# Spink Users Manual. Version v.2-21-beta 

A. U. Luccio<br>August 2007<br>\section*{Collider Accelerator Department}<br>Brookhaven National Laboratory

## U.S. Department of Energy <br> USDOE Office of Science (SC)

[^0]
## DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or any third party's use or the results of such use of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof or its contractors or subcontractors. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

# Spink User's Manual. Version v.2-21-beta 

A. U. Luccio



Collider-Accelerator Department
Brookhaven National Laboratory
Upton, NY 11973

# Spink User's Manual. Version v.2-21-beta * 

A.U. Luccio<br>Brookhaven National Laboratory, Upton NY 11973

July 31, 2007


#### Abstract

spink is a spin tracking code for polarized spin $1 / 2$ particles. The code tracks both trajectories in 3D and spin. It works using thick element modeling from $M A D$ to track trajectories and thin element modeling based on the BMT equation to track spin. The code is written in Fortran 77 and typically runs on a Linux platform, either sequentially or MPI-parallel.


[^1]
## Acnowledgments

spink has been around for several years. We used and still extensively use it in Brookhaven for polarized proton studies at the AGS and RHIC. The code has been exported and used in other laboratories, notably Jülich in Germany and RIKEN and KEK-JPARC in Japan.

Many peoples have contributed with discussions and actual work on the code, notably Haixin Huang, Nick Tsoupas, Mei Bai, Fanglei Lin at Brookhaven, Andreas Lehrach at Jülich and Meiqin Xiao at RIKEN. Nick D'Imperio (BNL) contributed in a fundamental way to the implementation of the GUI. Constant interest and support by Thomas Roser has been important for the success of the code.

Tracking spin has been always an important tool to understand and to resolve problems of the acceleration of polarized particles. For circular accelerators several code have been written to address full or in part some of the issues, such as depol by Ernest Courant (BNL), snake by Jean Buon (Saclay), sprint by MathiasVogt (DESY). Partly of fully implementation of spin tracking exist in other codes, like MAD (CERN) and COSY-infinity (Michigan).
spink is just another contribution.

## Contents

1 Physical Principles ..... 5
1.1 Orbit and spin tracking ..... 5
1.2 Siberian Snakes and Spin Rotators ..... 6
2 Flowchart ..... 6
3 Include files ..... 6
4 spink routines ..... 8
4.1 routine SpinkData ..... 8
4.2 routine FOpen ..... 8
4.3 routine NmlRead ..... 8
4.4 routine Tables ..... 8
4.5 routine TrackInit ..... 9
4.5.1 Betatron tune adjustment ..... 10
4.5.2 Discarded elements ..... 10
4.6 routine Track ..... 11
4.7 routine MiscInit ..... 12
4.8 routine Makepop ..... 13
4.9 routine Popul8 ..... 13
4.10 routine SyncParticle ..... 13
4.11 routine WorkParticle ..... 13
4.12 routine Orbit ..... 14
4.12.1 Treatment of lattice errors ..... 14
4.13 routine Acceleration ..... 15
4.14 routine RayTransfer ..... 16
4.15 routine MatrixUpdate ..... 16
4.16 routine OneTurnOrbitMatrix ..... 17
4.17 routine Sprot ..... 17
4.18 routine SpinMat ..... 18
4.19 routine Helix ..... 18
4.20 routine SpinGoodies ..... 19
4.21 routine Bumps ..... 20
4.22 routine Output ..... 20
4.23 routine RFDipole ..... 21
4.24 routine SpinFlipper ..... 22
4.25 routine Tunes ..... 22
4.26 routine Coup ..... 22
5 Diagnostics routines ..... 23
5.1 routine Diagnostics ..... 23
5.2 routine Sequence ..... 23
6 Additional routines ..... 23
6.1 routine FroissartStora ..... 23
6.2 routine BeamBeam ..... 23
6.3 routine OrbitFFT ..... 24
6.4 routine MPIOutput ..... 24
6.5 routine Clock ..... 24
7 Pre Processors ..... 24
7.1 mad8 ..... 24
7.2 mad_read ..... 25
7.3 GUI ..... 26
7.3.1 GUI: input files ..... 26
7.3.2 GUI: output files ..... 27
7.3.3 GUI: diagnostics files ..... 28
7.3.4 GUI: tables ..... 29
7.3.5 GUI: flags ..... 29
7.3.6 GUI: dynamics ..... 30
7.3.7 GUI: Orbit FFT ..... 30
7.3.8 GUI: Population ..... 31
7.3.9 GUI: Synthetic Snakes ..... 31
7.3.10 GUI: Synthetic Spin Rotators ..... 31
7.3.11 GUI: Helix ..... 32
7.3.12 GUI: Stable Axis ..... 32
7.3.13 GUI: Froissart-Stora ..... 32
7.3.14 GUI: Spin Flipper ..... 32
7.3.15 GUI: BeamBeam ..... 33
7.3.16 GUI: RF Dipole ..... 33
7.3.17 GUI: Cycle ..... 33
7.4 Compilation and Run ..... 34
8 Suggested improvements ..... 34
List of Figures
1 Flowchart of spink and its preprocessors ..... 7
2 SPINK driving menu ..... 36
3 SPINK modules ..... 36
4 Left: input files, Right: output files. If a file name has a \# as a first character, the file will be ignored. The opening of some file is equivalent to a flag to do something ..... 37
5 Left: diagnostics files. Right: available files. If a file name has a \# as a first character, the file will be ignored. The opening of some file is equivalent to a flag to do something 38
6 Left: Tables. If a table name starts with $\#$, the table will be ignored. The openingof some table is equivalent to a flag to do something.Right: Logical flags39
7 Left: Dynamic variables for tracking. Right: OrbitFFT variables ..... 40
8 Left: Particle distribution variables. Right: Synthetic snakes parameters ..... 41
9 Left: Synthetic snakes parameters. Right: Helix parameters ..... 42
10 Left: Parameters for stable axis calculation. Right: Parameters for Froissart-Stora calculation ..... 43
11 Left: Spin flippers parameters. Right: BeamBeam parameters ..... 44
12 Left: Radio frequency dipole parameters. Right: Cycle parameters ..... 45

## 1 Physical Principles

spink, that may stand for 'Spin Tracking', is a spin tracking code for spin $1 / 2$ polarized particles [1]. The six phase space coordinates for the orbit and three coordinate representing the cartesian components of the unitary spin vector are tracked. Orbit tracking is done by propagating the vector $\vec{r}$ through first and second order maps generated by $M A D[2]$. These maps are for "thick" elements. Spin tracking is done via "thin" elements spin rotation matrices.

The output of $M A D$ is transformed by the pre-processor Mad_read to an input file (lattice descriptor) suitable for spink. Two versions of $M A D$ are currently supported by mad_read: mad8c (CERN) and BNLMad (J.Niederer). Work is in progress to accept input from madX, which is the only version of $M A D$ presently supported by CERN, see 7.1.
spink sequentially tracks several particles, averaging the final results. This task can also be performed "trivially" on a parallel computer since the code has been implemented using the MPI library.
spink uses MKSA units and canonical orbital phase space variables as used by $M A D$, namely

$$
\begin{equation*}
\mathbf{r}=\left(x, x^{\prime}=\frac{p_{x}}{p}, y, y^{\prime}=\frac{p_{y}}{p}, \Delta \phi=-c \Delta t, \frac{\Delta E}{p c}\right) . \tag{1}
\end{equation*}
$$

The spin is treated as a 3-dimension real vector

$$
\begin{equation*}
\mathbf{S}=\left(S_{x}, S_{y}, S_{z}\right) \tag{2}
\end{equation*}
$$

We follow the 'US accelerator' coordinate convention with $x$ radial, $y$ vertical, and $z$ longitudinal, while in the spin literature $z$ is often taken as vertical.

Input and output of phase space are in [mm] and [mrad] for transverse coordinates, $\left(x, x^{\prime}\right)$, radial and $\left(y, y^{\prime}\right)$, vertical. In input units are [rad] for the longitudinal coordinate $\Delta \phi$ and $[\mathrm{GeV}]$ for the energy coordinate $\Delta E . \Delta \phi$ and $\Delta E$ are evaluated with respect to the phase $\phi_{s}$ and energy $E_{s}$ of a reference synchronous particle.

### 1.1 Orbit and spin tracking

spink uses $M A D$ thick elements matrices to track particle orbits to first order. For second order the code uses either second order $M A D$ generated maps, or thin element representation of machine elements that would require high order transfer map representation, such as multipoles.
$M A D$ transfer maps are written for coordinate units [ m ] and [rad] in the transverse phase space, [ m ] for the phase, since in $M A D$ it is $\Delta \phi \rightarrow-c \Delta t$, and [ 0 ] for the energy coordinate, since in that code it is $\Delta E \rightarrow \Delta E / p c$. In spink we use $M A D$ coordinates.

The propagation of the transverse coordinates of the beam through the lattice is done using the equation

$$
\begin{equation*}
r_{i}^{(\beta)}=\sum_{j} R_{i j} r_{j}^{(\beta)}+\sum_{j k} T_{i j k} r_{j}^{(\beta)} r_{k}^{(\beta)}, \quad i, k=1,4 \tag{3}
\end{equation*}
$$

with $R_{i j}$ first order matrices, $T_{i j k}$ second order maps, and $\mathbf{r}^{(\beta)}=\mathbf{r}-\mathbf{r}^{(C O D)}$ is the displacement in space and angle of the particle from the closed orbit. The propagation of the longitudinal coordinates is in particular discussed in Sec. 4.13.

The accelerator lattice used by spink can be static, i.e. read once from $M A D$ and never modified, or dynamic. Two ways are implemented in the code to dynamically modify the lattice using tables: (a) reading from a table a list of $M A D$ and mad_read produced lattice descriptors for different energies, in which case, once a given energy is reached during acceleration, the track initializer routine TrackInit, Sec. 4.5, will read the next lattice; (b) reading tables of K1 values of all quadrupoles, etc. and update the transport matrices accordingly, Sec. 4.15.

The propagation of the spin coordinates is done by $3 \times 3$ matrices for thin elements. Spin matrices are simply derived from the BMT differential equation for spin motion [3], [4]

$$
\begin{equation*}
\frac{d \mathbf{S}}{d t}=\frac{e}{m c} \mathbf{S} \times\left[\frac{1}{\gamma}(G \gamma+1) \mathbf{B}-\frac{G \gamma}{\gamma+1}(\vec{\beta} \cdot \mathbf{B}) \vec{\beta}-\left(G+\frac{1}{\gamma+1}\right) \vec{\beta} \times \mathbf{E}\right] \tag{4}
\end{equation*}
$$

as described in Sec.?? [5]. $\mathbf{B}$ is the magnetic field, $\mathbf{E}$ is the electric field in the laboratory frame, and $\vec{\beta}=\mathbf{v} / c$ is the relativistic velocity of the particle.

### 1.2 Siberian Snakes and Spin Rotators

Siberian snakes [6] rotate the spin by a fixed angle. They are essential devices in circular accelerators for polarized beams to break the periodic condition that produces spin resonances and polarization losses.

Spin Rotators are used to orient the spin in a given direction. They are important in polarized beam colliders, to orient the beam polarization longitudinally or transversely at a collision point.

In spink, snakes and rotators are modeled in three ways:

- Synthetic
- Table driven
- Analytic

A synthetic snake or rotator is optically a thin element represented in the $M A D$ lattice by a unit transfer map. Angle of spin rotation $\mu$ and angles of the rotation axis $\phi$, longitude, and $\theta$, latitude are given as input, Sec. 7.3.9, and Sec. 7.3.10.

A table driven synthetic snake is a thick element, optically represented by the elements of a $6 \times 6$ matrix, tables(11) and (12), and by the three angles, as a function of the relativistic energy $\gamma$.

Analytic snakes and rotators are represented by orbit and spin matrices obtained from analytic expressions, Sec. 4.18.

## 2 Flowchart

The flowchart of spink and its pre-processors is shown in Fig.1. To run the code, three actions are needed, in sequence:

1. Run $M A D$ to generate the lattice descriptor. Presently it is mad8c, as described in Sec. 7.1. mad8 generates three output files: runname.twiss, runname.echo, runname.madout. The first two files are always needed; the third is only needed if there are lattice errors.
2. Whith the names of these files inserted in the input file mad_read.in, run mad_read. This generates a lattice descriptor that combines the three files in one file runname.sy, Sec. 7.2.
3. Using the GUI build the input file spink.in to spink, and run spink. These operations are conveniently done through a Unix script, Sec. 7.3.

The above flowchart only shows the main routines. Individual subprograms will be discussed in the following sections.

## 3 Include files

Two include files are supplied, SpinkGlobal and SpinkName

- SpinkGlobal contains the definition of data types for all the variables, integer, boolean, real, strings and also all the common blocks used in the program;
- SpinkName contains the namelist blocks, with the list of the input variables in the blocks. Input to spink is through namelist, contained in the input file spink.in.


Figure 1: Flowchart of spink and its preprocessors

## 4 spink routines

## 4.1 routine SpinkData

In this subroutine variables are initialized.

## 4.2 routine FOpen

Open files and tables.
In this subroutine the following functions are performed

- Namelist blocks are read and parsed through a call to NmlRead, Sec. 4.3
- Input and output files are opened. All file names are arbitrary.
- There is a convention here: if a file name has a \# (in the GUI), or a blank ' ' (in spink.in), as a first character, the file is ignored. The opening of a file is equivalent to set a flag, boolean $f s t a t(n)=$.true. with $n$ the number of the unit associated to the file. Some actions will be performed or not according to a file being open or not, as we will see later. If a file name starts with $\mathrm{a}^{\prime}+{ }^{\prime}$ as a first character, the file is opened with access=append.
- Some actions of the code are driven by a table of values vs. turn number in the tracking or vs. values of the energy, when there is acceleration in the synchrotron, as it will be described later, Sec. 4.4. Tables are initialized (opened) in this routine, with the same conventions used for the files. For tables the status boolean flag is $\operatorname{tstat}(n)$, with $n$ the table number.
- Some initialization is performed in FOpen, in particular for subsequent averaging: entry Param, and entry pInit. The program run ends also here: entry Finis.


## 4.3 routine NmlRead

Read namelist input.
This routine, called from FOpen, serves the dual purpose to accept a namelist driver (input file spink.in) created by the GUI or a namelist file directly edited. The point is that in a namelist file created by a GUI some namelist blocks are missing, when the corresponding module is checked off. In this case, Fortran would signal an error in reading and stop execution. A solution to this problem ${ }^{1}$ is implemented in the present subroutine.

NmlRead has also a quick diagnostics to signal any error in reding a namelist, a very common source of troubles, if spink.in is directly created by editing and not through the GUI.

## 4.4 routine Tables

Tables are being used in spink for functions or parameters that vary in the course of a run. Some tables use the number of turns as the independent variable (first column), some use the relativistic energy factor $\gamma$.

- The first line of a table contains an integer number: the number of columns after the first, for the rest this line is available for comments or a header.
- A table is read (opened) in the subroutine FOpen, Sec. 4.2. For tables we use the same convention as for files, namely if the first character of the table name is a \# (GUI) or a blank ' ' the table is ignored. Table names included in [ ] brackets (GUI) are intended to be name arrays.
- Tables are read in the entry Tables:TableRead, their content is temporarily put in a reserved buffer file, unit(11), and then stored in memory.

[^2]- if the first column is turn number, table entries are stored as $n T a b(k, j), v T a b(k, j, l)$ where $k$ is the table number, $j$ is the row number, $l$ is the column number.
- if the first column in the table is $\gamma$, table entries are stored as $\operatorname{gamTab}(k, j), v T a b(k, j, l)$.
- A specific record in a table can be retrieved calling the entry Tables:TableReadRecord
- Interpolation (linear) in the tables is done by calling the entry Tables:TableInterp
- A variable value in the last row of the table can be retrieved calling the entry Tables:TableInterpLast

The GUI windows for tables is described in Sec. 7.3.4.

## 4.5 routine TrackInit

Track initialization.
The subroutine reads the runname.sy file created by the pre-processor mad_read from MAD, as described in Sec. 7.2, and stores in memory values of the parameters.

- read 1.st record, for the entire lattice:

1: version of $M A D$, order of tracking (1.st or $2 . \mathrm{nd}$ ),
flag for errors in the lattice (0/1), total number of machine elements.

- for each machine element, read records $2,3,4$

2: el_name = element name, el_keyw = keyword, or element type, a 4 character string
el_len $=$ elem. length [m],
3: el_ang $=$ bend angle, el_k1 $=$ quadrupole gradient $=-(\partial B / \partial r) / B \rho,\left[m^{-1}\right]$,
4: el_k2 $=$ sextupole gradient $=-\left(\partial^{2} B / \partial r^{2}\right) / B \rho,\left[m^{-2}\right.$, el_tilt $=$ tilt angle [rad],
el_kickh $=$ horizontal kick [rad], el_kickv=vertical kick [rad]

- for each element, read records 5,6 if there are lattice errors:

5: values of el_der $=$ displacement $(\delta x, \delta y, \delta z[\mathrm{~m}])$, angle errors $(\delta \phi, \delta \theta, \delta \psi[\mathrm{rad}])$,
6: values of field errors $e l \_d k$.see $M A D$ manual

- for each element, read record 7 , if error, or 5:

7 [5]: values for the Distorted Closed Orbit, $r_{-} \operatorname{cod}=\mathrm{x}, \mathrm{y}, \mathrm{z}[\mathrm{m}]$, and the longitudinal position of the element end, s_cod [m].

- for each element, read records 8-13, if error, or 6-11:
$6 \times 6$ orbit transport matrix R_mat:,
- for each element, if track order $=2$, and error, read records 9-43, if no error read records 6-41: a $6 \times 6 \times 6$ second order transport map T_mat.
- finally, for the entire lattice:
values of mad_tune $=$ betatron tunes $\nu_{x}, \nu_{y}$, gamtr $=$ transition gamma $=\gamma_{T}$,
mad_circ $=$ total lenght of the ring $[\mathrm{m}], 6 \times 6$ one turn matrix OT_mat

Some spink keywords are not recognized by $M A D$. In this case, TrackInit uses their name as keyword. E.g. a (thin) Siberian Snake should be defined as a MARKER in MAD, with name SNAK as follows:

SNAK: MARKER
In spink both name and keyword will be SNAK.
Keywords used for different machine elements are

| Spink |  |  | MAD type |
| :---: | :---: | :---: | :---: |
| keyword | Spink type | Spink translation |  |
| INIT | marker |  | MARKER |
| MARK | marker |  | MARKER |
| DRIF | drift |  | DRIFT |
| SBEN | sector bend |  | SBEND |
| QUAD | quadrupole |  | QUADRUPOLE |
| SEXT | sextupole |  | SEXTUPOLE |
| MULT | multipole |  | MULTIPOLE |
| RFCA | RF Cavity |  | RFCAVITY |
| SNAK | Snake | el_name $\rightarrow$ el_keywd | defined as MARKER |
| SPIN | Spin Rotator | el_name $\rightarrow$ el_keywd | defined as MARKER |
| HELI | Helix | el_name $\rightarrow$ el_keywd | defined as MARKER |
| KICK | Kicker |  | KICKER |
| HKIC | Hor. Kicker |  | HKICK |
| VKIC | Vert.Kicker |  | VKICK |
| FLIP | Spin Flipper | el_name $\rightarrow$ el_keywd | defined as MARKER |
| RFDH | RF Dipole (H) | el_name $\rightarrow$ el_keywd | defined as MARKER |
| RFDV | RF Dipole (V) | el_name $\rightarrow$ el_keywd | defined as MARKER |
| SOLE | Solenoid |  | SOLENOID |
| MATR | Matrix |  | MATRIX |

### 4.5.1 Betatron tune adjustment

This is a special features of TrackInit.
Each quadrupole in the ring is padded by two thin addional quadrupoles, that provide a fine adjustment of the machine tune. This is useful is some run when one may want to scan a small interval of tunes with no need to run $M A D$ again for each tune.

The algorithm is contained in the subroutine Tunes:Detune.

### 4.5.2 Discarded elements

Sometimes the lattice contains many machine elements that do not rotate the spin and that we want to discard to expedite spink runs. TrackInit can find them on a file, unit(14), and just ignores them. This may save substantial run time.

## 4.6 routine Track

This is the most important routine in SPINK. Track is based on three nested loops

| call SyncParticle |  |  | Initialize Synch. particle |
| :--- | :--- | :--- | :--- |
| $\{$ | call WorkParticle |  | Loop on particles |
|  | call npInit |  | Read Work Particle |
|  | $\{$ | call ntInit | Initialize per particle |
|  |  | Loop on turns |  |
|  |  |  | Initialize per turn |
|  |  | call Orbit | Loop on components |
|  |  | call Sprot transfer | Spin rotation |
|  |  |  | End loop on components |
|  |  |  | Write Output at turn nt_pr |
|  |  |  | End loop on turns |
|  |  | End loop on particles |  |

The propagation of the orbit vector $\mathbf{r}$ is done in Orbit, Sec. 4.12, based on MAD maps, and the propagation of the spin vector $\mathbf{S}$ is done in Sprot, Sec. ?? with spin matrices based on the BMT equation.

It is important to recall that spink deals with low intensity polarized beams, so all particles in the tracking are independent. The only interaction is done when averaging variables over all particles in the beam. spink is essentially a sequentially structured code, where each particle is treated separately, hence the outer 'particle' loop.

Since tracking on a large accelerator with many lattice elements may take a considerable time, the use of a multi process parallel platform is essential to determine enseble results. Processors can be run independently, 'one case for each processor'. However, spink has also been implemented with the MPI library [7] for parallel computation. For this 'trivial' case MPI features must be enabled at compilation time, Sec. 7.4.

- The first call of Track is to SyncParticle, subroutine where the parameters of the reference (Synchronous) particle are set.
- The first call inside the particle loop (loop global variable $n p$ ) is to WorkParticle, subroutine where the parameters of each individual particle in the simulation are set. This routine reads the coordinates of each Work Particle either defined on the contour of an ellipse of given emittance (see the GUI, variables for the module Pop and relative help string) or produced randomly by the subroutine MakePop, Sec. 4.8.
- Some variables are initialized per particle in the entry npInit in the subroutine MiscInit. If active, tables are at this point initialized by a call to the entry TableInit of the subroutine Tables, Sec. 4.4.
- Then, the loop on turns starts. In this loop the running variable is the $n t$ turn number in the synchrotron (a global variable). Another leading variable is $G \gamma$, where G is the gyromagnetic anomaly of the particles $(G=1.7928456$ for the proton and $=-0.14301$ for the deuteron $)$ and $\gamma$ is the relativistic energy factor, "energy". A run is terminated at the maximum number of turns n_turn or, alternatively, by the initial and final value of $G \gamma$, whichever is reached first.
- The first call in the turn loop is to Table(8). This table, if activated, contains a list of names of $M A D$-created input lattice descriptors of the machine and is used to update the lattice at intervals in energy, Sec. 4.4.
- Then, the One Turn Spin matrix is initialized by call to the entry SpinGoodies:OneTurnSpinMatInit and quantities per turn are initialized by the entry MiscInit:ntInit, in particular the counters for special lattice elements, like snakes, Sec. 4.7.
- Then the loops on lattice elements starts. Track explores the lattice and finds all the elements, recognized by their keyword el_keyw, Sec. 4.5.
- The phase space vector for the orbit is updated by the subroutine Orbit, Sec. 4.12, and the spin is updated by the subroutine Sprot, Sec 4.17. Calls to the diagnostic entry Diagnostics:PrintAtElement are done if the diagnostics file unit(99) is open.
- The output is written to records when machine elements, spcified in the input, are encountered, by the entry Output:PrepareOtput. The record will be written to the specific files (units(51, $52,53, \ldots$ ) at turn end in the entry Output:WriteOutput.
- Many other operations are performed at the end of each turn, like performing averages or calculate betatron tune and spin tune.
- Similar operations are performed at the end of the other two external loops, turn loop, and particle loop.


## 4.7 routine MiscInit

Some variables must be initialized or re-initialized once per particle, once per turn and once per machine elements in the three nested loops of Track. This is done in MiscInit that has three entries:

- entry MiscInit:npInit. Initialization per particle. RFDipoles (tables(3) and (4)), Sec. 4.23, Froissart-Stora, Sec. 6.1, Spin Flipper, Sec. 4.24, and quantities to be averaged over many particles.
- entry MiscInit:ntInit. Initialization per turn of counters for helices, snakes, rotators, bumps, RF cavities, RF dipoles in the lattice.

$$
\begin{array}{ll}
\text { HLX_k, } & \text { counter for helices } \\
\text { SNK_k, } & \text { counter for snakes } \\
\text { ROT_k, } & \text { counter for spin rotators } \\
\text { BMP_k, } & \text { counter for orbit bumps } \\
\text { RF_k, } & \text { counter for RadioFrequency cavities } \\
\text { RFDH_k } & \text { counter for horizontal RF Dipoles } \\
\text { RFDV_k, } & \text { counter for vertical RF Dipoles. }
\end{array}
$$

- entry MiscInit:jelInit. Initialize and re-initialize quantities per element, such that field components and orbit curvature to zero.


## 4.8 routine Makepop

Generates a random beam with KV, Water Bag, Linear, or Gaussian distribution in 6D, matched in transverse to alfa, beta, emittance (rms) ${ }^{2}$.

Makepop writes a particle distribution to unit(13), input file to spink. If this unit is read, it overwrites any distribution directly generated according to Sec. 7.3.8. Each record of the file associated to unit(13) contains the phase space and spin coordinates of one particle

$$
\begin{equation*}
(\mathbf{r}, \mathbf{S}) \equiv\left(x, x^{\prime}, y, y^{\prime}, \Delta \phi, \Delta E / p c, S_{x}, S_{y}, S_{z}\right) \tag{5}
\end{equation*}
$$

## 4.9 routine Popul8

Function Popul8 = poprec, where poprec is the string of Eq. (5) containing the phase space and spin coordinate of a particle to be used in Track:WorkParticle.

If any of the external population files, unit(13), generated by Makepop, Sec. 4.8, or unit(12), manually edited, is enabled, a record of the file is read into Popul8 = poprec.

Otherwise, particle un-normalized transverse coordinates on an ellipse of emittance specified by input parameters, sec. 7.3 .8 are created. The distribution is normalized to an invariant emittance in WorkParticle, Sec. 4.11.

In this latter case, the algorithm to generate $n$ particles on a phase space ellipse is to put coordinates at equal angles of $2 \pi / n$, starting from the prescribed angle.

### 4.10 routine SyncParticle

Defines the synchronous (reference) particle in the beam. SyncParticle is the same for all particles, so it it is called before a particle (outermost) loop of Track is initiated

Sets initial and final quantities for a track run. Calculates the synchronous phase on the basis of the prescribed energy gain per turn, from input, Sec. 7.3.6, or from acceleration table(1), if enabled, and from the total RF cavities' accelerating voltage

$$
\begin{equation*}
\phi_{s}=\arcsin \frac{E_{0} \gamma_{s}}{e V_{t o t}} \tag{6}
\end{equation*}
$$

### 4.11 routine WorkParticle

Reads the initial coordinates of a particle from Popul8, Sec. 4.9, and builds work particle parameters. If the flag coup is true, coordinates are interpreted in the Edward-Teng couple mode, call Coup, Sec. 4.26.

If a non zero closed distorted orbit (COD) is present, transverse coordinates $r r$, absolute coordinate, and $r_{\beta}$ coordinates with rispect to the COD are established

$$
r r_{i}=r_{i}^{(\beta)}+r_{i}^{(C O D)}, \quad i=1,4
$$

Pendulum coordinates for the longitudinal motion are also initialized, Sec. 4.13.

[^3]
### 4.12 routine Orbit

Orbital phase space propagation. The orbital motion of the particles in spink is controlled by subroutine Orbit called by Track at each element encountered in the lattice.

Algorithms are applied in the following order:

- Define a new phase space variable $\mathbf{r}^{(\beta)}=\mathbf{r}-\mathbf{r}^{(C O D)}$, i.e. the particle trajectory with respect to the COD orbit. 5. th and 6 .th components of the COD vector are zero.
- Look for the element keyword, Sec. 4.5, and act accordingly. There is an entry for each type of element in the lattice.

| keyword | action | subroutine: entry |
| :--- | :---: | :--- |
|  |  |  |
| QUAD | call | MatrixUpdate: QuadMat |
| BEN | call | MatrixUpdate: DipoleMat |
| RFCA | call | Acceleration |
| SNAK | synth=return | Sec 1.2 |
|  | table=call | MatrixUpdate: SnakeMat |
| MATR | call | MatrixUpdate: SnakeMat, Sec. 1.2 |
| SPIN | return | Sec. 1.2 |
| HELI | return, or call | Helix:HelixOrbit. Sec 1.2 |
| RFDH | call | RFDipole: RFDOrbit |
| RFDV | call | RFDipole: RFDOrbit |
| KIC | call | Bumps: BumpOrbit |
| MARK | return |  |

- Call RayTransfer to propagate the phase space vector $\mathbf{r}$ to the next element according to Eq. (3).
- Back transform the phase space vector $\mathbf{r}=\mathbf{r}^{(\beta)}+\mathbf{r}^{(C O D)}$

Beam-Beam is recognized by name rather than keyword and is dealt with differently, Sxec. 6.2. So is SpinFlipper, Sec. 4.23

### 4.12.1 Treatment of lattice errors

If the machine lattice has errors, these errors are transmitted to spink via the $M A D$ output file runname.madout, Sec. 7.1, read and processed by mad_read. Lattice errors are misalignments in the three directions, $x, y, z$, and magnet errors, around axes oriented as $x, y, z^{3}$. To deal with misaligment (for spin dynamics vertical errors are especially important) the coordinates of a particle are displaced in Orbit with the opposite sign of the misalignment upon entering the element, and restored at the exit.

Similarly, the particle coordinate system should be rotated opposite to the rotation errors ${ }^{4}$.

[^4]
### 4.13 routine Acceleration

Longitudianl orbital phase space
When Track find a machine element whose el_keyw $=R F C A$, it increases the momentum of the synchronous particle according to the specified acceleration rate and the momentum of a working particle, proportionally to the applied RF voltage in the cavity. RFCA is an element type known to $M A D$. There can be several RF stations in the lattice, each with its voltage and harmonic number, as specified in the input, Sec. 7.3.6.

Acceleration can be done (1) at a constant rate (linear) by specifying in input the change of $\gamma$ per turn, or (2) reading a table(1) of $\gamma$ vs. turn number (if such a table is open, the constant rate is overwritten).

The algorithms that are applied, are, in this order:

- Increase the energy of the syncronous particle

$$
E_{s}:=E_{s}+\delta \gamma_{s, i} E_{0}
$$

where $\delta \gamma_{s, i}$ is the increase of $\gamma_{s}$ per turn in the $i-$ th cavity;

- Calculate the synchronous phase

$$
\phi_{s}=\arcsin \frac{E_{0} \delta \gamma_{s}}{e V}
$$

where $\delta \gamma_{s}=\sum \delta \gamma_{s, i}$ total variation of $\gamma_{s}$ per turn, and $e V=\sum e V_{i}$ total voltage in the cavities (vector sum);

- Transform the phase of the working particle being considered from the unit of $r(5)=-c \Delta t$, meters, to a unit of angle, radians

$$
\begin{equation*}
\Delta \phi=-2 \pi \frac{h_{i} \beta_{s}}{L} r(5) \tag{7}
\end{equation*}
$$

where $h_{i}$ is the harmonic number of the $i-$ th cavity, $\beta_{s}$ is the relativistic velocity of the synch. particle, $L$ is the entire length of the accelerator;

- Check if the energy is above transition. In this case, apply a phase jump

$$
\phi_{s}:=\pi-\phi_{s}, \text { and } \phi:=\pi-\phi
$$

- Apply a step increase to the energy of the working particle, calculated as

$$
\begin{equation*}
r(6):=r(6)+\frac{e V_{i}}{p c_{s}}\left(\sin \phi-\sin \phi_{s}\right) \tag{8}
\end{equation*}
$$

where the sixth phase space coordinate is $r(6)=\Delta E / p c$.

- Apply a kick to the transverse momenta of the particle ${ }^{5}$

$$
\begin{equation*}
p_{x}:=\frac{p_{x}}{1+\delta p / p}, \text { and } p_{y}:=\frac{p_{y}}{1+\delta p / p} \tag{9}
\end{equation*}
$$

where $\delta p / p=\delta \gamma_{s} /\left(\gamma_{s} \beta_{s}^{2}\right)$
The increment of energy per turn is subdivided in parts proportional to the voltage in each cavity

$$
\delta \gamma_{i}=\delta \gamma_{s} \frac{e V_{i}}{\sum e V_{i}}
$$

[^5]
### 4.14 routine RayTransfer

This routine actually propagate the 6 D phase space vector $\mathbf{r}^{(\beta)}$ across a machine element. The equation implemented here is Eq. (3), to first and second order tracking.

Since second order tracking may be rather lengthy for large lattices, we may want to execute it only for certain machine elements, e.g. sextupoles. An input string corresponding to a class of elements, Sec. 7.3.17, switches on this action for that class. The string could be: NONE (no element tracked to second order), ALL (all elements to second), SEXT (only sextupoles to second), etc.

### 4.15 routine MatrixUpdate

Update matrices for a dynamical representation of the lattice using tables.
The general form of a first order transfer matrix with no coupling ${ }^{6}$ is

$$
\mathbf{R}=\left(\begin{array}{cccccc}
c_{x} & s_{x} & 0 & 0 & 0 & D  \tag{10}\\
c_{x}^{\prime} & s_{x}^{\prime} & 0 & 0 & 0 & D^{\prime} \\
0 & 0 & c_{y} & s_{y} & 0 & 0 \\
0 & 0 & c_{y}^{\prime} & s_{y}^{\prime} & 0 & 0 \\
E & E^{\prime} & 0 & 0 & 1 & G \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
$$

with $c, c^{\prime}, s, s^{\prime}$ matrix elements for the transverse motion, $D, D^{\prime}$ dispersion terms, $E, E^{\prime}$ synchrobetatron terms, and $G$ the transition term, that can be defined through the momentum compaction. For a symplectic matrix it is $E^{\prime}=-D, E=-D^{\prime}$.

Entries of MatrixUpdate for different type of matrices are

- Entry MatrixUpdate:QuadMat. For a quadrupole whose name is found in table(9), the K1 strength vs. $\gamma$ is interpolated in the table and the $4 \times 4$ matrix transverse elements are updated. Note that to first order a quadrupole matrix has no dispersion terms, then no coupling between phase space variables $\phi$ and $\Delta E / p c$.
- Entry MatrixUpdate:DipoleMat. For a dipole with combined functions whose name is found in table(10), the K1 strength vs. $\gamma$ is interpolated in the table. The $4 \times 4$ matrix transverse elements are updated. this way. A dipole matrix has dispersion terms that are a function of energy, and also the transition element is a function of energy, namely

$$
D=A / \gamma^{2}, \quad D^{\prime}=B / \gamma^{2}, \quad G=C-H / \gamma^{2}
$$

with A, B, C, H quantities defined in MAD [8].

- Entry MatrixUpdate:SnakeMat. For Siberian Snakes whose name is found in tables(11) and (12), two for the moment, the optics is changing with energy, so all the matrix elements should be updated according with the tables.
${ }^{6}$ and for the most common case of a synchrotron entirely laying on a horizontal plane


### 4.16 routine OneTurnOrbitMatrix

Update the OneTurnOrbitMatrix to calculate betatron tunes turn by turn.
Betatron tunes are calculate in spink with three methods, two of them based on counting orbit betatron oscillation, Sec. 4.25 or by an FFTof the orbit, Sec. 6.3. In the present routine betatron tunes (the fractional part of) are calculated from the eigenvalues of the one turn orbit matrix, so they are available turn-by-turn.

The routine has three entries:

- entry OneTurnOrbitMatrix:OTMatrixInit. Here, the one turn matrix is initialized at the beginning of a turn.
- entry OneTurnOrbitMatrix:OTMatrixUpdate. Here, the one turn matrix is calculated at each machine element by progressive multiplication of the element orbit matrices.
- entry OneTurnOrbitMatrix:FracTuneFromOTMat. Here, the fractional part of the betatron tunes is calculated from the trace of the one turn matrix.


### 4.17 routine Sprot

Spin rotation,
At variance with orbital phase space propagation, spin rotation $3 \times 3$ matrices are built on the fly, because spin rotation is a function of the magnetic field found at the instantaneous position of the particle. Spin matrices are constructed starting from the BMT Eq. (4), for thin elements [9]

$$
\frac{d \mathbf{S}}{d t}=\vec{\omega} \times \mathbf{S}
$$

where $\vec{\omega}$ is a function of the magnetic field calculated at the instantaneous position of the particle.

$$
S_{n}=\sin \mu, C_{s}=1-\cos \mu
$$

The Spin rotation matrix $S \_m a t$ is

$$
\mathbf{M}^{(S)}=\left(\begin{array}{ccc}
1-\left(a_{2}^{2}+a_{3}^{2}\right) C_{s} & a_{1} a_{2} C_{s}+a_{3} S_{n} & a_{1} a_{3} C_{s}-a_{2} S_{n}  \tag{11}\\
a_{2} a_{1} C_{s}-a_{3} S_{n} & 1-\left(a_{1}^{2}+a_{3}^{2}\right) C_{s} & a_{2} a_{3} C_{s}+a_{1} S_{n} \\
a_{3} a_{1} C_{s}+a_{2} S_{n} & a_{3} a_{2} C_{s}-a_{1} S_{n} & 1-\left(a_{1}^{2}+a_{2}^{2}\right) C_{s}
\end{array}\right)
$$

with

$$
\begin{gathered}
\left\{\begin{array}{l}
a_{1}=C\left[(1+G \gamma) B_{x}-G(\gamma-1)\left(\mathbf{r}^{\prime} \cdot \mathbf{B}\right) x^{\prime}\right] \\
a_{2}=C\left[(1+G \gamma) B_{y}-G(\gamma-1)\left(\mathbf{r}^{\prime} \cdot \mathbf{B}\right) y^{\prime}\right] \\
a_{3}=C\left[(1+G \gamma) B_{z}-G(\gamma-1)\left(\mathbf{r}^{\prime} \cdot \mathbf{B}\right)\right] \\
\omega^{2}=a_{1}^{2}+\left(a_{2}-1 / \rho\right)^{2}+a_{3}^{2}, \quad \mu=\omega L
\end{array},\right.
\end{gathered}
$$

$C=\frac{1}{B \rho}(1+x / \rho), L$ is the length of a machine element, and the field components are calculated midway through the element.

Sprot is called by Track after Orbit and calls in turn SpinMat that produces matrices for different type of machine elements, Sec. 4.18. In the case of elements that do not rotate spin, like DRIFT or MARKER Sprot returns immediately to Track.

### 4.18 routine SpinMat

Spin Matrices.
The routine has many entries, for different types of machine elements. In this routine the magnetic field at the particle transverse position and in the middle of the machine element, $\mathbf{r}_{W}$ is calculated and returned to Sprot.

- entry SpinMat:UnitSprot. Initialize a spin matrix as unitary.
- entry SpinMat:BendSprot. Spin rotation in a bend. If a bend is pure, i.e. has no gradient nor sextupole, its magnetic field is simply vertical and the entry returns with that component of the field. If the bend has multipoles, it returns to Sprot but subsequently multipole spin rotation is called.
- entry SpinMat:MultipoleSprot. Magnetic field at the particle', for quadrupoles, tilt quadrupoles and sextupoles.
- entry SpinMat: SolenoidSprot. In good approximation the magnetic field of a solenoid is purely longitudinal.
- entry SpinMat:SnakeSynthSprot. For each Synthetic Siberian Snake in the lattice, in sequence (Snake counter $S N K \_k$, with rotation angles given as input, Sec. 7.3.9, all coefficients for the spin matrix of Sprot are calculated and returned.
- entry SpinMat:SnakeTabSprot. For a snake represented by a table, coefficients for spin matrix are calculated from angles in the table and returned to Sprot
- entry SpinMat:SpinrSynthSprot. Spin rotators, treated as for synthetic snakes, with rotation angles given as input, Sec. 7.3.10
- entry SpinMat:SprotInKicker. Calculate and return spin matrix elements in kickers, horizontal and vertical kicks.

For each element the global variable spin rotation angle $\mu \equiv m u$ is calculated.

### 4.19 routine Helix

Orbit and spin matrix for a helix based on some analytical description.
Some Siberaian snakes and spin rotators are made with a sequence of helical dipoles [10]. Input parameters for a helix are the helix field, the helix pitch, the helicity sign (right- or left-handed), and the orientation of the field.with the vertical. Parameters are entered through the GUI, Sec. 7.3.11, for their description. A counter $H L X \_k$ numbers helices accordindg to their occurrence while visiting the accelerator.

Helix has two entries:

- entry Helix:HelixOrbit. E.Courant model [11]. Here the beam transfer matrix elements are calculated, then the routine returns to Orbit.
- entry Helix:HelixSprotAnalyt. M.Syphers model [12]. Here the spin matrix alements are calculated, the the routine returns to Sprot.


### 4.20 routine SpinGoodies

Some spin related quantities.
Entries are:

- entry SpinGoodies:SpinInit. Called from WorkParticle Initializes spin components, after two of them (out of three) have been read from input, Sec. 7.3.8, or from unit(12) or (13) associated with the population file.. Initializes One Turn Spin Matrix quantities for spin tune calculation.
- entry SpinGoodies:SpinAngles. Called from Track. Calculates and print on unit(33) (diagnostics file) the spin components and the anles of longitude and latitude of the spin vector along the orbit.
- entry SpinGoodies:SpinSum. Called from Track. For spin average over many particles, builds

$$
\sum_{n p} S(i), \quad \text { and } \quad \sum_{n p} S(i)^{2}, \quad i=1,3
$$

- entry SpinGoodies:SpinTuneDo, called from Track. Writes on unit(34) (diagnostics file) spin and spin cone data at the beginning of a turn and then collects and writes to (34) spin tune calculated from spin precession. Deprecated
- entry SpinGoodies:SpinAverage, called from Track. Calculates spin averaged by stroboscopic mean and variance over all particles at the end of each turn. Calculates and prints at the end of the run the angles of spin cone. If we want to center the input spin cone on the stable axis, we use
the input spin cone of beam on stable axis, the stroboscopic averaged spin components are written on the input population file, unit(13), next to the phase space coordinates ${ }^{7}$.
In this entry also the average spin tune from the eigenvalues of the One Turn Spin matrix are calculated and printed to stdout at the end of the run.
- entry SpinGoodies:SpinFileWrite. Deprecated
- entry SpinGoodies: OneTurnSpinMatInit, called from Track. Initialize one turn spin matrix at the beginning of each turn. If unit(44) (diagnostics) is alive the stable axis is also initialized.
- entry SpinGoodies:OneTurnSpinMatAtTurnEnd, called from Track. If unit(30) (diagnostics), is alive, Calculate Spin One Turn matrix trace and stable axis components at turn end
- entry SpinGoodies:SpinTune3. Called at turn end. Calculate spin One-Turn variables betweenn points of same phase space by averaging spin tune over turns (stroboscopic average). There is an approximation to calculate spin tune in one spin one-turn and re-normalize the One Turn Spin Matrix ${ }^{8}$.
- entry SpinGoodies:StoreSpinSum. Store spin and tune sums at marker \#1 for sequential average over particles. Write and read spin to unit(24), use unit(91) (disposable) as buffer. Write and read spin tune to unit(25), use unit(92) (disposable) as buffer.
- entry SpinGoodies:MultiPartAverage. Averages spin and tune values at marker $\# 1$ over all npart particles at run end. Writes results for spin average to unit(24) and spin tune average to unit(25).

[^6]
### 4.21 routine Bumps

Orbit Bumps
Orbit bumps are sometimes used on circular machines for special purposes. An example are the horizontal orbit bumps to offset the orbit in an helical Siberian Snake[13]. Orbit bumps must be represented by RayTransfer matrices for the orbital motion as well as by Spin matrices, because they generate a spin rotation. Bumps are called in sequence, with counter $B M P \_k$

Bumps has two entries

- entry Bumps:BumpOrbit. Bumps effect on orbit. Called from Orbit for the orbital motion. Uses bump table(7). Bumps may be horizontal and vertical. Angle kicks on $p_{x}$ or $p_{y}$, respectively.
- entry Bumps:BumpSprot. Bump effect on spin. Called from Sprot for the spin motion. Bumps may be horizontal and vertical. Spin matrix elements are calculated and returned.


### 4.22 routine Output

Prepares and prints out tracking output files.
spink can generate several output files, for orbit and spin coordinates at given location around the ring. Each location (here a MARKER) is tagged by the el_name of an element in the lattice. The list of print location is an input to the code.

- The entry Output:PrepareOutput is called at each one of the markers as they are found in the lattice during the element loop in Track and a record containing the variables at the marker is written for each marker (up to 9). In this entry, also the transverse Courant-Snyder invariants for the beam are calculated and recorded ${ }^{9}$

$$
\epsilon_{n}=\frac{\gamma_{T} u^{2}+2 \alpha_{T} u v+\beta_{T} v^{2}}{\sqrt{\beta \gamma}}, \quad \text { with } \quad u, v=x, x^{\prime}, \quad \text { or } \quad y, y^{\prime}
$$

with $\alpha_{T}, \beta_{T}, \gamma_{T}$ Twiss parameters, and $\beta, \gamma$ relativistic factors

- In the entry Output:WriteAtMarkers spin information can be written to a special file, unit(29).
- The entry Output:WriteOutput is called at give print intervals, Sec. 7.3.6, where output files, units(51),..(59) are written, one for each marker.
- Other files are written by the latter entry:
unit(40), spin rotation in the snakes,
unit(20), containing betatron tune and spin tune,
unit(30), containing stable spin axis components, calculated from the One Turn Spin Matrix, and the projection of the spin vector on that axis.

[^7]
### 4.23 routine RFDipole

Radio frequency dipoles
The formalism is due to Mei Bai [14]. An RF Dipole gives a RF kick to the beam and produces both an orbit bump and a spin rotation. Input parameters are the integrated magnetic field strength $B d l$, in [Tesla.meter] and the RFD tune $\nu_{m}$. The orbit kick and phase are given by

$$
K=\frac{B d l}{B \rho} \cos \phi+\text { phase }, \quad \phi=2 \pi \int \nu_{m} .
$$

The kick is applied to the beam, either in the horizontal or vertical direction, as

$$
p_{x}:=p_{x}-K, \quad \text { or } \quad p_{y}:=p_{y}-K
$$

The spin rotation angle is

$$
\mu=(1+G \gamma) K
$$

and the spin rotation matrices are

$$
\sigma_{H}=\left(\begin{array}{ccc}
\cos \mu & 0 & \sin \mu \\
0 & 1 & 0 \\
-\sin \mu & 0 & \cos \mu
\end{array}\right), \quad \sigma_{V}=\left(\begin{array}{ccc}
1 & 0 & 0 \\
0 & \cos \mu & \sin \mu \\
0 & -\sin \mu & \cos \mu
\end{array}\right)
$$

for the horizontal and vertical plane, respectively.
Each RF Dipole must appear in the $M A D$ description as elements with name RFDH or RFDV and type MARKER. Since $M A D$ doesn't have those element types, the routine TrackInit recognizes the name (el_name) and convert it into a keyword (el_keyw) RFDH or RFDV, Sec. 4.5 .

When Track find in the lattice a machine element with el_keyw $=$ RFD, the RFD action is modeled in two routine entries: RFDipole:RFDOrbit, that calculates and applies a kick to the orbit, and RFDipoleRFDSprot, that calculates and applies a kick to the spin.

Reference value for $B d l$ and $\nu_{m}$ and phase are given as input, Sec. 7.3.16. The arrays of value refer to the first, second, ... RFD found in the lattice. In the example, only the second RFD found is active.

RF Dipoles are activated by enabling table(3) and (4). for an horizontal field RFD, and for a vertical field RFD, respectively After a first row containing the number of columns- 1 to be read by TableRead, an RFD table contains, for each dipole
turn number, a multiplier for RFD_BdlMax, a multiplier for RFD_tune_m0

An example of table for two RFDV's (RFDV.tab, for the example above) is the following

|  |  | 1.st dipole <br> 6 |  |  | turn | Bdl |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- |
|  |  | tune | phase | Bdl2.nd dipole <br>  <br>  <br> 0 | 0.000000 | 1.000000 |
| tune | 90. | $\ldots$ |  |  |  |  |
| 300 | 0.000000 | 1.000000 | 90. | $\ldots$ |  |  |
| 3300 | 1.000000 | 1.000000 | 90. | $\ldots$ |  |  |
| 9300 | 1.000000 | 1.000000 | 90. | $\ldots$ |  |  |
| 12300 | 0.000000 | 1. | 90. | $\ldots$ |  |  |
| 13300 | 0.000000 | 1. | 90. | $\ldots$ |  |  |
| 20000 | 0.000000 | 1. | 90. | $\ldots$ |  |  |

In the example, the kick is linearly built between turns 300 and 3300 , stays constant between 3300 and 9300 , and linearly decays to zero between 9300 and 12300 . Any shape is possible, since TableInterp interpolates between values in the table.

If activated, two files are written, for RFDH, unit(35), and for RFDV, unit(36), containing columns of

$$
n t \quad B d l \quad \nu_{m} \quad \phi \quad \text { Kick }
$$

### 4.24 routine SpinFlipper

This is for a Radio Frequency spin flipper. A Rf field is applied to the beam and generates a rotation of spin around an axis in the horizontal x-z plane, or the axis of a RF solenoid. The frequency may sweep through the resonant condition or, in the case of the solenoid may remain constant. The temporal shape of the applied field can be prescribed my means of table(13).

SpinFlipper has one entry

- SpinFlipper:SFSprot, called from Sprot. Specify as input, Sec. 7.3.14, at what turn to start action, the value of the integrated field $B d l$, the spin flipper frequency, the frequency variation per turn, and how the field is applied, whether rotating or transverse, in $x$ or $y$. or oscillating longitudinally (solenoid).

SpinFlipper calculates and returns to Sprot the spin matrix elements.

### 4.25 routine Tunes

Calculate tune and detune from betatron oscillations zero counts.
It is one of the methods to calculate betatron tunes. A better one, turn-by-turn, is to calculate betatron tunes (fractional part of) from the eigenvalue of the orbital one turn metrix.

Tunes has 5 entries:

- entry Tunes:TuneInit. Initialize tune average over particles.
- entry Tunes:CountBetaZero. Count orbit zero crossings for tune evaluation, or comparing a tune oscillation wavelengt with the circumferende of the ring.
- entry Tunes:BetaTuneDo. Calculate betatron tune for each particle at end of run.
- entry Tunes:TuneAverage. Average tune for many particles
- entry Tunes:Detune, called by TrackInit, Sec. 4.5. Padding quadrupoles with two thin quads to obtain detuning. Every quadrupole transport matrix is transformed as follows

$$
\begin{aligned}
\mathbf{R}_{x} & =\left(\begin{array}{cc}
R_{1,1}-\frac{1}{4} R_{1,2} \delta q & R_{1,2} \\
R_{2,1}-R_{1,1} \delta q+\frac{1}{4} R_{1,2} \delta q^{2} & R_{2,2}-\frac{1}{2} R_{1,2} \delta q
\end{array}\right) \\
\mathbf{R}_{y} & =\left(\begin{array}{cc}
R_{3,3}+\frac{1}{2} R_{3,4} \delta q & R_{3,4} \\
R_{4,3}+R_{3,3} \delta q+\frac{1}{4} R_{3,4} \delta q^{2} & R_{4,4}+\frac{1}{2} R_{3,4} \delta q
\end{array}\right) .
\end{aligned}
$$

with $\delta q$ a (small) detuning parameter given in input, Sec. 7.3.6.

### 4.26 routine Coup

After the formalism of Edwards and Teng [15] [16]
If the boolean flag couple $=$.true. after the introduction of a few matrices based on the One Turn Matrix (OTM), we end up building a $4 \times 4$ matrix EDR that is being used in WorkParticle to transform the coupled input phase space coordinates $\mathbf{u}$ to $\mathbf{r}$

$$
\mathbf{r}=\mathbf{E D R} \cdot \mathbf{u}
$$

The latter equation is in WorkParticle

## 5 Diagnostics routines

## 5.1 routine Diagnostics

Various debugging diagnostics.
Diagnostics has three entries:

- entry Diagnostics:PrintAtElement. Print at element for turns between nt_pr_0 and nt_pr_1, given as input, Sec. 7.3.6. If unit(99) is open many relevant dynamic quantities, including phase space particle position $\vec{r}$ and spin $\vec{S}$, are printed element by element for a limited number of turns (1) before Orbit, (2) before Sprot, (3) after Sprot.
- entry Diagnostics:PrintAllMatrices. Print all orbit matrices at a certain energy on unit(42). Deprecated.
- entry Diagnostics:PrintStatusAtTrackEnd. Write phase space and spin at end of track on unit(43).


## 5.2 routine Sequence

Track and profile call sequence and spink flow.
Activate at compilation time with the directive $S E Q=$ true.

## 6 Additional routines

## 6.1 routine FroissartStora

Polarization loss crossing an isolated spin resonance is well modeled by the Froissart-Stora formula [17]

$$
\begin{equation*}
P_{2}=P_{1}\left[\exp -\frac{\pi \epsilon_{k}^{2}}{2 \alpha}-1\right] \tag{12}
\end{equation*}
$$

where $P_{2}$ is the polarization after the resonance and $P_{1}$ is the pol. before the resonance, $\epsilon$ is the resonance strenght and $\alpha$ the speed of resonance crossing.. Conversely, the formula can be used to calculate the resonance strenght by knowing $P_{2} / P_{1}$.

This is what is done in this routine. We specify in input, Sec. 7.3.13, at what energy FroSto_Ggam we want to do this, and after how many turn we want to look at $P_{2}$ and the routine calculates the resonance $\epsilon_{k}$ and writes it to stdout.

## 6.2 routine BeamBeam

In a collision between two beams of polarized hadrons, the particles in both beams suffer a deflection and also the polarization may change due to the motion of a polarized particle in the field of the other beam. In BeamBeam these effects are simulated by studying the kicks on orbit and spin of a particle in the field generated by a distribution, assumed Gaussian, of the other counterstreaming beam [18]. It is the so-called 'weak-strong beam-beam scenario'. Collision happen at a location specified in input, Sec. 7.3.15, coincident with one of the lattice elements, e.g. a marker. We specify also at what energy we want BeamBeam to occur and the number of particles in the beam.

BeamBeam contains three entries:

- entry BeamBeam:BBCoast. Slow down acceleration. Gradually turn on beam-beam. We want to see the effect on a coasting beam.
- entry BeamBeam:BBOrbit. Calculate Beam-beam orbit matrix -angle kick.
- entry BeamBeam:BBSprot. Beam-beam spin matrix. Calculate Beam-beam spin matrix.


## 6.3 routine OrbitFFT

Betatron tune an orbit harmonics content is calculated by FFT. First, sample orbit at equal intervals, then calculates tune using routines four1.fft and twofft.f from 'Numerical Recipes in Fortran' [19] that are included..

## 6.4 routine MPIOutput

Output for parallel computing with the MPI (Message Passing Interface) library.
To use MPI the boolean at compilation time is USEMPI $=$ true. Each particle in the simulation is assigned a rank number my_rank, is individually tracked, and the results are averaged on the fly, at variance with sequential averaging were the averages are completed at run end. It is a trivial parallelization ${ }^{10}$. If there is only one particle and one process, as for sequential average, the rank of the particle is my_rank $=0$.

After a first block where spin averages are done, the routine presents two entries

- entry MPIOutput:MPILogRec. Write to Internal records initial and final parameters for each macro.
- entry MPIOutput:MPILogOut. Retrieve and write to $\log$ file 7 initial and final parameters for each macro


## 6.5 routine Clock

Set clock to measure run time for the first particle in a run. Enable in input, with cclock=.true. Sec. 7.3.5.

## 7 Pre Processors

## 7.1 mad8

At present use mad8. Work is in progress to accept input from madX.
Run with the command
$=>$ mad8c runname. mad
The driver file runname.mad will produce the following

- A runname.twiss file containing the Twiss functions calculated at the physical end of each machine component and the distorted closed orbit COD, generated by the mad8 command TWISS, tape='runname.twiss'
- A runname.echo file ${ }^{11}$, that contains the first and second order orbit transfer maps, generated by the command
SELECT, FLAG=first [second], RANGE= $\# S / \# E$
- A runname.madout file, containing tables of misalignment and field errors in the machine elements (if any), generated with the commands PRINT, RANGE= $\# S / \# E$
SELECT, ERROR ; EPRINT, RANGE $=\# S \# E$
runname.twiss, runname.echo runname.madout constitute the input to mad_read.

[^8]
## 7.2 mad_read

mad_read reads and combines the three $M A D$ output files to create a single file runname.sy that contains Twiss functions, transfer maps, COD and information about machine component misalignment or field errors. If there are no errors, only the first two files are needed. All namefiles must be included in the driver file for mad_read.in.

The program is run with
$=>$ mad_read $<$ mad_read.in
The symplecticity of the matrices translated from $M A D$ to runname.sy is improved with the use of an iterative algorithm and routine due to F.Neri.

The arrangement of runname.sy is as follows:
First record contains:

| 1 | Mad vers., | track order $[=1$ or 2$]$ | errors $[=0$ or 1$]$ | $\mathrm{n} \#$ of elementss $[=$ jel $]$ |
| :--- | :--- | :--- | :--- | :--- |

If track order $=1$, and no errors, for each machine element a block of 12 lines follows:

| 2 | el. \# | name | keyword | length |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :--- |
| 3 | angle | K1 | K2 | tilt | Hkick | Vkick |
| 4 | orbit -4 values (deprecated) |  |  |  |  |  |
| 5 | $\alpha_{x}$ | $\beta_{x}$ | $m u_{x}$ | $D_{x}$ | $D_{x}^{\prime}$ |  |
| 6 | $\alpha_{y}$ | $\beta_{y}$ | $m u_{y}$ | $D_{y}$ | $D_{y}^{\prime}$ |  |
| 7 | COD -4 values |  |  |  |  |  |
| $8-13$ | $6 \times$ transfer matrix |  |  |  |  | R_mat coord. |

If track order $=2$, and no errors, 36 lines follow for each block, containing the elements of T_mat:

$$
\begin{array}{|l|l|}
\hline 14-50 & 6 \times 6 \times 6 \text { second order transfer map elements } \\
\hline
\end{array}
$$

If there are errors, after line 3 there are two lines containing position and angle errors, and field errors (see $M A D$ for the description)

| 3 a | $\delta x$ | $\delta y$ | $\delta z$ | $\delta \phi[$ pitch] | $\delta \theta$ [yawl] | $\delta \psi[\mathrm{roll}]$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 3b | $\delta B$ | $\delta K 1$ | $\delta K 2$ | $\delta K 3$ |  |  |

### 7.3 GUI

SPINK is launched through a motif-window based GUI. The GUI is a convenient interface to set values for the variables in the program, and to minimize errors of transcription

At the prompt, the GUI is started with the command
$=>$ uspink [runname]
where uspink is a C-shell Unix script and runname is an arbitrary name for the run. If a configuration file runname.conf exists because it has been previously created, a menu window with 3 buttons shows up, as in Fig. 2. If the conf file does not exist you will be warned.

To edit the configuration file, click on Setup: a windows containing a list of the modules of SPINK will appear, Figs. 3, for 18 modules.

On this, and on each of the other module editing windows there is a help button ? that commentss the meaning of each variable, string or Boolean parameter. The variable names are the same used in the actual code. When finished with editing, go back to the start GUI window of Fig. 2 (still open on the screen) and select Store.

There is also another button, Load, that allows the user to load into the GUI the values of another configuration file previously created.

When finished with the GUI, close the window with the upper right cross X . The script will convert the conf file to a namelist input runname_spink.inand run SPINK.

One can also run directly $S P I N K$, if you have already the spink.in file as follows $=>$ spink < spink.in

We will now visit all modules (GUI windows) that will be opened by clicking on edit, with detailed explanation of the meaning of all variables

### 7.3.1 GUI: input files

The GUI windows for input files is shown in Fig. 4, left. Some filenames in the example have a \# as a first character. These files will not be opened. An array of values is indicated by a [...]

File names are arbitrary. Files whose first character is \# are ignored.


### 7.3.2 GUI: output files

The GUI windows for output files is shown in Fig. 4, right. Some filenames in the example have a \# as a first character. These files will not be opened. An array of values is indicated by a [...]

| file | example | description ? |
| :---: | :---: | :---: |
| ffile(7) | spink.log | Log file. <br> It is a copy of the screen plus some more info on the run For many particles run, it contains only the 1.st particle |
| ffile(8) | orbit.dat | Detailed orbit of the first particle in the simulation |
| ffile(15) | \#COD_short.dat | Closed Distorted Orbit. <br> Contains:elem_no, s, x_COD, y_COD |
| ffile(20) | \#tunes.dat | File contains $G \gamma, \gamma$, beta-tune(fraction), spin-tune |
| ffile(21) | \#fine_orbit.dat | Orbit at each location in the machine. Contains s(extended), x, y |
| ffile(23) | \#COD.dat | closed distorted orbit from machine descriptor el\#, s, x_COD, px_COD, y_COD, py_COD |
| ffile(24) | \#spin_ave.dat | spin average over many particles <br> nt, $G \gamma,\langle S x\rangle,\langle S y\rangle,\langle S z\rangle,\langle S x\rangle-\sigma . \ldots\langle S x\rangle+\sigma, \ldots$ |
| ffile(25) ffile(28) | \#tune_ave.dat \#phs.dat | tunes average over many particles <br> nt, $G \gamma,<Q x>,<Q y>,<Q z>,<Q x>-\sigma, \ldots<Q x>+\sigma, \ldots$ <br> phase space output |
| [ffile(31)] | RFD_spin1.dat RFD_spin2.dat | array of strings outputs for radio frequency dipole. <br> presently up to 2 files units 31 and 32 for 1.st and 2.nd RFD, respectively. Contains nt ,Bdl, tune, phase, kick |
| [ffile(35)] | \#RFD_orb1.dat RFD_orb2.dat .... | array of strings orbit outputs for radio frequency dipole presently up to 2 files: units 35 and 36 for 1.st and 2.nd RFD, respectively. Contains: nt, Bdl, tune, phase, kick |
| [ffile(51)] | NE260_INIT.dat | keep units 37 and 38 for extension array of strings output at markers one presently for up to six markers units $51,52,53,54,56,57$, respectively nt, $G \gamma, S x, S y, S z, x, p x, y, p y, f r a c(Q), S-t u n e$ |
| [ffile(71)] | \#process_1.dat ... | array of strings units 71 and up taken for MPI work |
| [ffile(81)] | \#process_1.dat ... | array of strings units 81 and up taken for MPI work |

### 7.3.3 GUI: diagnostics files

The GUI windows for diagnostics files is shown in Fig. 5, left. Some filenames in the example have a \# as a first character. These files will not be opened. An array of values is indicated by a [...]

| file | example | description ? |
| :--- | :--- | :--- |
| ffile(9) | \#sequence.dat | sequence of subroutines visited |
|  |  | For diagnostics |
| ffile(29) | \#spin_at_markers.dat | spin and spin_norm at markers |
| ffile(30) | \#spin_axis.dat | evolution of spin axis and projection <br> of spin on SA |
|  |  | Gr, Spin_axis, spin•spin_axis <br> details of kicks. For each kick: |
| ffile(33) | \#kick.dat | KickH: elkick,mu=',kickh,mu |
|  |  | KickV: elkick,mu=',kickv,mu |
|  |  | details of: spino, theta_cone, phi_cone |
| ffile(34) | \#fine_spin.dat | details of: SP_Ggam,SP_gam,mu_snk_out |
| ffile(40) | \#mu_snk_out.dat | One Turn Matrix |
| ffile(41) | \#OTMat.dat | all orbit matrices |
| ffile(42) | \#Matrices.dat | el\#, Rmat |
|  |  | Print Status At Track End |
| ffile(43) | \#StatAtTrackEnd.dat | Spin Matrix and One Turn Spin matrix |
| ffile(44) | \#SpinMatrix.dat | auxiliary for spin average over particles |
| ffile(91) | \#spin_ave_sum.dat | it accumulates spin sum and spin ${ }^{2}$ sum |
|  |  | for all turns |
| ffile(92) | \#tune_ave_aux.dat | auxiliary for tune average |
| ffile(94) | \#r_mat.dat | print all orbit matrices |
| ffile(98) | \#FFT_aux.dat | writes: |
|  | sficontinuous), x, y, field, element_name |  |
| ffile(99) | diagnostics.dat | writes: |
|  |  | s(continuous), x, y, field, element_name |

### 7.3.4 GUI: tables

The GUI windows for tables is shown in Fig. 6, left. Some table names in the example have a \# as a first character. These tables will not be used. An array of values is indicated by a [...]

| file | example | description ? |
| :--- | :--- | :--- |
| table(1) | \#gamma.tab | Table of $\gamma$ vs. turn no. Acceleration ramp |
| table(2) | \#detune.tab | Table of detune parameter vs. turn no. |
| table(3) | RFDH.tab | Horizontal kick RF dipole table. Up to 6 RFDH <br> table contains |
|  |  | nturn, for first RFD:v1 v2, for second RFD:v1,v2,... <br> table(4) |
|  | \#RFDV.tab | Vertical kick RF dipole table. Up to 6 RFDV |
| table(5) | \#SF.tab | nturn, for first RFD:v1 v2, for second RFD:v1,v2,... |
| table(6) | \#Snake.tab | Spin Flipper table vs. turn no. |
|  |  | Snake table |
| table(7) | \#Movable_Snake_Bump.tab | Snake $\mu, \phi($ Snakis), $\theta$ (axis) vs. turn no. |
| table(8) | \#gamma_step_lattice.tab table vs. turn no. |  |
|  |  | To update the machine descriptor during acceleration |
| table(9) | \#quadK1.tab | contains the name of lattice files vs. $\gamma$ |
| table(10) | \#snake_1.tab | Table of quad K1 values vs. $\gamma$ |
| table(11) | \#snake_1.tab | Table of dipole K1 values vs. $\gamma$ |
| table(12) | \#snake_2.tab | Table of snake_1 matrix elements and mu vs. $\gamma$ |
| table(13) | \#SolSpinFlipper.tab | Table of snake_2 matrix elements and mu vs. $\gamma$ |
|  |  | Table of solenoidal Spin Flipper Bdl vs. turn no. |

### 7.3.5 GUI: flags

The GUI windows for logical flags is shown in Fig. 6, right. Some table names in the example have a \# as a first character. These tables will not be used. An array of values is indicated by a [...]

| variables | example | description $?$ |
| :--- | :--- | :--- |
| Flags | on | Activate flags |
| no_spin | off | No spin track, only orbit |
| show_markers | off | Results at given markers in the lattice <br> cclock |
| Ereno | on | Show clock run time <br> Force renormalization of the Courant-Snyder invariant <br> to control emittance in acceleration |
| couple | off | Edwards-Teng coupled injection <br> Automated calculation of stable spin direction <br> stable_spin |
| off | to inject spin on the stable axis via <br> stroboscopic average <br> Diagnostics dump at Sprot <br> use with small number of turns: |  |
| Sprot_dmp | off | it dumps at all machine elements <br> Field correction in dipoles |
| field_correction | off |  |

### 7.3.6 GUI: dynamics

The GUI windows for dynamic variable setting for tracking is shown in Fig. 7, left. Some table names in the example have a \# as a first character. These tables will not be used. An array of values is indicated by a [...]

| variable | type | example | description ? |
| :---: | :---: | :---: | :---: |
| Dyna | boolean | on | This module must be on |
| nturn | integer | 100,000 | Total number of turns (1) |
| dnt_pr | integer | 117 | Turn print step |
| nt_pr_0 | integer | 1 | Turn where print starts |
| nt_pr_1 | integer | 1,000,000 | Turn where prin stops |
| SP_Ggam0 | double | 250. | Sync Particle start $G \gamma$ |
| SP_Ggamf | double | 270. | Sync Particle end $G \gamma$ (1) |
| SP_dgam0 | double | $3.2 \mathrm{E}-005$ | Rate of increase of $\gamma$ per turn if acceleration table is not used (2) |
| dq | double | 0. | Quadrupole detuning factor(3) |
| err_corr | double | 1. | Correction factor for lattice errors $=0$. means: cancel all magnet errors |
| kick_corr | double | 1. | Corr. factor for kicks $=0$. means: cancel all kicks |
| cod_corr | double | 1. | Corr. factor for Closed Distorted Orbit $=0$. means: cancel distortions of orbit |
| particle | sring | p | At present only protons (p) and deuterons (d) |
| [harm(1)] | double array | 2809 .. | Harmonic in each RF cavity |
| [volt(1)] | double array | 0.0050 .001 | Voltage in each RF cavity [GV] |

Notes: (1) track stop when nturn or SP_Ggamf is reached
(2) Table(2), Sec. 4.4
(3) Sec. 4.5

### 7.3.7 GUI: Orbit FFT

GUI window in Fig.7, right.

| variable | type | example | description ${ }^{\text {? }}$ |  |
| :---: | :---: | :---: | :---: | :---: |
| OrbFFT | boolean | off | FFT of the orbit |  |
| FFT_ntOrb1 | integer | 200 | Turn at which start FFT |  |
| FFT_ntOrb2 | integer | 225 | Turn at which end FFT |  |
| FFT_nOrb | integer | 8192 | Turns for FFT |  |
| FFT_OrbDz | double | 1. | Longitudinal step for orbit FFT |  |
| FFT_Orbq1 | double | 2.7 | Start frequency range |  |
| FFT_Orbq2 | double | 3.2 | End frequency range |  |

### 7.3.8 GUI: Population

GUI window in Fig.8, left.

| variable | type | example | description $?$ |
| :--- | :--- | :--- | :--- |
| Pop | boolean | on | Initial distribution of particles |
| npart | integer | 1 | No. of particles in the simulation |
| popseed | integer | 1936 | Seed for random extraction of population |
| deltaph | double | 0. | $\Delta \phi$ longit. coordinate |
| deltap | double | 0. | $\Delta E / p c$ longit. coordinate |
| $[\operatorname{emit}(1)]$ | double array | $6.0 \mathrm{E}-0066.0 \mathrm{E}-006$ | Emittance (H and V) [m-rad] |
| $[\operatorname{angle}(1)]$ | double array | 0.0. | Angles, if particles are generated on an ellipse(1) |
| $[$ spino(1)] | double array | 0.1 .0. | Initial spin coordinates for one particle. |
|  |  |  | If a pop file is used, each particle may have |
|  |  | an individual set of spin coordinates |  |
|  |  | It is enough to give two coords., |  |
|  |  |  |  |
|  |  |  |  |

Notes: (1) Sec. 4.9

### 7.3.9 GUI: Synthetic Snakes

GUI window in Fig.8, right.

| variable | type | example | description |  |
| :---: | :---: | :---: | :---: | :---: |
| Synth_Snakes | boolean | on | Setting of synthetic snakes a synth snake is a thin snake. it is represented by a marker called SNAKE in the MAD lattice descriptor. up to 6 snakes Spin rotation angle in each snake [deg] Snake axis angle in the horizontal plane [deg] Snake axis angle of elevation [deg] |  |
| [snake_mu(1)] | double array | 180. 180. |  |  |
| [snake_ax_phi(1)] | double array | 135. 45. |  |  |
| [snake_ax_theta(1)] | double array | 0. 0 . |  |  |

### 7.3.10 GUI: Synthetic Spin Rotators

GUI window in Fig.9, left.

| variable | type | example | description |
| :---: | :---: | :---: | :---: |
| Synth_SpinRot | boolean | off | Setting of synthetic spin rotator. a synth snake is a thin snake. it is represented by a marker called SPINR in the MAD lattice descriptor. normally, spin rotators go in pairs up to 2 spin rotator pairs. |
| [spinr_mu(1)] [spinr_ax_phi(1)] | double array double array | $\begin{aligned} & \text { 90. -90. 90. -90. } \\ & 4^{*} 90 . \end{aligned}$ | Spin rotation angle in each rotator [deg] <br> Spin rotator axis angle <br> in the horizontal plane [deg] |
| $\begin{aligned} & {[\text { spinr_ax_theta }(1)]} \\ & {[\text { spinr_bend }(1)]} \end{aligned}$ | double array double array | $\begin{aligned} & 4^{*} 0 \\ & 0.212^{*}-0.210 .21 \end{aligned}$ | Spin rotator angle of elevation [deg] Arc bend between rotators and the encompassed center point spin rotation is around an axis of angle given by $\phi+$ bend $\cdot G \gamma$ |

### 7.3.11 GUI: Helix

GUI window in Fig.9, right.


### 7.3.12 GUI: Stable Axis

GUI window in Fig.10, left.

| variable | type | example | description |
| :--- | :--- | :--- | :--- | ? .

### 7.3.13 GUI: Froissart-Stora

GUI window in Fig.10, right.

| variable | type | example | description $[?$ |
| :--- | :--- | :--- | :--- | :--- |
| Froissart-Stora | boolean | off | Froissart-Stora calculation of an isolated <br> Spin resonance strength from spin tracking |
| FroSto_Ggam | double | 3.82 E 002 | $G \gamma$ value where to calculate Froissart-Stora |

### 7.3.14 GUI: Spin Flipper

GUI window in Fig.11, left.

| variable | type | example | description $?$ |
| :--- | :--- | :--- | :--- |
| SFlipper | boolean | off | Activate Spin Flipper |
| SF_nt | integer | 5000 | How many turns SF is active |
| SF_Bdl | double | $2.5 \mathrm{E}-002$ | Integrated field in Spin Flipper |
| SF_freq0 | double | 3.8 E 004 | Start frequency |
| SF_dfreq | double | 1E-001 | Frequency variation per turn |
| SF_rot | string | x | Direction of field, "x", "y", "r", "s" <br> transverse, rotating, solenoidal |

### 7.3.15 GUI: BeamBeam

GUI window in Fig. 11.

| variable | type | example | description $[?$ |
| :--- | :--- | :--- | :--- |
| BBeam | boolean | off | Activate beam-beam |
| BB_Npp0 | double | 2E011 | No. of particles in beam |
| BB_Ggam0 | double | 4.7 E 002 | Start energy |
| BB_Ggam | double | 4.8 E 002 | Stop energy |
| BB_loc | string | CLOCK8 | Collision location name in the lattice |

### 7.3.16 GUI: RF Dipole

GUI window in Fig.12, left.

| variable | type | example | description |  |
| :---: | :---: | :---: | :---: | :---: |
| RFD | boolean | off | Activate radio frequency dipole one needs tables. up to six |  |
| [RFDH_BdlMax(1)] | double array | 0.020 .02 | Max Bdl in each horizontal RFD The 2.nd, 4th.. column in Table(3) will be multiplied by these values |  |
| [RFDH_Tune0(1)] | double array | 0.490 .49 | Tune in each horizontal RFD the $3 . n d, 5$ th.. column in Table(3) will be multiplied by these values |  |
| [RFDH_Dphase(1)] | double array | 0. -90. | Phase shift in each horizontal RFD [deg] |  |
| [RFDV_BdlMax(1)] | double array | 0. 0. 0. 0. | Max Bdl in ea | ach vertical RFD |
| [RFDV_Tune0(1)] | double array | 0. 0. 0. 0. | Tune in each | vertical RFD |
| [RFDV_Dphase(1)] | double array | 0.0.0.0. | Phase shift in | each vertical RFD [deg] |

### 7.3.17 GUI: Cycle

GUI window in Fig.12, right.

| variable | type | example | description $?$ |
| :--- | :--- | :--- | :--- | :--- |
| Cycle | boolean | on | Activate cycle and diagnostics |
| gam_diag | double | 1E002 | Value of gamma for diagnostics |
| $[$ marker(1)] | string array | INITIAL CLOCK6 | List of lattice locations where to record data |
| $[$ second(1)] | string array | SEXT SNAKE | Machine elements to be tracked to second order <br> options: el_names, NONE, ALL |

### 7.4 Compilation and Run

For portability, SPINK has been written in Fortran77. The code makes use of include files that contain Global and Local variables, Common and Namelist definitions, so it is very easy to add and subtract variables.

The Spink source is being distributed through a directory that contains also a makefile for compilation. The code has been run on different Unix platforms, but it is advisable to recompile it when it is imported to a new computer.

The code is write protected through RCS, so a user who would like to modify or add to it should first check out a routine with the command
$=>c o-l$ routinename.F
and after the file has been modified, check it in again with
$=>c i-u$ routinename. $F$
For sequential run of the code, the compilation is done (in Linux) with the gnu compiler $g^{77}$
and for parallel code with the compiler and directive
mpif777-DUSEMPI
Of course one can use other compilers in different machines.
To compile, do
$=>[$ make cleanall $]$, for the first time, then $=>$ make
To run the code through the GUI (recommended), invoke $=>$ uspink runname

To run the code directly $=>$ spink $<$ spink.in
where spink.in is the driver file containing namelist blocks.

## 8 Suggested improvements

spink is still under development and probably will always stay in this state. Immediate and not-so immediate improvements are

1. Since madX is currently supporetd at CERN, while mad8 is not any more, modifying mad_read to work with the former lattice generator seems a clear choice.
2. The present code works with thin spin rotation elements. This approximation seems well adequate to deal with large storage rings, like RHIC. For smaller machines thick spin elements are preferable. Already for tracking in the AGS we found that subdividing the bends in a few parts give results more in accord with experimental data.
3. Dynamic representation of the lattice. This is already implemented in the code in two different ways as explained in this manual. However, a real smooth implementation using tables is not yet working as we would like.
4. More work is needed for a complete characterization of the spin stable axis, stroboscopic averaging, and other spin goodies.
5. Tracking other particles and other spin states. Tracking vector and tensor polarization will require dealing with spin matrices with more dimensions.
6. Spin motion in the self field of intense beams may become an important issue for high luminosities colliders. This problem is addressed in spink in the BeamBeam routine, but a real study will require a large effort. For multi particle intense beam dynamics simulation an effective non-trivial parallelization is a must.

## References

[1] A.U.Luccio: Spin Tracking in RHIC-Code SPINK. In: Y.Onel, N.Paver, A.Penzo, editor: Proc. Adriatico Research Conf on Trends in Collider Spin Physics. Trieste, Italy, 12/5-8, 1995. World Scientific, p. 235.
[2] H.Grote and F.Ch.Iselin: The MAD program, Vers.8.19. Technical Report CERN/SL/9013, European Organization for Nuclear Research, Geneva, CH, 1996.
[3] D.J.Jackson: Classical Electrodynamics. Wiley, New York, 1975. Second edition.
[4] S.Y.Lee: Spin Dynamics and Snakes in Synchrotrons. World Scientific. Singapore, 1997.
[5] A.U.Luccio: Angles from Spin Matrices. Technical Report AGS/RHIC/SN No.03, Brookhaven National Laboratory. Upton, NY, October 81996.
[6] Ya.S.Derbenev and A.M.Kondratenko. Sov. Physics. Doklady, 20:562, 1976.
[7] W.Gropp, E.Lusk, A.Skjellum: Using MPI. MIT Press, 1998.
[8] Iselin, F.Christoph: The MAD Program. Physical Methods Manual. Technical Report CERN/SL/92(AP), European Organization for Nuclear Research, Geneva, CH, Geneva, Switzerland, September 151994.
[9] A.U.Luccio: Spin Rotation Matrices for Spin Tracking. Technical Report BNL-62371, Brookhaven National Laboratory. Upton, NY, Oct. 301995.
[10] V.I.Ptitsyn and Yu.M.Shatunov: Helical Siberian Snakes. In: Proc Third Workshop on Siberian Snakes and Spin Rotators BNL-52453, Sept.12-13 1994.
[11] E.D.Courant. Technical Report AGS/RHIC/SN 004, Brookhaven National Laboratory. Upton, NY, Upton, NY, Nov 81994.
[12] M.J.Syphers. Technical Report AGS/RHIC/SN 020, Brookhaven National Laboratory. Upton, NY, Upton, NY, February 1996.
[13] A.U.Luccio, R.Gupta, W.MacKay and T.Roser: Cold AGS Snake Optimization by Modeling. Technical Report C-A/AP/128, Brookhaven National Laboratory. Upton, NY, December 2003.
[14] M.Bai: Overcoming Intrinsic Spin Resonances Using an RF Dipole. PhD thesis, Indiana University, 1999.
[15] D.A.Edwards and L.C.Teng: Parametrization of Linear Coupled Motion. In: IEEE Trans. Nucl. Sci. 20, 885, 1973.
[16] T.Roser: Multiturn Injection with Coupling. Technical Report BNL AGS/AD/Tech. Note 354, November 71991.
[17] M.Froissart and R.Stora. Nuclear Inst. and Meth., (7):297, 1960.
[18] A.Luccio, M.Syphers: Effects of beam-beam interaction on spin motion. Technical Report Spin Note AGS/RHIC/SN No.068, Brookhaven National Laboratory. Upton, NY, 1998.
[19] W.H.Press, S.A.Teukolsky, W.T.Vetterling and B.P.Flannery: Numerical Recipes in Fortran. Cambridge University Press, 1992. Second edition.


Figure 2: SPINK driving menu


Figure 3: SPINK modules


Figure 4: Left: input files, Right: output files. If a file name has a $\#$ as a first character, the file will be ignored. The opening of some file is equivalent to a flag to do something


Figure 5: Left: diagnostics files. Right: available files. If a file name has a $\#$ as a first character, the file will be ignored. The opening of some file is equivalent to a flag to do something


Figure 6: Left: Tables. If a table name starts with $\#$, the table will be ignored. The opening of some table is equivalent to a flag to do something.
Right: Logical flags


Figure 7: Left: Dynamic variables for tracking. Right: OrbitFFT variables


Figure 8: Left: Particle distribution variables. Right: Synthetic snakes parameters


Figure 9: Left: Synthetic snakes parameters. Right: Helix parameters


Figure 10: Left: Parameters for stable axis calculation. Right: Parameters for Froissart-Stora calculation


Figure 11: Left: Spin flippers parameters. Right: BeamBeam parameters


Figure 12: Left: Radio frequency dipole parameters. Right: Cycle parameters


[^0]:    Notice: This technical note has been authored by employees of Brookhaven Science Associates, LLC under Contract No.DE-AC02-98CH10886 with the U.S. Department of Energy. The publisher by accepting the technical note for publication acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this technical note, or allow others to do so, for United States Government purposes.

[^1]:    *Work performed under the auspices of the U.S.Department of Energy.

[^2]:    ${ }^{1}$ thanks to Len Slatest of BNL

[^3]:    ${ }^{2}$ After Giuliano Franchetti

[^4]:    ${ }^{3}$ There is an utility, doom of $m a d X$ that automatically transfer measured magnet errors from a spreadsheet to $M A D$. This is an additional reason to migrate from $\operatorname{Mad} 8$ to $\operatorname{madX}$, Sec. 8
    ${ }^{4}$ Element rotation is not yet implemented in spink

[^5]:    ${ }^{5}$ This kick produces a change in the amplitude of the transverse oscillation of the beam, so that the Courant-Snyder emittance of the beam changes by a factor $1 / \sqrt{\beta \gamma}$ and the normalized emittance stays constant

[^6]:    ${ }^{7}$ this file will be used in a next run
    ${ }^{8}$ An exact one-turn for spin is when the particle is back in phase space where it started. It may take many accelerator turns because of betatron motion

[^7]:    ${ }^{9}$ These quantities must remain constant if the tracking is corret, or equivalently the emittances must decrease as $(\beta \gamma)^{-1 / 2}$

[^8]:    ${ }^{10} \mathrm{~A}$ better use of a parallel computer is in the parallelization of some algorithm or, in particular in the case of high intensity beams, in the study of space charge effects on beam dynamics and polarization
    ${ }^{11}$ The echo file was originally meant for diagnostics. Luckily it can contain the transport maps

