BNL-99372-2013-TECH
C-A/AP/222;BNL-99372-2013-IR

SIMBAD Users Manual. Version v.1.36

A. Luccio

October 2005

Collider Accelerator Department
Brookhaven National Laboratory

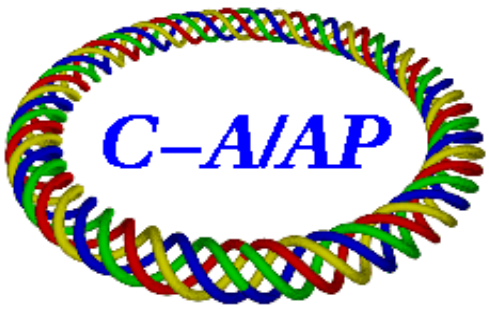
U.S. Department of Energy
USDOE Office of Science (SC)

DISCLAIMER

C-A/AP/#222
October 2005

SIMBAD User's Manual. Version v.1.36

A.U. Luccio and N.L. D'Imperio



Collider-Accelerator Department
Brookhaven National Laboratory
Upton, NY 11973

SIMBAD User's Manual. Version v.1.36

A.U. Luccio and N.L.D'Imperio
Brookhaven National Laboratory, Upton NY 11973
present address: FZ-Juelich, Germany

August 5, 2004

1 Introduction

SIMBAD is a Particle-in-Cell (PIC) tracking code [1] designed to track particles in 3-D in an accelerator. Many of its algorithms are designed and optimized with an hadron synchrotron or storage ring in mind. In *SIMBAD* a "herd" of representative macro particles (in short: macros) is injected and propagated through the structure of the accelerator represented by a sequence of transfer maps. In the present version, transfer maps are produced by the code *MAD*, that stands for Methodical Accelerator Design. The effect of space charge in the presence of accelerator walls and the effects of impedance on the beam is calculated.

SIMBAD is the BNL version of *ORBIT* [2]. It has been designed and optimized to run on a parallel computer, however for limited tasks, it can also be run on a serial machine. The language used is C++, the default compiler is the gnu compiler. The code has also been installed on machines using different compilers, like the IBM AIX. Default graphics is provided by *GNUPLOT* [3].

The code is simply run by reading parameters from a configuration file, as follows:

> SIMBAD filename.conf	serial
> mpirun -np N SIMBAD filename.conf	parallel *linux*
> lload filename.cmd	IBM AIX

The present *SIMBAD* User Manual is essentially a description of the configuration file and of the meaning of the various items in it. Before running *SIMBAD* one must edit the .conf file. A specific action is accomplished by uncomment the relative module.

2 Basic concepts

2.1 Nodes

Events happen sequentially during tracking. The propagation of the Herd through a machine element is one of such events, the calculation of Space Charge force is an event, dumping of the phase space coordinates of each macro is an event, etc. Events happen at locations in the lattice called Nodes. Each node has a name, an integer order number and a longitudinal coordinate. To start with, machine elements read from the *MAD* [4] output are given integer ordering number spaced by some input interval, say 1000, other nodes are later inserted between machine elements, just by assigning to them a number between these two numbers.

In *SIMBAD* the longitudinal distance variable is in a sense the independent variable. Sometimes the variable time is important for calculations, e.g. for space charge forces. In this case, space and time are transformed from one to the other

SIMBAD works using the Split Operator technique: some propagation in the "Bare Lattice", a calculation of some effect, a new propagation, an so on.

3 Preliminary Setting

The .conf file contains some preliminary settings. Some, marked with (*), here and in the following, should always been kept uncommented (activated)

command	*example*	*comment*
string version =	"1.36"	version of SIMBAD
int runID =	1821	run identification number
module Ring	(*)	this creates the ring ring basic structure
module SyncPart	(*)	create Synchronous particle
module MacroHerd	(*)	create MacroHerd
string herdName =	"MainHerd"	name of the herd [1]
module Parallel		initialize parallel computation
module Output		activate output module
double mSync =	1.0	mass of Sychronous particle/mass of the proton]
double charge =	1.0	charge of Synchronous particle/charge of the proton

NOTES: [1] There may be more than one herd

4 Run commands

Logical variables, here and in the folowing, may have a value [0/1 = false/true].

command	*example*	*comment*
int nTurns =	1000	number of turns to be completed
double tSync =	15.	start Kinetic Energy of Synchronous particle [GeV]
double matrixEnergy =	10.	kinetic Energy that was used to generate matrices [1]
double harmonicNumber =	1.0	harmonic number
int longTrackOnly =	[0/1]	longitudinal tracking only [2]
int usePendulum =	[0/1]	use pendulum equation for phi [3]
double rTime =	0.0	initialize ring time
int injTintegererval =	1	turn interval for injecting new particles
int nMcrsPTurn =	100000	number of macroparticles to inject per turn
int maxMacros =	100000	maximum number of macros to be injected [4]
double nRealMacros =	5.0e11	number of "real" particles
int singleNodeStep =	[0/1]	Interactive single step [5]
int nnodes =	100	number of nodes to traverse if nTurns == 0 [6]

NOTES: [1] This may be different than tSync. Defaults to tSync if not activated. It is there to initialize matrix elements scaling with energy. [2] Default value is false. [3] In this example all macros are injected at once. [4] use pendulum equation to track phi instead of matrices. default value is true. [5] Variable to specify interactive single stepping through nodes. [6] If nnodes == 0 then all nodes will be traversed.

5 Injection

5.1 External Distribution

A distribution of particles for tracking is created by a preprocessor (see Sect. 14, or generated internally. The internal generation is inherited from *ORBIT*. We tend to favor an external generation.

command	*example*	*comment*
module Injector	(*)	activate Injector module
string injectorName =	"Injector"	injector name
int injectorOrder =	2	
int injType =	1	type of population at injection [1]
string injFilename =	"HESR-240-de-4.dis"	name of file containing particle coordinates

NOTES: [0 = built in (Joho), 1 = read from file]

5.2 Built-in Distribution

This second setting if built-in distribution (Joho) is chosen. This is borrowed from *Accsim* [5].

command	*example*	*comment*
int injType =	0	type of population at injection
string injXInitializer =	"JohoXDist"	name of injection function for x
string injYInitializer =	"JohoYDist"	name of injection function for y
string injLongInitializer =	"JohoLDist"	name of injection function for longitudinal
double betaXInj =	13.588	X distribution parameters for matching [1]
double alphaXInj =	1.838	
double epsXLimInj =	100.0	
double MXJoho =	1.0	
double xTailFraction =	0.0	
double xTailFactor =	1.0	
double x0Inj =	0.0	
double xP0Inj =	0.0	
double MLJoho =	1.0	M value of Joho Logitudinal distribution
double lTailFraction =	0.0	
double lTailFactor =	1.0	
double phiLimInj =	180.0	injection bucket [-phiLimInj, phiLimInj] in deg
double dELimInj =	0.01	limiting energy spread in GeV
double deltaPhiBunch =	0.0	longitudinal center of the bunch [deg]
int nLongInjBunch =	1	
double phiMaxInj =	180.	
double phiMinInj =	-180	
double EInjMean =	0.0114	
double EInjSigma =	0.01	
double EInjMin =	0.0114	
double EInjMax =	0.0114	

NOTES: [1] for Joho epsXLimInj = epsXRMSInj * 2 * (MXJoho+1) . Same for Y.

6 Units. MAD input

SIMBAD uses MKSA units for all calculations. Input and output of phase space coordinates are in [mm] and [mrad] for transverse coordinate and angle, [rad] for longitudinal coordinate Φ and [GeV] for energy coordinate ΔE. Phase and ΔE are evaluated with respect to the phase and energy E of the synchronous particle.

MAD coordinates are in [m] and [rad] in the transverse phase space, in [m] for the phase –in *MAD* it is $\Phi : -c\Delta t$–, and [0] for the energy coordinate –in *MAD* it is $\Delta E : \Delta E/pc$– Some unit transformation is needed here and there. It is also important to note that the longitudinal coordinate, "position", of each accelerator element in *MAD* is evaluated at the end of element [6].

SIMBAD uses the output from *MAD-8* at the present time. Work is in progress to make *MAD-X* the standard.

Two *MAD* output files are used by *SIMBAD*: the ".twiss" file and an ".echo" file. The former contains twiss functions for each element, plus the value of some basic quantities for the lattice, like

the betatron tunes and the transition energy. This file is created with the command "twiss" in the input to *MAD*. The latter contains the transfer matrices for each element in the accelerator and the second order transfer maps. First order matrices are generated with the *MAD* commands "setopts, echo", then "select, flag=first". First and second order maps are generated with 'setopts, echo", then "select, flag=second".

SIMBAD reads and compares the two file to create a third file "Twiss Plus Matrices", or TPM, containing all the info necessary for bare lattice tracking. Commands and parameters are

command	*example*	*comment*
module TransMats		activate Transfer matrix module
string madFileTWISS =	"input/HESR1.twiss"	MAD Twiss file
string madFileTM =	"input/HESR1.echo"	MAD echo file
string madreadOutput =	"TPM.dat"	created Twiss+Maps
int second_order =	1	0=1.st order track, 1=2.nd order [1]

NOTES: [1] must be matched by the appropriate mad files. By default 0 is assumed.

7 Acceleration

Acceleration is simulated in *SIMBAD* by mimicking what happens in a real synchrotron control room. That is, a file containing a table of values of the main magnet field B vs. time, plus RF voltage for different harmonics and RF phase at each time. The code interpolates through the table, calculates the approriate energy for each value of the field and applies it to the synchronous particle. The RF voltage is applied to each macro at the nodes corresponding to RF cavities. The .conf file elements are as follows

command	*example*	*comment*
module RampBAccel		activate Ramp-B-Acceleration
string rampBASpecFile =	"input/Ramp_HESR1.in"	name of rampBA input file

The Ramp file contains informations for every ramp node with a line for each node. Line structure is (items separated by a blank space)

node name	node index	table	type (RAMPBV or RAMPV)	subroutine	bend radius [m]

Example of Ramp file

```
RAMPB1   15   RAMPB1.input   RAMPBV   INTERPOLATEBV   7.0
RAMPB2   30   RAMPB2.input   RAMPBV   INTERPOLATEBV   7.0
.....
```

There must be a table file for each node. A table type RAMPBV is structured as

```
number or RF harmonics: integer
time [msec]   B [T]   Volt(1) [kV]   Phase(1) [rad]   Volt(2)   Phase(2) ...
..
..
```

A table type RAMPV is structured as

```
number or RF harmonics: integer
time [msec]   Volt(1) [kV]   Phase(1) [rad]   Volt(2)   Phase(2) ...
..
..
```

8 Space Charge. Impedances

There are many methods to calculate transverse space charge self forces on a high intensity beam, all based on the solution of the Laplace-Poisson-Ampere equations. A force on a given particle P is due to the field directly generated at the particle' by every other particle Q in the beam, plus the

forces due to image charge and current on the walls of the accelerator vacum chamber. Radial forces transalte into radial angle kicks, longitudinal forces in longitudinal energy kicks.

In the present version of *SIMBAD* only the transverse Poisson problem is solved, i.e. only wall charge images are considered, and image currents are not. In addition, only perfectly conducting walls are considered. Poisson equation can be written either in integral form or in differential form. In the first case, the solution is found in *SIMBAD* using two methods: "Brute Force" or direct numerical integration, and FFT, where the Poisson equation is reduced to a convolution. In the second case, the differential Poisson equation is solved by writing the Laplacian on a transverse mesh and inverting it using standard linear equation solvers, like LU decomposition, or iterative methods.

After a solution is found of the transverse problem, the longitudinal kick is calculated by the potential difference at any transverse x, y location between adiacent slices.

Finite conductivity of the walls and variation of the vall geometry along the beam path can be represented by longitudinal and transverse impedance tables, with real an imaginary part of the impedance listed for each harmonic mode (frequency). Longitudinal and transverse kicks are calculated in *SIMBAD*, following a method developed for the one-dimensional FermiLab code *ESME* [7] by first performing an FFT analysis of the beam current, in order to find harmonic components of the current. Then, each component of the impedance, in [ohm], is combined with a component of the current, in [A], to generate a voltage kick, with the appropriate amplitude, in [volt], and phase. These kicks produce a distorsion in the phase space distribution that can lead to instabilities and beam losses. It must be pointed out that impedance effects develop very slowly, and then require that the herd would be followed for many thousand revolutions in the simulation. Also, FFT of the beam can be made up to a large number of frequency with the needed accuracy only if the number of macros used in the simulation is large. For impedance studies parallel computing is a must. At the present time impedance effects are only calculated once per turn, using impedance budget tables for the full accelerator structure.

8.1 1-, 2-, and 3-dimension tracking

One dimension tracking is longitudinal tracking where we don't care about the transverse phase space that is only transformed along the lattice using the bare tune *MAD* maps. Longitudinal transformations are done by solving the coupled discretized "pendulum equations" for Φ and $\Delta p/p$.

Two dimension tracking, that is sometimes called $2\frac{1}{2}$, is performed by slicing the beam in many longitudinal partitions, and solving the space charge problem (see sec. /refsec:SpaceCharge) in each section. At each Transverse Space Charge node this operation is performed on each longitudinal slice, possibly with parallel computing. In this case, the longitudinal propagation is done by using the full 6×6 1.st order matrices and the $6 \times 6 \times 6$ second order transfer maps, or tensors.

Three dimension tracking is done by still slicing the beam, but solving the space charge problem with all macros in the herd reduced at the same time. We call this "freezing" the beam. Only in this case the space charge forces in all three dimensions are correctly calculated. The criterion for slicing the beam is that a slice should be a fraction of a β-function wave.

8.2 Longitudinal Space Charge. Longitudinal Impedances

Activate Longitudinal Space charge module based on FFT (don't use with freeze). This module is both calculating L space charge with a theoretical formula and Long impedances [8], if the impedance file is present. If this module is on and Transverse Space Charge is also on, it must be int nLongSC = 0 in TSC, otherwise *SIMBAD* would try to calculate the LSC twice with different methods

command	*example*	*comment*
module FFTLongSC		activate Longit. Space charge module
string fftLSCName =	"LSC1"	name of node
int fftLSCOrder =	770	node index
int nLongBins =	128	number of longitudinal bins
double b_a =	5.0	ratio between beam (round) and chamber b/a
int useAvg =	[0/1]	use averages
int nMacroLSCMin =	10	minimum number of Macros [1]
string fftLongSCSpecFile =	"input/ztab128.tab"	file of long. impedance data

NOTES: [1] for LSC and TSC calcs to be done.

The file containing the longitudinal impedance data has format

n0	real(Z)	imag(Z)
n1	real(Z)	imag(Z)
..		

At the present only one table of L Imped is present, as a budget for the entire ring.

8.3 Transverse Space Charge

command	*example*	*comment*
module TSpaceCharge		activate Transverse Space charge module
int nXBins =	32	number of mesh points in x [1]
int nYBins =	32	number of mesh points in y
double hDimension =	160	wall dimensions in mm [2]
double vDimension =	160	
string wallsFile =	'input/walls.in"	input table for variable walls option [3]
int actPartLosses =	[0/1]	activate particle losses for collisions with walls
int tscCalcType =	[1/2/3/4]	method of calculating Transverse Space Charge [4]
double pOpt =	0.999922	Jacobi spectral radius for SOR: optimal for 512x512
double pOpt =	0.999912	- optimal for 256x256
double pOpt =	0.9997	- optimal for 128x128
double pOpt =	0.9986	- optimal for 64x64
double pOpt =	0.99469	- optimal for 32x32
double eps =	0.001	smoothing parameter
double gridFactor =	0.001	grid factor for extra spacing around distribution
int useMPISOR =	[0/1]	variable to enable parallel SOR solve [5]
double nMacsMinPercent =	0.01	% of macros for which TSC calcs will not be done [6]

NOTES: [1] the number of mesh points must be even. [2] At the present, need to be same value for H and V. If different, the largest is the default. These number can be overwritten by the variable wall option. Walls are perfectly conducting. [3] In a wall file each entry denotes a new section of wall [mm] vs. location around the machine [m]. The format is as follows: horiz[mm] vert[mm] sPos[m] [4] "1" is Sparse LU method, "2" is FFT, "3" is for SOR iterative method, "4" is BF. The grid must be square for all methods and the dimension of the chamber must be square for methods 1 and 3. [5] This is only good with large grids. Make sure you test before committing to a production run. [6] For example, if the value is 1.0 then if the number of macros in a given element is less than one percent of the global number of macros the calcs will be skipped.

8.4 3-D

command	*example*	*comment*
int nLongSC =	0	no of times long. space charge is calculated per turn [1]
int longBins =	128	number of bins to subdivide the beam for LSC calcs
int bmEnvLayout =	[1/0]	layout spacecharge nodes according to beam envelope
int betaFuncFact =	4	variable to divide the space charge elements - according to the length of the beta function
int freezeBeam =	[1/0]	variable to enable beam freezing [2]
int trimPhi =	[0/1]	Trim particles with extreme phi values

NOTES: [1] the number of mesh points must be even. [2] At present only applies to iterative solver and only to serial runs.

8.5 Transverse Impedances

Transverse Impedances require an FFT of the beam transversely, to cople impedances with transverse beam modes

command	*example*	*comment*
module FFT_TImpedance		activate module
string fftTImpedName =	"TIMPED1"	name of node
int fftTImpedOrder =	780	node index
int nTImpedBins =	32	number of transverse bins
double b_a_TImped =	1.0	ratio of beam height to width
int useAvg_TImped =	[0/1]	use transverse averages
int nMacsMinTImped =	[0/1]	minumum number of Macros for calcs
string fftTimpedSpecFile =	"input/FFTTImped.input"	T impedances table

The FFT_TImpedance input file contains the information for the impedance values of the node. The file should contain (items separated by a blank)

(nTImpedBins) lines with 9 columns per line:
nfrequency number
real(ZXImped_nplus)
imag(ZXImped_nplus)
real(ZXImped_minus)
imag(ZXImped_minus)
real(ZYImped_nplus)
imag(ZYImped_nplus)
real(ZYImped_minus)
imag(ZYImped_minus)

9 Special Machine Elements

SIMBAD allows the insertion in the lattice of elements that have a special purpose, like foils, thin lenses or collimators

9.1 Foil

In some proton machines negative ions are injected and stripped to protons on a thin foil. Foils can also be placed in the lacttice for special purposes. The scattering and propagation through a foil is calculated. Hits to the foils are counted.

command	*example*	*comment*
module Foil		activate Foil module
string foilName =	"Foil"	name of the foil
int foilOrder =	2	node order of the foil
int useFoilScat =	[0/1]	flag to use foil scattering
double xfmin =	-100.	min. foil coordinate in x[mm]
double xfmax =	-25.	maximum foil in x [2]
double yfmin =	-100.0	miniumum foil in y
double yfmax =	100.0	maximum foil in y
double foilThickness =	300.	thickness of the foil [$\mu gram/mm^2$]
double foilFac =	0.665	scale edge so that 1% miss foil[1]

NOTES: [1] Foil factor = 1.33/((Injection::MXJoho + 1)/2). [2] inj− > x0Inj + sqrt(fabs(foilFac * inj-¿betaXInj * inj− >epsXLimInj))

9.2 Bump

Programmable orbit bump. Three types of bump. Most likely: injection bumps are simulated in this module. If a bump table is used

command	*example*	*comment*
module IdealBump1		activate Ideal Bump type 1
string bump1Name =	"UP-BUMP"	Bump 1 name
int bump1Order =	1	Bump 1 node index
module IdealBump2		activate Ideal Bump type 2
string bump2Name =	"DOWN-BUMP"	Bump 2 name
int bump2Order =	4	Bump 2 node indx
int bump1UD =	[0/1]	Up/Down bump1, 0=down, 1=up
string bumpFileName =	"input/bump.tab"	file from which to read bump.ramp
double tBmp0 =	0.0	initial Bump time
double tBmpF =	1.	final Bump time
double eFTX =	8	e-fold time ratio (after this, the bump is gone)
double eFTY =	0.00	e-fold time

If a bump table is NOT used, then

command	*example*	*comment*
string bumpFunct =	"EFoldBump"	bump function to be used
double xBmp0 =	-30.	initial value of the x-bump
double xBmpF =	0.	final value of the x-bump
double xPBmp0 =	-5.	initial value of the xP-bump
double yBmp0 =	0.0	initial value of the y-bump
double yBmpF =	0.0	final value of the y-bump
double yPBmp0 =	0.0	initial value of the yP-bump

9.3 Integrable Lens 2D

command	*example*	*comment*
module IntegLens2D		activate Integrable 2D Lens
string intLens2DSpecFile =	"input/IntLens2D.in"	name of integrable 2D Lens input file

The IntegLens2D input file contains the information for each node. The format of the file must be as follows (a blank between items)

name of the node	name of output file	node order number	:*first line*
non linear coefficient	linear coefficient		:*second line*

skip a line and repeat for the next IntegLens node

Example of IntegLens file

```
IL1    15
5.0    2.0

IL2    25
7.0    1.0
```

9.4 Rectangular Aperture

command	*example*	*comment*
module RectAperture		activate the module
string rectApFileName =	"input/RectAp.in"	name of input file

The RectAperture input file contains the information for each node. The format of the file must be as follows (a blank between items)

node name	node index				:*first line*
xmin	xmax	ymin	ymax	transparent[0/1]	:*second line*

skip a line and repeat for the next RectAperture node

Example of Rectangular Aperture file:

```
RectAp1        5
10.0           25.0    5.0    50.0    0

RectAp2 ....
....
```

9.5 Momentum Aperture

command	*example*	*comment*
module MomentumAperture		activate module
string momApFileName =	"input/MomentAp.in"	name of input file

The MomentumAperture input file contains the information for each node. The format of the file must be as follows (a blank between items)

name of the node	node order number	:*first line*
max(dp/p)	calcFreq	:*second line*

skip a line and repeat for the next MomentumAperture node

Example of Momentum Aperture file:

```
MomentAp1    5
10.0         3
```

9.6 Thin Multipole

command	*example*	*comment*
module ThinMPole		activate module
string thinMPoleSpecFile =	"input/ThinMPole.in"	name of input file

The ThinMPole input file contains the information for each node. The format of the file must be as follows (a blank between items)

name of the node	node order number		:*first line*
order of multipole	integrated strength[1]	skew variable	:*second line*

skip a line and repeat for next ThinMPole node

NOTES: [1] integrated strength of the field expansion (kl).

Example of Thin Multipole file:

```
TMP1    15
1       2.0    0

TMP2    25
2       3.0    1
```

9.7 Lattice Kicks

command	*example*	*comment*
module LatKicks		activate module
string tKickInput =	"input/TKicks.in"	name of input file

9.8 RF Cavity

To be used if you have a RF cavity at constant voltage and fixed beam energy (No Ramp)

command	*example*	*comment*
module RFCavity		activate module
string rfSpecFile =	"input/RFCav.in"	name of input file

The RFCavity input file contains the information for each RFCavity. The format of the file must be as follows (a blank between items)

node name	no of harmonics	node index	function	:*first line*
volt(1)[kv]	harmonic number	phase(1)		:*second line*
volt(2)[kv]	harmonic number	phase(2)		:*third line*
...				

skip a line and repeat for the next RFCavity node

Example of RF Cavity file

```
RF1     2      75     CONSTVOLTS
40.0    1.0    0.0
-20.0   2.0    0.0

RF2     2      100    CONSTVOLTS
41.0    3.0    0.0
-21.0   2.0    0.0
```

10 Output

command	*example*	*comment*
string of1 =	"Ring.dat"	name of file containing the ring structure
int showSingleNode =	[0/1]	display node information [no/yes] [1]
int outputScreenToFile =	[0/1]	output screen data to file [no/yes]
string outputFileName =	"screen.dat"	name of output file [2]
int dumpLostParts =	[0/1]	dump lost particles to a file [no/yes]
string lostPartsFileName =	"LostParts.dat"	name of file to contain lost particles
int showSingleNode =	[0/1]	display some info to screen after every node

NOTES: [1] switch to display some node information to screen after every node completes.
[2] Output file contains:

Turn	Time	n-Macros	n-Reals	hits/p	e-X-RMS	e-Y-RMS	eps-X	eps-Y	BF

11 Plot

Graphic output is done with *GNUPLOT*. One can run GNUPLOT interactively, i.e. getting continuous grapic output on the screen while *SIMBAD* is running, or dump data to graphic postscript files. Interactive plot is not possible with parallel runs. Command and setting for GNUPLOT are

command	*example*	*comment*
module GPlot		activate GPlot module
string gpNodename =	"gplot"	*GNUPLOT* node name
int gpPSOut =	[0/1]	output plots to postscript [no/yes]
string gpFilenameBase =	"HESR1.gp"	base name of GPlot postscript files
int gpScrOut =	[0/1]	output plots to screen [no/yes]
int gpNodes =	1	number of GPlot nodes
int gpInitialOIndex =	15	GPlot beginning node index
int gpOrderSpacing =	1000	GPlot node order spacing
int gpActTurnInt =	40	turn activation interval (..every n turns..)
int startPlots =	20	turn number in which to start plotting
int stopPlots =	600	turn number in which to stop plotting
int prune =	10	prunes the number of particles plotted [1]
int x_yPlot =	[0/1]	plot y vs x [no/yes]
int x_xpPlot =	[0/1]	plot xp vs x [no/yes]
int y_ypPlot =	[0/1]	plot yp vs y [no/yes]
int phi_dEPlot =	[0/1]	plot dE vs phi [no/yes]
int w_dots =	[0/1]	allows feature "with dots" [no/yes]
string gpScrSize =	"800x800"	screen size
int setRanges =	0	set ranges [0/1=no/yes] [2]

NOTES: [1] i.e. if prune = 10 then only 1 in 10 particles will be plotted.
[2] 0 = set autoscale. If 1, set ranges.

If setRanges = 1, no autoscale, plot ranges are prescribedd as

int setRanges =	1	
double xy_xMin =	-100.0	x-y range
double xy_xMax =	100.0	
double xy_yMin =	-100.0	
double xy_yMax =	100.0	
double x_xp_xMin =	-100.0	x-xp range
double x_xp_xMax =	100.0	
double x_xp_yMin =	-100.0	
double x_xp_yMax =	100.0	
double y_yp_xMin =	-100.0	y-yp range
double y_yp_xMax =	100.0	
double y_yp_yMin =	-100.0	
double y_yp_yMax =	100.0	
double phi_dE_xMin =	-3.14	phi-dE range
double phi_dE_xMax =	3.14	
double phi_dE_yMin =	-0.01	
double phi_dE_yMax =	0.01	

12 Diagnostics

SIMBAD has many diagnostics modules, inherited from *ORBIT*. Grdually, we are moving out of them, relying instead on post processing the fundamental output files, especially the Phase Dump. To generate a Phase Dump file we need some commands.

12.1 Phase Space Diagnostics

command	*example*	*comment*
module PhaseSpace		activate module
string phaseSpaceFName =	"input/PhaseSpace.in"	name of input file

The PhaseSpace input file contains the information necessary for each PhaseSpace node. The format of the file MUST be as follows. The first line contains these records (separated by blanks)

name of the PS Diagnostics node
name of the file to which dump information
node index
turn interval for activation of the node[1]
Freeze beam [1/0=yes/no]
modulo for output of particles [2]
skip a line and repeat for the next PhaseSpace Diagnostic node

NOTES: [1] f the turn interval activation number is 1, then diagnostic information will be dumped every turn, else every n turns. [2] 1=all particles and must be 0.

Example of Phase Spase Diagnostics file

```
PSnode1   "PSDump1.dat   1425   0   10   10

PSnode2   "PSDump2.dat   1725   0   10   10
..
```

12.2 Tune Diagnostics

Betatron and Synchrotron tune are also calculated in *SIMBAD* with various methods.

For the betatron tunes, a simple method is to count the number of transverse oscillations by every macro per turn, or, better to compare the average wavelength of a transverse oscillation with the length of one turn, for many turns. This method, still embedded int *SIMBAD* proper, requires a large number of turns for a good accuracy, and for the very same reason cannot be used to follow the evolution of the tune of a specific macro or a number of macros turn by turn. A second method for the betatron tune, also in the body of *SIMBAD*, is based on the FFT of orbits. This method can give interesting results -e.g. higher order tunes- but requires again many turns.

An alternative method, much more powerful, is based on the analysis of the eigen values of the one turn matrix. Betatron and synchrotron tunes can be calculated at every turn, using the phase space for the six preceding turns, with great accuracy. This method is impemented in a Post Processor, or Utility subroutine, described in Sec. 15.

The built-in routines for tune calculations are driven by the following

command	*example*	*comment*
module Tunes		activate module
string tunesFileName =	"input/tunes.in"	name of input file
int startTune =	400	turn to start the tune calculation[1]
int stopTune =	499	turn to stop the tune calculation
int calcCoherentTune =	[0/1]	flag to calculate the coherent tune
string coherentTuneFilename =	"CoherentTune.out"	coherent tune output file
int calcIncoherentTune =	[0/1]	flag to calculate the incoherent tune

NOTES: [1] Should have stopTune - startTune ¿ 10 to get a good statistics. -1 means never.

The Tunes input file contains the information necessary for each Tunes node. The format of the file must be as follows (a blank between items)

name of the node
The name of file to which dump information
node order number

Example of Tune Diagnostics file

```
TunesNode1   "tunes1.dat"   13
```

12.3 Momentum Diagnostics

command	*example*	*comment*
module Moment		activate module
string momentFileName =	"input/Moment.in"	name of input file
int momentOrder =	2	order to which moments are calculated

The format of the file must be as follows. The first line contains these records (items separated by blanks)

name of the node
The name of file to which dump information
node order number
turn interval
skip a line and repeat for the next RectAperture node

Example of Momentum Diagnostics file

MomNode1	"MomDump1.dat	1512	0	100
MomNode2	"MomDump2.dat	1712	0	100
..				
..				

12.4 TSC Kicks Diagnostics

command	*example*	*comment*
module TSCKicks		activate module
string tscKicksSpecFile =	"input/tscKicks2D.in"	configuration file

The TSCKicks input file contains the information required for each TSCKicks diagnostic node. The format of the file must be as follows (items separated by blanks)

name of the node
name of file to which dump information
NTSC element name [1]
turn interval
start turn
stop turn

NOTES: [1] NTSC is the transverse space charge element number that should immediately precede this diagnostic.

12.5 Accelerate Diagnostics

command	*example*	*comment*
module Accelerate		
string accelFileName =	"input/Accel.in"	configuration file
int calculateBucket =	[0/1]	flag to enable calculation of bucket

The Accelerate input file contains the information required for each Accelerate diagnostic node. The format of the file MUST be as follows

name of the node
name of output file
node order number
integer to specify the type of RF associated [1]
ordinal number of the specific RF node associated [2]
turn interval
skip a line and repeat for the next RectAperture node

NOTES: [1] 0 = an RFCavity node, and 1 = a RampBAccel node. [2] starting with zero

Output contains

Turn	Time [msec]	n-Macros	T-Sync [GeV]	B_f	beta	phase [deg]	Volt [kV]	w_synch [Hz]	E_bck [MeV]	E_bnc [MeV]	A_bck [eV-s]	A_bnc [eV-s]	dp/p %

For example, if you have a single RFCavity node in the ring at position 20 you might use the following...

```
AccNode1   "AccDump1.dat"   21   0   1   100
```

If you had 3 RampBAccel nodes in the ring at positions 51, 101, and 201 you might use the following...

```
AccNODE2   "AccDump2.dat"   52    1   0   100
AccNODE3   "AccDump3.dat"   102   1   1   75
AccNODE4   "AccDump4.dat"   202   1   2   50
```

12.6 Canonical Coordinate Diagnostics

command	*example*	*comment*
module CanonicalCoords		activate module
string ccFileName =	"input/CanonCoords.in"	name of configuration file

The Canonical Coordinate conf file contains the information required for each CC diagnostic node. The format of the file must be as follows (items separated by blanks)

name of the node	name of output file	node index	turn interval
skip a line and repeat for the next CCDiag node			

Example

```
CCNode1   CC1.dat   8   1
```

12.7 Transverse Emittance Diagnostics

command	*example*	*comment*
module Emittance		activate module
string emitFileName =	"input/TEmit.in"	name of config file

The Transverse Emittance conf file contains the information required for each TEM diagnostic node. The format of the file must be as follows (items separated by blanks)

name of the node	name of output file	node index	turn interval
skip a line and repeat for the next TEM node			

Example

```
EMITNode1   "TEM1.dat"   8   100
```

Output file contains for every print step:

X RMS Emittance π[mm-mrad]	X max Emittance π[mm-mrad]	Y RMS Emittance π[mm-mrad]	Y max Emittance π[mm-mrad]

12.8 Longitudinal Emittance Diagnostics

command	*example*	*comment*
module LongEmittance		
string lEmitFileName =	"input/LEmit.in"	name of conf file

The Longintudinal Emittance input file contains the information necessary for each LongEmit node. The format of the file must be as follows (items separated by blanks)

name of the node	name of output file	node index	turn interval
skip a line and repeat for the next LEM node			

Example

```
LEMNode1   "LEM1.dat"   8   50
```

12.9 Action Diagnostics

command	*example*	*comment*
module Actions		activate Action Diagnostics module
string actionsFileName =	"input/Actions.in"	name of conf file

The Actions input file contains the information necessary for each Actions node. The format of the file must be as follows (items separated by blanks)

name of the node	name of output file	node index	turn interval
skip a line and repeat for the next RectAperture node			

Example

ActNode1 "Act1.dat" 9 15

12.10 Stat Lat Diagnostics

command	*example*	*comment*
module StatLat		activate StatLat Diagnostic module
string statlatFileName =	"input/StatLat.in"	name of conf file

The StatLat input file contains the information necessary for each StatLat node. The format of the file must be as follows (items separated by blanks)

name of the node	name of output file	node index	turn interval
skip a line and repeat for the next RectAperture node			

Example

SLNode1 "SL1.dat" 9 25

12.11 Check Points

Enable if program checkpointing is desired

command	*example*	*comment*
module CheckPoint		activate module
int chkPntTurnInt =	25	turn interval for writing checkpoint files
int cleanChkPnt =	[0/1]	clean checkpoint files on exit

13 Modularity, Pre- and Post-Processors

Our effort is to make the structure of *SIMBAD* as modular as possible, with the use of an expanding number of Pre- and Post- processors. The purpose is twofold: (i) we find it desirable to decrease the sheer size of the code, devolving many tasks to auxiliary codes than are used only if needed; (ii) we want *SIMBAD* to migrate and work within the "Unified Accelerator Library", or *UAL*, as a wrapper, and for this modularity is a prerequisite.

The old *ORBIT* was full of preparatory routines and diagnostic modules. A preparatory routine is the one described in section 6 above, that reads the output of *MAD*. This is still part of the present version of *SIMBAD*, but is gradually being replaced by a pre-processor that will be able to read the optics not only from other versions of *MAD*, but also from other codes, like *TEEPOT* or others. Another pre-processor creates the initial particle distribution in the herd. Since any diagnostics is based on some analysis of the output of the phase space (6 coordinates), we considered it more profitable to do these analyses with post-mortem processors instead than burden the basic tracking code with specialized routines.

14 Pre Processors

A collection of pre-processors are available for *SIMBAD*. One, of course is *MAD*, if we dare to call it a pre processor. *MAD* creates the input optics files that contain the twiss functions and the transfer maps, as described in Sect. 6. Only *MAD-8* is compatible with *SIMBAD* so far.

14.1 MAD

A typical *MAD-8* input (e.g. HESR1.mad) must contain the following commands

example of MAD command lines	*comment*
PC := 14.5	particle momentum
beam, particle=proton, momentum=PC	set up particle [1]
call filename='HESR1.lat'	lattice file
setopts, echo	insures that maps are output to .echo file
select, flag=SECOND, range = full	writes transfer maps
twiss, tape='HESR1.twiss',deltap=00	produces and writes twiss.file [2]

NOTES: [1] This is important for the longitudinal components of the transport maps that depend on energy. [2] The created .echo file will carry the same name as the .mad file. In our example HESR1.echo

14.2 MAKEPOP: Particle Distribution Creator

The input particle distribution for tracking can either be creates inside *SIMBAD* or read from an external file, as described in Sect. 5.

An external distribution is created with the Fortran-77 code *MAKEPOP2*, based on formalism presented by G.Franchetti [9]. The operation of the code is as follows

> makepop2 < input_file

An example of input file containing the parameters for the distribution is

variable	*value*	*comment*
&par		namelist group begin mark
npart =	10000	number of particles in the distribution
Tdist =	'G1'	transverse distribution=random Gaussian [1]
iseed =	1812	seed for random number generation
alfax =	-0.33569	Twiss-α_x
betax_m =	68.8921	Twiss-β_x
ex_m_rad =	1.d-6	X-emittance [m-rad]
alfay =	-4.4297	Twiss-α_y
betay_m =	148.107	Twiss-β_y
ey_m_rad =	1.d-6	Y-emittance [m-rad]
dx_m =	0.d0	horizontal displacement of beam
dy_m =	0.d0	vertizontal displacement of beam
LDist =	'LG'	longitudinal distribution Linear-Gaussian [2]
Dphi_deg =	240.	r.m.s. Φ width [deg]
dEo =	0.014	r.m.s. ΔE half-width [GeV]
outfile =	'HESR1-e5-240-em-6-de-3.dis'	generated distribution file
/		namelist group end mark

NOTES: [1] Transverse options are 'G1' (Gaussian), 'L' (Linear), 'KV' (Kapchinskij-Vladimirskij), 'WB' (water bag). [2] Longitudinal distributions are: 'LG' (Linear-Gaussian), 'PG' (Parabolic-Gaussian), 'GG' (Gaussian-Gaussian)

15 Post Processors

A collection of post-processing routines are in the folder "utils" of the *SIMBAD* distribution. They are mostly operating on the Phase Space output file. The operation of each PostProc is described below; a description of their operation appears on the screen by simply invoking the routine.

First, let us recall that the Phase Space output consists of a file containing the macro number and the six phase space coordinates of the macro. The file is arranged in sectors, on for each turn dumped, separated by two blank lines. This arrangement permits to plot phase space diagrams for each turn separately using the "i" (index) option of *GNUPLOT*. The "prune" capability of the Phase Space Diagnostics module of SubSec. /refsubs:PSDiag allows to dump to the file only a subset of the macros used in the simulation, in order to limit the size of the file itself.

15.1 ttunes

This C++ routine operates on the Phase Space output file. Its purpose is to calculate the betatron and synchrotron tune from the eigenvalues of the One Turn Matrix (OTM) built from six turns of each macroparticle [10]. The output of *ttunes* can be either a file containing the tunes for all turn is dumped or a specific turn. Another alternative dump contains the OTM and the eigenvalues for a specific particle.

To work correctly, one should calculate the tunes at a location in the machine where the β-Twiss function for X and Y are as different as possible from each other. This will facilitate the attribution of the eigenvalue to the appropriate coordinate.

Let us invoke the routine

```
> ttunes
```

usage:

```
ttunes −i  <filename_in > -x <betaX> -y <betaY> [-m <M-partno¿ -n <recno>] [-a <average num>]

The input, filename_in should contain the phasespace output with data dumped each turn.
betaX should be the betaX value associated with the phasespace output
betaY should be the betaY value associated with the phasespace output
If the calculated matrix is desired M-partno is the particle number
for which the matrix calculated after the initial records are processed
and recno is the record number in which to print the matrix will be stored to file "matrix.dat".
The processing of the file will end at that point.
average_num is the number of turns over which the tunes are averaged. This means that the tune output
will be average_num turns per record.
average_num must be an integer greater than 1
Data output, consisting of partno, tunex, tuney, tunez,alphaX, betaX, alphaY, betaY, alphaZ, betaZ (OTM),
is on stdout, while status output, which consists of record (turn) number and particle count,
is on stderr.
```

Example to write tunes (and macro number) for each macro and each turn to a file "tune.dat". In this example the values of the : β-functs at the chosen locasion were 69 [m] and 148 [m], respectively

```
ttunes -i PSDump.dat -x 69. -y 148. > tunes.dat
```

Example to write the OTM for macro no. 10, at turn 100, to a file"matrix-10.100.dat" (first record has index 0)

```
ttunes -i PSDump.dat -x 69. -y 148. -m 10 -n 99 > matrix-10.100.dat
```

15.2 tunecontour

This C++ routine was designed to operate on output files created by the Tune Diagnostic module of SubSec. 12.2. It creates *GNUPLOT* compatible file for 2D and 3D contours of betatron data in the $Q_x - Q_y$ plane.

Let us invoke the routine

```
> tunecontour
```

usage:

```
tunecontour -d <dimension> -i <filename_in> -o <filename_out>

contours may be plotted using gnuplot with the following commands:
set cntrparam levels auto <number>
set nosurface
set contour
set view 0,0
splot '<filename_out>' w l
```

15.3 picktunes

This C++ routine was designed to operate on output files created by the Tune Diagnostic module of SubSec. 12.2. It picks up the betatrn tune of specific particles

Let us invoke the routine

```
> picktune
```

usage:

```
picktunes -i <filename_in> [-t <0,1>]

The input file should contain the tunes output.
Enter the particle numbers on std input.
The last entry should be the word "end"
The output is printed on stdout
If -t has argument 0, the format of the input is assumed to be the ORBIT tunes output file.
If -t has argument 1, the format of the input is assumed to be the output from ttunes.
The default is 1.
```

15.4 pickparts

This C++ routine was designed to operate on output files created by the Phase Space Diagnostic module of SubSec. 12.1. It picks up the phase space coordinates of specific particles for all turns dumped.

Let us invoke the routine

```
> pickparts
```

usage:

```
pickparts -i <filename_in> -o <filename_out> [-ns]

the input file should contain the phasespace output
enter the particle numbers on std input
the last entry should be the word "end"
the output file will contain the phasespace for those particles
-ns specifies that no spaces should be added between records
```

15.5 partsemit

This C++ routine was designed to operate on output files created by the Phase Space Diagnostic module of SubSec. 12.1. It calculates the transverse emittance of the Herd for all turns dumped with the formula

$$\epsilon_x = \left[< \hat{x}^2 >< (\hat{p_x}^2 > - < \hat{x}\hat{p_x} >^2 \right]^{1/2}, \hat{x} = x - < x >, \hat{p_x} = p_x - < p_x >$$

Let us invoke the routine

```
> partsemit
```

usage:

```
partsemit -a <alpha> -b <beta> -e <emmit> -l <lowerbnd> -u <upperbnd> -i <file_in>

each particle is tested against the c-s invariant < —lowerbnd—([upperbnd—)*emmit
the input file should be a PhaseSpace diagnostic file
the output will be printed to stdout
that are within the emittance range specified
```

15.6 longemit

This C++ routine was designed to operate on output files created by the Phase Space Diagnostic module of SubSec. 12.1. It calculates the longitudinal emittance of the Herd

Let us invoke the routine

```
> longemit
```

usage:

```
longemit -i <filename_in> -s <E> [-r <record♯>] [-e <E0>]

the input file should contain the phasespace output
<E> is the energy of the synchronous particle (GeV)
<record♯> denotes the record number in the phasespace output
<record♯y> should index starting from 0
<E0> is the rest energy of the particle (GeV), default is proton
```

15.7 extractrec

This C++ routine was designed to operate on output files created by the Phase Space Diagnostic module of SubSec. 12.1. It selects a single record (turn) from the file

Let us invoke the routine

```
> extractrec
```

usage:

```
extractrec -i <filename_in> [-r <record♯>]

the input file should contain the phasespace output
<record♯> denotes the record number in the phasespace output
<record♯> should index starting from 0
default value is 0
```

15.8 binphsp

This C++ routine was designed to operate on output files created by the Phase Space Diagnostic module of SubSec. 12.1. It bins and creates 3D histograms of the phase space

Let us invoke the routine

```
> binphsp
```

usage:

```
binphsp −x < xdim > −y < ydim > [−xparm < 1 − 6 >][−yparm < 1 − 6 >]
[−xmax < xmax >][−xmin < xmin >][−ymax < ymax >][−ymin < ymin >]
[−t < type >][−z < plotZero >] − i < inputfile >

binphsp bins phasespace parameters yparm vs xparm using a grid
(xmax-xmin) * (ymax-ymin) in size with xdim * ydim grid points
the input file should be the PhaseSpace output file
the type may be specified as 0 or 1 where 0=2D and 1=3D
plotZero may take values of 0 or 1 where 0=no and 1=yes
plotZero should be set to 1 if contour lines are desired
this will result in greater output volume
```

References

[1] R.W.Hockney and J.W.Eastwood: *Computer Simulation Using Particles*. Adam Hilger, IOP Publishing, New York, 1988.

[2] J.D.Galambos, J.A.Holmes, D.K.Olsen A.Luccio and J.Beebe-Wang: *Orbit User's Manual Vers. 1.10*. Technical Report SNS/ORNL/AP 011, Rev.1, 1999.

[3] T.Williams, C.Kelley: *Gnuplot Release 4-0-2*. Technical Report UTC, http://www.gnuplot.info, April 12, 2004.

[4] H.Grote and F.Ch.Iselin: *The MAD program, Vers.8.19*. Technical Report CERN/SL/90-13, Geneva, Switzerland, 1996.

[5] F.W. Jones, G.H. Mackenzie and H. Schönauer: *Accsim - A Program to Simulate the Accumulation of Intense Proton Beams*. Particle Accelerators, 31:199, 1990.

[6] F.Ch.Iselin: *The MAD Program. Version 8.13. Physical Methods Manual*. Technical Report CERN/SL/92-?? (AP), European Organization for Nuclear Research, Geneva, CH, 1994.

[7] J.A.MacLachlan: *ESME: Longitudinal Phase Space Particle Tracking-Program Documentation*. Technical Report Fermilab, TM-1274, 5/84, 1984.

[8] J.A.MacLachlan: *Longitudinal Phase Space Tracking With Space Charge and Wall Coupling Impedance*. Technical Report Fermilab, FN-446, February 1987.

[9] G.Franchetti: *Method for generating a generic matched distribution in N dimensions*. Technical Report GSI Report, unpublished, March 14 2003.

[10] A.U.Luccio and N.L.D'Imperio: *Eigenvalues of the One-turn Matrix*. Technical Report C-AD/AP/126, Brookhaven National Laboratory. Upton, NY, December 2003.