EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE

CERN - PS DIVISION

PS/ DI/ Note 95-26 (Tech.)

Multiturn Injection of Pb ions in LEAR

F. Motsch

Geneva, Switzerland
8 March 1996

Table of Contents

Multiturn Injection and Stacking of Pb ions in LEAR

Fabien Motsch[1]

CERN Geneva, Switzerland

1. Introduction

The need of multiturn injection in LEAR for the LHC program is considered in Refs. [1] and [2]. The LHC [3] is planned to start operation in the year 2004-8 and will have two main physics programs : proton collisions and Pb ion collisions. For the latter, there is a need to have high-density beams which cannot be delivered directly from the ECR ion source used. In fact, there is a factor of 50 between the intensity outgoing from the Linac and the intensity needed for LHC even with multiturn injection in the PS Booster. Therefore, it is necessary to accumulate the pulses from the linac in LEAR (after the antiproton physics program has stopped) re-baptized LEad Accumulator Ring, or in a LEAR-like machine. A LEAR-like machine would re-use the elements from LEAR to make a longer machine or LEAR would be slightly transformed, to move the injection section from SD1 to SD2 (see Figure 7 on page 7).

The information given in this report is quite general with respect to the simulation of injection and accumulation in LEAR. Changing the machine simply consists in changing the lattice file. The injection scheme considered is a multiturn injection (classical transverse injection or combined transverse-longitudinal injection) followed by cooling with an electron cooling device, and stacking before the next pulse.

2. Multiturn Injection

Multiturn injection aims at injecting more particles than it is possible by the usual single turn injection with a septum and kicker magnet. The simplest is the transverse multiturn injection ("classical" multiturn injection), but to increase the efficiency one can think of using in the injection section to achieve combined longitudinal and transverse multiturn injection.

2.1. Classical Multiturn Injection

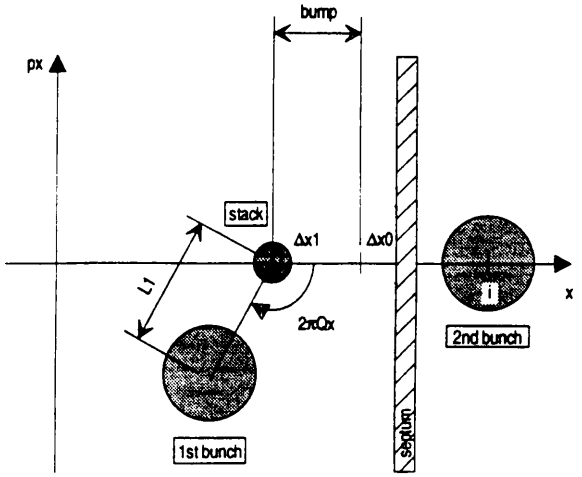
The classical scheme [4] consists in injecting particles at many consecutive turns in an accelerator. A septum magnet deflects the particle trajectories coming from the transfer line. The closed orbit of the machine is displaced in the injection region by bumper magnets to prevent particles from touching the inner edge of the septum after the first turns or later. If the beam is matched to the injection section Twiss functions, it has a circular shape and in both x and y normalized phase space planes its trajectory is a circle centred on the closed orbit. In case of a mismatch, the trajectories remain circles in normalized coordinates but the shape of the beam becomes elliptical. In general a mismatch in the plane at injection permits a more efficient filling of the machine acceptance.

At the beginning of the injection of the first turn, the closed orbit is displaced by Δx_0. During each turn, the bump decreases by a constant value *bump* and at the beginning of the second turn the bump equals Δx_1. The angle of rotation of the particles during one turn is given by the tune factor Q_x. As a consequence, the movement of the particles in phase space is the combination of the translation of the closed orbit (*bump*), and the rotation (angle $2\pi Q_x$) around the closed orbit with the new position after the first turn $\Delta x_1 = \Delta x_0 - bump$. The centre of the par-

1. Fabien Motsch obtained his engineering degree in 1994 from the Ecole Centrale in Paris. He spent 16 months, (from September 1994 to December 1995), at CERN as a French Coopérant and worked in the PS/DI group on multiturn injection in LEAR. He is now preparing a Phd on Aleph at CERN in collaboration with the CPPM in Marseille. More information : motsch@afsmail.cern.ch

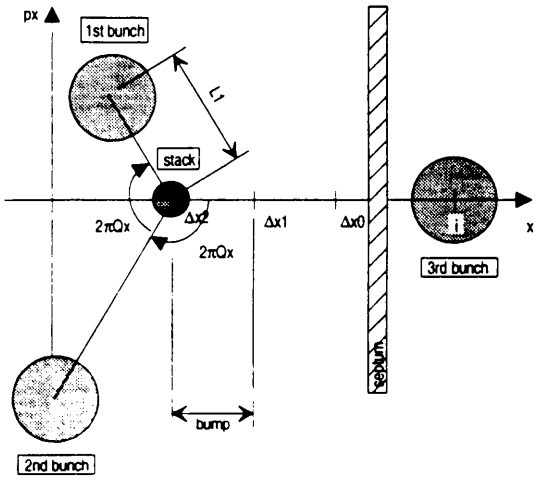
ticles distribution remains at a constant distance $L_1 = i - \Delta x_0$ from the stack (i is the position where the beam is injected).

Figure 1 Mutlitum injection (end of the first turn)



The process of translation and rotation is the same for all injected turns. It is assumed that there is no dispersion in the injection section of the machine or that the beam has a very small constant energy spread if there is dispersion. Then the batches always rotate around the same point : the closed orbit, but at different distances according to the moment when they were injected. L_1 for the first bunch and $L_2 = i - \Delta x_1 = i - \Delta x_0 + bump = L_1 + bump$ for the second one, etc.

Figure 2 Multitum injection (end of the second turn)



2.2. Combined Multiturn and Transverse Injection

Here the dispersion D is non zero in the injection section and advantage is taken of this to increase the number of injected bunches in the machine.

The principle illustrated in Figure 1 is still valid for the first turn in the case of Combined Injection. The closed orbit corresponds to the trajectory of particles with a given momentum. If the particles are not injected on the closed orbit they oscillate around it, following a circular trajectory in transverse phase space x or y. The closed orbit of a particle with a momentum deviation Δp_x is displaced by the dispersion by an amount $\Delta x = D\Delta p_x$ from the trajectory of the same particle without momentum deviation. In other words, one can consider that for this particle the closed orbit is translated by $D\Delta p_x$ in the x direction. To each energy (or momentum) state corresponds a different closed orbit. The combined injection scheme consists in increasing the momentum of the par-

ticles injected (coming from the Linac via the transfer line) at each turn by *deltap*. Then, the displacement of the closed orbit relative to the n^{th} bunch has two components : the bump $n \cdot bump$ and the displacement due to dispersion $Dn\Delta p_x$:

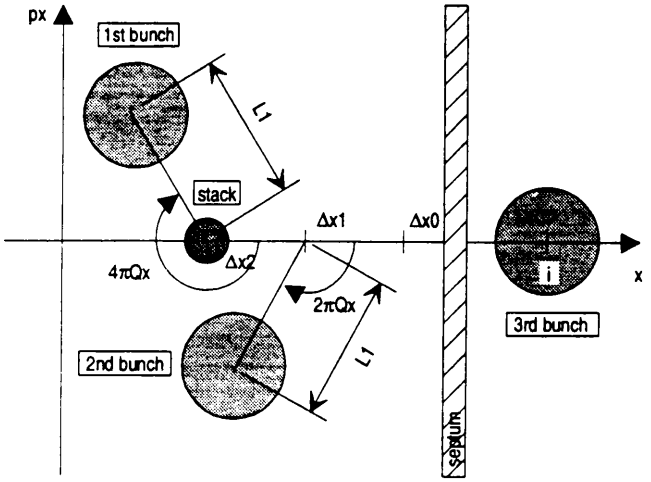
$$\Delta x_{co}(n) = n \cdot \Delta x_{dispersion} - (n \cdot bump) \qquad \text{(Eq.1)}$$

and the position of the n^{th} closed orbit is

$$x_{co}(n) = \Delta x_0 - \Delta x_{co}(n) = \Delta x_0 - (n \cdot D\Delta p_x - bump) \qquad \text{(Eq.2)}$$

So, in the particular case where $D\Delta p_x = bump$ the n^{th} batch "sees the same" closed orbit as the first one placed at Δx_0 as it arrives from the transfer line. In other words, it will rotate around the same point as the first bunch arriving before the first turn. It is as if the n^{th} batch was the first. This also means that every injected turn will rotate around its own closed orbit at the same distance $i-\Delta x_0$. As a consequence, there will be a higher density of particles in phase space than in the classical multiturn injection.

Figure 3 Combined injection



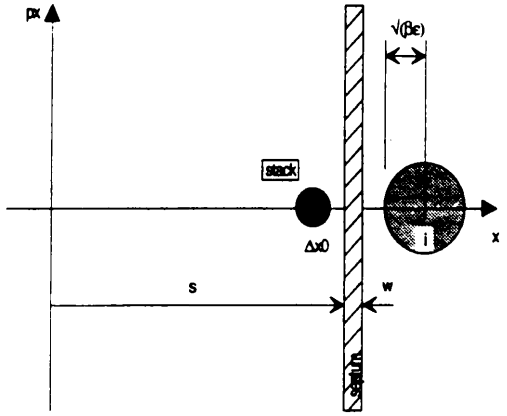
2.3. Injection Parameters

Let's suppose that one injects a matched beam and Twiss parameter $\alpha = 0$ at injection, that is to say, there is no β mismatch so, the injected beam has a circular shape and its trajectory in the normalized phase space of the machine is a circle. The first condition in order to avoid losses at the septum is that the injected bunch is far enough from the outer edge of the septum.

$$i - \sqrt{\beta\varepsilon} \geq s + w \qquad \text{(Eq.3)}$$

where i is the injection position, β is the horizontal beta function at the injection point, ε is the emittance of the incoming beam, s is the position of the inner edge of the septum and w its width.

Figure 4 Injection position



The second condition is on the initial closed orbit bump. It must be compatible with the size of the stack present in the machine.

$$\Delta x_0 + \sqrt{\beta \varepsilon_s} \leq s \tag{Eq.4}$$

A third condition to fulfil is that the beam does not touch the inner edge of the septum after one turn. This can be expressed as :

$$\Delta x_0 - bump + L\cos(2\pi \cdot Q_x) + \sqrt{\beta \varepsilon_b} \leq s \tag{Eq.5}$$

where

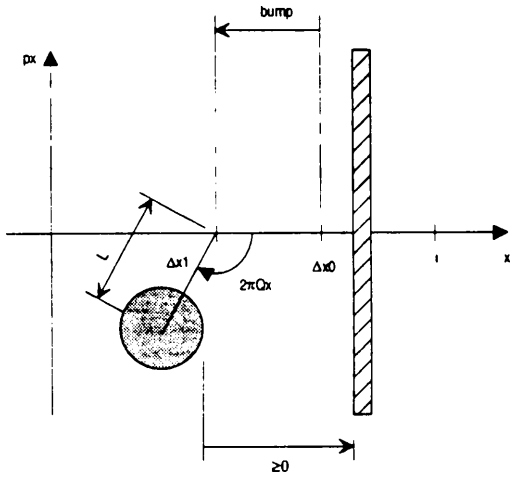
$$L = i - \Delta x_0 \tag{Eq.6}$$

so

$$\Delta x_0 - bump + (i - \Delta x_0)\cos(2\pi \cdot Q_x) + \sqrt{\beta \varepsilon_b} \leq s \tag{Eq.7}$$

$bump > 0$ is the displacement of the closed orbit at each turn and Δx_0 its initial position, Q_x is the horizontal tune factor.

Figure 5 After one turn



So, if one injects the beam at position i given by (Eq.3)

$$i = s + w + \sqrt{\beta\varepsilon_b} \tag{Eq.8}$$

the inequation (Eq.7) becomes

$$bump \geq \Delta x_0 + \sqrt{\beta\varepsilon_b} - s + (i - bump_0) \cdot \cos[2\pi \cdot Q_x] \tag{Eq.9}$$

This formula can be generalized for the n_b^{th} batch after the n_t^{th} turn.

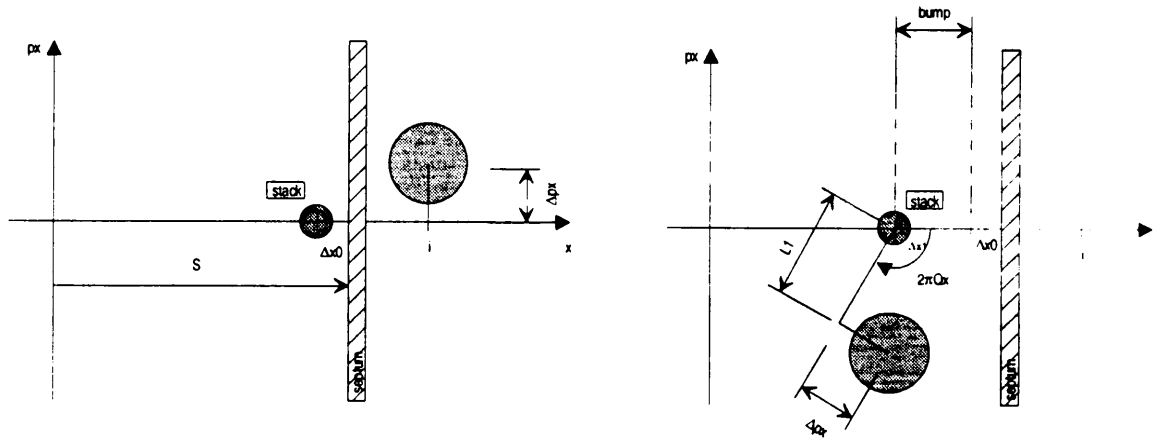
$$bump \geq \frac{\Delta x_0 + \sqrt{\beta\varepsilon_b} - s + (i - bump_0) \cdot \cos[2\pi (n_t - n_b + 1) \cdot Q_x]}{n_t - (n_b - 1) \cdot \cos[2\pi (n_t - n_b + 1) \cdot Q_x]} \tag{Eq.10}$$

The injection position i is determined by (Eq.3) and the initial bump is given by (Eq.4). For a given machine and beam, Q_x and $\sqrt{\beta\varepsilon_b}$ are known as well as the septum position s and width w, and one can choose a bump rate which satisfies (Eq.10) for a couple of values (n_t, n_b). These values should be chosen in order to have $(n_t - n_b + 1) \cdot Q_x$ close to an integer. This corresponds to the n_b^{th} batch coming back close to the inner edge of the septum after n_t turns in the machine. The choice of the bump rate gives the number of turns that can be injected knowing the initial bump Δx_0 and assuming that the bump returns to zero at the end of injection. The choice of the number of turns effectively injected and of the bump rate also depend on the acceptance of the machine.

2.4. Injection at an Angle

It is possible to introduce some refinements in the previous formulae such as in the case of an injection with a non-zero angle. If the injected beam has a certain angle of incidence in the horizontal plane at injection, then the beam translates along the momentum axis in the phase space in the injection section. This angle corresponds to a certain horizontal momentum deviation Δp_x.

Figure 6 Injection at an angle



A new term has to be added to the previous equations. The condition after one turn (Eq.4) becomes (assuming again $\alpha_x = 0$ at the septum) :

$$\Delta x_0 - bump + (i - \Delta x_0)\cos(2\pi \cdot Q_x) + \sqrt{\beta\varepsilon_b} + \beta\Delta p_x \sin(2\pi \cdot Q_x) \leq s \tag{Eq.11}$$

so, the condition on the bump rate $bump$ is now

$$bump \geq \Delta x_0 + \sqrt{\beta\varepsilon_b} - s + (i - \Delta x_0) \cdot \cos(2\pi \cdot Q_x) + \beta\Delta p_x \sin(2\pi \cdot Q_x) \tag{Eq.12}$$

and for the n_b^{th} batch after the n_t^{th} turn

$$bump \geq \frac{\Delta x_0 + \sqrt{\beta \varepsilon_b} - s + (i - \Delta x_0) \cdot \cos\left[2\pi (n_t - n_b + 1) \cdot Q_x\right] + \beta \Delta p_x \sin\left[2\pi (n_t - n_b + 1) \cdot Q_x\right]}{n_t - (n_b - 1) \cdot \cos\left[2\pi (n_t - n_b + 1) \cdot Q_x\right]} \quad \text{(Eq.13)}$$

The use of an injection angle allows to sweep a wider area in the momentum direction of the phase space because the distance between the centre of the injected beam and the stack is

$$\sqrt{L_1^2 + (\beta \Delta p_x)^2} \quad \text{(Eq.14)}$$

instead of L_1 for injection without angle.

3. LEAR Machine Data

3.1. The LEAR Accelerator

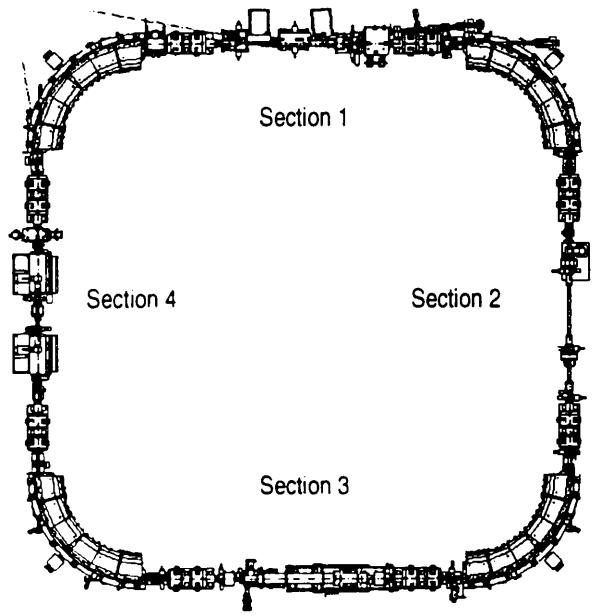
The LEAR (Low Energy Antiproton Ring) accelerator was designed and built in the 1980's in order to provide low energy antiprotons to a large number of experiments. It supplies antiproton beams in the range 60 MeV/c - 2 GeV/c, with different extraction schemes : very slow extraction, slow extraction and single turn extraction.

The accelerator has a square shape (Figure 7) and in the actual configuration each straight section has a specific role in the machine.

- SD1 Injection/ejection
- SD2 Jet Set experiment (internal target and detector)
- SD3 Electron cooling
- SD4 Radio Frequency cavities

This layout was valid for the antiproton program and will be modified for transformation in LEad Accumulator Ring for LHC. For instance, injection and ejection would take place in sections 1 and 4 respectively.

Figure 7 LEAR



3.2. Machine Data

Table 1 contains the important machine data for injection : the Twiss parameters in the injection section. They only depend on the optical set-up of the machine and not on the beam characteristics. They can be adjusted to fulfil injection requirements such as tune or dispersion.

Table 1 Twiss parameters at injection

Machine	a_x	β_x	D_x	α_y	β_y	D_y	Q_x	Q_y
LEAR_Michel	0	2.036	37.687	0	6.294	0	2.314	2.62
LEAR1	0	2.034	37.642	0	6.321	0	2.315	2.621
LEAR2	0	1.384	0.014	0	8.038	0	2.46	2.42
LEAR3	0	9.416	0.066	0	6.021	0	1.8	2.42
LEAR3+	0	10.0	0.008	0	12.435	0	1.85	2.6
Long_LEAR	0	2.938	108.674	0	4.614	0	1.796	2.754
Long_LEAR7	0	2.099	105.291	0	8.217	0	1.60	2.55

4. Remarks on Injection Constraints

There are several constraints to fulfil in order to achieve an efficient injection. The conditions on the injection position and the initial bump can be seen in sections 2.3 and 2.4. There are also conditions on the accelerator itself and more precisely on the Twiss parameters in the injection section. Combined injection requires horizontal dispersion to be non-zero, $D_x \neq 0$. The alpha function is equal to zero because the injection point is a symmetry point of the lattice. The tune factors have to be chosen carefully for multiturn injection. Apart from the resonance conditions which have to be avoided, the horizontal tune factor Q_x should not be too near an integer thus reducing the bump step at each turn. An optimum value is about $1/4$, this allows to inject 4 turns before the first injected turn passes near the septum again. Although, only the horizontal multiturn and combined injection has been studied, the choice of the vertical tune is not completely free. If the working point of the machine is near a first order resonance of the type $Q_x - Q_y = 0$ a coupling between horizontal and vertical motion introduces an exchange of energy between both directions and allows a better repartition of the particles in phase space.

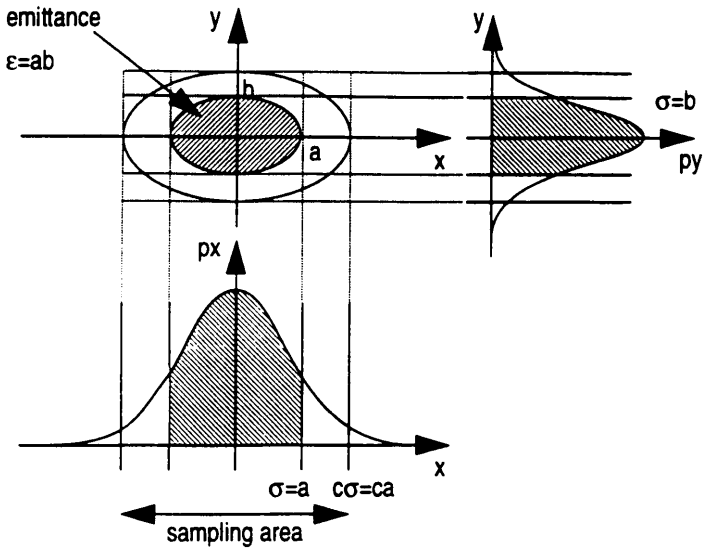
5. Simulation of Multiturn Injection

Multiturn injection is a complex process that depends on a great number of parameters. Simulation is necessary to predict the effects of coupling (due to skew quadrupoles), the effect of dispersion, the loss of particles and the effect of electron cooling. The simulation consists of sampling a beam of particles, to specify the injection parameters such as the injection position, the number of turns and to run a program that tracks the particles along the machine and removes the lost ones.

5.1. Sampling of the Beam

The beam is considered as a 2-dimension Gaussian elliptical distribution in both x and y transverse planes (Figure 8). In the longitudinal plane, the beam is uniformly sampled in a time interval to adjust the bunch length, and the energy (or momentum) is a Gaussian distribution. In fact all the Gaussian distributions are truncated at $c \cdot \sigma$ where the usual value is $c = 1.64$ because the Gaussian distribution in the interval $[-1.64\sigma; 1.64\sigma]$ represents 90% of the probability and a distribution over $[-\infty,\infty]$ has no physical meaning. The emittance is defined by the surface of the ellipse cut at $c \cdot \sigma$, that is to say $\varepsilon = ab$ in $\pi \cdot mm \cdot mrad$ units.

Figure 8 Sampling



Distribution can be truncated to get different percentages of the total Gaussian distribution. Assuming that the total number of particles in the Gaussian distribution is N_0

$$N = N_0\left[1 - e^{-\frac{\varepsilon}{2\varepsilon_\sigma}}\right] \qquad \text{(Eq.15)}$$

with

$$\varepsilon_0 = \pi\sigma^2/\beta \qquad \text{(Eq.16)}$$

and

$$\varepsilon = \pi(\sigma c)^2/\beta \qquad \text{(Eq.17)}$$

$$N = N_0\left[1 - e^{-\frac{c^2}{2}}\right] \qquad \text{(Eq.18)}$$

So, the choice of parameter c makes it possible to get any percentage of the total Gaussian distribution.

Table 2

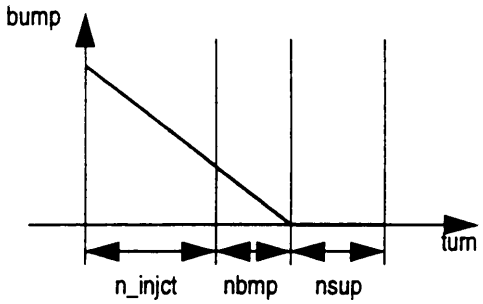
c	percentage
$\sqrt{6}$ = 2.49	95%
2	86%
$\sqrt{2}$ = 1.41	63%

5.2. The Input File

The `bunch.dat` file contains the data necessary to specify injection : data on the beam, bump sweeping, number of turns. Some of the parameters are illustrated in Figure 10.

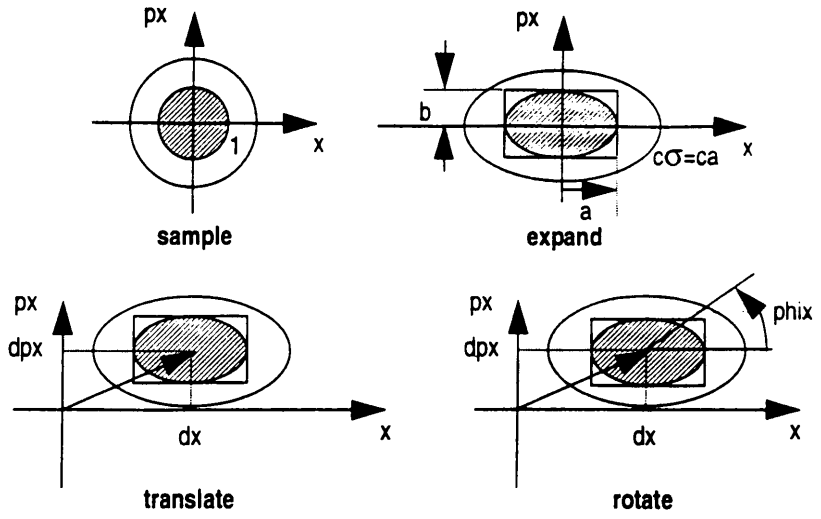
n_injct	number of particles injected per turn
nb_turn	number of turns to be injected
nbmp	number of turns during the bump continues to decrease without injecting particles
nsup	number of turns after injection and bump decrease are completed

Figure 9 Bump definition



x	size of the beam at $1 \cdot \sigma_x$ along the x-axis in metres
xp	size of the beam at $1 \cdot \sigma_x$ along the px-axis
dx	displacement of the centre of the distribution along the x-axis
dxp	displacement of the centre of the distribution along the px-axis
phi_x	angle of rotation of the ellipse in the (x,px) plane
sigma_x	value of σ_x where the emittance ε_x is defined
y	size of the beam at $1 \cdot \sigma_y$ along the y-axis in metres
yp	size of the beam at $1 \cdot \sigma_x$ along the py-axis
dy	displacement of the centre of the distribution along the x-axis
dyp	displacement of the centre of the distribution along the px-axis
phi_y	angle of rotation of the ellipse in the (y,py) plane
sigma_y	value of σ_y where the emittance ε_y is defined
dT	length of the beam in metres
deltat	fraction of the beam length to be plotted (negative, -1<deltat<0)
p	momentum (deviation from the reference momentum)
deltap	momentum spread
sept_pos	position of the inner edge of the septum in metres
sept_width(m)	width of the septum in metres
deltax	initial value of the bump
bumprate(m)	bump decrease per turn in metres
injct_pos(m)	position of injection of the centre of the beam distribution in metres
deltapini	initial momentum deviation from the momentum p
deltaprate	decrease of the momentum per turn
Trev (m)	revolution time along the machine in metres (c.time)

Figure 10 Definitions relative to the sampled beam



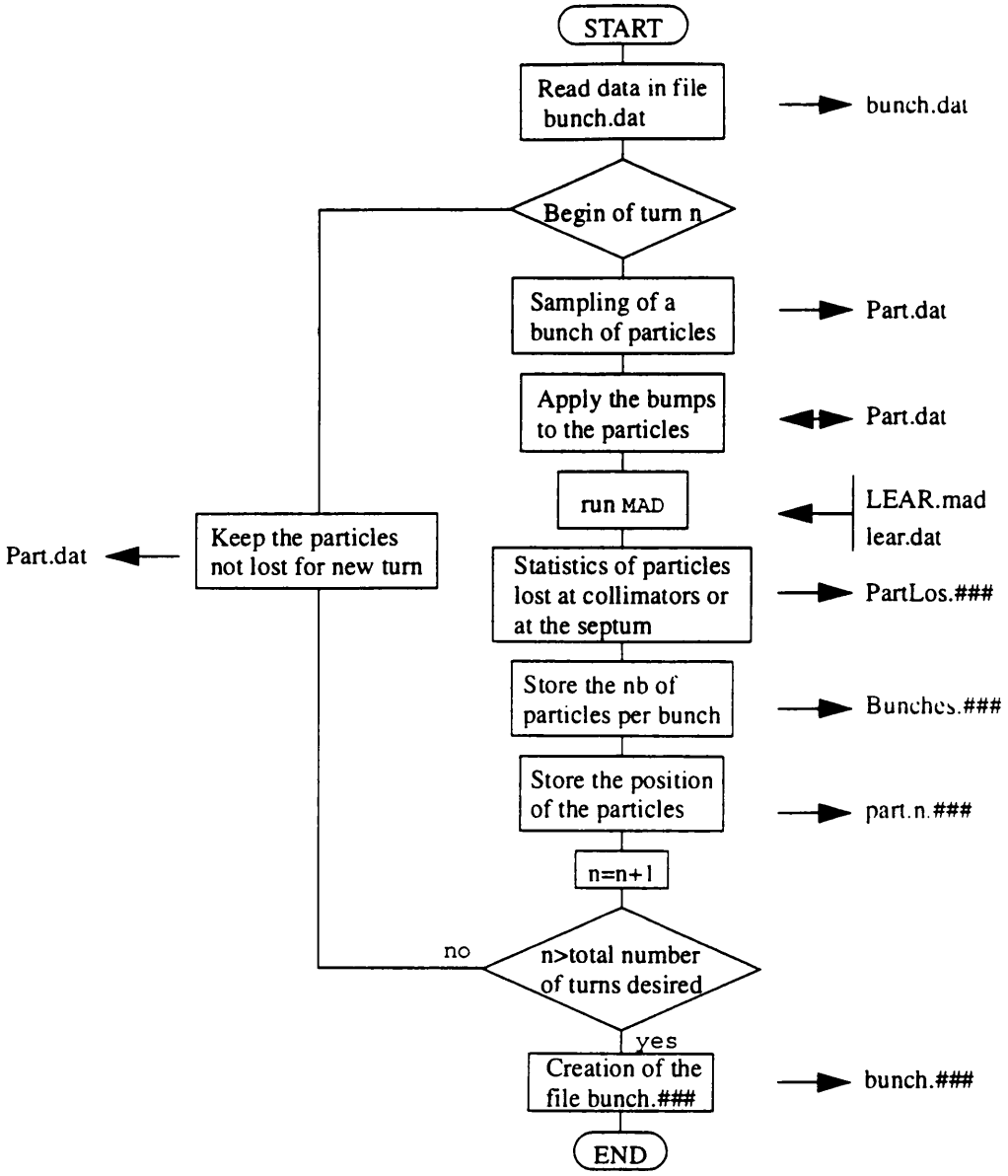
First of all, the program randomly samples a 2-dimension Gaussian distribution over a circle of unit radius that corresponds to σ and cut at radius c. It is then expanded to an ellipse by multiplying the x and y coordinates by the semi-axis lengths of the ellipse. The centre of the distribution is translated by a vector (dx, dpx) and is rotated around its centre by an angle *phix* relative to the x axis. So, the three parameters (dx, dpx) and *phix* make it possible to inject the beam at a certain angle *dpx* and position *dx* and to tilt it by an angle *phix*. This can also be done in the y plane.

5.3. Structure of the Program

Structure of the program is described in Figure 11. The complete listing can be found in Appendix A.2. on page 36 and the details of each routine is in section 8. on page 18. For the moment, the general ideas of the program will be seen.

The core of the program is MAD [6], which performs the tracking of the particles along the machine. First of all, the program reads the parameters defining the beam and the injection in the file `bunch.dat`, then it samples a batch of particles according to these data. The bump needed for multiturn injection (simple transverse as well as for combined) is not implemented in MAD, but is artificially applied to the particles. The closed orbit remains the same during all the simulation, there are no bumper magnets to alter it in the injection area, but the coordinates of the particles injected are corrected so that the distance between a particle and the closed orbit without bump, is the same as the distance between the particle and the closed orbit with the bump (see section 8.1.11. on page 21). Therefore, the particle will undergo the same oscillation as if there was a real bump. After one turn, the results of the tracking by MAD are exploited to get the number of particles lost at the collimators. This information is available in the file `print.#####` which contains the coordinates of the particles lost and where it occurs. Loss at the septum is calculated in the program by examining the coordinates of the particles at the end of the tracking and by correcting their coordinates, relative to the closed orbit to get their absolute coordinates relative to the centre of the machine by tacking the bump into account (see section 8.1.11. on page 21). The coordinates of particles which are not lost are stored and kept for the next turn. A new batch of particles will be added to them if the injection is not finished, otherwise they can be tracked for a few more turns to see if there are some losses.

Figure 11 Structure of the program



6. Results of the Simulation

6.1. Output and Postprocessing

The simulation output consists of several files which are first treated by a program called `Stat.exe`[1] and then the data are visualized using PAW [5].

The interesting output files from the simulation are listed below (### represents the number allocated to each run of the program, and $$ represents a number of turns).

- `part.$$.###` coordinates of the particles in phase space, after turn $$ for the run number ###.
- `bunch.###` reproduction of the data contained in the file `bunch.dat` for the run number ###, in addition with the value of the Twiss functions α_x and β_x at injection, the number of collimators, the list of the elements placed after the collimators.

1. `Stat.f` is written in XL fortran for AIX.

• Bunches.### number of particles per bunch after each turn.

• PartLos.### for each turn, list of particles lost at the septum, total number of particles lost at the septum and at the collimators along the machine.

The program Stat.exe[1] (see Appendix A.3. on page 47 for the complete listing) processes the data from Bunches.### to give the fractions of particles effectively injected relative to the total number of particles injected at each turn (the number of particles per bunch times the number of turns). This fraction is calculated for each bunch and stored in the file bnchstt.plt.The sum of all fractions gives the total efficiency of injection.The file partlos.plt contains the number of particles lost relative to the total "injected" particles. The file collost.plt gives the fraction of particles lost per turn at each location of a collimator.

The program Stat.exe processes the data from Bunches.### to give the number of particles effectively injected. This number is calculated for each bunch and stored in the file bnchstt.plt for plotting with PAW. The sum of all numbers gives the total number of particles injected.The file partlos.plt contains the number of particles lost relative to the total "injected" particles (the number of particles per bunch times the number of turns). The file collost.plt gives the fraction of particles lost per turn at each location of a collimator. The file Nbinjct.### contains the total number of particles injected at each turn. This file can then be used to compare different configurations of injection when plotting a curve for each of them as with Excel for instance.

6.2. Plots

Plots are made with the visualisation program PAW [5]. There are three different kumac scripts to plot different types of data resulting from the simulation.

• plot.kumac plots the coordinates of the particles in the horizontal transverse phase space for each turn (data from the files part.$$.###)

• distr.kumac plots the particle distribution in the transverse x and y phase planes for each turn (data from the files part.$$.###)

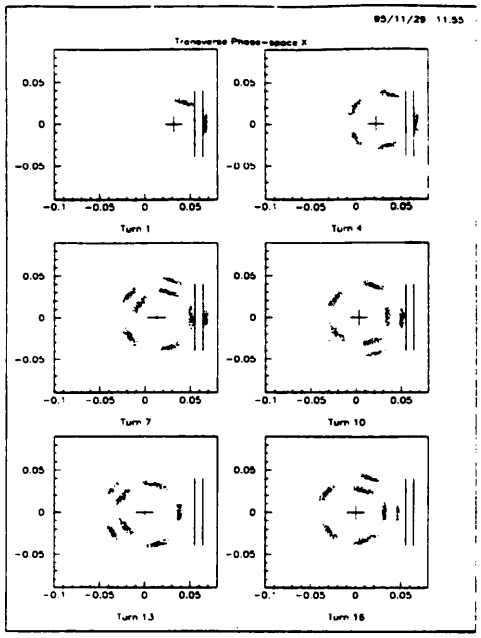
• collim.kumac plots the number of particles injected at each turn, the particle loss, and their repartition along the machine (data from the files bnchstt.plt, partlos.plt and collost.plt)

6.2.1. Plot.kumac

The paw script plot.kumac[2] plots the coordinates of the particles in the transverse phase space (Figure 12). Six turns are represented on the same page and in the same eps file. The cross symbolizes the closed orbit and enables to see the bump, the two lines on the left side indicate the position of the septum edges.

1. See A.4. on page 63 for the complete listing
2. See A.4.2. on page 65

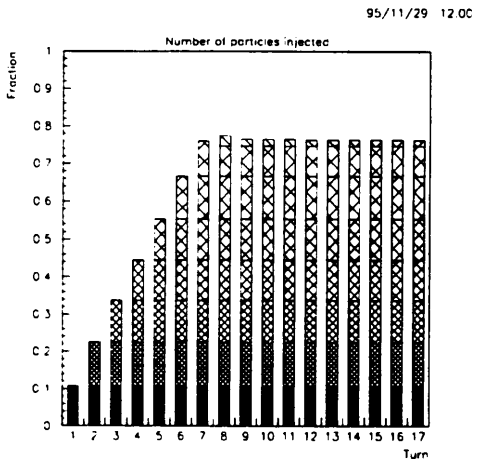
Figure 12 `transv.$.###.eps` file



6.2.2. Collim.kumac

The `collim.kumac` file[1] handles data concerning the losses of particles either at the septum or at collimators. The `effcy###.eps` plot (Figure 13) represents the number of particles injected at each turn[2]. The different parts of each histogram bar correspond to a different bunch. The first injected bunch is the darkest. This allows to see the repartition of losses amongst the bunches.

Figure 13 `effcy###.eps` file

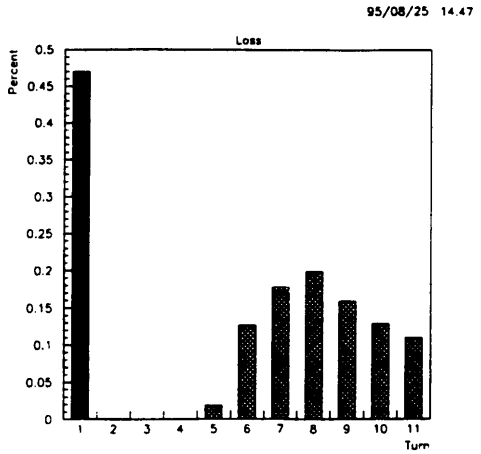


There are two possible ways to lose particles : at the septum (hitting the outer edge at injection or the inner edge after one or several turns) or at collimators along the machine. Collimators are placed where there are aperture limitations, i.e. quadrupoles. It is therefore interesting to see where the losses occur at each turn. On the plot `plost###.eps` (Figure 14) the number of particles[2] lost at the septum is represented by the darkest bar. In the

1. See Appendix A.4.1. on page 63
2. The number of particles injected or lost is normalized to the total number of particles which could be injected, i.e. the number of turns done in the run times the number of particles injected per turn.

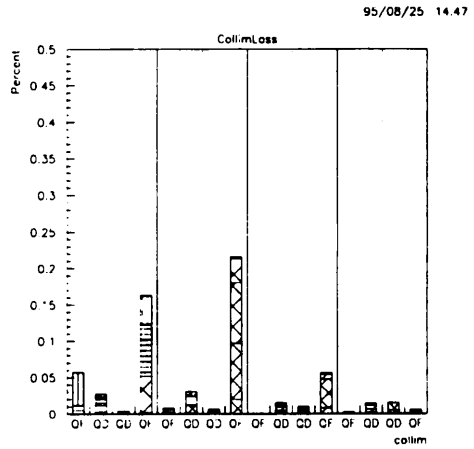
case of Figure 14, particles are lost at the septum only during the first turn, there are no losses during turns 2, 3 and 4, then particles are lost at collimators.

Figure 14 `plost###.eps` file



Plot `colim###.eps` (Figure 14) gives the position where particles are lost along the machine. The different colours used for each bar correspond to different bunches and the position given is the name of the element following the collimator in the beam line.

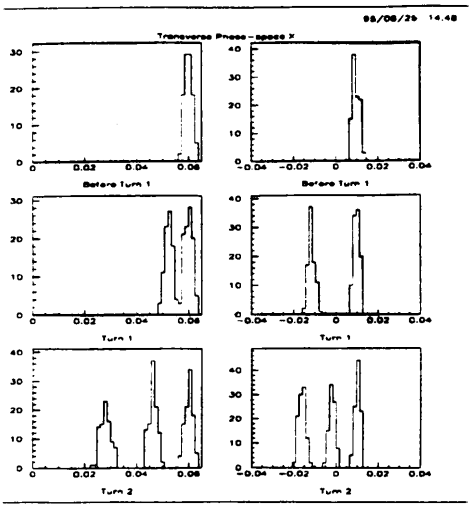
Figure 15 `colim###.eps` file



6.2.3. Distr.kumac

For each turn, the script `Distr.kumac` plots the histograms of the x and px transverse distributions of particles.

Figure 16 `distr.$.###.eps` file



7. How to use the Program

How to set-up the program for personal use will be shown, as some features depend on user ID, then how to compile it. Before a run, the input files have to be prepared with the correct parameters for the injected beam and machine. The run can then be made in batch mode, which is preferred for a few tens of turns, otherwise the program would crash. Eventually, how to deal with the program's output to produce graphics will be described.

7.1. Setting-up the Program

Some features in the program depend on the user and have to be set before compiling and running it.

The program Multinj runs MAD [6] as many times as the number of turns requested. MAD is run in batch mode by using the command `s=SYSTEMF('madbatch -q medium -p8.16 LEAR.mad')`. The program has to wait until the end of this batch to go on and process the output of MAD. For this purpose one uses the `llq` command which gives the list of batch processes on PaRC[1] [8]. This is implemented in `Finished` (page 20) in the form of a pipped command `s=SYSTEMF('llq|grep -c motsch > end.dat')` which counts the number of occurrence of the user's name (here `motsch`) in the list of batches and redirects it in the file `end.dat`. Therefore each user running the program on his personal PaRC account must place here his username.

This has a drawback : it is impossible to run several programs in batch. It is only possible to run `Multinj` in batch, then the program knows it is run in batch and takes it into account when reading the contents of the file `end.dat`.

The program was initially used for the LEAR machine so the lattice file and the MAD file names are `lear.dat` and `LEAR.mad` respectively. If one wishes to give different names to these files they also have to be changed. The name of the MAD file occurs only once in the program when running the batch command `s=SYSTEMF('madbatch -q medium -p8.16 LEAR.mad')` and the name of the lattice file occurs once in the MAD input file `LEAR.MAD(CALL, FILENAME='lear.dat')`.

To summarize :

- change username in the subroutine `Finished` (compulsory)
- change the names of the lattice file (in the MAD input file) and of the MAD input file (not compulsory)

7.2. Compiling the Program

The program is written in `fortran90`[2] and has to be compiled with the command :

1. PaRC has migrated to CERNSP in 1996.
2. XL Fortran for AIX [12]

```
xlf -q extname -L /cern/pro/lib -l kernlib -l mathlib file_name
```

This compiles and links the program to the specified libraries and the default name given to the executable is `a.out`. This command can be defined as an `alias` [8]. For instance, to create the `alias f90` corresponding to this command just type

```
alias f90 xlf -q extname -L /cern/pro/lib -l kernlib -l mathlib
```

Typing `f90 multinj.f` will do the same thing.

7.3. Setting-up the Input Files

7.3.1. Machine Lattice

The machine lattice is defined in the file `lear.dat`[1]. It is a plain MAD lattice file, so all the details concerning the syntax,... is in [6].

7.3.2. Injection Data

The injection's parameters and description of the bunch, the bump,...etc. are specified in the file `bunch.dat`. The detailed parameters are listed in section 5.2. on page 9.

7.4. Running the Program

There are two ways to run the program : the "normal" and the batch mode. It will be assumed that the program name is `mutlinj.exe`.

7.4.1. Normal Run

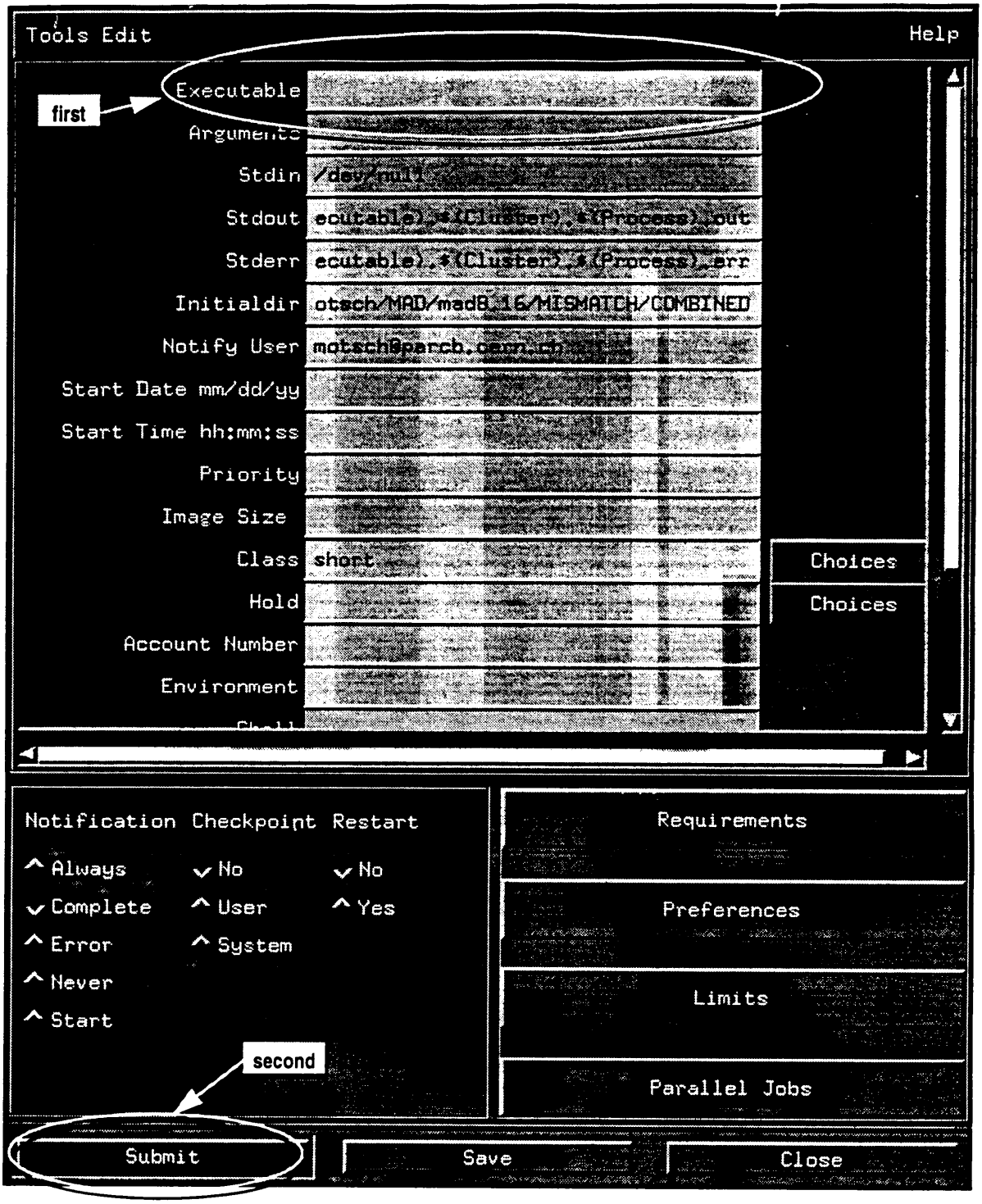
The simplest way to run the program is to type `mutlinj.exe` or `multinj.exe and` to run it in background. It is often useful to redirect the output of the program to a file to avoid getting all the messages on the screen (`multinj.exe > output &`).

7.4.2. Batch Run

This is a preferable way to run the program and almost compulsory when running it for large numbers of particles per bunch or large number of turns. In this case the run-time may overcome the limits allowed and the program would crash. The simplest way is to use the menu driven batch utility `xloadl` [8]. In the menu `File` select `Build a Job` and fill in the pop-up window (Figure 17 on page 17). Write the name of the executable in the frame `Executable` and press the button `Submit`. It is then possible to exit `xloadl` and to see the status of the run by using the `llq` command or by listing the files created.

1. See "Setting-up the Program" on page 15 if you wish to change its name.

Figure 17 xloadl window for submitting a batch



7.5. Postprocessing the Output

The postprocessing of the output of `Multinj` is made by the fortran program `Stat.exe` (see section 6.1. on page 11). This program asks for the number of the run which has to be processed and creates the files needed for plotting.

7.6. Plotting the Results

The simulation results can be visualized by using PAW. The different PAW scripts and their purpose is described in section 6.2. on page 12. To run PAW, just type paw. The program asks the workstation type which is 7879 for an xterm. Then paw opens a tektronix(Tek) window used for the drawings whilst the previous window is used to enter commands after the prompt PAW >.

The command to run a PAW script, let's call it script.kumac, is simply exec script.

8. Program Multinj

This paragraph describes all the subroutines and functions composing the program. For each of them, there is first a list of variables used as arguments and their meaning. A short description of the purpose of the procedure (subroutine or function) is then given and the manner in which it is achieved.

8.1. SUBROUTINES

Each subroutine is described in the following paragraphs. First the input or output variables are listed, then a short description of the subroutine is given. The page number refers to the listing of the program in Appendix A.2. on page 36.

8.1.1. Oneturn (page 39)

RunNb	identification number of the MAD run
nom	common suffix given to the files generated by the program
tour	number of the turn being performed
initlos	number of particles lost at the septum before the beginning of the turn
septpos	position of the inner edge of the septum
septwidth	width of the septum
ninjct	number of particles injected per turn
bunch	table containing the number of particles per bunch
bunchLoss	table containing the number of particles lost in each bunch
totLoss	total loss of particles during turn tour
betax	horizontal β function in the injection section
alphax	horizontal α function in the injection section

This subroutine analyses the results from MAD. It uses the subroutines Partlostdat (see section 8.1.2. on page 18) and Finalpos (see section 8.1.3. on page 18) to get the statistics of particles lost at the collimators or at the septum respectively.

8.1.2. Partlostdat (page 41)

RunNb	(see 8.1.1. on page 18)
nlost	number of particles lost at collimators
PartLost	table containing the list of lost particles
Sbunch	cumulated number of particles per bunch
bunchLoss	table containing the number of particles lost in each bunch

This subroutine reads the data in the print.RunNb file generated by MAD to find the particles lost at collimators along the machine. It identifies the bunch to which the particle belongs and counts the number of particles lost in each bunch as well as the total loss during the turn. The number of particles lost at each collimator is written in the file PartLos.###.

8.1.3. Finalpos (page 42)

string	string containing a line from the file print.###

SeptLoss	number of particles lost at the inner edge of the septum
PartLost	(see 8.1.2. on page 18)
Sbunch	(see 8.1.2. on page 18)
bunchLoss	(see 8.1.2. on page 18)
septpos	(see 8.1.1. on page 18)
septwidth	(see 8.1.1. on page 18)
stackposref	closed orbit position at the beginning of the turn
bumprate	value of the bump decrease per turn

This subroutine reads the coordinates of the particles remaining after the turn in the printLRunNb file. It calculates the stack position stackpos depending on the moment when the particle crosses the injection section to calculate the real position of the particles relative to the closed orbit without bump : X+stackpos. This value is compared with the position of the inner edge of the septum septpos to show whether or not the particle is lost. The table PartLost stores the number of each particle lost. The number of particles lost in each bunch is in bunchLoss and the total number of particles lost at the septum is SeptLoss.

8.1.4. Startfile (page 43)

PartLost	(see 8.1.2. on page 18)
ntot	total number of particles remaining after the turn
totLoss	(see 8.1.1. on page 18)
CoLoss	Number of particles lost at collimators
RunNb	(see 8.1.1. on page 18)

This subroutine creates the part.dat file for the new turn. The coordinates of the particles are read from the files coord.### created by MAD[6]. There are ntot-CoLoss particles in the file, which correspond to the particles which have not been automatically removed by MAD because they were lost at a collimator. The particles lost at the septum are still there because they are not processed by MAD but by the current program. The particle number is checked in the list of lost particles PartLost, if it is not lost at the septum the six coordinates X, PX, Y, PY, T, DELTAP are written in the part.dat file.

8.1.5. Newbunchfile (page 44)

turn	number of the current turn
bunch	number of particles per bunch
bunchLoss	(see 8.1.2. on page 18)

Stores the number of particles per bunch[1] after the turn in the file Bunches.###.

8.1.6. Paquet (page 45)

n	number of particles to be sampled
seed	value of the seed parameter for random sampling

This subroutine samples the bunches of particles using the subroutines Gauss1 for the momentum distribution, Gauss2 for the 2-dimension Gaussian distribution in the transverse planes x and y and the subroutine Uniform for the longitudinal distribution of the particles. If it is the first turn, the coordinates are stored in the file Part.dat, and if it is not, they are added to the file Part.dat after the coordinates of the remaining particles after the previous turn.

8.1.7. Gauss2 (page 46)

n	(see 8.1.6. on page 19)
a	horizontal dimension of the emittance ellipse

1. Bunch(i)=bunch(i)-bunchLoss(i)

b	vertical dimension of the emittance ellipse
da	horizontal translation of the centre of the ellipse
db	vertical translation of the centre of the ellipse
alpha	angle of rotation of the ellipse relative to the horizontal axis
cut	value of the standard deviation where the distribution is cut
Vect	2-dimension table containing the coordinates of the sampled particles in phase space
seed	(see 8.1.6. on page 19)

This subroutine samples a 2-dimension Gaussian distribution according to the definitions used in section 5. on page 8. The coordinates are stored in the table Vect and transferred back to the calling routine. The algorithm used is based on the Gaussian distribution on a circle (Appendix A.1. on page 35), which can be inverted.

8.1.8. Gauss1 (page 46)

DELTAP	table containing the sampled values
n	(see 8.1.6. on page 19)
Mean	mean value of the Gaussian distribution
Sigma	standard deviation of the distribution

This subroutine generates a Gaussian distribution used for the distribution of the momentum of the particles.

8.1.9. Finished (page 47)

nbfile	identification of the MAD run
BATCH	logical value, true if the program is run in batch mode

This subroutine detects the end of the MAD run. It redirects the output of the command `llq | grep -c motsch` in the file `fini.dat`. This command counts the number of occurrences of the user ID in the list of batch jobs. This has to be changed for each different user, see section 7.1. on page 15. Then according to the number read in the file and if the program is run in batch mode, the MAD run is either finished or not.

Table 3

batch	yes	no
Number in *fini.dat*	1	0
MAD finished	yes	yes

8.1.10. BeginTurn (page 48)

initlos	(see 8.1.1. on page 18)
septpos	(see 8.1.1. on page 18)
septwidth	(see 8.1.1. on page 18)
turn	(see 8.1.5. on page 19)
deltax	initial bump of the closed orbit
bumprate	(see 8.1.3. on page 18)
notlost	number of particles remaining

This subroutine checks if any particles from the file Part.dat hits the outer edge of the septum before the start of MAD. The position of the particles from the file Part.dat are relative to the closed orbit and take into account the bump, the coordinates used for the calculation are translated by the value of the bump at the moment when the particle crosses the injection section to get the real coordinates of the particles relative to the centre of the machine.

$$X + stackT \qquad \text{(Eq.19)}$$

where

$$stackT = stackpos - \frac{T \cdot bumprate}{T_{rev}} \qquad \text{(Eq.20)}$$

and *stackT* is the position of the stack at a given time T (when the particle crosses the injection section) calculated on the basis of the stack position at the beginning of the turn *stackpos*.

These coordinates are compared with the position of the outer edge of the septum $septpos + septwidth$.

8.1.11. MakeBump (page 49)

`turn` (see 8.1.5. on page 19)

`notlost` (see 8.1.10. on page 20)

This subroutine simulates the closed orbit bump. Given the injection position *injctpos* and the value of the bump at a given time T the coordinates of the particles relative to the closed orbit bump are

$$X + injctpos - bump \qquad \text{(Eq.21)}$$

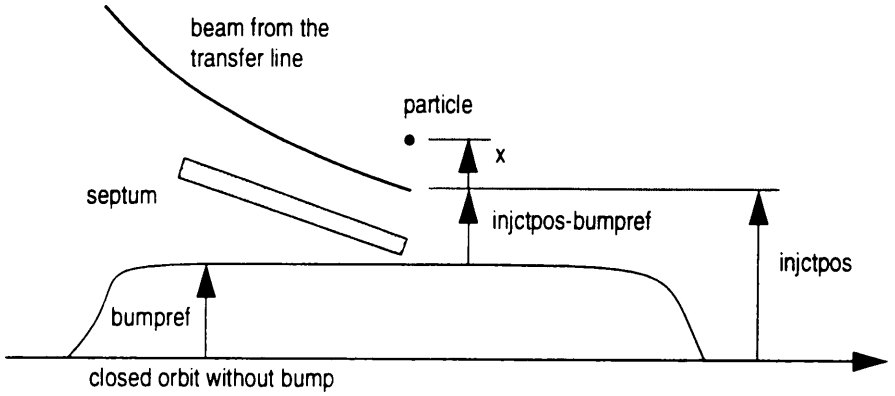
where *bump* is the bump at time T (expressed in metres in MAD)

$$bump = bumpref - \frac{T \cdot bumprate}{T_{rev}} \qquad \text{(Eq.22)}$$

and *bumpref* is the bump at the beginning of the turn

$$bumpref = \Delta x - (turn - 1) \cdot bumprate \qquad \text{(Eq.23)}$$

Figure 18 Simulation of the closed orbit bump



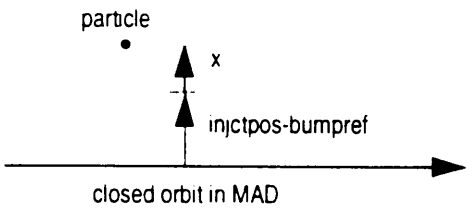
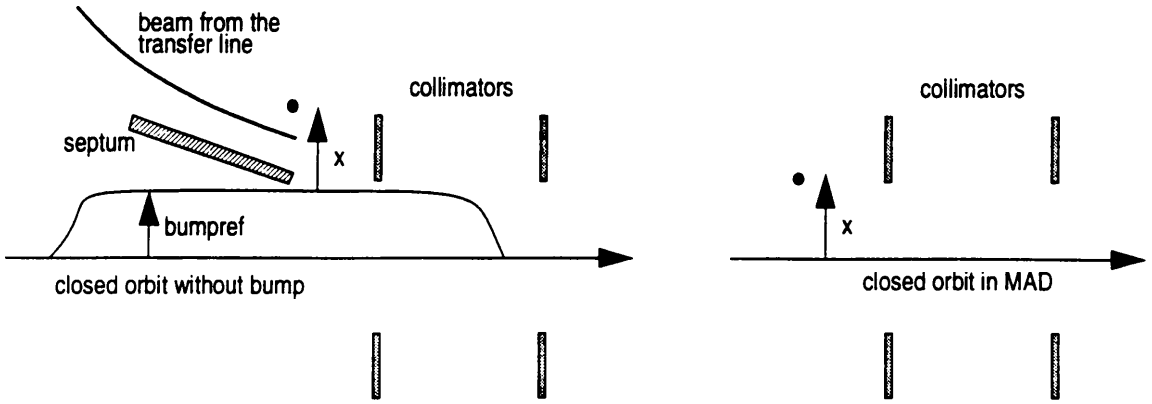


Figure 19 Collimators and bump



8.1.12. ReadData (page 50)

`n` (see 8.1.6. on page 19)

`nbturn` number of turns to inject

`ns` number of turns after injection

`Trev` revolution period in metres

This subroutine reads all the parameters specifying the bunch and the injection in the file `bunch.dat`.

8.1.13. InitBunch (page 51)

`bunch` (see 8.1.5. on page 19)

`bunchLoss` (see 8.1.2. on page 18)

This subroutine initializes tables `bunch` and `bunchLoss` containing the number of particles per bunch and the number of particles lost per bunch.

8.1.14. MakeSbunch (page 51)

`turn` (see 8.1.5. on page 19)

`bunch` (see 8.1.5. on page 19)

`Sbunch` (see 8.1.2. on page 18)

This subroutine creates the table `Sbunch`, where `Sbunch(i)` is the sum of the number of particles in the bunches $j \leq i$. This is used to identify the bunch to which a particle belongs.

8.1.15. InitPartlost (page 51)

`Partlost` (see 8.1.2. on page 18)

`bunchLoss` (see 8.1.2. on page 18)

This subroutine initializes the tables `Partlost` and `bunchLoss`.

8.1.16. StorePart (page 52)

`tour` number of the turn to add to the file name

`name` string containing the number of the run to add to the file name

`deltax` (see 8.1.10. on page 20)

`bumprate` (see 8.1.3. on page 18)

`betax` (see 8.1.1. on page 18)

`alphax` (see 8.1.1. on page 18)

`deltat` gives the fraction of particles relative to the length of the machine to be kept in the file

This subroutine stores the particle coordinates from the file `Part.dat` in a file. The coordinates of the particles relative to the closed orbit used for the calculation are converted in absolute coordinates relative to the centre of the machine by adding the value of the bump at the time the particle crosses the injection section. The coordinates can also be normalized.

8.1.17. Uniform (page 53)

`n` number of particles to sample

`T` table containing the uniformly sampled variables

`dT` upper edge of the sampled interval

`seed` (see 8.1.6. on page 19)

This subroutine uses the function `RANF` from the library to generate `n` uniform values in the interval `[0,dT]` stored in Table `T`. The value of the `seed` parameter is stored outside this routine to avoid regenerating the same distribution at each run.

8.1.18. Store (page 54)

`name` name to be given to the file where the coordinates are stored.

This subroutine stores the content of the file Part.dat in a file called `name`, that is to say, the coordinates of the particles relative to the closed orbit.

8.1.19. Store2 (page 54)

`tour` (see 8.1.16. on page 22)

`deltax` (see 8.1.10. on page 20)

`bumprate` (see 8.1.3. on page 18)

This subroutine stores the absolute coordinates of the particles relative to the centre of the machine in a file called `part2.plot`.

8.1.20. EnergyRamp (page 55)

`turn` (see 8.1.5. on page 19)

`notlost` (see 8.1.10. on page 20)

This subroutine creates the bump in momentum required for combined injection. The momentum deviation is

$$d\Delta p = \Delta p_{ref} + \frac{T \cdot \Delta p_{rate}}{T_{rev}} \qquad \text{(Eq.24)}$$

8.1.21. Collimator (page 56)

`Colim_name` table containing the name of the collimators used in the lattice

`nbcolim` number of collimator types in the whole lattice

This subroutine finds in the file `lear.dat` defining the machine the names and the number of collimators placed in the lattice. The names of the `nbcolim` collimators are stored in `Colim_name`.

8.1.22. RemoveSpace (page 56)

`name` string from which a substring delimited by spaces has to be isolated

`name_length` length of the string

This subroutine extracts the name of the collimator from `string`. The name is placed at the beginning and is surrounded by spaces that are removed.

8.1.23. ColimDat (page 57)

`Colim_pos` table containing the position of the collimators along the lattice

`colimNum` number of collimators in the lattice

`RunNb` (see 8.1.1. on page 18)

`Colimlist` list of the collimators along the lattice

`long` length of the list containing the collimator list

This subroutine reads the file `print.###` to find the position and the succession of the collimators along the machine. It uses the names of the collimators given by the subroutine `Collimator`.

8.1.24. ReplaceZero (page 58)

`string` string in which the character 0 has to be replaced by a space

This subroutine replaces the 0 characters in `string` with spaces. This enables the comparison of the number represented in `string` to be compared with another number.

8.2. FUNCTIONS

8.2.1. CHARACTER*5 FUNCTION NumberToAscii (page 47)

`n` integer to be converted into the corresponding string

This function converts the number `n` in the form of a 5 character string. If the number to convert has less than 5 digits the empty spaces are filled with 0.

8.2.2. INTEGER FUNCTION WhichCollim (page 58)

`pos` position of the collimator

`Colim_pos` table of collimator positions

`nbcolim` number of collimators in the list

This function identifies the collimator corresponding to a given position `pos` and the list of collimator positions in Table `Colim_pos`. Then it will allow to count the number of particles lost at a given position where a collimator is placed.

9. Multiturn Injection Simulation

Presented here, are the results of the simulations in two cases which might be tested. Two different lattices were used and the beam parameters have been optimized to maximize the efficiency.

9.1. Lattices used

The Twiss functions in the injection section of the machines used are listed in Table 1 on page 7. Both LEAR3 and LEAR3+ lattices are not fully symmetric : the pattern is 4(A,B). Collimators are placed to simulate aperture limitations. Their location corresponds to places where the beta function is low and the particles may be lost : centre of quadrupoles (which are split in two), entrance of the dipoles. A solenoid can be switched ON to take into account the solenoidal field in the electron cooler in section 3. The following are the lattice files defining the machines LEAR3 and LEAR3+ in MAD format.

9.1.1. Machine LEAR3

```
TITLE, S="Machine 3 for MD Pb -> LEAR"

ED1 =-0.0120348
ED2 = 0.0157712
ED3 = 0.0962689

INJ : MARKER
EC  : MARKER

DBI : DRIFT, L=0.07299
DBA : DRIFT, L=0.01575
DS  : DRIFT, L=1.059107
DL  : DRIFT, L=3.72925
DSOL: DRIFT, L=2.97925

C1 : RCOLLIMATOR,L=0.0,XSIZE=0.05,YSIZE=0.027
C2 : RCOLLIMATOR,L=0.0,XSIZE=0.05,YSIZE=0.06

BI1 : SBEND, L=2.13554, ANGLE=0.544434, E1=ED3
BA1 : SBEND, L=1.11684, ANGLE=0.240964, E1=ED2, E2=ED1
BA2 : SBEND, L=1.11684, ANGLE=0.240964, E1=ED1, E2=ED2
BI2 : SBEND, L=2.13554, ANGLE=0.544434, E2=ED3
BB  : LINE = ( DBI, C1, BI1, C1, DBA, C1, BA1, C1 )
BE  : LINE = ( C1, BA2, C1, DBA, C1, BI2, C1, DBI )

SOL : SOLENOID, L=1.5, KS=0.0

QF11 : QUADRUPOLE ,L=0.2529, K1= 0.9742
QD11 : QUADRUPOLE ,L=0.25575, K1=-1.3327
QF22 : QUADRUPOLE ,L=0.2529, K1= QF11[K1]
QD22 : QUADRUPOLE ,L=0.25575, K1=-1.1250

QF1 : LINE = ( QF11, C2 , QF11)
QD1 : LINE = ( QD11, C2 , QD11)
QF2 : LINE = ( QF22, C2 , QF22)
QD2 : LINE = ( QD22, C2 , QD22)

SF: SEXTUPOLE, L=0.33535
SD: SEXTUPOLE, L=0.33535

PER11: LINE = ( BB, DS, QD1, SF, QF1, SD, DL )
PER12: LINE = ( DL, SD, QF1, SF, QD1, DS, BE )

PER21: LINE = ( BB, DS, QD2, SF, QF2, SD, DL )
PER22: LINE = ( DL, SD, QF2, SF, QD2, DS, BE )

PER31: LINE = ( BB, DS, QD1, SF, QF1, SD, DSOL )
PER32: LINE = ( DSOL, SD, QF1, SF, QD1, DS, BE )

PER41: LINE = ( BB, DS, QD2, SF, QF2, SD, DL )
PER42: LINE = ( DL, SD, QF2, SF, QD2, DS, BE )

LEAR : LINE = ( PER12, PER21, PER22, PER31, SOL, PER32, PER41, PER42, PER11)
```

9.1.2. Machine LEAR3+

```
TITLE, S="Machine 3 for MD Pb -> LEAR"

ED1 =-0.0120348
ED2 = 0.0157712
ED3 = 0.0962689

INJ : MARKER
EC  : MARKER

DBI : DRIFT, L=0.07299
DBA : DRIFT, L=0.01575
DS  : DRIFT, L=1.059107
DL  : DRIFT, L=3.72925
DSOL : DRIFT, L=2.97925

C1 : RCOLLIMATOR,L=0.0,XSIZE=0.055,YSIZE=0.027
C2 : RCOLLIMATOR,L=0.0,XSIZE=0.055,YSIZE=0.06

BI1 : SBEND, L=2.13554, ANGLE=0.544434, E1=ED3
BA1 : SBEND, L=1.11684, ANGLE=0.240964, E1=ED2, E2=ED1
BA2 : SBEND, L=1.11684, ANGLE=0.240964, E1=ED1, E2=ED2
BI2 : SBEND, L=2.13554, ANGLE=0.544434, E2=ED3
BB  : LINE = ( DBI, C1, BI1, C1, DBA, C1, BA1, C1 )
BE  : LINE = ( C1, BA2, C1, DBA, C1, BI2, C1, DBI )

SOL : SOLENOID, L=1.5, KS=0.0

QF11 : QUADRUPOLE ,L=0.2529, K1=0.92090
QD11 : QUADRUPOLE ,L=0.25575, K1=-1.13417
QF22 : QUADRUPOLE ,L=0.2529, K1=1.08290
QD22 : QUADRUPOLE ,L=0.25575, K1=-1.38628

QF1 : LINE = ( QF11, C2 , QF11)
QD1 : LINE = ( QD11, C2 , QD11)
QF2 : LINE = ( QF22, C2 , QF22)
QD2 : LINE = ( QD22, C2 , QD22)

SF: SEXTUPOLE, L=0.33535
SD: SEXTUPOLE, L=0.33535

PER11: LINE = ( BB, DS, QD1, SF, QF1, SD, DL )
PER12: LINE = ( DL, SD, QF1, SF, QD1, DS, BE )

PER21: LINE = ( BB, DS, QD2, SF, QF2, SD, DL )
PER22: LINE = ( DL, SD, QF2, SF, QD2, DS, BE )

PER31: LINE = ( BB, DS, QD1, SF, QF1, SD, DSOL )
PER32: LINE = ( DSOL, SD, QF1, SF, QD1, DS, BE )

PER41: LINE = ( BB, DS, QD2, SF, QF2, SD, DL )
PER42: LINE = ( DL, SD, QF2, SF, QD2, DS, BE )

LEAR : LINE = ( PER12, PER21, PER22, PER31, SOL, PER32, PER41, PER42, PER11)
```

9.2. Injection Parameters

9.2.1. Beam Parameters

Table 4 gives the beam parameter used to perform the simulations in the machine LEAR3 and its modified version LEAR3+. The differences come from the Twiss functions at injection. In both cases the horizontal mismatch

Table 4 Beam parameters injected in LEAR3 and LEAR3+

LEAR	x	xp	dx	dxp	Φx	σx	y	yp	dy	dyp	Φy	σy	dT	p	Δp
3	2.2E-3	9.45E-4	0	0	0	1.64	3.6E-3	5.91E-4	0	0	0	1.64	-829.226	0	0
3+	2.29E-3	9.17E-4	0	0	0	1.64	5.11E-3	4.11E-4	0	0	0	1.64	-829.226	0	0

is 4. The vertical beam parameters are not important so far as coupling is not introduced in the lattice. For all simulations, the number of particles per batch is 200.

9.2.2. Machine Parameters

The main machine parameters such as Twiss functions in the injection section and the tunes can be found in Table 1 on page 7 and the lattice files are listed above.

9.3. Results of the Simulation

9.3.1. Machine LEAR3

The injection parameters are listed in Table 5. The initial bump Δx and the injection position injc_pos have been calculated with (Eq.3) and (Eq.4) .The bump rate results from the application of (Eq.10) for the different batches.

Table 5 Injection parameters in LEAR3

run	sept_pos (m)	sept_width (m)	Δx (m)	bumprate (m)	injct_pos (m)	Δp_ini (1)	Δp_rate (1)	nb_turn	nbmp	nsup
008	0.055	0.009	0.0382	0.00347	0.0662	0.0	0.0	7	4	5
009	0.055	0.009	0.0382	0.00347	0.0662	0.0	0.0	11	0	5
010	0.055	0.009	0.0382	0.00347	0.0662	0.0	0.0	6	5	5
011	0.055	0.009	0.0382	0.003183	0.0662	0.0	0.0	7	5	5
012	0.055	0.009	0.0382	0.003183	0.0662	0.0	0.0	8	4	5

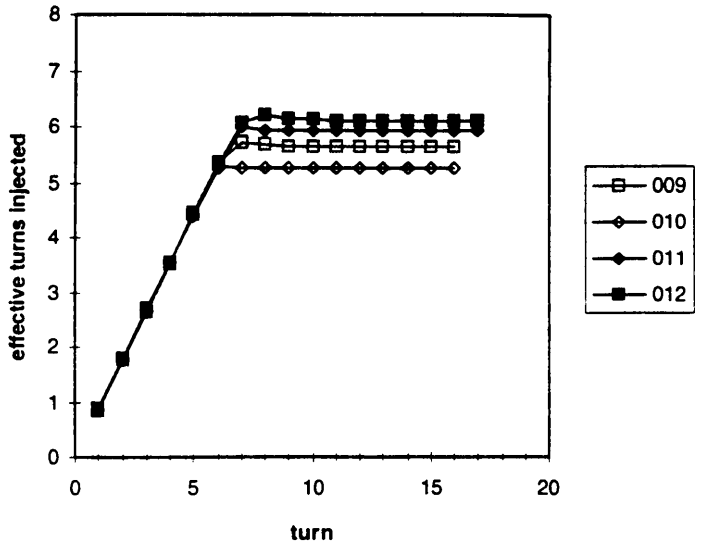
The number of turns during which the bump decreases is *nbturn + nbmp* and there are only *nbturn* during which particles are effectively injected. This number is adjusted according to the results of the simulation. In run 007, the maximum efficiency is reached at the 7^{th} turn so, at run 008, we only inject 7 turns and let the bump decrease to zero for 4 turns. In Figure 20, the evolution of the effective number of turns injected for the different situations

Table 6 Effective turns injected in LEAR3

run	008	009	010	011	012
effective turns injected	5.65	5.65	5.28	5.92	6.1

simulated can be seen. The best results can be seen in more detail at run 012 : more than 6 batches injected.

Figure 20 Injection in LEAR3 with small collimators (5.0 cm)



First of all, let us look at the transverse phase space in Figure 21 which shows the position of the closed orbit symbolized by the cross and the septum which is represented with two lines, the inner and the outer edge. It can be seen that only a little more than 7 batches are effectively injected thus it is not necessary to try to inject more than 8 batches.

Figure 21 transv012.eps

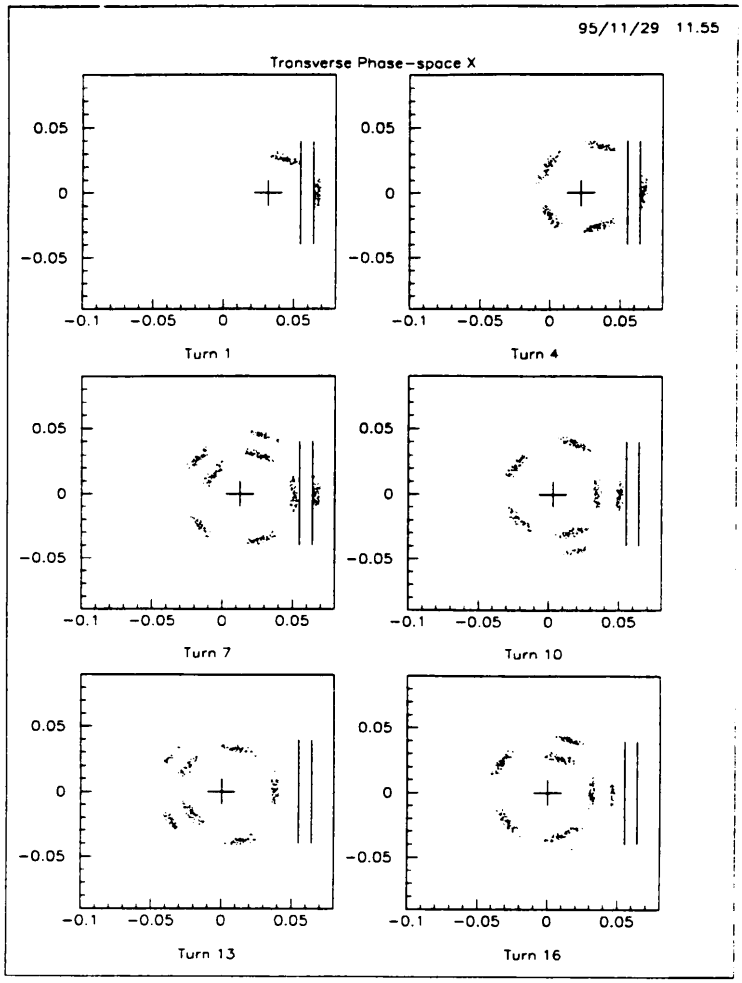


Figure 22 shows this more precisely. There are no loss of particles amongst the first 7 batches (represented with different hatches) during the 17 turns performed in the machine. Figure 23 indicates that the particles are lost at the collimators (light hatches) mainly at the 8^{th} turn. The fraction is the number of particles lost relative to the total number of particles that could be injected, i.e. the number of turns during batches are injected times the number of particles per batch.

Figure 22 effcy012.eps

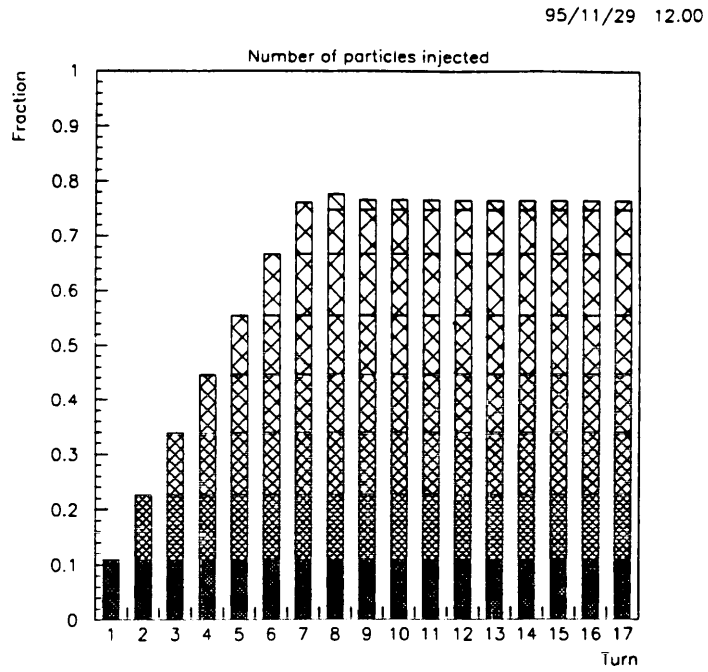
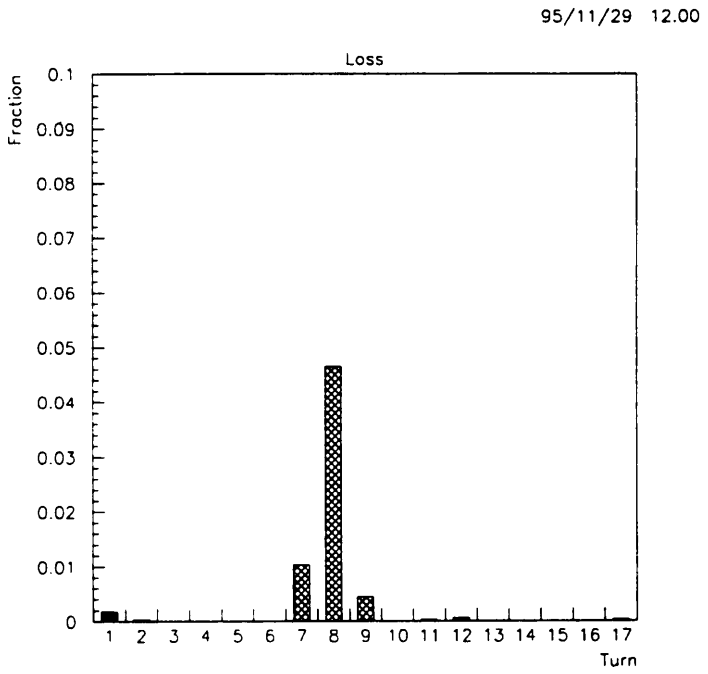


Figure 23 plost012.eps



The optimum parameters are as those for run 012. The limited number of turns injected comes from the collimators placed in the machine (horizontal aperture 0.05m).

Figure 24 shows that by increasing the size of the collimators makes it possible to inject more turns : on average

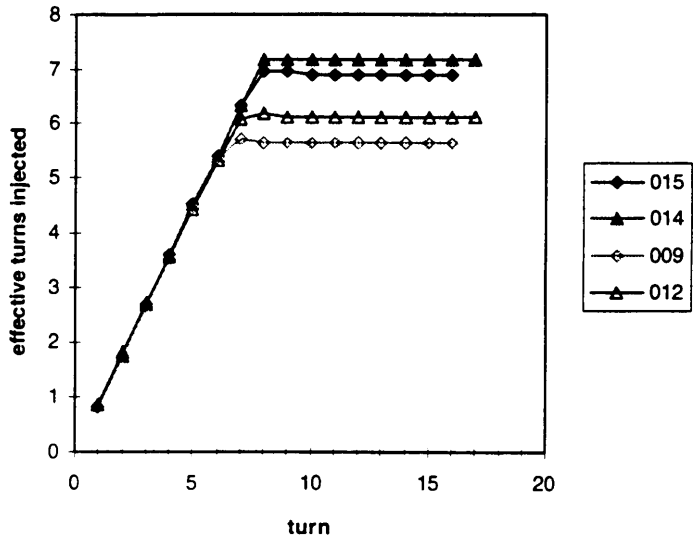
one more turn is injected with 5.5 cm and 5.0 cm collimators, with the parameters given in Table 7. The parame-

Table 7 Injection parameters in LEAR3+ (with 5.0 cm wide collimators)

run	sept_pos (m)	sept_width (m)	Δx (m)	bumprate (m)	injct_pos (m)	Δp_ini (1)	Δp_rate (1)	nb_turn	nbmp	nsup
014	0.055	0.009	0.0382	0.003183	0.0662	0.0	0.0	8	4	5
015	0.055	0.009	0.0382	0.00347	0.0662	0.0	0.0	11	0	5

ters for runs 014 and 015 are the same as for runs 012 and 009 respectively.

Figure 24 Injection in LEAR3 : influence of the collimator size (hollow markers correspond to 5.0 cm wide collimators and full markers to 5.5 cm wide collimators), the same shapes correspond to the same configurations of the injection parameters.



9.3.2. Machine LEAR3+

This machine is derived from LEAR3. The main difference is the value of the tunes. The horizontal tune has

Table 8 Injection parameters in LEAR3+ (with 5.0 cm wide collimators)

run	sept_pos (m)	sept_width (m)	Δx (m)	bumprate (m)	injct_pos (m)	Δp_ini (1)	Δp_rate (1)	nb_turn	nbmp	nsup
016	0.055	0.009	0.0327	0.00275	0.0713	0.0	0.0	12	0	5
017	0.055	0.009	0.0327	0.00275	0.0713	0.0	0.0	7	5	5
019	0.055	0.009	0.0377	0.002513	0.0663	0.0	0.0	15	0	5
020	0.055	0.009	0.0377	0.002513	0.0663	0.0	0.0	12	3	5
021	0.055	0.009	0.0377	0.002513	0.0663	0.0	0.0	12	2	5

been chosen to be more irrational than the horizontal tune of LEAR3 (Q_x = 1.8). The slight difference could allow the injection of more particles by decreasing the bump more slowly together with the fact that the operating point is further from resonance lines. The other parameters such as the α and β function in the injection section, the dispersion, are of the same order of magnitude.

The number of turns effectively injected is slightly larger than in the machine LEAR3. This difference mainly

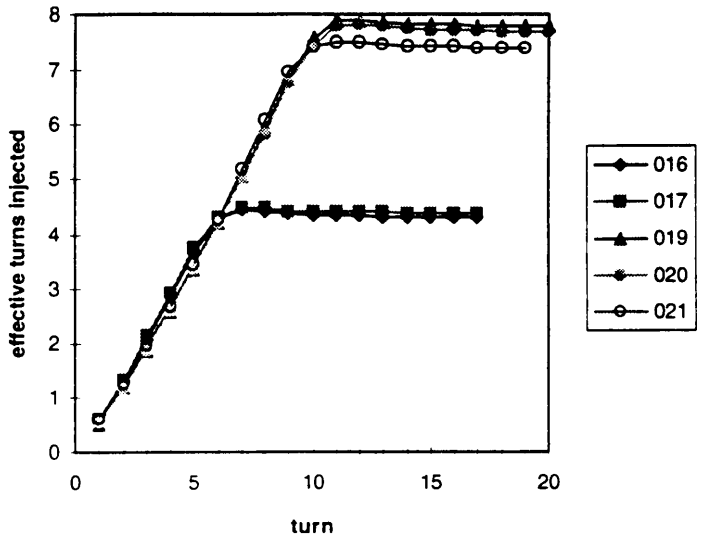
Table 9 Effective turns injected in LEAR3+ (5.5 cm collimators)

run	016	017	019	020	021
effective turns injected	4.3	4.38	7.77	7.67	7.39

comes from different sizes of the collimators used.

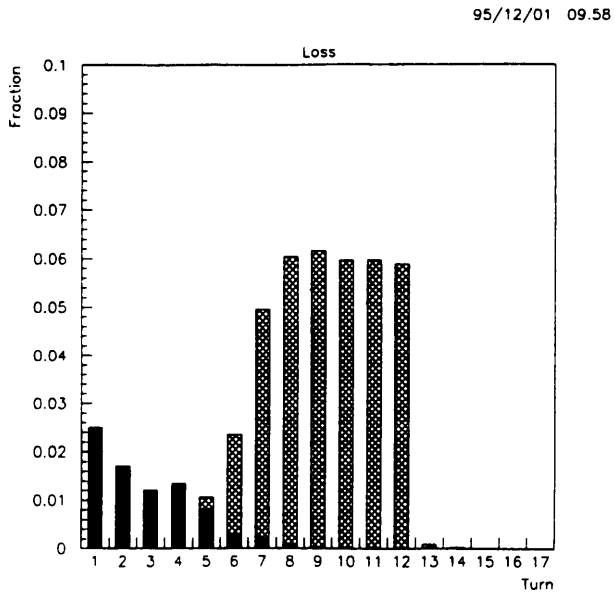
The number of turns injected is highly dependant on the value of the bump step at each turn. If it is too big, the particles oscillate far from the reference orbit after a few turns and come close to the aperture limitations.

Figure 25 Injection in LEAR3+ (collimators size : 5.5 cm)



This happens during runs 016 and 017 (Figure 25) where the number of turns is limited to 4.3 after 6 turns in the machine due to the particles hitting the collimators (Figure 26).

Figure 26 plost016.eps : particles lost during injection in LEAR3+ light hatched bars represent particles lost at the collimators and dark ones, particles lost at the septum



A smaller bump step was then used for runs 019, '020 and 021. The injection position was reduced so that the injected beam almost touched the outer edge of the septum. This lead to a tremendous increase in the number of turns injected : 3 more turns.

In order to compare the results of LEAR3+ with those of LEAR3, the size of the collimators was changed to 5.5 cm. The injection parameters are the same as for runs 016 and 019. There is a 2-turn difference between both

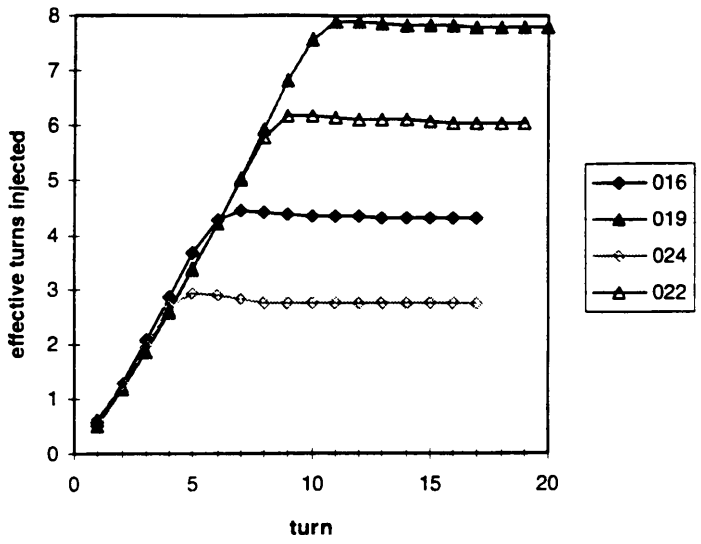
Table 10 Injection parameters in LEAR3+ (with 5.0 cm wide collimators)

run	sept_pos (m)	sept_width (m)	Δx (m)	bumprate (m)	injct_pos (m)	Δp_ini (1)	Δp_rate (1)	nb_turn	nbmp	nsup
022	0.055	0.009	0.0377	0.002513	0.0663	0.0	0.0	15	0	5
024	0.055	0.009	0.0327	0.00275	0.0713	0.0	0.0	12	0	5

cases (Figure 27). So, adjusting the bump step gives 3.5 more turns (run 024 to run 022) and then there is still a strong dependence on the collimator size which gives 2 more turns (run 022 to run 019).

The size of the collimators is a very sensitive parameter which should be carefully adjusted with experimental results to calibrate the simulations.

Figure 27 Injection in LEAR3+ : influence of the collimator size (hollow markers correspond to 5.0 cm wide collimators and full markers to 5.5 cm wide collimators, the same shape corresponds to the same configuration of the injection parameters)



10. Conclusion

The program presented allows the simulation of classical and combined multiturn injection. It gives the number of particles injected per turn and makes it possible to adjust the lattice, the injection and the beam parameters in order to optimize injection. Some features could be added to the program. For instance, it would be useful to simulate the presence of a stack of given emittance circulating in the machine and to see how it behaves during the closed orbit displacement. One would like to keep as many particles from the stack as possible and to add new ones by multiturn injection and cooling. As proposed by Christian Carli, a pre-distortion of the closed orbit by means of the bending dipoles could be used to get sufficient displacement in the injection section with the existing two bumpers. In addition, the energy of the circulating stack could be reduced to make use of dispersion and to prevent the stack from touching the inner edge of the septum during injection. All the results of the program should be calibrated with experiments to set a correct size to the collimators and to place them at the appropriate location.

Appendix

A.1. 2-dimension Gaussian Distribution

The density function of a 2-dimensional Gaussian distribution over a circle is the following

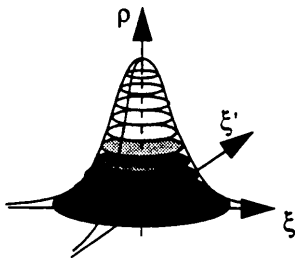
$$\rho(\xi,\xi') = N_0 \frac{1}{2\pi\sigma^2} e^{-\frac{\xi^2+\xi'^2}{2\sigma^2}} \qquad \text{(Eq.25)}$$

where ξ and ξ' are the two coordinates in the plane, σ is the standard deviation and N_0 is the total number of particles. It only depends on the distance

$$r = \sqrt{\xi^2+\xi'^2} \qquad \text{(Eq.26)}$$

between the centre and the point (ξ,ξ')

Figure 28 Gaussian distribution



Let $N(r)$ be the number of particles enclosed in a circle of radius r centred on the distribution.

$$N(r) = \int_0^{r} \int_0^{2\pi} \rho(\tilde{r})\,\tilde{r}\,d\tilde{r}\,d\varphi \qquad \text{(Eq.27)}$$

The integration over the angle φ is straightforward

$$N(r) = \int_0^{r} 2\pi\tilde{r} N_0 \frac{1}{2\pi\sigma^2} e^{-\frac{\tilde{r}^2}{2\sigma^2}} d\tilde{r} \qquad \text{(Eq.28)}$$

This is easily integrated after making the substitution

$$u = \frac{\tilde{r}^2}{2\sigma^2} \text{ and } \sigma^2 du = \tilde{r}\,d\tilde{r} \qquad \text{(Eq.29)}$$

$$N(r) = N_0 \int_0^{r^2/2\sigma^2} e^{-u} du \qquad \text{(Eq.30)}$$

Thus, one gets

$$N(r) = N_0\left[1 - e^{-\frac{r^2}{2\sigma^2}}\right] \qquad \text{(Eq.31)}$$

Different choices of r allow to cut the distribution and typical values are listed in Table 11.

Table 11

r	percentage
$\sqrt{6}\sigma = 2.49\sigma$	95%
2σ	86%
$\sqrt{2}\sigma = 1.41\sigma$	63%

(Eq.31) is easily inverted

$$r = \sqrt{2} \cdot \sigma \cdot \sqrt{ln\left(\frac{N_0}{1-N}\right)} \qquad \text{(Eq.32)}$$

and makes it possible to sample a set of Gaussian point (x, y) on the circle one needs two random variables (rnd_1, rnd_2) uniformly distributed over [0,1] and independent.

$$\begin{aligned} x &= \sqrt{2} \cdot \sigma \cdot \sqrt{ln\left(\frac{N_0}{1-rnd_1}\right)} \cdot \cos(2\pi \cdot rnd_2) \\ y &= \sqrt{2} \cdot \sigma \cdot \sqrt{ln\left(\frac{N_0}{1-rnd_1}\right)} \cdot \sin(2\pi \cdot rnd_2) \end{aligned} \qquad \text{(Eq.33)}$$

A.2. Simulation Program

```
      PROGRAM Multiturn
*
*     Simulation of multiturn injection in a circular accelerator.
*     The tracking is made by the program MAD from CERN.
*
*     version 2.1 - october 1995
*     Fabien MOTSCH (motsch@parcb.cern.ch)
*
      IMPLICIT        NONE
      INTEGER         l,nbturn,k,t,filenb,ninjct,SYSTEMF,tour,ttour,initlos,s
      INTEGER         totLoss,notlost,colimNum,length,ns,nbmp
      INTEGER         bunch(100),bunchLoss(100),Partlost(1000)
      LOGICAL         logE,BATCH
      REAL            Colim_pos(100)
      DOUBLE PRECISIONseptpos,septwidth
      DOUBLE PRECISIONx,xp,dx,dxp,phix,sigmax
      DOUBLE PRECISIONy,yp,dy,dyp,phiy,sigmay
      DOUBLE PRECISIONdT,p,sigmap,seed
      DOUBLE PRECISIONdeltax,bumprate,injctpos
      DOUBLE PRECISIONbetax,alphax
      DOUBLE PRECISIONdeltat,Trev
      DOUBLE PRECISIONdeltapini,deltaprate
      CHARACTER*3     filename,nom
      CHARACTER*5     NumberToAscii,RunNb
      CHARACTER*11    name
      CHARACTER*40    string
      CHARACTER*300   Colim_list
      COMMON /X_Plane/x,xp,dx,dxp,phix,sigmax
      COMMON /Y_Plane/y,yp,dy,dyp,phiy,sigmay
      COMMON /T_Plane/dT,deltat,p,sigmap
      COMMON /Septum/ septpos,septwidth
```

```
      COMMON /Bump/   deltax,bumprate,injctpos
      COMMON /Time/   Trev
      COMMON /Energy/ deltapini,deltaprate
      COMMON /Colim/  Colim_pos,colimNum
*
      BATCH=.FALSE.
      s=SYSTEMF('llq|grep -c motsch > end.dat')
      OPEN(UNIT=10,FILE='end.dat',STATUS='OLD')
      READ(10,*) s
      CLOSE(10,STATUS='DELETE')
      BATCH = (s.EQ.1)
      WRITE(*,*) 'batch running:',BATCH
*
      INQUIRE(FILE='seed.dat',EXIST=logE)
      IF (logE) THEN
           OPEN (UNIT=10,FILE='seed.dat',STATUS='OLD')
           READ(10,*) seed
           CLOSE(UNIT=10,STATUS='DELETE')
      ELSE
           seed = 12345.6789
      ENDIF
*
*     Finds the name to be given to the new run
*
      l = 0
      s = SYSTEMF('ls|grep FilesNb| cut -c9-11|tail -1 > lastrun')
      OPEN(UNIT=17,FILE='lastrun',STATUS='OLD')
           READ(17,*,END=600) l
 600       l = l + 1
      CLOSE(UNIT=17,STATUS='DELETE')
*
      filename   = NumberToAscii(l)(3:5)
      name(1:8)  = 'FilesNb.'
      name(9:11) = filename
*
      OPEN(UNIT=17,FILE=name,STATUS='NEW')
*
*     creation of the PartLoss file
*
      name(1:8)= 'PartLos.'
      OPEN(UNIT=11,FILE=name,STATUS='NEW')
*
*     Reads the data for sampling and simulation
*
      CALL ReadData(ninjct,nbturn,nbmp,ns,Trev)
*
*     Creation of the Bunches file for the first turn
*
      name(1:8)='Bunches.'
      OPEN(UNIT=12,FILE=name,STATUS='NEW')
*
      CALL InitBunch(bunch,bunchLoss)
*
      DO 5000 tour = 1 , nbturn + nbmp + ns
           WRITE(*,*) '==== Begining of turn:',tour,' ===='
           initlos    = 0
           ttour      = tour
           WRITE(11,'(A5,I8)') 'turn=',tour
           CALL InitPartlost(PartLost,bunchLoss)
           IF (tour.LE.nbturn) CALL Paquet(ninjct,seed)
```

```
            IF (tour.EQ.1) THEN
                 name(1:8) = 'samples.'
                 CALL Store(name)
            ENDIF
            IF (tour.LE.nbturn) THEN
                 CALL MakeBump(ttour,notlost)
                 CALL EnergyRamp(ttour,notlost)
            ENDIF
            IF (tour.LE.nbturn) THEN
                 CALL BeginTurn(initlos,septpos,septwidth,ttour,
     &           deltax,bumprate,notlost)
                 bunch(tour) = ninjct - initlos
            ELSE
                 bunch(tour) = 0
            ENDIF
*
            s=SYSTEMF('madbatch -q medium -p8.16 LEAR.mad')
*
            CALL Finished(RunNb,BATCH)
            WRITE(17,*) RunNb
            ttour = tour
            IF (tour.EQ.1) THEN
                 CALL ColimDat(Colim_pos,colimNum,RunNb,
     &                Colim_list,length)
            ENDIF
            CALL Oneturn(RunNb,name(9:11),ttour,initlos,
     &           septpos,septwidth,ninjct,bunch,bunchLoss,
     &           totLoss,betax,alphax,nbturn+nbmp)
            CALL StorePart(ttour,nbturn+nbmp,name(9:11),deltax,
     &           bumprate,betax,alphax,deltat,nbturn+nbmp)
            s    = SYSTEMF('mv Part.new Part.dat')
            notlost= notlost + ninjct - totLoss - initlos
            CALL Newbunchfile(tour,bunch,bunchLoss)
            s    = SYSTEMF('rm m*out')
            s    = SYSTEMF('rm m*err')
            s    = SYSTEMF('rm script*')
            s    = SYSTEMF('rm echo*')
            s    = SYSTEMF('rm print*')
            s    = SYSTEMF('rm coord*')
            WRITE(*,*) '==== End of turn:',tour,' ===='
 5000 CONTINUE
      CALL Newbunchfile(nbturn+nbmp+ns+1,bunch,bunchLoss)
      CALL StorePart(nbturn+nbmp+ns+1,nbturn+nbmp,name(9:11),deltax,
     &           bumprate,betax,alphax,deltat)
*
      s    = SYSTEMF('rm m*out')
      s    = SYSTEMF('rm m*err')
      s    = SYSTEMF('rm script*')
      s    = SYSTEMF('rm echo*')
      s    = SYSTEMF('rm print*')
      s    = SYSTEMF('rm coord*')
      s    = SYSTEMF('rm Part.nob')
      s    = SYSTEMF('rm Part.dat')
*
*     Creation of the bunch.### file used by the plotting program (PAW script)
*
      name(1:6) = 'bunch.'
      name(7:9) = filename
      OPEN(UNIT=14,FILE='bunch.dat',STATUS='OLD')
      OPEN(UNIT=15,FILE=name(1:9),STATUS='NEW')
```

```
 5010      READ(14,'(A40)',END=5020) string
           WRITE(15,*) string
           GOTO 5010
 5020      WRITE(15,5100) alphax,'alphax'
           WRITE(15,5100) betax,'betax'
           WRITE(15,5110) colimNum,'collimators'
           WRITE(15,*) Colim_list(1:length)
      CLOSE(15)
      CLOSE(14)
*
      CLOSE(17)
      CLOSE(11)
      CLOSE(12)
*
*     Stores the last seed for the next run
*
      OPEN (UNIT=10,FILE='seed.dat',STATUS='NEW')
      WRITE(10,*) seed
      CLOSE(UNIT=10)
*
 5100 FORMAT (F6.2,10X,A6)
 5110 FORMAT (I6,10X,A11)
*
      END
*
*
*
*
      SUBROUTINE Oneturn(RunNb,nom,tour,initlos,septpos,septwidth,
     &                   ninjct,bunch,bunchLoss,totLoss,
     &                   betax,alphax,nbt)
*
*     Analyzes the output from MAD to get the number of particles
*     lost along the machine and at the septum. Generates the
*     new 'part.dat' file with the particles that have not been
*     lost.
*
      IMPLICIT   NONE
      LOGICAL          logE,endbmp
      INTEGER          turn,num,nlost,ninjct,CoLoss,SeptLoss,count
      INTEGER          i,ntot,npart,totLoss,nbt
      INTEGER          SYSTEMF,tour,initlos,s,colimNum
      INTEGER          bunch(100),Sbunch(100),bunchLoss(100)
      INTEGER          PartLost(1000)
      REAL             Colim_pos(100)
      DOUBLE PRECISIONa(6),pos,septpos,septwidth,stackposref
      DOUBLE PRECISIONdeltax,bumprate,injctpos
      DOUBLE PRECISIONbetax,alphax
      CHARACTER*3      nom
      CHARACTER*5      RunNb
      CHARACTER*8      char
      CHARACTER*11     fname
      CHARACTER*16     place
      CHARACTER*130    string
      PARAMETER        (npart=100)
      COMMON /Bump/    deltax,bumprate,injctpos
      COMMON /Colim/   Colim_pos,colimNum
*
      WRITE(*,*) '+++ Oneturn +++'
*
```

```
      nlost      = 0
      CoLoss     = 0
      totLoss    = 0
      SeptLoss   = 0
      turn       = tour
      endbmp     = (turn.GT.nbt)
      IF (turn.LE.nbt) THEN
           stackposref= deltax - turn * bumprate
      ELSE
           stackposref= deltax - nbt * bumprate
      ENDIF
*
      fname(7:11)= RunNb
      fname(1:6) = 'print.'
*
      OPEN(UNIT=10,FILE=fname,STATUS='OLD')
*
      CALL MakeSbunch(turn,bunch,Sbunch)
*
      READ(10,'(A130)',END=110) string(1:130)
      READ(10,'(A130)',END=110) string(1:130)
      IF (string(2:25).EQ.'Linear lattice functions') THEN
           DO 105 i=1,7
                READ(10,'(A29)') string(1:29)
 105       CONTINUE
           IF ((string(26:29).EQ.'.000').AND.
 &              (string(2:11).EQ.'begin LEAR')) THEN
                BACKSPACE 10
                READ(10,'(A29,2X,2D7.3)') string(1:29),betax,alphax
           ENDIF
      ENDIF
*
 100  CONTINUE
           READ(10,'(A130)',END=110) string(1:130)
           IF (string(2:9).EQ. 'Particle') THEN
                BACKSPACE 10
                CALL Partlostdat(RunNb,nlost,PartLost,Sbunch,bunchLoss)
                CoLoss     = CoLoss + nlost
                nlost      = CoLoss + 1
                GOTO 100
           ELSEIF (string(2:6).EQ.'Final') THEN
                SeptLoss=CoLoss+1
                CALL Finalpos(string,SeptLoss,PartLost,Sbunch,bunchLoss,
 &                   septpos,septwidth,stackposref,bumprate,
 &                   endbmp)
                GOTO 110
           ELSE
                GOTO 100
           ENDIF
 110  CONTINUE
*
      IF (SeptLoss.NE.0) WRITE(11,*) 'au septum'
*
      DO 400 i = CoLoss + 1,CoLoss + SeptLoss
           WRITE(11,*) PartLost(i)
 400  CONTINUE
*
      WRITE(11,'(I6,1X,A31)') CoLoss,'Pertes totales aux collimateurs'
      WRITE(11,'(I6,1X,A24)') SeptLoss,'Pertes totales au septum'
*
```

```
      ntot       = Sbunch(turn)
      totLoss    = CoLoss + SeptLoss
      CALL STARTFILE(PartLost,ntot,totLoss,CoLoss,RunNb)
      turn       = turn + 1
*
      CLOSE(10)
      WRITE(*,*) '--- Oneturn ---'
*
      END
*
*
*
      SUBROUTINE Partlostdat(RunNb,nlost,PartLost,Sbunch,bunchLoss)
*
*     Finds the particles lost along the machine and where.
*
      IMPLICIT        NONE
      INTEGER         nlost,nblost,j,nbunch,num,i,count
      INTEGER         nbcolim,WhichCollim,label,colimNum
      INTEGER         Sbunch(100),bunchLoss(100)
      INTEGER         PartLost(1000)
      REAL            Colim_pos(100)
      CHARACTER*5     RunNb
      CHARACTER*6     Colim_name(10)
      CHARACTER*8     char
      CHARACTER*16    place
      CHARACTER*130   string
      DOUBLE PRECISIONa(6),pos
      COMMON /Colim/  Colim_pos,colimNum
*
      j          = nlost
      nblost     = 0
      count      = 0
      WRITE(*,*) '+++ Partlostdat +++'
*
      READ(10,'(A130)') string
*
      place=string(104:119)
      READ(string(66:78),'(E13.6)') pos
      label = WhichCollim(pos,Colim_pos,colimNum)
      READ(10,'(A6)') char(1:6)
*
 1000 CONTINUE
           READ(10,'(I7,4X,6(1X,D15.9))') numa(,(i),i=1,6)
           i = 1
 1100      CONTINUE
           IF (num.LE.Sbunch(i)) THEN
                bunchLoss(i) = bunchLoss(i) + 1
           ELSE
                i = i + 1
                goto 1100
           ENDIF
           nblost     = nblost + 1
           j          = j + 1
           count      = count + 1
           PartLost(j)= num

*
      READ(10,'(A130)') string
      IF (string(2:9).EQ.' ') THEN
```

```
            WRITE(11,'(I6,A23,E13.6,A18,I6)') nblost,
 &               ' particles lost at s = ',pos,' at collimator nb ',label
            GOTO 1050
      ELSEIF (string(2:9).EQ.'Particle') THEN
            WRITE(11,'(I6,A23,E13.6,A18,I6)') nblost,
 &               ' particles lost at s = ',pos,' at collimator nb ',label
            nblost= 0
            place=string(104:119)
            READ(string(66:78),'(E13.6)') pos
            label = WhichCollim(pos,Colim_pos,colimNum)
            READ(10,'(A6)') char(1:6)
            GOTO 1000
      ELSE
            BACKSPACE 10
            GOTO 1000
      ENDIF
 1050 nlost = count
*
      WRITE(*,*) '--- Partlostdat ---'
*
      END
*
*
*
      SUBROUTINE Finalpos(string,SeptLoss,PartLost,Sbunch,bunchLoss,
 &                        septpos,septwidth,stackposref,bumprate,
 &                        endbmp)
*
*     At the end of a turn, it checks if a particle is lost at the
*     injection septum.
*
      IMPLICIT          NONE
      INTEGER           num,SeptLoss,i,j,k
      INTEGER           Sbunch(100),bunchLoss(100)
      INTEGER           PartLost(1000)
      LOGICAL           Septum,hit,endbmp
      REAL              f(6)
      DOUBLE PRECISIONseptpos,septwidth,stackposref,stackpos
      DOUBLE PRECISIONbumprate,Trev,T,X
      CHARACTER*8       char
      CHARACTER*130     string
      COMMON /Time/     Trev
*
      WRITE(*,*) ' +++ Finalpos +++'
      j           = SeptLoss
      SeptLoss    = 0
*
      READ(10,'(A7)') char(1:7)
*
 2000 CONTINUE
      DO 2100 k=1,3
      IF (k.EQ.1) THEN
            READ(10,2120,END=2110) num,char(1:6),(f(i),i=1,6)
      ELSE
            READ(10,2130) (f(i),i=1,6)
      ENDIF
            IF (k.EQ.1) THEN
                  X=f(1)
            ELSEIF (k.EQ.3) THEN
                  T=f(1)
```

```
          ENDIF
 2100 CONTINUE
      IF (endbmp) THEN
           stackpos= stackposref
      ELSE
           stackpos = stackposref - (T/Trev) * bumprate
      ENDIF
      hit   = (((X + stackpos).GE.septpos))
      IF (hit) THEN
           i = 1
 2010      CONTINUE
                IF (num.LE.Sbunch(i)) THEN
                     bunchLoss(i) = bunchLoss(i) + 1
                ELSE
                     i = i + 1
                     goto 2010
                ENDIF
           SeptLoss        = SeptLoss + 1
           PartLost(j)     = num
           j               = j + 1
      ENDIF
      GOTO 2000
 2110 CONTINUE
*
      WRITE(*,*) '--- Finalpos ---'
*
 2120 FORMAT (I6,A6,2F16.8,2F14.8,F16.8,F12.8)
 2130 FORMAT (12X,2F16.8,2F14.8,F16.8,F12.8)
*
      END
*
*
*
      SUBROUTINE Startfile(PartLost,ntot,totLoss,CoLoss,RunNb)
*
*     Generates the new 'part.dat' file using the particles
*     that have not been lost during the previous turn. Then a
*     sample of newly injected particles will be added to the
*     file.
*
      IMPLICIT         NONE
      INTEGER          ntot,totLoss,CoLoss,i,j,n,partnb
      INTEGER          PartLost(1000)
      LOGICAL          lost,logE
      DOUBLE PRECISIONX,PX,Y,PY,T,DELTAP
      CHARACTER*5      RunNb
      CHARACTER*11     filename
      CHARACTER*80     line1
*
      WRITE(*,*) '+++ Startfile +++'
      filename(7:11)=RunNb
      filename(1:6)='coord.'
*
      IF (ntot-CoLoss.EQ.0) THEN
           OPEN(UNIT=14,FILE='Part.new',STATUS='NEW')
           CLOSE(UNIT=14)
      ELSE
           logE=.TRUE.
           INQUIRE(FILE=filename,EXIST=logE)
           OPEN(UNIT=13,FILE=filename,STATUS='OLD')
```

```
          OPEN(UNIT=14,FILE='Part.new',STATUS='NEW')
*
          DO 3000 i=1,ntot-CoLoss
               lost=.FALSE.
               READ(13,'(A80)') line1
               READ(line1(19:24),'(I6)') partnb
               READ(13,3110) X,PX
               READ(13,3120) Y,PY
               READ(13,3130) T,DELTAP
               DO 3100 j=1,totLoss
                    lost=lost.OR.(PartLost(j).EQ.partnb)
 3100          CONTINUE
               IF (.NOT.lost) THEN
                    WRITE(14,3110) X,PX
                    WRITE(14,3120) Y,PY
                    WRITE(14,3130) T,DELTAP
               ENDIF
 3000     CONTINUE
*
          CLOSE(13)
          CLOSE(14)
*
     ENDIF
*
     WRITE(*,*) '--- Startfile ---'
*
 3110 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
 3120 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 3130 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
*
     END
*
*
*
     SUBROUTINE Newbunchfile(turn,bunch,bunchLoss)
*
*    Stores the number of particles per bunch
*
     IMPLICIT        NONE
     INTEGER         turn,i
     INTEGER         bunch(100),bunchLoss(100)
*
     WRITE(*,*) '+++ Newbunchfile +++'
     WRITE(12,4010) 'turn=',turn-1
*
     DO 4000 i=1,turn
          WRITE(12,*) bunch(i)
*
          WRITE(*,*) bunch(i),bunchLoss(i)
*
*         WRITE(12,*) bunch(i)-bunchLoss(i)
          bunch(i)=bunch(i)-bunchLoss(i)
 4000 CONTINUE
*
     WRITE(*,*) '--- Newbunchfile ---'
*
 4010 FORMAT (A5,I8)
*
     END
*
```

```
*
*
 SUBROUTINE Paquet(n,seed)
*
*     Generates the 'n' injected particles.
*
      IMPLICIT        NONE
      INTEGER         n,c,s,i,SYSTEMF
      LOGICAL         logE
      CHARACTER*5     char
      CHARACTER*8     fname
      REAL            RANF,PI
      DOUBLE PRECISIONx,xp,dx,dxp,phix,sigmax,alphax
      DOUBLE PRECISIONy,yp,dy,dyp,phiy,sigmay,alphay
      DOUBLE PRECISIONdT,p,sigmap,seed,deltat
      DOUBLE PRECISIONHoriz(2,500),Vert(2,500)
      DOUBLE PRECISIONT(500),DELTAP(500)
      PARAMETER       (PI=3.141592653589793238)
      COMMON /X_Plane/x,xp,dx,dxp,phix,sigmax
      COMMON /Y_Plane/y,yp,dy,dyp,phiy,sigmay
      COMMON /T_Plane/dT,deltat,p,sigmap
*
      WRITE(*,*) '+++ Paquet +++'
*
      alphax     = phix*PI/180.0
      alphay     = phiy*PI/180.0
      fname      = 'Part.dat'
*
      INQUIRE(FILE=fname,EXIST=logE)
      IF (logE) THEN
           OPEN(UNIT=20,FILE=fname,STATUS='OLD')
 6000      CONTINUE
           READ(20,'(A5)',END=6100) char
           GOTO 6000
      ELSE
           OPEN(UNIT=20,FILE=fname,STATUS='NEW')
      ENDIF
*
 6100 CONTINUE
      CALL Gauss1(DELTAP,n,p,sigmap)
      WRITE(*,*) 'dxp:',dxp
      CALL Gauss2(n,x,xp,dx,dxp,alphax,sigmax,Horiz,seed)
      CALL Gauss2(n,y,yp,dy,dyp,alphay,sigmay,Vert,seed)
      CALL Uniform(n,T,dT,seed)
      DO 6101 i=1,n
           WRITE(20,6110) Horiz(1,i),Horiz(2,i)
           WRITE(20,6120) Vert(1,i),Vert(2,i)
           WRITE(20,6130) T(i),DELTAP(i)
 6101 CONTINUE
*
 CLOSE(20)
      WRITE(*,*) '--- Paquet ---'
*
 6110 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
 6120 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 6130 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
*
 END
*
*
```

```
*
 SUBROUTINE Gauss2 (n,a,b,da,db,alpha,cut,Vect,seed)
*
*     Generates a 2D gaussian distribtion over an ellipse
*     given by its half axes 'a' and 'b' which center is
*     rotated of an angle 'alpha' arround its center and
*     shifted from the origin by a vector ('da','db')
*     The gaussian distribution is cut at 'cut'.
*
      IMPLICIT        NONE
      INTEGER         count,n
      REAL            RANF,PI
      DOUBLE PRECISIONu,v,a,b,da,db,alpha,cut,tmp,seed
      DOUBLE PRECISIONVect(2,500)
      PARAMETER       (PI=3.141592653589793238)
*
      WRITE(*,*) '+++ Gauss2 +++'
      WRITE(*,*) 'db:',db
      count=0
      CALL RANSET(seed)
 6200 CONTINUE
      u     =RANF(seed)
      v     =RANF(seed)
      tmp   =u
      u     =SQRT(-2*log(u))*cos(2*PI*v)
      v     =SQRT(-2*log(tmp))*sin(2*PI*v)
*
      IF (SQRT(u**2+v**2).LE.cut) THEN
            count=count+1
            u     =a*u
            v     =b*v
            tmp   =u
            u     =u*cos(alpha)-v*sin(alpha)
            v     =tmp*sin(alpha)+v*cos(alpha)
            Vect(1,count)=u+da
            Vect(2,count)=v+db
      ENDIF
      IF (count.LE.n) GOTO 6200
      CALL RANGET(seed)
      WRITE(*,*) '--- Gauss2 ---'
*
      END
*
*
*
 SUBROUTINE Gauss1(DELTAP,n,Mean,Sigma)
*
*     Generates a 1D Gaussian distributed random set of 'n'
*     values in 'DELTAP'. The gaussian distribution is
*     characterized by 'Mean' and 'Sigma'.
*
      IMPLICIT        NONE
      INTEGER         n,i
      REAL            Vect(500)
      DOUBLE PRECISIONMean,Sigma,DBLE
      DOUBLE PRECISIONDELTAP(500)
*
      WRITE(*,*) '+++ Gauss1 +++'
*
      CALL RNORML(Vect,n)
```

```
      DO 6400 i=1,n
           DELTAP(i)=Sigma*DBLE(Vect(i))+Mean
 6400 CONTINUE
      WRITE(*,*) '--- Gauss1 ---'
*
      END
*
*
*
      SUBROUTINE Finished(nbfile,BATCH)
*
*     Tests if MAD run is finished, and gives the identification
*     number of the run 'nbfile'.
*
      IMPLICIT        NONE
      INTEGER         s,lec,stat,SYSTEMF
      CHARACTER*5     nbfile
      LOGICAL         logE,BATCH
*
      WRITE(*,*) '+++ Finished +++'
*
      lec   =16
*
      s     =SYSTEMF('llq | grep -c motsch > fini.dat')
*
 7000 CONTINUE
           OPEN (UNIT=lec,FILE='fini.dat',STATUS='OLD')
           READ (lec,'(I1)') stat
           CLOSE(UNIT=lec,STATUS='DELETE')
           s=SYSTEMF('llq | grep -c motsch > fini.dat')
      IF (((stat.NE.1).OR.(.NOT.BATCH)).AND.
     &         ((stat.NE.0).OR.BATCH)) GOTO 7000

*
      OPEN (UNIT=lec,FILE='fini.dat',STATUS='OLD')
      CLOSE(UNIT=lec,STATUS='DELETE')
*
      s=SYSTEMF('ls|grep print|cut -c7-11|tail -1 > lastfile.dat')
*
      OPEN (UNIT=lec,FILE='lastfile.dat',STATUS='OLD')
      READ (lec,'(A5)',ERR=7100) nbfile
 7100 CLOSE(UNIT=lec,STATUS='DELETE')
*
RI    WTE(*,*) '--- Finished ---'
*
      END
*
*
*
*
      CHARACTER*5 FUNCTION NumberToAscii(n)
*
*     Converts the number 'n' in the corresponding
*     string 'NumberToAscii(n)'.
*
      IMPLICIT   NONE
      INTEGER          n,i,tmp
      CHARACTER*5chaine
*
      WRITE(*,*) '+++ NumberToAscii +++'
```

```
      DO 8000 i=1,5
            tmp   =INT(n/10**(5-i))
            n     =n-(10**(5-i))*tmp
            chaine(i:i)=CHAR(tmp+48)
 8000 CONTINUE
      NumberToAscii=chaine
      WRITE(*,*) '--- NumberToAscii ---'
*
      END
*
*
*
      SUBROUTINE BeginTurn(initlos,septpos,septwidth,turn,
     &      deltax,bumprate,notlost)
*
*     Tests if some particles hit the outer part of the septum at
*     injection and removes them from the particle file.
*
      IMPLICIT              NONE
      INTEGER               initlos,s,SYSTEMF,turn,notlost,count
      LOGICAL               hit
      DOUBLE PRECISION      septpos,septwidth
      DOUBLE PRECISION      X,PX,Y,PY,T,DELTAP
      DOUBLE PRECISION      deltax,bumprate,stackpos
      DOUBLE PRECISION      Trev,stackT
      COMMON /Time/         Trev
*
      WRITE(*,*) '+++ BeginTurn +++'
*
      initlos    = 0
      count      = 0
      stackspo   = deltax-(turn-1)*bumprate
*
      OPEN(UNIT=18,FILE='Part.dat',STATUS='OLD')
      OPEN(UNIT=19,FILE='Part.new',STATUS='NEW')
*
 8000 CONTINUE
            count=count+1
            READ(18,8010,END=8100) X,PX
            READ(18,8020) Y,PY
            READ(18,8030) T,DELTAP
            stackT=stackpos-T*bumprate/Trev
*
            IF (count.GT.notlost) THEN
                  hit=((X+stackT).LE.(septpos+septwidth))
            ELSE
                  hit=.FALSE.
            ENDIF
*
            IF (hit) initlos=initlos+1
*
            IF (.NOT.hit) THEN
                  WRITE(19,8010) X,PX
                  WRITE(19,8020) Y,PY
                  WRITE(19,8030) T,DELTAP
                  ENDIF
      GOTO 8000
*
 8100 CONTINUE
*
```

```
      CLOSE(18)
      CLOSE(19)
*
      s=SYSTEMF('rm Part.dat')
      s=SYSTEMF('mv Part.new Part.dat')
*
      WRITE(*,*) '--- BeginTurn ---'
*
 8010 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
 8020 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 8030 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
*
      END
*
*
*
 SUBROUTINE MakeBump(turn,notlost)
*
*     Transforms the particles coordinates to simulate
*     a bump at turn 'turn' on the newly injected particles
*     that is to say on all particles except the 'notlost'
*     first particles.
*
      IMPLICIT          NONE
      INTEGER           turn,s,SYSTEMF,notlost,count
      DOUBLE PRECISION bump,bumprate,deltax,injctpos,bumpref
      DOUBLE PRECISION X,PX,Y,PY,T,DELTAP,Trev
      COMMON /Bump/     deltax,bumprate,injctpos
      COMMON /Time/     Trev
*
      WRITE(*,*) '+++ MakeBump +++'
*
      bumpref     = deltax-(turn-1)*bumprate
      count       = 0
*
      OPEN(UNIT=18,FILE='Part.dat',STATUS='OLD')
      OPEN(UNIT=19,FILE='Part.new',STATUS='NEW')
*
 9000 CONTINUE
           READ(18,9010,END=9100) X,PX
           READ(18,9020) Y,PY
           READ(18,9030) T,DELTAP
           count=count+1
           IF (count.GT.notlost) THEN
                bump =bumpref-T*bumprate/Trev
                X    =X+injctpos-bump
           ENDIF
           WRITE(19,9010) X,PX
           WRITE(19,9020) Y,PY
           WRITE(19,9030) T,DELTAP
      GOTO 9000
*
 9100 CONTINUE
      CLOSE(18)
      CLOSE(19)
*
      s=SYSTEMF('mv Part.dat Part.nob')
      s=SYSTEMF('mv Part.new Part.dat')
*
      WRITE(*,*) '--- MakeBump ---'
```

```
*
 9010 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
 9020 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 9030 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
*
      END
*
*
*
      SUBROUTINE ReadData(n,nbturn,nbmp,ns,Trev)
*
*     Reads the data for the simulation in the file 'bunch.dat'
*     n     number of particles injected per turn
*     nbturnnumber of turns
*
      IMPLICIT         NONE
      INTEGER          n,nbturn,nbmp,ns
      DOUBLE PRECISIONx,xp,dx,dxp,phix,sigmax
      DOUBLE PRECISIONy,yp,dy,dyp,phiy,sigmay
      DOUBLE PRECISIONdT,deltat,p,sigmap
      DOUBLE PRECISIONseptpos,septwidth
      DOUBLE PRECISIONdeltax,bumprate,injctpos
      DOUBLE PRECISIONdeltapini,deltaprate
      DOUBLE PRECISIONTrev
      COMMON /X_Plane/x,xp,dx,dxp,phix,sigmax
      COMMON /Y_Plane/y,yp,dy,dyp,phiy,sigmay
      COMMON /T_Plane/dT,deltat,p,sigmap
      COMMON /Septum/ septpos,septwidth
      COMMON /Bump/   deltax,bumprate,injctpos
      COMMON /Energy/ deltapini,deltaprate
*
      WRITE(*,*) '+++ ReadData +++'
*
      OPEN(UNIT=10,FILE='bunch.dat',STATUS='OLD')
           READ(10,*) n
           READ(10,*) nbturn
           READ(10,*) nbmp
           READ(10,*) ns
           READ(10,*) x
           READ(10,*) xp
           READ(10,*) dx
           READ(10,*) dxp
           READ(10,*) phix
           READ(10,*) sigmax
           READ(10,*) y
           READ(10,*) yp
           READ(10,*) dy
           READ(10,*) dyp
           READ(10,*) phiy
           READ(10,*) sigmay
           READ(10,*) dT
           READ(10,*) deltat
           READ(10,*) p
           READ(10,*) sigmap
           READ(10,*) septpos
           READ(10,*) septwidth
           READ(10,*) deltax
           READ(10,*) bumprate
           READ(10,*) injctpos
           READ(10,*) deltapini
```

```
          READ(10,*) deltaprate
          READ(10,*) Trev
     CLOSE(10)
*
     WRITE(*,*) '----- ReadData -----'
*
     END
*
*
*
     SUBROUTINE InitBunch(bunch,bunchLoss)
*
*    Initializes the vectors 'bunch' and 'bunchLoss'
*
     IMPLICIT   NONE
     INTEGER    nombre,i
     INTEGER    bunch(100),bunchLoss(100)
*
     WRITE(*,*) '+++ InitBunch +++'
     nombre=100
     DO 1111 i=1,nombre
          bunch(i)=0
          bunchLoss(i)=0
1111 CONTINUE
*
     WRITE(*,*) '--- InitBunch ---'
*
     END
*
*
*
     SUBROUTINE MakeSbunch(turn,bunch,Sbunch)
*
*    Creates the vector 'Sbunch' (summ of the Bunches)
*    from the bunch composition given in 'bunch'.
*    bunch(i) contains the number of particles in the
*    bunch number 'i', and Sbunch(i) the total number
*    of particles in the bunches from 1 to 'i'.
*
     IMPLICIT        NONE
     INTEGER         turn,i
     INTEGER         bunch(100),Sbunch(100)
*
     WRITE(*,*) '+++ MakeSbunch +++'
*
     DO 90 i=1,turn
          IF (i.EQ.1) THEN
               Sbunch(i)=bunch(1)
          ELSE
               Sbunch(i)=Sbunch(i-1)+bunch(i)
          ENDIF
90   CONTINUE
*
     WRITE(*,*) '--- MakeSbunch ---'
*
     END
*
*
*
     SUBROUTINE InitPartlost(Partlost,bunchLoss)
```

```
*
*     Initializes the vectors 'Partlost' and 'bunchLoss'
*     prior their use to count the lost particles...
*
      IMPLICIT        NONE
      INTEGER         i
      INTEGER         Partlost(1000),bunchLoss(100)
*
      WRITE(*,*) '+++ InitPartlost +++'
      DO 9998 i=1,1000
           Partlost(i)=0
 9998 CONTINUE
      DO 9999 i=1,100
           bunchLoss(i)=0
 9999 CONTINUE
      WRITE(*,*) '--- InitPartlost ---'
*
      END
*
*
*
      SUBROUTINE StorePart(tour,nb,name,deltax,bumprate,betax,alphax,
     &              deltat)
*
*     Stores the position of the particles given in the file
*     Part.'tour'.'name'
*     The position of the particles is shifted according to the bump
*     value at the moment when the particle passes the injection
*     section. ('deltax', 'bumrate')
*     'betax' and 'alphax' are used to normalize the positions
*     Only a slice of particles is represented: those passing
*     within the time 'deltat' after the reference particle
*
      IMPLICIT             NONE
      INTEGER              tour,nb
      REAL                 RANF
      DOUBLE PRECISION     X,PX,Y,PY,T,DELTAP,tmp,end
      DOUBLE PRECISION     deltax,bumprate,stackpos
      DOUBLE PRECISION     betax,alphax,Trev,stackT
      DOUBLE PRECISION     deltat
      CHARACTER*3          name
      CHARACTER*5          NumberToAscii
      CHARACTER*11         filename
      COMMON /Time/        Trev
*
      WRITE(*,*) '+++ StorePart +++'
*
      end             = 1.0
      filename(1:5)   ='part.'
      filename(6:7)   =NumberToAscii(tour-1)(4:5)
      filename(8:8)   ='.'
      filename(9:11)  =name
*
      OPEN(UNIT=15,FILE=filename,STATUS='NEW')
      OPEN(UNIT=16,FILE='Part.dat',STATUS='OLD')
*
      IF ((tour-1).LE.nb) THEN
           stackpos=deltax-(tour-1)*bumprate
      ELSE
           stackpos=deltax-nb*bumprate
```

```
          end   = 0.0
      ENDIF
*
 9200 CONTINUE
          READ(16,9110,END=9300)X,PX
          READ(16,9120)Y,PY
          READ(16,9130)T,DELTAP
          IF ((T.LE.0.0).AND.(T.GE.deltat*Trev)) THEN
*
*         Normalized coordinates
*
               stackT     = stackpos - end*T*bumprate/Trev
               tmp        = X
               X          = X + stackT
               PX         = alphax*tmp + PX*betax
               WRITE(15,9140) X,PX,Y,PY,T,DELTAP
          ENDIF
      GOTO 9200
 9300 CONTINUE
*
      CLOSE(16)
      CLOSE(15)
*
      WRITE(*,*) '--- StorePart ---'
*
 9110 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
 9120 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 9130 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
 9140 FORMAT (6(E19.12,1X))
*
      END
*
*
*
      SUBROUTINE Uniform(n,T,dT,seed)
*
*     Generates n uniformly distributed random variables
*     The variables are in T and belong to the interval [0,dT]
*     The value of the seed is kept for further sampling
*
      IMPLICIT             NONE
      INTEGER              n,i
      REAL                 RANF
      DOUBLE PRECISION     dT,seed
      DOUBLE PRECISION     T(500)
*
      WRITE(*,*) '+++ Uniform +++'
      CALL RANSET(seed)
*
      DO 9200 i=1,n
           T(i)=RANF(seed)*dT
 9200 CONTINUE
*
      CALL RANGET(seed)
      WRITE(*,*) '--- Uniform ---'
*
      END
*
*
*
```

```
      SUBROUTINE Store(name)
*
      IMPLICIT              NONE
      CHARACTER*11          name
      DOUBLE PRECISION      X,PX,Y,PY,T,DELTAP,tmp
*
      WRITE(*,*) '+++ Store +++'
*
      OPEN(UNIT=15,FILE='Part.dat',STATUS='OLD')
      OPEN(UNIT=16,FILE=name,STATUS='NEW')
!     OPEN(UNIT=16,FILE='Part.new',STATUS='OLD')
!     OPEN(UNIT=16,FILE='Part.dat',STATUS='OLD')
 1111 CONTINUE
           READ(15,9115,END=1112) X,PX
           READ(15,9125) Y,PY
           READ(15,9135) T,DELTAP
           WRITE(16,9145) X,PX,Y,PY,T,DELTAP
      GOTO 1111
 1112 CONTINUE
*
      CLOSE(15)
      CLOSE(16)
*
      WRITE(*,*) '--- Store ---'
*
 9115 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
 9125 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 9135 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
 9145 FORMAT (6(E19.12,1X))
*
      END
*
*
*
      SUBROUTINE Store2(tour,deltax,bumprate)
*
      IMPLICIT              NONE
      INTEGER               tour
      DOUBLE PRECISION      X,PX,Y,PY,T,DELTAP,tmp
      DOUBLE PRECISION      deltax,bumprate,stackpos
      DOUBLE PRECISION      Trev,stackT
      COMMON /Time/         Trev
*
      WRITE(*,*) '+++ Store2 +++'
      OPEN(UNIT=15,FILE='part2.plot',STATUS='NEW')
      OPEN(UNIT=16,FILE='Part.dat',STATUS='OLD')
*
      stackpos=deltax-(tour-1)*bumprate
 1212 CONTINUE
           READ(16,9117,END=1213) X,PX
           READ(16,9127) Y,PY
           READ(16,9137) T,DELTAP
           stackT =stackpos+T*bumprate/Trev
           X =X+stackT
           WRITE(15,9147) X,PX,Y,PY,T,DELTAP
      GOTO 1212
 1213 CONTINUE
*
      CLOSE(16)
      CLOSE(15)
```

```
*
      WRITE(*,*) '--- Store2 ---'
*
 9117 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
 9127 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 9137 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
 9147 FORMAT (6(E19.12,1X))
*
      END
*
*
*
      SUBROUTINE EnergyRamp(turn,notlost)
*
      IMPLICIT              NONE
      INTEGER               count,s,SYSTEMF,notlost,turn
      DOUBLE PRECISION      X,PX
      DOUBLE PRECISION      Y,PY
      DOUBLE PRECISION      T,DELTAP
      DOUBLE PRECISION      DELTAPref,dDELTAP,deltaprate,deltapini
      DOUBLE PRECISION      Trev
      COMMON /Time/         Trev
      COMMON /Energy/       deltapini,deltaprate
*
      count      =0
!     deltaprate =2.9E-4
!     deltapini  =-11.6E-4
*
      WRITE(*,*) '+++ EnergyRamp +++'
*
      DELTAPref=deltapini+(turn-1)*deltaprate
*
      OPEN(UNIT=14,FILE='Part.dat',STATUS='OLD')
      OPEN(UNIT=15,FILE='Part.new',STATUS='NEW')
*
 9400 CONTINUE
           READ(14,9410,END=9450)X,PX
           READ(14,9420) Y,PY
           READ(14,9430)T,DELTAP
           count=count+1 .
           IF (count.GT.notlost) THEN
                dDELTAP    = DELTAPref+deltaprate*T/Trev
                DELTAP     = DELTAP+dDELTAP
           ENDIF
           WRITE(15,9410)X,PX
           WRITE(15,9420)Y,PY
           WRITE(15,9430)T,DELTAP
      GOTO 9400
*
 9450 CONTINUE
*
      CLOSE(15)
      CLOSE(14,STATUS='DELETE'
*
      s=SYSTEMF('mv Part.new Part.dat'
!     s=SYSTEMF('cp Part.dat Part.glob')
*
      WRITE(*,*) '--- EnergyRamp ---'
*
 9410 FORMAT ('START, X = ',E19.12,', PX = ',E19.12,',&')
```

```
 9420 FORMAT (' Y = ',E19.12,', PY = ',E19.12,',&')
 9430 FORMAT (' T = ',E19.12,', DELTAP = ',E19.12)
*
      END
*
*
*
*
      SUBROUTINE Collimator(Colim_name,nbcolim)
*
      IMPLICIT         NONE
      INTEGER          i,nbcolim,name_length
      CHARACTER*10     name
      CHARACTER*6      Colim_name(10)
      CHARACTER*100    string
*
      WRITE(*,*) '+++ Collimator +++'
*
      nbcolim=0
 9600 CONTINUE
      OPEN(UNIT=25,FILE='lear.dat',STATUS='OLD')
      READ(25,'(A100)',END=9620) string
      i=INDEX(string,'COLLIMATOR')
      IF (i.NE.0) THEN
           nbcolim          = nbcolim+1
           name_length      = i-1
           name             = string(1:i-1)
           CALL RemoveSpace(name,name_length)
           Colim_name(nbcolim) = name(1:name_length)
      ENDIF
      GOTO 9600
 9620 CONTINUE
*
      CLOSE(25)
*
      WRITE(*,*) '--- Collimator ---'
*
      END
*
*
*
*
      SUBROUTINE RemoveSpace(name,name_length)
*
      IMPLICIT         NONE
      INTEGER          name_length
      CHARACTER*10     name
*
      WRITE(*,*) '+++ RemoveSpace +++'
 9500 CONTINUE
      IF (name(1:1).EQ.' ') THEN
           name       = name(2:
           name_length= name_length-1
      ELSE
           GOTO 9550
      ENDIF
 9550 CONTINUE
      j          = INDEX(name,' ')
      name       = name(1:j-1)
      name_length= j-1
```

```
*
      WRITE(*,*) '--- RemoveSpace ---'
*
      END
*
*
*
*
      SUBROUTINE ColimDat(Colim_pos,colimNum,RunNb,Colimlist,long)
*
      IMPLICIT              NONE
      INTEGER               nbcolim,colimNum,j,long,n,i
      INTEGER               elmtNb
      LOGICAL               lattice
      REAL                  pos
      REAL                  Colim_pos(100)
      CHARACTER*5           RunNb,Enb,NumberToAscii,tmp
      CHARACTER*6           Colim_name(10)
      CHARACTER*11          filename
      CHARACTER*30          ligne
      CHARACTER*300         Clist,Colimlist
*
      WRITE(*,*) '+++ ColimDat +++'
*
      colimNum   = 0
      n          = 1
      lattice    = .FALSE.
      filename(1:6)= 'print.'
      filename(7:11)= RunNb
*
      CALL Collimator(Colim_name,nbcolim)
      OPEN(UNIT=29,FILE=filename,STATUS='OLD')
 9700      READ(29,'(A30)',END=9720) ligne
           IF (INDEX(ligne,'Linear lattice functions').NE.0) THEN
                lattice = .TRUE.
                GOTO 9700
           ELSEIF (INDEX(ligne,'end LEAR').NE.0) THEN
                GOTO 9720
           ELSEIF (lattice) THEN
                j = 1
 9710           CONTINUE
                IF ((INDEX(ligne,Colim_name(j)).NE.0).AND.
 &                        (j.LE.nbcolim)) THEN
                     colimNum   = colimNum + 1
                     READ(ligne(20:29),'(F10.3)') pos
                     Colim_pos(colimNum)= pos
*
                     READ(ligne(2:6),'(I5)',ERR=9740) elmtNb
                     elmtNb= elmtNb + 1
                     tmp   = NumberToAscii(elmtNb)
                     CALL ReplaceZero(tmp)
 9740                CONTINUE
                     READ(29,'(A30)',END=9720) ligne
                     Enb = ligne(2:6)
                     IF (Enb.NE.tmp) GOTO 9740
*
                     Clist(n:n+2)= ligne(8:9)//' '
                     n           = n + 3
                ELSEIF (j.LT.nbcolim) THEN
                     j = j + 1
```

```
                    GOTO 9710
               ENDIF
               GOTO 9700
          ELSE
               GOTO 9700
          ENDIF
 9720 CONTINUE
      long = n - 1
      Colimlist= Clist(1:long)
 9730 CONTINUE
      CLOSE(29)
*
      WRITE(*,*) '--- ColimDat ---'
*
      END

      INTEGER FUNCTION WhichCollim(pos,Colim_pos,nbcolim)
*
      IMPLICIT             NONE
      INTEGER              nbcolim,i
      LOGICAL              found
      REAL                 Colim_pos(100),epsilon
      DOUBLE PRECISION     pos
*
       WRITE(*,*) '+++ WhichCollim +++'
*
      epsilon = 0.001
      i=1
*
 9800 CONTINUE
      IF ((ABS(REAL(pos)-Colim_pos(i)).LE.epsilon)
           & .AND.(i.LE.nbcolim)) THEN
           WhichCollim = i
           GOTO 9810
      ELSEIF (i.GE.nbcolim) THEN
           WhichCollim = 0
           GOTO 9810
      ELSE
           i = i + 1
           GOTO 9800
      ENDIF
 9810 CONTINUE
*
      WRITE(*,*) '--- WhichCollim ---'
*
      END
*
*
*
*
      SUBROUTINE ReplaceZero(string)
*
      INTEGER              index
      CHARACTER*5          string
*
      WRITE(*,*) '+++ ReplaceZero +++'
      index = 1
*
 9900 CONTINUE
          IF (string(index:index).EQ.'0') THEN
```

```
            string(index:index) = ' '
            INDEX = INDEX + 1
            GOTO 9900
         ENDIF
*
      WRITE(*,*) '--- ReplaceZero ---'
*
      END
```

A.3. Analysis Program: Statnew.f

```
      PROGRAM Stat
*
      IMPLICIT        NONE
      LOGICAL         logE
      INTEGER         ninjct,turn,i,part,nbturn,tmp,colimNum
      INTEGER         Bunch(50),Injct(50),TotInjct,nsup,nbmp
      INTEGER         cursor
      CHARACTER*3     nb
      CHARACTER*11    filename
      CHARACTER*50    string
      CHARACTER*300   transfer
*
      filename(1:8)= 'Bunches.'
      WRITE(*,*) 'File name (Bunches.###):'
      READ(*,'(A3)')filename(9:11)
      nb         = filename(9:11)
*
      OPEN(UNIT=10,FILE=filename,STATUS='OLD')
*
      filename(1:8)= 'Effinjc.'
      filename(9:11)= nb
      CALL EraseFile(filename)
      OPEN(UNIT=16,FILE=filename,STATUS='NEW')
*
      filename(1:8)= 'Nbinjct.'
      filename(9:11)= nb
      CALL EraseFile(filename)
      OPEN(UNIT=18,FILE=filename,STATUS='NEW')
*
      filename(1:8)= 'bnchstt.'
      filename(9:11)= nb
      CALL EraseFile(filename)
      OPEN(UNIT=12,FILE=filename,STATUS='NEW')
*
      filename(8:11)= '.plt'
      CALL EraseFile(filename)
      OPEN(UNIT=14,FILE=filename,STATUS='NEW')
*
      filename(1:6)= 'bunch.'
      filename(7:9)= nb
      INQUIRE(FILE=filename(1:9),EXIST=logE)
      IF (logE) THEN
         OPEN(UNIT=24,FILE=filename(1:9),STATUS='OLD')
         READ(24,*) ninjct
         WRITE(*,*) 'nbturn:',ninjct
         READ(24,*) nbturn
         WRITE(*,*) 'nbturn:',nbturn
         READ(24,*) nbmp
         WRITE(*,*) 'nbmp:',nbmp
```

```
            READ(24,*) nsup
            WRITE(*,*) 'nsup:',nsup
            CLOSE(UNIT=24)
      ELSE
            WRITE(*,*) 'Nombre de particules injectees:'
            READ(*,*) ninjct
            WRITE(*,*) 'Nombre de tours injectes'
            READ(*,*) nbturn
            WRITE(*,*) 'Nombre de tour fin bmp'
            READ(*,*) nbmp
            WRITE(*,*) 'Nombre de tours supplementaires'
            READ(*,*) nsup
      ENDIF
*
      turn = 0
*
      DO 50 i=1,50
            Bunch(i)= ninjct
 50   CONTINUE
*
      cursor = 7 * (nbturn+nbmp+nsup)
 100  CONTINUE
            READ(10,'(A5)',END=110) string(1:5)
            IF (string(1:5).EQ.'turn=') THEN
                  TotInjct = 0
                  turn = turn + 1
                  IF ((turn.LE.(nbturn+nbmp+nsup+1)).AND.(turn.GT.1)) THEN
                  DO 300 i=1,turn-1
                        READ(10,*) part
                        Injct(i) = part
                        TotInjct = TotInjct + part
                        WRITE(12,1000) turn,part,i,TotInjct
 300              CONTINUE
                  WRITE(18,1400) REAL(TotInjct)/REAL(nbturn*ninjct)
                  WRITE(16,1300) turn,TotInjct,REAL(TotInjct)/REAL(nbturn*ninjct)
*                 WRITE(14,1200) ((REAL(Injct(i))/REAL(nbturn*ninjct)),i=1,42)
                  WRITE(transfer,1200)
 &                      ((REAL(Injct(i))/REAL(nbturn*ninjct)),i=1,42)
                  WRITE(14,*) transfer(1:cursor)
                  ENDIF
            ENDIF
      GOTO 100
 110  CONTINUE
*
*     WRITE(transfer,1200) (0.00,i=1,42)
*     WRITE(14,*) transfer(1:cursor)
*     WRITE(14,*) transfer(1:cursor)
*
      CALL CollimatorNb(nb,colimNum)
      WRITE(*,*) colimNum,' collimateurs installes'
      WRITE(*,*) 'turn:',turn
      CALL LossStat(nb,colimNum,turn,ninjct)
*
      CLOSE(16)
      CLOSE(18)
      CLOSE(14)
      CLOSE(12)
      CLOSE(10)
*
 1000 FORMAT(4(I6,1X))
```

```
 1100 FORMAT(6X,10(I6,1X))
 1200 FORMAT(42(F6.4,1X))
 1300 FORMAT(2(I6,1X),F6.4)
 1400 FORMAT(F6.4)
*
      END
*
*
*
      SUBROUTINE EraseFile(filename)
*
      LOGICAL              logE
      CHARACTER*11         filename
*
      WRITE(*,*) '+++++ EraseFile +++++'
*
      INQUIRE(FILE=filename,EXIST=logE)
      IF (logE) THEN
           OPEN(UNIT=22,FILE=filename,STATUS='OLD')
           CLOSE(UNIT=22,STATUS='DELETE')
      ENDIF
*
      WRITE(*,*) '----- EraseFile -----'
*
      END
*
*
*
      SUBROUTINE CollimatorNb(nb,colimNum)
*
      IMPLICIT        NONE
      INTEGER         colimNum
      CHARACTER*3     nb
      CHARACTER*9     filename
      CHARACTER*30    string
*
      WRITE(*,*) '+++ Collimator +++'
*
      filename(1:6)= 'bunch.'
      filename(7:9)= nb
      OPEN(UNIT=24,FILE=filename,STATUS='OLD')
 3000 CONTINUE
           READ(24,'(A30)',END=3100) string
           IF (INDEX(string,'collimators').EQ.0) THEN
                GOTO 3000
           ELSE
                READ(string,'(I6)') colimNum
           ENDIF
 3100 CONTINUE
*
      WRITE(*,*) '+++ Collimator +++'
*
      CLOSE(24)
*
      END
*
*
*
      SUBROUTINE LossStat(nb,colimNum,nbturn,ninjct)
*
```

```
      IMPLICIT        NONE
      INTEGER         colimNum,nbturn,turn,pos,i,nblost
      INTEGER         tmp,ninjct,cursor
      REAL            c_lost,s_lost,tot_lost
      INTEGER         septum(70)
      CHARACTER*3     nb
      CHARACTER*11    filename
      CHARACTER*100   string
      CHARACTER*500   transfer
*
      WRITE(*,*) '+++ LossStat +++'
*
      turn       = 0
      cursor     = 7 * colimNum
*
      filename(1:8)   = 'PartLos.'
      filename(9:11)  = nb
      WRITE(*,*) 'filename:',filename
      OPEN(UNIT=24,FILE=filename,STATUS='OLD')
*
 filename(1:8)  = 'partlos.'
 filename(9:11) = 'plt'
 CALL EraseFile(filename)
 OPEN(UNIT=20,FILE=filename,STATUS='NEW')
*
      filename(1:8)= 'collost.'
      CALL EraseFile(filename)
      OPEN(UNIT=26,FILE=filename,STATUS='NEW')
*
 3500 CONTINUE
      READ(24,'(A100)',END=3520) string
      WRITE(*,*) ':',string(1:30),':'
      IF (INDEX(string,'turn').NE.0) THEN
           turn = turn + 1
           WRITE(*,*) 'turn:',turn
           IF (turn.GT.1) THEN
*               WRITE(26,3610) (REAL(septum(i))/REAL(ninjct*nbturn),i=1,70)
            WRITE(transfer,3610)
     &                        (REAL(septum(i))/REAL(nbturn*ninjct),i=1,70)
            WRITE(26,*) transfer(1:cursor)
           ENDIF
           DO 3510 i = 1,100
                septum(i) = 0
 3510      CONTINUE
           GOTO 3500
*
      ELSEIF (INDEX(string,'particles lost at').NE.0) THEN
           READ(string(1:6),'(I6)') nblost
           READ(string(61:66),'(I6)') pos
           WRITE(*,*) nblost,' particles lost at ',pos
           septum(pos) = nblost
           GOTO 3500
      ELSEIF (INDEX(string,'Pertes totales').NE.0) THEN
           READ(string(1:6),'(I6)') tmp
           c_lost     = REAL(tmp)
           READ(24,'(I6)') tmp
           s_lost     = REAL(tmp)
           tot_lost   = s_lost + c_lost
           WRITE(20,3700) s_lost/(ninjct*(nbturn-1)),
     &          c_lost/(ninjct*(nbturn-1))
```

```
            GOTO 3500
      ELSE
            GOTO 3500
      ENDIF
*
 3520 CONTINUE
*
*     WRITE(26,3610) (REAL(septum(i))/REAL(nbturn*ninjct),i=1,70)
      WRITE(transfer,3610) (REAL(septum(i))/REAL(nbturn*ninjct),i=1,70)
      WRITE(26,*) transfer(1:cursor)
*
 3530 CONTINUE
*
      CLOSE(20)
      CLOSE(24)
      CLOSE(26)
*
      WRITE(*,*) '--- LossStat ---'
*
 3610 FORMAT(70(F6.4,1X))
 3700 FORMAT(F6.4,1X,F6.4)
*

      END
```

A.4. Plotting Programs

A.4.1. collim.kumac

```
ve/delete *
option nbox
set xsiz 17
set ysiz 17
set xlab 1.7
set ygti16

filenb='001'
read filenb
mess [filenb]
*
filename='bunch.'//[filenb]
ve/cre data(1)
ve/re data [filename] ! ! /nb_turn/(*)
ve/pri data
nbturn = data(1)
ve/re data [filename] ! ! /nbmp (*)
ve/pri data
nbmp = data(1)
ve/re data [filename] ! ! /nsup (*)
ve/pri data
nsup = data(1)
ve/re data [filename] ! ! /collimators (*)
ve/pri data
nbcolim= data(1)
i=2

nbtot=[nbturn]+[nbmp]+[nsup]
*ve/cre A(24,[nbtot])
ve/cre A(42,[nbtot])
ve/cre B(2,[nbtot])
ve/cre C(70,[nbtot])
```

```
ve/cre motif(27) I 144 244 344 444 544 644 744 844 944 305 359 315 351 325 352 335
353 345 354 365 356 375 357 385 358 395 350
*
fortran/file 66 'Effcy'//[filenb]//'.eps'
graphics/meta 66 -113
*
gra/option DATE
gra/option DVXI
nbdiv=[nbturn]+[nbmp]+[nsup]+1
set NDVX '-'//[nbdiv]//'.05'
hi/cre/title_global 'Number of particles injected'
gra/hplot/null 1 [nbdiv] 0.0 1.0
gra/hplot/atitle 'Turn' 'Fraction'

*ve/re A 'bnchstt.plt' 24(1X,F6.4)
ve/re A 'bnchstt.plt' 42(1X,F6.4)
*
DO N=1,[nbdiv]-1
     set htyp motif([N])
     IF ([N].EQ.1) THEN
          ve/draw A([N]) ! SB
     ELSE
          ve/draw A([N]) ! +B
     ENDIF
ENDDO
*ve/write A ! (/,24(1X,F6.4))
ve/write A ! (/,42(1X,F6.4))
fortran/close 66
*
fortran/file 66 'Plost'//[filenb]//'.eps'
graphics/meta 66 -113
*
gra/option NOPG
gra/option DATE
gra/option DVXI
nbdiv=[nbturn]+[nbmp]+[nsup]+1
set NDVX '-'//[nbdiv]//'.05'
hi/cre/title_global 'Loss'
gra/hplot/null 1 [nbdiv] 0.0 0.5
gra/hplot/atitle 'Turn' 'Fraction'
*
ve/re B 'partlos.plt' 2(F6.4,1X)
ve/write B ! (/,2(F6.4,1X))
*
DO N=1,2
     set htyp motif([N])
     IF ([N].EQ.1) THEN
          ve/draw B([N]) ! SB
     ELSE
          ve/draw B([N]) ! +B
     ENDIF
*    ve/write B([N]) ! 2(F6.4,1X)
ENDDO
fortran/close 66

fortran/file 66 'Colim'//[filenb]//'.eps'
graphics/meta 66 -113
*
gra/option NOPG
gra/option DATE
```

```
gra/option DVXI
nbdiv=-808.01
mess nbdiv
mess [nbdiv]
mess [nbcolim]
*set NDVX '-'//[nbdiv]//'.05'
*set NDVX '-'//[nbdiv]
LABELS 1 48 QF QD BA DB BI DB BI DB BA DS QD QF QF QD BA DB BI DB BI DB BA DS QD QF
QF QD BA DB BI DB BI DB BA DS QD QF QF QD BA DB BI DB BI DB BA DS QD QF
set NDVX [nbcolim]+2.15
hi/cre/title_global 'CollimLoss'
!gra/hplot/null 1 [nbcolim]+1 0.0 0.35
gra/hplot/null 1 [nbcolim]+1 0.0 0.5
gra/hplot/atitle 'collim' 'Fraction'
*
ve/re C 'collost.plt' 70(F6.4,1X)
ve/write C ! (/,70(F6.4,1X))
*
DO N=1,[nbdiv]-1
     set htyp motif([N])
     IF ([N].EQ.1) THEN
          ve/draw C(:[nbcolim],[N]) ! SB
     ELSE
          ve/draw C(:[nbcolim],[N]) ! +B
     ENDIF
ENDDO
set BASL 0.01
DO N=1,4
     set ltyp 10
     xline=[nbcolim]*[N]/4+1
     dxline=[xline]-[nbcolim]/8
     Graphics/primitives/line [xline] 0.0 [xline] 1.0
     set ltyp 15
     Graphics/primitives/line [dxline] 0.0 [dxline] 1.0
ENDDO
fortran/close 66
```

A.4.2. plot.kumac

```
*fortran/file 66 transv_x.ps
*graphics/meta 66 -111
graphics/viewing/size 19.6 28.7
option 'date'
his/create/title_global 'Transverse Phase-space X'

his/delete *
zon 2 3
filenb='001'
read filenb
mess [filenb]
filename='bunch.'//[filenb]
mess [filename]
*
* Reads information concerning the bump

ve/cre databmp(1)

ve/re databmp [filename] ! ! /nb_turn/(*)
ve/pri databmp
nbturn=databmp(1)
ve/re databmp [filename] ! ! /deltax/(*)
```

```
ve/pri databmp
deltax=databmp(1)
ve/re databmp [filename] ! ! /bumprate/(*)
ve/pri databmp
bumprate=databmp(1)
ve/re databmp [filename] ! ! /sept_pos/(*)
ve/pri databmp
sept_pos=databmp(1)
ve/re databmp [filename] ! ! /sept_width/(*)
ve/pri databmp
sept_width=databmp(1)
ve/re databmp [filename] ! ! /alphax/(*)
mess 'alphax'
ve/pri databmp
alphax=atabmp(1)
ve/re databmp [filename] ! ! /betax/(*)
mess 'betax'
ve/pri databmp
betax=databmp(1)
mess [betax]

accept=465
radius=$SIGMA(sqrt([accept]*[betax]*1E-6))
mess radius [radius]

* Plots the bunches in the transverse X-plane
page=1
fortran/file 66 'transv.'//[page]//'.'//[filenb]//'.eps'
graphics/meta 66 -113

DO N=2,[nbturn]+1,6
*    page=INT([N]/6)
*    IF (([N]-6*[page]).EQ.0) THEN
*         fortran/file 66 'transv.'//[page]//'.'//[filenb]//'.eps'
*         graphics/meta 66 -113
*    ENDIF
     T=[N]-1.0
     bump=[deltax]-[T]*[bumprate]
     mess [bump]
     mess [N]
     nt/cre [N] 'beam' 6 ' ' 1000 x px y py t deltap
     filename='part.'
     IF ([N].LE.10) THEN
          s=[filename]//'0'//[T]//'.'//[filenb]
     ELSE
          s=[filename]//[T]//'.'//[filenb]
     ENDIF
     text='Turn '
     IF ([N].EQ.1) THEN
          title='Before '//[text]//[N]
     ELSE
          title=[text]//[T]
     ENDIF
     mess [s]
     nt/rea [N] [s]
*    h/cre/2dhisto 110 [title] 1000 -0.02 0.08 100 -0.05 0.05 100
     h/cre/2dhisto 110 [title] 1000 -0.04 0.06 100 -0.05 0.05 100
     nt/proj 110 [N].px%x
     set ndvx 510
     h/pl 110
```

```
      Graphics/primitives/line [sept_pos] -0.04 [sept_pos] 0.04
      Graphics/primitives/line [sept_pos]+[sept_width] -0.04 [sept_pos]+[sept_width]
0.04
      Graphics/primitives/line [bump] -0.01 [bump] 0.01
      Graphics/primitives/line [bump]-0.01 0.0 [bump]+0.01 0.0
*     Graphics/primitives/arc [bump] 0.0 [radius]
      h/delete 110
      IF (([N]-6*INT([N]/6)).EQ.0) THEN
           fortran/close 66
           IF ([N].LT.([nbturn]+1)) THEN
                page=[page]+1
                fortran/file 66 'transv.'//[page]//'.'//[filenb]//'.eps'
                graphics/meta 66 -113
           ENDIF
      ENDIF
ENDDO
fortran/close 66
```

References

[1] S. Maury and D. Möhl, *Combined Longitudinal and Transverse Multiturn Injection in a Heavy Ion Accumulator Ring*, CERN/PS/AR/Note 94-12, 31 May 1994, Geneva, Switzerland.

[2] P. Lefèvre and D. Möhl, *Lead Ion Accumulation Scheme for LHC*, CERN/PS/93-45 (DI), LHC Note 257, 26 October 1993, Geneva, Switzerland.

[3] The LHC Study Group, P. Lefèvre and T. Pettersson (editors), *LHC, The Large Hadron Collider, Conceptual Design*, CERN/AC/95-05 (LHC), 20 october 1995, Geneva, Switzerland.

[4] P. D. V. van der Stok, *Multiturn Injection into the CERN Proton Synchrotron Booster*, CERN/PS/BR 81-28, 22 December 1981, Geneva, Switzerland.

[5] *PAW - Physics Analysis Workstation, An Introductory Tutorial*, Geneva, Switzerland.

[6] H. Grote and F. Christoph Iselin, *The MAD Program (Methodical Accelerator Design), Version 8.13/8, User's Reference Manual*, CERN/SL/90-13 (AP)(Rev. 4), Geneva, Switzerland.

[7] Application Software and Database, *CERNLIB*, Computing and Networks Division, Geneva, Switzerland.

[8] Sebastian Masso and Franz Rohner, *PaRC User's Guide*, CERN/CN/93/8.

[9] S. Baird et al., *Beam Lifetime Tests for Pb^{52+}, Pb^{53+} and Pb^{54+} ions subject to electron cooling in LEAR (performed in June 1995)*, CERN/PS/AR/Note 95-12, 26 September 1995, Geneva, Switzerland (published in Physics Letters B 361 (1995) 184-186).

[10] K. Schindl and P. D. V. van der Stok, *Increase of Betatron Stacking Efficiency via Linear Coupling in AG Proton Synchrotrons ("Skew Injection"). Application to the CERN PS Booster*, CERN/PS/BR 76-19, CERN/PS/OP/ 76-5, 26 October 1976, Geneva, Switzerland.

[11] ed. G. Glass, *Design Study of a Facility for Experiments with Low Energy Antiprotons (LEAR)*, CERN/PS/DL 80-7, 16 May 1980.

[12] *XL Fortran for AIX, User's Guide*, Version 3 Release 2.