

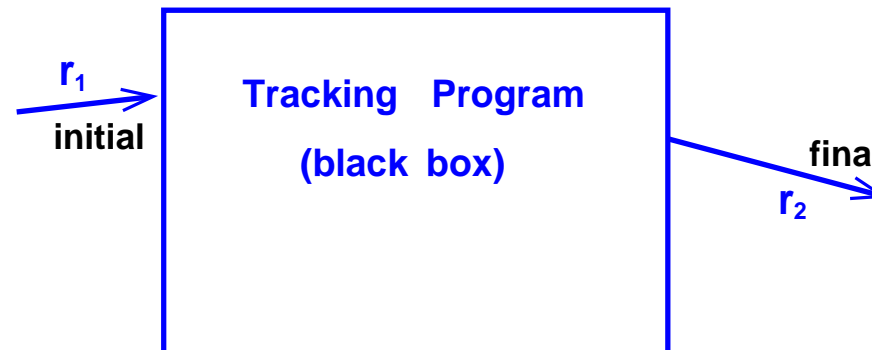
Nonlinear Dynamics - Methods and Tools

—

**and a Contemporary (1985 +) framework
to treat Linear and Nonlinear Beam Dynamics**

PART 3

What about a complicated arrangement, e.g. one that can only be simulated by a computer program ?



A tracking program takes some input, e.g. initial coordinates, and produces (after some time) an output, e.g. final coordinates

- Tracking particles with a computer code is the most reliable (and flexible) method, but we cannot track every particle
- But can we get an "effective Hamiltonian" (and therefore a Normal Form) for a huge and messy computer code ?

Coming down to the point: → TPSA !

Truncated Power Series Algebra [EF1, EF2, AC1, MB, WH1] :

- **Extend operations on real numbers to act on truncated power series**
- **Output of the simulation is a Power Series of the computer code !!**

This allows a Normal Form analysis of the computer code !

Assume our tracking code has established the behaviour of a particle starting at the position a (in phase space). But we cannot track every particle ...

What can we infer for the behaviour of other particles ?

Recall the Taylor series:

$$f(a + \Delta x) = f(a) + \sum_{n=1}^{\infty} \frac{f^{(n)}(a)}{n!} \Delta x^n$$

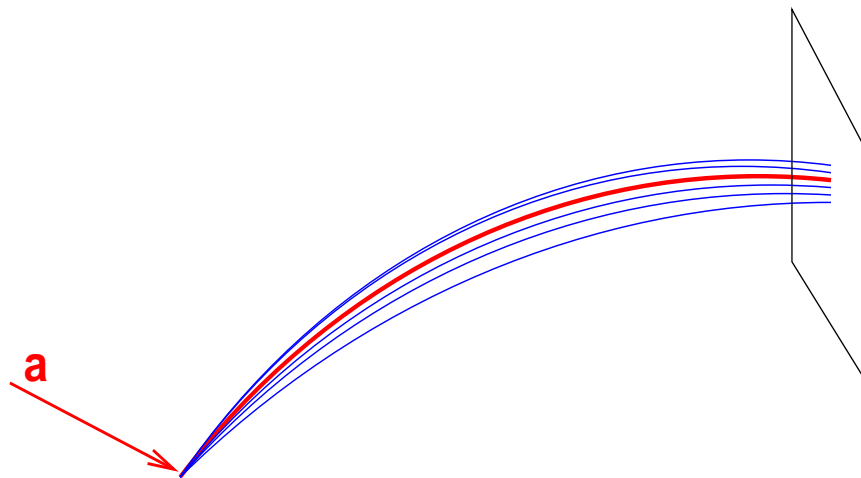
$$f(a + \Delta x) = f(a) + \frac{f'(a)}{1!} \Delta x^1 + \frac{f''(a)}{2!} \Delta x^2 + \frac{f'''(a)}{3!} \Delta x^3 + \dots$$

- The coefficients determine the behaviour of small deviations Δx from the reference a**
- $f(a)$ is the "tracked particle", $f(a + \Delta x)$ is a "close particle"**

If we truncate the expansion to the m-th order:

$$f(a + \Delta x) = f(a) + \sum_{n=1}^m \frac{f^{(n)}(a)}{n!} \Delta x^n$$

→ We can represent the function $f(x)$ near a by the "sequence" of coefficients $(f(a), f'(a), f''(a), \dots, f^{(m)}(a))$



This vector is what we need to understand the behaviour of particles in the neighbourhood of the particle starting at a

→ We have now an analytical solution for our problem

→ How to get these coefficients (derivatives) ?

Numerical differentiation

The problem getting the derivatives $f^{(n)}(a)$ of $f(x)$ at a :

$$f'(a) = \frac{f(a + \Delta x) - f(a)}{\Delta x}$$

- Need to subtract almost equal numbers and divide by small number.
- For higher orders f'' , f''' .., accuracy hopeless !
- The way out: we use Truncated Power Series Algebra (TPSA) (M. Berz, 1988 and [\[MB\]](#))

Truncated Power Series Algebra

- The tracking of a particle in a complicated system relates the output **numerically** to the input:

$$z_1 = (x, p_x, y, p_y, s, \delta)_1 \xRightarrow{\text{tracking}} z_2 = (x, p_x, y, p_y, s, \delta)_2$$

- Can a tracking code directly produce an "analytic" one-turn-map ??

For example the coefficients of a Taylor series ?

$$z_1 = (x, p_x, y, p_y, s, \delta)_1 \xRightarrow{???} z_2 = \sum C_j(a) z_1^j = \sum f^{(n)}(a) z_1^j$$

- The answer is **automatic differentiation**

- Step by step →

1. Define a pair (q_0, q_1) , with q_0, q_1 real numbers

1. Define a pair (q_0, q_1) , with q_0, q_1 real numbers

2. Define operations on a pair like:

$$(q_0, q_1) + (r_0, r_1) = (q_0 + r_0, q_1 + r_1) \text{ obvious !}$$

$$c \cdot (q_0, q_1) = (c \cdot q_0, c \cdot q_1) \text{ obvious !}$$

$$(q_0, q_1) \cdot (r_0, r_1) = (q_0 \cdot r_0, q_0 \cdot r_1 + q_1 \cdot r_0) \text{ not obvious !}$$

1. Define a pair (q_0, q_1) , with q_0, q_1 real numbers

2. Define operations on a pair like:

$$(q_0, q_1) + (r_0, r_1) = (q_0 + r_0, q_1 + r_1) \text{ obvious !}$$

$$c \cdot (q_0, q_1) = (c \cdot q_0, c \cdot q_1) \text{ obvious !}$$

$$(q_0, q_1) \cdot (r_0, r_1) = (q_0 \cdot r_0, q_0 \cdot r_1 + q_1 \cdot r_0) \text{ not obvious !}$$

3. And some ordering:

$$(q_0, q_1) < (r_0, r_1) \text{ if } q_0 < r_0 \text{ or } (q_0 = r_0 \text{ and } q_1 < r_1)$$

$$(q_0, q_1) > (r_0, r_1) \text{ if } q_0 > r_0 \text{ or } (q_0 = r_0 \text{ and } q_1 > r_1)$$

1. Define a pair (q_0, q_1) , with q_0, q_1 real numbers

2. Define operations on a pair like:

$$(q_0, q_1) + (r_0, r_1) = (q_0 + r_0, q_1 + r_1) \text{ obvious !}$$

$$c \cdot (q_0, q_1) = (c \cdot q_0, c \cdot q_1) \text{ obvious !}$$

$$(q_0, q_1) \cdot (r_0, r_1) = (q_0 \cdot r_0, q_0 \cdot r_1 + q_1 \cdot r_0) \text{ not obvious !}$$

3. And some ordering:

$$(q_0, q_1) < (r_0, r_1) \text{ if } q_0 < r_0 \text{ or } (q_0 = r_0 \text{ and } q_1 < r_1)$$

$$(q_0, q_1) > (r_0, r_1) \text{ if } q_0 > r_0 \text{ or } (q_0 = r_0 \text{ and } q_1 > r_1)$$

4. This implies something strange and something very strange:

$$(0, 0) < (0, 1) < (r, 0) \text{ (for any positive } r)$$

$$(0, 1) \cdot (0, 1) = (0, 0) \rightarrow (0, 1) = \sqrt{(0, 0)}!!$$

This means that $(0, 1)$ is between 0 and ANY real number \rightarrow
infinitely small !!!

We call this therefore "differential unit" $\epsilon \stackrel{def}{=} (0, 1)$
(plays the role of Δx in classical calculus ..)

Of course $(q, 0)$ is just the real number q and we define "real part"
and "differential part" (a bit like complex numbers..):

$$q_0 = \mathcal{R}(q_0, q_1) \quad \text{and} \quad q_1 = \mathcal{D}(q_0, q_1)$$

With our rules we can further see that (useful formulae):

$$(1, 0) \cdot (q_0, q_1) = (q_0, q_1)$$

$$(q_0, q_1)^{-1} = \left(\frac{1}{q_0}, -\frac{q_1}{q_0^2} \right)$$

Move from "classical" to "new" calculus

For example, a function f acts on the pair using our rules:

$$x \xrightarrow{\text{becomes}} (x, 0) \quad \text{and} \quad f(x) \xrightarrow{\text{becomes}} f(x, 0)$$

Adding the differential (a la "normal" calculus):

$$f(x) + \epsilon \xrightarrow{\text{becomes}} f(x, 1)$$

The real part corresponds to:

$$\mathcal{R} [f(x, q_1)] = f(x) \quad \text{for any } q_1, \quad \text{i.e. also for } \mathcal{R} [f(x, 1)] = f(x)$$

What about the differential part \mathcal{D} ?

$$\mathcal{D} [f(x, q_1)] \quad \text{in particular what is } \mathcal{D} [f(x, 1)] \quad ???$$

Automatic Differentiation:

Key formula (without proof): for a function $f(x)$:

$$\mathcal{D}[f(x + \epsilon)] = \mathcal{D}[f((x, 0) + (0, 1))] = \mathcal{D}[f(x, 1)] = f'(x)$$

An example instead:

$$f(x) = x^2 + \frac{1}{x}$$

then using school calculus:

$$f'(x) = 2x - \frac{1}{x^2}$$

For $x = 2$ we get then:

$$f(2) = \frac{9}{2}, \quad f'(2) = \frac{15}{4}$$

The miracle:

For x in:

$$f(x) = x^2 + \frac{1}{x}$$

we substitute: $x \rightarrow x + \epsilon \rightarrow (x, 1) = (2, 1)$ **and use our rules:**

$$\begin{aligned} f(2, 1) &= (2, 1)^2 + (2, 1)^{-1} \\ &= (4, 4) + \left(\frac{1}{2}, -\frac{1}{4}\right) \\ &= \left(\frac{9}{2}, \frac{15}{4}\right) = (f(2), f'(2)) \quad !!! \end{aligned}$$

The computation of derivatives becomes an algebraic problem, no need for small numbers, exact !

If anything, remember that:

$$f(x, 1) = (f(x), f'(x))$$

$$(x, 1)^3 = (x^3, 3x^2)$$

$$\sin(x, 1) = (\sin(x), \cos(x))$$

etc. ...

Higher orders:

1. The pair $(q_0, 1)$, becomes a "vector" of order N (just add more 0):

$$(q_0, 1) \rightarrow (q_0, \mathbf{1}, \mathbf{0}, \mathbf{0}, \dots, \mathbf{0}) \quad \epsilon = (\mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \dots, \mathbf{0})$$

2. $(q_0, q_1, q_2, \dots, q_N) + (r_0, r_1, r_2, \dots, r_N) = (q_0 + r_0, q_1 + r_1, \dots, q_N + r_N)$

3. $c \cdot (q_0, q_1, q_2, \dots, q_N) = (c \cdot q_0, c \cdot q_1, c \cdot q_2, \dots, c \cdot q_N)$

4. $(q_0, q_1, q_2, \dots, q_N) \cdot (r_0, r_1, r_2, \dots, r_N) = (s_0, s_1, s_2, \dots, s_N)$

$$\text{with : } s_i = \sum_{k=0}^i \frac{i!}{k! (i-k)!} q_k r_{i-k}$$

for example: $s_0 = q_0 r_0, \quad s_1 = q_1 r_0 + q_0 r_1, \quad s_2 = q_2 r_0 + 2 q_1 r_1 + q_0 r_2, \dots$

5. Note: $(\mathbf{1}, \mathbf{0}, \mathbf{0}, \dots) \cdot (q_0, q_1, q_2, \dots, q_N) = (q_0, q_1, q_2, \dots, q_N)$

Examples:

$$(x, 0, 0, 0, \dots)^n = (x^n, 0, 0, 0, \dots)$$

$$(0, 1, 0, 0, \dots)^n = (0, 0, 0, \dots, \overbrace{n!}^{n+1}, 0, 0, \dots)$$

$$(x, 1, 0, 0, \dots)^2 = (x^2, 2x, 2, 0, \dots)$$

$$(x, 1, 0, 0, \dots)^3 = (x^3, 3x^2, 6x, 6, \dots)$$

And an exercise: $(x, 1, 0, 0, \dots)^{-3} = (?, ?, ?, \dots)$

$$f(x) \rightarrow f(x, 1, 0, 0, \dots) = \underbrace{(x, 1, 0, 0, \dots)^{-3}}_{\text{next : multiply both sides with } (x, 1, 0, 0)^3} = (f_0, f', f'', f''', \dots)$$

$$(1, 0, 0, \dots) = (x, 1, 0, 0, \dots)^3 \cdot (f_0, f', f'', f''', \dots)$$

$$(1, 0, 0, \dots) = (x^3, 3x^2, 6x, 6, \dots) \cdot (f_0, f', f'', f''', \dots)$$

$$(1, 0, 0, \dots) = (\underbrace{x^3 \cdot f_0}_{q_0}, \underbrace{3x^2 \cdot f_0 + x^3 \cdot f'}_{q_1}, \underbrace{6x \cdot f_0 + 2 \cdot 3x^2 \cdot f' + x^3 \cdot f''}_{q_2}, \dots)$$

Easily solved by forward substitution:

$$1 = x^3 \cdot f_0 \quad \rightarrow \quad f_0 = x^{-3}$$

$$0 = 3x^2 \cdot f_0 + x^3 \cdot f' \quad \rightarrow \quad f' = -3x^{-4}$$

$$0 = 6x \cdot f_0 + 2 \cdot 3x^2 \cdot f' + x^3 \cdot f'' \quad \rightarrow \quad f'' = 12x^{-5}$$

....

If you are bored or already fascinated:

Try: $f(x) = e^x$

Hint:

$$e^{(x,1,0,0,0)} = e^{(x,0,0,0,0) + (0,1,0,0,0)} = e^{(x,0,0,0,0)} \cdot e^{(0,1,0,0,0)}$$

Maybe try also: $f(x) = \sin(x), \cos(x), \dots$

Note:

the vector $(0, 1, 0, 0)$ is small, but $(0, 2, 0, 0)$ is small, $(0, 2, 5, 0)$ is small, $(0, 0, 2, 5, 0)$ is small ...

The key is: $(0, x, y, z, \dots)$!

Can be extended to more variables x, p_x :

$$x = (a, 1, 0, 0, 0, \dots) \quad \epsilon_x = (0, 1, 0, 0, 0, \dots)$$

$$p_x = (b, 0, 1, 0, 0, \dots) \quad \epsilon_{p_x} = (0, 0, 1, 0, 0, \dots)$$

and get (with modified multiplication rules):

$$(q_{00}, q_{10}, q_{01}, q_{20}, \dots) \cdot (r_{00}, r_{10}, r_{01}, r_{20}, \dots) = (s_{00}, s_{10}, s_{01}, s_{20}, \dots)$$

with:

$$s_{mn} = \sum_{k=0}^m \sum_{l=0}^n q_{kl} \cdot r_{m-k, n-l} \frac{m! n!}{k! (m-k)! l! (n-l)!}$$

$$\rightarrow f(x, p_x) = \left(f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial p_x}, \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial x \partial p_x}, \dots \right) (a, b)$$

Note: no panic, all this is (easily) done on a computer

What is the use of that:



Assume we have written a complex simulation code (maybe more than 100000 lines ..)

1. We replace all standard operations with our new definitions
2. We push the vector $f(x) = (a, 1, 0, 0, 0\dots)$ through the algorithm as if it is a vector in phase space
3. We extract a truncated Taylor map with the desired accuracy and to any order

How to proceed:

- Replacing the operations with a new definition is very easy using polymorphism of modern languages (e.g. C++, FORTRAN 95,..) and operator overloading, see following example
- We have a Taylor map, i.e. analytical representation of a computer code (somewhat metaphorically) !!!
- The resulting map can be used for tracking in a computer program
- Normal form analysis on Taylor series is much easier: the coefficients are directly related to the various aberrations !!

What is the use of that:

- Demonstrate with simple examples (FORTRAN 95):
 - First show the concept
 - Simple FODO cell
 - Normal form analysis of the FODO cell with octupoles
- All examples and all source code in:

Website: <http://cern.ch/Werner.Herr/CAS2017/DA>

Small DA package provided by E. Forest

Look at this small example: $f(a) = \sin(\frac{\pi}{6})$

PROGRAM DATEST1

use my_own_da

real x,z, dx



my_order=3

dx=0.0

x=3.141592653/6.0 + dx

call track(x, z)

call print(z,6)

END PROGRAM DATEST1

SUBROUTINE TRACK(a, b)

use my_own_da

real a,b



b = sin(a)

END SUBROUTINE TRACK

PROGRAM DATEST2

use my_own_da

type(my_taylor) x,z, dx

my_order=3

dx=1.0.mono.1 ! this is our (0,1)

x=3.141592653/6.0 + dx

call track(x, z)

call print(z,6)

END PROGRAM DATEST2

SUBROUTINE TRACK(a, b)

use my_own_da

type(my_taylor) a,b

b = sin(a)

END SUBROUTINE TRACK

Key: operations on "real" are overloaded, changing the declaration

Look at the results:

(0,0) 0.50000000000000E+00

(0,0) 0.50000000000000E+00

(1,0) 0.8660254037844E+00

(0,1) 0.00000000000000E+00

(2,0) -0.25000000000000E+00

(0,2) 0.00000000000000E+00

(1,1) 0.00000000000000E+00

(3,0) -0.1443375672974E+00

(0,3) 0.00000000000000E+00

(2,1) 0.00000000000000E+00

(1,2) 0.00000000000000E+00

We have $\sin(\frac{\pi}{6}) = 0.5$ all right, but what is the rest ??

Look at the results:

(0,0) 0.50000000000000E+00

(0,0) 0.50000000000000E+00

(1,0) 0.8660254037844E+00

(0,1) 0.00000000000000E+00

(2,0) -0.25000000000000E+00

(0,2) 0.00000000000000E+00

(1,1) 0.00000000000000E+00

(3,0) -0.1443375672974E+00

(0,3) 0.00000000000000E+00

(2,1) 0.00000000000000E+00

(1,2) 0.00000000000000E+00

$$\sin\left(\frac{\pi}{6} + \Delta x\right) = \sin\left(\frac{\pi}{6}\right) + \cos\left(\frac{\pi}{6}\right)\Delta x^1 - \frac{1}{2}\sin\left(\frac{\pi}{6}\right)\Delta x^2 - \frac{1}{6}\cos\left(\frac{\pi}{6}\right)\Delta x^3$$


What is the use of that:

- We have used a simple algorithm here (*sin*) but it can be **anything** very complex (try: $\frac{\sin(x)}{x}$ using the DA package provided on the webpage)
- We can compute nonlinear maps as a Taylor expansion of **anything** the program computes
- Simply by:
 - Replacing regular (e.g. REAL) types by TPSA types (*my_taylor*)
 - Variables x, p are automatically replaced by $(x, 1, 0, ..)$ and $(p, 0, 1, 0, ..)$ etc.
 - Operators and functions $(+, -, *, =, ..., exp, sin, ...)$ automatically overloaded, i.e. behave according to new type

What is the use of that:


Assume the *Algorithm* describes one turn, then:

Normal tracking:

 $X_n = (x, p_x, y, p_y, s, \delta)_n \rightarrow X_{n+1} = (x, p_x, y, p_y, s, \delta)_{n+1}$

 **Coordinates after one completed turn**

TPSA tracking:

 $X_n = (x, p_x, y, p_y, s, \delta)_n \rightarrow X_{n+1} = \sum C_j X_n^j$

 **Taylor coefficients after one completed turn**

 **The C_j contain useful information about behaviour**

 **Taylor map directly used for normal form analysis** 

Demo 1:

➤ Track through 8 FODO cells:

QF - DRIFT - QD - DRIFT

Now we use **three** variables:

$x, p, \Delta p = (z(1), z(2), z(3))$

Thin lenses:

Quadrupole treated as $\rightarrow \frac{L_q}{2} - K1 - \frac{L_q}{2}$

```
program ex1
```

```
use my_own_da
```

```
use my_analysis
```

```
type(my_taylor) z(3)
```

```
type(normalform) NORMAL
```

```
type(my_map) M,id
```

```
real(dp) L,DL,k1,k3,fix(3)
```

```
! set up initial parameters
```

```
my_order=4 ! maximum order 4
```

```
fix=0.0 ! fixed point
```

```
id=1
```

```
z=fix+id
```

```
! set up lattice parameters
```

```
LC=62.5 ! half cell length
```

```
DL=3.0 ! quadrupole length
```

```
kf= 0.00295278 ! strength
```

```
kd=-0.00295278 ! strength
```

```
do j = 1,8 ! track through 8 FODO cells
```

```
z(1) = z(1)+DL/2*z(2)
```

```
z(2) = z(2)-kf*DL*z(1)/(1 + z(3))
```

```
z(1) = z(1)+DL/2*z(2)
```

```
! track through F quadrupole
```

```
z(1)=z(1)+LC*z(2) ! drift of half cell length
```

```
z(1) = z(1)+DL/2*z(2)
```

```
z(2) = z(2)-kd*DL*z(1)/(1 + z(3))
```

```
z(1) = z(1)+DL/2*z(2)
```

```
! track through D quadrupole
```

```
z(1)=z(1)+LC*z(2) ! drift of half cell length
```

```
enddo
```

```
call print(z(1),6)
```

```
call print(z(2),6)
```

```
M=z ! overloads coefficient with the map
```

```
normal=m ! overloads map with normal form
```

```
write(6,*) normal%tune, normal%dtune_da
```

```
end program ex1
```

The result is (single cell):

(0,0,0) 0.9369211296691E-01
(0,0,1) -0.9649503806747E-01

(1,0,0) 0.9083165810508E-01
(0,1,0) 0.1667704101367E+03

(1,0,1) 0.1238115392391E+01
(0,1,1) -0.3527698956093E+02
(1,0,2) -0.1567062442887E+01
(0,1,2) 0.3478356898518E+02
(1,0,3) 0.1896009493384E+01
(0,1,3) -0.3429014840944E+02

(1,0,0) -0.5139797664004E-02
(0,1,0) 0.1572511594903E+01
(1,0,1) 0.1027959532801E-01
(0,1,1) -0.5648018984066E+00
(1,0,2) -0.1541939299201E-01
(0,1,2) 0.5570922019106E+00
(1,0,3) 0.2055919065602E-01
(0,1,3) -0.5493825054146E+01

From the elements in the Taylor expansion, the result for the matrix per cell:

$$\Delta x_f = 0.09083\Delta x_i + 166.77\Delta p_i$$

$$\Delta p_f = -0.00514\Delta x_i + 1.5725\Delta p_i$$

The output from the normal form analysis are (per cell !):

$$\text{Tune} = (0,0,0) = 0.093692$$

$$\text{Chromaticity} = (0,0,1) = -0.096495$$

The "defined assignment": $M = z$, makes a map M out of the coefficients z

The "defined assignment": $NORMAL = M$, makes a normal form $NORMAL$ out of the map M

In FORTRAN95 derived "type" plays the role of "structures" in C, and $NORMAL$ contains:

$NORMAL\%tune$ is the tune Q

$NORMAL\%dtune_da$ is the detuning with amplitude $\frac{dQ}{da}$

$NORMAL\%R$, $NORMAL\%A$, $NORMAL\%A^{**-1}$ are the matrices

R , A , A^{-1} from the normal form transformation and we get $\alpha, \beta, \gamma \dots$

`real z(3) → type(my_taylor) z(3)`

`M = z`

`NORMAL = M`

Polymorphism at its best !

What about dipoles ??

Of course they are described by a matrix^a, but:

Only useful in full 3D $(x, p_x, y, p_y, z, \delta)$

They do not change the reference orbit

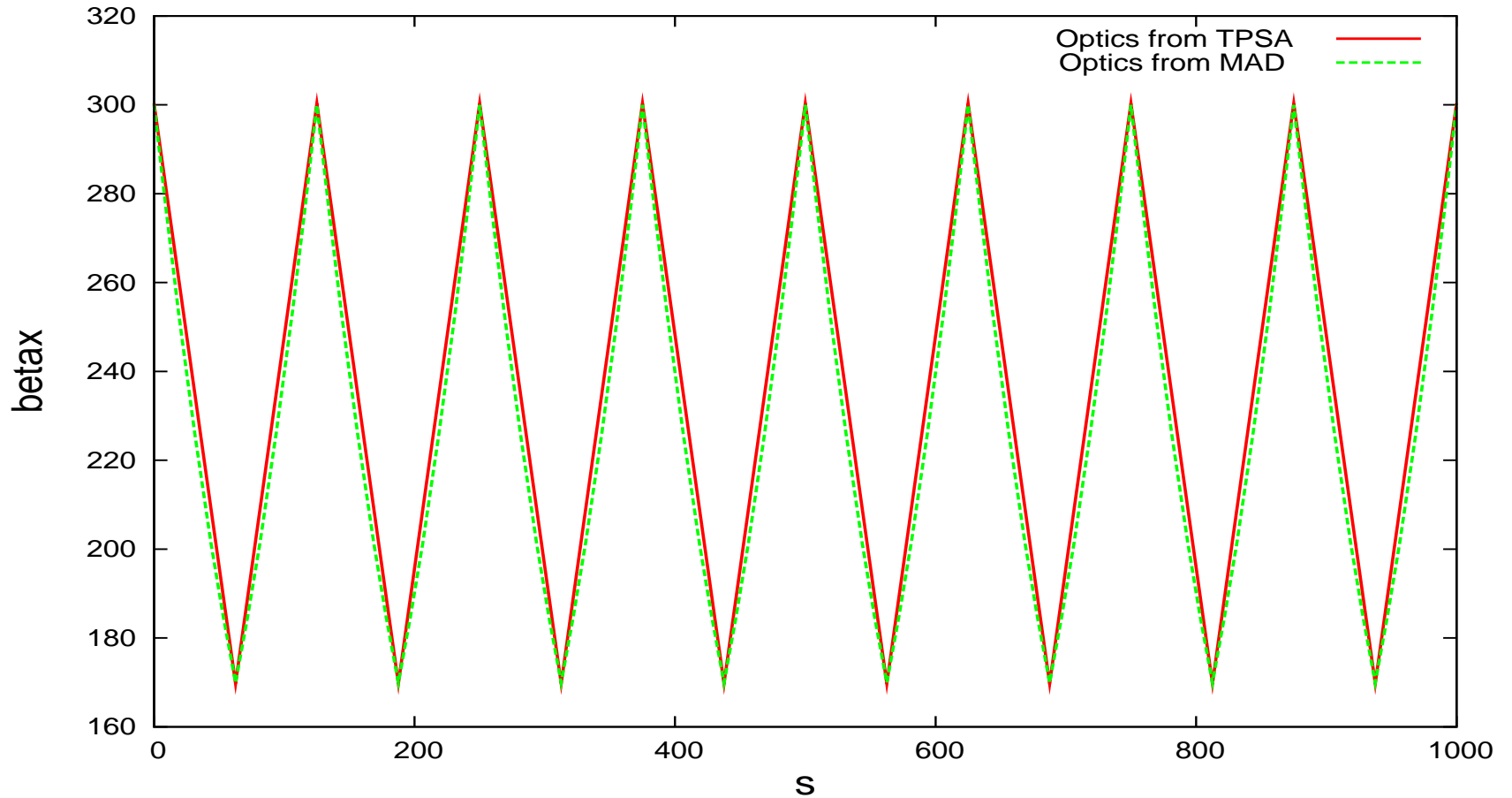
Focusing terms in the matrix

The path length difference z relative to the design orbit has changed: a.k.a. dispersion

To think about it tonight: why did I not use sextupoles to correct the chromaticity in my example ?

^asee e.g. C. Iselin, Part. Acc. Vol.17 (1985) pp. 143

beta function from tracking



$$\beta_{max} \approx 300 \text{ m}, \quad \beta_{min} \approx 170 \text{ m}$$



exercise 1 in the optics course ! (ex1.f90 in the package)

```
program ex1_oct
```

```
use my_own_da
```

```
use my_analysis
```

```
type(my_taylor) z(3)
```

```
type(normalform) NORMAL
```

```
type(my_map) M,id
```

```
real(dp) L,DL,k1,k3,fix(3)
```

```
! set up initial parameters
```

```
my_order=4 ! maximum order 4
```

```
fix=0.0 ! fixed point
```

```
id=1
```

```
z=fix+id
```

```
! set up lattice parameters
```

```
LC=62.5 ! half cell length
```

```
DL=3.0 ! quadrupole length
```

```
kf= 0.00295278 ! strength
```

```
kd=-0.00295278 ! strength
```

```
do j = 1,8 ! track through 8 FODO cells
```

```
z(1) = z(1)+DL/2*z(2)
```

```
z(2) = z(2)-kf*DL*z(1)/(1 + z(3))
```

```
z(1) = z(1)+DL/2*z(2)
```

} ! track through F quadrupole

```
z(2)=z(2)-k3*z(1)**3/(1 + z(3)) ! add octupole kick
```

```
z(1)=z(1)+LC*z(2) ! drift of half cell length
```

```
z(1) = z(1)+DL/2*z(2)
```

```
z(2) = z(2)-kd*DL*z(1)/(1 + z(3))
```

```
z(1) = z(1)+DL/2*z(2)
```

} ! track through D quadrupole

```
z(1)=z(1)+LC*z(2) ! drift of half cell length
```

```
enddo
```

```
call print(z(1),6)
```

```
call print(z(2),6)
```

```
M=z ! overloads coefficient with the map
```

```
normal=m ! overloads map with normal form
```

```
write(6,*) normal%tune, normal%dtune_da
```

```
end program ex1_oct
```

The result is:

(0,0,0) 0.9369211296691E-01
(0,0,1) -0.9649503806747E-01

(2,0,0) 0.5383744464902E+02
(0,2,0) 0.5383744464902E+02
(0,0,2) 0.1009289258270E+00
(2,0,1) 0.2116575633218E+02

.....

(1,0,0) 0.9083165810508E-01
(0,1,0) 0.1667704101367E+03
(1,0,1) 0.1238115392391E+01
(0,1,1)-0.3527698956093E+02
(3,0,0)-0.1578216232118E+01
(2,1,0)-0.1429958442579E+02
(1,2,0)-0.4318760015031E+02

.....

(1,0,0)-0.5139797664004E-02
(0,1,0) 0.1572511594903E+01
(1,0,1) 0.1027959532801E-01
(0,1,1)-0.5648018984066E+00
(3,0,0)-0.1505298087837E-01

Linear matrix as before, but effects of the octupole.

The output from the normal form analysis are (per cell !):

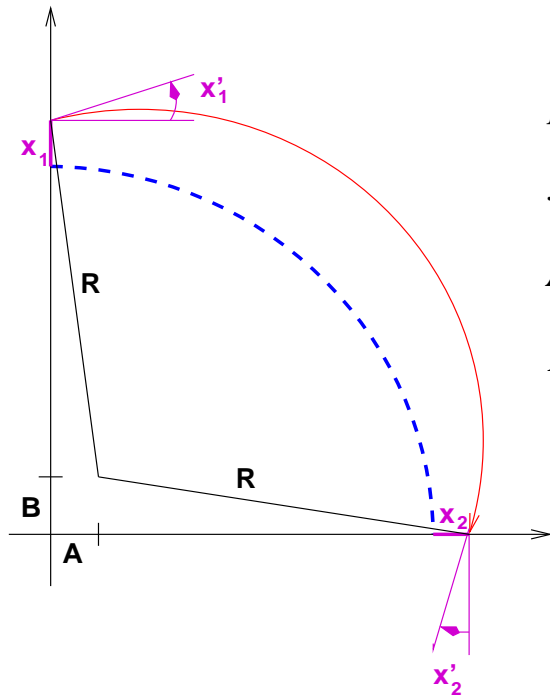
Tune = (0,0,0) = 0.093692

Chromaticity = (0,0,1) = -0.096495

Detuning with amplitude = (2,0,0) = 53.74 !

This was trivial - now a (normally) hard one

The exact map:



$$p_2 = \sin(x'_2) = -\frac{B}{R}$$

$$x_2 = A - R(1 - \cos(x'_2)) = A - R(1 - \sqrt{1 - p_2^2})$$

$$A = R \cdot p_1 = R \cdot \sin(x'_1)$$

$$B = R(1 - \cos(x'_1)) + x_1 = R(1 - \sqrt{1 - p_1^2}) + x_1$$

A 90° bending magnet ..

How to apply Differential Algebra here ...

➤ **Start with initial coordinates in DA style:**

$$x_1 = (0, 1, 0, \dots)$$

$$p_1 = (0, 0, 1, \dots) \quad \text{and have:}$$

$$A = (0, 0, R, 0, \dots)$$

$$B = (0, 1, 0, 0, 0, R, 0, \dots)$$

➤ **After pushing them through the algorithm:**

$$\rightarrow x_2 = (0, 0, R, -\frac{1}{R}, 0, 0, 0\dots) = (0, \frac{\partial x_2}{\partial x_1}, \frac{\partial x_2}{\partial p_1}, \frac{\partial^2 x_2}{\partial x_1^2}, \frac{\partial^2 x_2}{\partial x_1 \partial p_1}, \dots)$$

$$\rightarrow p_2 = (0, -\frac{1}{R}, 0, 0, 0, -1, 0\dots) = (0, \frac{\partial p_2}{\partial x_1}, \frac{\partial p_2}{\partial p_1}, \frac{\partial^2 p_2}{\partial x_1^2}, \frac{\partial^2 p_2}{\partial x_1 \partial p_1}, \dots)$$

➤ **Automatically evaluates all nonlinearities to any desired order**

..

How to apply Differential Algebra here ...

➤ **Start with initial coordinates in DA style:**

$$x_1 = (0, 1, 0, \dots)$$

$$p_1 = (0, 0, 1, \dots) \quad \text{and have:}$$

$$A = (0, 0, R, 0, \dots)$$

$$B = (0, 1, 0, 0, 0, R, 0, \dots)$$

➤ **After pushing them through the algorithm:**

$$\rightarrow x_2 = (0, \mathbf{0}, \mathbf{R}, -\frac{1}{R}, 0, 0, 0, \dots) = (0, \frac{\partial x_2}{\partial x_1}, \frac{\partial x_2}{\partial p_1}, \frac{\partial^2 x_2}{\partial x_1^2}, \frac{\partial^2 x_2}{\partial x_1 \partial p_1}, \dots)$$

$$\rightarrow p_2 = (0, -\frac{1}{R}, \mathbf{0}, 0, 0, -1, 0, \dots) = (0, \frac{\partial p_2}{\partial x_1}, \frac{\partial p_2}{\partial p_1}, \frac{\partial^2 p_2}{\partial x_1^2}, \frac{\partial^2 p_2}{\partial x_1 \partial p_1}, \dots)$$

➤ **Automatically evaluates all nonlinearities to any desired order**

..

Some we know ...

Linear transfer matrix of a dipole:

$$M_{dipole} = \begin{pmatrix} \cos\left(\frac{L}{R}\right) & R \sin\left(\frac{L}{R}\right) \\ -\frac{1}{R} \sin\left(\frac{L}{R}\right) & \cos\left(\frac{L}{R}\right) \end{pmatrix} = \begin{pmatrix} \frac{\partial x_2}{\partial x_1} & \frac{\partial x_2}{\partial p_1} \\ \frac{\partial p_2}{\partial x_1} & \frac{\partial p_2}{\partial p_1} \end{pmatrix}$$

For a 90° bending angle we get the linear matrix:

$$M_{dipole} = \begin{pmatrix} 0 & R \\ -\frac{1}{R} & 0 \end{pmatrix}$$

as computed, but we also have **all** derivatives and nonlinear effects !

Bottom line ...

- **TPSA provides analytic expression (Truncated Taylor polynomial)^{*)} for the one turn map produced by tracking with a computer code (to arbitrary high order !)**
- **Does not require anything beyond what is in the tracking code (in particular can be used with measured data)**
- **Typical use: Normal Form Analysis discussed earlier, rather straightforward from a Taylor polynomial**
- **TPSA easily represents the dependence of a map on machine parameters. This dependence can be used to study, determine, correct (!) the effects of parameter variation (i.e. correction of aberrations)**

^{*)}

For the interested : "jets"

Twiss and More ...

Optics and tracking codes in the future will be based on these techniques

They allow more exact and flexible calculations and analyses

Conventional matrix based techniques are moribund will become/became a thing of the past

**For example, existing modern codes are:
COSY infinity, PTC, BMAD, MADX-PTC**

Short summary

- **Nonlinearities have a strong impact on the beam dynamics and the methods to study them**
- **Many tools already available, usually in form of libraries or tool boxes. Can be used easily without knowing the details**
- **Numerical methods (e.g. simulations) and analytical methods (e.g. mapping and normal forms) form a unified tool kit**

Short summary

- **Nonlinearities have a strong impact on the beam dynamics and the methods to study them**
- **Many tools already available, usually in form of libraries or tool boxes. Can be used easily without knowing the details**
- **Numerical methods (e.g. simulations) and analytical methods (e.g. mapping and normal forms) form a unified tool kit**

Still left to do:

- **Find its way into textbooks ..**

Bibliography/References:

[EF1] E. Forest, Beam Dynamics - A New Attitude and Framework, Harwood Academic Publishers, 1998.

[EF2] E. Forest, From Tracking Code to Analysis, Springer, 2016.

[MB] M. Berz, Modern Map Methods in Particle Beam Physics, Academic Press, 1999.

[AD] A. Dragt, Lie Methods for Nonlinear Dynamics with Applications to Accelerator Physics, University of Maryland

[AW] A. Wolski, Beam Dynamics in High Energy Particle Accelerators, Imperial College Press, 2014.

[AC1] A. Chao, Lecture Notes on Topics in Accelerator Physics, SLAC, 2001.

[AW1] A. Wolski, Nonlinear Single Particle Dynamics, Univ. Liverpool.

[WH1] W. Herr, Mathematical and Numerical Methods for Nonlinear Dynamics Proc. CERN Accelerator School: Advanced Accelerator Physics (2013).