

EUROPEAN LABORATORY FOR PARTICLE PHYSICS
CERN – BE Department

CERN/BE/2010-??

PTC
User's Reference Manual

E. Forest and F. Schmidt

Abstract

Geneva, Switzerland
October 15, 2010

Contents

1	Introduction	1
2	Type INTERNAL_STATE of PTC	2
2.1	Definition of a State	2
2.2	Eternal States	2
2.3	State Default	2
2.4	State Time	3
2.5	State Totalpath	3
2.6	State Nocavity	3
2.7	State Fringe	3
2.8	State Spin	3
2.9	States controlling Radiation	3
2.9.1	State Radiation	3
2.9.2	State Envelope	3
2.9.3	State Stochastic	3
2.10	State Modulation	3
2.11	States Affectiong Map Calculations	3
2.11.1	State Only_4d	3
2.11.2	State Delta	4
3	Geometry Routines of PTC	5
3.1	Survey Routines	5
3.2	Over Fibres	5
3.3	Over Integration_node: FILL_SURVEY_DATA_IN_NODE_LAYOUT(R)	5
4	PTC Tracking Routines	6
4.1	Layouts Types	6
4.2	Objects which can be tracked (Rays, spin, Taylor maps)	6
4.2.1	Real or Real_8 X(1:6)	6
4.2.2	Probe and Probe_8	6
4.2.2.1	Probe	6
4.2.2.2	Probe_8	7
4.2.3	TEMPORAL_PROBE	7
4.2.4	Tracking 6 Rays as X(1:6): without wrapping a probe	7
4.2.4.1	Single ELEMENT TRACK(EL,X,K)	7
4.2.4.2	Single FIBRE TRACK(C,X,K)	7
4.2.4.3	Single INTEGRATION_NODE TRACK_NODE_SINGLE(T,X,K)	8
4.2.4.4	ORBIT_TRACK_NODE(I,X,K)	8
4.2.4.5	TRACK(R, X, I1, I2, K)	8
4.2.4.6	Result=TRACK_FLAG (R,X,I1,I2,K)	8
4.2.4.7	Object TRACK(X,K,FIBRE1,FIBRE2)	8
4.2.5	Tracking 6 Rays as X(1:6) wrapping a probe	8
4.2.5.1	TRACK_NODE_X(T,X,K)	9
4.2.5.2	Position TRACK_PROBE_X(R,X,K, FIBRE1,FIBRE2,NODE1,NODE2)	9
4.2.5.3	Object TRACK_PROBE_X(X,K, FIBRE1,FIBRE2,NODE1,NODE2)	9
4.2.6	Tracking Probe	9
4.2.6.1	TRACK_NODE_PROBE(T,XS,K)	10

4.2.6.2	Object TRACK_PROBE(XS,K, FIBRE1,FIBRE2,NODE1,NODE2)	10
4.2.6.3	Position TRACK_PROBE(R,XS,K, FIBRE1,FIBRE2,NODE1,NODE2)	10
4.2.6.4	Position TRACK_PROBE2(R,XS,K, I1,I2)	10
4.2.7	Tracking THREE_D_INFO	10
4.2.7.1	TRACK_NODE_V(T,V,K,REF)	10
4.2.8	Filling a ray in the NODE_LAYOUT	10
4.2.8.1	TRACK_FILL_REF(R,FIX,I1,K))	10
4.2.9	Tracking TEMPORAL_PROBE	11
4.2.9.1	TRACK_TIME(XT,DT,K)	11
5	Exception Handling in Tracking and Apertures	12
5.1	The various ways to crash!	12
5.2	Two types of routines in PTC	13
5.3	Example of using exceptions: correctly and incorrectly	13
5.4	Physical and mathematical aperture	14
5.4.1	Element Aperture	14
5.4.2	Distributed Aperture	15
6	Tracking with Radiation in PTC	16
6.1	Tracking with Classical Radiation	16
6.1.1	Finding the Energy loss: GET_LOSS(R,energy,deltap)	16
6.2	Tracking a Quantum Envelope and computing the equilibrium sizes	17
6.3	Tracking the beam sizes around	18
6.3.1	“Correctly”: Routine extract_beam_sizes	18
6.3.2	“Incorrectly”: using lattice functions and Chao emittances	18
6.3.3	“Incorrectly” fixing the problem using nonsymplectic lattice function	19
6.4	Tracking a with a stochastic kick: two ways	20
6.4.1	Kicking the energy at every node: flag compute_stoch_kick	20
6.4.2	Kicking the energy at every turn using the result of the normal form	20
7	Slow RF Modulation	22
7.1	The types rf_phasor and rf_phasor_8	22
7.2	Effect on the magnet	22
7.3	Tracking with RF modulation	23
7.4	Normal Form with RF modulation	24
8	Tracking Spin in PTC	25
8.1	Find Orbit with Spin	25
8.2	Invariant Spin field with Normal Form	25
8.2.1	Invariant Spin field with Normal Form	25
8.2.2	Spin field Lattice Funcions	26
8.2.3	Stroboscopic Average	26
9	Beam Beam Kick	29
9.1	Beam-Beam Flags	29
9.2	Approximately locating the beam-beam kick: standard rubbish	29
9.3	Exactly locating the beam-beam kick: locate_beam_beam(r,o,m,t,f)	30

10 Small Example Programs	31
10.1 Subroutine Track_a_ray(RADIATION_FLAG,SPIN_FLAG)	31
10.2 Subroutine Track_a_ray_time(RADIATION_FLAG,SPIN_FLAG)	34
10.3 Subroutine Track_a_ray_orbit	35
10.4 Subroutine Track_a_ray_around_ring	36
10.5 Subroutines to test the apertures flags	37
10.6 Subroutine displaying Radiation in PTC	41
10.7 Subroutine Showing a Beam-Beam Kick	46
10.8 Subroutine Track_with_rf_modulation	48
10.9 Subroutine Track_a_ray_with_spin	50
11 Review of normal form: Orbital and Spin	56
11.1 Very short review of the Orbital Normal Form	56
11.1.1 The Orbital Normal Form Itself	56
11.1.2 The Normal Form Itself	57
11.2 The case of spin	57
11.2.1 Spin Maps in FPP	58
11.2.2 Spin Maps Concatenation	58
11.2.3 Spin Normal Form	58
11.2.4 Factoring the Map T and the invariant spin axis \vec{n}	59
11.3 Corollaries of normal form: phase advance and stroboscopic average	61
11.3.1 Phase Advance	61
11.3.2 “Stroboscopic” average	62
Bibliography	64

List of Tables

Chapter 1. Introduction

1

Chapter 2. Type INTERNAL_STATE of PTC

2.1 Definition of a State

States control the tracking to some extent and also the TPSA calculation.

```
TYPE INTERNAL_STATE
  INTEGER TOTALPATH
  LOGICAL(LP) TIME,RADIATION,NOCAVITY,FRINGE,EXACTMIS,STOCHASTIC,ENVELOPE
  LOGICAL(LP) PARA_IN,ONLY_4D,DELTA
  LOGICAL(LP) SPIN,MODULATION
END TYPE INTERNAL_STATE
```

Only the logical PARA_IN is not associated to a state. It is activated by a unary plus on a state, for example,

```
CALL TRACK(R, X, I1, I2, +STATE)
```

This unary + activates parameter dependence in PTC. Parameters are magnet properties (b_n for example), which have been declared as TPSA parameters.

2.2 Eternal States

These are parameters saved to construct any state out. All the eternal states have a 0 appended at the end of their names. Generally it is better to use eternal states in constructing one's own state. For example:

```
my_state=default0+spin0
```

This state tracks with a cavity and spin but without radiation. The eternal states are defined as follows:

```
TYPE(INTERNAL_STATE),PARAMETER::DEFAULTO=INTERNAL_STATE (0,f,f,f,f,f,f,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::TOTALPATHO=INTERNAL_STATE (1,f,f,f,f,f,f,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::TIMEO=INTERNAL_STATE (0,t,f,f,f,f,f,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::RADIATIONO=INTERNAL_STATE (0,f,t,f,f,f,f,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::NOCAVITYO=INTERNAL_STATE (0,f,f,t,f,f,f,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::FRINGEO=INTERNAL_STATE (0,f,f,f,t,f,f,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::STOCHASTICO=INTERNAL_STATE(0,f,f,f,f,t,f,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::ENVELOPEO=INTERNAL_STATE (0,f,f,f,f,f,t,f,f,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::ONLY_4dO=INTERNAL_STATE (0,f,f,t,f,f,f,f,t,f,f,f)
TYPE(INTERNAL_STATE),PARAMETER::DELTAO=INTERNAL_STATE (0,f,f,t,f,f,f,f,t,t,f,f)
TYPE(INTERNAL_STATE),PARAMETER::SPINO=INTERNAL_STATE (0,f,f,f,f,f,f,f,f,f,t,f)
TYPE(INTERNAL_STATE),PARAMETER::MODULATIONO=INTERNAL_STATE(0,f,f,f,f,f,f,f,f,f,t)
```

2.3 State Default

This is the default state of PTC. All parameters are set to false. For example, in the default state, time is **not** used and a cavity is present, i.e., Time=false and Nocavity=false. Thus for example.

```
STATE=DEFAULTO
CALL TRACK(R, X, I1, I2, STATE)
```

tracks a particle X(1:6) replacing cavities by drifts and using path length and momentum rather than the more correct time and energy.

2.4 State Time

Time is used instead of path length.

2.5 State Totalpath

The total pathlength (or time) is computed. Very useful in recirculators and any bizarre machines. It is also **essential** if the frequency of a cavity is varied either numerical or as a TPSA parameter.

2.6 State Nocavity

Replaces cavities by drifts.

2.7 State Fringe

Turns on multipole fringe fields up to the level HIGHEST_FRINGE which is defaulted to two (quadrupole). Dipoles fringe fields are always on by default on dipoles whose main purpose is to bend the “design orbit” orbit. Anyway in PTC it is possible to control these fringe effects completely.

2.8 State Spin

Spin is turned on. Probe tracking only.

2.9 States controlling Radiation

2.9.1 State Radiation

Radiation is turned on. Probe tracking only.

2.9.2 State Envelope

Envelope is turned on. Probe tracking only. (Equivalent to the theory of synchrotron integrals, more accurate.)

2.9.3 State Stochastic

Previously small stochastic kicks are used to produce a genuinely random variable for the energy.

2.10 State Modulation

Modulation of magnets with one frequency is turned on. Probe tracking only.

2.11 States Affecting Map Calculations

2.11.1 State Only_4d

The TPSA part of the orbital phase space is limited to (x, p_x, y, p_y) .

2.11.2 State Delta

The TPSA part of the orbital phase space is limited to (x, p_x, y, p_y, δ) .

Chapter 3. Geometry Routines of PTC

3.1 Survey Routines

3.2 Over Fibres

3.3 Over Integration_node: `FILL_SURVEY_DATA_IN_NODE_LAYOUT(R)`

Chapter 4. PTC Tracking Routines

4.1 Layouts Types

There are two type of layouts in PTC.

1. The usual LAYOUT made of FIBREs.
2. The NODE_LAYOUT which depends on the LAYOUT

A LAYOUT “R” contains a NODE_LAYOUT R%T if once elects to create this layout. Neither radiation nor spin nor beam-beam lens can be used with L. One must create the NODE_LAYOUT R%T.

N.B. Spin, Classical Radiation, Stochastic Radiation, distributed apertures, Beam-Beam kicks and magnet modulation all require the NODE_LAYOUT.

4.2 Objects which can be tracked (Rays, spin, Taylor maps)

4.2.1 Real or Real_8 X(1:6)

REAL_8 is a type of the FPP package. It can be real, Taylor or a so-called knob. The knob is a simple Taylor. This is described elsewhere. Here it suffices to say that polymorphs are useful for producing Taylor maps.

4.2.2 Probe and Probe_8

4.2.2.1 Probe

The probe is defined in PTC as:

```
type probe
  real(dp) x(6)
  type(spinor) s(3)
  type(rf_phasor) ac
  logical u
  type(integration_node),pointer :: lost_node
end type probe
```

The probe has a real part x(6) which is simply a ray to be tracked through the lattice. The spinor consists of 3 spin axis. One can initialize a probe as follows:

```
Probe=X
```

where X is an array of 6 real numbers. In that case, Probe%X=X. The spinor, which is a FPP construct containing a full basis, is initialized as the three independent spin axes, \vec{i} , \vec{j} and \vec{k} .

The logical probe%u is set to true if the probe becomes unstable. probe%lost_node points to the place where the probe was lost. Finally probe%ac is used in modulating magnets in PTC and is described elsewhere.

4.2.2.2 Probe_8

Probe_8 contains one unexpected term: the real array E_ij(6,6). This array contains the stochastic kicks to the quadratic moments due to radiation.

```

type probe_8
  type(real_8) x(6)
  type(spinor_8) s(3)
  type(rf_phasor_8) AC
  real(dp) E_ij(6,6)
  logical u
  type(integration_node), pointer :: lost_node
end type probe_8

```

4.2.3 TEMPORAL_PROBE

```

type TEMPORAL_PROBE
  TYPE(probe) XS
  TYPE(INTEGRATION_NODE), POINTER :: NODE
  real(DP) DS, POS(6)
END type TEMPORAL_PROBE

```

The position in space is given by the location of its closest INTEGRATION_NODE stored at NODE. DS is the distance from that node. XS%X contains the 6 usual variables of PTC.

$$XT\%POS(1:6) = (x, y, z, p_x, p_y, p_z) \text{ in metres and GeV} \quad (4.1)$$

4.2.4 Tracking 6 Rays as X(1:6): without wrapping a probe

These routines do not have radiation nor spin nor magnet modulation. X is an array of six real(dp) or REAL_8.

4.2.4.1 Single ELEMENT TRACK(EL,X,K)

This routine is called inside the FIBRE. It is the routine that a standard accelerator physics code calls: pushes through a magnet or its polymorphic version. **Users should avoid this unless they know what they are doing. This routine is useful for “on the bench” factory fitting of an element of the DNA for example although FIBRE tracking may often do the job as well.**

4.2.4.2 Single FIBRE TRACK(C,X,K)

This the fundamental routine that tracks through a single FIBRE.

- C is a FIBRE.
- X is an array of initial conditions and final upon return.
- K is an INTERNAL_STATE.

4.2.4.3 Single INTEGRATION_NODE TRACK_NODE_SINGLE(T,X,K)

This is the fundamental routine that tracks through a single INTEGRATION_NODE fully consistent with the FIBRE. **It has no radiation, no spin and no modulation. It does not reset the stability flags: user must do that on his own if he elects to use this routine in a piece of code.**

- T is a INTEGRATION_NODE.,
- X is an array of initial conditions and final upon return.
- K is an INTERNAL_STATE.

4.2.4.4 ORBIT_TRACK_NODE(I,X,K)

Tracks through a special object called the ORBIT_LATTICE attached to the NODE_LAYOUT. This is used in pancake style space charge calculations.

- I is the integer corresponding to the group of ORBIT_NODES
- X is an array of initial conditions and final upon return.
- K is an INTERNAL_STATE.

4.2.4.5 TRACK(R, X, I1, I2, K)

- R is a LAYOUT,
- X is an array of initial conditions and final upon return.
- K is an INTERNAL_STATE.
- Integer I1 is the position of the first FIBRE.
- I2 is the position of the second FIBRE: optional variable. PTC tracks all the way to the front of I2. If not present, the particle goes one turn or to the end of the beam line.

4.2.4.6 Result=TRACK_FLAG (R,X,I1,I2,K)

This routine returns 0 if stable otherwise 1 if unstable. Calls the routine of 4.2.4.5

4.2.4.7 Object TRACK(X,K,FIBRE1,FIBRE2)

- Here one specifies the initial FIBRE in FIBRE1 and the program tracks to the front of FIBRE2
- FIBRE2 is optional. If not there, tracking continues all the way to the front of FIBRE1 or to a terminated FIBRE pointer.

Warning: this routine could easily go into an infinite do-loop. Advanced users can also create strange topologies which will allow transfer from one layout to another using this routine.

4.2.5 Tracking 6 Rays as X(1:6) wrapping a probe

These routines **can** have radiation, spin and/or magnet modulation.

4.2.5.1 TRACK_NODE_X(T,X,K)

- T is an INTEGRATION_NODE.
- K is an INTERNAL_STATE.

4.2.5.2 Position TRACK_PROBE_X(R,X,K, FIBRE1,FIBRE2,NODE1,NODE2)

- R is a LAYOUT,
- X is an array of initial conditions and final upon return.
- K is an INTERNAL_STATE.
- FIBRE1,FIBRE2,NODE1,NODE2 are all integers.
- Here one specifies the initial FIBRE **position** in FIBRE1 and the program tracks to the front of FIBRE2
- FIBRE2 is optional. If not there, tracking continues all the way to the front of FIBRE1 or to a terminated FIBRE pointer.
- Here one specifies the initial NODE1 and the program tracks to the front of NODE2
- NODE2 is optional. If not there, tracking continues all the way to the front of NODE1 or to a terminated NODE pointer.

Nodes and Fibres are mixable. For example, TRACK_PROBE_X(R,X,K, FIBRE1=p1,NODE2=n2) is acceptable.

4.2.5.3 Object TRACK_PROBE_X(X,K, FIBRE1,FIBRE2,NODE1,NODE2)

- X is an array of initial conditions and final upon return.
- K is an INTERNAL_STATE.
- FIBRE1,FIBRE2,NODE1,NODE2 are all objects: FIBREs and INTEGRATION_NODES.
- Here one specifies the initial FIBRE in FIBRE1 and the program tracks to the front of FIBRE2
- FIBRE2 is optional. If not there, tracking continues all the way to the front of FIBRE1 or to a terminated FIBRE pointer.
- Here one specifies the initial INTEGRATION_NODE in NODE1 and the program tracks to the front of NODE2
- NODE2 is optional. If not there, tracking continues all the way to the front of NODE1 or to a terminated NODE pointer.

Nodes and Fibres are mixable. For example, TRACK_PROBE_X(X,K, FIBRE1=p1,NODE2=n2) is acceptable.

4.2.6 Tracking Probe

The following routines track a probe and thus contain, if needed, spin information and magnet modulation. Of course they also can have radiation.

4.2.6.1 TRACK_NODE_PROBE(T,XS,K)

- T is an INTEGRATION_NODE.
- K is an INTERNAL_STATE.

4.2.6.2 Object TRACK_PROBE(XS,K, FIBRE1,FIBRE2,NODE1,NODE2)

- XS is a probe.

All the other comments of 4.2.5.3 apply to this routine.

4.2.6.3 Position TRACK_PROBE(R,XS,K, FIBRE1,FIBRE2,NODE1,NODE2)

- XS is a probe.

All the other comments of 4.2.5.2 apply to this routine.

4.2.6.4 Position TRACK_PROBE2(R,XS,K, I1,I2)

- XS is a probe.
- I1 and I2 are INTEGRATION_NODE positions.

This routine is basically privately used to generate 4.2.6.3.

4.2.7 Tracking THREE_D_INFO

The type THREE_D_INFO permits tracking of an object containing all the information necessary to plot the trajectory in 3 dimension. This is absolutely necessary in a system where magnets are positioned “arbitrary”, i.e., far from their “MAD8” survey positions.

4.2.7.1 TRACK_NODE_V(T,V,K,REF)

- T is an INTEGRATION_NODE.
- V is of type THREE_D_INFO.
- K is an INTERNAL_STATE.
- REF is a LOGICAL. If TRUE a reference ray, part of V, is used to scale the data to be plotted. Typical the closed orbit or the design trajectory.
- If false, the reference ray is read from the INTEGRATION_NODE itself. This ray is filled with TRACK_FILL_REF.

4.2.8 Filling a ray in the NODE_LAYOUT**4.2.8.1 TRACK_FILL_REF(R,FIX,I1,K)**

- R is a LAYOUT.
- FIX(1:6) is a ray to be tracked from I1 to the end of the LAYOUT (or all around).
- I1 is the integer position of the first FIBRE.
- K is an INTERNAL_STATE.

On each node of the layout the entrance coordinates (x, y) are stored on the node $(T\%REF(1), T\%REF(2))$. The exit coordinate are on $(T\%REF(3), T\%REF(4))$.

4.2.9 Tracking TEMPORAL_PROBE

Temporal probes permit first order accurate time tracking. If there is nothing but the lattice itself (no collective effects, no wake effect, etc...), then it is equivalent to the ordinary tracking if one projects the result back to the start of a node.

4.2.9.1 TRACK_TIME(XT,DT,K)

- XT is a TEMPORAL_PROBE.
- DT is the time elapsed ($T_1 - T_0$).
- K is an INTERNAL_STATE.

Chapter 5. Exception Handling in Tracking and Apertures

PTC and FPP have a collection of exception flags and settings to prevent undefined operations, exceeding apertures— physical and/or mathematical.

Input settings

1. `aperture_flag (=true)` \implies Turns aperture checks on or off globally.
2. `check_madx_aperture (=true)` \implies Turns magnet aperture on globally if defined.
3. `s_aperture_check (=true)` \implies Turns INTEGRATION_NODE aperture on globally if defined for case0 node.
4. `absolute_aperture (= 1)` \implies Turns generic aperture on: prevents crashes and NAN.
5. `hyperbolic_aperture (= 10)` \implies Prevents various FPP defined functions with exponential arguments to crash.
6. `no_hyperbolic_in_normal_form (=false)` \implies If true, prevents a “hyperbolic” fixed point normal form, i.e., as in the case of an integer or half integer resonance.
7. `real_warning (=true)` \implies Detects single precision in FPP and returns an error.
8. `da_absolute_aperture (= 106)` \implies Limits sizes while tracking with Berz’s PPUSH algorithm.

Output information

1. `check_stable (=true)` \implies Detects single precision in FPP and returns an error.
2. `stable_da (=true)` \implies Detects a problem in Dabnew
3. `messagelost (= Aperture has been reset)` \implies Default message which is changed when an exception is thrown.
4. `lost_fibre nullified` pointer to the FIBRE where the particle was lost.
5. `lost_node nullified` pointer to the INTEGRATION_NODE where the particle was lost.
6. `xlost(1:6) zeros` Contains the coordinates at the moment the particle gets lost at a PTC aperture.

5.1 The various ways to crash!

PTC can be forced to stop execution for several reasons:

1. Physics: things get too big for the apertures— physical aperture set by the users on each magnet or the global aperture `absolute_aperture`.
2. Things are not well defined mathematical: square roots of negatives, exponential which are too large, arcsin of numbers greater than one, etc...

3. The Taylor series package tries to do crazy things.
4. Normal forms, which are essential part of accelerator theory applied to tracking codes, can go wild. `no_hyperbolic_in_normal_form` will prevent the normalizaton of a plane into $x^2 - p^2$. That is to say that one can reject hyperbolic fixed points.

5.2 Two types of routines in PTC

There are two type of tracking routines in PTC:

1. Routines that ignore previous exceptions and restart tracking as if nothing ever happened. These are typically called by the casual user.
2. Routines that are permanently disabled as long as the flag `check_stable` is false. These are internal routines which the casual user rarely calls. 4.2.4.2 4.2.4.3

5.3 Example of using exceptions: correctly and incorrectly

The code, between the rows of `#` signs, executes a check after each call to track. If one skips this inside a loop (`CORRECT=FALSE`), the unstable particle will be dragged incorrectly to the end of the lattice.

N.B. If you use a `NODE_LAYOUT` you can look at `lost_node` as well as `lost_fibre`.

```

write(6,*) " Do you want a correct implementation of exceptions -> yes=true "
read(5,*) CORRECT
WRITE(6,*) "    "
WRITE(6,*) " GIVE A 'BIG' INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R (example 0.1) "
READ(5,*) R
X0(1:6)=R

x=x0
p=>als%start
do while(.not.associated(p%next,als%start)) ! this is one turn minus the last element

    call track(x,state,fibre1=p,fibre2=p%next)

#####
if(.not.check_stable.and.CORRECT) then
    write(6,*) "In the loop"
    write(6,"(a255)") messagelost
    write(6,*) lost_fibre%pos,lost_fibre%mag%name
    write(6,*) "Unstable ray"
    WRITE(6,'(6(1x,E15.8))') Xlost
    exit
endif
#####

p=>p%next
enddo

```

```

if(.not.check_stable) then
  write(6,*) "End of loop"
  write(6,"(a255)") messagelost
  write(6,*) lost_fibre%pos,lost_fibre%mag%name
  write(6,*) "Unstable ray"
  WRITE(6,'(6(1x,E15.8))') Xlost
else
  write(6,*) "stable ray"
  WRITE(6,'(6(1x,E15.8))') X
endif

x=x0
call track(x,state,fibre1=als%start,fibre2=als%end) ! this is one turn minus the last element

if(.not.check_stable) then
  write(6,*) "Using one call to Track"
  write(6,"(a255)") messagelost
  write(6,*) lost_fibre%pos,lost_fibre%mag%name
  write(6,*) "Unstable ray"
  WRITE(6,'(6(1x,E15.8))') Xlost
else
  write(6,*) "stable ray"
  WRITE(6,'(6(1x,E15.8))') X
endif

```

5.4 Physical and mathematical aperture

5.4.1 Element Aperture

To see how that works, we put an aperture in the example program 10, we see at 10 the calls

```

call put_an_aperture(.002d0,0.001d0)
call remove_temporarily_an_aperture
call Track_and_test_aperture

```

And we call the subroutine `Track_and_test_aperture` 10.5. This subroutine has the following input:

```

call put_back_an_aperture

write(6,*) "aperture => True or false "
read(5,*) aperture_flag
write(6,*) " Check_madx_aperture (magnet aperture) => True or false "
read(5,*) check_madx_aperture
write(6,*) " Give value of the absolute_aperture "
read(5,*) absolute_aperture

WRITE(6,*) " GIVE A 'BIG' INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R (example 0.1) "
READ(5,*) R

```

The reader can check his understanding of the code by inputting values of the `absolute_aperture` and `R` either much bigger or much smaller than the arguments of `put_an_aperture`,—0.002 and 0.001 at the moment.

Some important points:

- The `Absolute_aperture` should be big, say 1 metre or more, however it should not be too big in order to prevent the apparition of a NAN— ideally, somewhere clearly outside the beam pipe but not on the Andromeda Galaxy.
- the flag `check_madx_aperture` controls the user defined apertures globally.
- Locally one can turn off an aperture by changing its “kind” to a negative number. See the example program at 10.5.

5.4.2 Distributed Aperture

A element aperture can be distributed on every step of integrations. This requires an `NODE_LAYOUT`.

Again the main program contains

```
call put_an_s_aperture(0.00001d0,0.00002d0,0.8d0)
call Track_and_test_s_aperture
```

Here 0.8d0 indicates that a single aperture will be put 80 % down the magnet. The aperture is tiny: $(x, y) = (0.00001, 0.00002)$. The subroutine `Track_and_test_s_aperture` is called. The following questions are asked.

```
write(6,*) "aperture => True or false "
read(5,*) aperture_flag
write(6,*) " Check_madx_aperture (magnet aperture) => True or false "
read(5,*) check_madx_aperture
write(6,*) " s_aperture_check ( Extended magnet aperture) => True or false "
read(5,*) s_aperture_check
write(6,*) " Give value of the absolute_aperture "
read(5,*) absolute_aperture

WRITE(6,*) " GIVE A 'BIG' INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R (example 0.1) "
READ(5,*) R
X0(1:6)=R
x=x0
```

`s_aperture_check` powers the distributed apertures.

N.B. The flag `check_madx_aperture` is necessary because distributed apertures are made of `NST+1` regular element apertures.

Chapter 6. Tracking with Radiation in PTC

PTC can track with radiation in 3 complementary modes:

1. Classical radiation can be turned on the state Radiation2.9.1.
2. The equilibrium beams sizes are computed using the state Envelope 2.9.2. The 3 Chao equilibrium “emittances” are extracted.
3. One can also track with a real stochastic kick using the State Stochastic.2.9.3

Only the routines action on a NODE_LAYOUT can handle radiation: 4.2.5, 4.2.6, 4.2.9

6.1 Tracking with Classical Radiation

```
!!!! Tracking with a wrapped command      !!!!!!!!!!!
wrapped_state=default0+radiation0
CALL FIND_ORBIT_x(als,X,wrapped_state,1.0e-5_dp,fibre1=1)

WRITE(6,'(A)') " Closed orbit with Radiation "
WRITE(6,'(6(1x,E15.8))') x
call GET_loss(als,energy,deltap)
write(6,'(a32,2(1x,E15.8))') "Energy loss: GEV and DeltaP/p0c ",energy,deltap

id=1
y=x+id
call TRACK_PROBE_X(ALS,y,wrapped_state, FIBRE1=1)
id=y

normal=id    ! Normal is a regular Normalform type
write(6,*) "Tunes "
write(6,*) normal%tune(1:3)
write(6,*) "Damping decrements"
write(6,*) normal%damping(1:3)

!!!!!!!!!!!! end of Tracking with a wrapped command      !!!!!!!!!!!!!
```

N.B. This is a deterministic map with classical radiation. Without a cavity, as in a real machine, the beam will eventually collapse into the aperture.

6.1.1 Finding the Energy loss: GET_LOSS(R,energy,deltap)

To find the energy lost in one turn, one must first find the closed orbit using for example:

```
CALL FIND_ORBIT_x(als,X,wrapped_state,1.0e-5_dp,fibre1=1)
```

During the tracking, PTC stores the energy regained in GeV at every RF cavity. To get the total energy lost (or regained), one calls:

```
GET_LOSS(R,energy,deltap)
```

deltap is the $x(5)$ of PTC at the first fibre. **N.B. In PTC the reference momentum p_0c can be different for each magnet: PTC has energy patches as well.**

6.2 Tracking a Quantum Envelope and computing the equilibrium sizes

These routines **must** track a polymorphic probe; PROBE_8. 4.2.6

```

type probe_8
  type(real_8) x(6)      ! Polymorphic orbital ray
  type(spino_8) s(3)    ! Polymorphic spin
  type(rf_phasor_8) AC  ! Modulation
  real(dp) E_ij(6,6)   ! Envelope kick
!  stuff for exception
  logical u
  type(integration_node),pointer :: lost_node
end type probe_8

```

The probe_8 contains implicitly the linear map M in the polymorph $x(6)$.

$$\Sigma^f = M(\Sigma^0 + E)M^\dagger \quad ; \quad \Sigma_{ij} = \langle x_i x_j \rangle \quad (6.1)$$

```

!!!!!!!!!!!! Tracking a probe_8 to get the map  !!!!!!!!!!!!!
state=default0+radiation0+envelope0
xs0=x      ! closed orbit intializing a probe
m=1        ! damapspin set to identity
xs=xs0+m   ! Probe_8 = closed orbit probe + Identity

call track_probe(als,xs,state,fibre1=1)
m=xs       ! damapspin = Probe_8
nf=m       ! normal_spin = damapspin (Normalization including spin (if present) or radiation
           ! envelope if present. (Spin without radiation)
write(6,*) nf%n%tune(1:3)
write(6,*) nf%emittance
write(6,*) nf%s_ij0(1,1),nf%s_ij0(5,5)
!!!!!!!!!!!! end Tracking a probe_8 to get the map  !!!!!!!!!!!!!

```

nf is of type normal_spin. It normalises the orbital map, spin and radiation.

```

type normal_spin
  type(normalform) N    ! regular orbital normal form
  type(damapspin) a1    ! brings to fixed point
  type(damapspin) ar    ! normalises around the fixed point
  type(damapspin) as    ! pure spin map
  type(damapspin) a_t   ! !! (a_t%m,a_t%s) = (a1%m, I ) o (I ,as%s) o (ar%m,I)
!!! extra info
  integer M(NDIM,NRESO),MS(NRESO),NRES ! orbital and spin resonances to be left in the map
  type(real_8) n0(3)    ! n0 vector
  type(real_8) theta0  ! angle for the matrix around the orbit (analogous to linear tunes)
!!!Envelope
  real(dp) s_ij0(6,6)  ! equilibrium beam sizes
  real(dp) emittance(3),tune(3),damping(3) ! equilibrium emittances (partially well defined)
  logical(lp) AUTO,STOCHASTIC
  real(dp) KICK(3)     ! fake kicks for tracking stochastically
  real(dp) STOCH(6,6) ! Diagonalized of stochastic part of map for tracking stochastically
end type normal_spin

```

6.3 Tracking the beam sizes around

6.3.1 “Correctly”: Routine `extract_beam_sizes`

The example tracks the equilibrium beam sizes computed in section 6.2. The routine

```
extract_beam_sizes(xs,nf%s_ij0,sizes)

!!!!!!! Tracking a probe_8 to beam sizes around the ring !!!!!!!!
call kanalnummer(mf,"beam_xx_inf.dat")

state=default0+radiation0+envelope0
xs0=x      ! closed orbit intializing a probe
m=1       ! damaspin set to identity
xs=xs0+m  ! Probe_8 = closed orbit probe + Identity

p=>als%start

call extract_beam_sizes(xs,nf%s_ij0,sizes)
write(mf,*) p%mag%name, sizes(1,1)

do i=1,als%n
call track_probe(als,xs,state,fibre1=i,fibre2=i+1)
call extract_beam_sizes(xs,nf%s_ij0,sizes)
p=>p%next
write(mf,*) p%mag%name, sizes(1,1)
enddo
close(mf)
!!!!!!! Tracking a probe_8 to beam sizes around the ring using emittances !!!!!!!!
```

6.3.2 “Incorrectly”: using lattice functions and Chao emittances

This piece of code is instructive: the lattice functions are only valid in the symplectic case and the Chao emittances, **while better than Sands**, are not accurate concepts for large damping. Therefore this code, when compared with the results of section 6.3.1, gives us a idea of the accuracy of the Chao synchrotron integrals formalism. (Usually extremely accurate unless the machine is on a linear coupling resonance!)

Equation

$$\underbrace{\langle x_1^2 \rangle}_{\text{sizes}(1,1)} = \underbrace{(A_{11}^2 + A_{12}^2)}_{\text{bet}(1)} \varepsilon_1 + \underbrace{(A_{13}^2 + A_{14}^2)}_{\text{bet}(2)} \varepsilon_2 + \underbrace{(A_{15}^2 + A_{16}^2)}_{\text{bet}(3)} \varepsilon_3 \quad (6.2)$$

appears as

```
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
```

in the following piece of software.

```
!!!!!!! Tracking a probe_8 to beam sizes around the ring using emittances !!!!!!!!

call kanalnummer(mf,"beam_xx_inf_using_emittances.dat")
```

```

state=default0+radiation0+envelope0
xs0=x      ! closed orbit intializing a probe
m=nf%a_t  ! damapspin set to A
xs=xs0+m  ! Probe_8 = closed orbit probe + Identity

p=>als%start

bet(1)=(m%%v(1).sub.'1')**2+(m%%v(1).sub.'01')**2
bet(2)=(m%%v(1).sub.'001')**2+(m%%v(1).sub.'0001')**2
bet(3)=(m%%v(1).sub.'00001')**2+(m%%v(1).sub.'000001')**2
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
bet0=sizes(1,1)
write(mf,*) p%mag%name, sizes(1,1)

do i=1,als%n
call track_probe(als,xs,state,fibre1=i,fibre2=i+1)
m=xs
bet(1)=(m%%v(1).sub.'1')**2+(m%%v(1).sub.'01')**2
bet(2)=(m%%v(1).sub.'001')**2+(m%%v(1).sub.'0001')**2
bet(3)=(m%%v(1).sub.'00001')**2+(m%%v(1).sub.'000001')**2
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
p=>p%next
write(mf,*) p%mag%name, sizes(1,1)
enddo
betf=sizes(1,1)
write(mf,*) " log of change after one turn and damping "
write(mf,*) log(betf/bet0)/2.d0,nf%n%damping(1)
close(mf)
!!!!!!!!!!!! Tracking a probe_8 to beam sizes around the ring using emittances !!!!!!!!!!!!!

```

6.3.3 “Incorrectly” fixing the problem using nonsymplectic lattice function

In a true nonsymplectic system, it is possible to define lattice functions which are primarily “beta” in nature and another group which is “eta” in nature. The first group can be used to compute the effect on the complex part of the eigenvalues when a force is applied. The other group are connected to the change of the damping as a function of the force. There is a relation between these functions which the basis for connecting Chao’s formalism to Sands formalism.

Obviously these lattice functions are periodic.

$$\underbrace{\langle x_1^2 \rangle}_{\text{sizes}(1,1)} = \underbrace{(A_{11}A_{22}^{-1} - A_{12}A_{12}^{-1})}_{\text{bet}(1)} \varepsilon_1 + \underbrace{(A_{13}A_{42}^{-1} - A_{14}A_{32}^{-1})}_{\text{bet}(2)} \varepsilon_2 + \underbrace{(A_{15}A_{62}^{-1} - A_{16}A_{52}^{-1})}_{\text{bet}(3)} \varepsilon_3 \quad (6.3)$$

!!!!!!!!!!!! Tracking a probe_8 to beam sizes around the ring using emittances and super-lattice

```

call kanalnummer(mf,"beam_xx_inf_using_emittances_ripken.dat")

state=default0+radiation0+envelope0
xs0=x      ! closed orbit intializing a probe
m=nf%a_t  ! damapspin set to A

```

```

xs=xs0+m ! Probe_8 = closed orbit probe + Identity

p=>als%start

id=(m%m.sub.1)**(-1)
bet(1)=(m%m%v(1).sub.'1')*(id%v(2).sub.'01')-(m%m%v(1).sub.'01')*(id%v(1).sub.'01')
bet(2)=(m%m%v(1).sub.'001')*(id%v(4).sub.'01')-(m%m%v(1).sub.'0001')*(id%v(3).sub.'01')
bet(3)=(m%m%v(1).sub.'00001')*(id%v(6).sub.'01')-(m%m%v(1).sub.'000001')*(id%v(5).sub.'01')
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
write(mf,*) p%mag%name, sizes(1,1)

do i=1,als%n
call track_probe(als,xs,state,fibre1=i,fibre2=i+1)
m=xs
id=(m%m.sub.1)**(-1)
bet(1)=(m%m%v(1).sub.'1')*(id%v(2).sub.'01')-(m%m%v(1).sub.'01')*(id%v(1).sub.'01')
bet(2)=(m%m%v(1).sub.'001')*(id%v(4).sub.'01')-(m%m%v(1).sub.'0001')*(id%v(3).sub.'01')
bet(3)=(m%m%v(1).sub.'00001')*(id%v(6).sub.'01')-(m%m%v(1).sub.'000001')*(id%v(5).sub.'01')
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
p=>p%next
write(mf,*) p%mag%name, sizes(1,1)
enddo
close(mf)

!!!!!!!!!!!! End of Tracking a probe_8 to beam sizes around the ring using emittances and super

```

6.4 Tracking a with a stochastic kick: two ways

6.4.1 Kicking the energy at every node: flag compute_stoch_kick

The result of the beam envelope calculation in section 6.2, that is to say the quantum fluctuation $\langle \delta^2 \rangle_{quantum}$ is stored at every INTEGRATION_NODE is the flag compute_stoch_kick is true. This is used, if the state RADIATION is true, to track with a stochastic kick.

N.B. compute_stoch_kick is overwritten if one tracks with K%envelope=true in the internal state. This is not a problem if the beam envelope is always tracked on the radiative closed orbit.

6.4.2 Kicking the energy at every turn using the result of the normal form

The normal form normal_spin 6.2 contains a flag nf%stochastic. If true, FPP computes a matrix nf%STOCH(6,6) and 3 stochastic kicks in nf%KICK(3). The map is gotten by the following formula:

$$\vec{x}_{n+1} = S^{-1} \left(SM \vec{x}_n + \begin{pmatrix} t_1 k_1 \\ t_1 k_1 \\ t_2 k_2 \\ t_2 k_2 \\ t_3 k_3 \\ t_3 k_3 \end{pmatrix} \right)$$

where $S = nf\%STOCH$ and $k = nf\%KICK$ (6.4)


```
do i=1,n
  x=matmul(mat,x)
  x=matmul(stoch,x)

do j=1,3
  t=RANF()
  if(t>half) then
    t=one
  else
    t=-one
  endif
  x(2*j-1)= x(2*j-1)+ t*nf%kick(j)
  x(2*j  )= x(2*j)+ t*nf%kick(j)
enddo
  x=matmul(stochi,x)

  sizes(1,1)=sizes(1,1)+x(1)**2
  sizes(2,2)=sizes(2,2)+x(2)**2
  sizes(1,3)=sizes(1,3)+x(1)*x(3)
enddo

sizes=sizes/n
write(6,*) sizes(1,1), sizes(2,2), sizes(1,3)
write(6,*) nf%s_ij0(1,1), nf%s_ij0(2,2), nf%s_ij0(1,3)
```

Chapter 7. Slow RF Modulation

In PTC it is possible to modulate magnets with a single RF modulation frequency.

7.1 The types `rf_phasor` and `rf_phasor_8`

For the probe and the `probe_8` we refer to section 4.2.2.1.

The probe contains an instance of the type `rf_phasor`

```
type rf_phasor
  real(dp) x(2)
  real(dp) om
end type rf_phasor
```

and `probe_8` contains an instance of the type `rf_phasor_8`

```
type rf_phasor_8
  type(real_8) x(2)
  type(real_8) om
end type rf_phasor_8
```

For example, we initialize a probe (without spin) as follows:

```
Ray=x
ray%ac%om=0.001d0
ray%ac%x(1)=1.d0    ! fake phasor
ray%ac%x(2)=0.d0    !   fake phasor
```

The `rf_phasor` contains a phasor (a clock) which rotates according to a variable `D_AC` stored on the integration node. The frequency of the rotation is “`om`”.

In PTC the following code is used:

```
beta0=>C%PARENT_FIBRE%beta0
xt = cos(XS%AC%om * c%DS_AC/beta0) *XS%AC%X(1) + sin(XS%AC%om * c%DS_AC/beta0) *XS%AC%X(2)
XS%AC%X(2) = -sin(XS%AC%om * c%DS_AC/beta0) *XS%AC%X(1) + cos(XS%AC%om * c%DS_AC/beta0) *XS%AC%X(2)
XS%AC%X(1) = xt
```

`D_AC` for an ordinary lattice is the length along the “ideal” orbit. Obviously a user can modify this. It is clear that the phasor, like the spin, simply rotates around the ring and is not influenced by other variables. This can only be correct for slow modulation: **N.B. real time is not used.**

7.2 Effect on the magnet

The (a_n, b_n) of the magnet will be modulated by a multiplicative factor V given by:

$$V = EL_{DC_ac} + EL_{A_ac} * (XS_{AC}X(1) * COS(EL_{theta_ac}) - XS_{AC}X(2) * SIN(EL_{theta_ac}))$$

`DC_ac` (normally equal to one), `A_ac`, `theta_ac` are properties of the element as can be seen in the following piece of code:

```

call move_to(als,p,"BEND",pos)
write(6,*) pos,p%mag%name,p%mag%vorname

      if(.not.associated(P%MAG%DC_ac)) then
        allocate(P%MAG%DC_ac)
        allocate(P%MAG%A_ac)
        allocate(P%MAG%theta_ac)

        allocate(P%MAGP%DC_ac)
        allocate(P%MAGP%A_ac)
        allocate(P%MAGP%theta_ac)
        CALL alloc(P%MAGP%DC_ac)
        CALL alloc(P%MAGP%A_ac)
        CALL alloc(P%MAGP%theta_ac)

        P%MAG%DC_ac=DC_ac
        P%MAG%A_ac=A_ac
        P%MAG%theta_ac=theta_ac*twopi
        P%MAGP%DC_ac=DC_ac
        P%MAGP%A_ac=A_ac
        P%MAGP%theta_ac=theta_ac*twopi
        P%MAG%slow_ac=.true.
        P%MAGP%slow_ac=.true.
      ENDIF

```

N.B. The logical `slow_ac` must be true otherwise all RF modulation is ignored.

7.3 Tracking with RF modulation

In the example of this manual, there are no synchrotron oscillation and the 5-6 plane (Energy-like and time-like) is dormant completely. But of course all that follows could have synchrotron oscillations and spin. It could also have a coasting beam normal form: Jordan normal form.

```

      state=default0+only_4d0
CALL FIND_ORBIT_x(als,X,state,1.0e-5_dp,fibre1=1)
Ray=x
ray%ac%om=0.01d0
ray%ac%x(1)=1.d0      ! fake phasor
ray%ac%x(2)=0.d0      !   fake phasor

      state=state+modulation0+only_4d0
      write(6,*) "Tracking data in modulation.dat"
      call kanalnummer(mf,"modulation.dat")
write(6,*) ray%x
write(6,*) ray%ac%x
      WRITE(mf,'(4(1x,E15.8))') ray%x(1:4)
do i=1,1000
  call TRACK_PROBE(als,ray,state, FIBRE1=1)
  WRITE(mf,'(4(1x,E15.8))') ray%x(1:4)

```

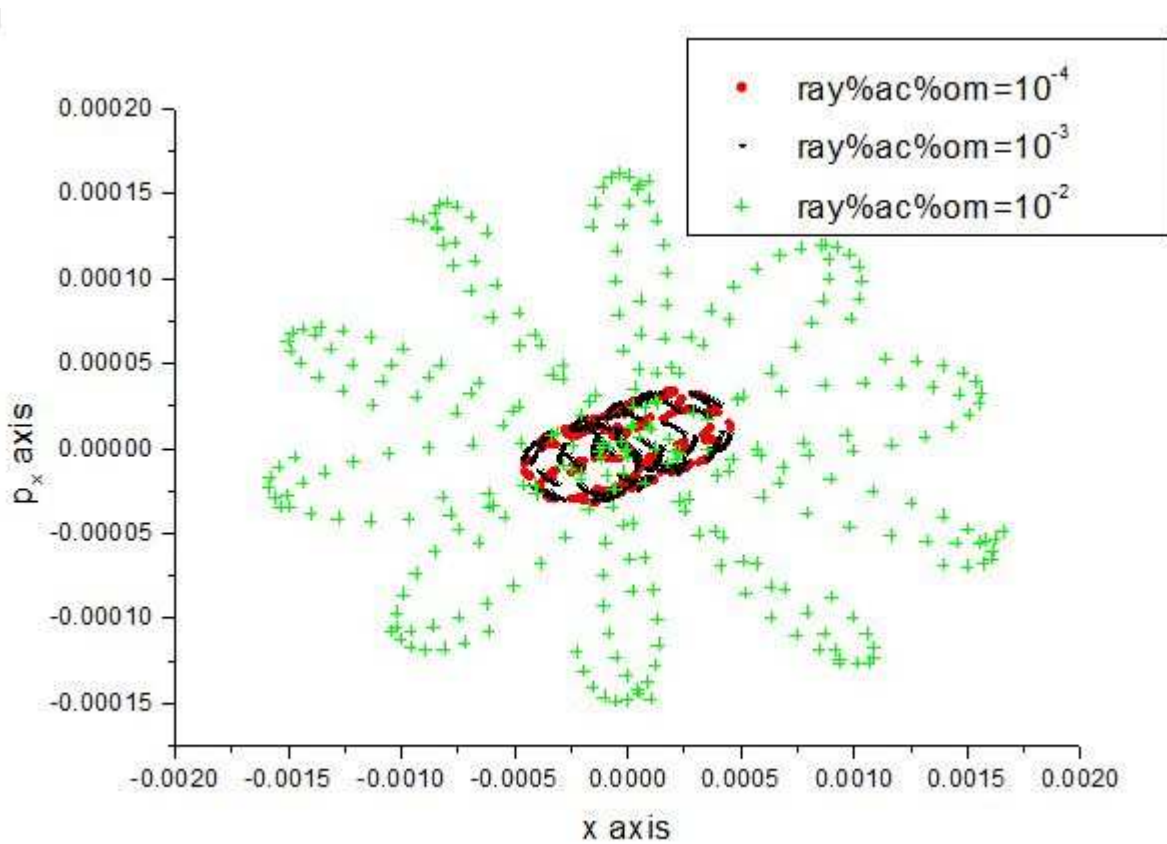
```

enddo
close(mf)

```

First the closed orbit is found without RF-modulation. In effect, the RF modulation is proportional to the “fake” phasor `ray%ac%x(1:2)`. Thus the closed orbit is obviously the solution when this phasor is at the origin since the modulation, in our approximation, is not influenced by the orbital motion. (We use a common “fake” time proportional to some s variable stored in `c%DS_AC` of the integration node `c`.)

Notice that modulation is possible if and only if the internal state contains modulation: `state=state+modulation0`



The above graph shows the plot of the for three different frequencies. Clearly for a slower modulation, the trajectory becomes the local closed orbit since we have an adiabatic change.

7.4 Normal Form with RF modulation

Chapter 8. Tracking Spin in PTC

For the full code, please see: Sec. (10.9).

The first thing to make sure is that you have the right kind of particle. The fibre parameter `fibre%AG` (which is $g - 2$) must have the value associated with the kind of particle you are using. This is automatic for electron, protons and muons but one can always print `fibre%AG` of any fibre to double check. PTC has the following $g - 2$ constants:

```
real(dp),parameter::A_ELECTRON=1.15965218111e-3_dp !frs NIST CODATA 2006
real(dp),parameter::A_MUON=1.16592069e-3_dp !frs NIST CODATA 2006
real(dp),parameter::A_PROTON=1.79284735e-0_dp !frs (approx) NIST CODATA 2006
```

8.1 Find Orbit with Spin

```
state=default0+DELTA0+SPINO
XS0=x0(1:6)
CALL FIND_ORBIT_probe_spin(ALS,XS0,STATE,c_1d_5,FIBRE1=1,theta0=theta0)
```

The triple spinor of `XS0%S(1 : 3)` is filled with 3 copies of the \vec{n}_0 invariant spin axis on the closed orbit for the user's convenience. The re

N.B. \vec{n}_0 should not be confused with the invariant spin field $\vec{n}(x, p_x, y, p_y, \delta_E, t)$ which is a far more complex entity to compute. See Sec. (8.2).

8.2 Invariant Spin field with Normal Form

It is one of the beauty of map based theory that all the operation in the code can be made (at the expense of speed sometimes), identical for anything we normalize. This includes lattice function calculations and phase advance calculations. **N.B. In actual practical coding, tricks are used in phase advance calculations to avoid a normal form at each step. This breaks the universality of the algorithm since the tricks depend on the actual object being normalized.**

8.2.1 Invariant Spin field with Normal Form

$T_1 = (M_1, S_1)$ is made of an orbital M_1 and a position dependent spin matrix S_1 .

$$\underbrace{(M_2, S_2)}_{T_2} \circ \underbrace{(M_1, S_1)}_{T_1} = (M_2 \circ M_1, S_2 \circ M_1 S_1) \quad (8.1)$$

In the library FPP, spin-orbital maps are of type "damapsin."

```
ID_S=1
XS=XS0+ID_S

WRITE(6,*)" Computing a one-turn spin taylor map "
CALL TRACK_PROBE(ALS,XS,STATE,FIBRE1=1)
ID_S=XS

WRITE(6,*)" Doing a Spin-Orbital Normal Form: NF_S=ID_S "
NF_S=ID_S
```

The Taylor map `ID_S` is of type “damapspin”: it is a spin orbital map. It is identity in both the spin part and the orbital part. Then the orbital closed orbit is added to it. The map is tracked for one-turn. The map `NF_S` is of type “normal_spin.” Both the orbital and the spin are normalized. (See FPP documentation)

8.2.2 Spin field Lattice Funcions

The loop below is a standard loop for lattice functions. It computes two types of things:

-
-

```

write(6,*) " Spin-orbit Twiss in file LATTICE_SPIN.TXT "

CALL KANALNUMMER(MF,"LATTICE_SPIN.TXT")
P=>ALS%START

XS=XS0+NF_S%A_T ! Probe= closed orbit + A (A as in M=AoRoA**(-1) )

DO I=1,ALS%N

! send probe across each element EXACTLY
CALL TRACK_PROBE(ALS,XS,STATE,FIBRE1=I,FIBRE2=I+1)
!!!!!! USER DEFINED
A_S=XS ! turn the exact probe into a Berz-like obscenity
! Compute an obscenity here. Here it is the Ripken beta function
BETA_XX=(A_S%M%V(1).SUB.'1')**2+(A_S%M%V(1).SUB.'01')**2
BETA_XY=(A_S%M%V(1).SUB.'001')**2+(A_S%M%V(1).SUB.'0001')**2
!!!!!! END OF USER DEFINED
!!!!!! EXTRA USER DEFINED
call factor(A_S,A_f,A_spin,A_l,A_nl) ! A_S= A_f o A_spin o A_l o A_nl
n_axis=2; ! creates e_y=(0,1,0) FPP arbitrary choice for the normal spin n-axis
n_axis=A_spin*n_axis ! The spin part of A_total acts on e_y. That gives you the n-field
!!!!!! END OF EXTRA USER DEFINED

P=>P%NEXT
DNO_DX(1)=N_AXIS%X(1).SUB.'001'
DNO_DX(2)=N_AXIS%X(2).SUB.'001'
DNO_DX(3)=N_AXIS%X(3).SUB.'001'
DNO_DX2(1)=2*N_AXIS%X(1).SUB.'002'
DNO_DX2(2)=2*N_AXIS%X(2).SUB.'002'
DNO_DX2(3)=2*N_AXIS%X(3).SUB.'002'
WRITE(MF,'(a8,1x,8(1x,E15.8))') P%MAG%NAME(1:8),BETA_XX,BETA_XY,DNO_DX,DNO_DX2

ENDDO
CLOSE(MF)

```

8.2.3 Stroboscopic Average

One can check the spin normal form against a numerical average. In fact, the existence of a normal form is directly connected to one-turn averages. The word “Stroboscopic Average”, in the context

of a map based theory, is an unfortunate locution introduced by Hoffstätter and Heinemann. It is a simple turn by turn average.

Here is that code that does it:

```

9 format(' (a20,3(1x,E15.8))')
2001 call stroboscopic_average(als,xs0,xst,1,STATE,n,nh,N_AXIS_avep)
2002 call stroboscopic_average(als,xs0m,xstm,1,STATE,n,nh,N_AXIS_avep)
write(6,*)" $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ "
WRITE(6,9) " Numerical dn/dy =", (N_AXIS_avep%x(:)-N_AXIS_avep%x(:))/dx/2.d0
WRITE(6,9) " Analytical dn/dy =", DNO_DX
WRITE(6,9) " Numerical d2n/dy2 =", (N_AXIS_avep%x(:)+N_AXIS_avep%x(:)-2*n0(:))/dx**2
WRITE(6,9) "Analytical d2n/dy2 =", DNO_DX2
write(6,*) " more -> 1 for yes"
read(5,*)i
if(i==1) goto 2001

```

The routine that performs this average (a toy routine really) is located in the Sr_spin.f90 file.

```

SUBROUTINE stroboscopic_average(ring,xs0,xst,pos,mstate0,nturn,kp,n)
  IMPLICIT NONE
  TYPE(layout),target,INTENT(INOUT):: RING
  type(probe) , intent(inout) :: xs0,xst
  TYPE(INTERNAL_STATE) mstate0,mstate
  integer, intent(in) :: kp,pos,nturn
  integer i,k,imax,nd2
  type(spino) n
  real(dp) norm,norm0,n0(3),theta0

  mstate=mstate0+spin0
  nd2=6
  if(mstate%nocavity) nd2=4
  do k=1,nturn
    call track_probe(ring,xs0,mstate,node1=pos) !,fibre2=3)

    do i=1,3
      xst%s(i)%x=xs0%s(i)%x+xst%s(i)%x ! <---- Stroboscopic average
    enddo

    if(mod(k,kp)==0) then ! kp
      write(6,*) k,"$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"
      do i=1,3
        norm=root(xst%s(1)%x(i)**2+xst%s(2)%x(i)**2+xst%s(3)%x(i)**2)
        if(norm>=zero) then
          norm=1.d0/norm
        endif
        write(6,*) 1.d0/norm
        write(6,*) xst%s(1)%x(i)*norm,xst%s(2)%x(i)*norm,xst%s(3)%x(i)*norm
      enddo
    endif ! kp
  enddo

```

```
norm0=zero
do i=1,3
  norm=root(xst%s(1)%x(i)**2+xst%s(2)%x(i)**2+xst%s(3)%x(i)**2)
  if(norm>=norm0) then
    imax=1
    n%x(1)=xst%s(1)%x(i)/norm
    n%x(2)=xst%s(2)%x(i)/norm
    if(abs(n%x(2))>=abs(n%x(1))) imax=2
    n%x(3)=xst%s(3)%x(i)/norm
    if(abs(n%x(3))>=abs(n%x(2))) imax=3
    if(n%x(imax)<0) n%x=-n%x
    norm0=norm
  endif
enddo

end SUBROUTINE stroboscopic_average
```


Chapter 9. Beam Beam Kick

The beam beam kick requires a NODE_LAYOUT, say for example, ALS%t.

A beam-beam kick of type Beam_beam_node is given by:

```
TYPE BEAM_BEAM_NODE
  REAL(DP), POINTER :: S
  REAL(DP), POINTER :: SX,SY,FK
  REAL(DP), POINTER :: XM,YM ! trivial transverse displacement: not needed
  REAL(DP), POINTER :: BBK(:) ! kick on closed orbit
  REAL(DP), POINTER :: A(:) ! patch angles
  REAL(DP), POINTER :: D(:) ! patch translation
  INTEGER, POINTER :: A_X1,A_X2 ! patch inverse propagator (Should not be needed)
  LOGICAL(LP), POINTER :: PATCH ! true= patch
END TYPE BEAM_BEAM_NODE
```

and is located on a node “t” as “t%bb”. In PTC, if the beam is round (actually nearly round), the kick of the x-momentum is given by:

$$\Delta p_x = \frac{(x - xm) fk}{(x - xm)^2 + (y - ym)^2} \left(1 - \exp \left(- \frac{\{(x - xm)^2 + (y - ym)^2\}}{\sigma_x^2 + \sigma_y^2} \right) \right) \quad (9.1)$$

This defines the variables, xm,ym,fk, σ_x and σ_y .

The variables A(:) and B(:) as well as the discrete patches A_X1 and A_X2 must be set by the user to move the kick somewhere else. The logical PATCH is also needed if the patches are too be used.

9.1 Beam-Beam Flags

if(associated(t%bb).and.dobb.andand.do_beam_beam) then **Input settings**

1. dobb = **true** \implies Parameter set to true. A user who never uses beam-beam may want to set this to false. A good compiler will perhaps ignore all statements containing dobb.
2. do_beam_beam = **false** \implies Beam is by default turned off by this global variable.
3. t%bb%patch = **false** \implies If the user places the beam-beam kick as some arbitrary location close to the entrance of a node, then a patch must be used. However the flag patch of this kick must be true. “t” is the integration node on which the beam-beam kick sits.

9.2 Approximately locating the beam-beam kick: standard rubbish

The beam-beam kick is located on an integration node of type t%cas=case0, i.e., on a regular integration step. There cannot beam-beam kick on patches or fake fringe fields nodes. Furthermore, if beam-beam patches are necessary, it is advisable to call the routines of section Sec. (3.3) to fill in the geometrical information on each node. (It can be located on a type t%cas=caset. There are special types which replace a case0 type with a Taylor map.)

```
fk=.00000005d0
sx=.0004d0
sy=.0004d0
```

```

loc=0
s=3.00d0
call s_locate_beam_beam(als,s,loc,t,f)

if(found) then
  t%bb%fk=fk
  t%bb%sx=sx
  t%bb%sy=sy

  write(6,*) t%a          ! node entrance position
  write(6,*) t%ent(1,1:3) ! node entrance e_1 vector
  write(6,*) t%ent(2,1:3) ! node entrance e_2 vector
  write(6,*) t%ent(3,1:3) ! node entrance e_3 vector
  write(6,*) " s variable of node and following node "
  write(6,*) t%s(1),t%next%s(1)

endif

```

The routine `s_locate_beam_beam` is a cheap routine below the dignity of a true user of PTC. It is the kind of rubbish one would find in typical tracking code. In normal tracking code, the distance “*s*” along the “ideal design orbit” is a good locator of position. PTC makes a total mockery of this concept, or more precisely, of the people of still think that we need an ideal orbit to track a particle. Nevertheless, for standard lattice, i.e., no patch or tiny patches needed (say the LHC), the *s*-variable is still relevant.

The routine can also accept “loc” which is the integer index of the integration node where the beam-beam kick is located.

N.B. If nothing else is done, the beam-beam kick is placed at the entrance of the node. The call `FIND_PATCH(t%a,t%ent,o ,mid,D,ANG)` needs to be invoked to place the beam-beam kick

9.3 Exactly locating the beam-beam kick: `locate_beam_beam(r,o,m,t,f)`

In general one might know exactly where the beam-beam kick is located. This is certainly true if both rings of the collider are in PTC during the run.

```

write(6,*) " Using the actual location of the beam-beam kick "
call locate_beam_beam(als,o,m,t,f)
  call FIND_PATCH(t%a,t%ent,o ,m,t%bb%d,t%bb%a)
  write(6,*) " displacements "
  write(6,*) t%bb%d
  write(6,*) " angles "
  write(6,*) t%bb%a
  t%bb%PATCH=.true.
do_beam_beam=my_true

```

The centre of the beam-beam kick is located at a point $O(3)$ in the global frame. The orientation of the kick is given at $M(3,3)$. For example, the x-direction of the beam-beam kick is given by:

$$\vec{e}_x = M(1,1)\vec{i} + M(1,2)\vec{j} + M(1,3)\vec{k} \quad (9.2)$$

`Find_patch` computes the displacements and rotations necessary to place the beam-beam in the proper place.

Chapter 10. Small Example Programs

```
program ptc_geometry
use madx_ptc_module
use pointer_lattice
implicit none

character*48 :: command_gino

!-----

Lmax = 10.d0
use_info = .true.

!== user stuff : one layout necessary before starting GUI
call ptc_ini_no_append

CALL READ_AND_APPEND_VIRGIN_general(M_U,"ALS_FLAT.DAT")
m_U%start%name="ALS "
write(6,*) "Making an node layout"
CALL MAKE_NODE_LAYOUT(m_U%start)
write(6,*) " "
write(6,*) "Making an orbit layout"
call ORBIT_MAKE_NODE_LAYOUT(m_U%start,my_true)
write(6,*) m_U%start%t%ORBIT_LATTICE%ORBIT_N_NODE, " Orbit Nodes "

call put_an_s_aperture(0.00001d0,0.00002d0,0.5d0)
call remove_temporarily_an_aperture

!CALL TRACK_A_RAY(MY_FALSE,MY_FALSE)
call Track_and_test_aperture
!call Track_a_ray_around_ring
!call Track_a_ray_orbit
!CALL TRACK_A_RAY(MY_TRUE,MY_FALSE)
!CALL TRACK_A_RAY(MY_TRUE,MY_FALSE)
!CALL TRACK_A_RAY(MY_TRUE,MY_FALSE)
!CALL TRACK_A_RAY(MY_TRUE,MY_TRUE)

999 command_gino = "opengino"
call context(command_gino) ! context makes them capital
call call_gino(command_gino)
!== vaguely necessary baloney ==
command_gino = "closegino"
call call_gino(command_gino)
call ptc_end
end program ptc_geometry
```

10.1 Subroutine Track_a_ray(RADIATION_FLAG,SPIN_FLAG)

```
subroutine Track_a_ray(RADIATION_FLAG,SPIN_FLAG)
```

```

use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
REAL(dp) X0(6),X(6),R,DT
TYPE(PROBE) XS
TYPE(TEMPORAL_PROBE) XT
TYPE(INTERNAL_STATE) STATE
LOGICAL(LP) RADIATION_FLAG,SPIN_FLAG
integer i

als=>M_U%start
STATE=DEFAULT+TOTALPATH

IF(RADIATION_FLAG) THEN
  STATE=STATE+RADIATIONO
  WRITE(6,*) "RADIATION IS ON "
ENDIF
IF(SPIN_FLAG) THEN
  STATE=STATE+SPINO
  WRITE(6,*) "SPIN IS ON "
ENDIF

write(6,*) " "
WRITE(6,*) " "
WRITE(6,*) " GIVE INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R "
READ(5,*) R
XO(1:6)=R
write(6,*) "ONE TURN AROUND WITH FIBRES -> NO RADIATION AND NO SPIN"
!EXAMPLE 1.1
! ONE TURN AROUND WITH FIBRES -> NO RADIATION AND NO SPIN
X=XO
CALL TRACK(ALS,X,1,STATE)
IF(CHECK_STABLE) THEN
  WRITE(6,'(6(1x,E15.8))') X
ELSE
  WRITE(6,"(A255)") MESSAGELOST
  WRITE(6,"(A24)") LOST_FIBRE%MAG%NAME
  WRITE(6,*) " FIBRE POSITION ",LOST_FIBRE%POS
ENDIF
!CALL RESET_APERTURE_FLAG
write(6,*) " "
write(6,*) "ONE TURN AROUND WITH WRAPPED PROBE -> NO SPIN"
!EXAMPLE 1.2
! ONE TURN AROUND WITH WRAPPED PROBE -> NO SPIN
X=XO
CALL TRACK_PROBE_X(ALS,X,STATE,FIBRE1=1)
IF(CHECK_STABLE) THEN

```

```

WRITE(6,'(6(1x,E15.8))') X
ELSE
  WRITE(6,"(A255)") MESSAGELOST
  WRITE(6,"(A24)") LOST_FIBRE%MAG%NAME
  WRITE(6,*) " FIBRE POSITION ",LOST_FIBRE%POS
  WRITE(6,*) " NODE POSITION ",LOST_NODE%POS
ENDIF

write(6,*) " "
write(6,*) "ONE TURN AROUND WITH PROBE -> SPIN AND RADIATION POSSIBLE"
!EXAMPLE 1.3
! ONE TURN AROUND WITH PROBE -> SPIN AND RADIATION POSSIBLE
XS=X0
CALL TRACK_PROBE(ALS,XS,STATE,FIBRE1=1)
IF(CHECK_STABLE) THEN
WRITE(6,'(6(1x,E15.8))') XS%X
WRITE(6,*) " SPINORS "
WRITE(6,'(6(1x,E15.8))') XS%S(1)
WRITE(6,'(6(1x,E15.8))') XS%S(2)
WRITE(6,'(6(1x,E15.8))') XS%S(3)
ELSE
  WRITE(6,"(A255)") MESSAGELOST
  WRITE(6,"(A24)") LOST_FIBRE%MAG%NAME
  WRITE(6,*) " FIBRE POSITION ",LOST_FIBRE%POS
  WRITE(6,*) " NODE POSITION ",LOST_NODE%POS
ENDIF

call FILL_SURVEY_DATA_IN_NODE_LAYOUT(ALS)

write(6,*) " "
write(6,*) "TIME TRACKING "

XS=X0
XT%XS=XS
XT%POS=0.DO
XT%NODE=>ALS%T%START
XT%DS=0.DO
  DT=(0.19683851D+03+1.D-3)

CALL TRACK_time(XT,DT,STATE)
IF(CHECK_STABLE) THEN
WRITE(6,*) " TIME TRACKING ", dt
WRITE(6,'(6(1x,E15.8))') XT%XS%X
WRITE(6,*) " SPINORS "
WRITE(6,'(6(1x,E15.8))') XT%XS%S(1)
WRITE(6,'(6(1x,E15.8))') XT%XS%S(2)
WRITE(6,'(6(1x,E15.8))') XT%XS%S(3)
WRITE(6,*) " POSITION "
WRITE(6,'(6(1x,E15.8))') XT%POS

```

```

WRITE(6,*) XT%NODE%PARENT_FIBRE%POS, XT%NODE%PARENT_FIBRE%MAG%NAME
ELSE
  WRITE(6,"(A255)") MESSAGELOST
  WRITE(6,"(A24)") LOST_FIBRE%MAG%NAME
  WRITE(6,*) " FIBRE POSITION ",LOST_FIBRE%POS
  WRITE(6,*) " NODE POSITION ",LOST_NODE%POS
ENDIF

end subroutine Track_a_ray

```

10.2 Subroutine Track_a_ray_time(RADIATION_FLAG,SPIN_FLAG)

```

subroutine Track_a_ray_time(RADIATION_FLAG,SPIN_FLAG)
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
REAL(dp) X0(6),X(6),R,DT
TYPE(PROBE) XS
TYPE(TEMPORAL_PROBE) XT
TYPE(INTERNAL_STATE) STATE
LOGICAL(LP) RADIATION_FLAG,SPIN_FLAG
integer i

als=>M_U%start
STATE=DEFAULT+TOTALPATH

IF(RADIATION_FLAG) THEN
  STATE=STATE+RADIATIONO
  WRITE(6,*) "RADIATION IS ON "
ENDIF
IF(SPIN_FLAG) THEN
  STATE=STATE+SPINO
  WRITE(6,*) "SPIN IS ON "
ENDIF

WRITE(6,*) " "
WRITE(6,*) " GIVE INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R "
READ(5,*) R
X0(1:6)=R

call FILL_SURVEY_DATA_IN_NODE_LAYOUT(ALS)

XS=X0
XT%XS=XS

```

```

XT%POS=0.DO
XT%NODE=>ALS%T%START
XT%DS=0.DO
  DT=(0.19683851D+03+1.D-3)/10

  r=zero

do i=1,10
CALL TRACK_time(XT,DT,STATE)
IF(CHECK_STABLE) THEN
  write(6,*) "T = ",i*dt
  WRITE(6,*) " POSITION "
  WRITE(6,'(6(1x,E15.8))') XT%POS
  WRITE(6,*) XT%NODE%PARENT_FIBRE%POS, XT%NODE%PARENT_FIBRE%MAG%NAME
ENDIF
enddo
IF(CHECK_STABLE) THEN
WRITE(6,*) " TIME TRACKING ALL THE WAY ", 10*dt
WRITE(6,'(6(1x,E15.8))') XT%XS%X
WRITE(6,*) " SPINORS "
WRITE(6,'(6(1x,E15.8))') XT%XS%S(1)
WRITE(6,'(6(1x,E15.8))') XT%XS%S(2)
WRITE(6,'(6(1x,E15.8))') XT%XS%S(3)
WRITE(6,*) " POSITION "
WRITE(6,'(6(1x,E15.8))') XT%POS
WRITE(6,*) XT%NODE%PARENT_FIBRE%POS, XT%NODE%PARENT_FIBRE%MAG%NAME
ELSE
  WRITE(6,"(A255)") MESSAGELOST
  WRITE(6,"(A24)") LOST_FIBRE%MAG%NAME
  WRITE(6,*) " FIBRE POSITION ",LOST_FIBRE%POS
  WRITE(6,*) " NODE POSITION ",LOST_NODE%POS
ENDIF

end subroutine Track_a_ray_time

```

10.3 Subroutine Track_a_ray_orbit

```

subroutine Track_a_ray_orbit
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
REAL(dp) X0(6),X(6),R
TYPE(INTERNAL_STATE) STATE
integer i
als=>M_U%start

state=default

WRITE(6,*) " "

```

```
WRITE(6,*) " GIVE INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R "
READ(5,*) R
X0(1:6)=R
```

```
WRITE(6,*) " Tracking "
als%t%ORBIT_LATTICE%ORBIT_USE_ORBIT_UNITS=my_false
```

```
X=X0
do i=1,als%t%ORBIT_LATTICE%ORBIT_N_NODE
  call ORBIT_TRACK_NODE(i,x,state)
  if(x(1)> 1000.d0) exit
enddo
write(6,*) x
```

```
X=X0
CALL TRACK_PROBE_X(ALS,X,STATE,FIBRE1=1)
write(6,*) x
```

```
end subroutine Track_a_ray_orbit
```

10.4 Subroutine Track_a_ray_around_ring

```
subroutine Track_a_ray_around_ring
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
REAL(dp) X0(6),X(6),R
TYPE(INTERNAL_STATE) STATE
integer i
logical CORRECT
type(fibre), pointer :: p

als=>M_U%start
state=default

write(6,*) " Do you want a correct implementation of exceptions -> yes=true "
read(5,*) CORRECT
WRITE(6,*) " "
WRITE(6,*) " GIVE A 'BIG' INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R (example 0.1) "
READ(5,*) R
X0(1:6)=R

x=x0
p=>als%start
do while(.not.associated(p%next,als%start)) ! this is one turn minus the last element

  call track(x,state,fibre1=p,fibre2=p%next)
  if(.not.check_stable.and.CORRECT) then
    write(6,*) "In the loop"
```



```

        write(6,"(a255)")  messagelost
        write(6,*)  lost_fibre%pos,lost_fibre%mag%name
        write(6,*)  "Unstable ray"
        WRITE(6,'(6(1x,E15.8))') Xlost
        exit
    endif
p=>p%next
enddo

if(.not.check_stable) then
    write(6,*)  "End of loop"
    write(6,"(a255)")  messagelost
    write(6,*)  lost_fibre%pos,lost_fibre%mag%name
    write(6,*)  "Unstable ray"
    WRITE(6,'(6(1x,E15.8))') Xlost
else
    write(6,*)  "stable ray"
    WRITE(6,'(6(1x,E15.8))') X
endif

x=x0
call track(x,state,fibre1=als%start,fibre2=als%end) ! this is one turn minus the last element

if(.not.check_stable) then
    write(6,*)  "Using one call to Track"
    write(6,"(a255)")  messagelost
    write(6,*)  lost_fibre%pos,lost_fibre%mag%name
    write(6,*)  "Unstable ray"
    WRITE(6,'(6(1x,E15.8))') Xlost
else
    write(6,*)  "stable ray"
    WRITE(6,'(6(1x,E15.8))') X
endif

end subroutine  Track_a_ray_around_ring

```

10.5 Subroutines to test the apertures flags

```

subroutine  Track_and_test_aperture
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
REAL(dp) X0(6),X(6),R
TYPE(INTERNAL_STATE) STATE

als=>M_U%start
state=default
call put_back_an_aperture

```

```

write(6,*) "aperture => True or false "
read(5,*) aperture_flag
write(6,*) " Check_madx_aperture (magnet aperture) => True or false "
read(5,*) check_madx_aperture
write(6,*) " Give value of the absolute_aperture "
read(5,*) absolute_aperture

WRITE(6,*) " GIVE A 'BIG' INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R (example 0.1) "
READ(5,*) R
X0(1:6)=R
x=x0
  call track(x,state,fibre1=als%start)

  if(.not.check_stable) then
    write(6,*) "Using one call to Track"
    write(6,"(a255)") messagelost
    write(6,*) lost_fibre%pos,lost_fibre%mag%name
    write(6,*) "Unstable ray"
    WRITE(6,'(6(1x,E15.8))') Xlost
  else
    write(6,*) "stable ray"
    WRITE(6,'(6(1x,E15.8))') X
  endif
call remove_temporarily_an_aperture

end subroutine  Track_and_test_aperture

!=====
subroutine  put_an_aperture(x,y)
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
integer pos
type(fibre), pointer :: p
real(dp) x,y

als=>M_U%start
als%last=>als%start
als%lastpos=als%last%pos

call move_to(als,p,"BEND",pos)
write(6,*) pos,p%mag%name,p%mag%vorname
call move_to(als,p,"BEND",pos)
write(6,*) pos,p%mag%name,p%mag%vorname

call alloc(p%mag%p%aperture)

```

```

call alloc(p%magp%p%aperture)

p%mag%p%aperture%kind=2
p%mag%p%aperture%x=x
p%mag%p%aperture%y=y
p%mag%p%aperture%dx=0.d0
p%mag%p%aperture%dy=0.d0
write(6,*) " Aperture size ",x,y

end subroutine  put_an_aperture

subroutine  remove_temporarily_an_aperture
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
integer pos
type(fibre), pointer :: p

als=>M_U%start
als%last=>als%start
als%lastpos=als%last%pos

call move_to(als,p,"BEND",pos)
call move_to(als,p,"BEND",pos)

p%mag%p%aperture%kind=-p%mag%p%aperture%kind
p%magp%p%aperture%kind=-p%magp%p%aperture%kind

end subroutine  remove_temporarily_an_aperture

subroutine  put_back_an_aperture
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
integer pos
type(fibre), pointer :: p

als=>M_U%start

als%last=>als%start
als%lastpos=als%last%pos
call move_to(als,p,"BEND",pos)
call move_to(als,p,"BEND",pos)

p%mag%p%aperture%kind=iabs(p%mag%p%aperture%kind)
p%magp%p%aperture%kind=iabs(p%magp%p%aperture%kind)

end subroutine  put_back_an_aperture

```

```

subroutine put_an_s_aperture(x,y,r)
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
integer pos,i
type(fibre), pointer :: p
type(MADX_APERTURE), pointer :: a
real(dp) x,y,r
integer s

als=>M_U%start
als%last=>als%start
als%lastpos=als%last%pos

call move_to(als,p,"QFA1",pos)
write(6,*) pos,p%mag%name,p%mag%vorname

s=nint(r*p%mag%p%nst)

write(6,*) " Aperture put at position ",s

call alloc(p%mag%p%a,p%mag%p%nst+1)
call alloc(p%magp%p%a,p%mag%p%nst+1)

do i=1,p%mag%p%nst+1
a=> p%mag%p%a(i)%aperture
if(i==s+1) then
  a%x=x
  a%y=y
else
  a%x=absolute_aperture
  a%y=absolute_aperture
endif
a%kind=2
a%dx=0.d0
a%dy=0.d0
p%magp%p%a(i)%aperture=a
enddo

write(6,*) " Aperture size ",x,y

end subroutine put_an_s_aperture

!=====
subroutine Track_and_test_s_aperture
use madx_ptc_module
use pointer_lattice

```

```

implicit none
type(layout), pointer :: ALS
REAL(dp) X0(6),X(6),R
real(sp) per
TYPE(INTERNAL_STATE) STATE

als=>M_U%start
state=default
!

write(6,*) "aperture => True or false "
read(5,*) aperture_flag
write(6,*) " Check_madx_aperture (magnet aperture) => True or false "
read(5,*) check_madx_aperture
write(6,*) " s_aperture_check ( Extended magnet aperture) => True or false "
read(5,*) s_aperture_check
write(6,*) " Give value of the absolute_aperture "
read(5,*) absolute_aperture

WRITE(6,*) " GIVE A 'BIG' INITIAL VALUE (ONE NUMBER) FOR X(1:6)= R (example 0.1) "
READ(5,*) R
X0(1:6)=R
x=x0
  call track_probe_x(x,state,fibre1=als%start)

  if(.not.check_stable) then
    write(6,*) "Using one call to Track"
    write(6,"(a255)") messagelost
    write(6,*) lost_fibre%pos,lost_fibre%mag%name
    write(6,*) " Position lost, beginning of element,  end of element "
    write(6,*) lost_node%s(1),lost_node%parent_fibre%t1%s(1),lost_node%parent_fibre%t2%next%s(1)
    per= 100*(lost_node%s(1)-lost_node%parent_fibre%t1%s(1))/lost_node%parent_fibre%mag%1
    write(6,*) " Lost at ", per," % down the magnet"
    write(6,*) "Unstable ray"
    WRITE(6,'(6(1x,E15.8))') Xlost
  else
    write(6,*) "stable ray"
    WRITE(6,'(6(1x,E15.8))') X
  endif
call remove_temporarily_an_aperture

end subroutine  Track_and_test_s_aperture

```

10.6 Subroutine displaying Radiation in PTC

```
!=====
```

```

subroutine Track_and_with_rad
use madx_ptc_module
use pointer_lattice
use gauss_dis
implicit none
type(layout), pointer :: ALS
real(dp) x(6),sig0(6),energy,deltap,sizes(6,6),bet(3),bet0,betf,t
type(probe) xs0,xs1
type(probe_8) xs
type(real_8) y(6)
type(damapspin) m,m1,m2
type(damap) id
type(internal_state) state,wrapped_state
type(fibre), pointer ::p
integer i,mf,j,n
type(normal_spin) nf
type(normalform) normal
real(dp) stoch(6,6),stochi(6,6),r(3),mat(6,6)

als=>M_U%start
state=default0+radiation0+envelope0

p=>als%start
do i=1,als%n
p%AG=.1d0 ! a_etienne
p=>p%next
enddo

sig0=1.d-6
sig0(6)=1.d-4

call MESS_UP_ALIGNMENT(als,SIG0,3.d0)

CALL INIT(state,2,0)

call alloc(m)
call alloc(m1)
call alloc(m2)
call alloc(xs)
call alloc(nf)
call alloc(id)
call alloc(y)
call alloc(normal)
nf%stochastic=my_true

!!!! Tracking with a wrapped command      !!!!!!!!!!!!!
wrapped_state=default0+radiation0

```

```

CALL FIND_ORBIT_x(als,X,wrapped_state,1.0e-5_dp,fibre1=1)

WRITE(6,'(A)') " Closed orbit with Radiation "
WRITE(6,'(6(1x,E15.8))') x
call GET_loss(als,energy,deltap)
write(6,'(a32,2(1x,E15.8))') "Energy loss: GEV and DeltaP/p0c ",energy,deltap

id=1
y=x+id
call TRACK_PROBE_X(ALS,y,wrapped_state, FIBRE1=1)
id=y

normal=id ! Normal is a regular Normalform type
write(6,*) "Tunes "
write(6,*) normal%tune(1:3)
write(6,*) "Damping decrements"
write(6,*) normal%damping(1:3)
!!!!!!!!!!!! end of Tracking with a wrapped command !!!!!!!!!!!!!

!!!!!!!!!!!! Tracking a probe_8 to get the map !!!!!!!!!!!!!
state=default0+radiation0+envelope0
xs0=x ! closed orbit intializing a probe
m=1 ! damapspin set to identity
xs=xs0+m ! Probe_8 = closed orbit probe + Identity

call track_probe(als,xs,state,fibre1=1)
m=xs ! damapspin = Probe_8
nf=m ! normal_spin = damapspin (Normalization including spin (if present) or radiation
! envelope if present. (Spin without radiation)
write(6,*) nf%n%tune(1:3)
write(6,*) nf%emittance
write(6,*) nf%s_ij0(1,1),nf%s_ij0(5,5)
!!!!!!!!!!!! end Tracking a probe_8 to get the map !!!!!!!!!!!!!
mat=m%m
!!!!!!!!!!!! Tracking a probe_8 to beam sizes around the ring !!!!!!!!!!!!!
call kanalnummer(mf,"beam_xx_inf.dat")

state=default0+radiation0+envelope0
xs0=x ! closed orbit intializing a probe
m=1 ! damapspin set to identity
xs=xs0+m ! Probe_8 = closed orbit probe + Identity

p=>als%start

call extract_beam_sizes(xs,nf%s_ij0,sizes)
write(mf,*) p%mag%name, sizes(1,1)

```

```

do i=1,als%n
  call track_probe(als,xs,state,fibre1=i,fibre2=i+1)
  call extract_beam_sizes(xs,nf%s_ij0,sizes)
  p=>p%next
  write(mf,*) p%mag%name, sizes(1,1)
enddo
close(mf)

!!!!!!!!!!!! Tracking a probe_8 to beam sizes around the ring using emittances !!!!!!!!!!!!!

call kanalnummer(mf,"beam_xx_inf_using_emittances.dat")

state=default0+radiation0+envelope0
xs0=x      ! closed orbit intializing a probe
m=nf%a_t   ! damapspin set to A
xs=xs0+m   ! Probe_8 = closed orbit probe + Identity

p=>als%start

bet(1)=(m%m%v(1).sub.'1')**2+(m%m%v(1).sub.'01')**2
bet(2)=(m%m%v(1).sub.'001')**2+(m%m%v(1).sub.'0001')**2
bet(3)=(m%m%v(1).sub.'00001')**2+(m%m%v(1).sub.'000001')**2
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
bet0=sizes(1,1)
write(mf,*) p%mag%name, sizes(1,1)

do i=1,als%n
  call track_probe(als,xs,state,fibre1=i,fibre2=i+1)
  m=xs
  bet(1)=(m%m%v(1).sub.'1')**2+(m%m%v(1).sub.'01')**2
  bet(2)=(m%m%v(1).sub.'001')**2+(m%m%v(1).sub.'0001')**2
  bet(3)=(m%m%v(1).sub.'00001')**2+(m%m%v(1).sub.'000001')**2
  sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
  p=>p%next
  write(mf,*) p%mag%name, sizes(1,1)
enddo
betf=sizes(1,1)
write(mf,*) " log of change after one turn and damping "
write(mf,*) log(betf/bet0)/2.d0,nf%n%damping(1)
close(mf)
!!!!!!!!!!!! Tracking a probe_8 to beam sizes around the ring using emittances !!!!!!!!!!!!!

call kanalnummer(mf,"beam_xx_inf_using_emittances_ripken.dat")

state=default0+radiation0+envelope0
xs0=x      ! closed orbit intializing a probe
m=nf%a_t   ! damapspin set to A
xs=xs0+m   ! Probe_8 = closed orbit probe + Identity

p=>als%start

```



```

id=(m%m.sub.1)**(-1)
bet(1)=(m%m%v(1).sub.'1')*(id%v(2).sub.'01')-(m%m%v(1).sub.'01')*(id%v(1).sub.'01')
bet(2)=(m%m%v(1).sub.'001')*(id%v(4).sub.'01')-(m%m%v(1).sub.'0001')*(id%v(3).sub.'01')
bet(3)=(m%m%v(1).sub.'00001')*(id%v(6).sub.'01')-(m%m%v(1).sub.'000001')*(id%v(5).sub.'01')
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
write(mf,*) p%mag%name, sizes(1,1)

do i=1,als%n
call track_probe(als,xs,state,fibre1=i,fibre2=i+1)
m=xs
id=(m%m.sub.1)**(-1)
bet(1)=(m%m%v(1).sub.'1')*(id%v(2).sub.'01')-(m%m%v(1).sub.'01')*(id%v(1).sub.'01')
bet(2)=(m%m%v(1).sub.'001')*(id%v(4).sub.'01')-(m%m%v(1).sub.'0001')*(id%v(3).sub.'01')
bet(3)=(m%m%v(1).sub.'00001')*(id%v(6).sub.'01')-(m%m%v(1).sub.'000001')*(id%v(5).sub.'01')
sizes(1,1)=(bet(1)*nf%emittance(1)+bet(2)*nf%emittance(2)+bet(3)*nf%emittance(3))
p=>p%next
write(mf,*) p%mag%name, sizes(1,1)
enddo
close(mf)

!!!!!!! End of Tracking a probe_8 to beam sizes around the ring using emittances and super-
stoch=nf%stoch
id=stoch
id=id**(-1)
stochi=id

x=0.d0
sizes=0.d0
write(6,*) " Give the number of turns > 1 million"
read(5,*) n
do i=1,n
x=matmul(mat,x)
x=matmul(stoch,x)
do j=1,3
t=RANF()
if(t>half) then
t=one
else
t=-one
endif
x(2*j-1)= x(2*j-1)+ t*nf%kick(j)
x(2*j) = x(2*j)+ t*nf%kick(j)
enddo
x=matmul(stochi,x)
sizes(1,1)=sizes(1,1)+x(1)**2
sizes(2,2)=sizes(2,2)+x(2)**2
sizes(1,3)=sizes(1,3)+x(1)*x(3)

enddo

```

```

sizes=sizes/n
write(6,*) sizes(1,1), sizes(2,2), sizes(1,3)
write(6,*) nf%s_ij0(1,1), nf%s_ij0(2,2), nf%s_ij0(1,3)
call kill(m)
call kill(m1)
call kill(m2)
call kill(xs)
call kill(nf)
call kill(id)
call kill(y)
call kill(normal)

end subroutine Track_and_with_rad

```

10.7 Subroutine Showing a Beam-Beam Kick

```

!=====
subroutine Track_with_beam_beam
use madx_ptc_module
use pointer_lattice
use gauss_dis
implicit none
type(layout), pointer :: ALS
type(fibre), pointer :: p
integer i,mf,j,n
type(normal_spin) nf
type(normalform) normal
real(dp) s,fk,sx,sy,d(3),OMEGA(3),ang(3),a(3),ent(3,3),o(3),m(3,3)
integer loc,pos
type(integration_node), pointer :: t
logical(lp) f

als=>M_U%start

call FILL_SURVEY_DATA_IN_NODE_LAYOUT(ALS)

! TYPE BEAM_BEAM_NODE
! REAL(DP), POINTER :: S
! REAL(DP), POINTER :: SX,SY,FK
! REAL(DP), POINTER :: XM,YM ! trivial transverse displacement: not needed
! REAL(DP), POINTER :: BBK(:) ! kick on closed orbit
! REAL(DP), POINTER :: A(:) ! patch angles
! REAL(DP), POINTER :: D(:) ! patch translation
! INTEGER, POINTER :: A_X1,A_X2 ! patch inverse propagator (not needed)
! LOGICAL(LP), POINTER :: PATCH ! true= patch
! END TYPE BEAM_BEAM_NODE

```

```

!     TYPE(BEAM_BEAM_NODE), POINTER :: BB

fk=.00000005d0
sx=.0004d0
sy=.0004d0
loc=0
s=196.838521200000d0*two/12.d0+3.00d0
call s_locate_beam_beam(als,s,loc,t,f)

if(f) then
  t%bb%fk=fk
  t%bb%sx=sx
  t%bb%sy=sy

  write(6,*) t%a           ! node entrance position
  write(6,*) t%ent(1,1:3)  ! node entrance e_1 vector
  write(6,*) t%ent(2,1:3)  ! node entrance e_2 vector
  write(6,*) t%ent(3,1:3)  ! node entrance e_3 vector
  write(6,*) " s variable of node and following node "
  write(6,*) t%s(1),t%next%s(1)

endif

do_beam_beam=my_true

t%bb%d=0.00d0
t%bb%d(1)=8.7d-002
t%bb%d(3)=0.342d0
t%bb%a=0.0d0
t%bb%a(2)=0.523598775598299d0
t%bb%PATCH=.true.

      CALL INVERSE_FIND_PATCH(t%a,t%ent, t%bb%d,t%bb%a,o,m)

      write(6,*) "   out "
      write(6,*) o
      write(6,*) m

      call FIND_PATCH(t%a,t%ent,o ,m,D,ANG)
      write(6,*) " d "
      write(6,*) d
      write(6,*) t%bb%d
      write(6,*) " ang "
      write(6,*) ang
      write(6,*) t%bb%a

call survey(als)

```

```

call FILL_SURVEY_DATA_IN_NODE_LAYOUT(ALS)

write(6,*) "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"
write(6,*) " Using the actual location of the beam-beam kick "
call locate_beam_beam(als,o,m,t,f)
    call FIND_PATCH(t%a,t%ent,o ,m,t%bb%d,t%bb%a)
    write(6,*) " displacements "
    write(6,*) t%bb%d
    write(6,*) " angles "
    write(6,*) t%bb%a
    t%bb%PATCH=.true.
    do_beam_beam=my_true
1000 continue

end subroutine Track_with_beam_beam

```

10.8 Subroutine Track_with_rf_modulation

```

subroutine Track_with_rf_modulation
use madx_ptc_module
use pointer_lattice
use gauss_dis
implicit none
type(layout), pointer :: ALS
type(fibre), pointer :: p
integer i,mf,j,n
type(normal_spin) normal
type(vecresonance) vecres
type(probe) Ray
type(probe_8) Ray_8
type(damapspin) m
integer pos
type(internal_state) state
real(dp) DC_ac,A_ac,theta_ac,x(6),deps

DC_ac=1.0d0
A_ac=1.d-3
theta_ac=0.01d0

als=>M_U%start

als%last=>als%start
als%lastpos=als%last%pos

call move_to(als,p,"BEND",pos)
write(6,*) pos,p%mag%name,p%mag%vorname

    if(.not.associated(P%MAG%DC_ac)) then
        allocate(P%MAG%DC_ac)

```

```

        allocate(P%MAG%A_ac)
        allocate(P%MAG%theta_ac)

        allocate(P%MAGP%DC_ac)
        allocate(P%MAGP%A_ac)
        allocate(P%MAGP%theta_ac)
        CALL alloc(P%MAGP%DC_ac)
        CALL alloc(P%MAGP%A_ac)
        CALL alloc(P%MAGP%theta_ac)

        P%MAG%DC_ac=DC_ac
        P%MAG%A_ac=A_ac
        P%MAG%theta_ac=theta_ac*twopi
        P%MAGP%DC_ac=DC_ac
        P%MAGP%A_ac=A_ac
        P%MAGP%theta_ac=theta_ac*twopi
        P%MAG%slow_ac=.true.
        P%MAGP%slow_ac=.true.
    ENDIF
    state=default0+only_4d0
CALL FIND_ORBIT_x(als,X,state,1.0e-5_dp,fibre1=1)
Ray=x
ray%ac%om=0.01d0
ray%ac%x(1)=1.d0    ! fake phasor
ray%ac%x(2)=0.d0    !   fake phasor

    state=state+modulation0
    write(6,*) "Tracking data in modulation.dat"
    call kanalnummer(mf,"modulation.dat")
    write(6,*) ray%x
    write(6,*) ray%ac%x
        WRITE(mf,'(4(1x,E15.8))') ray%x(1:4)
    do i=1,10000
        call TRACK_PROBE(als,ray,state, FIBRE1=1)
        WRITE(mf,'(4(1x,E15.8))') ray%x(1:4)
    enddo
    close(mf)
    write(6,*) "Normal form data in modulation.dat"
    call kanalnummer(mf,'modulation_result.txt')
    write(6,*) ray%x
    write(6,*) ray%ac%x

    CALL INIT(state,3,0)

    call alloc(m)
    call alloc(ray_8)
    call alloc(normal)
    call alloc(vecres)
    Ray=x
    ray%ac%om=0.001d0

```

```

ray%ac%x(1)=1.d0
ray%ac%x(2)=0.d0

m=1
ray_8=ray+m
  Write(mf,*) " Initial Probe_8 Ray_8 "
  call print(ray_8,mf)
  call TRACK_PROBE(als,ray_8,state, FIBRE1=1)
Write(mf,*) " Final Probe_8 Ray_8 "
call print(ray_8,mf)
m=ray_8
  Write(mf,*) " Final damapspin m"
  call print(m,mf)
normal=m

  Write(mf,*) " Tunes including RF modulation"
  write(mf,*) normal%n%tune
  vecres=normal%n%normal%nonlinear
  Write(mf,*) " "
  Write(mf,*) " Sine Part of Vector Field of Normal Form in resonance basis (Tunes) "
  Write(mf,*) " including RF modulation "
  deps=1.d-8
  call print(vecres%sin,mf,deps)

close(mf)

call kill(m)
call kill(ray_8)
call kill(normal)
call kill(vecres)

end subroutine Track_with_rf_modulation

```

10.9 Subroutine Track_a_ray_with_spin

```

!=====
subroutine Track_a_ray_with_spin
use madx_ptc_module
use pointer_lattice
implicit none
type(layout), pointer :: ALS
REAL(dp) x0(6),X(6),sig0(6),BETA_XX,BETA_XY,theta0,NO(3),DNO_DX(3),DNO_DX2(3),dx
type(probe_8) xs
type(probe) xs0,XST,xs0m,XSTm, xs02,XST2,xs0m2,XSTm2
type(real_8) ray8(6)
TYPE(INTERNAL_STATE) STATE
type(fibre), pointer :: p
type(damap) id,one_turn,A

```

```

integer i,mf,n,nh
type(normalform) nf
type(damapspin) id_s,one_turn_S,a_S,A_f,A_spin,A_l,A_nl,NORMALSPIN
type(normal_spin) nf_S
TYPE(spinor_8) N_AXIS
TYPE(spinor) N_AXIS_avep,N_AXIS_avem
als=>M_U%start
STATE=DEFAULT0+only_4d0

STATE=STATE+SPINO
WRITE(6,*) "SPIN IS ON "

als=>m_U%start
sig0=0.00001d0
call MESS_UP_ALIGNMENT(als,SIG0,5.d0)

x0=0.d0
state=default0+nocavity0+DELTA0

CALL FIND_ORBIT(ALS,x0,1,STATE,c_1d_5)

write(6,*) "CLOSED ORBIT "
WRITE(6,'(6(1x,E15.8))') x0

CALL INIT(STATE,1,0)

call alloc(ray8)
call alloc(id,one_turn,A)
call alloc(nf)

id=1

ray8=x0+id

call track(als,ray8,1,state)

one_turn=ray8
nf=one_turn

write(6,*) " TUNES "
WRITE(6,'(2(1x,E15.8))') nf%tune(1:2)

RAY8=x0+NF%A_T

write(6,*) " Regular Twiss in file LATTICE.TXT "

```

```

CALL KANALNUMMER(MF,"LATTICE.TXT")

P=>ALS%START
  A=RAY8
  BETA_XX=(A%V(1).SUB.'1')**2+(A%V(1).SUB.'01')**2
  BETA_XY=(A%V(1).SUB.'001')**2+(A%V(1).SUB.'0001')**2
  WRITE(MF,*) P%MAG%NAME,BETA_XX,BETA_XY

DO I=1,ALS%N
  call track(als,ray8,I,I+1,state)
  A=RAY8
  BETA_XX=(A%V(1).SUB.'1')**2+(A%V(1).SUB.'01')**2
  BETA_XY=(A%V(1).SUB.'001')**2+(A%V(1).SUB.'0001')**2

  P=>P%NEXT
  WRITE(MF,'(a8,1x,2(1x,E15.8))') P%MAG%NAME(1:8),BETA_XX,BETA_XY
ENDDO
CLOSE(MF)

  call kill(ray8)
  call kill(id)
  call kill(nf)

state=default0+DELTA0+SPINO
XS0=x0
  CALL FIND_ORBIT_probe_spin(ALS,XS0,STATE,c_1d_5,FIBRE1=1,theta0=theta0)
write(6,*) "CLOSED ORBIT "
WRITE(6,'(6(1x,E15.8))') XS0%X
write(6,*) " NO "
WRITE(6,*) XS0%S(1)%X
WRITE(6,*) " THETA0"
WRITE(6,'(1x,E15.8)') THETA0
NO=XS0%S(1)%X

WRITE(6,*)" Setting the second spin axis to e_y : XS0%S(2)=2 "

XS0%S(2)=2

CALL TRACK_PROBE(ALS,XS0,STATE,FIBRE1=1)

WRITE(6,*)" tracking n0: should be returning to itself "
WRITE(6,'(3(1x,E15.8))') XS0%S(1)%X
WRITE(6,*)" tracking y_y: should not be invariant "
WRITE(6,'(3(1x,E15.8))') XS0%S(2)%X

```



```

WRITE(6,*)" "
WRITE(6,*)" Doing a Spin Normal Form for n-vector "

      CALL INIT(STATE,2,0)

      call alloc(ray8)
      call alloc(id_S,one_turn_S,A_S,A_f,A_spin,A_l,A_nl,NORMALSPIN)
      call alloc(nf_S)
      ID_S=1

      XS=XS0+ID_S

WRITE(6,*)" Computing a one-turn spin taylor map "

CALL TRACK_PROBE(ALS,XS,STATE,FIBRE1=1)

ID_S=XS

WRITE(6,*)" Doing a Spin-Orbital Normal Form: NF_S=ID_S "

NF_S=ID_S
A_S=NF_S%A_T
call factor(A_S,A_f,A_spin,A_l,A_nl)

      WRITE(16,*) " FIXED POINT "
      CALL PRINT(A_f,16)
      WRITE(16,*) " SPIN PART "
      CALL PRINT(A_spin,16)
      WRITE(16,*) " ORBITAL LINEAR "
      CALL PRINT(A_L,16)
      WRITE(16,*) " ORBITAL NONLINEAR "
      CALL PRINT(A_NL,16)

      NORMALSPIN=A_S**(-1)*ID_S*A_S
      WRITE(16,*) " NORMAL FORM "
      CALL PRINT(NORMALSPIN,16)

N_AXIS=2
      WRITE(16,*) " NORMALIZED N-AXIS "
      CALL PRINT(N_AXIS,16)
      WRITE(16,*) " N-AXIS "
      N_AXIS= A_spin*N_AXIS
      CALL PRINT(N_AXIS,16)
      NO(1)=N_AXIS%X(1).SUB.'0'
      NO(2)=N_AXIS%X(2).SUB.'0'
      NO(3)=N_AXIS%X(3).SUB.'0'
      WRITE(6,*) " n0 computed from N-AXIS "
      WRITE(6,'(3(1x,E15.8))') NO

```

```

DNO_DX(1)=N_AXIS%X(1).SUB.'001'
DNO_DX(2)=N_AXIS%X(2).SUB.'001'
DNO_DX(3)=N_AXIS%X(3).SUB.'001'
DNO_DX2(1)=2*N_AXIS%X(1).SUB.'002'
DNO_DX2(2)=2*N_AXIS%X(2).SUB.'002'
DNO_DX2(3)=2*N_AXIS%X(3).SUB.'002'

write(6,*) " Spin-orbit Twiss in file LATTICE_SPIN.TXT "

CALL KANALNUMMER(MF,"LATTICE_SPIN.TXT")
P=>ALS%START

XS=XS0+NF_S%A_T ! Probe= closed orbit + A (A as in M=AoRoA**(-1) )

DO I=1,ALS%N

! send probe across each element EXACTLY
CALL TRACK_PROBE(ALS,XS,STATE,FIBRE1=I,FIBRE2=I+1)
!!!!!! USER DEFINED
A_S=XS ! turn the exact probe into a Berz-like obscenity
! Compute an obscenity here. Here it is the Ripken beta function
BETA_XX=(A_S%M%V(1).SUB.'1')**2+(A_S%M%V(1).SUB.'01')**2
BETA_XY=(A_S%M%V(1).SUB.'001')**2+(A_S%M%V(1).SUB.'0001')**2
!!!!!! END OF USER DEFINED
!!!!!! EXTRA USER DEFINED
call factor(A_S,A_f,A_spin,A_l,A_nl) ! A_S= A_f o A_spin o A_l o A_nl
n_axis=2; ! creates e_y=(0,1,0) FPP arbitrary choice for the normal spin n-axis
n_axis=A_spin*n_axis ! The spin part of A_total acts on e_y. That gives you the n-field
!!!!!! END OF EXTRA USER DEFINED

P=>P%NEXT
DNO_DX(1)=N_AXIS%X(1).SUB.'001'
DNO_DX(2)=N_AXIS%X(2).SUB.'001'
DNO_DX(3)=N_AXIS%X(3).SUB.'001'
DNO_DX2(1)=2*N_AXIS%X(1).SUB.'002'
DNO_DX2(2)=2*N_AXIS%X(2).SUB.'002'
DNO_DX2(3)=2*N_AXIS%X(3).SUB.'002'
WRITE(MF,'(a8,1x,8(1x,E15.8))') P%MAG%NAME(1:8),BETA_XX,BETA_XY,DNO_DX,DNO_DX2

ENDDO
CLOSE(MF)

WRITE(6,*) " !!! STROBOSCOPIC AVERAGE A LA HOFFSTADER !!!! "
!!! STROBOSCOPIC AVERAGE AS HOFFSTADER !!!!

n=10000

```

```

dx=1.d-4
nh=n*2 ! To avoid printing intermediate results
XS0=x0
XS0%X(3)=XS0%X(3)+dx
XS0%S(1)=1
XS0%S(2)=2
XS0%S(3)=3
XST=0

XS0m=x0
XS0m%X(3)=XS0m%X(3)-dx
XS0m%S(1)=1
XS0m%S(2)=2
XS0m%S(3)=3
XSTM=0

9 format(' (a20,3(1x,E15.8))')
2001 call stroboscopic_average(als,xs0,xst,1,STATE,n,nh,N_AXIS_avep)
2002 call stroboscopic_average(als,xs0m,xstm,1,STATE,n,nh,N_AXIS_avep)
write(6,*)" $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ "
WRITE(6,9) " Numerical dn/dy =", (N_AXIS_avep%x(:)-N_AXIS_avep%x(:))/dx/2.d0
WRITE(6,9) " Analytical dn/dy =", DNO_DX
WRITE(6,9) " Numerical d2n/dy2 =", (N_AXIS_avep%x(:)+N_AXIS_avep%x(:)-2*n0(:))/dx**2
WRITE(6,9) "Analytical d2n/dy2 =", DNO_DX2
write(6,*) " more -> 1 for yes"
read(5,*)i
if(i==1) goto 2001

end subroutine Track_a_ray_with_spin

```

Chapter 11. Review of normal form: Orbital and Spin

11.1 Very short review of the Orbital Normal Form

We will now describe the normal form in FPP for the orbital part. Since the spin is a spectator in PTC, it makes sense to have analysis tools which first deal with the orbital part. Once the orbital part is normalized, then the orbital dependent spin map is normalized into a rotation around the vertical axis.

11.1.1 The Orbital Normal Form Itself

The content of this section are documented at

http://mad.web.cern.ch/mad/PTC_proper/

It suffices to say that FPP allows for the normalization of a type damap.

```
TYPE DAMAP
```

```
TYPE (TAYLOR) V(NDIM2) ! NDIM2=6 BUT ALLOCATED TO ND2=2,4,6
```

```
END TYPE DAMAP
```

The normal form is central to all the self-consistent algorithms of PTC. So what is a normal form? A normal form is a reduction of the one-turn map into a form so simple that anyone can discover the basic property of the map. For example, in the case of a complete orbital normalization, we have:

$$m = a \circ r \circ a^{-1} \quad (11.1)$$

The map r has the following Lie representation:

$$r = \exp \left(: h(J_1, J_2, J_3; \vec{k}) : \right) I \quad (11.2)$$

or

$$= \exp \left(: h(J_1, J_2, x_5; \vec{k}) : \right) I \quad \text{where } I = \text{identity map of phase space} \quad (11.3)$$

In Eq. (11.2), the map is fully normalized into 3 amplitude dependent rotations. In Eq. (11.3), the transverse part of the map is normalized into 2 rotations. The longitudinal part is normalized into a drift-like map; here x_5 is an energy like variable (either energy or momentum). It is canonically conjugate to either time or path length. This last normal form, a Jordan normal form, applies to the case of a coasting beam. In PTC one can choose either:

$$z_5 = \delta = \frac{p - p_0}{p_0} \text{ with } z_6 = L \quad \text{or} \quad z_5 = p_t = \frac{E - E_0}{p_0 c} \text{ with } z_6 = cT \text{ (TIME state of PTC)} \quad (11.4)$$

If we accept the potential or formal existence of this normal form, then many things become trivial. If one instead is obstinate, then even the simplest of properties become mysterious even to the smartest of accelerator physicists. For example, it has been stated by some that the amplitude dependent path length is the chromaticity produced by the quadrupoles only. In the normal form of Eq. (11.2), this can be immediately computed as the derivative of h with respect to x_5 . Also the tunes are the derivative with respect to J_1 and J_2 . It is clear that the term in h which of the form $x_5(a_1 J_1 + a_2 J_2)$ will produce the chromaticities and the amplitude dependent path length.

So what went wrong? The original treatment had no concept of normal form what so ever and thus this trivial mistake crept in due to an incorrect averaging process.

There exists other normal forms. For example we can leave one resonance in the map: this could describe appropriately an isolated resonance disrupting phase space. We can also normalize the map in the presence of radiation: this is useful in the theory of synchrotron radiation.

11.1.2 The Normal Form Itself

The map a which brings us into normal form is far from being useless: not all maps are made of circles! It tells us the actual shape of the invariant quantities in real space. It allows us to transform any function into normal coordinates so that averages can be performed. Therefore all the lattice functions are computed from a and/or its inverse.

The map a can be factored as follows:

$$a = a_1 \circ a_\ell \circ a_{n\ell} \quad (11.5)$$

The map a_1 brings the map m to its parameter dependent fixed point. The parameters are \vec{k} and additionally z_5 in the case of a coasting beam. Therefore the map

$$m_1 = a_1^{-1} \circ m \circ a_1 \quad (11.6)$$

sends the origin of phase space into itself for all values of the parameters. The map a_ℓ normalizes the map m_1 linearly for all values of the parameters. Therefore a_ℓ provides us with the linear lattice functions as a function of the parameters. The end result,

$$r_\ell = a_\ell^{-1} \circ a_1^{-1} \circ m \circ a_1 \circ a_\ell \quad (11.7)$$

is a purely non-linear map around the parameter dependent fixed point. Finally $a_{n\ell}$ diagonalizes the map completely into the rotation r .

These maps in Eq. (11.5) can be obtained from a using the PTC subroutines

```
CALL FACTOR(A,A_1,A_L,A_NL) ! A=A_1 o A_L o A_NL
```

The idea is to extend this to spin maps.

11.2 The case of spin

We will now describe the normal form in FPP for the spin orbital map when the spin is a spectator. Again it is simplest to describe it in words first. In the commutative normal form of the spin orbital map, the orbital part moves on its usual circles with an amplitude dependent tune. Nothing is new there. The spin maps are all rotation along an arbitrary axis, we pick the vertical y -axis. The angle of the spin rotation depends only of the orbital motion, i.e., on the radii of the circles— on the action variables!

This is so trivial that a child can understand it. First of all, the y -axis is certainly an invariant. The spin angle being a function of the actions J_i is certainly a well defined tune since it is constant along the trajectory. It is the so-called spin tune. Finally since it is a commutative normal form, we can define easily a spin phase advance and a spin adiabatic invariant. But we are going too far here.

The normalized motion in the presence of spin is like the small and large arms of an old fashion watch. The small arm marks the time (orbital) and the little arm moves at a faster rate totally related to the hourly rate of the small arm. The amplitude dependence would be like saying that making the clock smaller/larger in size would result in a clock not well adjusted. Actually that would probably be the case if one scales the size of the clock without retuning it.

In the rest of the text we may use the locution “normal norm” to describe the Taylor map algorithm of FPP. The reader will perhaps excuse the abuse of language. Always one should remember that the tracking code is the final arbiter on the validity of an algorithmically computed normal form.

11.2.1 Spin Maps in FPP

In FPP spin map are of the type “DAMAPSPIN”. They are defined as follows:

```

TYPE DAMAPSPIN
  TYPE(DAMAP) M
  TYPE(REAL_8) S(3,3)
  real(dp) e_ij(6,6) ! Envelope radiation stuff
END TYPE DAMAPSPIN

```

11.2.2 Spin Maps Concatenation

So if T is a spin map, it is then given by a couple of objects:

$$T = (m, S) \quad (11.8)$$

where m is an orbital map and S is an orbital dependent spin matrix. If a beam line #1 is followed by beam line #2, then the spin map for the full beam line is given by:

$$\begin{aligned} T_2 \circ T_1 &= (m_2, S_2) \circ (m_1, S_1) \\ &= (m_2 \circ m_1, S_2 \circ m_1 S_1) \end{aligned} \quad (11.9)$$

The matrix $S_2 \circ m_1 S_1$ is simply the product of $S_2 S_1$ where $S_2(\bar{z})$ is evaluated at $\bar{z} = m_2(z)$ with $z = (x, p_x, y, p_y, z_5, z_6)$.

11.2.3 Spin Normal Form

Let us imagine that a spin-orbit transformation $U = (a, A)$ such that:

$$T = (m, S) = \underbrace{(a, A)}_U \circ \underbrace{(r, e^{\theta L_y})}_R \circ \underbrace{(a, A)^{-1}}_{U^{-1}} \quad (11.10)$$

It is clear that the map R can be formally computed as a Taylor map in both orbital and spin variables around the origin. It is now implemented in FPP. The type normal_spin form is given by:

```

type normal_spin
  type(normalform) N ! regular orbital normal form
  type(damapspin) a1 ! brings to fixed point
  type(damapspin) ar ! normalises around the fixed point
  type(damapspin) as ! pure spin map
  type(damapspin) a_t ! !! (a_t%m,a_t%s) = (a1%m, I ) o (I ,as%s) o (ar%m,I)
!!! extra spin info
  integer M(NDIM,NRESO),MS(NRESO),NRES ! orbital and spin resonances to be left in the map
  type(real_8) n0(3) ! n0 vector
  type(real_8) theta0 ! angle for the matrix around the orbit (analogous to linear tunes)
!!!Envelope radiation stuff
  real(dp) s_ij0(6,6) ! equilibrium beam sizes
  ! equilibrium emittances (partially well defined only for infinitesimal damping)
  real(dp) emittance(3),tune(3),damping(3)
  logical(lp) AUTO,STOCHASTIC
  real(dp) KICK(3) ! fake kicks for tracking stochastically
  real(dp) STOCH(6,6) ! Diagonalized of stochastic part of map for tracking stochastically
end type normal_spin

```

If NS is of type normal_spin and DS is a damapspin, then DS can be normalized as follows:

$$\text{NS=DS}$$

in complete analogy with the orbital normal form. In fact the analogy on the computer is complete.¹

So, as I said, if you believe in the normal form of Eq. (11.10), then even a stupid person can comprehend the concept of the invariant spin axis in the normalized space.

In the normalized space, we notice that the spin vector $\vec{e}_2 = (0, 1, 0)$ is an invariant of $R = (r, \exp(\theta L_y))$. Of course the choice of e_2 for the spin normal form is arbitrary. In the rarified air of mathematics, we are free to use anything for the normal form axis. We chose a “vertical” axis. Further more the angle of the spin rotation θ is itself invariant because it depends only on the actions and the parameters of the orbital map r . So the spin axis is defined.

We now intend to show the obvious: the vector $A\vec{e}_2$ is the spin field axis \vec{n} .

11.2.4 Factoring the Map T and the invariant spin axis \vec{n}

The map U can be conveniently factorized as follows:

$$U = (a_1, I) \circ (I, A) \circ (a_\ell, I) \circ (a_{n\ell}, I) \quad (11.11)$$

The total map T is thus given

$$\begin{aligned} T &= (m, S) \\ &= \underbrace{\left(m_f, e^{\theta \circ a_f^{-1} L_y} \right)}_{(a_f, I)} \\ &= (a_1, I) \circ (I, A) \circ \underbrace{(a_\ell, I) \circ (a_{n\ell}, I)}_{(a_f, I)} \circ (r, e^{\theta L_y}) \circ (a_{n\ell}^{-1}, I) \circ (a_\ell^{-1}, I) \circ (I, A^{-1}) \circ (a_1^{-1}, I) \quad (11.12) \\ &\quad \underbrace{\hspace{10em}}_{\text{Map around the parameter dependent fixed point} = (m_f, S_f)} \end{aligned}$$

First, let us ignore the transformation (a_1, I) ; this transformation brings us to a parameter dependent system, i.e., it does not tell us how the system is but how the system would be if we changed some parameters, for example quadrupole strengths. Therefore let us look at the map:

$$\begin{aligned} T_f &= (m_f, S_f) \\ &= \underbrace{\left(m_f, e^{\theta \circ a_f^{-1} L_y} \right)}_{(a_f, I)} \\ &= (I, A) \circ \underbrace{(a_\ell, I) \circ (a_{n\ell}, I)}_{(a_f, I)} \circ (r, e^{\theta L_y}) \circ (a_{n\ell}^{-1}, I) \circ (a_\ell^{-1}, I) \circ (I, A^{-1}) \quad (11.13) \\ &\quad \underbrace{\hspace{10em}}_{\text{Map around the parameter dependent fixed point} = (m_f, S_f)} \end{aligned}$$

This map is truly the map used by the tracking code. We will consider the following map

$$T_f \circ (I, A) \quad (11.14)$$

¹The fact that the spin is a spectator breaks the analogy slightly— we can avoid normalizing the orbital and spin parts simultaneously. We can also find the closed orbit

computed two different ways. First, we use the obvious ways:

$$\begin{aligned} T_f \circ (I, A) &= (m_f, S_f) \circ (I, A) \\ &= (m_f, S_f A) \end{aligned} \quad (11.15)$$

Now, let us use the normal form representation of Eq. (11.13) and rewrite Eq. (11.15):

$$\begin{aligned} T_f \circ (I, A) &= (I, A) \circ \left(m_f, e^{\theta \circ a_f^{-1} L_y} \right) \circ (I, A^{-1}) \circ (I, A) \\ &= (I, A) \circ \left(m_f, e^{\theta \circ a_f^{-1} L_y} \right) \\ &= \left(m_f, A \circ m_f e^{\theta \circ a_f^{-1} L_y} \right) \end{aligned} \quad (11.16)$$

Comparing Eqs. (11.15) and (11.16), we conclude that:

$$S_f A = A \circ m_f e^{\theta \circ a_f^{-1} L_y} \quad (11.17)$$

We now apply Eq. (11.17), on the vector \vec{e}_2 using the Einstein summation convention on repeated indices:

$$\begin{aligned} S_f A \vec{e}_2|_a &= A \circ m_f e^{\theta \circ a_f^{-1} L_y} \vec{e}_2|_a \\ S_{f;ab} A_{b2} &= A_{a2} \circ m_f \\ \text{because } e^{\theta \circ a_f^{-1} L_y} \vec{e}_2 &= \vec{e}_2 \text{ i.e., normal form!} \end{aligned} \quad (11.18)$$

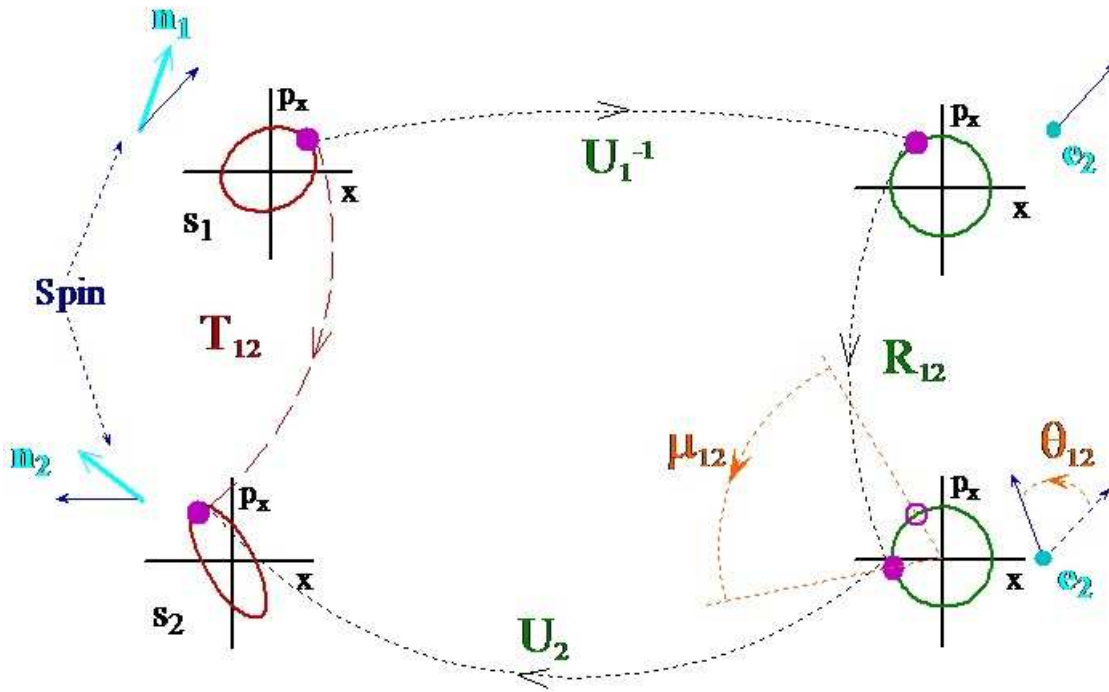
Therefore, we deduce from the Eq. (11.18) that the vector \vec{n} defined as $A \vec{e}_2$ transforms as follows:

$$S_f \vec{n} = \vec{n} \circ m_f \quad (11.19)$$

Equation (11.19) indicates that \vec{n} is truly an invariant vector function of phase space as well as being a solution of the spin motion. In other words it transforms under the action S_f as an orbital function would. Of course it was designed to be so by virtue of the assumed existence of the normal form.

11.3 Corollaries of normal form: phase advance and stroboscopic average

11.3.1 Phase Advance



Depiction of the orbital and spin phase advances induced by U

The concept of phase advance is best explained by the above picture. The one-turn map T_i at some arbitrary position s_i is normalized by the map U_i :

There are few points to make from which we deduce the phase advance:

1. The map at positions 1 and 2 are rotations in phase space and rotation in spin space around e_2 .
2. The sizes of the circles at 1 and 2 are the same. This is automatically true if canonical transformations are used. If not, it could be enforced.

Item 1 is simply the assumption of the normal form. The reader may wonder how is this assumption justified? Is there any basis for it? Actually, thanks to the process of perturbation theory, a package like FPP is equipped with a normal form algorithm on the Taylor map. Therefore the formal existence of the normal form, i.e., its approximate existence closed to the origin is not in doubt.

Item 2 is a little more tricky. If the transformation U is symplectic, then one can show that the circles at 1 and 2 are identical. If one allows a non-canonical transformation, it would not be the case. We disallow this for symplectic maps. In any event, the reader can see easily that one could scale the circles at position 2 to insure an identical size if non-canonical transformations were used.

Then it follows trivially that

$$R_{12} = U_2^{-1} \circ T_{12} \circ U_1 = (r_{12}, e^{\theta_{12} L_y}) \tag{11.20}$$

is a rotation in both phase space and spin space. By construction, it is of the same nature as the normal form. So the angle of the rotation in spin space is only a function of the phase space invariant radii.

The phase advance depends critically on the definition chosen for U_s . If U_s is a functional of the one-turn map, then the phase advance is obviously unique between matched position, i.e., between point with identical one-turn map.

11.3.2 “Stroboscopic” average

The “stroboscopic” average is a technique first proposed by Heinemann and Hoffstätter for the computation of the invariant spin axis \vec{n} . I put the word stroboscopic in quotation mark because the expression is highly unfortunate for a person dealing primarily with a map based theory. In a map based theory, a stroboscopic average would involved recording data not at every turn but once in a while. We assume that Heinemann and Hoffstätter were thinking about equations of motion when they coined the term; their paper confirms the prejudice. A true stroboscopic average is rarely discussed in accelerator theory; for example dispersion can be defined rigorously in the presence of 3 tunes as a stroboscopic skipping over p turns if the longitudinal tune ν_3 is nearly equal to q/p where q and p are integers. We will retain the term stroboscopic average by respect for the originators and also to avoid undue confusion.

The assumption of a normal form, whether we can compute it analytically or not, gives us a way to average quantities. Namely the idea is that an average over turns is actually an average over the invariant tori.

Let us start with the expression for the one-turn map assuming a normal form. It is given by Eq. (11.16). Dropping the subscript f , we get for the j -turn map:

$$\begin{aligned} T^j &= (I, A) \circ (m^j, e^{j\theta\alpha^{-1}L_y}) \circ (I, A^{-1}) \\ &= \left(m^j, A \circ m^j e^{j\tilde{\theta}L_y} A^{-1} \right) \quad \text{where } \tilde{\theta} = \theta(\vec{J}) \circ a^{-1} \equiv \theta(\vec{I}) \end{aligned} \quad (11.21)$$

If we apply this spin-orbital map to an arbitrary spin \vec{s}_0 at location \vec{x}_0 in orbital space, we get for \vec{s}_j at turn j :

$$\vec{s}_j = A_{\vec{x}_j} e^{j\tilde{\theta}L_y} A_{\vec{x}_0}^{-1} \vec{s}_0 \quad (11.22)$$

However in Eq. (11.22), we can see that the role of the initial position is special unlike the case of the normalized space. Indeed in actual space there is no guaranty that the spin axis at some position \vec{x}_j bares any resemblance to the spin axis at the initial point \vec{x}_0 . Therefore, we invert Eq. (11.22):

$$\vec{s}_0 = A_{\vec{x}_0} e^{-j\tilde{\theta}L_y} A_{\vec{x}_j}^{-1} \vec{s}_j \quad (11.23)$$

We now sum and take the limit:

$$\bar{\Pi} = \lim_{j \rightarrow \infty} \frac{1}{N} \sum_{j=1}^N A_{\vec{x}_0} e^{-j\tilde{\theta}L_y} A_{\vec{x}_j}^{-1} \quad (11.24)$$

Eq. (11.24) is a little tricky. The problem resides in the index j appearing in the matrix $A_{\vec{x}_j}^{-1}$. To perform the average we use the assumption of an orbital normal form and express the matrix $A_{\vec{x}_j}^{-1}$ in terms of the initial action angle variables $(\vec{\psi}, \vec{I})$:

$$\begin{aligned} A_{\vec{x}_j}^{-1} &= \sum_{\vec{m}} \Gamma_{\vec{m}}^0(\vec{I}) \exp\left(i\vec{m} \cdot \{\vec{\psi} + j\vec{\mu}\}\right) \\ &= \sum_{\vec{m}} \Gamma_{\vec{m}}(\vec{\psi}, \vec{I}) \exp\left(ij\vec{m} \cdot \vec{\mu}\right) \end{aligned} \quad (11.25)$$

The matrices $\Gamma_{\vec{m}}$ in Eq. (11.25) can be rewritten as three columns vectors:

$$\Gamma_{\vec{m}} = \left(\vec{\gamma}_1^{\vec{m}}, \vec{\gamma}_2^{\vec{m}}, \vec{\gamma}_3^{\vec{m}} \right) \quad (11.26)$$

The matrix Π becomes

$$\Pi = A_{\vec{x}_0} \sum_{\vec{m}} \lim_{j \rightarrow \infty} \frac{1}{N} \sum_{j=1}^N \left(e^{-j\tilde{\theta}L_y} \vec{\gamma}_1^{\vec{m}}, e^{-j\tilde{\theta}L_y} \vec{\gamma}_2^{\vec{m}}, e^{-j\tilde{\theta}L_y} \vec{\gamma}_3^{\vec{m}} \right) \exp(ij\vec{m} \cdot \vec{\mu}) \quad (11.27)$$

In Eq. (11.27), each vector $\vec{\gamma}_k^{\vec{m}}$ is made of 3 components which transform differently under the effect of $e^{-j\tilde{\theta}L_y}$. First the second component $\vec{\gamma}_{k;2}^{\vec{m}}$ is left invariant by the rotation. The first and third component can be expressed using spin phasors:

$$\vec{\sigma}_{\pm} = \begin{pmatrix} \pm i \\ 0 \\ 1 \end{pmatrix} \quad \text{where} \quad e^{-j\tilde{\theta}L_y} \vec{\sigma}_{\pm} = e^{\mp ij\tilde{\theta}} \vec{\sigma}_{\pm} \quad (11.28)$$

Therefore we have :

$$\vec{\gamma} = \frac{\gamma_3 - i\gamma_1}{2} \vec{\sigma}_+ + \frac{\gamma_3 + i\gamma_1}{2} \vec{\sigma}_- + \gamma_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad (11.29)$$

The vector in Eq. (11.29) represents any of the three column vectors of Eq. (11.27).

$$e^{-j\tilde{\theta}L_y} e^{ij\vec{m} \cdot \vec{\mu}} \vec{\gamma} = e^{-ij(\tilde{\theta} + \vec{m} \cdot \vec{\mu})} \frac{\gamma_3 - i\gamma_1}{2} \vec{\sigma}_+ + e^{-ij(-\tilde{\theta} + \vec{m} \cdot \vec{\mu})} \frac{\gamma_3 + i\gamma_1}{2} \vec{\sigma}_- + e^{-ij\vec{m} \cdot \vec{\mu}} \gamma_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad (11.30)$$

From Eq. (11.30), we see that the infinite series of Eq. (11.27) will converge zero for all terms such that $\vec{m} \neq 0$ provided we are not sitting on some spin-orbital resonance.

$$\begin{aligned} \Pi &= A_{\vec{x}_0} \begin{pmatrix} 0 & 0 & 0 \\ \vec{\gamma}_{1;2}^{\vec{m}=0} & \vec{\gamma}_{2;2}^{\vec{m}=0} & \vec{\gamma}_{3;2}^{\vec{m}=0} \\ 0 & 0 & 0 \end{pmatrix} \\ &= \left(\vec{\gamma}_{1;2}^{\vec{m}=0} \vec{n} \mid \vec{\gamma}_{2;2}^{\vec{m}=0} \vec{n} \mid \vec{\gamma}_{3;2}^{\vec{m}=0} \vec{n} \right) \end{aligned} \quad (11.31)$$

Remark: The convergence of the stroboscopic average towards a vector \vec{n} satisfying Eq. (11.19) requires that the first two terms in Eq. (11.30) average to zero. This will happen even if the map is on an orbital resonance. Thus the vector must exist even on an isolated orbital resonance. This is actually confirmed perturbatively by the perturbative approach to normal form. It is not clear however if a proper phase advance can be defined since it involves removing resonant terms from the rotation around the normal axis e_2 — a process not necessary for the simple existence of \vec{n} .

Bibliography

- [1] P.K. Skowronski, F. Schmidt, R. de Maria and E. Forest, “New Developments of MAD-X Using PTC”, paper presented at the ICAP06 Conference in Chamonix, France, <http://cern.ch/Frank.Schmidt/report/WEPPP12.PDF>.
- [2] E. Forest, Y. Nogiwa and F. Schmidt, “The FPP Documentation”, paper presented at the ICAP06 Conference in Chamonix, France, <http://cern.ch/Frank.Schmidt/report/WEPPP04.PDF>.
- [3] E. Forest, Y. Nogiwa and F. Schmidt, “The FPP and PTC libraries”, paper presented at the ICAP06 Conference in Chamonix, France, <http://cern.ch/Frank.Schmidt/report/MOM1MP02.PDF>.
- [4] Catia Milardi (INFN/LNF, Frascati (Roma)), Etienne Forest (KEK, Ibaraki), F. Schmidt (CERN, Geneva, Switzerland), “MAD-X/PTC Lattice Design for DAFNE at Frascati, CERN-AB-2006-062, paper presented at the 10th EPAC Conference in Edinburgh 2006, <http://cern.ch/Frank.Schmidt/report/MOPLS041.pdf>
- [5] E. Forest, S.C. Leemann (KEK, Tsukuba), F. Schmidt, “Fringe Effects in MAD PART I, Second Order Fringe in MAD-X for the Module PTC”, http://cern.ch/Frank.Schmidt/report/fringe-part_I.pdf.
- [6] E. Forest, S.C. Leemann (KEK, Tsukuba), F. Schmidt, “Fringe Effects in MAD PART II, Bend Curvature in MAD-X for the Module PTC”, http://cern.ch/Frank.Schmidt/report/fringe_II_madx.pdf.
- [7] F. Schmidt (MAD-X custodian), “MAD-X PTC Integration”, paper presented at the 2005 PAC Conference in Knoxville, USA, <http://cern.ch/Frank.Schmidt/report/MPPE012.pdf>.
- [8] E. Forest, E. McIntosh and F. Schmidt, “Introduction to the Polymorphic Tracking Code Fibre Bundles, Polymorphic Taylor Types and “Exact Tracking””, CERN-SL-2002-044 (AP), KEK-Report 2002-3,
Cover: http://cern.ch/Frank.Schmidt/report/small_cover.pdf,
Report: <http://cern.ch/Frank.Schmidt/report/small.pdf>.