetP(''K1L'')<<endl;
```

Another example to set all sextupole "SF" strength to be 0., where SetP() is used,

```
//---set SF strength to 0.
for(i=0;i<rhic.Ncell;i++)
  if(rhic.Cell[i]->NAME==string(''SF'') and rhic.Cell[i]->TYPE==string(''SEXT'')  )
        rhic.Cell[i]->SetP(''K2L'',0.);
```

Again, to access or set the specific parameters of one type of element, we need to use functions GetP() and SetP() of that type of element. However, to access the common paramters of an element, like Twiss parameters etc, we use pointers.

2.5 Create an element

To create an element with a specific type in the source code, we use the construction function of that element type.

```
Element * temp_element;
temp_element= new DRIFT(string name, double l);
temp_elemnt = new SBEND(string name, double l, double angle, double e1, double e2);
temp_element= new  QUAD(string name, double l, double k1l )
temp_element= new SKEWQ(string name, double l, double k1sl)
temp_element= new SEXT(string name, double l, double k2l)
temp_element= new OCT(string name, double l, double k3l)
temp_element= new MULT(string name, double l, double knl[11], double knsl[11])
temp_element= new KICKER(string name, double l, double hkick, double vkick)
temp_element= new HKICKER(string name, double l, double hkick)
temp_element= new VKICKER(string name, double l, double vkick)
temp_element= new HACDIP(string name, double l, double hkickmax, double nud, double phid, int turns, int turne )
temp_element= new VACDIP(string name, double l, double vkickmax, double nud, double phid, int turns, int turne )
temp_element= new ACMULT(string name, double l)
temp_element= new COOLING(string name, double l, double alpha )
temp_element= new HBPM(string name, double l)
temp_element= new VBPM(string name, double l)
temp_element= new BPM(string name, double l)
temp_element= new MARKER(string name, double l)
temp_element= new RCOLL(string name, double l, double rsize)
temp_element= new ECOLL(string name, double l, double xsize, double ysize)
temp_element= new SOLEN(string name, double l, double ks)
temp_element= new RFCAV(string name, double l, double vrf, double frf, double phase0)
temp_element= new INSTR(string name, double l)
temp_element= new BEAMBEAM(string name, int treatment, double np, double sigmal, int nslice,
              double emitx, double emity,double betemp_element= new tax, double alfax, double betay, double alfay)
temp_element= new ELENS(string name, double l, double ne, int nslice, double betae,
              double sigmax, double sigmay)
temp_element= new MATRIX(string name, double l, double m66[36], double xco_in[6], double xco_out[6])
temp_element= new ELSEP(string name, double l)
```

In SimTrack, BEAMBEAM type of element must have zero length.

3 Line

3.1 Definitions

Line is a sequence of specific elements. Line has the following parameters and functions,

```
vector  <Element *> Cell;
double line_length;
long   Ncell;
double Tune1, Tune2, chrom1x, chrom1y, chrom2x, chrom2y;
void Update();
void Append(Element * x)
void Delete(int i)
void Insert( int i, Element * temp)
void Empty()
```

The definitions of Line's parameters are

```
Cell         :  a vector to hold elements of a line, totally number not limited.
line_length  :  total length of the line.
Ncell        :   number of elements in the line.
Tune1, Tune2, chrom1x, chrom1y, chrom2x, chrom2y: tunes and chromaticities of the line.
```

The definitions of line manipulation functions

```
Update()           :  update the line to set S of individual elements.
Append(Element *x) :  append an element to the end of the line.
Delete(int i)      :  delete the element  with index i, i=0,1,.., Ncell-1.
Insert( int i, Element * temp): insert an element, new element will have index i.
Empty()            : empty the whole line, Ncell=0;
```

There are two line manipulation functions which are not the member functions of line.

```
void Rewind_Line(Line & linename,  int k )
void Inverse_Line(Line & linename)
```

Rewind_Line() is to set the new starting point of a ring. The new starting element will be Cell[k] of previous line. Rewind_Line() doesn't change the order of elements. Inverse_Line() is to revert the order of elements of the line. Inverse_Line() doesn't change the strengths of elements.

3.2 Some examples

As an example, following block assign voltage to an existing rf cavity element and delete, define, insert beam-beam elements in the line "rhic".

```
  //---install RF
  if(true) {
    for(i=0; i<rhic.Ncell;i++){
      if( rhic.Cell[i]->TYPE=="RFCAV" ) {
        loc_rf=i;  break;
      } }
    rhic.Cell[loc_rf]->SetP("VRF",0.3);
  }

  //---install BB at IP6 and IP8
  if(true) {

    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP6" ) loc_ip6=i;
    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP8" ) loc_ip8=i;

    rhic.Delete(loc_ip6);
    temp_element= new BEAMBEAM("IP6", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip6,temp_element);

    rhic.Delete(loc_ip8);
    temp_element= new BEAMBEAM("IP8", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip8,temp_element);

  }
```

As another example, the following source code to revert the rhic Blue ring lattice to check the parameter changes.

```
#include <iostream>
#include "simtrack.h"

using namespace  std;

main()
{
  int i;
  Line  rhic;

  //---read in lattices
  Read_MADXLattice("./parameters_input_2010Au_Yellow",rhic);

  Inverse_Line(rhic);

  for(i=0;i<rhic.Ncell;i++)
    if(rhic.Cell[i]->TYPE==string("SBEND") ) rhic.Cell[i]->SetP("ANGLE", -1.0*rhic.Cell[i]->GetP("ANGLE") );

  for(i=0;i<rhic.Ncell;i++)
     if(rhic.Cell[i]->TYPE==string("SEXT") ) rhic.Cell[i]->SetP("K2L", -1.0*rhic.Cell[i]->GetP("K2L") );

  Get_Twiss(rhic,0.0);
  cout<<"   Tunes:  "<<rhic.Tune1<<"  "<<rhic.Tune2<<endl;
  Get_Chrom(rhic);
  cout<<"   Chrom1: "<<rhic.chrom1x<<"  "<<rhic.chrom1y<<endl;
  cout<<"   Chrom2: "<<rhic.chrom2x<<"  "<<rhic.chrom2y<<endl;

  return 0;

 }
```

The following source code creates a simple ring and check its tunes.

```
#include <iostream>
#include "simtrack.h"

using namespace  std;

main()
{
  int i,j;
  Line  rhic;
  Element * element_m66;
  Element * element_bb;

  //---the linear one-turn map
```

```
  double m66[36];
  double qx=28.695, qy=29.685, betax=0.35, betay=0.35;
  double xco_in[6], xco_out[6];

  for(i=0;i<6;i++)
    for(j=0;j<6;j++) m66[i*6+j]=0.;
  m66[0*6+0]= cos(2.0*PI*qx);        m66[0*6+1]=betax*sin(2.0*PI*qx);
  m66[1*6+0]=-sin(2.0*PI*qx)/betax;  m66[1*6+1]=      cos(2.0*PI*qx);
  m66[2*6+2]= cos(2.0*PI*qy);        m66[2*6+3]=betay*sin(2.0*PI*qy);
  m66[3*6+2]=-sin(2.0*PI*qy)/betay;  m66[3*6+3]=      cos(2.0*PI*qy);
  m66[4*6+4]= 1.0;                   m66[5*6+5]=1.0;

  for(i=0;i<6;i++) {
    xco_in[i]=0.0;     xco_out[i]=0.0;   }

  //-----the elements

  element_m66= new MATRIX("M66", 3833.8451, m66, xco_in, xco_out);
  element_bb=  new BEAMBEAM("IP6", 4, 2.0e11, 0.25, 11, 2.5e-06, 2.5e-06, betax, 0., betay, 0.);

  //---create A ring

  rhic.Append(element_m66);
  rhic.Append(element_bb);

  //--check the parameters

  if(false) {
    for(i=0;i<rhic.Ncell;i++)
      cout<<rhic.Cell[i]->NAME<<"  "<<rhic.Cell[i]->TYPE<<"  "<<rhic.Cell[i]->S<<endl;
    Get_Twiss(rhic,0.0);
    cout<<rhic.Cell[0]->X[0]<<"   "<<rhic.Cell[0]->X[2]<<endl;
    cout<<1.0+rhic.Tune1<<"  "<<1.0+rhic.Tune2<<endl;
    cout<<rhic.Cell[0]->Beta1<<"   "<<rhic.Cell[0]->Beta2<<endl;
  }

  //---the end

  return 0;

}
```

The following source code to insert an AC dipole into the ring and track a particle 10000 turns and print out turn-by-turn data.

```
#include <iostream>
#include <iomanip>
#include "simtrack.h"

using namespace  std;

int main()
{

  int i,j,k;
  int loc;
  double x[6];

  //----read in lattice from madx output
  Line  rhic;
  Read_MADXLattice("./parameters_input",rhic);

  //---install ACDIPOLE
  for(i=0; i< rhic.Ncell; i++){
    if(rhic.Cell[i]->NAME==string("OX3C") ){
      loc=i;
    }
  }

  Element * temp_element;
  temp_element= new HACDIP("ACDIP1", 0., 1.0e-06, 0.677,0.0,500,15000)  ;
  rhic.Insert(loc,temp_element);

  //---let do tracking
  for(i=0;i<6;i++) x[i]=0.;

  if(true) {
    for(GP.turn=1;GP.turn<=10000;GP.turn++){
      for(j=0;j<rhic.Ncell;j++) rhic.Cell[j]->Pass(x);
      cout<<GP.turn<<"  "<<x[0]<<"  "<<x[1]<<"  "<<x[2]<<"   "<<x[3]<<endl;
    }
  }

  //--- the end
```

```
  return 0 ;
}
```

3.3 Assumed global parameters

The following are the default line parameters used in the SimTrack library. They are based on rhic 250 GeV polarized proton run.

```
      charge=1;
      mass0 =0.9382723;      // default test particle is proton
      energy=250000.;        // RF  affected
      gamma =266;            // Beambeam affected
      brho = 831;            // SOLEN affected
      frev0=78250.42279;     // RF affected
      harm=360;              // RF affected
      turn =0 ;              // tracking affected
      bbscale =1.0 ;         // force scaling for beambeam, bbscale =1.0 for proton-proton,  bbscale =79*79/197=31.68 for Au-Au,..
```

4 Optics, Fitting and Closed Orbit Correction

4.1 Optics calculation

SimTrack supplies general optical calculation functions, such as,

```
void Get_Orbit(Line & linename, double deltap)
void Get_Twiss(Line & linename, double deltap)
void Get_Chrom( Line & linename)
void Get_Dispersion(Line & linename, double deltap)
```

Coupling parameters are calculated with Get_Twiss(). And Get_Twiss() already includes Get_Orbit(). Get_Twiss() doesn't calculate dispersion. Dispersion is calculated with function Get_Dispersion(). Get_Dispersion() should be proceeded by Get_Twiss().

SimTrack supplies limited fitting functions, such as,

```
void Fit_Tune(Line & linename, double q1, double q2, const char * qf_name, const char * qd_name)
void Fit_Chrom(Line & linename, double chrom1x_want, double chrom1y_want, const char * sf_name, const char * sd_name )
void Fit_Chrom_RHIC8fam(Line & linename, double chrom1x_want, double chrom1y_want )
```

The following functions simplify the reading and setting strengths of a magnet family which share the same name. In SimTrack, the elements with same names can have different strengths in SimTrack, which is different from MADX.

```
double Get_KL(Line & linename, const char * name, const char *  kl)
void Set_KL(Line & linename, const char * name, const char *  kl, double strength)
void Set_dKL(Line & linename, const char * name, const char * kl, double dstrength)
```

The following functions work one-turn and section maps. Most users don't need to know or use these functions. They are used during optics calculation. The Twiss and coupling parameters are calculated in SimTrack based on Ref. [5].

```
void Get_OneTurnMap(Line & linename, double deltap)
void Get_ElementMap(Line & linename, double deltap)
void Get_SectionMap(Line & linename, int i1, int i2, double deltap, double t66[36] )
void Cal_A44(Line & linename, double deltap)
void Trace_A44(Line & linename, double deltap)
```

As an example, let us look into function Fit_Chrom(),

```
void Fit_Chrom(Line & linename, double chrom1x_want, double chrom1y_want, const char * sf_name, const char * sd_name )
{
  double chrom1x0, chrom1y0,  dchrom1x, dchrom1y;
  double sf_k2l_0, sd_k2l_0, dk2l_sf, dk2l_sd, dchrom1x_sf,  dchrom1y_sf, dchrom1x_sd, dchrom1y_sd;
  double scale_sf, scale_sd;

  Get_Chrom(linename) ;
  chrom1x0= linename.chrom1x;
  chrom1y0= linename.chrom1y;

  while( (chrom1x_want-chrom1x0)*(chrom1x_want-chrom1x0) + (chrom1y_want-chrom1y0)*(chrom1y_want-chrom1y0)  > 0.0001 ) {
    sf_k2l_0= Get_KL(linename,sf_name,"K2L");
    sd_k2l_0= Get_KL(linename,sd_name,"K2L");

    dk2l_sf= sf_k2l_0 * 0.005;
    Set_dKL(linename,sf_name, "K2L", dk2l_sf);
    Get_Chrom(linename);
```

```
    dchrom1x_sf=linename.chrom1x - chrom1x0;
    dchrom1y_sf=linename.chrom1y - chrom1y0;
    Set_dKL(linename,sf_name, "K2L",-dk2l_sf);

    dk2l_sd=  sd_k2l_0 * 0.005;
    Set_dKL(linename,sd_name, "K2L", dk2l_sd);
    Get_Chrom(linename);
    dchrom1x_sd=linename.chrom1x - chrom1x0;
    dchrom1y_sd=linename.chrom1y - chrom1y0;
    Set_dKL(linename,sd_name, "K2L",-dk2l_sd);

    dchrom1x=chrom1x_want- chrom1x0;
    dchrom1y=chrom1y_want- chrom1y0;

    LinearEquations(dchrom1x_sf, dchrom1x_sd, dchrom1x, dchrom1y_sf, dchrom1y_sd, dchrom1y, scale_sf, scale_sd);
    Set_dKL(linename,sf_name, "K2L", dk2l_sf * scale_sf);
    Set_dKL(linename,sd_name, "K2L", dk2l_sd * scale_sd);

    Get_Chrom(linename);
    chrom1x0=linename.chrom1x;
    chrom1y0=linename.chrom1y;
  }
}
```

4.2 Closed orbit correction

SimTrack also supplies functions for closed orbit correction,

```
void Correct_Orbit_SVD(Line & linename, int m, int n, vector<int> bpm_index, vector<int> kicker_index, int plane)
void Correct_Orbit_SlidingBump1(Line & linename, int m, int n, vector<int> bpm_index, vector<int> kicker_index, int plane)
void Correct_Orbit_SlidingBump2(Line & linename, int m, int n, vector<int> bpm_index, vector<int> kicker_index, int plane)
void Orbit_Status( Line linename, vector<int> bpm_index, int plane, double &orbit_mean, double & orbit_rms )
```

The input of orbit correction are vector containers of the index of BPMs and correctors. Flag plane = 0 means horizontal orbit correction.

Following is an example to do horizontal closed orbit correction with all available horizontal BPMs and horizontal dipole correctors. The initial random closed orbit is generated with quadrupole horizontal misalignments.

```
#include <iostream>
#include <fstream>
#include "simtrack.h"

using namespace  std;

main()
{
  int i,j,k;
  Line  rhic;
  double  orbit_mean, orbit_rms;

  //---read in lattices
  Read_MADXLattice("./parameters_input",rhic);

  //---get the index of hbpm an dhkicker
  vector< int >    hbpm_index, hkicker_index;

  for(i=0;i<rhic.Ncell;i++)
     if (rhic.Cell[i]->TYPE==''HBPM'' )  hbpm_index.push_back(temp_name);

  for(i=0;i<rhic.Ncell;i++)
     if (rhic.Cell[i]->TYPE==''HKICKER'' ) hkicker_index.push_back(temp_name);

  //---generate a random orbit with quadrupole offset
  if( true ) {
    double seed=17823559.;
    for(i=0; i<rhic.Ncell;i++){
      if(rhic.Cell[i]->TYPE==string("QUAD") ) rhic.Cell[i]->DX= (rnd(seed)*2-1)*0.00002 ;
    }
  }

  //----horizontal cod correction:
  if( true ) {
    Get_Twiss(rhic,0.0);
    Orbit_Status( rhic, hbpm_index, 0, orbit_mean, orbit_rms );
    cout<<"Horizontal plane: "<< orbit_mean*1000. <<" +/- "<< orbit_rms*1000.<<"  mm."<<endl;

    Correct_Orbit_SVD(rhic, hbpm_index.size(), hkicker_index.size(),  hbpm_index,  hkicker_index, 0);

    Get_Twiss(rhic,0.0);
    Orbit_Status( rhic, hbpm_index, 0, orbit_mean, orbit_rms );
    cout<<"Horizontal plane: "<< orbit_mean*1000. <<" +/- "<< orbit_rms*1000.<<"  mm."<<endl;
```

```
  }

  //------horizontal orbit after correction
  if(true){
    cout<<rhic.Tune1<<"  "<<rhic.Tune2<<endl;
    for(i=0;i<rhic.Ncell;i++) cout<<rhic.Cell[i]->S<<"   "<<rhic.Cell[i]->X[0]<<"   "<<rhic.Cell[i]->X[2]<<endl;
  }

 //---the end

  return 0;

}
```

5 Tracking

5.1 Track with Pass()

SimTrack supplies several conventional tracking functions,

```
void Track(Line & linename, double x[6], int nturn, int & stable, int & lost_turn, int & lost_post)
void Track_tbt(Line & linename, double x[6], int nturn, double  x_tbt[], int & stable, int & lost_turn, int & lost_post)
void Track_tbt_last100(Line & linename, double x[6], int nturn, double x_tbt[], int & stable, int & lost_turn, int & lost_post)
```

They all use function Pass(x[6]) in the tracking. The physical apertures of each element are used to determine if the particle is lost or not. The definitions of the input parameters are:

```
double x[6]  :  initial coordinates
int nturn    :  tracking turns
int stable   :  flag, if particle lost, it will be 0
int lost_turn:  the turn when the particle is lost
int lost_post:  the particle loss place
```

For example, let us look into the function Track(),

```
void Track(Line & linename, double x[6], int nturn, int & stable, int & lost_turn, int & lost_post)
{
  int j;
  //-----quick check
  if( abs(x[0]) > 0.1 ||  abs(x[2]) > 0.1 || stable ==0 ) {
    stable = 0;    lost_turn= 0;    lost_post = 0;    return; }

  //----now we do tracking
  for(GP.turn=0; GP.turn < nturn; GP.turn++) {
    for(j=0;j<linename.Ncell;j++) {
      if(stable == 1 ) {
        linename.Cell[j]->Pass(x);
        if( abs(x[0]) > linename.Cell[j]->APx or abs(x[2]) > linename.Cell[j]->APy  ) {
            stable=0;  lost_turn= GP.turn;  lost_post=j; return ;
      } }
    }
  }
}
```

5.2 Fast tracking without class

To improve tracking speed, SimTrack also supplies some functions to allow track particles without c++ class,

```
void Prepare_Track_Fast(Line & linename)
void Track_Fast(Line & linename, double x[6], int nturn, int & stable, int & lost_turn, int & lost_post )
```

Before running Prepare_Track_Fast(), we normally first run the following two functions,

```
void MakeThin(Line & linename)
void Concat_Drift(Line & linename)
```

MakeThin() makes nonlinear elements to be zero length with drift-kick-drift model. Concat_Drift() concatenates the adjacent DRIFT elements. MakeThin() must be run before Prepare_Track_Fast() since Prepare_Track_Fast() treat MULT as zero length kick.

Function Prepare_Track_Fast() extracts and saves all lattice information and save them in some global variables which can will be used in function Track_Fast() use. For speed consideration, Track_fast() currently only supports the following types of elements: DRIFT, SBEND, QUAD, SEXT, MULT, RFCAV, BEAMBEAM, ELENS, ACMULT, COOLING, MATRIX. Other types of elements can be easily added into the fast tracking, too.

Following is an example for a long-term tracking,

```
#include <iostream>
#include "simtrack.h"

using namespace  std;

int main()
{
  int i,j,k;
  int loc_ip6, loc_ip8, loc_ip10, loc_elens,loc_rf;
  Line  rhic;
  Element * temp_element;

  //---initiate MPI
  //MPI_Init( NULL, NULL);

  //---read in lattices
  Read_MADXLattice("./parameters_input",rhic);

  //---install RF
  if(true) {
    for(i=0; i<rhic.Ncell;i++){
      if( rhic.Cell[i]->TYPE=="RFCAV" ) {
      loc_rf=i;  break;
      } }
    rhic.Cell[loc_rf]->SetP("VRF",0.3);
  }

  //---install BB and BBC
  if(true) {

    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP6" ) loc_ip6=i;
    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP8" ) loc_ip8=i;

    rhic.Delete(loc_ip6);
    temp_element= new BEAMBEAM("IP6", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip6,temp_element);

    rhic.Delete(loc_ip8);
    temp_element= new BEAMBEAM("IP8", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip8,temp_element);

  }

  //---prepare for fast tracking
  MakeThin(rhic);
  Concat_Drift(rhic);

  //---re-match with BB and Elens after MakeThin()
  Fit_Tune(rhic, 28.67, 29.68, "QF", "QD");
  Fit_Chrom_RHIC8fam(rhic, 1.0, 1.0);

  //---prepare tarck_fast
  Prepare_Track_Fast(rhic);
  rhic.Empty();

  //----test of a long-term tracking
  double x[6];
  for(i=0;i<6;i++) x[i]=0.;
  x[x_]=0.00001;
  x[px_]=0.000003;
  x[y_]=0.00006;
  x[py_]=0.000005;
  x[z_]=0.000;
  x[delta_]=0.1e-03;

  system("date");
  int stable=1,  lost_turn=0, lost_pos=0;
  Track_Fast(rhic, x, 1000000, stable, lost_turn, lost_pos);
  if(stable == 0 ) cout<<''Particle lost.''<<endl;
  system("date");
}
```

6 Other Supporting Functions

SimTrack also supplies some mathematics tools, like SVD, fine tune finder,etc.

```
long int  fac(int n)
double rnd(double & r)
double guass(double u,double g, double & r)
void mat_mult(double A[],  double B[], double C[], int m, int n, int k)
```

```
double mat_det(double a[],int n)
int mat_inv(double a[],int n)
void LinearEquations(double a1, double b1, double c1, double a2, double b2, double c2, double & x, double & y
void  EigenSolver(double Matrix[4][4] , double wr[4], double  wi[4], double vr[4][4], double vi[4][4])
int bmuav(double a[],int m,int n,double *u,double *v,double eps,int ka)  // SVD method
void fft(int m, double*x, double*y)
void  Sin2FFT(int n, double*xr, double*xi, int & nfft, double*famp, double*fphi)
void FineTuneFinder(int nfft, int m,  double *x, double & fpeak)
```

SimTrack also supplies function to calculate frequency map, dynamic apertures and so on.

```
void Track_tbt_FMA( Line & linename, double deltap0, double sigmax0, double sigmay0 )
void Track_tbt_Lyapunov( Line & linename, double deltap0, double sigmax0, double sigmay0 )
void Track_tbt_last100_ActionDiff( Line & linename, double deltap0, double exn0, double eyn0 )
void Track_Fast_DA( Line & linename, int nturn, double delta, double sigmax0, double sigmay0 )
```

Following is an example to calculate frequency map with SimTrack library.

```
#include <iostream>
#include "simtrack.h"

using namespace  std;

int main()
{
  int i,j,k;
  int loc_ip6, loc_ip8, loc_ip10, loc_elens,loc_rf;
  Line  rhic;
  Element * temp_element;

  //---read in lattices
  Read_MADXLattice("./parameters_input",rhic);

  //---install RF
  if(false) {
    for(i=0; i<rhic.Ncell;i++){
      if( rhic.Cell[i]->TYPE=="RFCAV" ) {
loc_rf=i;  break;
      } }
    rhic.Cell[loc_rf]->SetP("VRF",0.3);
  }

  //---prepare for fast tracking
  MakeThin(rhic);
  Concat_Drift(rhic);

  //---install BB and BBC
  if(true) {

    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP6" ) loc_ip6=i;
    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP8" ) loc_ip8=i;
    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP10" ) loc_ip10=i;

    rhic.Delete(loc_ip6);
    temp_element= new BEAMBEAM("IP6", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip6,temp_element);

    rhic.Delete(loc_ip8);
    temp_element= new BEAMBEAM("IP8", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip8,temp_element);

    rhic.Delete(loc_ip10);
    rhic.Delete(loc_ip10);
    temp_element= new DRIFT("TEMPD",1.0);
    rhic.Insert(loc_ip10,temp_element);
    temp_element= new ELENS("BEL1", 1.0, 2.0e11, 3, 0.0, 0.31e-3, 0.31e-3);
    rhic.Insert(loc_ip10+1,temp_element);
    temp_element= new ELENS("BEL2", 1.0, 2.0e11, 3, 0.0, 0.31e-3, 0.31e-3);
    rhic.Insert(loc_ip10+2,temp_element);
    temp_element= new DRIFT("TEMPD",6.8);
    rhic.Insert(loc_ip10+3,temp_element);

    loc_elens=loc_ip10+2;

  }

  //---re-match with BB and Elens after MakeThin()
  Fit_Tune(rhic, 28.67, 29.68, "QF", "QD");
  Fit_Chrom_RHIC8fam(rhic, 1.0, 1.0);
```

```
  //---check the FMA

  Track_tbt_FMA( rhic, 0.0, 0.06855e-03, 0.06855e-03 );

  //--the end

  return(0);

}
```

Following is an example to calculate dynamic aperture with SimTrack library.

```
#include <iostream>
#include "simtrack.h"

using namespace  std;

int main()
{
  int i,j,k;
  int loc_ip6, loc_ip8, loc_ip10, loc_elens,loc_rf;
  Line  rhic;
  Element * temp_element;

  //---read in lattices
  Read_MADXLattice("./parameters_input",rhic);

  //---install RF
  if(true) {
    for(i=0; i<rhic.Ncell;i++){
      if( rhic.Cell[i]->TYPE=="RFCAV" ) {
loc_rf=i;  break;
      } }
    rhic.Cell[loc_rf]->SetP("VRF",0.3);
  }

  //---prepare for fast tracking
  rhic.MakeThin();
  rhic.Concat();

  //---install BB and BBC
  if(true) {

    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP6" ) loc_ip6=i;
    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP8" ) loc_ip8=i;
    for(i=0; i<rhic.Ncell;i++)
      if( rhic.Cell[i]->NAME=="IP10" ) loc_ip10=i;

    rhic.Delete(loc_ip6);
    temp_element= new BEAMBEAM("IP6", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip6,temp_element);

    rhic.Delete(loc_ip8);
    temp_element= new BEAMBEAM("IP8", 6, 2.0e11, 0.4545, 11, 2.5e-06, 2.5e-06, 0.53, 0., 0.53, 0.);
    rhic.Insert(loc_ip8,temp_element);

    rhic.Delete(loc_ip10);
    rhic.Delete(loc_ip10);
    temp_element= new DRIFT("TEMPD",1.0);
    rhic.Insert(loc_ip10,temp_element);
    temp_element= new ELENS("BEL1", 1.0, 2.0e11, 3, 0.0, 0.31e-3, 0.31e-3);
    rhic.Insert(loc_ip10+1,temp_element);
    temp_element= new ELENS("BEL2", 1.0, 2.0e11, 3, 0.0, 0.31e-3, 0.31e-3);
    rhic.Insert(loc_ip10+2,temp_element);
    temp_element= new DRIFT("TEMPD",6.8);
    rhic.Insert(loc_ip10+3,temp_element);

    for(i=0;i<rhic.Ncell;i++){
      if(rhic.Cell[i]->NAME==string("BEL1") ) {
loc_elens= i;
break;
      }
    }

  }

  //---re-match with BB and Elens after MakeThin()
  Fit_Tune(rhic, 28.67, 29.68, "QF", "QD");
  Fit_Chrom_RHIC8fam(rhic, 1.0, 1.0);

  //---prepare track_fast
  Prepare_Track_Fast(rhic);
```

```
  rhic.Empty();

  //----DA tracking
  int nturn=1000000;
  double delta =0.0005;
  double sigmax0=sqrt(2.5e-06 *0.53 / 266.);
  double sigmay0=sqrt(2.5e-06 *0.53 / 266.);
  Track_Fast_DA( rhic, nturn, delta, sigmax0, sigmay0);

  //--the end
  return(0);

}
```

7 Expansion: Define a New Type of Element

To define a new type of element in SimTrack is straight-forward. You need define this type's specific parameters and its construction function, GetP(), SetP() and Pass(), DAPass(). As an example, following is the block to define a RF cavity type of element in simtrack.h.

```
//----------------------------RFCAV CLASS-------------------------------
class RFCAV: public Element
{
 public:
  RFCAV(string name, double l, double vrf, double frf, double phase0): Element(name)
    {
      TYPE=string("RFCAV");
      VRF= vrf;
      FRF= frf;
      PHASE0=phase0;
      L=l;
    }
  void   SetP(const char *name, double value)
    {
     if (strcmp( name, "VRF" ) == 0)
        {
  VRF=value;
        }
     else if (strcmp( name, "FRF" ) == 0)
        {
  FRF=value;
        }
     else if (strcmp( name, "PHASE0" ) == 0)
        {
  PHASE0=value;
        }
     else
        {
  cout<<"RFCAV does not have  parameter of "<<name<<endl;
          exit(0);
        }
    }
  double GetP(const char *name)
    {
     if (strcmp( name, "VRF" ) == 0)
        {
  return VRF;
        }
     else if (strcmp( name, "FRF" ) == 0)
        {
  return FRF;
        }
     else if (strcmp( name, "PHASE0" ) == 0)
        {
  return PHASE0;
        }
      else
        {
  cout<<"RFCAV does not have  parameter of "<<name<<endl;
          exit(0);
        }
    }
  void   Pass(double x[6]){
    if(VRF !=0.)
      {
        drift_pass(x, L/2.);
        if( VRF !=0.)   x[5] =x[5] + (VRF*1.0/GP.energy)*sin(2.0*PI*FRF*x[z_]/3.0e8);
        drift_pass(x, L/2.);
      }
    else
      {
```

```
      drift_pass(x, L);
    }
  }
  void   DAPass(tps x[6]){
      drift_pass(x, L/2.);
      if( VRF !=0.)        x[5] =x[5] + (VRF*1.0/GP.energy)*sin(2.0*PI*FRF*x[z_]/3.0e8);
      drift_pass(x, L/2.);
  }
 private:
  double VRF, FRF, PHASE0;
};
```

8 Acknowledgement

The author would like to thank Hyung Jin Kim for cross check of the 6-D weak-strong beam-beam treatment in SimTrack library.

References

[1] R.D. Ruth, "A canonical Integration Technique", IEEE Trans. Nucl. Sci., vol. NS-30, PP.2669-2671 (1983).

[2] J. Bengtsson, "The Sextupole Scheme for Swiss Light Source (SLS): An Analytic Approach", SLS Note 9/97, March 1997.

[3] M. Bassetti and G.A. Erskine, *Closed expression for the electricalfield of a two-dimensional Gaussian charge*, CERN-ISR-TH/80-06.

[4] K. Hirata, H. Moshammer, F. Ruggiero, A symplectic beam-beam interaction with energy change, Particle Accel. 40 (1993) 205-228.

[5] Y. Luo, Phys. Rev. ST Accel. Beam **7**, 124001 (2004).