The FPP and PTC libraries[*]

E. Forest, Y. Nogiwa, KEK, Tsukuba, Japan,
F. Schmidt, CERN, Geneva, Switzerland

Abstract

In this short article we summarize the FPP package and the tracking code PTC which is crucially based on FPP. PTC is remarkable for its use of beam structures which take into full account the three dimensional structure of a lattice and its potential topological complexities such as those found in colliders and recirculators.

FULLY POLYMORPHIC PACKAGE: FPP

FPP overloads in Fortran90 [1] the famous "DA-package" [2] of Berz. More precisely it overloads the package which Berz developped at the now defunct SSC-Central design group. It also overload a library based on Berz's package called Lielib[1] and developped by Forest.

FPP most salient feature is to create a Taylor-Real polymorphic type which changes shape at execution time. To do so using Berz's original package, we use a piece of code based on a Fortran77 prototype of J. Bengtsson. In fact, polymorphism at execution time is an idea of Bengtsson. This is to be contrasted with interpreted polymorphism which is the mechanism which underpins the code COSY-INFINITY [4] of Berz. Both are valid ideas and can even be combined simultaneously. This is not done in FPP.

The user of FPP writes a standard (symplectic) integrator [5, 6, 7] which simply pushes particles through a lattice. If the real variables "real(8)" are replaced everywhere by a new type called REAL_8 then Taylor series can be produced thanks to the underlying package of Berz.

It should be pointed out that our decision to extract Taylor series directly from an integrator rather than using techniques such as those found in COSY-INIFINITY is dictated by the physics of large (circular) machines. The vast majority of people [8, 9, 10, 11, 12, 13, 14, 15, 16] in that field prefer to use integrators, even with rather poor models, than to compute some high order Taylor maps. Thus in our field, the production of Taylor maps is for analysis mainly (computation of lattice functions for example), rather than for tracking particles. There are a lot of applications where the techniques pioneered by Berz recently (interval arithmetic, validated computation, Taylor models, etc...) are more desirable than bone headed integration. However we will stick here to what we often refer to as the Talman view of accelerator simulation: take a symplectic model and stick to it fanatically. Nevertheless, one can only lament that the present work was not done in concert with Berz's most recent tools. Perhaps this situation can be corrected in the near future: one can even imagine COSY-INFINITY being called from Fortran90 combining compiled and interpreted polymorphism.

[*] Work supported by KEK and Kasokukishoureikai.

[1]The initial theory was described in reference [3]

POLYMORPHIC TRACKING CODE: PTC

PTC [17] may appear novel in two ways. First it uses FPP for all perturbative calculations. Secondly it has novel structures to fully exploit the magnet/object in a dynamical setting.

The usage of FPP in PTC is by far the more ancient and well known aspect of PTC. One of the author has been pushing for the inclusion of "DA" and the related Lielib in "kick codes" since the days of the defunct SSC-CDG. Taylor polymorphism is also an old story by now with Berz and Bengtsson as main proponents.

The development of structures more adapted to the mathematics of "s"-tracking is far less understood and more recent except for the fact that Forest and Bengtsson were already discussing these things in the early days of the C++ disaster known as CLASSIC. In part we did not want to get involved with this project precisely because their class design was not motivated by the underlying mathematical structure of "s" or magnet based tracking.

We will now summarize very briefly what can be done with FPP in PTC or for that matter in any other integrator code equipped with FPP.

FPP in PTC

PTC is above all an integrator. It pushes a particle from one magnet to the next by integrating one step at a time. Thus, in the light of the previous discussion, if equipped with FPP, PTC can produce a Taylor map. For example, the following command tracks the ray $X = (0, 0, 0, 0, 0, 0)$ around a ring from position 1 back to position 1:

```
REAL(8)      X(6)
TYPE(REAL_8) Y(6)
    .
    .
    .
X=0.D0
Y=X
CALL TRACK(MY_RING,Y,1,DEFAULT)
```

However, the following command will produce a Taylor series map using Berz's package:

```
TYPE(REAL_8) X(6)
TYPE(DAMAP) IDENTITY
    .
    .
    .
  X=0.D0
  IDENTITY=1   ! MAKES A MAP IDENTITY
  Y=IDENTITY+X

  CALL TRACK(MY_RING,Y,1,DEFAULT)
```

More remarkably is the way the full nonlinear Courant-Snyder [18, 19, 7] theory looks in this overloaded environment. In the map based theory, one normalizes first the one-turn map around the closed orbit and then propagates the canonical transformation obtained through normalization:

```
TYPE(REAL_8) CLOSED_ORBIT(6)
TYPE(DAMAP) IDENTITY
TYPE(NORMALFORM) NORMAL
     .
     .
     .
IDENTITY=1   ! MAKES A MAP IDENTITY
Y=IDENTITY+CLOSED_ORBIT

CALL TRACK(MY_RING,Y,1,DEFAULT)
NORMAL=Y   ! THE MAP Y IS NORMALIZED (1)
Y=CLOSED_ORBIT+NORMAL%A_T  ! A_T NORMALIZES Y AT I=1

DO I=1,MY_RING%N

CALL TRACK(MY_RING,Y,I,I+1,DEFAULT)  ! (2)

 !!! HERE Y CONTAINS THE CANONICAL TRANSFORMATION
 !!! AT EVERY POSITION I AROUND THE RING
 !!! LATTICE FUNCTIONS OF ALL SORTS CAN NOW
 !!! BE EXTRACTED FROM Y USING OPERATORS OF FPP
 !!! FOR EXAMPLE BETA_X_1, THE DEPENDENCE OF THE
 !!! BEAM SIZE ON THE FIRST INVARIANT IN THE RIPKEN
 !!! FORMALISM

BETA_X_1= (Y(1).SUB.'10')**2+(Y(1).SUB.'01')**2  ! (3)
 !!! BETA_X_1 IS COMPUTED USING OPERATORS DEFINED IN
 !!! THE FPP SYNTAX
ENDDO
```

In the above loop, the syntax mimicks the theory although certain abuse of language are introduced in PTC proper. For example, `NORMAL=Y` strickly speaking makes no sense since one can only normalize "DAMAPs" and `Y` is a collection of six polymorphs. For completeness, we list the mathematical expressions which are represented by the above FORTRAN code:

- The map is normalized or line (1)

$$r = a_1^{-1} \circ m_1 \circ a_1 \tag{1}$$

- The canonical transformation is tracked on line (2). Notice that the polymorphs are first initialized as the closed orbit plus the initial canonical transformation A_T expressed in variables expressing deviation from the closed orbit.

$$b_i = m_{1\ i} \circ a_1 \tag{2}$$

The map b_i diagonalizes the one-turn map at the discrete location $s = i$. Therefore all the usual and well as less usual lattice functions can be extracted from it. For example, linear theory shows that the average of the function x^2 denoted by $< x^2 >$ is given by

$$\begin{aligned} \left\langle x^2 \right\rangle &= \sum_k \beta_{xk} \left\langle \varepsilon_k \right\rangle \\ \beta_{xk} &= A^2_{1\ 2k-1} + A^2_{1\ 2k} \end{aligned} \tag{3}$$

The computation of β_{x1} is done in line (3). This uses one of the numerous operators provided by FPP. For example, the operator `.SUB.` can be used to extract the coefficient of a monomial from a Taylor series (or a polymorph). If for example

$$Y = 5.0x_1^2 + 2.0x_1x_2 \tag{4}$$

then

$$\mathrm{Y.SUB.}'11' = 2.0 \tag{5}$$

and

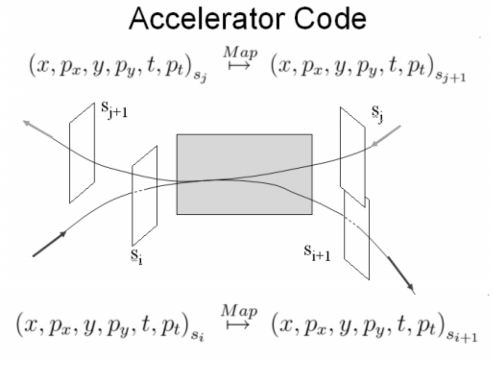
$$\mathrm{Y.SUB.}'20' = 5.0 \tag{6}$$

They are tons of other operators related to the plain manipulation of Taylor series as well as types related to DAMAPS and their various Lie represenetions. These representations include the normal form, the Dragt-Finn and inverse Dragt-Finn as well as the one-Lie exponent representation. Routines overloading the equal sign ("=") permit the conversion from one type of map to the other.

Most of these things are documented on the Web site: http://mad.web.cern.ch/mad/PTC_proper/.

PTC proper

Most if not all tracking codes germane to accelerator physics view the world as a collection of magnet propagators. That is to each magnet labelled by an index "i" they associate a propagator m_i. This propagator moves the ray from a surface at the beginning of the magnet to one at the end of the magnet. In Fig. 1 we depict two possible



Accelerator Code
$(x,p_x,y,p_y,t,p_t)_{s_j} \overset{Map}{\mapsto} (x,p_x,y,p_y,t,p_t)_{s_{j+1}}$
s_{j+1}
s_j
s_i
s_{i+1}
$(x,p_x,y,p_y,t,p_t)_{s_i} \overset{Map}{\mapsto} (x,p_x,y,p_y,t,p_t)_{s_{i+1}}$

Figure 1: Magnet with forward and reversed propagators.

propagators for the i^{th} magnet: one forward and one backward through the magnet. Standard codes will then define a beam line as an ordered sequence of magnets:

$$Line = (M_1, M_2, \cdots, M_N) \tag{7}$$

Normally we think of the line of Eq. 7 as a series of forward propagators such as those depicted on Fig. 1. For example, if the propagator through magnet M_1 is m_{01}, then it is assumed that the map going through the beam line of Eq. 7 is:

$$m_t = m_{N-1\ N} \circ \cdots \circ m_{12} \ \circ m_{01} \ , \tag{8}$$

The total map m_t in concert with the frames attached to the magnet (see Fig. 1), produce an implicit positioning of the beam line in the machine. In reality, magnets should have no implied relative position; the frames of reference are mathematical artefacts of the "s" representation which neither the particle nor the original Lorentz equation cares about. Thus the question arises: if we want to exploit the map attached to a magnet and free it from the tyranny imposed by the definition of Eq. 7, how should we define a beam line?

The answer is simply that the beam line is a collection of "containers" called fibres in PTC by anology with the mathematics which supports this entire "s"-tracking apparatus. The fibres represent the variable "s" in discretized form. The fibre, as a computer science object, has a pointer to three distinct objects: to the standard magnets M_i, to a series of patches (element of the translation-rotation group) and to some frame of references (a chart) locating the ideal position of this magnet in space. The need for fibres is

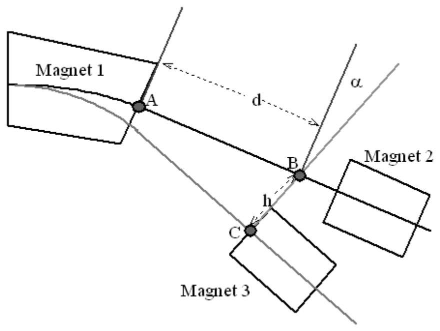


Figure 2: Recirculator problem.

clearly displayed in Fig. 2. Magnet 1 is the beam separator of a recirculator. At high energy, the particle goes from magnet 1 to magnet 2. Then it is decelerated in a linac preceding magnet 1 and thus it is deflected into magnet 3. We must notice a couple important points:

- The magnet 1 is traversed twice. If we described the recirculator as a sequence of distinct magnets, the second time around it must reappear as a clone of itself on the computer. This is not correct. It is the same magnet. A link list if magnets would definitely fail miserably since the same magnet would follow magnet 1. This is the present MAD-X setup. An array of pointers to magnet would potentially solved this problem but the not the following issue.

- If we assume that the frame of references between magnet 1 and magnet 2 are smoothly lined up, then it is certainly not that case between magnet 1 and magnet 3. In fact in standard codes, we may also have problems with the reference energy between 1 and 3.

We see that no matter how we slice it, this problem cannot be solved by retaining the magnet and its associated propagator(s) as the element of a beam line.

The solution of PTC is to represent the beam line, as we said, as a linked list of containers called fibres. It is just the discretized variable "s". The first time around, magnet 1 is fibre $s = 1$ and magnet 2 is in fibre $s = 2$. These magnets (ignoring the drift for this discussion) are lined up so that no special patching is required: they are in the standard MAD position. The next time around, let us say that magnet 1 is in the fibre $s = 100$. It is a different fibre and it contains a patch to the standard frame of reference of magnet 3. On Fig. 2, this patch is made of a rotation of angle α around the vertical axis followed by a translation of h in the horizontal transverse direction. Since fibre $s = 1$ and $s = 100$ point to the same magnet (# 1), all changes that affect # 1 are correctly taken into account without hacking the code. Obviously fibre $s = 100$ points to fibre $s = 101$ which contains magnet 3. Thus the tracking follows the correct sequence. Finally since fibres $s = 1$ and $s = 100$ are different, they can contain different patches. The passage from magnet 1 to magnet 3 is definitely not compatible to a standard MAD survey.

Of course by default all the patches of PTC are initialized as the identity map when a beam line is created. This insures that when a standard sequence is passed from MAD-X, the resulting beam line is what one would expect from a traditional code.

BEYOND AND INSIDE THE FIBRE

It makes sense in an accelerator dominated by single particle dynamics to produce computer objects which are tantamount to the usual magnet. This object will know how to propagate a ray and how to draw itself. These functions will respond to a misalignment of the magnet within the fibre as well as a total repositioning of the fibre. Furthermore, consistent with this style of programming, one does not allow oneself to look or change data inside the object. The magnet's physical and mathematical integrity is preserved.

Unfortunately there are lots of reasons why we would like to have access to the inside of a magnet. First of all internal tracking data such as rays or lattice functions are often wanted by designers. This is reflected in the annoying practice of slicing a magnet in two equal parts at a symmetry point of the lattice.

Secondly and most importantly we have to include collective effects. Beam-beam, space charge and wake effects do not choose to act at the arbitrary entrance and exit surfaces of a fibre! So ideally we need time based tracking which runs counter to a magnet oriented description. However if we have an integrator we can theoretically interrupt tracking at any integration step; this provides us with a first order accurate location of the constant time surface.

For the purpose of looking inside the fibre, we split the fibre in five distinct regions which we call integration nodes: an entrance patch, an entrance fringe field, the body of the magnet, an exit fringe field and an exit patch. Thus when a particle enters a fibre, it undergoes the following transformations:

1. The entrance patch node consists of an actual patch which connects the geometry of the preceding fibre with the present one. Then follow the misalignment operators which are mathematically of the same nature as the patches. Finally a vertical tilt which is used in vertical magnets.

2. The entrance fringe node contains approximate fringe effects, various obscenities such as wedges and also occasionally the conversion between canonical and non-canonical variables.

3. “NST” integration steps going through the magnet. Often these steps are identical; sometimes they are not.

4. The exit fringe node which is the “reverse” of the entrance fringe node.

5. The exit patch node which is needed to complete the placement of a fibre and the misalignment of magnet within the fibre.

Thus we conclude that a fibre in PTC is made of 4+NST integration nodes.

The Node Layout

Just as in the standard PTC the lattice is a linked list of fibres, the node layout is a linked list of nodes. The data in the node reside on the fibres themselves. In other words the nodes do not contain new data; they have no existence of their own. This means that changes affecting the magnets (field changes, etc...) and all changes affecting the fibres (displacements of the fiducial position) are carried over to the nodes faithfully.

Survey of the Node Layout: 3d plots

The survey command of PTC normally locates the fibre and the magnet within the fibre. If a user has placed fibres in a bizarre (non-MAD) location, the survey command will leave the fibres unchanged if the patches are correctly computed. Thus it is a command useful to check the patching algorithm.

However it is clear that the position of each individual nodes, particularly the NST nodes in the body of the magnet, depend on the general architecture of the integrator used for that magnet. The survey command for the node layout computes the frame of reference corresponding to

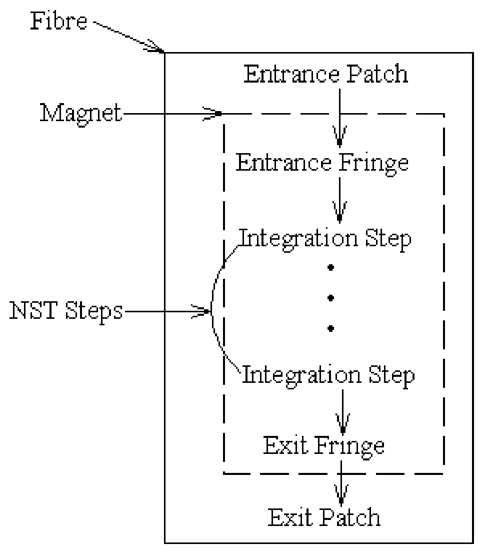


Figure 3: The exposed fibre.

each node. Obviously it is a command which must be tailored to each new magnet model introduced by the programmer. This command truly permits us to locate a trajectory in 3d. Discontinuous changes of coordinates such as tilts are thus taken into account.

Collective effects: extension of the coordinates

The nodes allow us to look at any point in the layout. One of the most important applications is to permit collective forces. Moreover we can extend the coordinates of a particle to permit first order time based tracking within the higher order “s” framework. To do so, we add to the particle two coordinates:

$$\vec{z} = (x, p_x, y, p_y, t, p_t; \delta s, p_n) \tag{9}$$

The coordinate “δs” is the distance from the beginning of the integration node measured in the coordinate “s” use by the integrator. The variable “p_n” is a pointer to the integration node in which the particle finds itself at time “t.” So during normal PTC s-based tracking δs is always zero. In a time tracking mode a drift is assumed between nodes to estimate the time of a particle; the result δs is computed. Since inverse drifts are exactly known, this reduces to the normal s-based tracking when collective effects are absent.

One immediate application is the tracking of several macro-particles in a recirculator with important wake field effects between the macro-particles: time ordering of the bunches is crucial and painlessly done in our framework.

CONCLUSION

There is an obvious philosophical choice which underpins all magnet based tracking codes. We emphasize the maps attached to a magnet. One of the author realized in the days of SSC-CDG that the discontinuities of such a theory makes our tracking codes mathematically incompatible [18, 20] with a standard Courant-Snyder theory. We cannot on an *arbitrary* trajectory computed by an s-based tracking code define a Frenet system of tangent, normal and binormal vectors. This is *mathematically impossible* because of the discontinuous boundaries which are present in our models. Thus a theory based on finite maps is the only fully self-consistent theory; in fact in the presence of patches and misalignments it is the only logical choice. In the limit of maps representing tiny nodes ($ds \approx 0$) and smooth fields, then we regain the full Hamiltonian structure.

Others have noticed independently [19, 21, 22] this discontinuous aspect of the theory and constructed mathematical tools to handle it. However the first author of this paper always believed that only a serious computer implementation would allow these ideas to propagate. Thus emerged our early collaboration with Berz and the resulting FPP package. This allows the computation the self-consistent lattice functions (linear and nonlinear) in a painless way.

Although time based tracking is more fundamental, it has been rejected by accelerator physicists at the onset. This makes perfect sense in a system dominated, at low current, by independent magnets for design and simulation reasons. But, in this paper, we have shown that if a code possesses the right structures for its beam line "classes" then one regains time tracking. We recognize however that this tracking is first order in $\epsilon\, dt$ where ϵ represent the strength of the collective effect.

The combination of Taylor tools (Berz package, Lielib and FPP) and modern computed structures is a powerful way to represent machines of all sorts. The linking of this FPP/PTC conglomerate with MAD-X adds a extra-layer of user friendliness. The only thing missing is interpreted polymorphism which would allow a user to operate on the machines and compute perturbative things from an input file. This could be achieved only by writing an interpreter for FPP or by relinking the full FPP with the more powerful "DA" tools of an extended COSY-INFINITY.

REFERENCES

[1] E. Forest and F. Schmidt. The "full polymorphic package" (FPP). *ACM SIGPLAN Fortran Forum*, 20:12–17, 2001. (N.Y., USA).

[2] M. Berz. *Part. Accel.*, 24:109, 1989.

[3] E. Forest, M. Berz, and J. Irwin. *Part. Accel.*, 24:91, 1989.

[4] M. Berz. COSY INFINITY Version 8. Technical report, Michigan State University, January 2000.

[5] E. Forest. Geometric integration for particle accelerators. *J. Phys. A: Math. Gen.*, 39:5321–5377, 2006.

[6] E. Forest, M. F. Reusch, D. Bruhwiler, and A. Amiry. The Correct Local Description for Tracking In Rings. *Part. Accel.*, 45:66, 1994.

[7] E. Forest. *Beam Dynamics: A New Attitude and Framework*. Harwood Academic Publishers, Amsterdam, The Netherlands, 1997.

[8] F. Méot. The ray-tracing code Zgoubi. *NIM*, A 427:353–356, 1999. This code is a plain integrator for rings without special enforcement of symplecticity.

[9] Y. K. Yu, E. Forest, and D. S. Robin. Explicit Symplectic Integrator for s-dependent static magnetic field. *Phys. Rev. E*, 68, 2003.

[10] H. Weidemann. Technical Report PEP Note 220, SLAC, 1976.

[11] H. Weidemann. Technical Report PEP Technical Memo 230, SLAC, 1981.

[12] A. Wrulich. RACETRACK: A Computer code for the simulation of particle motion. Technical Report DESY 84-026, 1984.

[13] F. Schmidt. SIXTRACK, version 1.2, single particle tracking code treating transverse motion with synchrotron oscillations in a symplectic manner. Technical Report CERN SL/94–56 (AP), CERN, 1994.

[14] G. Ripken. Technical Report 85–084, DESY, 1984. This is the oldest reference on SIXTRACK.

[15] Code by K. Oide, Japanese documentation can be found on KEK internet site.

[16] L. Schachinger and R. Talman. *Part. Accel.*, 22:35, 1987. E. Forest checked TEAPOT against the PSR lattice paper of Dragt for the appendix of this paper.

[17] E. Forest, F. Schmidt, and E. McIntosh. Introduction to the Polymorphic Tracking Code. Technical Report CERN-SL-2002-044, KEK-Report 2002-3, 2002.

[18] E. Forest. A Hamiltonian-free description of single particle dynamics for hopelessly complex periodic systems. *J. Math. Phys.*, 31:1133, 1990. originally, SSC-111, 1987.

[19] A. Bazzanni, P. Mazzanti, G. Servizi, and G. Turchetti. *Nuovo Cimento*, B(102):51, 1988.

[20] E. Forest. Canonical integrators as tracking codes (or How to integrate perturbation theory with tracking). Technical Report SSC-138, SSC-CDG, 1987. Also part of lectures delivered at Fermi National Laboratory.

[21] A. Bazzani, M. Giovannozzi, and E. Todesco. A Program to Compute Birkhoff Normal Forms of Symplectic Maps in R^4 . *Comp. Phys. Comm.*, 86:199, 1995.

[22] A. Bazzani, E. Todesco, G. Turchetti, and G. Servizi. Technical Report CERN 94-02, CERN, March 1994.