

# Manual for FPP: The Fully Polymorphic Package

Étienne Forest  
Tsukuba, Japon

August 21, 2018

# Contents

<b>1</b>	<b>Distinguishing PTC from FPP</b>	<b>3</b>
1.1	Tracking and analysis . . . . .	3
1.2	Taylor series and Polymorphs . . . . .	4
1.3	map . . . . .	6

# 1 Distinguishing PTC from FPP

## 1.1 Tracking and analysis

In E. Forest's book [1], it is alleged that accelerator theory and simulation can be cleanly separated into two parts:

1. A tracking code which simply tracks rays, for example  $(q, p)$ , if the code has only one degree of freedom (1-d-f).
2. An analysis part which computes lattice functions, tunes, tune shifts, etc... via Taylor series map produced by the tracking code.

If a code simply tracks rays, i.e., two real variables  $(q, p)$ , it does not seem feasible that Taylor series maps will be magically produced. Therefore it may not seem possible that items 1 and 2 can co-exist within the same programming environment.

However here enters the magic of Truncated Power Series Algebra introduced in our field by Martin Berz[2]. With little effort, a code which pushes real numbers can be converted into a code which tracks Taylor series. For example, a code denoted by  $T$  which tracks the closed orbit of a ring, say  $(q_0, p_0)$ , can be coerced to track the linear matrix approximation around this orbit:

$$\begin{aligned} \text{if } T \begin{pmatrix} q_0 \\ p_0 \end{pmatrix} &= \begin{pmatrix} q_0 \\ p_0 \end{pmatrix} \\ T \begin{pmatrix} q_0 + dz_1 \\ p_0 + dz_2 \end{pmatrix} &= \begin{pmatrix} q_0 \\ p_0 \end{pmatrix} + \underbrace{\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}}_M \begin{pmatrix} dz_1 \\ dz_2 \end{pmatrix} \end{aligned} \quad (1)$$

It is well-known that knowing the matrix  $M$  allows a user to compute the lattice functions and the tune:

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix} = \begin{pmatrix} \cos(\mu) + \alpha \sin(\mu) & \beta \sin(\mu) \\ -\gamma \sin(\mu) & \cos(\mu) - \alpha \sin(\mu) \end{pmatrix} \quad (2)$$

The code  $T$ , which we can view as a function, must be instructed to take derivatives with respect to the phase space. If one evaluates

$$T \begin{pmatrix} q_0 + dz_1 \\ p_0 \end{pmatrix}$$

then the output will be a Taylor series but the output **cannot** be considered an approximate rendition of the map/code  $T$ . It is simply a Taylor expansion of  $T$  with respect to the first phase space coordinate.

Additionally one can expand  $T$  with respect to system parameters: multipole strengths, lengths, etc. . . The resulting Taylor expansion will not be a *bona fide* approximate map of  $T$  unless the vector of infinitesimals  $(dz_1, dz_2)$  is added to the initial  $(q_0, p_0)$  by the magic of TPSA .

In conclusion, the code PTC tracks rays and also produces Taylor series. The library FPP can analyse the output if and only if the approximate map is a *bona fide* approximate map of  $T$ . As we will see, an enormous amount of things can be computed: lattice functions, phase advance for orbital and spin dynamics as described by Forest in [3].

It is therefore important to distinguish a Taylor expansion of the ray and the *bone fide* approximate maps. We will see this in Sec. (1.3). But first let us introduce a Taylor type and a polymorphic type

## 1.2 Taylor series and Polymorphs

As is hinted in the previous section, the programmer does not *a priori* if the ray will be a Taylor series or simply real numbers. To deal with this issue, we created a polymorphic type called real\_8. Here is its definition found in the file h\_definition.f90 of FPP(PTC). We also include the definition of Taylor.

```

TYPE TAYLOR
  INTEGER I ! integer I is a pointer in old da-package of Berz
END TYPE TAYLOR

TYPE REAL_8
  TYPE (TAYLOR) T ! USED IF TAYLOR
  REAL(DP) R ! USED IF REAL
  INTEGER KIND ! 0,1,2,3 (1=REAL,2=TAYLOR,3=TAYLOR KNOB, 0=SPECIAL)
  INTEGER I ! USED FOR KNOBS AND SPECIAL KIND=0
  REAL(DP) S ! SCALING FOR KNOBS AND SPECIAL KIND=0
  LOGICAL(LP) :: ALLOC 1 IF TAYLOR IS ALLOCATED IN DA-PACKAGE
END TYPE REAL_8

```

One notes that Taylor is simply an integer. This integer points to the  $I^{th}$  Taylor series in the LBNL version of Berz' package. In that package all the Taylor series use the same amount of memory. For example, a third order

Taylor series in two variables would contain  $(3 + 2)!/3!/2!$  terms, namely 10 terms:

$$t = t_{00} + t_{10}z_1 + t_{01}z_2 + t_{20}z_1^2 + t_{11}z_1z_2 + t_{02}z_2^2 + t_{30}z_1^3 + t_{21}z_1^2z_2 + t_{12}z_1z_2^2 + t_{03}z_2^3 \quad (3)$$

In any code the size of PTC it would be totally infeasible memory-wise to make every variable of the code of type Taylor. Instead we invented the type `real_8`. The type `real_8` contains a type `taylor` which is not “activated” if the polymorph is real but suddenly becomes Taylor if the said polymorph must be a Taylor series. This is illustrated in the program `z-tpsa0.f90`.

```

program example0
use madx_ptc_module
use pointer_lattice
implicit none
type(real_8) x_8
real(dp) x
longprint=.false.
call ptc_ini_no_append
call append_empty_layout(m_u)
call set_pointers; use_quaternion=.true.;

call init(only_2d0,3,0) ! TPSA set no=3 and nv=2

x=0.1d0

call alloc(x_8)

x_8=x ! insert a real into a polymorph

write(6,*) " Must be real "
call print(x_8)

x_8=x_8+dz_8(1)

write(6,*) " Must be Taylor : 0.1+dx "
call print(x_8)

x_8=x_8**4

write(6,*) " Must (0.1+dx)**4 to third order "

call print(x_8)
end program example0

```

The result of this program is:

```

Must be real
0.1000000000000000
Must be Taylor : 0.1+dx

Properties, NO =    3, NV =    2, INA =   21
*****

  0  0.1000000000000000      0  0
  1  1.0000000000000000      1  0

Must (0.1+dx)**4 to third order

Properties, NO =    3, NV =    2, INA =   21
*****

  0  0.1000000000000000E-03  0  0
  1  0.4000000000000001E-02  1  0
  2  0.6000000000000001E-01  2  0
  3  0.4000000000000000      3  0

```

The last output is simply:

$$x_8 = 10^{-3} + 4 \times 10^{-3} z_1 + 6 \times 10^{-2} z_1^2 + 4 \times 10^{-1} z_1^3 + \dots \quad (4)$$

The code behaved as expected: once the infinitesimal  $dz_1$  (stored in `dz_8(1)`) is added to `x_8`, the entire expression must become a Taylor map. In our example, the assignment `x_8=x_8+dz_8(1)`, forces the variable `x_8%kind` to be set to 2 and `x_8%alloc` to true. The polymorph is now the Taylor series `x_8%t`.

We now look at the probe and the `probe_8` of the code PTC which contains all the trackable objects.: phase space and spin primarily.

### 1.3 map

## References

- [1] E. Forest, *Beam Dynamics: A New Attitude and Framework* (Harwood Academic Publishers, Amsterdam, The Netherlands, 1997).
- [2] M. Berz, Nucl. Instr. and Meth. **A258**, 431 (1987).
- [3] E. Forest, *From Tracking Code to Analysis, -Generalised Courant-Snyder Theory for any Accelerator Model* (Springer Japan, Tokyo, Japan, 2016).