Abstract

SimTrack is a simply c++ library designed for the numeric particle tracking in the high energy accelerators. It adopts the 4th order symplectic integrator for the optical transportation in the magnetic elements. The 4-D and 6-D weak-strong beam-beam treatments are included for the beam-beam studies. SimTrack is written with c++ class and standard template library. It provides versatile functions to manage elements and lines. New type of elements can be easily created in the library. It calculates Twiss, coupling and fits tunes, chromaticities and correct closed orbits. During tracking, the parameters of elements can be changed or modulated on the fly.

INTRODUCTION

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 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. For example, AC dipole, AC multipole and electron lens are all available in this library. SimTrack allows change of element parameters during tracking.

SimTrack library currently has only one file simtrack.h with about 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.

ELEMENT AND LINE

Abstract Element

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

```cpp
string NAME, TYPE;
double S, L, DX, DY, DT;
double X[6], T[36], M[36], A[16];
double Beta1, Alpha1, Beta2, Alpha2, Mu1, Mu2,
    r, c11, c12, c21, c22;
```

Specific Elements

SimTrack supports the following types of element. Each type of element has its own specific parameters besides the above common parameters. To get and set the specific parameters we need to use member functions GetP() and SetP(). Following lists the accepted types in SimTrack.

```cpp
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
    int Nint, Norder
KICKER: dipole kicker
    double HKICK, VKICK
HACDIP: horizontal ac dipole
    double HKICKMAX, PHID
VACDIP: vertical ac dipole
    double HKICKMAX, HUID, PHID
ACMULT: AC multipole
    int Norder, TURNS
```

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

```cpp
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

```cpp
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.

05 Beam Dynamics and Electromagnetic Fields
D06 Code Developments and Simulation Techniques

* This work was supported by Brookhaven Science Associates, LLC under Contract No. DE-AC02-98CH10886 with the U.S. Department of Energy.
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 : rectangular collimator
double EXSIZE, EYSIZE
SOLEN : solenoid
double KS
RFCAV : rf cavity
double VRF, FRF, PHASE0
INSTR : instrumentation
BEAMBEAM: beam-beam
int NSLICE,
double BETAX, ALFAX, BETAY, ALFAY
ELENS: electron lens
int NSLICE
MATRIX: 6x6 linear matrix
double M66[36], M66[6]
double XGCD_IN[6], XGCD_OUT[6]
ELSEEP: static 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. Following example sets the strength of all sextupole named “SF” to be 0.,

```cpp
for(i=0;i<rhic.Ncell;i++)
if(rhic.Cell[i]->NAME==string('SF')){
    rhic.Cell[i]->SetP('K2L',0.0);
}
```

To create an element with a specific type in the source code, we use its construction function. For example, to create a AC dipole,

```cpp
for(i=0;i<rhic.Ncell;i++)
if(rhic.Cell[i]->NAME==string('OX3C')){
    loc=i; break;
}
```

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

**Line**

Line is a sequence of specific elements. Simtrack accepts more than one lines in one source code. Line has the following parameters and functions,

```cpp
vector <Element *> Cell;
long Ncell;
double line_length;
void Empty()
void Insert( int i, Element * temp)
void Delete(int i)
void Append(Element * x)
void Update();
```

The line manipulation functions are

- `Update()`: update the line.
- `Append(Element *x)`: append an element to the end of the line.
- `Delete(int i)`: delete the element with index i.
- `Insert( int i, Element * temp)`: insert an element at location i.
- `Empty()`: empty the whole line, Ncell=0;

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

**OPTICAL CALCULATION**

**Optics Parameters**

SimTrack supplies general optical calculation functions based on Ref. [5].

- `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)`

Twiss and 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 called after `Get_Twiss()`.

**Fitting Tune and Chromaticity**

SimTrack supplies limited fitting functions,

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

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,

- `double Get_KL(Line & linename, const char * name, ...)`
- `void Set_KL(Line & linename, const char * name, ...)`
- `void Set_dKL(Line & linename, const char * name, ...)`

**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)`
- `void Correct_Orbit_SlidingBump(Line & linename, vector<int> bpm_index, vector<int> kicker_index)`

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

Track with Pass()

SimTrack supplies several convenient tracking functions,

```cpp
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)
```

Both of them use member function Pass(x[6]) to track. The physical apertures of each element are used to determine if the particle is lost or not. The definitions of the input and output 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(),

```cpp
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]) > 1.0 || abs(x[2]) > 1.0 || 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 || abs(x[2]) > linename.Cell[j]->APy) {
                    stable = 0; lost_turn = GP.turn; lost_post = j; return;
                }
            }
        }
    }
}
```

Fast Tracking

To improve tracking speed, SimTrack also supplies functions to track without using Pass(),

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

Function Prepare_Track_Fast() extracts all lattice information and save them in global variables which will be used in function Track_Fast() use.

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

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

MakeThin() makes thin nonlinear elements. Concat_Drift() concatenates the adjacent DRIFT elements.

ONE EXAMPLE

To conclude, I give an example of a long term tracking using Track_Fast().

```cpp
# include <iostream>
# include "simtrack.h"
using namespace std;

int main()
{
    int i, j;
    double x[6] = {0.0};
    int nturn = 500000;
    int stable = 1, lost_turn = 0, lost_post = 0;
    Track_Fast(rhic, x, nturn, stable, lost_turn, lost_post);
    if (stable == 0) cout << "Particle lost.";
}
```

REFERENCES