THE FPP DOCUMENTATION *

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

Abstract

In this short article we summarize the Web documentation surrounding the FPP package.

FULL POLYMORPHIC PACKAGE: FPP

FPP overloads in Fortran90 [1] an old version of the famous "DA-package" [2] of Berz as well as a library of Forest called Lielib[1] which is based on Berz's package.

Real-Taylor Polymorphism

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, though not done in FPP.

In a standard mode, the user of FPP writes an (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. The decision to produce a Taylor map can de done at execution time rather than at compile time.

Manipulation and Analysis of Taylor Maps

The other aspect of FPP which is essential in linac and ring physics is the overloading of the library Lielib. At the SSC-CDG, Berz and Forest started a collaboration whose central purpose was the creation of Lielib. Berz had already a prototype of his "DA" package [8] which he brought to SSC-CDG where it was further developed. A later version of this package is overloaded and is at the root of the type REAL_8 of FPP. However, it was clear to Forest that the sole production of Taylor maps in a circular ring is of limited value. Thus we needed tools to normalize the one-turn map and track it. This was done in the library Lielib which calls routines from the "DA package" to achieve this goal. FPP overloads both packages in Fortran90.

THE FIRST WEB PAGE

The documentation sits at the URL: http://mad.web.cern.ch/mad/PTC_proper/. While it does contain a link to the tracking library PTC [9], everything on that page is FPP proper. The first four FPP links are:

1. What is the Full Polymorphic Package? (Check out this toy overloaded DA-Package, it shows you on a simple example how things work: click here)
2. Initializing FPP : for pure TPSA calculations and for phase space calculations involving type DAMAP. (with examples)
3. Fundamental Types: Taylor, Real Polymorph, Complex Taylor and Complex Polymorph: Including the Knob State of Polymorphs
4. Functions acting on the four fundamental types: Exponential, Logarithm, trigonometric and more...

In item # 1 we repeat more or less the introduction of this paper. We also point to a fully functional "toy" DA package. This has been useful as a pedagogical tool for those who might wonder what the plain "DA" package of Berz does and what it does not.

Item # 2 refers to the two ways FPP (or old Lielib) can be initialized. One can simply ignore accelerator physics and initialize the package as a plain Taylor polymorphic package. Alternatively one can reserve the first 2, 4 or 6 variables of the Taylor series to be phase space variables. In most tracking codes, when maps are produced, this mode is selected.

The next interesting link

Big Table Summarizing many useful Operations

refers to the various Fortran90 operations available to the programmer. These operations perform simple tasks: taking derivatives, extract the coefficient of a monomial or a piece of the Taylor series, etc... As shown on Fig. 1, upon clicking the mouse, the following small program will be displayed:

```
program example
use polymorphic_complextaylor
implicit none
integer no,nv
type(taylor) f,df

! no: the order of the polynomial
! nv: the number of variables
no=4; nv= 2;
! initializes taylor series without maps
call init(no,nv)

! must be constructed after init
call alloc(f,df)
!  Creates 2.d0 x_1 x_ 2 ^2+3.d0 x_1x_2 +4.d0
```

* Work supported by KEK and Kasokukishoureikai.

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

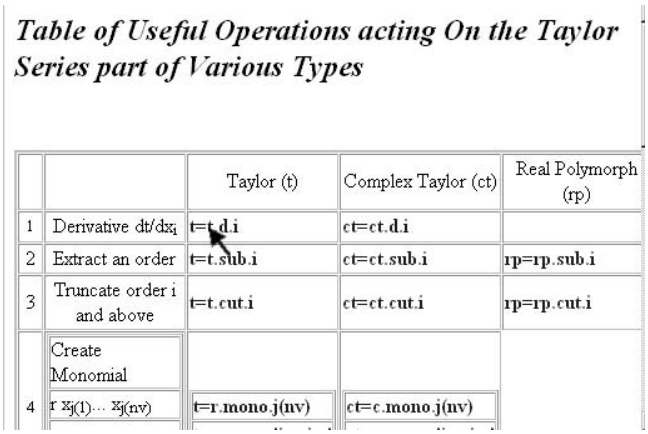

Table of Useful Operations acting On the Taylor Series part of Various Types

		Taylor (t)	Complex Taylor (ct)	Real Polymorph (rp)
1	Derivative dt/dx$_i$	t=t.d.i	ct=ct.d.i	
2	Extract an order	t=t.sub.i	ct=ct.sub.i	rp=rp.sub.i
3	Truncate order i and above	t=t.cut.i	ct=ct.cut.i	rp=rp.cut.i
4	Create Monomial r x$_{j(1)}$... x$_{j(nv)}$	t=r.mono.j(nv)	ct=c.mono.j(nv)	

Figure 1: Part of the "Big Table"

```
f=(2.d0.mono.'12') + (3.d0.mono.'11') + 4.d0
!      df/dx_2    Creates 4.d0 x_1x_2 + 3.d0 x_1
df=f.d.2
call print(f,6)
call print(df,6)

call kill(f,df)  ! must be destroyed
end program example
```

This program should run without problem on all popular platforms. As indicated in the table, it takes the derivative of a Taylor series. These trivial examples are most useful in conjunction with complex examples embedded in a tracking code [9] such as PTC. But nevertheless, they can be used on their own.

All the remaining links are related to maps.

1. Taylor Maps: what is that good for?
2. Plain DAMAP
3. Gmap : an all purpose map

In particular, "plain DAMAP" refers to the most important object of beam dynamics: a map from phase space to phase space. This map, expressed around some orbit, usually the closed orbit in a ring, approximates the map of the tracking code if an integrator is used such as PTC.

The next links are more complex and warrant a section of their own.

OPERATIONS ON DAMAPS

In a single pass system it is sometime sufficient to look at the coefficients of the Taylor map for the system and immediately deduce something useful out of it. For example, people acquainted to the famous code TRANSPORT of Karl Brown [10], often describe the property of their system using the plain coefficients. They will say that M_{166} represents the second order dispersion, i.e., the dependence of the position on the square of the energy variable.

In a ring, things are more complex. This is because we are interested in stability issues: what happen if a map is iterated. In the linear case, rather than looking at M_{11} or M_{12} directly, we perform some "diagonalization" operation, called a normal form. The objects of interest are refereed to collectively as the "lattice functions" and the "tunes." Generally they are averages over time or turn number. The calculational process to extract them from a Taylor map is obviously more involved than a simple reading of the coefficients. These objects and their nonlinear equivalent are readily available in the "DA/Lielib" setting provided by the old package of Berz and the old library of Forest. FPP creates Fortran90 types to overload all the operations in order to use a friendly syntax. For example, if we concatenate two DAMAPs of phase space A and B, and copy the result in C, the syntax is simply:

$$C = B * A \tag{1}$$

This simple expression replaces amongst other things:

$$call\ etcct(B\%v\%i, A\%v\%i, C\%v\%i) \tag{2}$$

Here "A%v(1:6)%i" is an array of six integer pointers to Berz's package. In fact, the Lielib routine $etcct$ calls $DACCT$, the concatenation routine of the DA-package. We should point out that "$*$" is a "differential algebraic" concatenation: it ignores the constant part of the map. It is also possible to make a "truncated power series algebra" concatenation (TPSA) using the syntax:

$$C = B.o.A \tag{3}$$

The myriad of such operations is partly explained if one follows the link:

- Tables Summarizing many useful Operations related to DAMAP

Clicking on the above link gives the reader the following choices:

1. Map Concatenation
2. Norm of a Map
3. Power of a Map
4. Partial Inversion of a Map
5. Vector Fields Action on Taylor Series
6. Various Lie Representation of the Map (besides Normal Form)
7. Exponentiation of the Vector Fields of the above representations using Texp
8. Transforming Vector Fields by Map Directly: Differential Algebraic Operation Only
9. Available DA Concatenation
10. Simple Numerical Operations on DAMAPs

	DA Concatenation	TPSA Concatenation	Click before for explanations
MoM	M=M*M	M=M .o. M	Explanations/Explanations
toM	⊨ * M	⊨ .o. M	<-- See program
r(Inv)=Mor()		r=M*r()	Explanations
r(6)=matrix(6,6) ∘ r(6)		r=matrix*r	Explanations
r(Inv)=tree ∘ r() Tree is a fast trackable map	r=tree*r()	Constant part not present in a matrix!	Explanations
r(Inv)=g ∘ r() Generating function Tracking		Constant part always ignored with generating function	Explanations

Figure 2: Concatenation Table

For example under item #1, Map Concatenation, a table is displayed.

By clicking on the entries of the table one gets an example program and/or an explanation. Several other tables are present on this page:

$M_2=\mathbf{Texp}(F\cdot\nabla)M_1$ and $M_2=\mathbf{Texp}(:f:)M_1$
Introducing Vector Fields and Poisson Bracket Fields using COSY-Infinity style technique to create a FODO cell. 1. Making maps (type damap) with vectors fields (type vecfield), click **here** 2. Making maps (type damap) with Poisson bracket fields (type pbfield), click **here**

Figure 3: Vector Field

Table 3 describes the syntax to generate maps using a single vector field as done in the code COSY-INFINITY [4] of Berz.

One Lie Exponent	$M = \exp(F\cdot\mathrm{grad})\mathrm{Id} \quad F = \sum F_k$	Explanations
Dragt-Finn	$M = \exp(O_{No})\ldots\exp(O_2)\ L\ T\mathrm{Id}$	
Reverse Dragt-Finn	$M = L\exp(O_2)\ldots\exp(O_{No})\ T\mathrm{Id}$ $O_k=F_k\cdot\mathrm{grad}$	

Figure 4: Various Lie Representations

In Table 4, the reader is exposed to three important Lie representations of the map which appear naturally in perturbation calculations, namely in normal forms.

There are three additional tables on that page. The reader should visit the site and click around!

NORMAL FORM

The last big topic introduced on the main page of the web site is normal form. By clicking on the link

Normal Form : A type Central to Accelerator Calculations

one accesses some simple information on normal forms. First there is an FPP program that computes numerically the tune shift with amplitude in a pendulum which can also be solved analytically. The result is compared with the corresponding FPP program.

An less trivial example involving the code PTC is also provided. The user is given access to a flat file of the ALS (the Berkeley Advanced Light Source) and the main program of PTC which reads this file and produces a Twiss table around that machine.

Finally at the end of the main page, we address the issue of the resonance basis or the phasors as they are known in accelerator physics. These are very useful when looking into the resonance content of a map.

CONCLUSION

The main purpose of this web site is to provide simple examples that are supposed to compile and run flawlessly. It is not a web site on the theory and analysis of Taylor maps. It is most useful if one is trying to understand the syntax and logic of an existing FPP/PTC piece of code.

For the theory, one should consult appropriate references and pray that this type of work will start appearing in accelerator schools.

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] M. Berz. The method of power series tracking for the mathematical description of beam dynamics. *Nucl. Instr. and Meth.*, A258:431, 1987.

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

[10] K. L. Brown. A First and Second Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers. Technical Report SLAC Report 75, SLAC, June 1982.