EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN

ICOSIM++: a tracking program for collimator studies

Manual for version 1.0

Joël Clivaz

december 2011

1 Introduction

ICOSIM++ is a tracking program derived from ICOSIM and written in C++. It is used for collimator studies of a beam of nuclei or protons. In this manual, we give some explanation on how to install and run it. We also explain quickly how it is working, without going into details. A big part of this manual comes directly from [1].

A simulation done with ICOSIM++ begins with the generation of a bunch of particles or the reading of initial coordinates from a file. Then a linear tracking can be done between the primary collimators and finally a detailed tracking is performed between all the elements of the lattice. During that part of the simulation, we apply some special treatments to the collimators and we make aperture checks all along the ring.

2 General description

In this section, we make a quick explanation on what the program is doing, without going into details.

2.1 Reading of the input parameters

The very first step of a simulation with ICOSIM++ is the reading of needed parameters from input files. Two files are containing this kind of information: the collimator file and the optics file. Those two files are mandatory for every run. They are detailed further.

2.2 Generation of particles

Once the input parameters read, the program will generate initial particles coordinates using one of the following distribution: KV (Kapchinski-Vladimirski), waterbag, r1r2 or Gaussian. The choice of the type of distribution as well as the number of particles must be set in the collimator file.

Note that the user can also choose to read the initial coordinates from a file.

2.3 Tracking

The particle coordinates are transformed element by element. The tracking is done with linear transfer matrices in x, y, $\frac{dx}{ds}$, $\frac{dy}{ds}$, t and $\frac{\delta p}{p}$, linear dispersion in bending magnet, chromaticity in quadrupoles to leading order, sextupoles in thin element kick approximation and no acceleration. A definition of the particle coordinates can be found in the appendix.

To be more accurate, the tracking is following two different steps.

2.3.1 Tracking between the primary collimators

The first tracking is done only between the primary collimators (TCP, CRYSTAL or TCRYO collimators). When a particle hits a collimator, its starting coordinates in this turn are saved. The transport of the particles is done by Twiss transforms. A blowup of the beam is done if the blow up period divides the revolution number.

2.3.2 Detailed tracking

Then, a second more detailed tracking is done between all the elements of the accelerator lattice, using as input coordinates the ones saved during the first tracking (coordinates at the beginning of the turn in which the particle hit a primary collimator). During that tracking, we apply some special treatment to the collimators to simulate precisely the interactions between the particles and the collimator's material. We also determine if some particles hit the aperture during the run.

Note that it is possible to directly do a detailed tracking between all the elements, without primary tracking.

2.4 Special treatments for the collimators during the detailed tracking

As explained before, during the second detailed tracking, the collimators are studied more in detail. To do that, it exist four different ways of describing the effect of collimators.

2.4.1 Standard collimators

This treatment can apply to a collimator with amorphous material. To describe the interaction between the particles and the material, we use cross-section tables for nuclear fragmentation and electromagnetic dissociation processes which are produced separately with FLUKA. If there is no cross-section table for an isotope, an analytical approximation is calculated. The ionization energy loss is modelled by the Bethe-Bloch formula for heavy ions with shell and density corrections. The multiple scattering is described using a Gaussian approximation of the scattering distribution. The effective particle path L_{eff} is calculated at impact time for each particle. Then, if $L_{eff} > 10L_{int}$, with L_{int} the interaction length, the particle is assumed to be absorbed, otherwise the probability for a fragmentation process is randomly computed using the cross-section tables. Here, in the case of a fragmentation, only the heavier isotope is kept.

2.4.2 Magnetic collimators

This treatment is similar to the previous one, except for the fact that a magnetic field is here present. Also note that, during the detailed tracking, the distance between the collimator's edge is increased.

2.4.3 Fluka collimators

For this treatment, we use the program FLUKA to describe the interaction between the collimator and the particles. It can be applied to collimators with amorphous material.

2.4.4 Crystal collimators

This is a specific treatment for crystals used as collimators. The routine used to simulate interaction between particles and a crystal collimator has been written by Yannick Bel-Hammar, derived from a routine written in FORTRAN77 by Valentina Previtali based on an original algorithm of I.A. Yazynin.

2.5 Outputs

At the end of the run, some scripts are created. They can be directly read with Gnuplot to have some graphical output related to the run. Some other files are also created to give other information about the previous run.

3 How to install and compile ICOSIM++

ICOSIM++ can be installed on every Unix plateform running the g++ compiler. To use it at its best level, the user will also have to install a version of the FLUKA program (to do that, please find more info on *http://www.fluka.org/fluka.php*). The library FlukaIO is also needed to link ICOSIM++ with FLUKA. This library can also be found on svn *https://svnweb.cern.ch/world/wsvn/FlukaIO* and check out from *https://svn.cern.ch/reps/FlukaIO*.

The first part to be compiled must be the FlukaIO library. To do that, the user will just have to follow the instructions in the *README* of FlukaIO. Be careful to have the paths well defined in the Makefile.

The next step is to compile the part in *branches/flukaIntegration/fluka*. Before doing it, the user must be sure to have Fluka well installed and the environnemental variable FLUPRO well defined. Be careful to also redefine the paths at the beginning of the Makefile. Please see the Fluka documentation for more infos. If Fluka is installed and FLUPRO defined, then the only thing to do is to type:

```
make
```

under *flukaIntegration/fluka*.

Note that if you don't want to use ICOSIM++ coupled with FLUKA, you can skip that point and just install FlukaIO.

The final compilation must be done under the (*source*) directory. First of all, in the *makefile*, set the right path to the FlukaIO directory (FLUKAIOlib). Be carefule that the library name could contain 32 or 64 depending on which machine you run FlukaIO. Then, if you want to build ICOSIM++ with files that can be used to test the code (for programmers), just call the makefile with

```
make
```

If you just want to use ICOSIM++, you can type

```
make partial
```

and you will have only the executables used to run ICOSIM++.

Using ICOSIM++, you will have the possibility to make all the files related to a run appear in one folder specific to that run. This will allows you to make other simulations without crashing any previous result or to redo the same simulation being sure to use the right files. It could be also very useful when studying several runs to be sure that right files are corresponding to the right runs. To use this feature, you will have to run ICOSIM++ with the scripts provided in the *script* folder.

To use the scripts, before running ICOSIM++, you will have to rename them from *script_name.sh.sample* to *script_name.sh* under the *script* directory. You may also have to change the mode of these scripts to execute them. To do that type:

```
chmod +x runicosim++.sh
chmod +x runwithfluka.sh
```

Remark 3.1. If you are running ICOSIM++ on a Mac (at any cas on OSX Lion), you will have to modify a line in the scripts. Please change the line:

```
RUNDIR=$(mktemp -d -p $PWD ${1}.XXX)
```

to

```
RUNDIR=$(mktemp -d $PWD ${1}.XXX)
```

Remark 3.2. If you want to use ICOSIM++ coupled with FLUKA, you will have to use the script *runwithfluka.sh.*

Remark 3.3. It is clear that these two scripts are just examples that can be used as a basis to construct more elaborated ones.

Remark 3.4. Don't forget to decompress the cross section folder in the sample repository before running ICOSIM++.

4 How to run ICOSIM++

There are different ways of running ICOSIM++, using different inputs. So let's first have a look at the input files.

4.1 Input files

The most imprtant input files needed by ICOSIM++ are the collimator file and the optics file, the first one describing the collimators and the second one the optics. Some examples of such files are already in *trunk/sample.* The user must in any case check that the values in these files correspond to what he wants to simulate. We describe those files in this section.

It exists also some other input files that are also quickly described below.

The input files must be in the same directory, which has to be given to the program in the collimator file.

4.1.1 The collimator file

The collimator file is the input file used to describe specifically the collimators. It is a .csv (comma separated) file and its name MUST be *collimatorfile_<specific name>.csv.* The *<specific name>* stand for the name chosen by the user. The collimator file decomposes in two parts.

The first half of it is a list of parameters with their values. At least, the following parameters must be there:

- OPTICSPATH: name of the optics file;
- INPUTPATH: path where the input files are;
- BEAMFLAG: circulating direction in which the particles turn into the machine (1 if in the sense of the MAD element sequence, 2 if the opposite);
- MASSNUMBER: number of nucleons in the projectile;
- CHARGESTATE: charge of projectile;
- MASS: rest mass of the projectile [GeV/c^2];
- ENERGY: total energy of the projectile, [GeV];
- EX: emittance in $x - x'$ plane [m $\cdot$ rad];
- EY: emittance in $y - y'$ plane [m $\cdot$ rad];
- NPART: number of particles per bunch in the beam;
- KBUNCH: number of bunches;
- SIGDPP: standard deviation $\frac{p - p_0}{p_0}$ for an ion in the beam, where p is the momentum of a particle, and p_0 is the momentum of the reference particle;
- TAUBEAM: expected lifetime of the beam [s];
- NPARTI: number of particles we consider in the simulation;
- PARTDISTR: initial particle distribution (kv, waterbag, r1r2 or gauss);
- R1R2SKIN: describes the thickness of the initial distribution shell when PARTDISTR = "r1r2";
- NREV1: number of revolutions for the linear tracking;
- NREV2: number of revolutions for the nonlinear tracking;
- BLOWUP2: the square of the blow-up parameter that determines the diffusion in the linear tracking;
- BLOWUPPERIOD: the beam diffuses every BLOWUPPERIOD revolution during the linear tracking;
- STOPLINEARTRACKING: in the beginning of the first revolution of the nonlinear tracking, the transformation between the elements is linear, STOPLINEARTRACKING is the name of the element where the linear tracking stops;
- SCALEORBIT: scaling factor for closed orbit coordinates;
- WECOLLI: tilt angle of the collimators [rad];
- NSIGI: size of the initial beam distribution [sigma];

- CROSSSECTIONPATH: path to the folder containing cross sections data;
- PLOTFLAG: 'Yes' to make the plot of the trajectory, 'No' otherwise;
- THICKNESSMAGNETICFIELD: distance from the edge of a magnetic collimator to the point where the field strength is about 10% of BMAX [m];
- BMAX: maximum strength of the magnetic field [T];
- DELTAGAP: how much we move the collimator edge when using magnetic collimators [m];
- FREQRF: frequency of the rf cavity [Hz];
- RFVOLTAGE: voltage of the rf cavity [V];
- RFHARMONBER: harmonic number for the rf cavity;
- RUNNINGFLAG: choice of the type of tracking (see below);
- IDPART: choice of a particle to spy (see below);
- IDELT: choice of an element after which print some coordinates (see below);
- OUTCOORD: choice of the type of coordinates printing (see below);

This parameters can come in any order, and the file may also contain other parameters.

The second part of the collimator file is a list of the collimators that are in the lattice. For each of them, we have the parameters:

- NAME: name of the collimator as in the optics file (this name MUST finish with *.B1* or *.B2*, this corresponding to the BEAMFLAG parameter. (Note that if the name of the collimators in the optics file don't finish with *.B1* or *.B2*, the user will have to change it manually));
- ANGLE: postition angle of the collimator in the vacuum chamber (horizontal, vertical or skew) [deg]. The angle should be zero if you are using Fluka;
- LENGTH: length of the collimator;
- NSIG: opening gap of the collimator jaws [sigma];
- MATERIAL: collimator's material, which is also the name of the folder where we can find material parameters and cross sections information (carbon, copper, iron, MARS or tungsten);
- METHOD: the kind of collimator we are considering (standard, magnetic, crystal or fluka).

The order of these parameters must be this given above. Be careful to have a header line between the two parts of the collimator file, containing the name of the parameters of the second part. All the parameters must be written in capital letters.

Remark 4.1. If we use a crystal collimator, the only important values in the second part of the collimator are the name and the method, but it must be seven values for each collimator in that file. So for a crystal collimator please use dummy values for the angle, length, phase, nsig and material.

Remark 4.2. Note that ICOSIM++ is also working with the collimator files used for ICOSIM. In that case, it is not possible to print the coordinates of the particles, the RF cavity is seen as another element and all the input files must be in the same directory as the source code.

Remark 4.3. If the user wants to make a simulation without computing the effect of the RF cavity, he could do it by deleting the RF parameters in the collimator file.

4.1.2 The optics file

The second input file is the optics file, which is basically describing in detail the lattice of the accelerator. It is also a .csv file and stand in two parts. It's name MUST be *optics-file_<specific name>.csv* and the *<specific name>* must correspond to the one chosen for the collimator file.

The first part contains a list of parameters which are not described here, because we don't use them in the simulation. Anyway, these parameters can be useful to have a more precise idea on the run.

The second part of the optics file is a list of all the elements in the lattice. For each element, we have the following parameters:

- NAME: name of the element;
- KEYWORD: keyword;
- PARENT: parent;
- S: position of the element measured as the distance along the accelerator [m];
- L: length of the element [m];
- K0L: $K0L = K0 \cdot L$, where $K0$ is the normal dipole coefficient, and L the magnetic length;
- K0SL: $K0SL = K0S \cdot L$, where $K0S$ is the skew dipole coefficient, and L the magnetic length;
- K1L: $K1L = K1 \cdot L$, where $K1$ is the normal quadrupole coefficient, and L the magnetic length;
- K1SL: $K1SL = K1S \cdot L$, where $K1S$ is the skew quadrupole coefficient, and L the magnetic length;
- K1L: $K2L = K2 \cdot L$, where $K2$ is the normal sextupole coefficient, and L the magnetic length;
- K1SL: $K2SL = K2S \cdot L$, where $K2S$ is the skew sextupole coefficient, and L the magnetic length;

- XC: horizontal position x of the closed orbit referred to the ideal orbit [m];
- PXC: horizontal canonical momentum p_x of the closed orbit referred to the ideal orbit, divided by the reference momentum;
- YC: horizontal position y of the closed orbit referred to the ideal orbit [m];
- PYC: horizontal canonical momentum p_y of the closed orbit referred to the ideal orbit, divided by the reference momentum;
- TC: $TC = -c \cdot t$, velocity of light times the negative time difference with respect to the reference particle [m]. A positive TC means that the particle arrives ahead of the reference particle;
- PTC: $PTC = \dfrac{\Delta E}{p_s c}$, energy error divided by the reference momentum times the velocity of light;
- BETX: amplitude function β_x [m];
- ALFX: correlation function $\alpha_x = -0.5\beta'_x(s)$;
- MUX: phase advance μ_x [2π];
- DX: dispersion of x [m];
- DPX: dispersion of p_x;
- BETY: amplitude function β_y [m];
- ALFY: correlation function $\alpha_y = -0.5\beta'_y(s)$;
- MUY: phase advance μ_y [2π];
- DY: dispersion of y [m];
- DPY: dispersion of p_y;
- APERTYPE: type of aperture (RECTANGLE, ELLIPSE, CIRCLE, RECTELLIPSE or NONE);
- APER_1: aperture parameter;
- APER_2: aperture parameter;
- APER_3: aperture parameter;
- APER_4: aperture parameter.

The optics file comes in fact from the program MAD-X, so more detailed explanation of these parameters must be found in the MAD-X documentation. See for example [2].

Note that the parameters above can appear in any order in the optics file, but that it must be a header line just before the beginning of the list of the lattice elements containing the parameter names and giving the order used below.

Giving a Twiss file (a direct output of MAD-X), the user can transform it in an optics file compatible with ICOSIM++ using the routine *twiss_optics*, included in ICOSIM++. To do it, just type

```
./twiss_optics
```

and follows the indication the program will give to you.

4.1.3 Input parameters for crystal collimators

If we make a simulation with one or more crystal collimators, we must have another input file giving their parameters. This file is named *crystalinfo.csv.* In this file, each line represent a crystal and for each crystal, we have the following parameters in this order:

- NAME: name of the crystal as in the collimator file;
- C_orient: crystal orientation (1 for 110, 2 for 111);
- IS: integer that define the substance (0 for Si, 1 for W, 2 for C, 3 for Ge) for a crystal;
- C_xmax: maximum in the x direction (dimension of the crystal) [m];
- C_ymax: maximum in the y direction (dimension of the crystal) [m];
- Cry_length: crystal length [m];
- Rcurv: curvature radius of the crystal [m];
- C_rotation: rotation angle vs vertical axis and crystal axis [rad];
- C_aperture: total aperture of the crystal collimator [m];
- C_offset: if the crystal is considered contained in a box, shift made from a face of that box touching the lateral axis of the crystal in order to lay on the bottom corner between the lateral and the entrance face of the crystal [m];
- C_tilt: angle between the entrance of the crystal and the vertical axis [rad];
- Cry_tilt: crystal tilt [rad].

The last parameters are related to the position of the crystal. The change of referential used in the crystal routine is detailed in the appendice to have a better understanding of these parameters.

4.1.4 Input file for the graphical output

Some parameters related to the final graphical outputs have to be set in the file *infoPlot-LossSpectra.csv.* This file contains the following parameters used in the method *PlotLossSpectra*:

- interactionPoint: which IP we choose for the plot (the name of an IP element);
- d1: starting point (distance after IP [m]);

- d2: ending point (distance after IP [m]);
- selflag: to indicate if we want to show the difference between the different isotopes ('Yes' or 'No');
- fragcutoff: cutoff if selflag is 'Yes';
- PowerOrPartFlag: determine if the plot shows power of particles ('Power' or 'Particles');
- decdPoP: determine if the plot shows particles with charge-mass ratio ('Anything', 'Bigger than reference particle' or 'Smaller than reference particle');
- s1: starting point [m] (if no IP element is specified);
- s2: ending point [m] (if no IP element is specified).

4.1.5 Cross section folder

The cross section folder contains information used if interaction happens between particles and a standard or a magnetic collimator. It contains files related to different kind of material (carbon, copper, iron, MARS and tungsten). In each of these files, we have a file called *materialinfo.csv* which contains the parameters

- stre: total EMD cross section for the main projectile particle on the collimator material;
- strh: total hadronic cross section for the main projectile particle on the collimator material;
- atarget: mass number of the target material;
- ztarget: atomic number of the target material;
- rho: material density [g/cm^2];
- abbreviation: chemical symbol of the material, which is also the first letters of the cross section files;
- X0: Sternheimer parameter needed for density correction in the Bethe-Bloch formula;
- X1: Sternheimer parameter needed for density correction in the Bethe-Bloch formula;
- m: Sternheimer parameter needed for density correction in the Bethe-Bloch formula;
- a: Sternheimer parameter needed for density correction in the Bethe-Bloch formula;
- C: Sternheimer parameter needed for density correction in the Bethe-Bloch formula.

Every material file also contains cross section files. The name of such files has the structure:

chemical symbol - projectile mass number *Apr* - projectile atomic number *Zpr* - "nuclEM.dat"

They contain tabulated total cross sections (hadronic and EMD) for the production of different isotopes. We can see in the first row the negative of *Apr*, the negative of *Zpr* and the total cross section in this order. The following rows contain the change in mass number, the shift in atomic number and the partial cross section for the production of each possible daughter.

4.1.6 FLUKA input file

For a use of ICOSIM++ with a Fluka collimator, the program needs an specific input file for the program FLUKA. It must be a .inp file and be written as described in the FLUKA documentation (see for example the online documentation on *http://www.fluka.org/fluka.php*). Please note that you must use the SOURCE card at the beginning of your input file and the USRBDX at the end of it, to permit the connection between ICOSIM++ and FLUKA. The rest of the FLUKA input file depends on the element you want to describe.

4.1.7 Initial description of the particles in a file

ICOSIM++ offers the possibility not to generate the initial bunch of particles, but to take it from a file. This possibility can be useful if the user wants to use special particles for its run. To do that, he must put the particles coordinates in the file *initial.dat.* In this file, every line will correspond to a particle and the parameters needed for each particle are, in this order:

- x position of the particle [m];
- x' of the particle;
- y position of the particle [m];
- y' of the particle;
- $\delta = \dfrac{p - p_0}{p_0}$: relative momentum offset [GeV/c].

Remark 4.4. Please note that some problems can appear during the reading of the input files regarding the exploitation system on which you are running ICOSIM++ and the coding of the input files. Please be sure that the input files have the right encoding corresponding to your exploitation system. If it is not the case, you can convert the files from an encoding to another using Perl or another converter. For example, to convert a Mac OS text file to a Unix text file at the Unix shell prompt, enter

```
perl -p -e 's/\r/\n/g'  < macfile.txt > unixfile.txt
```

To convert a Unix text file to a Mac OS text file, at the Unix shell prompt, enter

```
perl -p -e 's/\n/\r/g' < unixfile.txt > macfile.txt
```

To convert a Windows text file to a Unix text file, enter

```
perl -p -e 's/\r$//' < winfile.txt > unixfile.txt
```

and to convert a Unix text file to a Windows text file, enter

```
perl -p -e 's/\n/\r\n/' < unixfile.txt > winfile.txt
```

4.2 Running ICOSIM++

In this section, we see more in detail how to run ICOSIM++ and how to use ICOSIM++ with his full possibilities.

First of all, you must set the variable RUNNINGFLAG into the collimator file. Set it to:

- 1 if you want to generate a bunch of particle and to compute first a linear tracking between the primary collimators and next a non linear tracking between each elements of the particles hitting primary collimators;
- 2 if you want to do the same kind of tracking as above, but with initial particles coordinates taken from the file *initial.dat*;
- 3 if you want to generate a bunch and directly do a non linear tracking with all the particles between each elements of the lattice;
- 4 if you want to do the same kind of tracking as 3, but with initial particles coordinates taken from the file *initial.dat*.

Basically, there are two different ways to run ICOSIM++, depending on the kind of collimators that you want to simulate, and depending on if you have to link ICOSIM++ with Fluka or not.

4.2.1 Direct run with only standard, magnetic or crystal collimators

The first possibility is to run ICOSIM++ using only standard, magnetic or crystal collimators. To do it directly, without using any script, you will just have to type

```
./icosim++ <collimator file path> <output path>
```

under the *source* directory. Here *<collimator file path>* is a path to the collimator file that you want to use for the simulation and *<output path>* is a path to the folder in which you want to have the outputs.

4.2.2 Run with only standard, magnetic or crystal collimators using the script *runicosim++.sh*

To run ICOSIM++ in that way, go under the *script* directory and type

```
./runicosim++.sh <specific name>
```

with <specific name> corresponding to the specific name used for the collimator and optics files.

All the files related to the run are then put under the directory *<specific name>*.*. This allows you to make another run without crashing anything from the previous one. All the information related to the run number * will stay under its own repository.

4.2.3 Run with a Fluka collimator

If you want to run ICOSIM++ with a Fluka element, you will have to use the script *runwith-fluka.sh*.

To run ICOSIM++ in that case:

```
./runwithfluka.sh <FLUKA input file> <specific name>
```

with <FLUKA input file> the name of the FLUKA input file describing the Fluka collimator, without .inp, and <specific name> corresponding to the specific name used for the collimator and optics file.

Here too, a repository named this time _<FLUKA input file>.*_, is created with all the output files.

4.3 Output files

We will now describe the ouptuts of ICOSIM++.

4.3.1 Standard output

The most standard output is the file *output.txt* which is created at each run. This file gives information about the run and the hitting positions of particles. We can see in this file:

- ip: position of the *TCP, CRYSTAL* and *TCRYO* collimators in the lattice;
- is: position of the *TCS* collimators in the lattice;
- ips: position of all the collimators in the lattice;
- nsig: initial size of the beam [m];
- Apr: mass number of the reference particle;
- Zpr: charge of the reference particle;
- nparti: number or particles in the simulation;
- PlossPb: average power loss of the beam;
- Aphit: mass for particles hitting the aperture;
- Zphit: charge for particles hitting the aperture;
- asumrem: number of particles staying in the accelerator for each turn;
- asumhitcolli: number of particles that have hit a collimator during the current turn or a preceding one at each turn;
- asumhits: number of particles that have hit another element during the current turn or a preceding one at each turn;
- nhitcolli: number of particles that have hit the collimators, for each collimator;

- hits: hit position along the accelerator for the particles that have hit another element than a collimator;
- taubeam: expected lifetime of the beam [s].

Note that *asumrem, asumhitcolli* and *asumhits* are not updated during the linear tracking at the beginning of the first revolution.

You can also follow the run of the simulation in the file *tracker.log* under *<specific name>.** or *<FLUKA input name>.**. If you are running ICOSIM++ with FLUKA, the file *server_1.log* will show you how FLUKA has been used as the server. In that case, more details on the particle exchange between ICOSIM++ and FLUKA can be found in the file *<FLUKA input file>001.out* under the directory *1*. You can follow there the run of FLUKA.

4.3.2 Special output for a run with a crystal collimator

When runned with a crystal collimator, ICOSIM++ also produces the file *crystal_output.out*. This file gives more detailed information on the crystal and on the passage of the particles throught it (number of particles in amorphous, dechanneling, channeling, volume reflexion, volume capture mode).

4.3.3 Graphical outputs

At the end of each run, the scripts *PlotRunSummary1.p*, *PlotRunSummary2.p*, *PlotRunSummary3.p* and *PlotLossSpectra.p* are created. If the parameter PLOTFLAG is set to 'Yes' in the collimator file, we also have the scripts *PlotTrajectory1.p* and *PlotTrajectory2.p*.

This scripts contains commands that can be read by Gnuplot to produce some graphical outputs. To print some plot, just open Gnuplot and load the script typing in the Gnuplot terminal:

```
load 'script_name.p'
```

We have some examples of the different graphical outputs in the appendix.

The three first plots gives a good summary of the run.

The script *PlotRunSummary1.p* produces a plot of the power loss in the ring as a function of *s*. The position of the IP elements is also shown on the plot.

The script *PlotRunSummary2.p* produces a plot of the power load on each collimator averagely during the beam lifetime.

A plot representing how many particles are left in the beam, how many particles have been lost on a collimator and how many particles have been lost on the aperture as a function of the number of revolutions can be drawn using the script *PlotRunSummary3.p*.

The second kind of graphical output is related to the losses on the aperture along the ring during the last turn. The plot is obtained using the script *PlotLossSpectra.p* and depends on the parameters in the file *infoPlotLossSpectra.csv*. On this plot, we can see the power load or particle loss rates along the ring, which can differentiate the different isotope or no. We have also on this plot the aperture, the dispersion function and the beta function (in the x direction). The position of the IP elements is marked as well as the position of the RBEND, SBEND, QUADRUPOLE and RCOLLIMATOR. These kind of elements are represented by a box which width is equal to the distance between that element and the previous one.

The last kind of possible graphical outputs are plots showing the trajectory of the particles, the trajectory of the reference particle, the beam envelope, the aperture and the collimators for the last turn. These plot can be obtain using *PlotTrajectory1.p* for the x-direction and *PlotTrajectory2.p* for the y-direction. Let's note that the parameter PLOTFLAG must be set to 'Yes' in the collimator file to have this kind of plots.

4.3.4 How to print the coordinates of the particles

There are three useful parameters that are made for printing the coordinates of a chosen particle at special moments (IDPART, IDELT and OUTCOORD). They can be modified in the collimator file.

The first one is the parameter IDPART. This parameter represent the ID of the particle which will be spied.

With the parameter IDELT, the user can set the number of the element after which he wants to have the coordinates.

The parameter OUTCOORD determine the kind of printing.

Using them, the user can see the coordinates of the particles trough different methods.

The first one is the function *outCoord*, which prints the coordinates for a chosen particle between every element in the file *coordinates.dat*. To use it, you must first of all set the particle that you want to track. To do it, set the parameter IDPART to the identification corresponding to the particle that you want to track and set the parameter OUTCOORD to 3.

The second funtion is *outPunctual*, which print the coordinates at each turn at a certain point of the accelerator. To do it, you will have to choose the particle that you want to track as above and to choose the number of the element after which you want the coordinates using the parameter IDELT. Then set OUTCOORD to 4 and the coordinates will appears at the end of the run in the file *coordinates_punctual.dat*.

The user can also print the coordinates of all the particles after a chosen element. To do that, he will have to set IDELT with the chosen element and OUTCOORD to 2.

Another very useful way to track the particles coordinates during the run is to set OUTCOORD to 1. This produces an output of the 6 coordinates at each element for all particles during all the simulation. Basically, doing that you will have the coordinates of the particles at every time at every place around the accelerator. This output can be seen in the file *tracker.log* if we use the scripts to run ICOSIM++ or directly in the terminal. Note that doing some kind of printing will make the running time grows.

If the user do not want any output of the coordinates of the particles, he will have to set OUTCOORD to 0.

The coordinates printed are (x, y, x', y' and $\frac{p-p_0}{p_0}$, t) in columns in this order.

We can see a summary of the different ways of printing coordinates in the Table 1.

To print	OUTCOORD	IDPART	IDELT	In the file
Nothing	0	-	-	-
All the part. after all the elts	1	-	-	tracker.log
All the part. after a chosen elt	2	-	Elt nber	coordinates_elt.dat
One part. after all the elts	3	Part. nber	-	coordinates.dat
One part. after a chosen elt	4	Part. nber	Elt nber	coordinates_punctual.dat

Table 1: Different ways to print the coordinates of particles during the simulation.

5 Others executables created by the makefile

By typing *make* in the source directory, the user will also create some other executables which can be used to test or debug different part of the program. For all of them, the interest is that they can be used to test some modifications of the code without the need to make these modifications directly in the source code and using the fact that these files are already built in the makefile and that the necessary files are already included. They could be useful especially for people who wants to make further development of ICOSIM++ and other programmers. To run one of these executables, just type

```
./executable_name
```

These files are detailed there.

5.1 testParticle

In the main of that file, we can test the generation of particles and the coordinates just after the generation.

5.2 testLattice

This file can be used to test the construction of a test lattice composed with elements and collimators and to test the lattice's attributs and methods.

5.3 testElement

We can use this file to test the constructors of the class Element.

5.4 testCollimator

It can be used to test the different methods and attributs of the class Collimator.

5.5 debug

This file can be used to debug or test some part of the code.

A Coordinates used

The coordinates used to described each particle are:

- x: transverse horizontal position (relative to the orbit) [m];
- $x' = \dfrac{dx}{ds}$ (with s the distance along the accelerator);
- y: transverse vertical position (relative to the orbit) [m];
- $y' = \dfrac{dy}{ds}$ (with s the distance along the accelerator);
- t: time [s];
- $\delta = \dfrac{p - p_0}{p_0}$: relative momentum offset.

B Examples of graphical outputs

We show here some examples of the different graphical outputs that can be obtained using ICOSIM++.

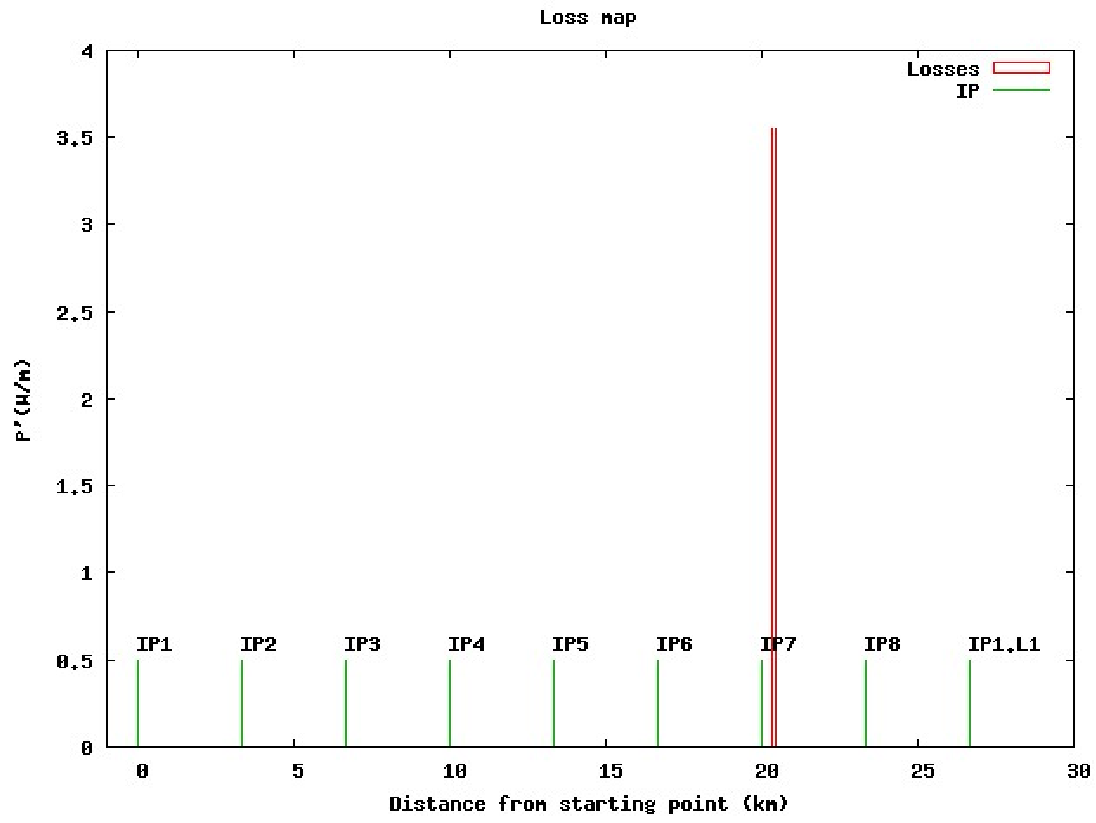

Figure 1: Power loss in the ring as a function of s.

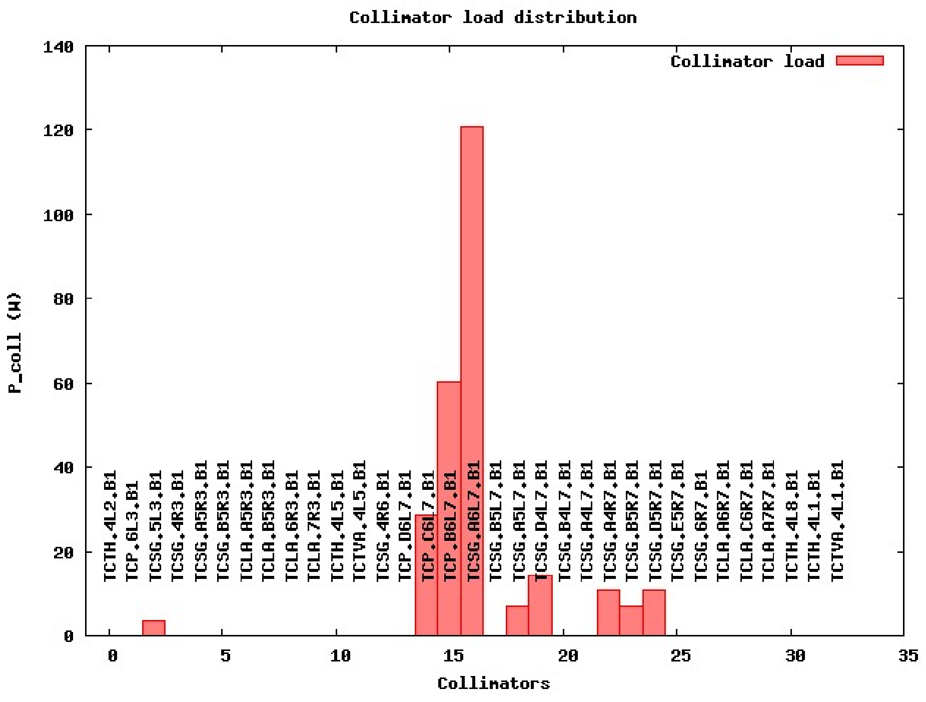

Figure 2: Power load on each collimator.

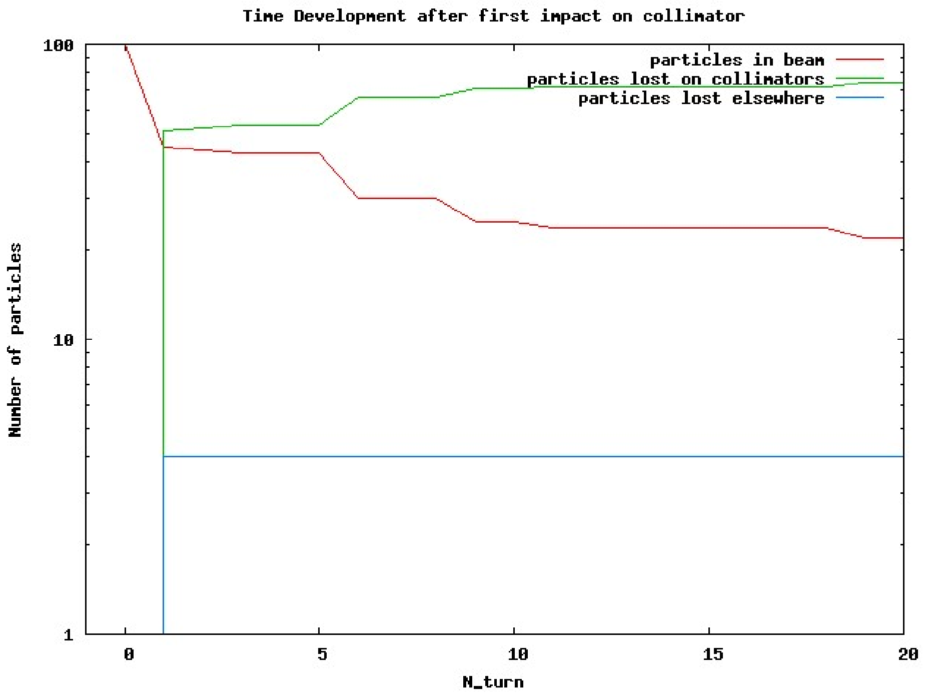

Figure 3: Behaviour of the particles during the run.

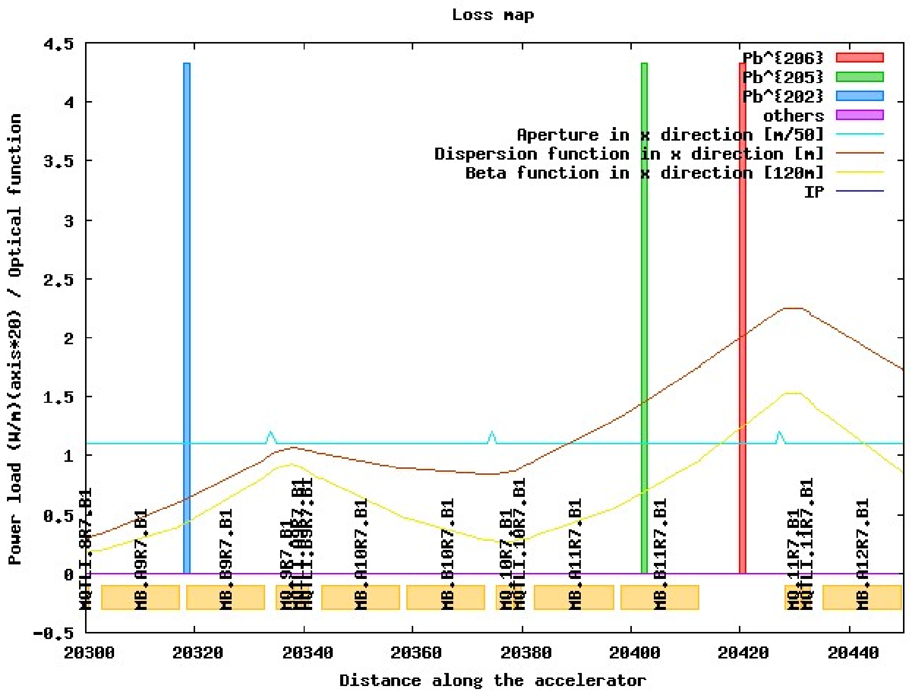

Figure 4: Losses on the aperture.

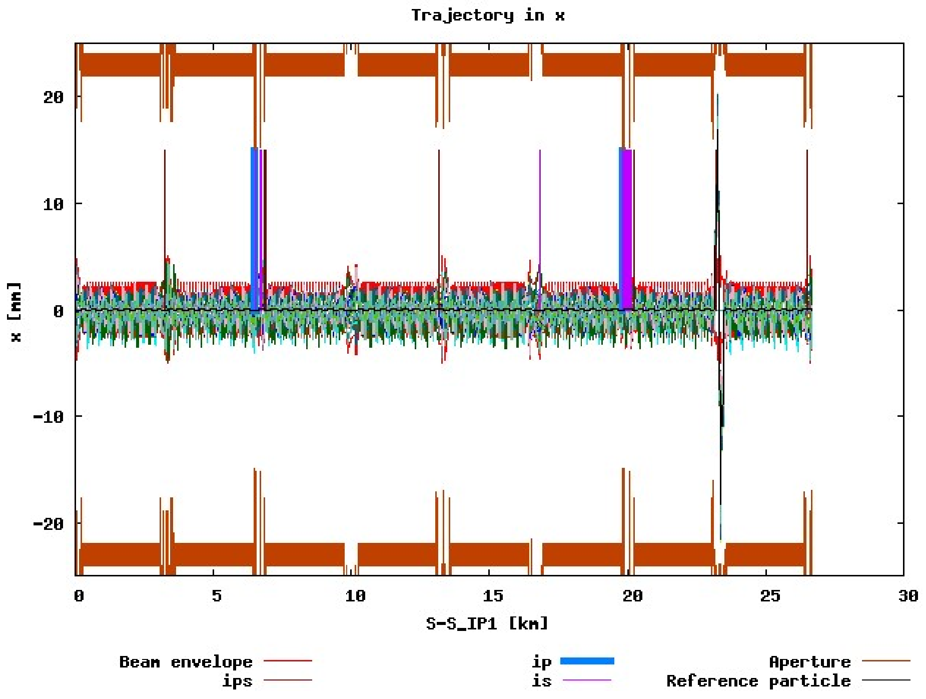

Figure 5: Particles trajectories.

C Change of referential in the crystal routine

More information concerning the crystal routine as well as this plot can be found in [3]. The crystal parameters concerning the position of the crystal in the file *crystalinfo.csv* are related to these different steps.

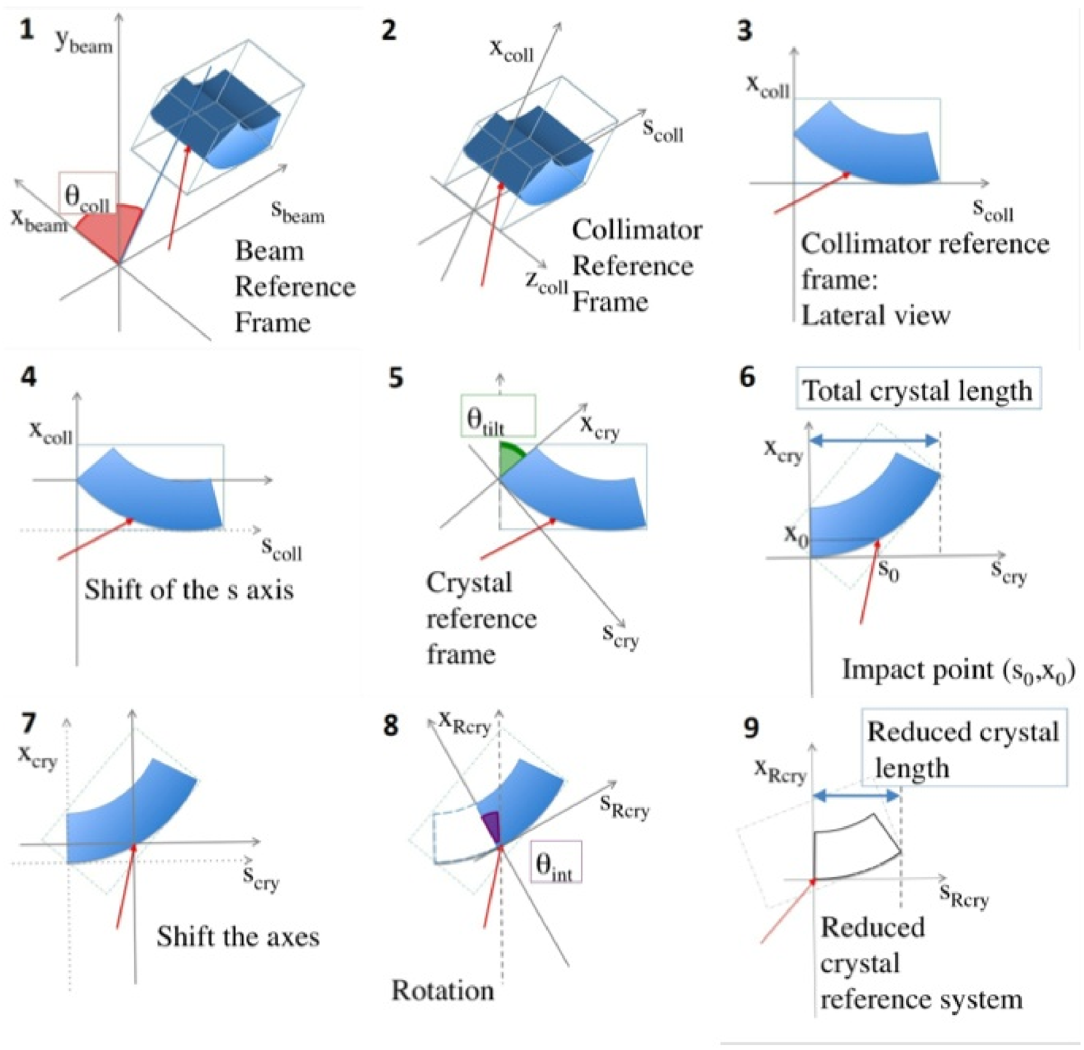

Figure 6: Steps used during the change of referential at the beginning of the crystal routine.

References

[1] N. Holden, *Development of the ICOSIM program and application to magnetised collimators in the LHC*, http://cdsweb.cern.ch/record/1151294/files/AB-Note-2008-054.pdf.

[2] F. Christoph Iselin, *The MAD Program, (Methodical Accelerator Design), Version 8.13, Physical Methods Manual*, http://project-madwindows.web.cern.ch/project-madwindows/MAD-resources/phys_guide.pdf

[3] Valentina Previtali, *Performance Evaluation of a Crystal-Enhanced Collimation System for the LHC*, Thèse Nr. 4794 (2010), Ecole Polytechnique Fédérale de Lausanne (EPFL), Faculté des Sciences de Base, Laboratoire de Physique des Accélérateurs de Particules.